

# **Introduction to Machine Learning**

**From Math to Code**

Ruye Wang



# Contents

<i>List of illustrations</i>	<i>page</i> viii
<i>List of tables</i>	ix
<i>List of boxes</i>	x
<i>List of contributors</i>	xi
<i>Notation</i>	xii
<b>Part I Mathematical Foundations</b>	<b>1</b>
<b>1 Solving Equations</b>	<b>3</b>
1.1 Linear Equation Systems	4
1.2 The Bisection and Secant methods	6
1.3 Fixed point iteration	10
1.4 Newton-Raphson Method (Univariate)	20
1.5 Newton-Raphson Method (Multivariate)	26
<b>2 Optimization</b>	<b>33</b>
2.1 Unconstrained Optimization	34
2.1.1 Optimization by Solving Equations	34
2.1.2 Newton's method	35
2.1.3 Gradient Descent Method	39
2.1.4 Line minimization	42
2.1.5 Quasi-Newton Methods	47
2.1.6 Conjugate gradient method	54
2.1.7 Issues of Local/Global Minimum	63
2.2 Constrained Optimization	64
2.2.1 Optimization with Equality Constraints	65
2.2.2 Optimization with Inequality Constraints	67
2.2.3 Duality and KKT Conditions	71
2.2.4 Linear Programming (LP)	74
2.2.5 The Simplex Algorithm	80
2.2.6 Quadratic Programming (QP)	86
2.2.7 Interior Point Methods	87
<b>3 Review of Probability and Statistics</b>	<b>95</b>

3.1	Univariate and Multivariate Random Variables	95
3.1.1	Univariate Variables	95
3.1.2	Multivariate random vectors	101
3.1.3	The Normal Distribution	105
3.1.4	Function of Random Variables	112
3.2	Entropy and Information	114
3.2.1	Entropy	114
3.2.2	Mutual Information and Kullback-Leibler Divergence	121
3.2.3	Entropy Maximization	123
3.3	Bayesian Inference	125
3.3.1	Bayes' Theorem	125
3.3.2	Maximum Likelihood Estimation of Model Parameters	128
3.3.3	Natural gradient	130
3.3.4	Fisher Information and Natural Gradient	131
3.4	Bias-Variance Tradeoff	133
3.5	Principal Component Analysis (PCA)	134
3.6	Student's t-test	137
3.7	Hypergeometric Distribution	139
<b>Part II</b>	<b>Regression</b>	<b>141</b>
<b>4</b>	<b>Linear Regression</b>	<b>143</b>
4.1	Regression as Optimization	143
4.2	Linear Least Squares Regression	147
4.3	Bias-Variance Tradeoff and Ridge Regression	154
4.4	Regression Based on Basis Functions	158
4.5	Bayesian Linear Regression	162
<b>5</b>	<b>Nonlinear Regression</b>	<b>169</b>
5.1	Nonlinear Least Squares Regression	169
5.2	Parameter Estimation by Natural Gradient Descent	175
<b>6</b>	<b>Logistic and Softmax Regression</b>	<b>177</b>
6.1	Logistic Regression and Binary Classification	177
6.2	Softmax Regression for Multiclass Classification	183
<b>7</b>	<b>Gaussian Process Regression and Classification</b>	<b>191</b>
7.1	Gaussian Process Regression	191
7.2	Gaussian Process Classifier – Binary	196
7.3	Gaussian Process Classifier – Multi-Class	203
<b>Part III</b>	<b>Feature Selection</b>	<b>211</b>
<b>8</b>	<b>Feature Selection</b>	<b>213</b>

---

8.1	Dimensionality Reduction	213
8.2	Distances and Separability Measurements	213
<b>9</b>	<b>Principal Component Analysis</b>	<b>221</b>
9.1	Covariance and Correlation	221
9.2	Karhunen-Loeve Transformation	223
9.3	Optimality of KLT	225
9.4	Geometric Interpretation of the KLT	228
9.5	Computation of the KLT	229
9.6	Comparison with Other Orthogonal Transforms	231
9.7	Application to Image Data	232
<b>10</b>	<b>Variations of PCA</b>	<b>237</b>
10.1	Kernel Methods	237
10.2	Kernel PCA	240
10.3	Factor Analysis and Expectation Maximization	243
10.4	Probabilistic PCA	249
10.5	Classical Multidimensional Scaling	253
10.6	t-Distributed Stochastic Neighbor Embedding (t-SNE)	257
<b>11</b>	<b>Independent Component Analysis</b>	<b>262</b>
11.1	Independence and Non-Gaussianity	262
11.2	Preprocessing	264
11.3	Non-Gaussianity Maximization	265
11.4	The FastICA Algorithm	266
11.5	Likelihood Maximization	269
<b>Part IV</b>	<b>Classification</b>	<b>273</b>
<b>12</b>	<b>Statistic Classification</b>	<b>275</b>
12.1	Discriminative vs. Generative Methods for Classification	275
12.2	K Nearest Neighbor and Minimum Distance Classifiers	277
12.3	Naive Bayes Classification	279
12.4	AdaBoost	286
<b>13</b>	<b>Support Vector machine</b>	<b>295</b>
13.1	Maximal Margin and Support Vectors	295
13.2	Kernel Mapping	301
13.3	Soft Margin SVM	305
13.4	Sequential Minimal Optimization Algorithm	308
13.5	Multi-Class Classification	316
13.6	Kernelized Bayes classifier	319
13.7	Hierarchical (Tree) Classifiers	325

---

<b>14</b>	<b>Clustering Analysis</b>	329
	14.1 K-means clustering	329
	14.2 Gaussian mixture model	333
	14.3 Mixture of Bernoulli	341
	14.4 General EM Algorithm	344
	14.5 Linear Models for Binary Classification	345
<b>Part V</b>	<b>Neural Networks</b>	349
<b>15</b>	<b>Biologically Inspired Networks</b>	351
	15.1 Biological Inspiration	351
	15.2 Hebbian Learning	354
	15.3 Hopfield Network	356
<b>16</b>	<b>Perceptron-Based Networks</b>	360
	16.1 Perceptron	360
	16.2 Back Propagation	367
	16.3 Autoencoder	375
	16.4 Deep Learning	381
	16.5 Generative Adversarial Networks	384
<b>17</b>	<b>Competition-Based Networks</b>	386
	17.1 Competitive Learning Network	386
	17.2 Self-Organizing Map (SOM)	390
<b>Part VI</b>	<b>Reinforcement Learning</b>	395
<b>18</b>	<b>Introduction to Reinforcement Learning</b>	397
	18.1 Reinforcement Learning	397
	18.2 Markov Decision Process	398
	18.3 Model-Based Planning	404
	18.4 Model-Free Evaluation and Control	408
	18.4.1 Monte Carlo (MC) Algorithms	412
	18.4.2 Temporal Difference (TD) Algorithms	415
	18.4.3 TD( $\lambda$ ) Algorithm	418
	18.5 Value Function Approximation	423
	18.6 Control based on Function Approximation	427
	18.7 Deep Q-learning	429
	18.8 Policy Gradient Methods	429
<b>Appendix A</b>	<b>A Review of Linear Algebra</b>	437
	A.1 Inner Product Space	437
	A.2 Orthogonal Basis and Gram-Schmidt Process	444
	A.3 Rank, Trace, Determinant, Transpose, and Inverse of Matrices	446

---

A.4	Normal, Unitary, and Similar Matrices	448
A.5	Positive/Negative (Semi)-Definite Matrices	449
A.6	Woodbury Matrix Identity and Sherman-Morrison Formula	450
A.7	Inverse and Determinant of Partitioned Symmetric Matrices	451
A.8	Unitary Transform	453
A.9	Eigenvalue Decomposition	454
A.10	Generalized Eigenvalue Problem	459
A.11	Rayleigh Quotient	461
A.12	Normal Matrices and Diagonalizability	461
A.13	Centering Matrix	463
A.14	Singular Value Decomposition	464
A.15	Pseudo-Inverse	468
A.16	Vector Norms	469
A.17	Matrix Norms	473
A.18	Vector and Matrix Differentiation	481
A.19	The Fundamental Theorem of Linear Algebra	482
A.20	Pseudo-Inverse Solutions Based on SVD	494
A.21	Rank-Nullity Theorem	501
A.22	Taylor Series Expansion	503
A.23	Normal Direction of a Plane	505
A.24	Convexity	506
A.25	Jensen Inequality	508
A.26	Sensitivity and Conditioning	509
A.27	Some Useful Inequalities	517
	<i>Notes</i>	520
	<i>Author index</i>	521
	<i>Subject index</i>	522

# Illustrations

1.1	Bisection Search	6
1.2		7



## Tables

## Boxes

## **List of contributors**

## Notation

AFM	atomic force microscope
AKPZ	anisotropic KPZ equation
$a_0$	lattice constant
$c_q(\ell)$	q-th order correlation function
$d_E$	embedding dimension
$d_f$	fractal dimension
$L$	system size
$\equiv$	<i>defined</i> to be equal
$\sim$	<i>asymptotically</i> equal (in scaling sense)
$\approx$	<i>approximately</i> equal (in numerical value)

# Part I

---

## Mathematical Foundations



# 1 Solving Equations

---

In this chapter we consider the methods for solving a set of  $M$  simultaneous equations of  $N$  variables:

$$\begin{cases} f_1(x_1, \dots, x_N) = f_1(\mathbf{x}) = 0 \\ \dots\dots\dots \\ f_M(x_1, \dots, x_N) = f_M(\mathbf{x}) = 0 \end{cases} \quad (1.1)$$

where  $\mathbf{x} = [x_1, \dots, x_N]^T \in \mathbb{R}^N$  is an N-D column vector, a point in the N-D vector space  $\mathbb{R}^N$  spanned by  $\{x_1, \dots, x_N\}$ . We further define  $\mathbf{f} = [f_1, \dots, f_M]^T$  as an M-D column vector, and represent this equation system more concisely as  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ .

Geometrically, a function  $f(\mathbf{x})$  defined over this space  $\mathbb{R}^N$  is a hypersurface in the N+1 dimensional space, of which the N+1st dimension represents the function value  $y = f(\mathbf{x})$  corresponding to a point  $\mathbf{x}$  in the N-D space. The *roots* or *zeros* of function  $f(\mathbf{x})$ , i.e., the solutions of equation  $f(\mathbf{x}) = 0$ , are all the points on a hypercurve in the N-D space, as the intersection of the hypersurface  $y = f(\mathbf{x})$  and the hyperplane  $y = 0$ , if they do intersect.

In general, no closed-form solution exists for a nonlinear equation system and the number of solutions is unknown. We therefore have to rely on numerical methods to find iteratively one or more such solutions if they do exist from an initial guess of the solution.

For example, in the special case of  $M = N = 2$ , functions  $f_1(x_1, x_2)$  and  $f_2(x_1, x_2)$  are two surfaces defined over the 2-D space spanned by  $x_1$  and  $x_2$ , and the roots of each of the two functions are on a curve as the intersection of the corresponding surface and the 2-D space. The solutions of the system composed of the two equations  $f_1 = 0$  and  $f_2 = 0$  are the intersection of these two curves, if they do intersect, otherwise no solution exists.

**Example:** Consider a simultaneous equation system:

$$\begin{cases} f_1(x, y) = x^2 + y^2 - 2 = 0 \\ f_2(x, y) = x + y - C = 0 \end{cases}$$

The first function  $f_1(x, y)$  is a parabolic cone centrally symmetric to the vertical axis, and its roots form a circle  $x^2 + y^2 = 2$  on the x-y plane centered at the origin (0, 0) with radius 1; the second function  $f_2(x, y)$  is a plane through

the origin, and its roots form a straight line  $y = C - x$  on the x-y plane. The solutions of the equation system are where the two curves intersect:

- If  $C = 0$ , there are two solutions  $(1, -1)$  and  $(-1, 1)$ ;
- If  $C = 2$ , there is only one solution  $(1, 1)$ ;
- If  $C = 3$ , the two curves do not intersect, i.e., no solution exists.

## 1.1 Linear Equation Systems

Consider a linear equation system of  $M$  equations and  $N$  variables:

$$\begin{cases} \sum_{j=1}^N a_{1j}x_j - b_1 = 0 \\ \dots \\ \sum_{j=1}^N a_{Mj}x_j - b_M = 0 \end{cases} \quad (1.2)$$

which can be expressed in vector form:

$$\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0} \quad (1.3)$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{M1} & \dots & a_{MN} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_M \end{bmatrix} \quad (1.4)$$

In general, the existence and uniqueness of the solution  $\mathbf{x}$  can be determined by the *fundamental theorem of linear algebra*, (see Table 3 in Section A.19), based on the rank  $R$  of the coefficient matrix  $\mathbf{A}$  and whether the system is underdetermined (underconstrained) if  $M < N$ , or overdetermined (overconstrained) if  $M > N$ .

Specially if  $R = M = N$ ,  $\mathbf{A}$  is full rank square matrix and its inverse  $\mathbf{A}^{-1}$  exists. Then the system has a unique solution  $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ .

But if  $M > N = R$  and  $\mathbf{b}$  is not in the column space of  $\mathbf{A}$ , the system is overconstrained without a solution. In this case we can still find an optimal approximated solution. We first define the *residual* of the system as  $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$ , and then define the *sum-of-squares error (SSE)* as:

$$\begin{aligned} \varepsilon(\mathbf{x}) &= \frac{1}{2} \|\mathbf{r}\|^2 = \frac{1}{2} \mathbf{r}^T \mathbf{r} = \frac{1}{2} (\mathbf{b} - \mathbf{A}\mathbf{x})^T (\mathbf{b} - \mathbf{A}\mathbf{x}) \\ &= \frac{1}{2} (\mathbf{b}^T \mathbf{b} - \mathbf{x}^T \mathbf{A}^T \mathbf{b} - \mathbf{b}^T \mathbf{A}\mathbf{x} + \mathbf{x}^T \mathbf{A}^T \mathbf{A}\mathbf{x}) \end{aligned} \quad (1.5)$$

To find the optimal solution  $\mathbf{x}^*$  that minimizes  $\varepsilon$ , we set its derivative with respect to  $\mathbf{x}$  to zero (see section A.18 for differentiation with respect to a vector variable):

$$\begin{aligned} \frac{d}{d\mathbf{x}} \varepsilon(\mathbf{x}) &= \frac{1}{2} \frac{d}{d\mathbf{x}} \|\mathbf{r}\|^2 = \frac{1}{2} \frac{d}{d\mathbf{x}} (\mathbf{b}^T \mathbf{b} - \mathbf{x}^T \mathbf{A}^T \mathbf{b} - \mathbf{b}^T \mathbf{A}\mathbf{x} + \mathbf{x}^T \mathbf{A}^T \mathbf{A}\mathbf{x}) \\ &= -\mathbf{A}^T \mathbf{b} + \mathbf{A}^T \mathbf{A}\mathbf{x} = 0 \end{aligned} \quad (1.6)$$



Solving this matrix equation we get

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} = \mathbf{A}^- \mathbf{b} \quad (1.7)$$

where  $\mathbf{A}^- = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$  is the *pseudo-inverse* (Section A.15) of the non-square matrix  $\mathbf{A}$ .

If the  $M$  equations are barely independent, is the  $N \times N$  matrix  $\mathbf{A}^T \mathbf{A}$  it still has a full rank  $R = N$  and is invertible, but it is a near-singular matrix. In this case some of its eigenvalues is very close to zero, and correspondingly some of the eigenvalues of its inverse  $(\mathbf{A}^T \mathbf{A})^{-1}$  may be huge. Consequently the system becomes ill-conditioned or ill-posed, in the sense that a small change in the system (in either  $\mathbf{A}$  or  $\mathbf{b}$ ) due to noise may cause a large change in the solution  $\mathbf{x}$ .

This ill-posed problem can be addressed by *regularization*, by which the solution  $\mathbf{x}$  is under control so that it will not take unreasonably high values. Specifically we construct an *objective function* that contains a penalty term for large  $\mathbf{x}$ , as well as the error term  $\varepsilon = \|\mathbf{r}\|^2/2$ :

$$J(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}\|^2 + \frac{\lambda}{2} \|\mathbf{x}\|^2 \quad (1.8)$$

By minimizing  $J(\mathbf{x})$ , we can obtain a solution  $\mathbf{x}$  of small norm  $\|\mathbf{x}\|$  as well as low error  $\varepsilon(\mathbf{x})$ . Same as before, the solution  $\mathbf{x}$  can be obtained by setting the derivative of  $J(\mathbf{x})$  to zero

$$\frac{d}{d\mathbf{x}} J(\mathbf{x}) = -\mathbf{A}^T \mathbf{b} + \mathbf{A}^T \mathbf{A} \mathbf{x} + \lambda \mathbf{x} = \mathbf{0} \quad (1.9)$$

and solving the resulting equation to get:

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{A}^T \mathbf{b} \quad (1.10)$$

We see that even if  $\mathbf{A}^T \mathbf{A}$  is near singular, matrix  $\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I}$  is not due to the additional term  $\lambda \mathbf{I}$ .

By adjusting the hyperparameter  $\lambda$ , we can make a proper tradeoff between accuracy and stability.

- Small  $\lambda$ : the solution is more accurate but less stable as it is more prone to noise, i.e., the variance error may be large. This is called *overfitting*;
- Large  $\lambda$ : the solution is less accurate but more stable as it is less affected by noise. This is called *underfitting*.

The issue of overfitting versus underfitting is of central importance in machine learning in general, and will be more formally addressed while discussing various regression and classification algorithms in some later Chapters.

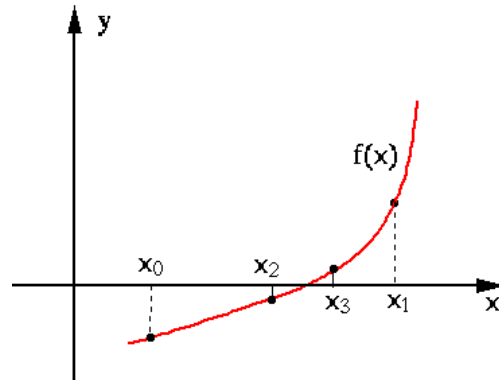


Figure 1.1 Bisection Search

## 1.2 The Bisection and Secant methods

Here we consider a set of methods that find the solution  $x^*$  of a single-variable nonlinear equation  $f(x) = 0$ , by searching iteratively through a neighborhood of the domain, in which  $x^*$  is known to be located.

### The bisection search

This method requires two initial guesses  $x_0 < x_1$  satisfying  $f(x_0)f(x_1) < 0$ . As  $f(x_0)$  and  $f(x_1)$  are on opposite sides of the  $x$ -axis  $y = 0$ , the solution  $x^*$  at which  $f(x^*) = 0$  must reside somewhere in between of these two guesses, i.e.,  $x_0 < x^* < x_1$ .

Given such two end points  $x_0$  and  $x_1$ , we find another point  $x_2$  in between and use it to replace one of the end points at which the function value  $f(x)$  has the same sign as that of  $f(x_2)$ . The new search interval is  $a = x_2 - x_0$  if  $f(x_2)f(x_1) > 0$ , or  $b = x_1 - x_2$  if  $f(x_0)f(x_2) > 0$ , either of which is smaller than the previous interval  $a + b = x_1 - x_0$ . This process is then carried out iteratively until the solution is eventually approached, with a guaranteed convergence.

While any point between the two end points can be chosen for the next iteration, we want to avoid the worst possible case in which the solution always happens to be in the larger of the two sections  $a$  and  $b$ . We can therefore choose the middle point between the two end points, i.e.,  $x_2 = (x_0 + x_1)/2$ , so that  $a = b = (x_1 - x_0)/2$ , i.e., the new search interval is always halved in each step of the iteration:

$$x_{n+1} = \frac{x_n + x_{n-1}}{2} \quad (1.11)$$

Here, without loss of generality, we have assumed  $f(x_{n-1})f(x_{n+1}) > 0$  and  $x_{n-1}$  is replaced by  $x_{n+1}$ .

For example, we assume the root is located at  $x^* = 2.2$  between the two initial values are  $x_0 = 0$  and  $x_1 = 8$ , then we get

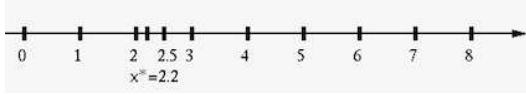


Figure 1.2

$n$	0	1	2	3	4	5	6	7	8
$x_n$	0	8	4	2	3	2.5	2.25	2.125	...
$ e_n $	2.2	5.8	1.8	0.2	0.8	0.3	0.05	0.075	...

(1.12)

Note that the error  $|e_n| = |x_n - x^*|$  does not necessarily always reduce monotonically. However, as in each iteration the search interval is always halved, the worst possible error is also halved:

$$|e_{n+1}| = |x_{n+1} - x^*| \leq \frac{|x_n - x_{n-1}|}{2} = \frac{|x_0 - x_1|}{2^n} \approx \frac{|e_n|}{2} \quad (1.13)$$

i.e., the convergence of the iteration is linear (of order  $p = 1$ ) and the rate of convergence is  $1/2$ :

$$\lim_{n \rightarrow \infty} \frac{|e_{n+1}|}{|e_n|^p} = \lim_{n \rightarrow \infty} \frac{|e_{n+1}|}{|e_n|} \leq \mu = \frac{1}{2} \quad (1.14)$$

Compared to other methods to be considered later, the bisection method converges rather slowly, but one of the advantages of the bisection method is that no derivative of the given function is needed. This means the given function  $f(x)$  does not need to be differentiable.

### The Secant method

Same as in the bisection method, here again we assume there are two initial values  $x_0$  and  $x_1$  available, but they do not have to satisfy  $f(x_0)f(x_1) < 0$ . Here the iteration is based on the zero-crossing of the secant line passing through the two points  $f(x_0)$  and  $f(x_1)$ , instead of their middle point. The equation for the secant is:

$$\frac{y - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \quad (1.15)$$

At the zero crossing,  $y = 0$  and the equation above can be solved for  $x$ , the position of the zero-crossing:

$$x = x_0 - \frac{x_1 - x_0}{f(x_1) - f(x_0)} f(x_0) = x_0 - \frac{f(x_0)}{\hat{f}'(x_0)} \quad (1.16)$$

where  $\hat{f}'(x_0)$  is the finite difference that approximates the derivative  $f'(x_0)$ :

$$\hat{f}'(x_0) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{\Delta f(x_0)}{\Delta x_0} \xrightarrow{\Delta x_0 \rightarrow 0} f'(x_0) \quad (1.17)$$

The new value  $x$  obtained above, as a better estimate of the root than either  $x_0$  or  $x_1$ , is used to replace one of them:

- If  $f(x_0)f(x_1) < 0$ , then  $x_i$  ( $i = 0, 1$ ) satisfying  $f(x_i)f(x) > 0$  is replaced by  $x_2 = x$  (same as the bisection method);
- If  $f(x_0)f(x_1) > 0$ , then  $x_i$  ( $i = 0, 1$ ) with greater  $|f(x_i)|$  is replaced by  $x_2 = x$ .

This process is then carried out iteratively to approach the root:

$$x_{n+1} = x_n - \frac{f(x_n)}{\hat{f}'(x_n)} \quad (1.18)$$

In case  $f(x_n) = f(x_{n-1})$ , the approximated derivative is zero  $\hat{f}'(x_n) = 0$ , and  $x_{n+1}$  cannot be found. In such a case, a root may exist between  $x_n$  and  $x_{n-1}$ . Therefore the way to resolve this problem is to combine the bisection search with the secant method so that when  $\hat{f}' = 0$ , the algorithm will switch to bisection search. This is Dekker's method:

$$x_{n+1} = \begin{cases} x_n - f(x_n)/\hat{f}'(x_n) & \text{if } f(x_n) \neq f(x_{n-1}) \\ (x_n + x_{n-1})/2 & \text{otherwise} \end{cases} \quad (1.19)$$

We now consider the order of convergence of the secant method. Let  $x^*$  be the root at which  $f(x^*) = 0$ . The error of  $x_{n+1}$  is:

$$\begin{aligned} e_{n+1} &= x_{n+1} - x^* = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) - x^* \\ &= \frac{(x_{n-1} - x^*)f(x_n) - (x_n - x^*)f(x_{n-1})}{f(x_n) - f(x_{n-1})} = \frac{e_{n-1}f(x_n) - e_n f(x_{n-1})}{f(x_n) - f(x_{n-1})} \end{aligned} \quad (1.20)$$

Consider the Taylor expansion of  $f(x_n)$ :

$$f(x_n) = f(x^* + e_n) = f(x^*) + f'(x^*)e_n + \frac{1}{2}f''(x^*)e_n^2 + O(e_n^3) = f'(x^*)e_n + \frac{1}{2}f''(x^*)e_n^2 + O(e_n^3) \quad (1.21)$$

(as  $f(x^*) = 0$ ) and similarly

$$f(x_{n-1}) = f'(x^*)e_{n-1} + \frac{1}{2}f''(x^*)e_{n-1}^2 + O(e_{n-1}^3) \quad (1.22)$$

Substituting these into the expression for  $e_{n+1}$  above we get

$$\begin{aligned} e_{n+1} &= \frac{e_{n-1}[f'(x^*)e_n + \frac{1}{2}f''(x^*)e_n^2 + O(e_n^3)] - e_n[f'(x^*)e_{n-1} + \frac{1}{2}f''(x^*)e_{n-1}^2 + O(e_{n-1}^3)]}{[f'(x^*)e_n + \frac{1}{2}f''(x^*)e_n^2 + O(e_n^3)] - [f'(x^*)e_{n-1} + \frac{1}{2}f''(x^*)e_{n-1}^2 + O(e_{n-1}^3)]} \\ &= \frac{e_{n-1}e_n f''(x^*)(e_n - e_{n-1})/2 + O(e_{n-1}^4)}{(e_n - e_{n-1})f'(x^*) + (e_n^2 - e_{n-1}^2)f''(x^*)/2 + O(e_{n-1}^3)} \\ &= \frac{e_{n-1}e_n f''(x^*)/2 + O(e_{n-1}^3)}{f'(x^*) + (e_n + e_{n-1})f''(x^*)/2 + O(e_{n-1}^2)} \\ &= \frac{e_{n-1}e_n f''(x^*)/2 + O(e_{n-1}^3)}{f'(x^*) + O(e_n) + O(e_{n-1}^2)} \end{aligned} \quad (1.23)$$

When  $n \rightarrow \infty$ , the lowest order terms in both the numerator and denominator

become the dominant terms as all other higher order terms approach to zero, and we have

$$e_{n+1} = \frac{e_{n-1}e_n f''(x^*)}{2f'(x^*)} = C e_n e_{n-1} \quad (1.24)$$

where we have defined a constant  $C = f''(x^*)/2f'(x^*)$ . To find the order of convergence, we need to find  $p$  in

$$|e_{n+1}| \leq |C| |e_{n-1}| |e_n| = \mu |e_n|^p \quad (1.25)$$

Solving this equation for  $|e_n|$  we get

$$|e_n| = \left( \frac{|C|}{\mu} |e_{n-1}| \right)^{1/(p-1)} = \left( \frac{|C|}{\mu} \right)^{1/(p-1)} |e_{n-1}|^{1/(p-1)} \quad (1.26)$$

On the other hand, when  $n \rightarrow \infty$  we also have

$$|e_n| = \mu |e_{n-1}|^p \quad (1.27)$$

Equating the right-hand sides of the two equations above we get

$$\left( \frac{|C|}{\mu} \right)^{1/(p-1)} |e_{n-1}|^{1/(p-1)} = \mu |e_{n-1}|^p \quad (1.28)$$

which requires the following two equations to hold:

$$p = \frac{1}{p-1}, \quad \mu = \left( \frac{|C|}{\mu} \right)^{1/(p-1)} = \left( \frac{|C|}{\mu} \right)^p \quad (1.29)$$

These two equations can be solved separately to get

$$p = \frac{1 + \sqrt{5}}{2} = 1.618, \quad \mu = |C|^{p/(p+1)} = C^{(\sqrt{5}-1)/2} = |C|^{0.618} \quad (1.30)$$

i.e.,

$$|e_{n+1}| = \mu |e_n|^p = |C|^{0.618} |e_n|^{1.618} = \left| \frac{f''(x)}{2f'(x)} \right|^{0.618} |e_n|^{1.618} \quad (1.31)$$

or

$$\lim_{n \rightarrow \infty} \frac{|e_{n+1}|}{|e_n|^{1.618}} = \mu = |C|^{0.618} \quad (1.32)$$

We see that the rate of convergence for the secant method is  $\mu = |C|^{0.618}$  and the order of convergence is  $p = 1.618$ , which is better than the linear convergence of the bisection search (which does not use the information of the specific function  $f(x)$ ), but worse than quadratic convergence of the Newton-Raphson method based on the function derivative  $f'(x)$  instead of the approximation  $\hat{f}'(x)$ , to be considered in the next section.

### The inverse quadratic interpolation

Similar to the secant method that approximates the given function  $f(x)$  by a straight line that goes through two consecutive points  $\{x_n, f(x_n)\}$  and  $\{x_{n-1}, f(x_{n-1})\}$ , the inverse quadratic interpolation method approximates the function by a quadratic

curve that goes through three consecutive points  $\{x_i, f(x_i)\}$ , ( $i = n, n-1, n-2$ ). As the function may be better approximated by a quadratic curve rather than a straight line, the iteration is likely to converge more quickly.

In general, any function  $y = f(x)$  can be approximated by a quadratic function  $q(x)$  based on three points at  $y_0 = f(x_0)$ ,  $y_1 = f(x_1)$ , and  $y_2 = f(x_2)$  by the *Lagrange interpolation*:

$$y = q(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2 \quad (1.33)$$

However, here we use the interpolation  $x = q^{-1}(y)$  to approximate the inverse (horizontal) function  $x = f^{-1}(y)$ :

$$x = q^{-1}(y) = \frac{(y-y_1)(y-y_2)}{(y_0-y_1)(y_0-y_2)}x_0 + \frac{(y-y_0)(y-y_2)}{(y_1-y_0)(y_1-y_2)}x_1 + \frac{(y-y_0)(y-y_1)}{(y_2-y_0)(y_2-y_1)}x_2 \quad (1.34)$$

Obviously when  $y = 0$ , the corresponding  $x$  is the zero-crossing of  $x = q^{-1}(y)$ :

$$x_3 = q^{-1}(y)|_{y=0} = \frac{y_1y_2}{(y_0-y_1)(y_0-y_2)}x_0 + \frac{y_0y_2}{(y_1-y_0)(y_1-y_2)}x_1 + \frac{y_0y_1}{(y_2-y_0)(y_2-y_1)}x_2 \quad (1.35)$$

which can be used as an estimate of the root  $x^*$  of  $y = f(x)$ . This expression can then be converted into an iteration by which the next root estimate  $x_{n+1}$  is computed based on the previous three estimates at  $x_n$ ,  $x_{n-1}$ , and  $x_{n-2}$ .

### 1.3 Fixed point iteration

We first consider the special case of solving the equation  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  when  $M = N = 1$ . To find a solution  $x^*$  that satisfies the equation  $f(x) = 0$ , we can first convert it into an equivalent equation  $g(x) = x$ , in the sense that an  $x$  satisfying one of the equations will also satisfy the other, and then carry out an iteration  $x_{n+1} = g(x_n)$  from some initial value  $x_0$ . If the iteration converges at a point  $x^*$ , i.e.,  $g(x^*) = x^*$ , then we also have  $f(x^*) = 0$ , i.e.,  $x^*$  is a solution of the equation  $f(x) = 0$ . Consider the following examples:

#### Example 1

To solve the equation

$$f(x) = \log(x) - 0.5 = 0$$

we can first construct another function

$$g(x) = x - f(x) = x - \log(x) + 0.5$$

so that the equation  $g(x) = x$  is indeed equivalent to the given equation  $f(x) = 0$ . We then carry out the iteration  $x_{n+1} = g(x_n)$  from some initial value, such as  $x_0 = 0.5$ , and get:

$n$	$x_n$	$g(x_n)$	$f(x_n)$
0	0.50000	1.69315	-1.19315
1	1.69315	1.66656	0.02659
2	1.66656	1.65580	0.01076
3	1.65580	1.65152	0.00428
4	1.65152	1.64982	0.00169
5	1.64982	1.64915	0.00067
6	1.64916	1.64889	0.00026
7	1.64889	1.64879	0.00010
8	1.64879	1.64875	0.00004
9	1.64875	1.64873	0.00002
10	1.64873	1.64873	0.00001
11	1.64873	1.64872	0.00000
12	1.64872	1.64872	0.00000

We see that the iteration converges to  $x^* = \lim_{n \rightarrow \infty} g(x_n) = 1.64872$  satisfying  $g(x^*) = x^* - f(x^*) = x^*$ , and, equivalently,  $x^*$  is also the solution of the given equation  $f(x) = 0.5 - \log(x) = 0$ :

$$0.5 - \log(1.64872) = 0, \quad \text{i.e.,} \quad e^{0.5} = \sqrt{e} = 1.64872$$

Alternatively, we could construct a different function also equivalent to  $f(x) = 0$ :

$$g(x) = x + f(x) = x + \log(x) - 0.5$$

But this iteration no longer converges. Why does it not work?

### Example 2

$$f(x) = e^x - 1/x = 0$$

We first convert this equation into an equivalent form

$$g(x) = x = e^{-x}$$

and then carry out the iteration:

$$x_{n+1} = g(x_n) = e^{-x_n} \xrightarrow{n \rightarrow \infty} x^* = 0.5671$$

which is the root of the given equation  $f(x) = e^x - 1/x = 0$ , i.e.,  $e^{-0.5671} = 0.5671$ .

Alternatively, the given equation can also be converted into a different form  $g(x) = x = -\ln(x)$ . However, the iteration based on this function no longer converges.

Why does this iterative method work in some cases but fails in others? To answer these questions we need to understand the theory behind the method, the *fixed point* of a contraction function.

If a single variable function  $g(x)$  satisfies

$$|g(x_1) - g(x_2)| \leq k|x_1 - x_2|, \quad k \geq 0 \quad (1.36)$$

it is *Lipschitz continuous*, and  $k$  is a *Lipschitz constant*. Specially, if  $k = 1$ , then  $g(x)$  is a *non-expansive* function, if  $0 \leq k < 1$ , then  $g(x)$  is a *contraction function* or simply a *contraction*. These concepts can be generated to functions of multivariables  $\mathbf{x} = [x_1, \dots, x_N]^T \in \mathbb{R}^N$  (a point in an N-D metric space  $V$ ):

$$g_n(\mathbf{x}) = g_n(x_1, \dots, x_N), \quad (n = 1, \dots, N) \quad (1.37)$$

which can also be expressed in vector form:

$$\mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}), \dots, g_N(\mathbf{x})]^T \quad (1.38)$$

**Definition:** In a metric space  $V$  with certain distance  $d(\mathbf{x}, \mathbf{y})$  defined between any two points  $\mathbf{x}, \mathbf{y} \in V$ , a function  $\mathbf{g} : V \rightarrow V$  is a *contraction* if

$$d(\mathbf{g}(\mathbf{x}), \mathbf{g}(\mathbf{y})) \leq k d(\mathbf{x}, \mathbf{y}) \quad (1.39)$$

The smallest  $k$  value that satisfies the above is called the (*best*) *Lipschitz constant*.

Intuitively, a contraction reduces the distance between points in the space, i.e., it brings them closer together. A function  $\mathbf{g}(\mathbf{x})$  may not be a contraction through out its entire domain, but it can be a contraction in the neighborhood of a certain point  $\mathbf{x}^* \in V$ , in which any  $\mathbf{x}$  is sufficiently close to  $\mathbf{x}^*$  so that

$$d(\mathbf{g}(\mathbf{x}), \mathbf{g}(\mathbf{x}^*)) \leq k d(\mathbf{x}, \mathbf{x}^*), \quad 0 \leq k < 1, \quad \text{when } d(\mathbf{x}, \mathbf{x}^*) < \epsilon > 0 \quad (1.40)$$

**Definition:** A *fixed point*  $\mathbf{x}^*$  of a function  $\mathbf{g}(\mathbf{x})$  is a point in its domain that is mapped to itself:

$$\mathbf{x}^* = \mathbf{g}(\mathbf{x}^*) \quad (1.41)$$

We immediately have

$$\mathbf{g}(\mathbf{g}(\mathbf{x}^*)) = \mathbf{g}^2(\mathbf{x}^*) = \mathbf{x}^*, \dots, \mathbf{g}(\mathbf{g}(\mathbf{g}(\dots \mathbf{g}(\mathbf{x}^*) \dots))) = \mathbf{g}^n(\mathbf{x}^*) = \mathbf{x}^* \quad (1.42)$$

A fixed point  $\mathbf{x}^*$  is an *attractive fixed point* if any point  $x$  in its neighborhood converges to  $\mathbf{x}^*$ , i.e.,  $\lim_{n \rightarrow \infty} \mathbf{g}^n(\mathbf{x}) = \mathbf{x}^*$ .

**Fixed Point Theorem :** Let  $\mathbf{g}(\mathbf{x})$  be a contraction function satisfying

$$d(\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})) \leq k d(\mathbf{x} - \mathbf{y}) \quad (0 \leq k < 1) \quad (1.43)$$

then there exists a unique fixed point  $\mathbf{x}^* = \mathbf{g}(\mathbf{x}^*)$ , which can be found by an iteration from an arbitrary initial point  $\mathbf{x}_0$ :

$$\lim_{n \rightarrow \infty} \mathbf{x}_n = \lim_{n \rightarrow \infty} \mathbf{g}(\mathbf{x}_{n-1}) = \lim_{n \rightarrow \infty} \mathbf{g}^2(\mathbf{x}_{n-2}) = \dots = \lim_{n \rightarrow \infty} \mathbf{g}^n(\mathbf{x}_0) = \mathbf{x}^* \quad (1.44)$$

**Proof**



Here the distance  $d(\mathbf{x} - \mathbf{y})$  is specifically defined as the  $p$ -norm (section A.16) of the vector  $\mathbf{x} - \mathbf{y}$ :

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p = \left( \sum_{n=1}^N (x_n - y_n)^p \right)^{1/p} \quad (1.45)$$

where  $p \geq 1$ , e.g.,  $p = 1, 2, \infty$ . Also, for convenience, we can drop  $p$  so that  $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ .

- We first prove constructively the existence of a fixed point. As  $\mathbf{g}(\mathbf{x})$  is a contraction, we have

$$\|\mathbf{x}_{n+1} - \mathbf{x}_n\| = \|\mathbf{g}(\mathbf{x}_n) - \mathbf{g}(\mathbf{x}_{n-1})\| \leq k \|\mathbf{x}_n - \mathbf{x}_{n-1}\| \leq k^2 \|\mathbf{x}_{n-1} - \mathbf{x}_{n-2}\| \leq \dots \leq k^n \|\mathbf{x}_1 - \mathbf{x}_0\| \quad (1.46)$$

As  $0 \leq k < 1$ , we have  $\lim_{n \rightarrow \infty} \|\mathbf{x}_{n+1} - \mathbf{x}_n\| = 0$ . This is a *Cauchy sequence* (Section A.1) that converges to some point  $\lim_{n \rightarrow \infty} \mathbf{x}_n = \mathbf{x}^*$  also in the space. We further have

$$\mathbf{g}(\mathbf{x}^*) = \mathbf{g}(\lim_{n \rightarrow \infty} \mathbf{x}_n) = \lim_{n \rightarrow \infty} \mathbf{g}(\mathbf{x}_n) = \lim_{n \rightarrow \infty} \mathbf{x}_{n+1} = \mathbf{x}^* \quad (1.47)$$

i.e., the limit of the Cauchy sequence  $\lim_{n \rightarrow \infty} \mathbf{x}_n = \mathbf{x}^*$  is a fixed point.

- We next prove the uniqueness of the fixed point. Let  $\mathbf{x}_1^*$  and  $\mathbf{x}_2^*$  be two fixed points of  $\mathbf{g}(\mathbf{x})$ , then we have

$$\|\mathbf{g}(\mathbf{x}_1^*) - \mathbf{g}(\mathbf{x}_2^*)\| \leq k \|\mathbf{x}_1^* - \mathbf{x}_2^*\| = k \|\mathbf{g}(\mathbf{x}_1^*) - \mathbf{g}(\mathbf{x}_2^*)\| \quad (1.48)$$

For any  $k \neq 0$ , the above holds only if  $\|\mathbf{x}_1^* - \mathbf{x}_2^*\| = \|\mathbf{g}(\mathbf{x}_1^*) - \mathbf{g}(\mathbf{x}_2^*)\| = 0$ , i.e.,  $\mathbf{x}_1^* = \mathbf{x}_2^*$  is the unique fixed point.

QED

**Theorem:** Let  $\mathbf{x}^* = \mathbf{g}(\mathbf{x}^*)$  be a fixed point of a differentiable function  $\mathbf{g}(\mathbf{x})$ , i.e.,  $\partial g_i / \partial x_j$  exists for any  $1 \leq i, j \leq N$ . If the norm of the Jacobian matrix is smaller than 1,  $\|\mathbf{J}_{\mathbf{g}}(\mathbf{x}^*)\| < 1$ , then  $\mathbf{g}(\mathbf{x})$  is a contraction at  $\mathbf{x}^*$ .

The Jacobian matrix of  $\mathbf{g}(\mathbf{x})$  is defined as

$$\mathbf{g}'(\mathbf{x}) = \mathbf{J}_{\mathbf{g}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \dots & \frac{\partial g_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_N}{\partial x_1} & \dots & \frac{\partial g_N}{\partial x_N} \end{bmatrix} \quad (1.49)$$

**Proof:**

Consider the Taylor expansion (Section A.22) of the function  $\mathbf{g}(\mathbf{x})$  in the neighborhood of  $\mathbf{x}^*$ :

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}^*) + \mathbf{g}'(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*) + R(\mathbf{x} - \mathbf{x}^*) = \mathbf{g}(\mathbf{x}^*) + \mathbf{J}_{\mathbf{g}}(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*) + R(\mathbf{x} - \mathbf{x}^*) \quad (1.50)$$

where  $R(\mathbf{x} - \mathbf{x}^*)$  is the remainder composed of second and higher order terms of  $\delta = \mathbf{x} - \mathbf{x}^*$ . Subtracting  $\mathbf{g}(\mathbf{x}^*)$  and taking any  $p$ -norm on both sides, we get

$$\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}^*)\|_p = \|\mathbf{J}_{\mathbf{g}}(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*) + R(\mathbf{x} - \mathbf{x}^*)\|_p \quad (1.51)$$

When  $\mathbf{x} \rightarrow \mathbf{x}^*$ , the second and higher order terms of  $\mathbf{x} - \mathbf{x}^*$  disappear and  $R(\mathbf{x} - \mathbf{x}^*) \rightarrow 0$ , we have

$$\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}^*)\|_p = \|\mathbf{J}_{\mathbf{g}}(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*)\|_p \leq \|\mathbf{J}_{\mathbf{g}}(\mathbf{x}^*)\|_p \cdot \|\mathbf{x} - \mathbf{x}^*\|_p, \quad (1.52)$$

The inequality is due to the *Cauchy-Schwarz inequality* (Section A.1) if  $\|\mathbf{J}_{\mathbf{g}}(\mathbf{x}^*)\| < 1$ , the function  $\mathbf{g}(\mathbf{x})$  is a contraction at  $\mathbf{x}^*$ .

QED

In particular, for a single-variable function in an  $N = 1$  dimensional space, we have

$$g(x) - g(x^*) = g'(x^*)(x - x^*) + \frac{1}{2}g''(x^*)(x - x^*)^2 + R(x - x^*) \quad (1.53)$$

and

$$|g(x) - g(x^*)| \leq |g'(x^*)(x - x^*)| \leq |g'(x^*)| |x - x^*| \quad (1.54)$$

If  $|g'(x)| < 1$ , then  $g(x)$  is a contraction at  $x^*$ .

Now we understand why in the examples of the previous section the iteration leads to convergence in some cases but divergence in other cases: if  $|g'(x)| < 1$ , the iteration will converge to the root  $x^*$  of  $f(x) = 0$ , but if  $|g'(x)| > 1$ , it never will converge.

The iterative process  $x_{n+1} = g(x_n)$  for finding the fixed point of a single-variable function  $g(x)$  can be shown graphically as the intersections of the function  $y = g(x)$  and the identity function  $y = x$ , as shown below. The iteration converges in the first two cases as  $|g'(x)| < 1$ , but it diverges in the last two cases as  $|g'(x)| > 1$ .

We next find the order of convergence of the fixed point iteration. Consider

$$\lim_{n \rightarrow \infty} \frac{|e_{n+1}|}{|e_n|} = \lim_{n \rightarrow \infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|} = \lim_{n \rightarrow \infty} \frac{|g(x_n) - g(x^*)|}{|x_n - x^*|} = |g'(x^*)| = \mu < 1 \quad (1.55)$$

We see that in general the fixed point iteration converges linearly. However, if the iteration function  $g(x)$  has zero derivative at the fixed point  $g'(x^*) = 0$ , we have

$$g(x) - g(x^*) = g'(x^*)(x - x^*) + \frac{1}{2}g''(x^*)(x - x^*)^2 + R(x - x^*) = \frac{1}{2}g''(x^*)(x - x^*)^2 + R(x - x^*) \quad (1.56)$$

and the iteration converges quadratically:

$$\lim_{n \rightarrow \infty} \frac{|e_{n+1}|}{|e_n|^2} = \lim_{n \rightarrow \infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^2} = \lim_{n \rightarrow \infty} \frac{|g(x_n) - g(x^*)|}{|x_n - x^*|^2} = \frac{1}{2}|g''(x^*)| = \text{constant} \quad (1.57)$$

Moreover, if  $g''(x^*) = 0$ , then the iteration converges cubically.

Consider a specific iteration function in the form of  $g(x) = x - f(x)\phi(x)$ , which is equivalent to  $f(x) = 0$ , as if  $x^*$  is the root of  $f(x)$  satisfying  $f(x^*) = 0$ , it also satisfies  $g(x^*) = x^*$ , which is indeed a fixed point of  $g(x)$ . The derivative of this function is

$$g'(x) = (x - f(x)\phi(x))' = 1 - f'(x)\phi(x) - f(x)\phi'(x) = 1 - f'(x)\phi(x) \quad (1.58)$$

If  $f'(x) \neq 0$ , we can define  $\phi(x) = 1/f'(x)$  so that  $g'(x) = 1 - f'(x)/f'(x) = 0$ , then the convergence becomes quadratic. This is actually the Newton-Raphson method discussed in the following section.

**Example 1**

Find the solution of the following equation:

$$f(x) = x^3 - x - 2 = 0$$

This equation can be converted into an equivalent form of  $g(x) = x$  in two different ways:

$$g_1(x) = x = \sqrt[3]{x+2}, \quad \text{and} \quad g_2(x) = x = x^3 - 2$$

In the figure below, the two functions  $g_1(x)$  (left) and  $g_2(x)$  (right) together with  $f(x)$  and the identity function are plotted:

The iteration based on  $g_1(x)$  converges to the solution  $x^* = 1.5214$  for any initial guess  $-\infty < x_0 < \infty$ , as  $g_1(x)$  is a contraction. However, the iteration based on  $g_2(x)$  does not converge to  $x^*$  as it is not a contraction in the neighborhood of  $x^*$ . In fact, the iteration will diverge towards either  $-\infty$  if  $x_0 < x^*$  or  $\infty$  if  $x_0 > x^*$ .

**Example 2**

To solve the following equation

$$f(x) = x^3 - 3x + 1 = 0$$

we first convert it into the form of  $g(x) = x$  in two different ways:

$$g_1(x) = x = \sqrt[3]{3x-1}, \quad g_2(x) = x = \frac{1}{3}(x^3 - 1)$$

As can be seen in the plots, this equation has three solutions,

$$x_1 = -1.8794, \quad x_2 = 0.3473, \quad x_3 = 1.5321$$

of which  $x_1$  and  $x_3$  can be obtained by the iteration based on  $g_1(x)$  and  $x_2$  can be obtained by the iteration based on  $g_2(x)$ . But neither of them can find all three roots.

- As shown in the plot on the left,  $|g_1'(x)| < 1$  for all  $-\infty < x < \infty$  except in the neighborhood of  $x_2 = 0.3473$ , i.e.,  $g_1(x)$  is a contraction mapping everywhere except around  $x_2$ . Therefore the iteration starting from any initial guess  $x_0$  will converge to either  $x_1 = -1.8749$  if  $x_0 < x_2$ , or  $x_3 = 1.5321$  if  $x_0 > x_2$ .

$$x_{n+1} = g_1(x_n) \xrightarrow{n \rightarrow \infty} \begin{cases} x_1 = -1.8749 & x_0 < x_2 \\ x_3 = 1.5321 & x_0 > x_2 \end{cases}$$

However, as  $g_1(x)$  is not a contraction mapping around  $x = x_2$ , the iteration will never converge to  $x_2$ .

- As shown in the plot on the right,  $|g_2'(x)| > 1$  for all  $-\infty < x < \infty$  except in the neighborhood of  $x_2$ , i.e.,  $g_2(x)$  is not a contraction mapping around either  $x_1$  or  $x_3$ . Therefore the iteration based on  $g_2(x)$  will not converge to either  $x_1$  or  $x_3$ , but it may converge to  $x_2$ , if the initial guess  $x_0$  is in the range  $x_1 < x_0 < x_3$ . However, if  $x_0$  is outside this range the iteration will diverge toward either  $-\infty$  if  $x_0 < x_1$  or  $\infty$  if  $x_0 > x_3$ .

$$x_{n+1} = g_2(x_n) \xrightarrow{n \rightarrow \infty} \begin{cases} -\infty & x_0 < x_1 \\ x_2 = 0.3473 & x_1 < x_0 < x_3 \\ \infty & x_0 > x_3 \end{cases}$$

$n$	$x_n$	$f(x_n)$
1	$-2.0408275e + 00$	$-1.3775173e + 00$
2	$-1.9240239e + 00$	$-3.5041089e - 01$
3	$-1.8919392e + 00$	$-9.6254050e - 02$
4	$-1.8829328e + 00$	$-2.7019276e - 02$
5	$-1.8803890e + 00$	$-7.6311660e - 03$
6	$-1.8796694e + 00$	$-2.1590449e - 03$
7	$-1.8794656e + 00$	$-6.1114717e - 04$
8	$-1.8794080e + 00$	$-1.7301761e - 04$
9	$-1.8793916e + 00$	$-4.8983741e - 05$
10	$-1.8793870e + 00$	$-1.3868146e - 05$
11	$-1.8793857e + 00$	$-3.9263250e - 06$
12	$-1.8793853e + 00$	$-1.1116151e - 06$

### Example 3

$$f(x) = \sin(x) - x^3 = 0$$

This equation can be converted into the form  $g(x) = x$  in different ways:

- 

$$g_0(x) = x = \sqrt[3]{\sin(x)}, \quad g_0'(x) = \frac{\cos(x)}{3 \sin(x)^{2/3}}$$

$n$	$x_n$	$f(x_n)$
0	$2.000000e + 00$	$-7.0907$
1	$9.688027e - 01$	$-0.085089$
2	$9.375886e - 01$	$-0.018075$
3	$9.306842e - 01$	$-0.0041048$
4	$9.291018e - 01$	$-0.00094611$
5	$9.287364e - 01$	$-0.00021881$
6	$9.286518e - 01$	$-5.0647e - 05$
7	$9.286322e - 01$	$-1.1725e - 05$
8	$9.286277e - 01$	$-2.7144e - 06$

- $$g_1(x) = x = \sin(x)/x^2, \quad g'_1(x) = \frac{\cos(x)}{x^2} - \frac{2 \sin(x)}{x^3}$$

- $$g_2(x) = x + \sin(x) - x^3, \quad g'_2(x) = \cos(x) - 3x^2 + 1$$

- $$g_3(x) = x - \frac{\sin(x) - x^3}{\cos(x) - 3x^2}, \quad g'_3(x) = -\frac{(6x + \sin(x))(\sin(x) - x^3)}{(\cos(x) - 3x^2)^2}$$

$n$	$x_n$	$f(x_n)$
0	2.000000e + 00	-7.0907
1	1.428913e + 00	-1.9276
2	1.106787e + 00	-0.46152
3	9.637850e - 01	-0.073886
4	9.304466e - 01	-0.0036294
5	9.286316e - 01	-1.0509e - 05
6	9.286263e - 01	-8.9029e - 11

**Example 4**

$$f(x) = e^x - \frac{1}{x}$$

- $$g_0(x) = x = e^{-x}, \quad g'_0(x) = e^{-x}$$

The iteration from any initial guess  $x_0 > 0$  will converge to  $x^* = g_1(x^*) = 0.56714$ .

- $$g_1(x) = x = -\log(x), \quad g'_1(x) = 1/x$$

Around  $x = x^* = 0.56714$ ,  $g'_1(x) = 1/x > 1$ , the iteration does not converge.

- $$g_2(x) = x - \frac{e^x - 1/x}{e^x + 1/x^2}, \quad g'_2(x) = \frac{(e^x - 1/x)(e^x - 2/x^3)}{(e^x + 1/x^2)^2}$$

**Example 5**

Consider a 3-variable linear vector function  $\mathbf{f}(\mathbf{x})$  of arguments  $\mathbf{x} = [x, y, z]^T$ :

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \quad \begin{cases} f_1(\mathbf{x}) = 6x + 3y + 2z - 18 = 0 \\ f_2(\mathbf{x}) = 2x + 7y + 3z - 25 = 0 \\ f_3(\mathbf{x}) = x + 3y + 5z - 22 = 0 \end{cases}$$

from which the g-function can be obtained:

$$\mathbf{g}(\mathbf{x}) = \mathbf{x}, \quad \begin{cases} g_1(\mathbf{x}) = x = -(3y + 2z - 18)/6 \\ g_2(\mathbf{x}) = y = -(2x + 3z - 25)/7 \\ g_3(\mathbf{x}) = z = -(x + 3y - 22)/5 \end{cases}$$

The Jacobian  $\mathbf{g}(\mathbf{x})$  of this linear system is a constant matrix

$$\mathbf{J}_g = \begin{bmatrix} 0 & -1/2 & -1/3 \\ -2/7 & 0 & -3/7 \\ -1/5 & -3/5 & 0 \end{bmatrix}$$

with the induced p=2 norm (maximum singular value)  $\|\mathbf{J}_g\| = 0.851 < 1$ . Consequently, the iteration  $\mathbf{x}_{n+1} = \mathbf{g}(\mathbf{x}_n)$  converges from an initial guess  $\mathbf{x}_0 = [1, 1, 1]^T$  to the solution  $\mathbf{x} = [1, 2, 3]^T$ .

Alternatively, the g-function can also be obtained as

$$\mathbf{g}'(\mathbf{x}) = \mathbf{x}, \quad \begin{cases} g'_1(\mathbf{x}) = x = -(3y + 5z - 22) \\ g'_2(\mathbf{x}) = y = -(2x + 3z - 25)/7 \\ g'_3(\mathbf{x}) = z = -(6x + 3y - 18)/2 \end{cases}$$

The Jacobian is

$$\mathbf{J}_{g'} = \begin{bmatrix} 0 & -3 & -5 \\ -2/7 & 0 & -3/7 \\ -3 & -3/2 & 0 \end{bmatrix}$$

with the induced p=2 norm  $\|\mathbf{J}_{g'}\| = 5.917 > 1$ . The iteration does not converge.

### Example 6

Consider a 3-variable nonlinear function  $\mathbf{f}(\mathbf{x})$  of arguments  $\mathbf{x} = [x, y, z]^T$ :

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \quad \begin{cases} f_1(\mathbf{x}) = x^2 - 2x + y^2 - z + 1 = 0 \\ f_2(\mathbf{x}) = xy^2 - x - 3y + yz + 2 = 0 \\ f_3(\mathbf{x}) = xz^2 - 3z + yz^2 + xy = 0 \end{cases}$$

The g-function can be obtained as

$$\mathbf{g}(\mathbf{x}) = \mathbf{x}, \quad \begin{cases} g_1(\mathbf{x}) = x = (x^2 + y^2 - z + 1)/2 \\ g_2(\mathbf{x}) = y = (xy^2 - x + yz + 2)/3 \\ g_3(\mathbf{x}) = z = (xz^2 + yz^2 + xy)/3 \end{cases}$$

With  $\mathbf{x}_0 = [0, 0, 0]^T$  and after  $n > 170$  iterations  $\mathbf{x}_{n+1} = \mathbf{g}(\mathbf{x}_n)$  converges to  $\mathbf{x}_n = [1.098933, 0.367621, 0.144932]^T$ , with error  $\|\mathbf{f}(\mathbf{x}_n)\| < 10^{-7}$ . However, the iteration may not converge from other possible initial guesses.

By Aitken's method, the iteration  $x_{n+1} = g(x_n)$  can be accelerated based on two consecutive points  $x_0$  and  $x_1$ , as shown in the figure below.

The secant line of  $g(x)$  that goes through the two points  $P_0 = (x_0, g(x_0) = x_1)$  and  $P_1 = (x_1, g(x_1) = x_2)$  is represented by the equation in terms of its slope:

$$\frac{y - g(x_0)}{x - x_0} = \frac{g(x_1) - g(x_0)}{x_1 - x_0} \quad (1.59)$$

Solving for  $y$ , we get

$$y = g(x_0) + (x - x_0) \frac{g(x_1) - g(x_0)}{x_1 - x_0} = x_1 + (x - x_0) \frac{x_2 - x_1}{x_1 - x_0} \quad (1.60)$$

To accelerate, instead of moving from  $x_0$  to  $x_1$ , we move to the point  $x$  at which this secant line intersects with the identity function  $y = x$ . We can therefore replace  $y$  in the equation above by  $x$  and solve the resulting equation

$$x = x_1 + (x - x_0) \frac{x_2 - x_1}{x_1 - x_0} \quad (1.61)$$

to get

$$x = \frac{x_1(x_1 - x_0) - x_0(x_2 - x_1)}{(x_1 - x_0) - (x_2 - x_1)} = x_0 - \frac{(x_1 - x_0)^2}{x_2 - 2x_1 + x_0} = x_0 - \frac{(\Delta x_0)^2}{\Delta^2 x_0} \quad (1.62)$$

where  $\Delta x_0$  and  $\Delta^2 x_0$  are respectively the first and second order differences defined below:

$$\Delta x_0 = x_1 - x_0, \quad \Delta x_1 = x_2 - x_1 \quad (1.63)$$

$$\Delta^2 x_0 = \Delta x_1 - \Delta x_0 = (x_2 - x_1) - (x_1 - x_0) = x_2 - 2x_1 + x_0 \quad (1.64)$$

This result can then be converted into an iterative process

$$\begin{cases} x_{n+1} = g(x_n) \\ x_{n+2} = g(x_{n+1}) \\ x_{n+3} = x_n - (x_{n+1} - x_n)^2 / (x_{n+2} - 2x_{n+1} + x_n) \end{cases} \quad (1.65)$$

Given  $x_n$ , we skip  $x_{n+1} = g(x_n)$  and  $x_{n+2} = g(x_{n+1})$  but directly move to  $x_{n+3}$  computed based on  $x_{n+1}$  and  $x_{n+2}$ , thereby making a greater step towards the solution.

#### Example

Solve  $x^3 - 3x + 1 = 0$ . Construct  $g(x) = (3x - 1)^{1/3}$ . It takes 18 iterations for the regular fixed point algorithm with initial guess  $x_0 = 1$ , to get  $x_{18} = 1.5320887$  that satisfies  $|f(x_n)| < 10^{-6}$ , but it only three iterations for Aitken's method to converge to the same result:

$n$	$x_n$	$f(x_n)$
0	1.0000	-1.0000
1	1.259921	-7.797631e - 01
2	1.406056	-4.384047e - 01
3	1.476396	-2.110206e - 01
4	1.507985	-9.476760e - 02
5	1.521751	-4.129594e - 02
6	1.527672	1.776362e - 02
7	1.530205	-7.598914e - 03
8	1.531286	3.242990e - 03
9	1.531747	-1.382619e - 03
10	1.531943	-5.892137e - 04
11	1.532027	-2.510521e - 04
12	1.532062	-1.069599e - 04
13	1.532078	-4.556840e - 05
14	1.532084	-1.941335e - 05
15	1.532087	-8.270551e - 06
16	1.532088	-3.523444e - 06
17	1.532089	-1.501066e - 06
18	1.532089	-6.394874e - 07

$n$	$x_n$	$f(x_n)$
0	1.0000	-1.0000
1	1.5937361	2.668730e - 01
2	1.5323992	1.254749e - 03
3	1.5320889	3.421160e - 08

## 1.4 Newton-Raphson Method (Univariate)

To solve equation  $f(x) = 0$ , we first consider the Taylor series expansion of  $f(x)$  at any point  $x_0$ :

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!}f''(x_0)(x - x_0)^2 + \cdots + \frac{1}{n!}f^{(n)}(x_0)(x - x_0)^n + \cdots \quad (1.66)$$

If  $f(x)$  is linear, i.e., its slope  $f'(x)$  is a constant for any  $x$ , then the second and higher order terms are all zero, and the equation  $f(x) = 0$  becomes

$$f(x) = f(x_0) + f'(x)(x - x_0) = 0 \quad (1.67)$$

Solving this equation we get the root  $x^*$  at which  $f(x^*) = 0$ :

$$x^* = x = x_0 - \frac{f(x_0)}{f'(x_0)} = x_0 + \Delta x_0 \quad (1.68)$$



where  $\Delta x_0 = -f(x_0)/f'(x_0)$  is the step we need to take to go from the initial point  $x_0$  to the root  $x^*$ :

$$\Delta x_0 = -\frac{f(x_0)}{f'(x_0)} \begin{cases} < 0 & \text{if } f(x_0) \text{ and } f'(x_0) \text{ are of the same sign} \\ > 0 & \text{if } f(x_0) \text{ and } f'(x_0) \text{ are of different signs} \end{cases} \quad (1.69)$$

If  $f(x)$  is nonlinear, the sum of the first two terms of the Taylor expansion is only an approximation of  $f(x)$ , and the resulting  $x$  found above can be treated as an approximation of the root, which can be improved iteratively to move from the initial point  $x_0$  towards the root, as illustrated in the figure below:

$$x_{n+1} = x_n + \Delta x_n = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \dots \quad (1.70)$$

The Newton-Raphson method can be considered as the fixed point iteration  $x_{n+1} = g(x_n)$  based on

$$g(x) = x - f(x)/f'(x) \quad (1.71)$$

The root  $x^*$  at which  $f(x^*) = 0$  is also the fixed point of  $g(x)$ , i.e.,  $g(x^*) = x^*$ . For the iteration to converge,  $g(x)$  needs to be a contraction with  $|g'(x)| < 1$ . Consider

$$g'(x) = \left( x - \frac{f(x)}{f'(x)} \right)' = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2} \quad (1.72)$$

We make the following comments:

- At the root  $x = x^*$  where  $f(x^*) = 0$ , if  $f'(x^*) \neq 0$ , then  $g'(x^*) = 0 < 1$ , i.e.,  $g(x)$  is a contraction and the iteration  $x_{n+1} = g(x_n)$  converges quadratically when  $x_n$  is close to  $x^*$ .
- If  $f'(x_n) = 0$  (the tangent is horizontal), the iteration cannot proceed, but we can modify  $x_n$  by adding a small value  $\epsilon$  to  $x_n$  so that  $f'(x + \epsilon) \neq 0$ .
- It is difficult to know the number of roots of a nonlinear equation (unlike the case of a linear equation), which can vary from zero (e.g.,  $f(x) = x^2 + 1$ ) to infinity (e.g.,  $f(x) = \cos(x)$ ). One can try different initial guesses in the range of interest to see if different roots can be found.
- Sometime a parameter  $\delta$  can be used to control the step size of the iteration:

$$x_{n+1} = x_n - \delta \frac{f(x_n)}{f'(x_n)} \quad (1.73)$$

- If  $\delta < 1$ , the iteration is de-accelerated. Although the convergence becomes slower, this may be desirable if the function  $f(x)$  is not smooth with many local variations.
- If  $\delta > 1$ , the iteration is accelerated. The convergence may or may not be accelerated. Due to the greater step size, the root may be skipped and missed. Sometimes the convergence may become significantly slowed or even oscillate around the true root, such as the example shown in the figure below with  $\delta = 2$ .

The order of convergence of the Newton-Raphson iteration can be found based on the Taylor expansion of  $f(x)$  at the neighborhood of the root  $x^* = x_n + e_n$ :

$$0 = f(x^*) = f(x_n) + f'(x_n)e_n + \frac{f''(x_n)}{2}e_n^2 + O(e_n^3) \quad (1.74)$$

where  $e_n = x^* - x_n$  is the error at the  $n$ th step. Substituting the Newton-Raphson's iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad \text{i.e.} \quad f(x_n) = f'(x_n)(x_n - x_{n+1}) \quad (1.75)$$

into the equation above, we get

$$\begin{aligned} 0 &= f'(x_n)(x_n - x_{n+1}) + f'(x_n)(x^* - x_n) + \frac{f''(x_n)}{2}e_n^2 + O(e_n^3) \\ &= f'(x_n)(x^* - x_{n+1}) + \frac{f''(x_n)}{2}e_n^2 + O(e_n^3) \\ &= f'(x_n)e_{n+1} + \frac{f''(x_n)}{2}e_n^2 + O(e_n^3) \end{aligned} \quad (1.76)$$

i.e.

$$e_{n+1} = -\frac{f''(x_n)}{2f'(x_n)}e_n^2 + O(e_n^3) \quad (1.77)$$

When  $n \rightarrow \infty$  all the higher order terms disappear, and the above can be written as

$$\lim_{n \rightarrow \infty} \frac{|e_{n+1}|}{|e_n|^2} \leq \frac{|f''(x^*)|}{2|f'(x^*)|} = \mu \quad (1.78)$$

Alternatively, we can get the Taylor expansion in terms of  $g(x)$ :

$$x_{n+1} = x^* - e_{n+1} = g(x_n) = g(x^* - e_n) = g(x^*) - g'(x^*)e_n + \frac{g''(x^*)}{2}e_n^2 + O(e_n^3) \quad (1.79)$$

Subtracting  $g(x^*) = x^*$  from both sides we get:

$$e_{n+1} = g'(x^*)e_n - \frac{g''(x^*)}{2}e_n^2 + O(e_n^3) \quad (1.80)$$

Now we find  $g'(x)$  and  $g''(x)$ :

$$g'(x) = \left( x - \frac{f(x)}{f'(x)} \right)' = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2} \quad (1.81)$$

and

$$\begin{aligned} g''(x) &= \left( \frac{f(x)f''(x)}{(f'(x))^2} \right)' = \frac{(f'(x)f''(x) + f(x)f'''(x))(f'(x))^2 - f(x)f''(x)2f'(x)f''(x)}{(f'(x))^4} \\ &= \frac{(f'(x))^3 f''(x)}{(f'(x))^4} = \frac{f''(x)}{f'(x)} \end{aligned} \quad (1.82)$$

Evaluating these at  $x = x^*$  at which  $f(x^*) = 0$ , and substituting them back

into the expression for  $e_{n+1}$  above, the linear term is zero as  $g'(x^*) = 0$ , i.e., the convergence is quadratic, and we get the same result:

$$\lim_{n \rightarrow \infty} \frac{|e_{n+1}|}{|e_n|^2} \leq \frac{|f''(x^*)|}{2|f'(x^*)|} = \frac{|g''(x^*)|}{2} \quad (1.83)$$

We see that, if  $f'(x^*) \neq 0$ , then the order of convergence of the Newton-Raphson method is  $p = 2$ , and the rate of convergence is  $\mu = |f''(x^*)|/2|f'(x^*)|$ . However, if  $f'(x^*) = 0$ , the convergence is linear rather than quadratic, as shown in the example below.

**Example:** Consider solving the equation

$$f(x) = x^3 - 4x^2 + 5x - 2 = (x-1)^2(x-2) = 0 \quad (1.84)$$

which has a repeated root  $x = 1$  as well as a single root  $x = 2$ . We have

$$f'(x) = 3x^2 - 8x + 5 = (3x-5)(x-1), \quad f''(x) = 6x - 8 \quad (1.85)$$

Note that at the root  $x^* = 1$  we have  $f'(x^*) = f'(1) = 0$ . We further find:

$$g(x) = x - \frac{f(x)}{f'(x)} = x - \frac{(x-1)^2(x-2)}{(3x-5)(x-1)} = x - \frac{(x-1)(x-2)}{3x-5} \quad (1.86)$$

and

$$\begin{aligned} g'(x) &= \frac{f(x)f''(x)}{(f'(x))^2} = \frac{(x-1)^2(x-2)(6x-8)}{(3x-5)^2(x-1)^2} \\ &= \frac{(x-2)(6x-8)}{(3x-5)^2} = \begin{cases} 1/2 & x = 1 \\ 0 & x = 2 \end{cases} \end{aligned}$$

We therefore have

$$e_{n+1} = g'(x)e_n - \frac{1}{2}g''(x)e_n^2 + O(e_n^3) \xrightarrow{n \rightarrow \infty} \begin{cases} e_n/2 & x = 1 \\ -g''(x)e_n^2/2 & x = 2 \end{cases} \quad (1.87)$$

We see the iteration converges quadratically to the single root  $x = 2$ , but only linearly to the repeated root  $x = 1$ .

We consider in general a function with a repeated root at  $x = a$  of multiplicity  $k$ :

$$f(x) = (x-a)^k h(x) \quad (1.88)$$

its derivative is

$$f'(x) = k(x-a)^{k-1}h(x) + (x-a)^k h'(x) = (x-a)^{k-1}[kh(x) + (x-a)h'(x)] \quad (1.89)$$

As  $f'(a) = 0$ , the convergence of the Newton-Raphson method to  $x = a$  is linear, rather than quadratic.

In such case, we can accelerate the iteration by using a step size  $\delta = k > 1$ :

$$g(x) = x - k \frac{f(x)}{f'(x)} \quad (1.90)$$

and

$$g'(x) = 1 - k \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = 1 - k + k \frac{f(x)f''(x)}{(f'(x))^2} \quad (1.91)$$

Now we show that this  $g'(x)$  is zero at the repeated root  $x = a$ , therefore the convergence to this root is still quadratic.

We substitute  $f(x) = (x - a)^k h(x)$ ,  $f'(x) = k(x - a)^{k-1}[kh + (x - a)h']$ , and

$$\begin{aligned} f''(x) &= [(x - a)^{k-1}[kh + (x - a)h']]' \\ &= 1 - k + k(k - 1)(x - a)^{k-2}[kh + (x - a)h'] \\ &\quad + (x - a)^{k-1}[(k + 1)h' + (x - a)h''] \end{aligned} \quad (1.92)$$

into the expression for  $g'(x)$  above to get (after some algebra):

$$\begin{aligned} g'(x) &= 1 - k + k \frac{f(x)f''(x)}{(f'(x))^2} \\ &= 1 - k + k \frac{h(k - 1)(kh + (x - a)h') + h(x - a)((k + 1)h' + (x - a)h'')}{(kh + (x - a)h')^2} \end{aligned} \quad (1.93)$$

At  $x = a$ , we get

$$g'(x) \Big|_{x=a} = 1 - k + k \frac{(k - 1)kh^2}{(kh)^2} = 0 \quad (1.94)$$

i.e., the convergence to the repeated root at  $x = a$  is no longer linear but quadratic.

The difficulty, however, is that the multiplicity  $k$  of a root is unknown ahead of time. If  $\delta > 1$  is used blindly some root may be skipped, and the iteration may oscillate around the real root.

**Example:** Consider solving  $f(x) = x^3 - 4x^2 + 5x - 2 = (x - 1)^2(x - 2) = 0$ , with a double root  $x = 1$  and a single root  $x = 2$ . In the following, we compare the performance of both  $g_1(x) = x - f(x)/f'(x)$  and  $g_2(x) = x - 2f(x)/f'(x)$ .

- First use an initial guess  $x_0 = 0.3$ .
  - When  $g_1(x)$  is used, the convergence is linear around the double root  $x = 1$ . It takes 16 iterations to get  $x_{16} = 0.999983$  with  $|e_{16}| < 10^{-9}$ :

$n$	$x_n$	$f(x_n)$	$f'(x_n)$
0	0.300000	-0.833000	2.870000
1	0.590244	-0.236698	1.323212
2	0.769125	-0.065609	0.621659
3	0.874665	-0.017678	0.297797
4	0.934027	-0.004640	0.145004
5	0.966023	-0.001194	0.071417
6	0.982737	-0.000303	0.035420
7	0.991296	-0.000076	0.017636
8	0.995629	-0.000019	0.008799
9	0.997810	-0.000005	0.004395
10	0.998904	-0.000001	0.002196
11	0.999452	-0.000000	0.001098
12	0.999726	-0.000000	0.000549
13	0.999863	-0.000000	0.000274
14	0.999931	-0.000000	0.000137
15	0.999966	-0.000000	0.000069
16	0.999983	-0.000000	

- When  $g_2(x)$  is used, the convergence is quadratic around the double root  $x = 1$ . It takes only 3 iterations to get  $x_3 = 0.999982$  with  $|e_3| < 10^{-9}$ :

$n$	$x_n$	$f(x_n)$	$f'(x_n)$
0	0.300000	-0.833000	2.870000
1	0.880488	-0.015990	0.281874
2	0.993944	-0.000037	0.012222
3	0.999982	-0.000000	

- Next use a different initial guess  $x_0 = 4$ .
  - If  $g_1$  is used, it takes 7 iterations to get  $x_7 = 2.0$  with  $|e_7| < 10^{-9}$ , the convergence is quadratic.

$n$	$x_n$	$f(x_n)$	$f'(x_n)$
1	$x = 3.00000$	4.00000	8.00000
2	$x = 2.50000$	1.12500	3.75000
3	$x = 2.20000$	0.28800	1.92000
4	$x = 2.05000$	0.05513	1.20750
5	$x = 2.00435$	0.00439	1.01745
6	$x = 2.00004$	0.00004	1.00015
7	$x = 2.00000$	0.00000	1.00000
8	$x = 2.00000$	0.00000	

- If  $g_2$  is used, oscillation happens as shown in the figure below. However, if a better initial guess  $x_0 = 3.5$  is used instead of  $x_0 = 4$ , it takes only one step to get  $x_1 = 2.0$  with  $|e_1| < 10^{-9}$ , the convergence is significantly accelerated.

$n$	$x_n$	$f(x_n)$	$f'(x_n)$
1	3.00000	4.00000	8.00000
2	2.00000	0.00000	

## 1.5 Newton-Raphson Method (Multivariate)

The Newton-Raphson method discussed above for solving a single-variable equation  $f(x) = 0$  can be generalized to the case of multivariate equation systems containing  $M$  equations of  $N$  variables in  $\mathbf{x} = [x_1, \dots, x_N]^T$ :

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_M(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} f_1(x_1, \dots, x_N) \\ \vdots \\ f_M(x_1, \dots, x_N) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} = \mathbf{0} \quad (1.95)$$

To solve the equation system, we first consider the Taylor series expansion of each of the  $M$  functions in the neighborhood of the initial point  $\mathbf{x}_0 = [x_{01}, \dots, x_{0N}]^T$ :

$$f_m(\mathbf{x}) = f_m(\mathbf{x}_0) + \sum_{n=1}^N \frac{\partial f_m(\mathbf{x}_0)}{\partial x_n} (x_n - x_{0n}) + r_m(\|\mathbf{x} - \mathbf{x}_0\|^2), \quad (m = 1, \dots, M) \quad (1.96)$$

where  $r_m(\|\mathbf{x} - \mathbf{x}_0\|^2)$  represents the second and higher order terms in the series beyond the linear term, which can be neglected if  $\|\mathbf{x} - \mathbf{x}_0\|$  is small. These  $M$  equations can be expressed in matrix form

$$\begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_M(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x}_0) \\ \vdots \\ f_M(\mathbf{x}_0) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1(\mathbf{x}_0)}{\partial x_1} & \dots & \frac{\partial f_1(\mathbf{x}_0)}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_M(\mathbf{x}_0)}{\partial x_1} & \dots & \frac{\partial f_M(\mathbf{x}_0)}{\partial x_N} \end{bmatrix} \begin{bmatrix} x_1 - x_{01} \\ \vdots \\ x_N - x_{0N} \end{bmatrix} + \begin{bmatrix} r_1 \\ \vdots \\ r_M \end{bmatrix} \quad (1.97)$$

or more concisely

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}_0) + \mathbf{J}(\mathbf{x}_0) (\mathbf{x} - \mathbf{x}_0) + \mathbf{r} \approx \mathbf{f}(\mathbf{x}_0) + \mathbf{J}(\mathbf{x}_0) (\mathbf{x} - \mathbf{x}_0) = \mathbf{f}_0 + \mathbf{J}_0 \Delta \mathbf{x} \quad (1.98)$$

where  $\Delta \mathbf{x} = \mathbf{x} - \mathbf{x}_0$ , and  $\mathbf{f}_0 = \mathbf{f}(\mathbf{x}_0)$  and  $\mathbf{J}_0 = \mathbf{J}(\mathbf{x}_0)$  are respectively the function  $\mathbf{f}(\mathbf{x})$  and its *Jacobian matrix*  $\mathbf{J}_f(\mathbf{x})$  both evaluated at  $\mathbf{x}_0$ . We further consider solving the equation system  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  in the following two cases:

- $M = N$ : The number of equations is the same as the number of unknowns, the Jacobian  $\mathbf{J}(\mathbf{x})$  is a square matrix and its inverse  $\mathbf{J}^{-1}$  exists in general. In the special case where  $\mathbf{f}(\mathbf{x})$  is linear, the Taylor series contains only the first two terms while all higher order terms are zero, and the approximation

in Eq. (1.98) becomes exact. To find the root  $\mathbf{x}^*$  satisfying  $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$ , we set  $\mathbf{f}(\mathbf{x})$  in Eq. (1.98) to zero and solve the resulting equation for  $\mathbf{x}$  to get

$$\mathbf{x}^* = \mathbf{x} = \mathbf{x}_0 - \mathbf{J}_0^{-1} \mathbf{f}_0 = \mathbf{x}_0 - \mathbf{A}^{-1} \mathbf{f}_0 \quad (1.99)$$

As in general  $\mathbf{f}(\mathbf{x})$  is nonlinear, it can only be approximated by the first two terms of the Taylor series, consequently the result above is only an approximation of the optimal solution. But this approximation can be improved iteratively to approach the optimal solution  $\mathbf{x}^*$ :

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta \mathbf{x}_n = \mathbf{x}_n - \mathbf{J}_n^{-1} \mathbf{f}_n \quad (1.100)$$

where we have defined  $\Delta \mathbf{x}_n = -\mathbf{J}_n^{-1} \mathbf{f}_n$  as the increment, which can also be denoted by  $\mathbf{d}_n = \Delta \mathbf{x}_n$  to represent the *search* or *Newton direction*. The iteration moves  $\mathbf{x}_n$  in the N-D space spanned by  $\{x_1, \dots, x_N\}$  from some initial guess  $\mathbf{x}_0$  along such a path that all function values  $f_m(\mathbf{x})$ ,  $m = 1, \dots, M$  are reduced. Same as in the univariate case, a scaling factor  $\delta_n$  can be used to control the step size of the iteration

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \delta_n \Delta \mathbf{x}_n = \mathbf{x}_n - \delta_n \mathbf{J}_n^{-1} \mathbf{f}_n \quad (1.101)$$

When  $\delta < 1$ , the step size becomes smaller and the convergence of the iteration is slower, however, we will have a better chance not to skip a solution, which may happen if  $\mathbf{f}(\mathbf{x})$  is not smooth and the step size is too big.

The algorithm is listed below:

- Select  $\mathbf{x}_0$
- Obtain  $\mathbf{f}_0 = \mathbf{f}(\mathbf{x}_0)$  and  $\mathbf{J}_0$
- Obtain  $\mathbf{J}_0^{-1}$
- $n = 0$
- While  $\|\mathbf{x}_{n+1} - \mathbf{x}_n\| > tol$  do
  - o  $\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}_n^{-1} \mathbf{f}_n$
  - o Find  $\mathbf{f}_{n+1}$  and  $\mathbf{J}_{n+1}$
  - o  $n = n + 1$
- $M > N$ : There are more equations than unknowns, i.e., equation  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  is an over-constrained system, and the Jacobian  $\mathbf{J}(\mathbf{x})$  is an  $M \times N$  non-square matrix without an inverse, i.e., no solution exists for the equation  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  in general. But we can still seek to find an optimal solution  $\mathbf{x}^*$  that minimizes the following sum-of-squares error:

$$\varepsilon(\mathbf{x}) = \frac{1}{2} \|\mathbf{f}(\mathbf{x})\|^2 = \frac{1}{2} \mathbf{f}(\mathbf{x})^T \mathbf{f}(\mathbf{x}) = \frac{1}{2} \sum_{m=1}^M f_m^2(\mathbf{x}) \quad (1.102)$$

The gradient vector of  $\varepsilon(\mathbf{x})$  is:

$$\mathbf{g}_\varepsilon(\mathbf{x}) = \frac{d}{d\mathbf{x}} \varepsilon(\mathbf{x}) = \frac{d}{d\mathbf{x}} \left( \frac{1}{2} \sum_{m=1}^M f_m^2(\mathbf{x}) \right) = \sum_{m=1}^M \frac{d}{d\mathbf{x}} f_m(\mathbf{x}) f_m(\mathbf{x}) \quad (1.103)$$

The  $n$ th component of  $\mathbf{g}_\varepsilon(\mathbf{x})$  is

$$\frac{\partial \varepsilon(\mathbf{x})}{\partial x_n} = \sum_{m=1}^M \frac{\partial f_m(\mathbf{x})}{\partial x_n} f_m(\mathbf{x}) = \sum_{m=1}^M J_{mn} f_m(\mathbf{x}) \quad (n = 1, \dots, N) \quad (1.104)$$

where  $J_{mn} = \partial f_m(\mathbf{x}) / \partial x_n$  is the component in the  $m$ th row and  $n$ th column of the Jacobian matrix  $\mathbf{J}_f(\mathbf{x})$  of  $\mathbf{f}(\mathbf{x})$ . Now the gradient can be written as

$$\mathbf{g}_\varepsilon(\mathbf{x}) = \mathbf{J}_f^T(\mathbf{x}) \mathbf{f}(\mathbf{x}) \quad (1.105)$$

If  $\mathbf{f}(\mathbf{x})$  is linear and can therefore be represented as the sum of the first two terms of its Taylor series in Eq. (1.98), then the gradient is:

$$\mathbf{g}_\varepsilon(\mathbf{x}) = \mathbf{J}^T(\mathbf{x}) \mathbf{f}(\mathbf{x}) = \mathbf{J}^T(\mathbf{x}_0) [\mathbf{f}(\mathbf{x}_0) + \mathbf{J}(\mathbf{x}_0) \Delta \mathbf{x}] = \mathbf{J}_0^T (\mathbf{f}_0 + \mathbf{J}_0 \Delta \mathbf{x}) \quad (1.106)$$

where  $\mathbf{x}_0$  is any chosen initial guess. If we assume  $\mathbf{x}$  is the optimal solution at which  $\varepsilon(\mathbf{x})$  is minimized and  $\mathbf{g}_\varepsilon(\mathbf{x})$  is zero:

$$\mathbf{g}_\varepsilon(\mathbf{x}) = \mathbf{J}_0^T (\mathbf{f}_0 + \mathbf{J}_0 \Delta \mathbf{x}) = \mathbf{0} \quad (1.107)$$

then the increment  $\Delta \mathbf{x}$  can be found by solving the equation

$$\Delta \mathbf{x} = -(\mathbf{J}_0^T \mathbf{J}_0)^{-1} \mathbf{J}_0^T \mathbf{f}_0 = -\mathbf{J}_0^- \mathbf{f}_0 \quad (1.108)$$

Here  $\mathbf{J}_0^- = (\mathbf{J}_0^T \mathbf{J}_0)^{-1} \mathbf{J}_0^T$  is the *pseudo-inverse* (Section A.15) of the non-square matrix  $\mathbf{J}_0$ . Now the optimal solution can be found as:

$$\mathbf{x}^* = \mathbf{x}_0 + \Delta \mathbf{x} = \mathbf{x}_0 - \mathbf{J}_0^- \mathbf{f}_0 \quad (1.109)$$

However, as  $\mathbf{f}(\mathbf{x})$  is nonlinear in general, the sum of the first two terms of its Taylor series is only an approximation. Consequently the result  $\mathbf{x} = \mathbf{x}_0 + \Delta \mathbf{x}$  above is not the optimal solution, but an estimate which can be improved by carrying out this step iteratively:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta \mathbf{x}_n = \mathbf{x}_n - \mathbf{J}_n^- \mathbf{f}_n \quad (1.110)$$

This iteration will converge to  $\mathbf{x}^*$  at which  $\mathbf{g}_\varepsilon(\mathbf{x}^*) = \mathbf{0}$ , and the squared error  $\varepsilon(\mathbf{x})$  is minimized.

Specially, for a linear equation system  $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$ , the Jacobian is simply  $\mathbf{J}_f(\mathbf{x}_n) = \mathbf{A}$ , and the optimal solution is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{A}^- \mathbf{f}_n = \mathbf{x}_n - (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T (\mathbf{A}\mathbf{x}_n - \mathbf{b}) = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} = \mathbf{A}^- \mathbf{b} \quad (1.111)$$

i.e., the optimal solution can be found from any initial guess in a single step. This result is the same as that in Eq. (1.7).

Comparing Eqs. (1.100) and (1.110), we see that the two algorithms are essentially the same, with the only difference that the regular inverse  $\mathbf{J}^{-1}$  is used when  $M = N$ , but the pseudoinverse  $\mathbf{J}^-$  is used when  $M > N$  and  $\mathbf{J}^{-1}$  does not exist.

The Newton-Raphson method assumes the availability of the analytical expressions of all partial derivatives  $J_{mn} = \partial f_m(\mathbf{x}) / \partial x_n$  ( $m = 1, \dots, M$ ,  $n =$



$1, \dots, N$ ) in the Jacobian matrix  $\mathbf{J}$ . However, when this is not the case,  $J_{mn}$  need to be approximated by forward or central difference (secant) method:

$$\begin{aligned} J_{mn} &= \frac{\partial f_m(x_1, \dots, x_N)}{\partial x_n} \approx \frac{f_m(x_1, \dots, x_n + h, \dots, x_N) - f_m(x_1, \dots, x_n, \dots, x_N)}{h} \\ &\approx \frac{f_m(x_1, \dots, x_n + h, \dots, x_N) - f_m(x_1, \dots, x_n - h, \dots, x_N)}{2h} \end{aligned} \quad (1.112)$$

where  $h$  is a small increment.

### Example 1

$$\begin{cases} 3x_1 - \cos(x_2x_3) - 3/2 = 0 \\ 4x_1^2 - 625x_2^2 + 2x_3 - 1 = 0 \\ 20x_3 + e^{-x_1x_2} + 9 = 0 \end{cases}$$

$$\mathbf{J} = \begin{bmatrix} 3 & x_3 \sin(x_2x_3) & x_2 \sin(x_2x_3) \\ 8x_1 & -1250x_2 & 2 \\ -x_2 e^{-x_1x_2} & -x_1 e^{-x_1x_2} & 20 \end{bmatrix}$$

$n$	$\mathbf{x}$	$error$
0	(1.000000, 1.000000, 1.000000)	$6.207e + 02$
1	(1.232701, 0.503132, -0.473253)	$1.541e + 02$
2	(0.832592, 0.251806, -0.490636)	$3.884e + 01$
3	(0.833238, 0.128406, -0.494702)	$9.517e + 00$
4	(0.833275, 0.069082, -0.497147)	$2.200e + 00$
5	(0.833281, 0.043585, -0.498206)	$4.063e - 01$
6	(0.833282, 0.036117, -0.498517)	$3.486e - 02$
7	(0.833282, 0.035343, -0.498549)	$3.741e - 04$
8	(0.833282, 0.035335, -0.498549)	$4.498e - 08$
9	(0.833282, 0.035335, -0.498549)	$5.551e - 16$

### Example 2

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \quad \begin{cases} f_1(\mathbf{x}) = x_1^2 - 2x_1 + x_2^2 - x_3 + 1 = 0 \\ f_2(\mathbf{x}) = x_1x_2^2 - x_1 - 3x_2 + x_2x_3 + 2 = 0 \\ f_3(\mathbf{x}) = x_1x_3^2 - 3x_3 + x_2x_3^2 + x_1x_2 = 0 \end{cases}$$

$$\mathbf{J} = \begin{bmatrix} 2x_1 - 2 & 2x_2 & -1 \\ x_2^2 - 1 & 2x_1x_2 - 3 + x_3 & x_2 \\ x_3^2 + x_2 & x_3^2 + x_1 & 2x_1x_3 - 3 + 2x_2x_3 \end{bmatrix}$$

With  $\mathbf{x}_0 = [1, 2, 3]^T$ , we get a root:

$n$	$\mathbf{x}$	$error$
0	(1.00000, 2.00000, 3.00000)	$2.064e + 01$
1	(0.10256, 1.64103, 2.56410)	$4.303e + 00$
2	(1.52062, 1.41113, 0.19859)	$2.689e + 00$
3	(1.94123, 0.77134, 0.89465)	$1.217e + 00$
4	(1.06737, 1.19117, 0.48353)	$1.144e + 00$
5	(1.26825, 0.95182, 0.88028)	$3.323e - 01$
6	(0.95899, 1.03384, 0.96813)	$1.171e - 01$
7	(1.00171, 1.00007, 0.99718)	$4.162e - 03$
8	(1.00000, 1.00000, 1.00000)	$6.701e - 06$

With  $\mathbf{x}_0 = [0, 0, 0]^T$ , we get another root:

$n$	$\mathbf{x}$	$error$
0	(0.00000, 0.00000, 0.00000)	$2.236e + 00$
1	(0.50000, 0.50000, 0.00000)	$5.728e - 01$
2	(0.83951, 0.47531, 0.13580)	$1.175e - 01$
3	(0.98582, 0.41849, 0.15069)	$2.639e - 02$
4	(1.05417, 0.38715, 0.14717)	$6.088e - 03$
5	(1.08565, 0.37339, 0.14558)	$1.264e - 03$
6	(1.09693, 0.36849, 0.14503)	$1.618e - 04$
7	(1.09888, 0.36764, 0.14494)	$4.817e - 06$

### Broyden's method

In the Newton-Raphson method, two main operations are carried out in each iteration: (a) evaluate the Jacobian matrix  $\mathbf{J}_{\mathbf{f}}(\mathbf{x}_n)$  and (b) obtain its inverse  $\mathbf{J}_{\mathbf{f}}^{-1}(\mathbf{x}_n)$ . To avoid these expensive computation for these operations, we can consider using Broyden's method, one of the *quasi-Newton methods*, which approximates the inverse of the Jacobian  $\mathbf{J}_{n+1}^{-1} = \mathbf{J}^{-1}(\mathbf{x}_{n+1})$  from the  $\mathbf{J}_n^{-1} = \mathbf{J}^{-1}(\mathbf{x}_n)$  in the previous iteration step, so that it can be updated iteratively from the initial  $\mathbf{J}_0^{-1} = \mathbf{J}^{-1}(\mathbf{x}_0)$ .

We first consider in the single-variable case how to estimate the next derivative  $f'_{n+1} = f'(x_{n+1})$  from the current  $f'_n = f'(x_n)$  by the secant method:

$$\begin{aligned} f'_{n+1} &\approx \hat{f}'_{n+1} = \frac{f_{n+1} - f_n}{x_{n+1} - x_n} = \frac{\delta f_n}{\delta x_n} = \frac{\hat{f}'_n \delta x_n - \hat{f}'_n \delta x_n + \delta f_n}{\delta x_n} \\ &= \hat{f}'_n + \frac{\delta f_n - \hat{f}'_n \delta x_n}{\delta x_n} = \hat{f}'_n + \delta f'_n \end{aligned} \quad (1.113)$$

where

- $\delta f_n = f_{n+1} - f_n$  is the true increment of the function over the interval  $\delta x_n = x_{n+1} - x_n$ ;

- $\hat{f}'_n \delta x_n$  is the estimated increment of the function based on the previous derivative  $\hat{f}'_n$ ;
- $\hat{\delta f}'_n$  is the estimated increment of the derivative:

$$\hat{\delta f}'_n = \frac{\delta f_n - \hat{f}'_n \delta x_n}{\delta x_n} \quad (1.114)$$

The equation above indicates that the derivative  $f'_{n+1}$  in the  $(n+1)$ th step can be estimated by adding the estimated increment  $\hat{\delta f}'_n$  to the derivative  $\hat{f}'_n$  in the current  $n$ th step.

Having obtained  $\hat{f}'_{n+1}$ , we can use the same iteration in the Newton-Raphson method to find  $x_{n+1}$ :

$$x_{n+1} = x_n - \frac{f(x_n)}{\hat{f}'_n} \quad (1.115)$$

This method for single-variable case can be generalized to multiple variable case for solving  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ . Following the way we estimate the increment of the derivative of a single-variable function in Eq. (1.114), here we can estimate the increment of the Jacobian of a multi-variable function:

$$\delta \hat{\mathbf{J}}_n = \frac{(\delta \mathbf{f}_n - \hat{\mathbf{J}}_n \delta \mathbf{x}_n) \delta \mathbf{x}_n^T}{\delta \mathbf{x}_n^T \delta \mathbf{x}_n} = \frac{(\delta \mathbf{f}_n - \hat{\mathbf{J}}_n \delta \mathbf{x}_n) \delta \mathbf{x}_n^T}{\|\delta \mathbf{x}_n\|^2} \quad (1.116)$$

where  $\delta \mathbf{x}_n = \mathbf{x}_{n+1} - \mathbf{x}_n$  and  $\delta \mathbf{f}_n = \mathbf{f}_{n+1} - \mathbf{f}_n = \mathbf{f}(\mathbf{x}_{n+1}) - \mathbf{f}(\mathbf{x}_n)$ . Now in each iteration, we can update the estimated Jacobian as well as the estimated root:

$$\hat{\mathbf{J}}_{n+1} = \hat{\mathbf{J}}_n + \delta \hat{\mathbf{J}}_n, \quad \mathbf{x}_{n+1} = \mathbf{x}_n + \delta \mathbf{x}_n = \mathbf{x}_n - \hat{\mathbf{J}}_n^{-1} \mathbf{f}(\mathbf{x}_n) \quad (1.117)$$

The algorithm can be further improved so that the inverse of the Jacobian  $\mathbf{J}_n$  is avoided. Specifically, consider the inverse Jacobian:

$$\hat{\mathbf{J}}_{n+1}^{-1} = \left( \hat{\mathbf{J}}_n + \delta \hat{\mathbf{J}}_n \right)^{-1} = \left[ \hat{\mathbf{J}}_n + \frac{(\delta \mathbf{f}_n - \hat{\mathbf{J}}_n \delta \mathbf{x}_n) \delta \mathbf{x}_n^T}{\|\delta \mathbf{x}_n\|^2} \right]^{-1} \quad (1.118)$$

We can apply the Sherman-Morrison formula (Section A.6):

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1} \mathbf{u}\mathbf{v}^T \mathbf{A}^{-1}}{1 + \mathbf{v}^T \mathbf{A}^{-1} \mathbf{u}} \quad (1.119)$$

to the right-hand side of the equation above by defining

$$\mathbf{A} = \hat{\mathbf{J}}_n, \quad \mathbf{u} = \frac{\delta \mathbf{f}_n - \hat{\mathbf{J}}_n \delta \mathbf{x}_n}{\|\delta \mathbf{x}_n\|^2}, \quad \mathbf{v} = \delta \mathbf{x}_n \quad (1.120)$$

and rewrite it as:

$$\hat{\mathbf{J}}_{n+1}^{-1} = \hat{\mathbf{J}}_n^{-1} - \frac{\hat{\mathbf{J}}_n^{-1} (\delta \mathbf{f}_n - \hat{\mathbf{J}}_n \delta \mathbf{x}_n) / \|\delta \mathbf{x}_n\|^2 \delta \mathbf{x}_n^T \hat{\mathbf{J}}_n^{-1}}{1 + \delta \mathbf{x}_n^T \hat{\mathbf{J}}_n^{-1} (\delta \mathbf{f}_n - \hat{\mathbf{J}}_n \delta \mathbf{x}_n) / \|\delta \mathbf{x}_n\|^2} = \hat{\mathbf{J}}_n^{-1} - \frac{(\hat{\mathbf{J}}_n^{-1} \delta \mathbf{f}_n - \delta \mathbf{x}_n) \delta \mathbf{x}_n^T \hat{\mathbf{J}}_n^{-1}}{\delta \mathbf{x}_n^T \hat{\mathbf{J}}_n^{-1} \delta \mathbf{f}_n} \quad (1.121)$$

We see that the next  $\hat{\mathbf{J}}_{n+1}^{-1}$  can be iteratively estimated directly from the previous  $\hat{\mathbf{J}}_n^{-1}$ , thereby avoiding computing the inverse of  $\hat{\mathbf{J}}_n$  altogether. The algorithm is listed below:

- Select  $\mathbf{x}_0$
- Find  $\mathbf{f}_0 = \mathbf{f}(\mathbf{x}_0)$ ,  $\mathbf{J}_0 = \mathbf{J}(\mathbf{x}_0)$ , and  $\mathbf{J}_0^{-1}$
- $\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{J}_0^{-1}\mathbf{f}_0$
- $n = 0$
- While  $\|\mathbf{x}_{n+1} - \mathbf{x}_n\| > tol$  do
  - $\delta\mathbf{x}_n = \mathbf{x}_{n+1} - \mathbf{x}_n$
  - $\mathbf{f}_{n+1} = \mathbf{f}(\mathbf{x}_{n+1})$
  - $\delta\mathbf{f} = \mathbf{f}_{n+1} - \mathbf{f}_n$
  - $\mathbf{J}_n^{-1} = \mathbf{J}_n^{-1} - (\delta\mathbf{x}_n - \mathbf{J}_n^{-1}\delta\mathbf{f}_n)\delta\mathbf{x}_n^T\mathbf{J}_n^{-1}/(\delta\mathbf{x}_n^T\mathbf{J}_n^{-1}\delta\mathbf{f}_n)$
  - $\mathbf{x}_n = \mathbf{x}_{n+1}$ ,  $\mathbf{f}_n = \mathbf{f}_{n+1}$
  - $\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}_n^{-1}\mathbf{f}_n$
  - $n = n + 1$

## 2 Optimization

---

Machine learning problems can often be formulated as an optimization problem. In many algorithms for both regression and classification, the goal is to optimize an *objective* function of a set of model parameters so that the model fits the given dataset optimally in certain sense.

Consider, for example, the construction of a model parameterized by  $\boldsymbol{\theta} = [\theta_1, \dots, \theta_N]^T$  to fit a set of observed data points called the *training dataset* or simply *training set* denoted by  $\mathcal{D} = \{(\mathbf{x}_n, y_n), n = 1, \dots, N\}$ . While the form of the model is typically assumed to be known (e.g., a linear or quadratic function, or certain probability density function (pdf) of the data), the parameters in  $\boldsymbol{\theta}$  are unknown and to be estimated based on the training set  $\mathcal{D}$ , using methods such as the following widely used in machine learning:

- *Least squares estimation (LSE):*

The sum of squared error (SSE) between the model output and the desired output based on the given data is to be minimized:

$$J(\boldsymbol{\theta}) = \sum_{n=1}^N [y_n - f(\mathbf{x}_n, \boldsymbol{\theta})]^2 \quad (2.1)$$

- *Maximum likelihood estimate (MLE):*

The likelihood function of the parameters in  $\boldsymbol{\theta}$  is to be maximized:

$$J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^N p(y_n|\mathbf{x}_n, \boldsymbol{\theta}) \quad (2.2)$$

Solving these optimization problems we get the optimal model parameter  $\boldsymbol{\theta}^*$  for the model to best fit the given data.

To prepare for the discussions of these algorithms in the future chapters, here in this chapter we consider a set of optimization algorithms for optimizing a generic real multivariate function  $y = f(\mathbf{x}) = f(x_1, \dots, x_N)$  by searching its domain, the N-D space spanned by  $\{x_1, \dots, x_N\}$ , to find the optimal solution  $\mathbf{x}^* = [x_1^*, \dots, x_N^*]^T$  at which the function value  $f(\mathbf{x}^*)$  is either maximized or minimized. As maximizing  $f(\mathbf{x})$  is equivalent to minimizing  $-f(\mathbf{x})$ , we typically consider only minimization problems to find:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{arg\,min}} f(\mathbf{x}), \quad \text{i.e.,} \quad f(\mathbf{x}^*) = \min_{\mathbf{x}} f(\mathbf{x}) \quad (2.3)$$

If the independent variable  $\mathbf{x}$  is allowed to take any value inside the function domain  $\mathbb{R}^N$ , then the optimization problem is *unconstrained*, otherwise if  $\mathbf{x}$  must be inside a subset of  $\mathbb{R}^N$ , the problem is *constrained*. Both types of optimization problems occur in machine learning.

## 2.1 Unconstrained Optimization

### 2.1.1 Optimization by Solving Equations

Before discussing specific optimization algorithms, we first show that the problems of unconstrained minimization and equation solving are equivalent as they can be converted to each other.

- To minimize an objective function  $f(\mathbf{x})$ , we can set its gradient vector  $\mathbf{g}_f(\mathbf{x})$  to zero and solve the resulting equation system:

$$\mathbf{g}_f(\mathbf{x}) = \nabla_{\mathbf{x}} f(\mathbf{x}) = \frac{df(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_N \end{bmatrix} = \mathbf{0} \quad (2.4)$$

The solution  $\mathbf{x}^*$  of this equation system, called a *critical, stationary, or stable point* of  $f(\mathbf{x})$ , is also the solution to the optimization problem, if it is not a saddle point, that either maximizes or minimizes  $f(\mathbf{x})$ .

This equation system can be solved by any of the methods discussed in the previous chapter, such as the Newton-Raphson method, which finds the root of a general function  $\mathbf{f}(\mathbf{x})$  iteratively:  $\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}_f^{-1}(\mathbf{x}_n) \mathbf{f}(\mathbf{x}_n)$ . Here, specifically, to solve the equation  $\mathbf{g}_f(\mathbf{x}) = \mathbf{0}$ , we first get the Jacobian  $\mathbf{J}_g(\mathbf{x}) = \mathbf{H}_f(\mathbf{x})$  of the gradient  $\mathbf{g}_f(\mathbf{x})$  of  $f(\mathbf{x})$ , which is the Hessian of  $f(\mathbf{x})$ , and then carry out the iteration below to eventually find  $\mathbf{x}^*$  that minimizes  $f(\mathbf{x})$ :

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}_g^{-1}(\mathbf{x}_n) \mathbf{g}_f(\mathbf{x}_n) = \mathbf{x}_n - \mathbf{H}_f^{-1}(\mathbf{x}_n) \mathbf{g}_f(\mathbf{x}_n) \quad (2.5)$$

The second equality is due to the fact that the Jacobian of the gradient  $\mathbf{g}_f$  of function  $\mathbf{f}(\mathbf{x})$  is the Hessian of the function, i.e.,  $\mathbf{J}_g = \mathbf{H}_f$ .

- To solve an equation system  $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), \dots, f_N(\mathbf{x})]^T = \mathbf{0}$ , we can minimize an objective function defined as

$$J(\mathbf{x}) = \frac{1}{2} \mathbf{f}^T(\mathbf{x}) \mathbf{f}(\mathbf{x}) = \frac{1}{2} \|\mathbf{f}(\mathbf{x})\|^2 = \frac{1}{2} \sum_{i=1}^N |f_i(\mathbf{x})|^2 \quad (2.6)$$

The solution that minimizes  $J(\mathbf{x})$  is also the solution of the original equation. To see this, we again set the gradient of  $J(\mathbf{x})$  zero to get as above:

$$\mathbf{g}_J(\mathbf{x}) = \nabla_{\mathbf{x}} J(\mathbf{x}) = \frac{d}{d\mathbf{x}} \left[ \frac{1}{2} \mathbf{f}^T(\mathbf{x}) \mathbf{f}(\mathbf{x}) \right] = \mathbf{f}'(\mathbf{x}) \mathbf{f}(\mathbf{x}) = \mathbf{J}_f(\mathbf{x}) \mathbf{f}(\mathbf{x}) = \mathbf{0} \quad (2.7)$$

As in general the Jacobian  $\mathbf{J}_f(\mathbf{x}) = \mathbf{f}'(\mathbf{x})$  is a full rank matrix, the homogeneous equation above only has a zero solution  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ , which is indeed the original equation.

### 2.1.2 Newton's method

We first consider Newton's method applied to minimizing a univariate function  $f(x)$ . The Taylor series expansion (Section A.22) of the function  $f(x)$  around a point  $x_0$  is:

$$\begin{aligned} f(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \cdots + \frac{1}{n!}f^{(n)}(x_0)(x - x_0)^n + \cdots \\ &\approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 = q(x) \end{aligned} \quad (2.8)$$

where the first and second order derivatives  $f'(x)$  and  $f''(x)$  represent respectively the local change and the more global curvature of the function.

If  $f(x)$  is a quadratic function denoted by  $q(x)$ , then its Taylor series contains only the first three terms (constant, linear, and quadratic terms). In this case, we can find the vertex point at which  $f(x) = q(x)$  reaches its extremum, by first setting its derivative to zero:

$$\begin{aligned} q'(x) &= \frac{d}{dx}q(x) = \frac{d}{dx} \left[ f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 \right] \\ &= f'(x_0) + f''(x_0)(x - x_0) = 0 \end{aligned} \quad (2.9)$$

and then solving the resulting equation to get:

$$x^* = x = x_0 - \frac{f'(x_0)}{f''(x_0)} = x_0 + \Delta x_0 \quad (2.10)$$

where  $\Delta x_0 = -f'(x_0)/f''(x_0)$  is the step we need to take to move from any initial point  $x_0$  to the solution  $x^*$  in a single step. Note that  $f(x^*)$  is a minimum if  $f''(x^*) > 0$  but a maximum if  $f''(x^*) < 0$ .

If  $f(x) \neq q(x)$  is not quadratic, the result above can still be considered as an approximation of the solution, which can be improved iteratively from the initial guess  $x_0$  to eventually approach the solution:

$$x_{n+1} = x_n + \Delta x_n = x_n - \frac{f'(x_n)}{f''(x_n)}, \quad n = 0, 1, 2, \dots \quad (2.11)$$

where  $\Delta x_n = -f'(x_n)/f''(x_n)$ . We see that in each step  $x_n$  of the iteration, the function  $f(x)$  is fitted by a quadratic functions  $q(x_n)$  and its vertex at  $x_{n+1}$  is used as the updated approximated solution, at which the function is again fitted by another quadratic function  $q(x_{n+1})$  for the next iteration. Through this process the solution can be approached as  $n \rightarrow \infty$ .

We note that the iteration  $x_{n+1} = x_n - f'(x_n)/f''(x_n)$  is just Eq. (1.18) in Chapter 1, the Newton-Raphson method applied to solving equation  $f'(x) = 0$ , as the necessary condition for an extremum of  $f(x)$ .

Newton's method for the minimization of a single-variable function  $f(x)$  can be generalized for the minimization of a multi-variable function  $f(\mathbf{x}) = f(x_1, \dots, x_N)$ . Again, we approximate  $f(\mathbf{x})$  by a quadratic function  $q(\mathbf{x})$  containing only the first three terms of its Taylor series at some initial point  $\mathbf{x}_0$ :

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \mathbf{g}_0^T (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \mathbf{H}_0 (\mathbf{x} - \mathbf{x}_0) = q(\mathbf{x}) \quad (2.12)$$

where  $\mathbf{g}_0$  and  $\mathbf{H}_0$  are respectively the gradient vector and Hessian matrix of the function  $f(\mathbf{x})$  at  $\mathbf{x}_0$ :

$$\mathbf{g}_0 = \mathbf{g}_f(\mathbf{x}_0) = \frac{d}{d\mathbf{x}} f(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial f(\mathbf{x}_0)}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x}_0)}{\partial x_N} \end{bmatrix},$$

$$\mathbf{H}_0 = \mathbf{H}_f(\mathbf{x}_0) = \frac{d}{d\mathbf{x}} \mathbf{g}_f(\mathbf{x}_0) = \frac{d^2}{d\mathbf{x}^2} f(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x}_0)}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\mathbf{x}_0)}{\partial x_1 \partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x}_0)}{\partial x_N \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x}_0)}{\partial x_N^2} \end{bmatrix} \quad (2.13)$$

The stationary point of  $f(\mathbf{x}) = q(\mathbf{x})$  can be found from any initial guess  $\mathbf{x}_0$  by setting its derivative to zero:

$$\begin{aligned} \frac{d}{d\mathbf{x}} q(\mathbf{x}) &= \frac{d}{d\mathbf{x}} \left[ f(\mathbf{x}_0) + \mathbf{g}_0^T (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \mathbf{H}_0 (\mathbf{x} - \mathbf{x}_0) \right] \\ &= \mathbf{g}_0 + \mathbf{H}_0 (\mathbf{x} - \mathbf{x}_0) = \mathbf{0} \end{aligned} \quad (2.14)$$

and solving the resulting equation to get

$$\mathbf{x}^* = \mathbf{x} = \mathbf{x}_0 - \mathbf{H}(\mathbf{x}_0)^{-1} \mathbf{g}(\mathbf{x}_0) \quad (2.15)$$

Similar to the single-variable case in which  $f(x^*)$  is either a minimum or maximum depends on whether the second order derivative  $f''(x^*)$  is greater or smaller than zero, here in multi-variable case the function value  $f(\mathbf{x}^*)$  at the stationary point is a maximum, minimum, or saddle point, depending on the curvature of the function represented by the Hessian matrix  $\mathbf{H}^* = \mathbf{H}^* = \mathbf{H}_f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*}$  evaluated at  $\mathbf{x}^*$ :

- If  $\mathbf{H}^* > 0$  is positive definite (all eigenvalues are positive),  $f(\mathbf{x}^*)$  is a minimum;
- If  $\mathbf{H}^* < 0$  is negative definite (all eigenvalues are negative),  $f(\mathbf{x}^*)$  is a maximum;
- If  $\mathbf{H}^*$  is indefinite (with both positive and negative eigenvalues),  $f(\mathbf{x}^*)$  is a saddle point (maximum in some directions, but minimum in others).

If  $f(\mathbf{x}) \neq q(\mathbf{x})$  is not quadratic, the result above is not a minimum or maximum, but it can still be used as an approximation of the solution, which can be improved to approach the solution iteratively:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta \mathbf{x}_n = \mathbf{x}_n - \mathbf{H}_n^{-1} \mathbf{g}_n = \mathbf{x}_n + \mathbf{d}_n, \quad n = 0, 1, 2, \dots \quad (2.16)$$



where  $\mathbf{g}_n = \mathbf{g}(\mathbf{x}_n)$ ,  $\mathbf{H}_n = \mathbf{H}(\mathbf{x}_n)$ , and the increment  $\mathbf{d}_n = \Delta\mathbf{x}_n = -\mathbf{H}_n^{-1}\mathbf{g}_n$  is called *Newton search direction*. We note that this iteration is just a generalization of  $x_{n+1} = x_n - f'(x_n)/f''(x_n)$  in 1-D case where  $\mathbf{g}_n = f'(x_n)$  and  $\mathbf{H}_n = f''(x_n)$ , and the iteration  $\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{H}_n^{-1}\mathbf{g}_n$  above is just Eq. (1.100) in Chapter 1, the Newton-Raphson method applied to solving equation  $f'(\mathbf{x}) = \mathbf{g}_f(\mathbf{x}) = \mathbf{0}$ , as the necessary condition for an extremum of  $\mathbf{f}(\mathbf{x})$ . The computational complexity for each iteration is  $O(N^3)$  due to the inverse operation  $\mathbf{H}_n^{-1}$ .

**Example 1:**

An over-constrained nonlinear equation system  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  of  $M$  equations and  $N < M$  unknowns can be solved by minimizing the following SSE:

$$\varepsilon(\mathbf{x}) = \frac{1}{2}\|\mathbf{f}(\mathbf{x})\|^2 = \frac{1}{2}\sum_{m=1}^M f_m^2(\mathbf{x}) \quad (2.17)$$

To use the Newton-Raphson method, we first find the gradient of the error function:

$$\mathbf{g}_\varepsilon(\mathbf{x}) = \frac{d}{d\mathbf{x}}\varepsilon(\mathbf{x}) = \frac{1}{2}\frac{d}{d\mathbf{x}}\|\mathbf{f}(\mathbf{x})\|^2 = \frac{d\mathbf{f}}{d\mathbf{x}}\mathbf{f} = \mathbf{J}_f^T\mathbf{f} \quad (2.18)$$

of which the  $i$ th component is:

$$g_i = \frac{\partial}{\partial x_i}\left(\frac{1}{2}\|\mathbf{f}\|^2\right) = \frac{1}{2}\sum_{m=1}^M\frac{\partial}{\partial x_i}f_m^2 = \sum_{m=1}^M\frac{\partial f_m}{\partial x_i}f_m = \sum_{m=1}^M J_{mi}f_m \quad (2.19)$$

where  $J_{mi} = \partial f_m/\partial x_i$  is the component of the function's Jacobian  $\mathbf{J}_f(\mathbf{x})$  in the  $m$ th row and  $i$ th column.

We further find the component of the Hessian  $\mathbf{H}_\varepsilon$  of the error function in the  $i$ th row and  $j$ th column:

$$\begin{aligned} H_{ij} &= \frac{\partial^2}{\partial x_i\partial x_j}\left(\frac{1}{2}\|\mathbf{f}\|^2\right) = \frac{\partial}{\partial x_j}g_i = \sum_{m=1}^M\frac{\partial}{\partial x_j}\left[f_m\frac{\partial f_m}{\partial x_i}\right] \\ &= \sum_{m=1}^M\left[\frac{\partial f_m}{\partial x_i}\frac{\partial f_m}{\partial x_j} + f_m\frac{\partial^2 f_m}{\partial x_i\partial x_j}\right] \\ &\approx \sum_{m=1}^M\frac{\partial f_m}{\partial x_i}\frac{\partial f_m}{\partial x_j} = \sum_{m=1}^M J_{mi}J_{mj} \end{aligned} \quad (2.20)$$

Here  $J_{ij}$  is the component of the Jacobian  $\mathbf{J}_f$  in the  $i$ th row and  $j$ th column, and we have dropped the second order terms  $f_m$  ( $\partial^2 f_m/\partial x_i\partial x_j$ ). Now the Hessian matrix can be written as

$$\mathbf{H}_\varepsilon(\mathbf{x}) = \frac{d^2}{d\mathbf{x}^2}\varepsilon(\mathbf{x}) = \frac{d}{d\mathbf{x}}\mathbf{g}_\varepsilon(\mathbf{x}) \approx \mathbf{J}_f^T\mathbf{J}_f \quad (2.21)$$

Based on Newton-Raphson's method, the optimal solution  $\mathbf{x}^*$  that minimizes  $\varepsilon(\mathbf{x})$  can be found iteratively:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{H}_n^{-1}\mathbf{g}_n \approx \mathbf{x}_n - (\mathbf{J}_n^T\mathbf{J}_n)^{-1}\mathbf{J}_n^T\mathbf{f}_n \quad (2.22)$$

This is the same as pseudo-inverse method in Eq. (1.110) in Chapter 1.

**Example 2:**

Consider the following quadratic function:

$$q(x, y) = \frac{1}{2}[x_1, x_2] \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{2}(ax_1^2 + bx_1x_2 + cx_2^2)$$

We have

$$\mathbf{g} = \begin{bmatrix} ax_1 + bx_2/2 \\ bx_1/2 + cx_2 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}, \quad \det \mathbf{H} = ac - b^2/4$$

We let  $a = 1$ ,  $c = 2$ , and consider the following values of  $b$ :

- $b = -2$ ,  $\lambda_1 = 2.618$ ,  $\lambda_2 = 0.382$ ,  $\det \mathbf{H} = \lambda_1\lambda_2 = 1$ ,  $\mathbf{H} > 0$  is positive definite,  $f(0, 0) = 0$  is the minimum.
- $b = 0$ ,  $\lambda_1 = 2$ ,  $\lambda_2 = 1$ ,  $\det \mathbf{H} = \lambda_1\lambda_2 = 2$ ,  $\mathbf{H} > 0$  is positive definite,  $f(0, 0) = 0$  is the minimum.
- $b = 2$ ,  $\lambda_1 = 2.618$ ,  $\lambda_2 = 0.382$ ,  $\det \mathbf{H} = \lambda_1\lambda_2 = 1$ ,  $\mathbf{H} > 0$  is positive definite,  $f(0, 0) = 0$  is the minimum.
- $b = 4$ ,  $\lambda_1 = 3.562$ ,  $\lambda_2 = -0.562$ ,  $\det \mathbf{H} = \lambda_1\lambda_2 = -2$ ,  $\mathbf{H}$  is indefinite,  $f(0, 0) = 0$  is a saddle point (minimum in one direction but maximum in another).

We can speed up the convergence by a bigger step size  $\delta > 1$ . However, if  $\delta$  is too big, the solution may be skipped and the iteration may not converge if it gets into an oscillation around the solution. Even worse, the iteration may become divergent. For such reasons, a smaller step size  $\delta < 1$  may be preferred sometimes.

In summary, Newton's method approximates the function  $f(\mathbf{x})$  at an estimated solution  $\mathbf{x}_n$  by a quadratic equation (the first three terms of the Taylor's series) based on the gradient  $\mathbf{g}_n$  and Hessian  $\mathbf{H}$  of the function at  $\mathbf{x}_n$ , and treat the vertex of the quadratic equation as the updated estimate  $\mathbf{x}_{n+1}$ .

Newton's method requires the Hessian matrix as well as the gradient to be available. Moreover, it is necessary calculate the inverse of the Hessian matrix in each iteration, which may be computationally expensive.

**Example 3:**

The Newton's method is applied to solving the following non-linear equation system of  $N = 3$  variables:

$$\begin{cases} f_1(x_1, x_2, x_3) = 3x_1 - (x_2x_3)^2 - 3/2 \\ f_2(x_1, x_2, x_3) = 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ f_3(x_1, x_2, x_3) = \exp(-x_1x_2) + 20x_3 + 9 \end{cases}$$

with the exact solution ( $x_1 = 0.5$ ,  $x_2 = 0$ ,  $x_3 = -0.5$ ). These equations can be expressed in vector form as  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  and solved as an optimization problem with the objective function  $J(\mathbf{x}) = \mathbf{f}^T(\mathbf{x})\mathbf{f}(\mathbf{x})$ . The iteration from an initial guess

$\mathbf{x}_0 = \mathbf{0}$  is shown below.

$n$	$\mathbf{x} = [x_1, x_2, x_3]$	$\ \mathbf{f}(\mathbf{x})\ $
0	0.000000, 0.000000, 0.000000	$1.016120e + 01$
1	0.500000, 0.500000, -0.500000	$1.552502e + 02$
2	0.499550, 0.250800, -0.493801	$3.881300e + 01$
3	0.500096, 0.126206, -0.496852	$9.702208e + 00$
4	0.500025, 0.063914, -0.498405	$2.425198e + 00$
5	0.500010, 0.032778, -0.499181	$6.059054e - 01$
6	0.500005, 0.017231, -0.499570	$1.510777e - 01$
7	0.500003, 0.009498, -0.499763	$3.737330e - 02$
8	0.500002, 0.005712, -0.499857	$8.959365e - 03$
9	0.500001, 0.003968, -0.499901	$1.900145e - 03$
10	0.500001, 0.003326, -0.499917	$2.577603e - 04$
11	0.500001, 0.003206, -0.499920	$8.932714e - 06$
12	0.500001, 0.003202, -0.499920	$1.238536e - 08$
13	0.500001, 0.003202, -0.499920	$2.371437e - 14$

We see that after 13 iterations the algorithm converges to the following approximated solution with accuracy of  $J(\mathbf{x}) = \|\mathbf{f}(\mathbf{x}^*)\|^2 \approx 10^{-28}$ :

$$\mathbf{x}^* = \begin{bmatrix} 0.5000008539707297 \\ 0.0032017070323056 \\ -0.4999200212218281 \end{bmatrix} \quad (2.24)$$

### 2.1.3 Gradient Descent Method

Newton's method discussed above is based on the gradient  $\mathbf{g}_f(\mathbf{x})$  and Hessian  $\mathbf{H}_f(\mathbf{x})$ , representing the local variation and more global curvature of the function  $f(\mathbf{x})$  to be minimized. The method is not applicable if the Hessian  $\mathbf{H}_f$  is not available, because either the analytical form of the function is not known or the cost for computing the inverse  $\mathbf{H}_f^{-1}$  is too high. In such cases, we can use the gradient descent method based only on the gradient but no longer the Hessian.

We first consider the minimization of a single-variable function  $f(x)$ . From any initial point  $x_0$ , we can move to a nearby point  $x$  along the opposite direction of the gradient, so that the function value will be smaller than that at the current position  $x_0$ :

$$x = x_0 - \delta f'(x_0) = x_0 + \Delta x_0, \quad \text{where } \Delta x_0 = -\delta f'(x_0), \quad \delta > 0 \quad (2.25)$$

We see that no matter whether  $f'(x_0)$  is positive or negative, the function value  $f(x)$  approximated by the first two terms of its Taylor series is always reduced if the positive step size  $\delta$  is small enough:

$$f(x) \approx f(x_0) + f'(x_0)\Delta x_0 = f(x_0) - |f'(x_0)|^2\delta < f(x_0) \quad (2.26)$$

This process can be carried out iteratively

$$x_{n+1} = x_n - \delta_n f'(x_n), \quad n = 0, 1, 2, \dots \quad (2.27)$$

until eventually reaching a point  $x^*$  at which  $f'(x^*) = 0$  and no further progress can be made, i.e. a local minimum of the function is reached.

This simple method can be generalized to minimize a multi-variable objective function  $f(\mathbf{x}) = f(x_1, \dots, x_N)$  in N-D space. The derivative of the 1-D case is generalized to the gradient vector  $\mathbf{g}(\mathbf{x}) = df(\mathbf{x})/d\mathbf{x}$  of function  $f(\mathbf{x})$ , which is in the direction along which the function increases most rapidly with the steepest slope, perpendicular to the contour or iso-lines of the function  $f(\mathbf{x})$ . The fastest way to reduce  $f(\mathbf{x})$  is to go down hill along the opposite direction of the gradient vector.

Specifically the gradient descent method (also called steepest descent or down hill method) carries out the iteration  $\mathbf{x} = \mathbf{x}_0 + \Delta\mathbf{x}$ , which can be considered as the first two terms of the Taylor series of  $f(\mathbf{x})$  with  $\Delta\mathbf{x} = -\delta\mathbf{g}$  ( $\delta > 0$ ):

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \mathbf{g}_0^T \Delta\mathbf{x} = f(\mathbf{x}_0) - \delta \mathbf{g}_0^T \mathbf{g}_0 = f(\mathbf{x}_0) - \delta \|\mathbf{g}_0\|^2 < f(\mathbf{x}_0) \quad (2.28)$$

As the function value is always reduced by this iteration, it can be carried out repeatedly

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \delta_n \mathbf{g}_n = (\mathbf{x}_{n-1} - \delta_{n-1} \mathbf{g}_{n-1}) - \delta_n \mathbf{g}_n = \dots = \mathbf{x}_0 - \sum_{i=0}^n \delta_i \mathbf{g}_i \quad (2.29)$$

until eventually reaching a point  $\mathbf{x}^*$  at which  $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$  and  $f(\mathbf{x})$  is minimized.

Comparing Eq. (2.29) with Newton's method where the search direction is  $\mathbf{d}_n = -\mathbf{H}_n^{-1} \mathbf{g}_n$  in Eq. (2.16) based on both  $\mathbf{H}_n$  and  $\mathbf{g}_n$ , we see that here in gradient descent method the search direction  $\mathbf{d}_n = -\mathbf{g}_n$  relies only on the local information contained in the gradient  $\mathbf{g}$  without more global information contained in the second order Hessian  $\mathbf{H}$ , i.e., the gradient method does not have as much information as Newton's method and therefore may not be as efficient. For example, when the function is quadratic, as discussed before, Newton's method can find the solution in a single step from any initial guess, but it takes the gradient descent method multiple steps to reach the solution, as it always follows the negative direction of the local gradient, which does not lead toward the solution directly in general. However, the gradient descent method is computationally less expensive as the Hessian matrix is not needed.

**Example 1:**

Consider a two-variable quadratic function in the following general form:

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} = [x_1 \ x_2] \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

If

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

is a symmetric positive semidefinite matrix, i.e.,  $a_{12} = a_{21}$  and  $\mathbf{A} \geq 0$ , then the function becomes

$$f(x_1, x_2) = \frac{1}{2} [x_1 \ x_2] \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{2} (2x_1^2 + 2x_1x_2 + x_2^2)$$

which has a minimum  $f(x_1, x_2) = 0$  at  $x_1 = x_2 = 0$ .

We let the initial guess be  $\mathbf{x}_0 = [1, 2]^T$ , at which the gradient is  $\mathbf{g}_0 = [4, 3]^T$ . Now we compare the gradient method with Newton's method, in terms of the search direction  $\mathbf{d}$  and progress of the iterations:

- Newton's method: Here we have

$$\mathbf{g} = \begin{bmatrix} 2x_1 + x_2 \\ x_1 + x_2 \end{bmatrix}, \quad \mathbf{H} = \mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{H}^{-1} = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$$

and the search direction is  $\mathbf{d}_0 = -\mathbf{H}^{-1}\mathbf{g}_0 = -[1, 2]^T$  (the red arrow in the figure). The iteration is:

$$\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{H}^{-1}\mathbf{g}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

which is the minimum of the function.

- The gradient descent method: the search direction is  $\mathbf{d}_0 = -\mathbf{g}_0 = -[4, 3]^T$ , perpendicular to the contour of the function. The first iteration is:

$$\mathbf{x}_1 = \mathbf{x}_0 - \delta \mathbf{g}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \delta \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 - \delta 4 \\ 2 - \delta 3 \end{bmatrix}$$

We need to determine the step size  $\delta$  (to be considered later) to find  $\mathbf{x}_1$ , and then continue the iteration.

The figure below shows the search direction of the Newton's method (red), which reaches the solution in a single step, based on not only the gradient  $\mathbf{g}_0$ , the local information at the point  $\mathbf{x}_0$ , but also the Hessian matrix  $\mathbf{H}$ , the more global information of the quadratic function representing its curvature, in comparison to the gradient descent method (blue) based only on the local gradient  $\mathbf{g}_0$ . Without global information  $\mathbf{H}$  representing the elliptical shape of the contour line of the quadratic function, the gradient descent method requires an iteration, and is therefore less effective.

In the gradient descent method, we also need to determine a proper step size  $\delta$ . If  $\delta$  is too small, the iteration may converge too slowly, especially when  $f(x)$  reduces slowly toward its minimum. On the other hand, if  $\delta$  is too large but  $f(x)$  has some rapid variations in the local region, the minimum of the function may be skipped and the iteration may not converge. We will consider how to find the optimal step size in the next section.

### 2.1.4 Line minimization

In general, in any iterative algorithm for minimizing a function  $f(\mathbf{x})$ , including Newton's method (Eq. (2.16)) and the gradient descent method (Eq. (2.29)), the variable  $\mathbf{x}$  is updated iteratively  $\mathbf{x}_{n+1} = \mathbf{x}_n + \delta_n \mathbf{d}_n$  based on both the search direction  $\mathbf{d}_n$  and the step size  $\delta$ , which need to be determined in such a way that the function value  $f(\mathbf{x})$  is maximally reduced at each iteration step.

First, the search direction  $\mathbf{d}_n$  needs to point away from the gradient  $\mathbf{g}_n$  along which  $f(\mathbf{x}_n)$  increases most rapidly. In other words, the angle between  $\mathbf{d}_n$  and  $\mathbf{g}_n$  should be greater than  $\pi/2$ :

$$\cos^{-1} \left( \frac{\mathbf{d}_n^T \mathbf{g}_n}{\|\mathbf{d}_n\| \|\mathbf{g}_n\|} \right) > \frac{\pi}{2} \quad \text{i.e.,} \quad \mathbf{d}_n^T \mathbf{g}_n < 0 \quad (2.30)$$

This condition is indeed satisfied in both the gradient descent method with  $\mathbf{d}_n = -\mathbf{g}_n$  and Newton's method with  $\mathbf{d}_n = -\mathbf{H}_n^{-1} \mathbf{g}_n$ :

$$\mathbf{d}_n^T \mathbf{g}_n = -\mathbf{g}_n^T \mathbf{H}_n^{-1} \mathbf{g}_n < 0, \quad \mathbf{d}_n^T \mathbf{g}_n = -\mathbf{g}_n^T \mathbf{g}_n = -\|\mathbf{g}_n\|^2 < 0 \quad (2.31)$$

Here  $\mathbf{H}_n$  is assumed to be positive definite so that a minimum exists.

Next, the optimal step size  $\delta_n$  needs to be such that the function value  $f(\mathbf{x}_{n+1}) = f(\mathbf{x}_n + \delta_n \mathbf{d}_n)$  is minimized along the search direction  $\mathbf{d}_n$ . To do so, we set to zero the derivative of the function value with respect to  $\delta_n$ , and get the *directional derivative* (Section A.18) along the direction of  $\mathbf{d}_n$  based on the chain rule:

$$\frac{d}{d\delta_n} f(\mathbf{x}_{n+1}) = \frac{d}{d\delta_n} f(\mathbf{x}_n + \delta_n \mathbf{d}_n) = \left( \frac{df(\mathbf{x}_{n+1})}{d\mathbf{x}} \right)^T \frac{d(\mathbf{x}_n + \delta_n \mathbf{d}_n)}{d\delta_n} = \mathbf{g}_{n+1}^T \mathbf{d}_n = 0 \quad (2.32)$$

This result indicates that the gradient  $\mathbf{g}_{n+1} = f'(\mathbf{x}_{n+1})$  at the next point  $\mathbf{x}_{n+1}$  should be perpendicular to the search direction  $\mathbf{d}_n$ . In other words, when traversing along  $\mathbf{d}_n$ , we should stop at the point  $\mathbf{x}_{n+1} = \mathbf{x}_n + \delta \mathbf{d}_n$  at which the gradient  $\mathbf{g}_{n+1}$  is perpendicular to  $\mathbf{d}_n$ , i.e., it has zero component along  $\mathbf{d}_n$ , and the corresponding  $\delta$  is the optimal step size.

To find the actual optimal step size  $\delta$ , we need to solve a 1-D optimization problem while treating  $f(\mathbf{x}_{n+1}) = f(\mathbf{x}_n + \delta \mathbf{d}_n)$  as a function of the step size  $\delta$ , and approximate it by the first three terms of its Taylor series at  $\delta = 0$  (Maclaurin series of function  $f(\delta)$ ):

$$\begin{aligned} f(\mathbf{x}_{n+1}) &= f(\mathbf{x}_n + \delta \mathbf{d}_n) \\ &\approx [f(\mathbf{x}_n + \delta \mathbf{d}_n)]_{\delta=0} + \delta \left[ \frac{d}{d\delta} f(\mathbf{x}_n + \delta \mathbf{d}_n) \right]_{\delta=0} + \frac{\delta^2}{2} \left[ \frac{d^2}{d\delta^2} f(\mathbf{x}_n + \delta \mathbf{d}_n) \right]_{\delta=0} \end{aligned} \quad (2.33)$$

where

$$[f(\mathbf{x}_n + \delta \mathbf{d}_n)]_{\delta=0} = f(\mathbf{x}_n) \quad (2.34)$$

$$\left[ \frac{d}{d\delta} f(\mathbf{x}_n + \delta \mathbf{d}_n) \right]_{\delta=0} = [\mathbf{g}(\mathbf{x}_n + \delta \mathbf{d}_n)^T \mathbf{d}_n]_{\delta=0} = \mathbf{g}_n^T \mathbf{d}_n \quad (2.35)$$

$$\begin{aligned} \left[ \frac{d^2}{d\delta^2} f(\mathbf{x}_n + \delta \mathbf{d}_n) \right]_{\delta=0} &= \left[ \frac{d}{d\delta} \mathbf{g}(\mathbf{x}_n + \delta \mathbf{d}_n)^T \right]_{\delta=0} \frac{d}{d\delta} (\mathbf{x}_n + \delta \mathbf{d}_n) \\ &= \left[ \frac{d}{d\mathbf{x}} \mathbf{g}(\mathbf{x}) \frac{d}{d\delta} (\mathbf{x}_n + \delta \mathbf{d}_n) \right]_{\delta=0}^T \mathbf{d}_n \\ &= (\mathbf{H}_n \mathbf{d}_n)^T \mathbf{d}_n = \mathbf{d}_n^T \mathbf{H}_n \mathbf{d}_n \end{aligned} \quad (2.36)$$

where  $\mathbf{H}_n = \mathbf{H}_n^T$  is the Hessian matrix of  $f(\mathbf{x})$  at  $\mathbf{x}_n$ . Substituting these back into the Taylor series above we get:

$$f(\mathbf{x}_n + \delta \mathbf{d}_n) \approx f(\mathbf{x}_n) + \delta \mathbf{g}_n^T \mathbf{d}_n + \frac{\delta^2}{2} \mathbf{d}_n^T \mathbf{H}_n \mathbf{d}_n \quad (2.37)$$

To find  $\delta$  that minimizes  $f(\mathbf{x}_n + \delta \mathbf{d}_n)$ , we set to zero its derivative with respect to  $\delta$ :

$$\begin{aligned} \frac{d}{d\delta} f(\mathbf{x}_n + \delta \mathbf{d}_n) &\approx \frac{d}{d\delta} \left( f(\mathbf{x}_n) + \delta \mathbf{g}_n^T \mathbf{d}_n + \frac{\delta^2}{2} \mathbf{d}_n^T \mathbf{H}_n \mathbf{d}_n \right) \\ &= \mathbf{g}_n^T \mathbf{d}_n + \delta \mathbf{d}_n^T \mathbf{H}_n \mathbf{d}_n = 0 \end{aligned} \quad (2.38)$$

and solve the resulting equation for  $\delta$  to get the optimal step size based on both  $\mathbf{g}_n$  and  $\mathbf{H}_n$ :

$$\delta_n = -\frac{\mathbf{g}_n^T \mathbf{d}_n}{\mathbf{d}_n^T \mathbf{H}_n \mathbf{d}_n} \quad (2.39)$$

Given this result, we can find the optimal step size for the Newton's method and gradient descent method considered previously:

- **Newton's method:**

The search direction is  $\mathbf{d}_n = -\mathbf{H}_n^{-1} \mathbf{g}_n$ , and the optimal step size is

$$\delta_n = -\frac{\mathbf{g}_n^T \mathbf{d}_n}{\mathbf{d}_n^T \mathbf{H}_n \mathbf{d}_n} = \frac{\mathbf{g}_n^T (\mathbf{H}_n^{-1} \mathbf{g}_n)}{(\mathbf{H}_n^{-1} \mathbf{g}_n)^T \mathbf{H}_n (\mathbf{H}_n^{-1} \mathbf{g}_n)} = \frac{\mathbf{g}_n^T (\mathbf{H}_n^{-1} \mathbf{g}_n)}{(\mathbf{H}_n^{-1} \mathbf{g}_n)^T \mathbf{g}_n} = 1 \quad (2.40)$$

This confirms that the iteration in Eq. (2.16) is indeed optimal:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \delta_n \mathbf{d}_n = \mathbf{x}_n - \delta_n \mathbf{H}_n^{-1} \mathbf{g}_n = \mathbf{x}_n - \mathbf{H}_n^{-1} \mathbf{g}_n \quad (2.41)$$

- **Gradient descent method:**

The search direction is  $\mathbf{d}_n = -\mathbf{g}_n$ , and Eq. (2.32) becomes:

$$\mathbf{g}_{n+1}^T \mathbf{d}_n = -\mathbf{g}_{n+1}^T \mathbf{g}_n = 0, \quad \text{i.e.,} \quad \mathbf{g}_{n+1} \perp \mathbf{g}_n \quad (2.42)$$

i.e., the search direction  $\mathbf{d}_{n+1} = -\mathbf{g}_{n+1}$  is always perpendicular to the previous one  $\mathbf{d}_n = -\mathbf{g}_n$ , i.e., the iteration follows a zigzag path composed

of a sequence of segments from the initial guess to the final solution. The optimal step size is

$$\delta_n = -\frac{\mathbf{g}_n^T \mathbf{d}_n}{\mathbf{d}_n^T \mathbf{H}_n \mathbf{d}_n} = \frac{\mathbf{g}_n^T \mathbf{g}_n}{\mathbf{g}_n^T \mathbf{H}_n \mathbf{g}_n} = \frac{\|\mathbf{g}_n\|^2}{\mathbf{g}_n^T \mathbf{H}_n \mathbf{g}_n} \quad (2.43)$$

Given  $\mathbf{H}_n$  as well as  $\mathbf{g}_n$ , we can find the optimal step size  $\delta_n$  with complexity  $O(N^2)$  for each iteration. However, as we assume the Hessian  $\mathbf{H}_n$  is not available in the gradient descent method, the optimal step size above cannot be computed. We can instead approximate  $f''(\delta)|_{\delta=0}$  in the third term of Eq. (2.33) at two nearby points at  $\delta = 0$  and  $\delta = \sigma$ , where  $\sigma$  is a small value:

$$\begin{aligned} \left[ \frac{d^2}{d\delta^2} f(\mathbf{x} + \delta \mathbf{d}) \right]_{\delta=0} &= \left[ \frac{d}{d\delta} f'(\mathbf{x} + \delta \mathbf{d}) \right]_{\delta=0} = \lim_{\sigma \rightarrow 0} \frac{f'(\mathbf{x} + \sigma \mathbf{d}) - f'(\mathbf{x})}{\sigma} \\ &\approx \frac{\mathbf{g}^T(\mathbf{x} + \sigma \mathbf{d}) \mathbf{d} - \mathbf{g}^T(\mathbf{x}) \mathbf{d}}{\sigma} = \frac{\mathbf{g}_\sigma^T \mathbf{d} - \mathbf{g}^T \mathbf{d}}{\sigma} \end{aligned} \quad (2.44)$$

where  $\mathbf{g} = \mathbf{g}(\mathbf{x})$  and  $\mathbf{g}_\sigma = \mathbf{g}(\mathbf{x} + \sigma \mathbf{d})$ . This approximation can be used to replace  $\mathbf{d}_n^T \mathbf{H}_n \mathbf{d}_n$  in Eq. (2.38) above:

$$\frac{d}{d\delta} f(\mathbf{x}_n + \delta \mathbf{d}_n) \approx \mathbf{g}_n^T \mathbf{d}_n + \frac{\delta}{\sigma} (\mathbf{g}_{\sigma n}^T \mathbf{d}_n - \mathbf{g}_n^T \mathbf{d}_n) = 0 \quad (2.45)$$

Solving for  $\delta$  we get the estimated optimal step size:

$$\delta_n = -\frac{\sigma \mathbf{g}_n^T \mathbf{d}_n}{\mathbf{g}_{\sigma n}^T \mathbf{d}_n - \mathbf{g}_n^T \mathbf{d}_n} = -\frac{\sigma \mathbf{g}_n^T \mathbf{d}_n}{(\mathbf{g}_{\sigma n} - \mathbf{g}_n)^T \mathbf{d}_n} \quad (2.46)$$

Specifically in the gradient descent method with  $\mathbf{d}_n = -\mathbf{g}_n$ , this optimal step size becomes:

$$\delta_n = -\frac{\sigma \mathbf{g}_n^T \mathbf{d}_n}{(\mathbf{g}_{\sigma n} - \mathbf{g}_n)^T \mathbf{d}_n} = -\frac{\sigma \mathbf{g}_n^T \mathbf{g}_n}{(\mathbf{g}_{\sigma n} - \mathbf{g}_n)^T \mathbf{g}_n} = \frac{\sigma \|\mathbf{g}_n\|^2}{\|\mathbf{g}_n\|^2 - \mathbf{g}_{\sigma n}^T \mathbf{g}_n} \quad (2.47)$$

and the iteration becomes

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \delta_n \mathbf{g}_n = \mathbf{x}_n + \frac{\sigma \|\mathbf{g}_n\|^2}{\mathbf{g}_{\sigma n}^T \mathbf{g}_n - \|\mathbf{g}_n\|^2} \mathbf{g}_n \quad (2.48)$$

### Example 2:

The gradient descent method applied to solve the same three-variable equation system previously solved by Newton's method:

$$\begin{cases} f_1(x_1, x_2, x_3) = 3x_1 - (x_2 x_3)^2 - 3/2 \\ f_2(x_1, x_2, x_3) = 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ f_3(x_1, x_2, x_3) = \exp(-x_1 x_2) + 20x_3 + 9 \end{cases}$$

The step size  $\delta_n$  is determined by the secant method with  $\sigma = 10^{-6}$ . The



iteration from an initial guess  $\mathbf{x}_0 = \mathbf{0}$  is shown below:

$n$	$\mathbf{x} = [x_1, x_2, x_3]$	$\ \mathbf{f}(\mathbf{x})\ $
0	0.0000, 0.0000, 0.0000	$1.032500e + 02$
10	0.4246, -0.0073, -0.5002	$1.535939e - 01$
20	0.5015, 0.0064, -0.4998	$2.448241e - 05$
30	0.5009, 0.0057, -0.4998	$9.178209e - 06$
40	0.5006, 0.0052, -0.4998	$4.17587e - 06$
50	0.5004, 0.0049, -0.4999	$2.122594e - 06$
60	0.5003, 0.0047, -0.4999	$1.182466e - 06$
100	0.5001, 0.0043, -0.4999	$1.805871e - 07$
150	0.5000, 0.0041, -0.4999	$2.720744e - 08$
200	0.5000, 0.0041, -0.4999	$4.934027e - 09$
250	0.5000, 0.0040, -0.4999	$9.630085e - 10$
300	0.5000, 0.0040, -0.4999	$1.939747e - 10$
350	0.5000, 0.0040, -0.4999	$3.961646e - 11$
400	0.5000, 0.0040, -0.4999	$8.140393e - 12$
450	0.5000, 0.0040, -0.4999	$1.677968e - 12$
500	0.5000, 0.0040, -0.4999	$3.462695e - 13$
550	0.5000, 0.0040, -0.4999	$7.136628e - 14$
600	0.5000, 0.0040, -0.4999	$1.474205e - 14$

We see that after the first 100 iterations the error is reduced to about  $J(\mathbf{x}) = \|\mathbf{f}(\mathbf{x}^*)\|^2 \approx 10^{-14}$ , and With 500 additional iterations the algorithm converges to the following approximated solution with accuracy of  $J(\mathbf{x}) \approx 10^{-28}$ :

$$\mathbf{x}^* = \begin{bmatrix} 0.5000013623816102 \\ 0.0040027495837189 \\ -0.4999000311539049 \end{bmatrix} \quad (2.49)$$

Although the gradient descent method requires many more iterations than Newton's method to converge, the computational cost in each iteration is much reduced, as no more matrix inversion is needed.

When it is difficult or too computationally costly to find the optimal step size along the search direction, some suboptimal step size may be acceptable, such as in the quasi-Newton methods (Section 2.1.5) for minimization problems. In this case, although the step size  $\delta$  is no longer required to be such that the function value at the new position  $f(\mathbf{x}_{n+1}) = f(\mathbf{x}_n + \delta \mathbf{d}_n)$  is minimized along the search direction  $\mathbf{d}_n$ , the step size  $\delta$  still has to satisfy the following *Wolfe conditions*:

- *Sufficient decrease (Armijo rule)*:

$$f(\mathbf{x}_{n+1}) = f(\mathbf{x}_n + \delta \mathbf{d}_n) \leq f(\mathbf{x}_n) + c_1 \delta \mathbf{d}_n^T \mathbf{g}_n \quad (2.50)$$

- *Curvature condition*:

$$\mathbf{g}_{n+1}^T \mathbf{d}_n \geq c_2 \mathbf{g}_n^T \mathbf{d}_n \quad (2.51)$$

As  $\mathbf{g}_n^T \mathbf{d}_n < 0$ , this condition can also be written in the following alternative form:

$$|\mathbf{g}_{n+1}^T \mathbf{d}_n| < |c_2 \mathbf{g}_n^T \mathbf{d}_n| \quad (2.52)$$

Here the two constants  $c_1$  and  $c_2$  above satisfy  $0 < c_1 < c_2 < 1$ .

In general, these conditions are motivated by the desired effect that after each iterative step, the function should have a shallower slope along  $\mathbf{d}_n$ , as well as a lower value, so that eventually the solution can be approached where  $f(\mathbf{x})$  is minimum and the gradient is zero.

Specifically, to understand the first condition above, we represent the function to be minimized as a single-variable function of the step size  $\phi(\delta) = f(\mathbf{x}_n + \delta \mathbf{d}_n)$ , and its tangent line at the point  $\delta = 0$  as a linear function  $L_0(\delta) = a + b\delta$ , where the intercept  $a$  can be found as  $a = L_0(0) = \phi(\delta)|_{\delta=0} = f(\mathbf{x}_n)$ , and the slope  $b$  can be found as the derivative of  $\phi(\delta) = f(\mathbf{x}_n + \delta \mathbf{d}_n)$  at  $\delta = 0$ :

$$b = \left. \frac{d}{d\delta} f(\mathbf{x}_n + \delta \mathbf{d}_n) \right|_{\delta=0} = \mathbf{g}_n^T \mathbf{d}_n < 0 \quad (2.53)$$

which is required to be negative for the function value to be reduced,  $f(\mathbf{x}_n + \delta \mathbf{d}_n) < f(\mathbf{x}_n)$ . Now the function of the tangent line can be written as

$$L_0(\delta) = a + b\delta = f(\mathbf{x}_n) + \mathbf{g}_n^T \mathbf{d}_n \delta \quad (2.54)$$

Comparing this with a constant line  $L_1(\delta) = f(\mathbf{x}_n)$  of slope zero, we see that any straight line between  $L_0(\delta)$  and  $L_1(\delta)$  can be described by  $L(\delta) = f(\mathbf{x}_n) + c_1 \mathbf{g}_n^T \mathbf{d}_n \delta$  with  $0 < c_1 < 1$ , with a slope  $0 < c_1 \mathbf{g}_n^T \mathbf{d}_n < \mathbf{g}_n^T \mathbf{d}_n$ . The Armijo rule is to find any  $\delta > 0$  that satisfies

$$f(\mathbf{x}_{n+1}) = f(\mathbf{x}_n + \delta \mathbf{d}_n) \leq f(\mathbf{x}_n) + c_1 \mathbf{g}_n^T \mathbf{d}_n \delta < f(\mathbf{x}_n) \quad (2.55)$$

We see that the value of  $f(\mathbf{x})$  is guaranteed to be reduced.

The second condition requires that at the new position  $\mathbf{x}_{n+1}$  the slope of the gradient  $\mathbf{g}_{n+1}$  along the search direction  $\mathbf{d}_n$  be sufficiently reduced to be less than a specified value (determined by  $c_2$ ), in comparison to the slope at the old position  $\mathbf{x}_n$ .

The reason why  $\mathbf{g}_{n+1} \perp \mathbf{g}_n$  can also be explained geometrically. As shown in the figure the gradient vectors at various points along the direction of  $-\mathbf{g}_n$  are shown by the blue arrows, and their projections onto the direction represent the slopes of the function  $f(\delta_n) = f(\mathbf{x}_n - \delta_n \mathbf{g}_n)$ . Obviously at  $\mathbf{x}_{n+1}$  where  $f(\delta_n)$  reaches its minimum, its slope is zero. In other words, the projection of the gradient  $\mathbf{g}_{n+1}$  onto the direction of  $-\mathbf{g}_n$  is zero, i.e.,  $\mathbf{g}_{n+1} \perp \mathbf{g}_n$  or  $\mathbf{g}_{n+1}^T \mathbf{g}_n = 0$ .

The gradient descent method gradually approaches a solution  $\mathbf{x}$  of an N-D minimization problem by moving from the initial guess  $\mathbf{x}_0$  along a zigzag path composed of a set of segments with any two consecutive segments perpendicular to each other. The number of steps depends greatly on the initial guess. As illustrated in the example below in an N=2 dimensional case, the best possible case is that the solution happens to be on the gradient direction of the initial

guess, which could be reached in a single step, while the worst possible case is that the gradient direction of the initial guess happens to be 45 degrees off from the gradient direction of the optimal case, and it takes many zigzag steps to go around the optimal path to reach the solution. Many of the steps are in the same direction as some of the previous steps.

To improve the performance of the gradient descent method we can include in the iteration a momentum term representing the search direction previously traversed:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \delta_n \mathbf{g}_n + \alpha_n (\mathbf{x}_n - \mathbf{x}_{n-1}) \quad (2.56)$$

Now two consecutive search directions are no longer perpendicular to each other and the resulting search path is smoother than the zigzag path without the momentum term. The parameter  $\alpha_n$  controls how much momentum is to be added.

Obviously it is most desirable not to repeat any of the previous directions traveled so that the solution can be reached in  $N$  steps, each in a unique direction in the  $N$ -D space. In other words, the subsequent steps are independent of each other, never interfering with the results achieved in the previous steps. Such a method will be discussed in the next section.

**Example 3:**

The Rosenbrock function

$$f(x_1, x_2) = (a - x_1)^2 + b(x_2 - x_1^2)^2 \quad (\text{e.g., } a = 1, b = 100)$$

is a two-variable non-convex function with a global minimum  $f(1, 1) = 0$  at the point  $(1, 1)$ , which is inside a long parabolic shaped valley as shown in the figure below. As the slope along the valley is very shallow, it is difficult for an algorithm to converge quickly to the minimum. For this reason, the Rosenbrock function is often used to test various minimization algorithms.

The figure below shows the search path of the gradient descent method based on the optimal step size given in Eq. (2.43). We see that the search path is composed a long sequence of 90 degree turns between consecutive segments.

When the Newton's method is applied to this minimization problem, it takes only four iterations for the algorithm to converge to the minimum, as shown in the figure below:

### 2.1.5 Quasi-Newton Methods

As we have seen above, Newton's methods can be used to solve both nonlinear systems to find roots of a set of simultaneous equations, and optimization problems to minimize a scalar-valued objective function based on the iterations of the same form. Specifically,

- Solving equations  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ :

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}(\mathbf{x}_n)^{-1} \mathbf{f}(\mathbf{x}_n) = \mathbf{x}_n - \mathbf{J}_n^{-1} \mathbf{f}_n \quad (2.57)$$

where  $\mathbf{J}_n = \mathbf{J}(\mathbf{x}_n)$  is the Jacobian matrix of  $\mathbf{f}(\mathbf{x}_n)$ ,

- Minimizing  $f(\mathbf{x})$ :

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{H}(\mathbf{x}_n)^{-1} \mathbf{g}(\mathbf{x}_n) = \mathbf{x}_n - \mathbf{H}_n^{-1} \mathbf{g}_n \quad (2.58)$$

where  $\mathbf{g}_n = \mathbf{g}(\mathbf{x}_n)$  and  $\mathbf{H}_n = \mathbf{H}(\mathbf{x}_n)$  are the gradient vector and Hessian matrix of  $f(\mathbf{x})$  at  $\mathbf{x}_n$ , respectively.

If the Jacobian  $\mathbf{J}(\mathbf{x})$  or the Hessian matrix  $\mathbf{H}(\mathbf{x})$  is not available, or if it is too computationally costly to calculate its inverse (with complexity  $O(N^3)$ ), the quasi-Newton methods can be used to approximate the Hessian matrix or its inverse based only on the first order derivative, the gradient  $\mathbf{g}$  of  $f(\mathbf{x})$  (with complexity  $O(N^2)$ ), similar to Broyden's method previously considered (Section 1.5).

In the following, we consider the minimization of a function  $f(\mathbf{x})$ . Its Taylor expansion around point  $\mathbf{x}_{n+1}$  is

$$f(\mathbf{x}) = f(\mathbf{x}_{n+1}) + (\mathbf{x} - \mathbf{x}_{n+1})^T \mathbf{g}_{n+1} + \frac{1}{2} (\mathbf{x} - \mathbf{x}_{n+1})^T \mathbf{H}_{n+1} (\mathbf{x} - \mathbf{x}_{n+1}) + O(\|\mathbf{x} - \mathbf{x}_{n+1}\|^3) \quad (2.59)$$

Taking derivative with respect to  $\mathbf{x}$ , we get

$$\frac{d}{d\mathbf{x}} f(\mathbf{x}) = \mathbf{g}(\mathbf{x}) = \mathbf{g}_{n+1} + \mathbf{H}_{n+1} (\mathbf{x} - \mathbf{x}_{n+1}) + O(\|\mathbf{x} - \mathbf{x}_{n+1}\|^2) \quad (2.60)$$

Evaluating at  $\mathbf{x} = \mathbf{x}_n$ , we have  $\mathbf{g}(\mathbf{x}_n) = \mathbf{g}_n$ , and the above can be written as

$$\begin{aligned} \mathbf{g}_{n+1} - \mathbf{g}_n &= \mathbf{H}_{n+1} (\mathbf{x}_{n+1} - \mathbf{x}_n) + O(\|\mathbf{x}_{n+1} - \mathbf{x}_n\|^2) \\ &= \mathbf{B}_{n+1} (\mathbf{x}_{n+1} - \mathbf{x}_n) \end{aligned} \quad (2.61)$$

where matrix  $\mathbf{B}_n$  is the secant approximation of the Hessian matrix  $\mathbf{H}_n$ , and the last equality is called the *secant equation*. For convenience, we further define:

$$\mathbf{s}_n = \mathbf{x}_{n+1} - \mathbf{x}_n, \quad \mathbf{y}_n = \mathbf{g}_{n+1} - \mathbf{g}_n \quad (2.62)$$

so that the equation above can be written as

$$\mathbf{B}_{n+1} \mathbf{s}_n = \mathbf{y}_n, \quad \text{or} \quad \mathbf{B}_{n+1}^{-1} \mathbf{y}_n = \mathbf{s}_n \quad (2.63)$$

This is the *quasi-Newton equation*, which is the *secant condition* that must be satisfied by matrix  $\mathbf{B}_n$ , or its inverse  $\mathbf{B}_n^{-1}$ , in any of the quasi-Newton algorithms, all taking the follow general steps:

1. Initialize  $\mathbf{x}_0$  and  $\mathbf{B}_0$ , set  $n = 0$ ;
2. Compute gradient  $\mathbf{g}_n$  and the search direction  $\mathbf{d}_n = -\mathbf{B}_n^{-1} \mathbf{g}_n$ ;
3. Get  $\mathbf{x}_{n+1} = \mathbf{x}_n + \delta \mathbf{d}_n$  with step size  $\delta$  satisfying the Wolfe conditions
4. update  $\mathbf{B}_{n+1} = \mathbf{B}_n + \Delta \mathbf{B}_n$  or  $\mathbf{B}_{n+1}^{-1} = \mathbf{B}_n^{-1} + \Delta \mathbf{B}_n^{-1}$ , that satisfies the quasi-Newton equation;
5. If termination condition is not satisfied,  $n = n + 1$ , go back to second step.

For this iteration to converge to a local minimum of  $f(\mathbf{x})$ ,  $\mathbf{B}_n$  must be positive definite matrix, same as the Hessian matrix  $\mathbf{H}$  it approximates. Specially, if  $\mathbf{B}_n = \mathbf{I}$ , then  $\mathbf{d}_n = -\mathbf{g}_n$ , the algorithm becomes the gradient descent method; also, if  $\mathbf{B}_n = \mathbf{H}_n$  is the Hessian matrix, then the algorithm becomes the Newton's method.

In a quasi-Newton method, we can choose to update either  $\mathbf{B}_n$  or its inverse  $\mathbf{B}_n^{-1}$  based on one of the two forms of the quasi-Newton equation. We note that there is a dual relationship between the two ways to update, i.e., by swapping  $\mathbf{y}_n$  and  $\mathbf{s}_n$ , an update formula for  $\mathbf{B}_n$  can be directly applied to its inverse and vice versa.

- The *Symmetric Rank 1 (SR1)* Algorithm

Here matrix  $\mathbf{B}_n$  is updated by an additional term:

$$\mathbf{B}_{n+1} = \mathbf{B}_n + \mathbf{u}\mathbf{u}^T \quad (2.64)$$

where  $\mathbf{u}$  a vector and  $\mathbf{u}\mathbf{u}^T$  is a matrix of rank 1. Imposing the secant condition, we get

$$\mathbf{B}_{n+1}\mathbf{s}_n = \mathbf{B}_n\mathbf{s}_n + \mathbf{u}\mathbf{u}^T\mathbf{s}_n = \mathbf{y}_n, \quad \text{i.e.,} \quad \mathbf{u}(\mathbf{u}^T\mathbf{s}_n) = \mathbf{y}_n - \mathbf{B}_n\mathbf{s}_n \quad (2.65)$$

This equation indicates  $\mathbf{u}$  is in the same direction as  $\mathbf{w} = \mathbf{y}_n - \mathbf{B}_n\mathbf{s}_n$ , and can therefore be written as  $\mathbf{u} = c\mathbf{w}$ . Substituting this back we get

$$\mathbf{u}\mathbf{u}^T\mathbf{s}_n = c^2\mathbf{w}\mathbf{w}^T\mathbf{s}_n = \mathbf{w} \quad (2.66)$$

Solving this we get  $c = 1/(\mathbf{w}^T\mathbf{s}_n)^{1/2}$ ,

$$\mathbf{u} = c\mathbf{w} = \frac{\mathbf{y}_n - \mathbf{B}_n\mathbf{s}_n}{(\mathbf{w}^T\mathbf{s}_n)^{1/2}} = \frac{\mathbf{y}_n - \mathbf{B}_n\mathbf{s}_n}{((\mathbf{y}_n - \mathbf{B}_n\mathbf{s}_n)^T\mathbf{s}_n)^{1/2}} \quad (2.67)$$

and the iteration of  $\mathbf{B}_n$  becomes

$$\mathbf{B}_{n+1} = \mathbf{B}_n + \mathbf{u}\mathbf{u}^T = \mathbf{B}_n + \frac{(\mathbf{y}_n - \mathbf{B}_n\mathbf{s}_n)(\mathbf{y}_n - \mathbf{B}_n\mathbf{s}_n)^T}{(\mathbf{y}_n - \mathbf{B}_n\mathbf{s}_n)^T\mathbf{s}_n} \quad (2.68)$$

Applying the Sherman-Morrison formula (Section A.6) we further get the inverse  $\mathbf{B}_{n+1}^{-1}$ :

$$\begin{aligned} \mathbf{B}_{n+1}^{-1} &= (\mathbf{B}_n + \mathbf{u}\mathbf{u}^T)^{-1} = \mathbf{B}_n^{-1} - \frac{\mathbf{B}_n^{-1}\mathbf{u}\mathbf{u}^T\mathbf{B}_n^{-1}}{1 + \mathbf{u}^T\mathbf{B}_n^{-1}\mathbf{u}} \\ &= \mathbf{B}_n^{-1} + \frac{(\mathbf{s}_n - \mathbf{B}_n^{-1}\mathbf{y}_n)(\mathbf{s}_n - \mathbf{B}_n^{-1}\mathbf{y}_n)^T}{(\mathbf{s}_n - \mathbf{B}_n^{-1}\mathbf{y}_n)^T\mathbf{y}_n} \end{aligned} \quad (2.69)$$

We note that this equation is dual to the previous one, and it can also be obtained based on the duality between the secant conditions for  $\mathbf{B}_n$  and  $\mathbf{B}_n^{-1}$ . Specifically, same as above, we impose the secant condition for the inverse  $\mathbf{B}_{n+1}^{-1}$  of the following form

$$\mathbf{B}_{n+1}^{-1} = \mathbf{B}_n^{-1} + \mathbf{u}\mathbf{u}^T \quad (2.70)$$

and get

$$\mathbf{B}_{n+1}^{-1}\mathbf{y}_n = \mathbf{B}_n^{-1}\mathbf{y}_n + \mathbf{u}\mathbf{u}^T\mathbf{y}_n = \mathbf{s}_n, \quad (2.71)$$

Then by repeating the same process above, we get the same update formula for  $\mathbf{B}_n^{-1}$ .

This is the formula for directly updating  $\mathbf{B}_n^{-1}$ . For the iteration to converge to a local minimum, matrix  $\mathbf{B}_{n+1}^{-1}$  needs to be positive definite as well as  $\mathbf{B}_n^{-1}$ , we therefore require

$$\mathbf{w}^T\mathbf{y}_n = (\mathbf{s}_n - \mathbf{B}_n^{-1}\mathbf{y}_n)^T\mathbf{y}_n = \mathbf{s}_n^T\mathbf{y}_n - \mathbf{y}_n^T\mathbf{B}_n^{-1}\mathbf{y}_n > 0 \quad (2.72)$$

It may be difficult to maintain this requirement through out the iteration. This problem can be avoided in the following DFP and BFGS methods.

- The *BFGS* (*Broyden-Fletcher-Goldfarb-Shanno*) algorithm

This is a rank 2 algorithm in which matrix  $\mathbf{B}_n$  is updated by two rank-1 terms:

$$\mathbf{B}_{n+1} = \mathbf{B}_n + \alpha\mathbf{u}\mathbf{u}^T + \beta\mathbf{v}\mathbf{v}^T \quad (2.73)$$

Imposing the secant condition for  $\mathbf{B}_n$ , we get

$$\mathbf{B}_{n+1}\mathbf{s}_n = (\mathbf{B}_n + \alpha\mathbf{u}\mathbf{u}^T + \beta\mathbf{v}\mathbf{v}^T)\mathbf{s}_n = \mathbf{B}_n\mathbf{s}_n + \mathbf{u}(\alpha\mathbf{u}^T\mathbf{s}_n) + \mathbf{v}(\beta\mathbf{v}^T\mathbf{s}_n) = \mathbf{y}_n \quad (2.74)$$

or

$$\mathbf{u}(\alpha\mathbf{u}^T\mathbf{s}_n) + \mathbf{v}(\beta\mathbf{v}^T\mathbf{s}_n) = \mathbf{y}_n - \mathbf{B}_n\mathbf{s}_n \quad (2.75)$$

which can be satisfied if we let

$$\mathbf{u} = \mathbf{y}_n, \quad \alpha = \frac{1}{\mathbf{s}_n^T\mathbf{y}_n}, \quad \mathbf{v} = \mathbf{B}_n\mathbf{s}_n, \quad \beta = -\frac{1}{\mathbf{v}^T\mathbf{s}_n} = -\frac{1}{\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n} \quad (2.76)$$

Substituting these into  $\mathbf{B}_{n+1} = \mathbf{B}_n + \alpha\mathbf{u}\mathbf{u}^T + \beta\mathbf{v}\mathbf{v}^T$  we get

$$\mathbf{B}_{n+1} = \mathbf{B}_n + \alpha\mathbf{u}\mathbf{u}^T + \beta\mathbf{v}\mathbf{v}^T = \mathbf{B}_n + \frac{\mathbf{y}_n\mathbf{y}_n^T}{\mathbf{y}_n^T\mathbf{s}_n} - \frac{\mathbf{B}_n\mathbf{s}_n\mathbf{s}_n^T\mathbf{B}_n}{\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n} \quad (2.77)$$

Given this update formula for  $\mathbf{B}_n$ , we can further find its inverse  $\mathbf{B}_n^{-1}$  and then the search direction  $\mathbf{d}_n = -\mathbf{B}_n^{-1}\mathbf{g}_n$ .

Alternatively, given  $\mathbf{B}_n$ , we can further derive an update formula directly for the inverse matrix  $\mathbf{B}_n^{-1}$ , so that the search direction  $\mathbf{d}_n$  can be obtained without carrying out the inversion computation. Specifically, we first define  $\mathbf{U} = [\mathbf{u}_1 \ \mathbf{u}_2]$  and  $\mathbf{V} = [\mathbf{v}_1 \ \mathbf{v}_2]$ , where

$$\mathbf{u}_1 = \mathbf{v}_1 = \frac{\mathbf{y}_n}{(\mathbf{s}_n^T\mathbf{y}_n)^{1/2}}, \quad \mathbf{u}_2 = -\mathbf{v}_2 = \frac{\mathbf{B}_n\mathbf{s}_n}{(\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n)^{1/2}} \quad (2.78)$$

so that the expression above can be written as:

$$\mathbf{B}_{n+1} = \mathbf{B}_n + \mathbf{u}_1\mathbf{v}_1^T + \mathbf{u}_2\mathbf{v}_2^T = (\mathbf{B}_n + \mathbf{U}\mathbf{V}^T)^{-1} \quad (2.79)$$

and then apply the Sherman-Morrison formula to get:

$$\begin{aligned}\mathbf{B}_{n+1}^{-1} &= (\mathbf{B}_n + \mathbf{U}\mathbf{V}^T)^{-1} = \mathbf{B}_n^{-1} - \mathbf{B}_n^{-1}\mathbf{U}(\mathbf{I} + \mathbf{V}^T\mathbf{B}_n^{-1}\mathbf{U})^{-1}\mathbf{V}^T\mathbf{B}_n^{-1} \\ &= \mathbf{B}_n^{-1} - \mathbf{B}_n^{-1}\mathbf{U}\mathbf{C}^{-1}\mathbf{V}^T\mathbf{B}_n^{-1} \\ &= \mathbf{B}_n^{-1} - \mathbf{B}_n^{-1}[\mathbf{u}_1 \ \mathbf{u}_2]\mathbf{C}^{-1}(\mathbf{B}_n^{-1}[\mathbf{v}_1 \ \mathbf{v}_2])^T\end{aligned}\quad (2.80)$$

where we have defined

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \mathbf{I} + \mathbf{V}^T\mathbf{B}_n^{-1}\mathbf{U} = \mathbf{I} + [\mathbf{v}_1 \ \mathbf{v}_2]^T\mathbf{B}_n^{-1}[\mathbf{u}_1 \ \mathbf{u}_2] \quad (2.81)$$

with

$$c_{11} = 1 + \mathbf{v}_1^T\mathbf{B}_n^{-1}\mathbf{u}_1 = 1 + \frac{\mathbf{y}_n^T\mathbf{B}_n^{-1}\mathbf{y}_n}{\mathbf{s}_n^T\mathbf{y}_n} \quad (2.82)$$

$$c_{22} = 1 + \mathbf{v}_2^T\mathbf{B}_n^{-1}\mathbf{u}_2 = 1 - \frac{\mathbf{s}_n^T\mathbf{B}_n\mathbf{B}_n^{-1}\mathbf{B}_n\mathbf{s}_n}{\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n} = 0 \quad (2.83)$$

$$c_{12} = \mathbf{v}_1^T\mathbf{B}_n^{-1}\mathbf{u}_2 = \frac{\mathbf{y}_n^T\mathbf{B}_n^{-1}\mathbf{B}_n\mathbf{s}_n}{(\mathbf{s}_n^T\mathbf{y}_n)^{1/2}(\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n)^{1/2}} = \frac{(\mathbf{s}_n^T\mathbf{y}_n)^{1/2}}{(\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n)^{1/2}} \quad (2.84)$$

$$c_{21} = \mathbf{v}_2^T\mathbf{B}_n^{-1}\mathbf{u}_1 = -c_{12} \quad (2.85)$$

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} \\ -c_{12} & 0 \end{bmatrix}, \quad \mathbf{C}^{-1} = \begin{bmatrix} 0 & -1/c_{12} \\ 1/c_{12} & c_{11}/c_{12}^2 \end{bmatrix} \quad (2.86)$$

Substituting this  $\mathbf{C}^{-1}$  into the expression for  $\mathbf{B}_{n+1}^{-1}$ , we get:

$$\begin{aligned}\mathbf{B}_{n+1}^{-1} &= \mathbf{B}_n^{-1} - [\mathbf{B}_n^{-1}\mathbf{u}_1 \ \mathbf{B}_n^{-1}\mathbf{u}_2] \begin{bmatrix} 0 & -1/c_{12} \\ 1/c_{12} & c_{11}/c_{12}^2 \end{bmatrix} [\mathbf{B}_n^{-1}\mathbf{v}_1 \ \mathbf{B}_n^{-1}\mathbf{v}_2]^T \\ &= \mathbf{B}_n^{-1} - \frac{1}{c_{12}}[\mathbf{B}_n^{-1}\mathbf{u}_2\mathbf{v}_1^T\mathbf{B}_n^{-1} - \mathbf{B}_n^{-1}\mathbf{u}_1\mathbf{v}_2^T\mathbf{B}_n^{-1}] - \frac{c_{11}}{c_{12}^2}\mathbf{B}_n^{-1}\mathbf{u}_2\mathbf{v}_2^T\mathbf{B}_n^{-1}\end{aligned}\quad (2.87)$$

where

$$\frac{1}{c_{12}}\mathbf{B}_n^{-1}\mathbf{u}_2\mathbf{v}_1^T\mathbf{B}_n^{-1} = \frac{(\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n)^{1/2}}{(\mathbf{s}_n^T\mathbf{y}_n)^{1/2}}\mathbf{B}_n^{-1}\frac{\mathbf{B}_n\mathbf{s}_n}{(\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n)^{1/2}}\frac{\mathbf{y}_n^T}{(\mathbf{s}_n^T\mathbf{y}_n)^{1/2}}\mathbf{B}_n^{-1} = \frac{\mathbf{s}_n\mathbf{y}_n^T\mathbf{B}_n^{-1}}{\mathbf{s}_n^T\mathbf{y}_n} \quad (2.88)$$

$$-\frac{1}{c_{12}}\mathbf{B}_n^{-1}\mathbf{u}_1\mathbf{v}_2^T\mathbf{B}_n^{-1} = \frac{(\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n)^{1/2}}{(\mathbf{s}_n^T\mathbf{y}_n)^{1/2}}\mathbf{B}_n^{-1}\frac{\mathbf{y}_n}{(\mathbf{s}_n^T\mathbf{y}_n)^{1/2}}\frac{\mathbf{s}_n^T\mathbf{B}_n}{(\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n)^{1/2}}\mathbf{B}_n^{-1} = \frac{\mathbf{B}_n^{-1}\mathbf{y}_n\mathbf{s}_n^T}{\mathbf{s}_n^T\mathbf{y}_n} \quad (2.89)$$

$$\begin{aligned}\frac{c_{11}}{c_{12}^2}\mathbf{B}_n^{-1}\mathbf{u}_2\mathbf{v}_2^T\mathbf{B}_n^{-1} &= -\left(1 + \frac{\mathbf{y}_n^T\mathbf{B}_n^{-1}\mathbf{y}_n}{\mathbf{s}_n^T\mathbf{y}_n}\right)\frac{\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n}{\mathbf{s}_n^T\mathbf{y}_n}\mathbf{B}_n^{-1}\frac{\mathbf{B}_n\mathbf{s}_n}{(\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n)^{1/2}}\frac{\mathbf{s}_n^T\mathbf{B}_n}{(\mathbf{s}_n^T\mathbf{B}_n\mathbf{s}_n)^{1/2}}\mathbf{B}_n^{-1} \\ &= -\left(1 + \frac{\mathbf{y}_n^T\mathbf{B}_n^{-1}\mathbf{y}_n}{\mathbf{s}_n^T\mathbf{y}_n}\right)\frac{\mathbf{s}_n\mathbf{s}_n^T}{\mathbf{s}_n^T\mathbf{y}_n}\end{aligned}\quad (2.90)$$

Substituting these three terms back into Eq. (2.87), we get the update formula for  $\mathbf{B}_n^{-1}$ :

$$\mathbf{B}_{n+1}^{-1} = \mathbf{B}_n^{-1} - \frac{\mathbf{B}_n^{-1} \mathbf{y}_n \mathbf{s}_n^T + \mathbf{s}_n \mathbf{y}_n^T \mathbf{B}_n^{-1}}{\mathbf{s}_n^T \mathbf{y}_n} + \left(1 + \frac{\mathbf{y}_n^T \mathbf{B}_n^{-1} \mathbf{y}_n}{\mathbf{s}_n^T \mathbf{y}_n}\right) \frac{\mathbf{s}_n \mathbf{s}_n^T}{\mathbf{s}_n^T \mathbf{y}_n} \quad (2.91)$$

In summary,  $\mathbf{B}_{n+1}^{-1}$  can be found by either Eq. (2.77) for  $\mathbf{B}_{n+1}$  followed by an additional inversion operation, or Eq. (2.91) directly for  $\mathbf{B}_{n+1}^{-1}$ . The results obtained by these methods are equivalent.

- The *DFP (Davidon-Fletcher-Powell)* Algorithm

Same as the BFGS algorithm, the DFP algorithm is also a rank 2 algorithm, where instead of  $\mathbf{B}_n$ , the inverse matrix  $\mathbf{B}_n^{-1}$  is updated by two rank-1 terms:

$$\mathbf{B}_{n+1}^{-1} = \mathbf{B}_n^{-1} + \alpha \mathbf{u} \mathbf{u}^T + \beta \mathbf{v} \mathbf{v}^T \quad (2.92)$$

where  $\alpha$  and  $\beta$  are real scalars,  $\mathbf{u}$  and  $\mathbf{v}$  are vectors. Imposing the secant condition for the inverse of  $\mathbf{B}_n$ , we get

$$\mathbf{B}_{n+1}^{-1} \mathbf{y}_n = (\mathbf{B}_n^{-1} + \alpha \mathbf{u} \mathbf{u}^T + \beta \mathbf{v} \mathbf{v}^T) \mathbf{y}_n = \mathbf{B}_n^{-1} \mathbf{y}_n + \mathbf{u} (\alpha \mathbf{u}^T \mathbf{y}_n) + \mathbf{v} (\beta \mathbf{v}^T \mathbf{y}_n) = \mathbf{s}_n \quad (2.93)$$

or

$$\mathbf{u} (\alpha \mathbf{u}^T \mathbf{y}_n) + \mathbf{v} (\beta \mathbf{v}^T \mathbf{y}_n) = \mathbf{s}_n - \mathbf{B}_n^{-1} \mathbf{y}_n \quad (2.94)$$

which can be satisfied if we let

$$\mathbf{u} = \mathbf{s}_n, \quad \alpha = \frac{1}{\mathbf{s}_n^T \mathbf{y}_n}, \quad \mathbf{v} = \mathbf{B}_n^{-1} \mathbf{y}_n, \quad \beta = -\frac{1}{\mathbf{v}^T \mathbf{y}_n} = -\frac{1}{\mathbf{y}_n^T \mathbf{B}_n^{-1} \mathbf{y}_n} \quad (2.95)$$

Substituting these into  $\mathbf{B}_{n+1}^{-1} = \mathbf{B}_n^{-1} + \alpha \mathbf{u} \mathbf{u}^T + \beta \mathbf{v} \mathbf{v}^T$  we get

$$\mathbf{B}_{n+1}^{-1} = \mathbf{B}_n^{-1} + \alpha \mathbf{u} \mathbf{u}^T + \beta \mathbf{v} \mathbf{v}^T = \mathbf{B}_n^{-1} + \frac{\mathbf{s}_n \mathbf{s}_n^T}{\mathbf{s}_n^T \mathbf{y}_n} - \frac{\mathbf{B}_n^{-1} \mathbf{y}_n \mathbf{y}_n^T \mathbf{B}_n^{-1}}{\mathbf{y}_n^T \mathbf{B}_n^{-1} \mathbf{y}_n} \quad (2.96)$$

Following the same procedure used for the BFGS method, we can also obtain the update formular of matrix  $\mathbf{B}_n$  for the DFP method:

$$\mathbf{B}_{n+1} = \mathbf{B}_n - \frac{\mathbf{B}_n \mathbf{s}_n \mathbf{y}_n^T + \mathbf{y}_n \mathbf{s}_n^T \mathbf{B}_n}{\mathbf{y}_n^T \mathbf{s}_n} + \left(1 + \frac{\mathbf{s}_n^T \mathbf{B}_n \mathbf{s}_n}{\mathbf{y}_n^T \mathbf{s}_n}\right) \frac{\mathbf{y}_n \mathbf{y}_n^T}{\mathbf{y}_n^T \mathbf{s}_n} \quad (2.97)$$

We note that Eqs. (2.96) and (2.77) form a duality pair, and Eqs. (2.97) and (2.91) form another duality pair. In other words, the BFGS and FDP methods are dual of each other.

In both the DFP and BFGS methods, matrix  $\mathbf{B}_n^{-1}$ , as well as  $\mathbf{B}_n$ , must be positive definite, i.e.,  $\mathbf{z}^T \mathbf{B}_n \mathbf{z} > 0$  must hold for any  $\mathbf{z} \neq \mathbf{0}$ . We now prove that this requirement is satisfied if the curvature condition of the Wolfe conditions is satisfied:

$$\mathbf{g}_{n+1}^T \mathbf{d}_n \geq c_2 \mathbf{g}_n^T \mathbf{d}_n \quad \text{i.e.} \quad (\mathbf{g}_{n+1} - c_2 \mathbf{g}_n)^T \mathbf{d}_n \geq 0 \quad (2.98)$$



Replacing the search direction  $\mathbf{d}_n$  by  $\delta\mathbf{d}_n = \mathbf{x}_{n+1} - \mathbf{x}_n = \mathbf{s}_n$ , we get:

$$(\mathbf{g}_{n+1} - c_2\mathbf{g}_n)^T \mathbf{s}_n = \mathbf{g}_{n+1}^T \mathbf{s}_n - c_2\mathbf{g}_n^T \mathbf{s}_n \geq 0 \quad (2.99)$$

We also have

$$\mathbf{y}_n^T \mathbf{s}_n = (\mathbf{g}_{n+1} - \mathbf{g}_n)^T \mathbf{s}_n = \mathbf{g}_{n+1}^T \mathbf{s}_n - \mathbf{g}_n^T \mathbf{s}_n \quad (2.100)$$

Subtracting the first equation from the second, we get

$$\mathbf{y}_n^T \mathbf{s}_n - (\mathbf{g}_{n+1}^T \mathbf{s}_n - c_2\mathbf{g}_n^T \mathbf{s}_n) = (c_2 - 1)\mathbf{g}_n^T \mathbf{s}_n > 0 \quad (2.101)$$

The last inequality is due to the fact that  $\mathbf{g}_n^T \mathbf{s}_n < 0$  and  $c_2 < 1$ . We therefore have

$$\mathbf{y}_n^T \mathbf{s}_n \geq \mathbf{g}_{n+1}^T \mathbf{s}_n - c_2\mathbf{g}_n^T \mathbf{s}_n \geq 0 \quad (2.102)$$

Given  $\mathbf{y}_n^T \mathbf{s}_n > 0$ , we can further prove by induction that both  $\mathbf{B}_{n+1}$  and  $\mathbf{B}_{n+1}^{-1}$  based respectively on the update formulae in Eqs. (2.77) and (2.96) are positive definite. We first assume  $\mathbf{B}_0$  and  $\mathbf{B}_n$  are both positive definite, and express  $\mathbf{B}_n$  in terms of its Cholesky decomposition,  $\mathbf{B}_n = \mathbf{L}\mathbf{L}^T$ . We further define  $\mathbf{a} = \mathbf{L}^T \mathbf{z}$ ,  $\mathbf{b} = \mathbf{L}^T \mathbf{s}$ , and get

$$\mathbf{a}^T \mathbf{a} = \mathbf{z}^T \mathbf{B}_n \mathbf{z}, \quad \mathbf{b}^T \mathbf{b} = \mathbf{s}^T \mathbf{B}_n \mathbf{s}, \quad \mathbf{a}^T \mathbf{b} = \mathbf{z}^T \mathbf{B}_n \mathbf{s} \quad (2.103)$$

Now we can show that the sum of the first and third terms of Eq. (2.91) is positive definite:

$$\mathbf{z}^T \left[ \mathbf{B}_n - \frac{\mathbf{B}_n \mathbf{s}_n \mathbf{s}_n^T \mathbf{B}_n}{\mathbf{s}_n^T \mathbf{B}_n \mathbf{s}_n} \right] \mathbf{z} = \mathbf{z}^T \mathbf{B}_n \mathbf{z} - \frac{(\mathbf{z}^T \mathbf{B}_n \mathbf{s}_n)^2}{\mathbf{s}_n^T \mathbf{B}_n \mathbf{s}_n} = \mathbf{a}^T \mathbf{a} - \frac{(\mathbf{a}^T \mathbf{b})^2}{\mathbf{b}^T \mathbf{b}} \geq 0 \quad (2.104)$$

The last step is due to the Cauchy-Schwarz inequality. Also, as  $\mathbf{s}_n^T \mathbf{y}_n \geq 0$ , the second term of Eq. (2.77) is positive definite

$$\mathbf{z}^T \left( \frac{\mathbf{s}_n \mathbf{s}_n^T}{\mathbf{s}_n^T \mathbf{y}_n} \right) \mathbf{z} \geq 0 \quad (2.105)$$

Combining these two results, we get

$$\mathbf{z}^T \mathbf{B}_{n+1} \mathbf{z} = \mathbf{z}^T \left( \mathbf{B}_n - \frac{\mathbf{B}_n \mathbf{s}_n \mathbf{s}_n^T \mathbf{B}_n}{\mathbf{s}_n^T \mathbf{B}_n \mathbf{s}_n} \right) \mathbf{z} + \mathbf{z}^T \left( \frac{\mathbf{y}_n \mathbf{y}_n^T}{\mathbf{s}_n^T \mathbf{y}_n} \right) \mathbf{z} \geq 0 \quad (2.106)$$

i.e.,  $\mathbf{B}_{n+1}$  based on the update formula Eq. (2.77) is positive definite. Following the same steps, we can also show that  $\mathbf{B}_{n+1}^{-1}$  based on Eq. (2.96) is positive definite.

### Examples:

The figures below shows the search path of the DFP and BPGS methods when applied to find the minimum of the Rosenbrock function.

### 2.1.6 Conjugate gradient method

The gradient descent method can be used to solve the minimization problem when the Hessian matrix of the objective function is not available. However, this method may be inefficient if it gets into a zigzag search pattern and repeat the same search directions many times. This problem can be avoided by the *conjugate gradient (CG)* method. If the objective function is quadratic, the CG method converges to the solution in  $N$  iterations without repeating any of the directions previously traversed. If the objective function is not quadratic, the CG method can still significantly improve the performance in comparison to the gradient descent method.

Again consider the approximation of the function  $f(\mathbf{x})$  to be minimized by the first three terms of its Taylor series:

$$f(\mathbf{x}) = f(\mathbf{x}_0 + \delta\mathbf{x}) \approx f(\mathbf{x}_0) + \mathbf{g}_0^T \delta\mathbf{x} + \frac{1}{2} \delta\mathbf{x}^T \mathbf{H}_0 \delta\mathbf{x} \quad (2.107)$$

If the Hessian matrix is positive definite, then  $f(\mathbf{x})$  has a minimum. If function  $f(\mathbf{x})$  is quadratic, the approximation above becomes exact and the function can be written as

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{b}^T \mathbf{x} + c \quad (2.108)$$

where  $\mathbf{A} = \mathbf{H}$  is the symmetric Hessian matrix, and its gradient and Hessian can be written as the following respectively:

$$\mathbf{g}(\mathbf{x}) = \frac{d}{d\mathbf{x}} f(\mathbf{x}) = \frac{d}{d\mathbf{x}} \left( \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{b}^T \mathbf{x} + c \right) = \mathbf{A} \mathbf{x} - \mathbf{b} \quad (2.109)$$

and

$$\mathbf{H}(\mathbf{x}) = \frac{d^2}{d\mathbf{x}^2} f(\mathbf{x}) = \frac{d}{d\mathbf{x}} \mathbf{g} = \frac{d}{d\mathbf{x}} (\mathbf{A} \mathbf{x} - \mathbf{b}) = \mathbf{A} \quad (2.110)$$

Solving  $\mathbf{g}(\mathbf{x}) = \mathbf{A} \mathbf{x} - \mathbf{b} = \mathbf{0}$ , we get the solution  $\mathbf{x}^* = \mathbf{A}^{-1} \mathbf{b}$ , at which the function is minimized to

$$f(\mathbf{x}^*) = \frac{1}{2} (\mathbf{A}^{-1} \mathbf{b})^T \mathbf{A} (\mathbf{A}^{-1} \mathbf{b}) - \mathbf{b}^T (\mathbf{A}^{-1} \mathbf{b}) + c = -\frac{1}{2} \mathbf{b}^T \mathbf{A}^{-1} \mathbf{b} + c \quad (2.111)$$

At the solution  $\mathbf{x}^*$  that minimizes  $f(\mathbf{x})$ , its gradient is

$$\mathbf{g}(\mathbf{x}^*) = \mathbf{A} \mathbf{x}^* - \mathbf{b} = \mathbf{0} \quad (2.112)$$

We also see that the minimization of the quadratic function  $f(\mathbf{x})$  is equivalent to solving a linear equation  $\mathbf{A} \mathbf{x} = \mathbf{b}$  with a symmetric positive definite coefficient matrix  $\mathbf{A}$ . The CG method considered here can therefore be used for solving both problems.

#### Conjugate basis vectors

We first review the concept of *conjugate vectors* (Section A.2) which is of essential importance in the CG method. Two vectors  $\mathbf{u}$  and  $\mathbf{v}$  are *mutually*

*conjugate* or *A-conjugate* to each other with respect to a symmetric matrix  $\mathbf{A} = \mathbf{A}^T$ , if they satisfy:

$$\mathbf{u}^T \mathbf{A} \mathbf{v} = (\mathbf{A} \mathbf{v})^T \mathbf{u} = \mathbf{v}^T \mathbf{A} \mathbf{u} = 0 \quad (2.113)$$

Specially, if  $\mathbf{A} = \mathbf{I}$ , the two conjugate vectors become orthogonal to each other, i.e.,  $\mathbf{v}^T \mathbf{u} = 0$ .

Similar to a set of  $N$  orthogonal vectors that can be used as the basis spanning an  $N$ -D space, a set of  $N$  mutually conjugate vectors  $\{\mathbf{d}_0, \dots, \mathbf{d}_{N-1}\}$  satisfying  $\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0$  ( $i \neq j$ ) can also be used as a basis to span the  $N$ -D space. Any vector in the space can be expressed as a linear combination of these basis vectors.

Also we note that any set of  $N$  independent vectors can be converted by the Gram-Schmidt process (Section A.2) to a set of  $N$  basis vectors that are either orthogonal or  $A$ -conjugate to each other.

**Example:**

Given two independent basis vectors of the 2-D space, and a positive-definite matrix:

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}$$

we can construct two  $A$ -orthogonal basis vectors by the Gram-Schmidt method:

$$\mathbf{u}_1 = \mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{u}_2 = \mathbf{v}_2 - \frac{\mathbf{u}_1^T \mathbf{A} \mathbf{v}_2}{\mathbf{u}_1^T \mathbf{A} \mathbf{u}_1} \mathbf{u}_1 = \begin{bmatrix} -1/3 \\ 1 \end{bmatrix}$$

The projections of a vector  $\mathbf{x} = [2, 3]^T$  onto  $\mathbf{v}_1$  and  $\mathbf{v}_2$  are:

$$\mathbf{p}_{\mathbf{v}_1}(\mathbf{x}) = \frac{\mathbf{v}_1^T \mathbf{x}}{\mathbf{v}_1^T \mathbf{v}_1} \mathbf{v}_1 = 2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \quad \mathbf{p}_{\mathbf{v}_2}(\mathbf{x}) = \frac{\mathbf{v}_2^T \mathbf{x}}{\mathbf{v}_2^T \mathbf{v}_2} \mathbf{v}_2 = 3 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}$$

The  $A$ -projections of the same vector  $\mathbf{x}$  onto  $\mathbf{u}_1$  and  $\mathbf{u}_2$  are:

$$\mathbf{p}_{\mathbf{u}_1}(\mathbf{x}) = \frac{\mathbf{u}_1^T \mathbf{A} \mathbf{x}}{\mathbf{u}_1^T \mathbf{A} \mathbf{u}_1} \mathbf{u}_1 = 3 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}, \quad \mathbf{p}_{\mathbf{u}_2}(\mathbf{x}) = \frac{\mathbf{u}_2^T \mathbf{A} \mathbf{x}}{\mathbf{u}_2^T \mathbf{A} \mathbf{u}_2} \mathbf{u}_2 = 3 \begin{bmatrix} -1/3 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$$

The original vector  $\mathbf{x}$  can be represented in either of the two bases:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} 2 \\ 3 \end{bmatrix} = 2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 3 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 2\mathbf{v}_1 + 3\mathbf{v}_2 = \mathbf{p}_{\mathbf{v}_1}(\mathbf{x}) + \mathbf{p}_{\mathbf{v}_2}(\mathbf{x}) \\ &= 3 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 3 \begin{bmatrix} -1/3 \\ 1 \end{bmatrix} = 3\mathbf{u}_1 + 3\mathbf{u}_2 = \mathbf{p}_{\mathbf{u}_1}(\mathbf{x}) + \mathbf{p}_{\mathbf{u}_2}(\mathbf{x}) \end{aligned}$$

**Search along a conjugate basis**

Similar to the gradient descent method, which iteratively improves the estimated solution by following a sequence of orthogonal search directions  $\{\mathbf{d}_0, \dots, \mathbf{d}_n, \dots\}$  with  $\mathbf{d}_n = -\mathbf{g}_n$  satisfying  $\mathbf{g}_n^T \mathbf{g}_{n+1} = 0$ , the CG method also follows a sequence of search directions  $\{\mathbf{d}_0, \dots, \mathbf{d}_{N-1}\}$   $A$ -orthogonal to each other, i.e.,  $\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0$  ( $i \neq j$ ):

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \delta_n \mathbf{d}_n = \dots = \mathbf{x}_0 + \sum_{i=0}^n \delta_i \mathbf{d}_i \quad (2.114)$$

We define the error at the  $n$ th step as  $\mathbf{e}_n = \mathbf{x}_n - \mathbf{x}^*$ . Then subtracting  $\mathbf{x}^*$  from both sides, we get the iteration in terms of the errors:

$$\mathbf{e}_{n+1} = \mathbf{e}_n + \delta_n \mathbf{d}_n = \cdots = \mathbf{e}_0 + \sum_{i=0}^n \delta_i \mathbf{d}_i \quad (2.115)$$

Due to  $\mathbf{g} = \mathbf{Ax} - \mathbf{b}$  in Eq. (2.112), we can find the gradient at the  $n$ th step  $\mathbf{x}_n$  as

$$\mathbf{g}_n = \mathbf{Ax}_n - \mathbf{b} = \mathbf{A}(\mathbf{x}^* + \mathbf{e}_n) - \mathbf{b} = \mathbf{Ax}^* - \mathbf{b} + \mathbf{Ae}_n = \mathbf{Ae}_n \quad (2.116)$$

As the gradient at the solution is zero  $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$ , we can consider the gradient  $\mathbf{g}_n$  at  $\mathbf{x}_n$  as the residual of the  $n$ th iteration, and  $\varepsilon = \|\mathbf{g}_n\|^2$  an error measurement representing how close  $\mathbf{x}_n$  is to the true solution  $\mathbf{x}^*$ .

The optimal step size given in Eq. (2.39) can now be written as

$$\begin{aligned} \delta_i &= -\frac{\mathbf{d}_i^T \mathbf{g}_i}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} = -\frac{\mathbf{d}_i^T \mathbf{A} \mathbf{e}_i}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} = -\frac{\mathbf{d}_i^T \mathbf{A} \left( \mathbf{e}_0 + \sum_{j=0}^{i-1} \delta_j \mathbf{d}_j \right)}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} \\ &= -\frac{\mathbf{d}_i^T \mathbf{A} \mathbf{e}_0 + \sum_{j=0}^{i-1} \delta_j \mathbf{d}_i^T \mathbf{A} \mathbf{d}_j}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} = -\frac{\mathbf{d}_i^T \mathbf{A} \mathbf{e}_0}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} \end{aligned} \quad (2.117)$$

The last equality is due to the fact that  $\mathbf{d}_i$  and  $\mathbf{d}_j$  are A-orthogonal,  $\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0$ . Substituting this into Eq. (2.115) we get

$$\mathbf{e}_{n+1} = \mathbf{e}_n + \delta_n \mathbf{d}_n = \mathbf{e}_n - \left( \frac{\mathbf{d}_n^T \mathbf{A} \mathbf{e}_0}{\mathbf{d}_n^T \mathbf{A} \mathbf{d}_n} \right) \mathbf{d}_n \quad (2.118)$$

On the other hand, we can represent the error  $\mathbf{e}_0 = \mathbf{x}_0 - \mathbf{x}^*$  associated with the initial guess  $\mathbf{x}_0$  as a linear combination of the A-orthogonal search vectors  $\{\mathbf{d}_0, \dots, \mathbf{d}_{N-1}\}$  as  $N$  basis vectors that span the N-D vector space:

$$\mathbf{e}_0 = \sum_{i=0}^{N-1} c_i \mathbf{d}_i = \sum_{i=0}^{N-1} \mathbf{p}_{\mathbf{d}_i}(\mathbf{e}_0) = \sum_{i=0}^{N-1} \left( \frac{\mathbf{d}_i^T \mathbf{A} \mathbf{e}_0}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} \right) \mathbf{d}_i \quad (2.119)$$

where  $\mathbf{p}_{\mathbf{d}_i}(\mathbf{e}_0)$  is the A-projection of  $\mathbf{e}_0$  onto the  $i$ th basis vector  $\mathbf{d}_i$ :

$$\mathbf{p}_{\mathbf{d}_i}(\mathbf{e}_0) = c_i \mathbf{d}_i = \left( \frac{\mathbf{d}_i^T \mathbf{A} \mathbf{e}_0}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} \right) \mathbf{d}_i \quad (2.120)$$

Note that the coefficient  $c_i$  happens to be the negative optimal step size in Eq. (2.117):

$$c_i = \frac{\mathbf{d}_i^T \mathbf{A} \mathbf{e}_0}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} = -\delta_i \quad (2.121)$$

Now the expression of  $\mathbf{e}_{n+1}$  in Eq. (2.115) can be written as

$$\mathbf{e}_{n+1} = \mathbf{e}_0 + \sum_{i=0}^n \delta_i \mathbf{d}_i = \sum_{i=0}^{N-1} c_i \mathbf{d}_i - \sum_{i=0}^n c_i \mathbf{d}_i = \sum_{i=n+1}^{N-1} c_i \mathbf{d}_i = \sum_{i=n+1}^{N-1} \mathbf{p}_{\mathbf{d}_i}(\mathbf{e}_0) \quad (2.122)$$

We see that in each iteration, the number of terms in the summation for  $\mathbf{e}_0$  is

reduced by one, i.e., the  $n$ th component  $\mathbf{p}_{\mathbf{d}_n}(\mathbf{e}_0)$  of  $\mathbf{e}_0$  along the direction of  $\mathbf{d}_n$  is completely eliminated. After  $N$  such iterations, the error is reduced from  $\mathbf{e}_0$  to  $\mathbf{e}_N = \mathbf{0}$ , and the true solution is obtained  $\mathbf{x}_N = \mathbf{x}^* + \mathbf{e}_N = \mathbf{x}^*$ .

Pre-multiplying  $\mathbf{d}_k^T \mathbf{A}$  ( $k \leq n$ ) on both sides of the equation above, we get

$$\mathbf{d}_k^T \mathbf{A} \mathbf{e}_{n+1} = \sum_{i=n+1}^{N-1} c_i \mathbf{d}_k^T \mathbf{A} \mathbf{d}_i = 0 \quad (2.123)$$

We see that after  $n+1$  iterations the remaining error  $\mathbf{e}_{n+1}$  is  $\mathbf{A}$ -orthogonal to all previous directions  $\mathbf{d}_0, \dots, \mathbf{d}_n$ .

Due to Eq. (2.116), the equation above can also be written as

$$\mathbf{d}_k^T \mathbf{A} \mathbf{e}_{n+1} = \mathbf{d}_k^T \mathbf{g}_{n+1} = 0 \quad (2.124)$$

i.e., the gradient  $\mathbf{g}_{n+1}$  is orthogonal to all previous search directions.

In the figure below, the conjugate gradient method is compared with the gradient descent method for the case of  $N = 2$ . We see that the first search direction is the same  $-\mathbf{g}_0$  for both methods. However, the next search direction  $\mathbf{d}_1$  is  $\mathbf{A}$ -orthogonal to  $\mathbf{d}_0$ , same as the next error  $\mathbf{e}_1$ , different from the search direction  $-\mathbf{g}_1$  in gradient descent method. The conjugate gradient method finds the solution  $\mathbf{x}$  in  $N = 2$  steps, while the gradient descent method has to go through many more steps all orthogonal to each other before it finds the solution.

#### Find the $\mathbf{A}$ -orthogonal basis

The  $N$   $\mathbf{A}$ -orthogonal search directions  $\{\mathbf{d}_0, \dots, \mathbf{d}_{N-1}\}$  can be constructed based on any set of  $N$  independent vectors  $\{\mathbf{v}_0, \dots, \mathbf{v}_{N-1}\}$  by the Gram-Schmidt process:

$$\mathbf{d}_n = \mathbf{v}_n - \sum_{j=0}^{n-1} \mathbf{p}_{\mathbf{d}_j}(\mathbf{v}_n) = \mathbf{v}_n - \sum_{m=0}^{n-1} \left( \frac{\mathbf{d}_m^T \mathbf{A} \mathbf{v}_n}{\mathbf{d}_m^T \mathbf{A} \mathbf{d}_m} \right) \mathbf{d}_m = \mathbf{v}_n - \sum_{m=0}^{n-1} \beta_{nm} \mathbf{d}_m \quad (2.125)$$

where  $\beta_{nm} = \mathbf{d}_m^T \mathbf{A} \mathbf{v}_n / (\mathbf{d}_m^T \mathbf{A} \mathbf{d}_m)$  and  $\beta_{nm} \mathbf{d}_m$  is the  $\mathbf{A}$ -projection of  $\mathbf{v}_n$  onto each of the previous direction  $\mathbf{d}_m$ .

We will gain some significant computational advantage if we choose to use  $\mathbf{v}_n = -\mathbf{g}_n$ . Now the Gram-Schmidt process above becomes:

$$\mathbf{d}_n = -\mathbf{g}_n - \sum_{m=0}^{n-1} \beta_{nm} \mathbf{d}_m, \quad (2.126)$$

where

$$\beta_{nm} = \frac{\mathbf{d}_m^T \mathbf{A} \mathbf{v}_n}{\mathbf{d}_m^T \mathbf{A} \mathbf{d}_m} = -\frac{\mathbf{d}_m^T \mathbf{A} \mathbf{g}_n}{\mathbf{d}_m^T \mathbf{A} \mathbf{d}_m} \quad (m < n) \quad (2.127)$$

The equation above indicates that  $\mathbf{g}_n$  can be written as a linear combination of all previous search directions  $\mathbf{d}_0, \dots, \mathbf{d}_n$ :

$$\mathbf{g}_n = \sum_{i=0}^n \alpha_i \mathbf{d}_i \quad (2.128)$$

Pre-multiplying  $\mathbf{g}_{n+1}^T$  on both sides, we get

$$\mathbf{g}_{n+1}^T \mathbf{g}_k = \mathbf{g}_{n+1}^T \left( \sum_{i=0}^k \alpha_i \mathbf{d}_i \right) = \sum_{i=0}^k \alpha_i \mathbf{g}_{n+1}^T \mathbf{d}_i = 0 \quad (2.129)$$

The last equality is due to  $\mathbf{g}_{n+1}^T \mathbf{d}_k = 0$  in Eq. (2.124). We see that  $\mathbf{g}_{n+1}$  is also orthogonal to all previous gradients  $\mathbf{g}_0, \dots, \mathbf{g}_n$ .

Pre-multiplying  $\mathbf{g}_k^T$  ( $k \geq n$ ) on both sides of Eq. (2.126), we get

$$\mathbf{g}_k^T \mathbf{d}_n = -\mathbf{g}_k^T \mathbf{g}_n - \sum_{m=0}^{n-1} \beta_{mn} \mathbf{g}_k^T \mathbf{d}_m = -\mathbf{g}_k^T \mathbf{g}_n = \begin{cases} -\|\mathbf{g}_n\|^2 & n = k \\ 0 & n < k \end{cases} \quad (2.130)$$

Note that all terms in the summation are zero as  $\mathbf{g}_k^T \mathbf{d}_m = 0$  for all  $m < n \leq k$  (Eq. (2.124)). Substituting  $\mathbf{g}_n^T \mathbf{d}_n = -\|\mathbf{g}_n\|^2$  into Eq. (2.117), we get

$$\delta_n = -\frac{\mathbf{g}_n^T \mathbf{d}_n}{\mathbf{d}_n^T \mathbf{A} \mathbf{d}_n} = \frac{\|\mathbf{g}_n\|^2}{\mathbf{d}_n^T \mathbf{A} \mathbf{d}_n} \quad (2.131)$$

Next we consider

$$\mathbf{g}_{m+1} = \mathbf{A} \mathbf{x}_{m+1} - \mathbf{b} = \mathbf{A}(\mathbf{x}_m + \delta_m \mathbf{d}_m) - \mathbf{b} = (\mathbf{A} \mathbf{x}_m - \mathbf{b}) + \delta_m \mathbf{A} \mathbf{d}_m = \mathbf{g}_m + \delta_m \mathbf{A} \mathbf{d}_m \quad (2.132)$$

Pre-multiplying  $\mathbf{g}_n^T$  with  $n > m$  on both sides we get

$$\mathbf{g}_n^T \mathbf{g}_{m+1} = \mathbf{g}_n^T \mathbf{g}_m + \delta_m \mathbf{g}_n^T \mathbf{A} \mathbf{d}_m = \delta_m \mathbf{g}_n^T \mathbf{A} \mathbf{d}_m \quad (2.133)$$

where  $\mathbf{g}_n^T \mathbf{g}_m = 0$  ( $m \neq n$ ). Solving for  $\mathbf{g}_n^T \mathbf{A} \mathbf{d}_m$  we get

$$\mathbf{g}_n^T \mathbf{A} \mathbf{d}_m = \frac{1}{\delta_m} \mathbf{g}_n^T \mathbf{g}_{m+1} = \begin{cases} \|\mathbf{g}_n\|^2 / \delta_{n-1} & m = n-1 \\ 0 & m < n-1 \end{cases} \quad (2.134)$$

Substituting this into Eq. (2.127) we get

$$\beta_{nm} = -\frac{\mathbf{d}_m^T \mathbf{A} \mathbf{g}_n}{\mathbf{d}_m^T \mathbf{A} \mathbf{d}_m} = \begin{cases} -\|\mathbf{g}_n\|^2 / \delta_{n-1} \mathbf{d}_{n-1}^T \mathbf{A} \mathbf{d}_{n-1} & m = n-1 \\ 0 & m < n-1 \end{cases} \quad (2.135)$$

which is non-zero only when  $m = n-1$ , i.e., there is only one non-zero term in the summation of the Gram-Schmidt formula for  $\mathbf{d}_n$ . This is the reason why we choose  $\mathbf{v}_n = -\mathbf{g}_n$ . We can now drop the second subscript  $m$  in  $\beta_{nm}$ , and Eq. (2.125) becomes:

$$\mathbf{d}_n = \mathbf{v}_n - \sum_{m=0}^{n-1} \beta_{nm} \mathbf{d}_m = -\mathbf{g}_n - \beta_n \mathbf{d}_{n-1} \quad (2.136)$$

Substituting the step size  $\delta_{n-1} = \|\mathbf{g}_{n-1}\|^2 / \mathbf{d}_{n-1}^T \mathbf{A} \mathbf{d}_{n-1}$  (Eq. (2.131)) into the above expression for  $\beta_{nm} = \beta_m$ , we get

$$\beta_n = -\frac{\|\mathbf{g}_n\|^2}{\|\mathbf{g}_{n-1}\|^2} \quad (2.137)$$

We note that matrix  $\mathbf{A}$  no longer appears in the expression.

### The CG algorithm

Summarizing the above, we finally get the conjugate gradient algorithm in the following steps:

1. Set  $n = 0$  and initialize the search direction (same as gradient descent):

$$\mathbf{d}_0 = -\mathbf{g}_0 \quad (2.138)$$

2. Terminate if the error  $\varepsilon = \|\mathbf{g}_n\|^2$  is smaller than a preset threshold. Otherwise, continue with the following:
3. Find optimal step size (Eq. (2.131)) and step forward

$$\delta_n = \frac{\|\mathbf{g}_n\|^2}{\mathbf{d}_n^T \mathbf{A} \mathbf{d}_n}, \quad \mathbf{x}_{n+1} = \mathbf{x}_n + \delta_n \mathbf{d}_n \quad (2.139)$$

4. Update gradient:

$$\mathbf{g}_{n+1} = \frac{d}{d\mathbf{x}} f(\mathbf{x}_{n+1}) \quad (2.140)$$

5. Find coefficient for the Gram-Schmidt process (Eq. (2.137)):

$$\beta_{n+1} = -\frac{\|\mathbf{g}_{n+1}\|^2}{\|\mathbf{g}_n\|^2} \quad (2.141)$$

6. Update search direction (Eq. (2.136)):

$$\mathbf{d}_{n+1} = -\mathbf{g}_{n+1} - \beta_{n+1} \mathbf{d}_n \quad (2.142)$$

Set  $n = n + 1$  and go back to step 2.

The algorithm above assumes the objective function  $f(\mathbf{x})$  to be quadratic with known  $\mathbf{A}$ . But when  $f(\mathbf{x})$  is not quadratic,  $\mathbf{A}$  is no longer available, we can still approximate  $f(\mathbf{x})$  as a quadratic function by the first three terms in its Taylor series in the neighborhood of its minimum. Also, the we need to modify the algorithm so that it does not depend on  $\mathbf{A}$ . Specifically, the optimal step size  $\delta_n$  calculated in step 3 above based on  $\mathbf{A}$  can also be alternatively found by line minimization based on any suitable algorithms for 1-D optimization.

The Matlab code for the conjugate gradient algorithm is listed below:

```
function xn=myCG(j,tol)      % j is the objective function to minimize
    syms d;                 % variable for 1-D symbolic function f(d)
    x=symvar(j).';          % symbolic variables in objective function
    J=matlabFunction(j);    % the objective function
    G=jacobian(j).';       % symbolic gradient of objective function
    G=matlabFunction(G);    % the gradient function
    xn=zeros(length(x),1);  % initial guess of x
    xc=num2cell(xn);
    e=J(xc{:});             % initial error
    gn=G(xc{:});            % gradient at xn
    dn=-gn;                 % use negative gradient as search direction
    n=0;
```





$n$	$\mathbf{x} = [x_1, x_2]$	$f(\mathbf{x})$
0	1.500000, -0.750000	2.812500
1	0.250000, -0.750000	0.468750e - 01
2	0.250000, -0.125000	7.812500e - 02
3	0.041667, -0.125000	1.302083e - 02
4	0.041667, -0.020833	2.170139e - 03
5	0.006944, -0.020833	3.616898e - 04
6	0.006944, -0.003472	6.028164e - 05
7	0.001157, -0.003472	1.004694e - 05
8	0.001157, -0.000579	1.674490e - 06
9	0.000193, -0.000579	2.790816e - 07
10	0.000193, -0.000096	4.651361e - 08
11	0.000032, -0.000096	7.752268e - 09
12	0.000032, -0.000016	1.292045e - 09
13	0.000005, -0.000016	2.153408e - 10

However, as expected, the conjugate gradient method takes exactly  $N = 2$  steps from any initial guess to reach at the solution:

$n$	$\mathbf{x} = [x_1, x_2]$	$f(\mathbf{x})$
0	1.500000, -0.750000	2.812500e + 00
1	0.250000, -0.750000	4.687500e - 01
2	0.000000, -0.000000	1.155558e - 33

For an  $N = 3$  example of  $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}$  with

$$\mathbf{A} = \begin{bmatrix} 5 & 3 & 1 \\ 3 & 4 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

from an initial guess  $\mathbf{x}_0 = [1, 2, 3]^T$ , it takes the gradient descent method 41 iterations to reach  $\mathbf{x}_{41} = [3.5486e - 06, -7.4471e - 06, 4.6180e - 06]^T$  corresponding to  $f(\mathbf{x}) = 8.5429e - 11$ . From the same initial guess, it takes the conjugate gradient method only  $N = 3$  iterations to converge to the solution:

$n$	$\mathbf{x} = [x_1, x_2, x_3]$	$f(\mathbf{x})$
0	1.000000, 2.000000, 3.000000	4.500000e + 01
1	-0.734716, -0.106441, 1.265284	2.809225e + 00
2	0.123437, -0.209498, 0.136074	3.584736e - 02
3	-0.000000, 0.000000, 0.000000	3.949119e - 31

For an  $N = 9$  example, it takes over 4000 iterations for the gradient descent method to converge with  $\|\mathbf{e}\| \approx 10^{-10}$ , but exactly 9 iterations for the CG method to converge with  $\|\mathbf{e}\| \approx 10^{-16}$ .

**Example:**

The figure below shows the search path of the conjugate gradient method applied to the minimization of the Rosenbrock function:

**Example:**

Solve the following  $N = 3$  non-linear equation system by the CG method:

$$\begin{cases} f_1(x_1, x_2, x_3) = 3x_1 - (x_2x_3)^2 - 3/2 \\ f_2(x_1, x_2, x_3) = 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ f_3(x_1, x_2, x_3) = \exp(-x_1x_2) + 20x_3 + 9 \end{cases}$$

The solution is known to be  $x_1 = 0.5$ ,  $x_2 = 0$ ,  $x_3 = -0.5$ .

This equation system can be represented in vector form as  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  and the objective function is  $J(\mathbf{x}) = \mathbf{f}^T(\mathbf{x})\mathbf{f}(\mathbf{x})$ . The iteration of the CG method with an initial guess  $\mathbf{x}_0 = \mathbf{0}$  is shown below:

$n$	$\mathbf{x} = [x_1, x_2, x_3]$	$J(\mathbf{x})$
0	0.0000, 0.0000, 0.0000	1.032500e + 02
1	0.0113, 0.0050, -0.5001	3.160163e + 00
2	0.0188, -0.0021, -0.5004	3.095894e + 00
3	0.5009, -0.0018, -0.5004	7.268252e - 05
4	0.5009, -0.0017, -0.5000	1.051537e - 05
5	0.5008, -0.0012, -0.5000	6.511151e - 06
6	0.5001, -0.0005, -0.5000	6.365321e - 07
7	0.5001, -0.0005, -0.5000	5.667357e - 07
8	0.5002, -0.0004, -0.5000	2.675128e - 07
9	0.5001, -0.0003, -0.5000	1.344218e - 07
10	0.5001, -0.0002, -0.5000	1.241196e - 07
11	0.5000, -0.0001, -0.5000	2.120969e - 08
12	0.5000, -0.0001, -0.5000	1.541814e - 08
13	0.5000, -0.0001, -0.5000	7.282025e - 09
14	0.5000, -0.0001, -0.5000	4.801781e - 09
15	0.5000, -0.0000, -0.5000	4.463926e - 09

In comparison, the gradient descent method would need to take over 200 iterations (with much reduced complexity though) to reach this level of error.

**Conjugate gradient method used for solving linear equation systems:**

As discussed before, if  $\mathbf{x}$  is the solution that minimizes the quadratic function  $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} / 2 - \mathbf{b}^T \mathbf{x} + c$ , with  $\mathbf{A} = \mathbf{A}^T$  being symmetric and positive definite, it also satisfies  $df(\mathbf{x})/d\mathbf{x} = \mathbf{A} \mathbf{x} - \mathbf{b} = \mathbf{0}$ . In other words, the optimization problem is equivalent to the problem of solving the linear system  $\mathbf{A} \mathbf{x} - \mathbf{b} = \mathbf{0}$ , both can be solved by the conjugate gradient method.

Now consider solving the linear system  $\mathbf{A} \mathbf{x} = \mathbf{b}$  with  $\mathbf{A} = \mathbf{A}^T$ . Let  $\mathbf{d}_i$ , ( $i = 1, \dots, N$ ) be a set of  $N$  A-orthogonal vectors satisfying  $\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0$  ( $i \neq j$ ), which can be generated based on any  $N$  independent vectors, such as the standard basis vectors, by the Gram-Smidth method. The solution  $\mathbf{x}$  of the equation

$\mathbf{Ax} = \mathbf{b}$  can be represented by these  $N$  vectors as

$$\mathbf{x} = \sum_{i=1}^N c_i \mathbf{d}_i \quad (2.143)$$

Now we have

$$\mathbf{b} = \mathbf{Ax} = \mathbf{A} \left[ \sum_{i=1}^N c_i \mathbf{d}_i \right] = \sum_{i=1}^N c_i \mathbf{Ad}_i \quad (2.144)$$

Pre-multiplying  $\mathbf{d}_j^T$  on both sides we get

$$\mathbf{d}_j^T \mathbf{b} = \sum_{i=1}^N c_i \mathbf{d}_j^T \mathbf{Ad}_i = c_j \mathbf{d}_j^T \mathbf{Ad}_j \quad (2.145)$$

Solving for  $c_j$  we get

$$c_j = \frac{\mathbf{d}_j^T \mathbf{b}}{\mathbf{d}_j^T \mathbf{Ad}_j} \quad (2.146)$$

Substituting this back into the expression for  $\mathbf{x}$  we get the solution of the equation:

$$\mathbf{x} = \sum_{i=1}^N c_i \mathbf{d}_i = \sum_{i=1}^N \left( \frac{\mathbf{d}_i^T \mathbf{b}}{\mathbf{d}_i^T \mathbf{Ad}_i} \right) \mathbf{d}_i \quad (2.147)$$

Also note that as  $\mathbf{b} = \mathbf{Ax}$ , the  $i$ th term of the summation above is simply the A-projection of  $\mathbf{x}$  onto the  $i$ th direction  $\mathbf{d}_i$ :

$$\mathbf{p}_{\mathbf{d}_i}(\mathbf{x}) = \left( \frac{\mathbf{d}_i^T \mathbf{Ax}}{\mathbf{d}_i^T \mathbf{Ad}_i} \right) \mathbf{d}_i \quad (2.148)$$

One application of the conjugate gradient method is to solve the normal equation to find the least-square solution of an over-constrained equation system  $\mathbf{Ax} = \mathbf{b}$ , where the coefficient matrix  $\mathbf{A}$  is  $M$  by  $N$  with rank  $R = N < M$ . As discussed previously, the normal equation of this system is

$$\mathbf{A}^T \mathbf{Ax} = \mathbf{A}^T \mathbf{b} \quad (2.149)$$

Here  $\mathbf{A}^T \mathbf{A}$  is an  $N$  by  $N$  symmetric, positive definite matrix. This normal equation can be solved by the conjugate gradient method.

### 2.1.7 Issues of Local/Global Minimum

A main concern in optimization problems is whether the solution obtained by the algorithm is a local minimum or a global one. The simulated annealing (SA) algorithm is typically used to search for the global minimum in a high-dimensional space, by allowing up-hill moves in the iteration to avoid getting stuck by a local minimum. SA mimics the annealing process in metallurgy, in which the temperature is carefully controlled to decrease slowly so that the material reaches a state with minimum free energy.

The way the SA method avoids getting stuck at a local minimum is to allow the iteration steps to go up-hill. Specifically, the objective function  $f(\mathbf{x})$  evaluated at  $\mathbf{x}_n$  is treated as the energy at  $\mathbf{x}_n$ , and an up-hill move corresponding to an energy increase  $\Delta E = f(\mathbf{x}_{n+1}) - f(\mathbf{x}_n)$  can be accepted if some probability function is greater than a random threshold value  $Th$  within a certain range comparable with  $P$ :

$$P(\Delta E, T) = e^{-\Delta E/T} > Th \quad (2.150)$$

Obviously smaller  $\Delta E$  and higher  $T$  will cause higher  $P$ , it is more probable for a move to be accepted.

During the entire search process, the temperature  $T$  gradually reduces so that the probability of accepting an up-hill move also gradually reduces. In the early stage of the search, it is more probable to go up-hill, thereby getting out of a local minimum; but toward the later part of the search process, such a move is less likely to happen, as presumably in this late stage of the search, the energy level is already low, not likely to be at a local minimum with high energy level. Through out the search, the best result corresponding to the lowest energy level is always updated, to be returned as the final solution at the end of the process.

## 2.2 Constrained Optimization

An optimization problem is more complicated if it is constrained, i.e., the arguments  $\mathbf{x}$  of the objective function  $f(\mathbf{x})$  are subject to (abbreviated as s.t. in the following) certain equality or inequality constraints in terms of what values they can take. Such a generic constrained optimization problem can be formulated as:

$$\begin{cases} \text{max/min:} & f(\mathbf{x}) = f(x_1, \dots, x_N) \\ \text{s. t.:} & \begin{cases} h_i(\mathbf{x}) = 0, & (i = 1, \dots, m) \\ g_j(\mathbf{x}) \leq 0, & (j = 1, \dots, n) \end{cases} \end{cases} \quad (2.151)$$

The set composed of all values of  $\mathbf{x}$  that satisfy the  $m + n$  constraints is called the *feasible region* of the problem. The goal is to find a point  $\mathbf{x}^*$  in the feasible region at which  $f(\mathbf{x}^*)$  is an extremum.

In the most general case where the objective function  $f(\mathbf{x})$  and the constraint functions  $g_i(\mathbf{x})$  and  $h_j(\mathbf{x})$  are nonlinear, the process of solving this nonlinear optimization problem is called *nonlinear programming (NLP)*. Specially, if the objective function is quadratic while the constraints are linear, (the feasible region is a polytope), the process is called *quadratic programming (QP)*. More specially, if the objective function is linear as well as the constraints, the process is called *linear programming (LP)*. In the following, we will first consider NLP in general, and then discuss more specifically QP and LP.

### 2.2.1 Optimization with Equality Constraints

The optimization problems subject to equality constraints can be generally formulated as:

$$\begin{cases} \text{max/min:} & f(\mathbf{x}) = f(x_1, \dots, x_N) \\ \text{s. t.:} & h_i(\mathbf{x}) = h_i(x_1, \dots, x_N) = 0, \quad (i = 1, \dots, m) \end{cases} \quad (2.152)$$

where  $f(\mathbf{x})$  is the objective function and  $h_i(\mathbf{x}) = 0$  is the  $i$ th constraint. Here  $m \leq N$ , as in general there does not exist a solution that satisfies more than  $N$  equations in the  $N$ -D space.

This problem can be visualized in the special case with  $N = 2$  and  $m = 1$ , where both  $f(\mathbf{x})$  and  $h(\mathbf{x})$  are surfaces defined over the 2-D space spanned by  $x_1$  and  $x_2$ , and  $h(\mathbf{x}) = 0$  is the intersection line of  $h(\mathbf{x})$  and the 2-D plane. The solution  $\mathbf{x}^*$  must satisfy the following two conditions:

- $\mathbf{x}^*$  is on the intersection line, i.e.,  $h(\mathbf{x}^*) = 0$ ;
- $\mathbf{x}^*$  is on the contour line  $f(\mathbf{x}) = d$  (for some  $d$  as shown in the figure), so that  $f(\mathbf{x}^*)$  neither increase nor decrease (no point is higher or lower) in the neighborhood of  $\mathbf{x}^*$  along the contour line.

For both conditions to be satisfied, the two contours  $h(\mathbf{x}) = 0$  and  $f(\mathbf{x}) = d$  must coincide at  $\mathbf{x}^*$ , i.e., they must have the same tangents and therefore parallel gradients (perpendicular to the tangents):

$$\nabla_{\mathbf{x}} f(\mathbf{x}^*) = \lambda \nabla_{\mathbf{x}} h(\mathbf{x}^*) \quad (2.153)$$

where  $\nabla_{\mathbf{x}} f(\mathbf{x}) = [\partial f / \partial x_1, \dots, \partial f / \partial x_N]^T$  is the gradient of  $f(\mathbf{x})$ , and  $\lambda$  is a constant scaling factor which can be either positive if the two gradients are in the same direction or negative if they are in opposite directions. As here we are only interested in whether the two curves coincide or not, the sign of  $\lambda^*$  can be either positive or negative.

To find such a point  $\mathbf{x}^*$ , we first construct a new objective function, called the *Lagrange function* (or simply *Lagrangian*):

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda h(\mathbf{x}) \quad (2.154)$$

where  $\lambda$  is the *Lagrange multiplier*, which can be either positive or negative (i.e., the sign in front of  $\lambda$  can be either plus or minus), and then set the gradient of this objective function with respect to  $\lambda$  as well as  $\mathbf{x}$  to zero:

$$\nabla_{\mathbf{x}, \lambda} L(\mathbf{x}, \lambda) = \nabla_{\mathbf{x}, \lambda} [f(\mathbf{x}) - \lambda h(\mathbf{x})] = \mathbf{0} \quad (2.155)$$

This equation contains two parts:

$$\begin{cases} \nabla_{\mathbf{x}} f(\mathbf{x}) = \lambda \nabla_{\mathbf{x}} h(\mathbf{x}) \\ \nabla_{\lambda} L(\mathbf{x}, \lambda) = \partial L(\mathbf{x}, \lambda) / \partial \lambda = -h(\mathbf{x}) = 0 \end{cases} \quad (2.156)$$

This is an equation system containing  $N + m = 2 + 1 = 3$  equations for the same

number of unknowns  $x_1, x_2, \lambda$ :

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \lambda \frac{\partial h(\mathbf{x})}{\partial x_i} \quad (i = 1, 2), \quad h(\mathbf{x}) = 0 \quad (2.157)$$

The solution  $\mathbf{x}^* = [x_1^*, x_2^*]^T$  and  $\lambda^*$  of this equation system happens to satisfy the two desired relationships: (a)  $\nabla_{\mathbf{x}} f(\mathbf{x}) = \lambda \nabla_{\mathbf{x}} h(\mathbf{x})$ , and (b)  $h(\mathbf{x}^*) = 0$ .

Specially, consider the following two possible cases:

- If  $\lambda^* = 0$ , then  $\nabla_{\mathbf{x}} f(\mathbf{x}^*) = \lambda^* \nabla_{\mathbf{x}} h(\mathbf{x}^*) = 0$ , indicating  $f(\mathbf{x}^*)$  is an extremum independent of the constraint, i.e., the constraint  $h(\mathbf{x}) = 0$  is *inactive*, and the optimization is unconstrained.
- If  $\lambda^* \neq 0$ , then  $\nabla_{\mathbf{x}} f(\mathbf{x}^*) \neq 0$  (as in general  $\nabla_{\mathbf{x}} h(\mathbf{x}^*) \neq \mathbf{0}$ ), indicating  $f(\mathbf{x}^*)$  is not an extremum without the constraint, i.e., the constraint is *active*, and the optimization is indeed constrained.

The discussion above can be generalized from 2-D to an  $N > 2$  dimensional space, in which the optimal solution  $\mathbf{x}^*$  is to be found to extremize the objective  $f(\mathbf{x})$  subject to  $m$  equality constraints  $h_i(\mathbf{x}) = 0$  ( $i = 1, \dots, m$ ), each representing a contour surface in the  $N$ -D space. The solution  $\mathbf{x}^*$  must satisfy the following two conditions:

- $\mathbf{x}^*$  is on all  $m$  contour surfaces so that  $h_1(\mathbf{x}^*) = \dots = h_m(\mathbf{x}^*) = 0$ , i.e., it is on the intersection of these surfaces, a curve in the space (e.g., the intersection of two surfaces in 3-D space is a curve);
- $\mathbf{x}^*$  is on a contour surface  $f(\mathbf{x}^*) = d$  (for some value  $d$ ) so that  $f(\mathbf{x}^*)$  is an extremum, i.e., it does not increase or decrease in the neighborhood of  $\mathbf{x}^*$  along the curve.

Combining these conditions, we see that the intersection (a curve or a surface) of the  $m$  surfaces  $h_i(\mathbf{x}) = 0$  must coincide with the contour surface  $f(\mathbf{x}) = d$  at  $\mathbf{x}^*$ , i.e., the intersection of the tangent surfaces of  $h_i(\mathbf{x}) = 0$  through  $\mathbf{x}^*$  must coincide with the tangent surface of  $f(\mathbf{x}) = d$  at  $\mathbf{x}^*$ , therefore the gradient  $\nabla f(\mathbf{x}^*)$  must be a linear combination of the  $m$  gradients  $\nabla h_i(\mathbf{x}^*)$ :

$$\nabla_{\mathbf{x}} f(\mathbf{x}^*) = \sum_{i=1}^m \lambda_i^* \nabla_{\mathbf{x}} h_i(\mathbf{x}^*) \quad (2.158)$$

To find such a solution  $\mathbf{x}^*$  of the optimization problem, we first construct the Lagrange function:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i=1}^m \lambda_i h_i(\mathbf{x}) \quad (2.159)$$

where  $\boldsymbol{\lambda} = [\lambda_1, \dots, \lambda_m]^T$  is a vector for  $m$  *Lagrange multipliers*, and then set its gradient to zero:

$$\nabla_{\mathbf{x}, \boldsymbol{\lambda}} L(\mathbf{x}, \boldsymbol{\lambda}) = \nabla_{\mathbf{x}, \boldsymbol{\lambda}} \left[ f(\mathbf{x}) - \sum_{i=1}^m \lambda_i h_i(\mathbf{x}) \right] = \mathbf{0} \quad (2.160)$$

to get the following two equation systems of  $N$  and  $m$  equations respectively:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \sum_{i=1}^m \lambda_i \nabla_{\mathbf{x}} h_i(\mathbf{x}) \quad \text{i.e.} \quad \frac{\partial f(\mathbf{x})}{\partial x_j} = \sum_{i=1}^m \lambda_i \frac{\partial h_i(\mathbf{x})}{\partial x_j} \quad (j = 1, \dots, N), \quad (2.161)$$

and

$$\frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_i} = h_i(\mathbf{x}) = 0 \quad (i = 1, \dots, m) \quad (2.162)$$

The first set of  $N$  equations indicates that the gradient  $\nabla_{\mathbf{x}} f(\mathbf{x}^*)$  at  $\mathbf{x}^*$  is a linear combination of the  $m$  gradients  $\nabla_{\mathbf{x}} h_i(\mathbf{x}^*)$ , while the second set of  $m$  equations guarantees that  $\mathbf{x}^*$  also satisfy the  $m$  equality constraints. Solving these equations we get the desired solution  $\mathbf{x}^*$  together with the Lagrange multipliers  $\lambda_1, \dots, \lambda_m$ . This is the *Lagrange multiplier method*.

Specially, if  $\lambda_i = 0$ , then the  $i$ th constraint is not active. More specially if  $\lambda_i = 0$  for all  $i = 1, \dots, m$ , then

$$\nabla_{\mathbf{x}} f(\mathbf{x}^*) = \sum_{i=1}^m \lambda_i \nabla_{\mathbf{x}} h_i(\mathbf{x}) = \mathbf{0} \quad (2.163)$$

indicating  $f(\mathbf{x}^*)$  is an extremum independent of any of the constraints, i.e.,  $\mathbf{x}^*$  is an unconstrained solution, and the optimization problem is unconstrained.

### 2.2.2 Optimization with Inequality Constraints

The optimization problems subject to inequality constraints can be generally formulated as:

$$\begin{cases} \text{max/min:} & f(\mathbf{x}) = f(x_1, \dots, x_N) \\ \text{s. t.:} & g_j(\mathbf{x}) = g_j(x_1, \dots, x_N) \leq \text{ or } \geq 0, \quad (j = 1, \dots, n) \end{cases} \quad (2.164)$$

Again, to visualize the problem we first consider an example with  $N = 2$  and  $n = 1$ , as shown in the figure below for the minimization (left) and maximization (right) of  $f(\mathbf{x})$  subject to  $g(\mathbf{x}) > 0$ . The constrained solution  $\mathbf{x}^* = [x_1^*, x_2^*]^T$  is on the boundary of the feasible region satisfying  $g(x_1, x_2) = 0$ , while the unconstrained extremum is outside the feasible region.

Consider the following two possible cases.

- First, if the unconstrained extremum at which  $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$  is outside the feasible region, i.e., the inequality constraint is active, then the constrained solution  $\mathbf{x}^*$  must be
  - on the boundary of the feasible region, i.e.,  $g(\mathbf{x}^*) = 0$ ,
  - different from the unconstrained solution, i.e.,  $\nabla_{\mathbf{x}} f(\mathbf{x}^*) \neq \mathbf{0}$ ;

The problem becomes the same as an equality constrained problem considered before, which requires the objective function  $f(\mathbf{x})$  and the constraining function  $g(\mathbf{x})$  to have the same tangent at  $\mathbf{x}^*$ , or parallel gradients:

$$\nabla_{\mathbf{x}} f(\mathbf{x}^*) = \mu^* \nabla_{\mathbf{x}} g(\mathbf{x}^*) \quad \text{or} \quad \nabla_{\mathbf{x}} f(\mathbf{x}^*) - \mu^* \nabla_{\mathbf{x}} g(\mathbf{x}^*) = \mathbf{0} \quad (2.165)$$

However, different from before, now we are also concerned with whether the two gradients have the same or opposite directions, corresponding to  $\mu > 0$  or  $\mu < 0$ , respectively, as illustrated in the 1-D examples in the figure below:

Depending on whether  $f(\mathbf{x})$  is to be maximized or minimized, and whether the constraint is  $g(\mathbf{x}) \geq 0$  or  $g(\mathbf{x}) \leq 0$ , there exist four possible cases in terms of the sign of  $\mu$ , as summarized in the table below:

	$g(\mathbf{x}) \geq 0$	$g(\mathbf{x}) \leq 0$
$\max f(\mathbf{x})$	(1) $\nabla f(\mathbf{x}_1) = \mu \nabla g(\mathbf{x}_1), \mu < 0$	(3) $\nabla f(\mathbf{x}_3) = \mu \nabla g(\mathbf{x}_3), \mu > 0$
$\min f(\mathbf{x})$	(2) $\nabla f(\mathbf{x}_2) = \mu \nabla g(\mathbf{x}_2), \mu > 0$	(4) $\nabla f(\mathbf{x}_4) = \mu \nabla g(\mathbf{x}_4), \mu < 0$

(2.166)

Case	max/min	subject to	$\mu$
(1)	$\max f(\mathbf{x})$	$g(\mathbf{x}) \geq 0$	$\mu < 0$
(2)	$\min f(\mathbf{x})$	$g(\mathbf{x}) \geq 0$	$\mu > 0$
(3)	$\max f(\mathbf{x})$	$g(\mathbf{x}) \leq 0$	$\mu > 0$
(4)	$\min f(\mathbf{x})$	$g(\mathbf{x}) \leq 0$	$\mu < 0$

(2.167)

We can also consider the product of  $\mu$  and  $g(\mathbf{x})$  for either maximization or minimization:

$$\mu g(\mathbf{x}) \begin{cases} \leq 0 & \text{for maximization} \\ \geq 0 & \text{for minimization} \end{cases} \quad (2.168)$$

- Second, if the unconstrained extremum is inside the feasible region, i.e., the inequality constraint is inactive, then the problem is actually unconstrained and the results above are no longer valid. The solution  $\mathbf{x}^*$  of this unconstrained problem is

- the same as the unconstrained solution, i.e.,  $\nabla_{\mathbf{x}} f(\mathbf{x}^*) = \mathbf{0}$ ;
- not on the boundary of the feasible region, i.e.,  $g(\mathbf{x}^*) \neq 0$ ;

Now we need to assume  $\mu = 0$  and find the solution solving the same equation  $\nabla_{\mathbf{x}} f(\mathbf{x}) = \mu \nabla_{\mathbf{x}} g(\mathbf{x}) = \mathbf{0}$

Summarizing the two cases above, we see that  $g(\mathbf{x}^*) = 0$  but  $\mu^* \neq 0$  in the first case,  $g(\mathbf{x}^*) \neq 0$  but  $\mu^* = 0$  in the second case, i.e., the following holds in either case:

$$\mu^* g(\mathbf{x}^*) = 0 \quad (2.169)$$

The discussion above can be generalized from 2-D to  $N > 2$  dimensional space, in which the optimal solution  $\mathbf{x}^*$  is to be found to extremize the objective  $f(\mathbf{x})$  subject to  $n$  inequality constraints  $g_i(\mathbf{x}) = 0$  ( $i = 1, \dots, n$ ). To solve this inequality constrained optimization problem, we first construct the Lagrangian:

$$L(\mathbf{x}, \mu) = f(\mathbf{x}) - \sum_{i=1}^n \mu_i g_i(\mathbf{x}) \quad (2.170)$$



We note that in some literatures, a plus sign is used in front of the summation of the second term. This is equivalent to our discussion here so long as the sign of  $\mu$  indicated in Table 2.167 is negated.

We now set the gradient of the Lagrangian to zero:

$$\nabla_{\mathbf{x}, \mu} L(\mathbf{x}, \mu) = \nabla_{\mathbf{x}, \mu} \left[ f(\mathbf{x}) - \sum_{i=1}^n \mu_i g_i(\mathbf{x}) \right] = \mathbf{0} \quad (2.171)$$

and get two equation systems of  $N$  and  $n$  equations, respectively:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \sum_{i=1}^n \mu_i \nabla_{\mathbf{x}} g_i(\mathbf{x}) \quad (2.172)$$

and

$$\frac{\partial L(\mathbf{x}, \mu)}{\partial \mu_i} = g_i(\mathbf{x}) = 0 \quad (i = 1, \dots, n) \quad (2.173)$$

The result above for the inequality constrained problems is the same as that for the equality constrained problems considered before. However, we note that there is an additional requirement regarding the sign of the scaling coefficients. For an equality constrained problem, the direction of the gradient  $\nabla g(\mathbf{x})$  is of no concern, i.e., the sign of  $\lambda$  is unrestricted; but here for an inequality constrained problem, the sign of  $\mu$  needs to be consistent with those shown in Table 2.167, otherwise the constraints may be inactive.

As a simple example, consider the iterative minimization of a function  $f(\mathbf{x})$ . To find the best increment  $\Delta \mathbf{x}$  that minimizes  $f(\mathbf{x} + \Delta \mathbf{x}) \approx f(\mathbf{x}) + \mathbf{g}_f(\mathbf{x})^T \Delta \mathbf{x}$  but at the same time is also subjected to the constraint of limited length, i.e.,  $\|\Delta \mathbf{x}\|^2/2 \leq C$ , we first construct the Lagrangian:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \mathbf{g}_f \mathbf{x}^T \Delta \mathbf{x} - \mu (\|\Delta \mathbf{x}\|^2/2 - C) \quad (2.174)$$

and set its derivative to zero:

$$\frac{d}{d\Delta \mathbf{x}} L(\mathbf{x}, \lambda) \mathbf{g}_f(\mathbf{x}) - \mu \Delta \mathbf{x} = \mathbf{0} \quad (2.175)$$

where  $\mu < 0$  according to Table 2.167. We then solve the equation to get

$$\Delta \mathbf{x} = \frac{1}{\mu} \mathbf{g}_f(\mathbf{x}) \quad (2.176)$$

and the iteration

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \delta \mathbf{g}(\mathbf{x}_n) \quad (2.177)$$

same as the gradient descent method in Eq. (2.25).

We now consider the general optimization of an N-D objective function  $f(\mathbf{x})$  subject to multiple constraints of both equalities and inequalities:

$$\begin{aligned} \max/\min: & f(\mathbf{x}) = f(x_1, \dots, x_N) \\ \text{s. t.}: & \begin{cases} h_i(\mathbf{x}) = 0, & (i = 1, \dots, m) \\ g_j(\mathbf{x}) \leq 0 \text{ or } g_j(\mathbf{x}) \geq 0, & (j = 1, \dots, n) \end{cases} \end{aligned} \quad (2.178)$$

For notational convenience, we represent the  $m + n$  equality and inequality constraints in vector form as:

$$\begin{aligned} \text{max/min: } & f(\mathbf{x}) \\ \text{s. t.: } & \begin{cases} \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \text{ or } \mathbf{g}(\mathbf{x}) \geq \mathbf{0} \end{cases} \end{aligned} \quad (2.179)$$

where  $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \dots, h_m(\mathbf{x})]^T$  and  $\mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}), \dots, g_n(\mathbf{x})]^T$ .

To solve this optimization problem, we first construct the Lagrangian

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) - \sum_{i=1}^m \lambda_i h_i(\mathbf{x}) - \sum_{j=1}^n \mu_j g_j(\mathbf{x}) = f(\mathbf{x}) - \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) - \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}) \quad (2.180)$$

where the Lagrange multipliers in  $\boldsymbol{\lambda} = [\lambda_1, \dots, \lambda_m]^T$  and  $\boldsymbol{\mu} = [\mu_1, \dots, \mu_n]^T$  are for the  $m$  equality and  $n$  non-negative constraints, respectively. We then set to zero the gradient of  $L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$  with respect to both  $\boldsymbol{\lambda}$  and  $\boldsymbol{\mu}$  as well as  $\mathbf{x}$ . The solution  $\mathbf{x}^*$  can then be obtained by solving the resulting equation system. While  $\lambda_i$  can be either positive or negative, with sign of  $\mu_j$  needs to be consistent with those specified in Table 2.167. Otherwise the inequality constraints is inactive.

**Example:**

Find the extremum of  $f(x_1, x_2) = x_1^2 + x_2^2$  subject to each of the three different constraints:  $x_1 + x_2 = 1$ ,  $x_1 + x_2 \leq 1$ , and  $x_1 + x_2 \geq 1$ .

•

$$\begin{cases} \text{min/max: } & f(x_1, x_2) = x_1^2 + x_2^2 \\ \text{s. t.: } & h(x_1, x_2) = x_1 + x_2 - 1 = 0 \end{cases}$$

The Lagrangian is:

$$L(x_1, x_2, \lambda) = f(x_1, x_2) - \lambda g(x_1, x_2) = x_1^2 + x_2^2 - \lambda(x_1 + x_2 - 1)$$

Solving the following equations

$$\frac{\partial L}{\partial x_1} = 2x_1 - \lambda = 0, \quad \frac{\partial L}{\partial x_2} = 2x_2 - \lambda = 0, \quad \frac{\partial L}{\partial \lambda} = x_1 + x_2 - 1 = 0$$

we get  $\lambda^* = 1$ ,  $x_1^* = x_2^* = 0.5$ , i.e., the function is minimized at  $f(0.5, 0.5) = 0.5$ . We note that  $f(x_1, x_2)$  and  $g(x_1, x_2)$  have the same gradients:

$$\nabla f(0.5, 0.5) = \nabla h(0.5, 0.5) = [1, 1]^T$$

• Consider these two problems:

$$\begin{cases} \text{min: } & f(x_1, x_2) = x_1^2 + x_2^2 \\ \text{s. t.: } & g(x_1, x_2) = x_1 + x_2 - 1 \geq 0 \end{cases} \quad \begin{cases} \text{max: } & f(x_1, x_2) = x_1^2 + x_2^2 \\ \text{s. t.: } & g(x_1, x_2) = x_1 + x_2 - 1 \leq 0 \end{cases}$$

These problems are cases 2 and 3 in Table 2.167. They have the same Lagrangian as in the previous case and therefore the same results  $x_1^* = x_2^* = 0.5$  and  $\mu^* = 1 > 0$ . According to Table 2.167,  $\mathbf{x}^*$  is the solution for the minimization problem subject to  $g(\mathbf{x}) \geq 0$  (left), or the maximization problem subject to  $g(\mathbf{x}) \leq 0$ .

- Consider these two problems:

$$\begin{cases} \max: & f(x_1, x_2) = x_1^2 + x_2^2 \\ \text{s. t.}: & g(x_1, x_2) = x_1 + x_2 - 1 \geq 0 \end{cases} \quad \begin{cases} \min: & f(x_1, x_2) = x_1^2 + x_2^2 \\ \text{s. t.}: & g(x_1, x_2) = x_1 + x_2 - 1 \leq 0 \end{cases}$$

These problems are cases 1 and 4 in Table 2.167. They have the same Lagrangian as in the previous case and therefore the same results  $x_1^* = x_2^* = 0.5$ , and  $\mu^* = 1 > 0$ . According to Table 2.167,  $\mathbf{x}^*$  is *not* the solution of either the maximization problem subject to  $g(\mathbf{x}) \geq 0$  (left), or the minimization problem subject to  $g(\mathbf{x}) \leq 0$  (right), i.e., the constraint is inactive. We therefore need to assume  $\mu^* = 0$  and solve the following equations for an unconstrained problem:

$$\frac{\partial L}{\partial x_1} = 2x_1 = 0, \quad \frac{\partial L}{\partial x_2} = 2x_2 = 0$$

Now we get  $x_1^* = x_2^* = 0$  and  $\mathbf{x}^* = \mathbf{0}$ , with minimum  $f(\mathbf{x}^*) = 0$ . This is the solution for the minimization problem (right), at which  $g(\mathbf{x}^*) \neq 0$ , i.e., the constraint is inactive. However this is not the solution for the maximization problem (left) as the function is not bounded from above.

### 2.2.3 Duality and KKT Conditions

Previously we consider the constrained minimization problem:

$$\begin{cases} \min: & f_p(\mathbf{x}) \\ \text{s. t.}: & \mathbf{h}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{g}(\mathbf{x}) \leq \text{or } \geq \mathbf{0} \end{cases} \quad (2.181)$$

with its Lagrangian:

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f_p(\mathbf{x}) - \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) - \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}) \quad (2.182)$$

The optimal solution  $\mathbf{x}^*$  can be found by solving the following equation

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{0} \quad (2.183)$$

at which the objective function reaches its minimum  $p^* = f_p(\mathbf{x}^*)$ .

In particular, if the objective function  $f_p(\mathbf{x})$  and all inequality constraints  $\mathbf{g}_i(\mathbf{x})$  are convex and the equality constraints  $\mathbf{h}_i(\mathbf{x})$  are linear (affine), i.e., the feasible region is convex (Section A.24), then the problem is a *convex optimization* with some nice properties as shown below.

This problem is called the *primal problem*, as there also exists an equivalent *dual problem*, which may be easier to solve in certain sense. We first define the *dual function* as the infimum (lower bound) of the Lagrangian to be minimized:

$$f_d(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x}} \left[ f_p(\mathbf{x}) - \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) - \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}) \right] \quad (2.184)$$

Furthermore, to find the tightest lower bound, the supremum (upper bound) of

the dual function  $f_d(\boldsymbol{\lambda}, \boldsymbol{\mu})$  over its variables  $\boldsymbol{\lambda}$  and  $\boldsymbol{\mu}$ , we solve the following dual problem:

$$\begin{cases} \max: & f_d(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \\ \text{s. t.}: & \boldsymbol{\mu} \leq \text{or} \geq \mathbf{0} \end{cases} \quad (2.185)$$

Here  $\boldsymbol{\lambda}$  for the equality constraints of the primal problem can be either positive or negative, i.e., it is not constrained in the dual problem. However, the sign of  $\boldsymbol{\mu}$  for the inequalities constraint of the primal minimization problem needs to satisfy the following according to Eq. (2.168) (the second inequality for this minimization primal problem):

$$\boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}) \geq 0 \quad (2.186)$$

i.e.,  $\boldsymbol{\mu}$  is subject to the inequality constraint of the dual problem. We denote the optimal solution of the dual problem by  $\{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$ , and the maximum of the objective function by  $d^* = f_d(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ .

Obviously the primal objective function of any feasible  $\mathbf{x}$  is always no less than its lower bound, the dual objective function of any feasible  $\{\boldsymbol{\lambda}, \boldsymbol{\mu}\}$ :

$$f_p(\mathbf{x}) \geq f_d(\boldsymbol{\lambda}, \boldsymbol{\mu}) \quad (2.187)$$

and this inequality also holds for the optimal solutions  $\mathbf{x}^*$  and  $\{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$ :

$$p^* = f_p(\mathbf{x}^*) \geq f_d(\boldsymbol{\lambda}, \boldsymbol{\mu}) = d^* \quad (2.188)$$

The difference  $p^* - d^*$  is defined as the *duality gap*. Obviously  $p^* - d^* \geq 0$  always holds and the duality is called *weak duality*. In some problems where certain conditions (*Slater's conditions*) are satisfied,  $p^* = d^*$ , i.e.,  $d^*$  is the tightest lower bound, the reality is called *strong duality*.

We now consider the necessary and sufficient conditions for the solutions of both the primal and dual problems. We first consider the *Karush-Kuhn-Tucker (KKT)* conditions:

- *Stationarity*:  $\mathbf{x}^*$  maximizes or minimizes the Lagrangian, i.e.,

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{0} \quad (2.189)$$

- *Primal feasibility*:  $\mathbf{x}^*$  must be inside the feasible region:

$$h_i(\mathbf{x}^*) = 0, \quad g_j(\mathbf{x}^*) \leq \text{or} \geq 0, \quad (i = 1, \dots, m, j = 1, \dots, n) \quad (2.190)$$

- *Dual feasibility*: All multipliers in  $\boldsymbol{\mu}$  for the inequalities must take proper sign as specified by Table 2.167:

$$\mu_j^* \leq \text{or} \geq 0, \quad (j = 1, \dots, n) \quad (2.191)$$

- *Complementarity*: Either  $\mu_j^*$  or  $g_j(\mathbf{x}^*)$  must be zero depending on whether the  $j$ th constraint is inactive or not:
  - if inactive, then  $g_j(\mathbf{x}^*) \neq 0$ , but  $\mu_j^* = 0$ ;
  - if active, then  $\mu_j^* \neq 0$ , but  $g_j(\mathbf{x}^*) = 0$ .

Combining both cases, we get

$$(\boldsymbol{\mu}^*)^T \mathbf{g}(\mathbf{x}^*) = 0 \quad (2.192)$$

We then introduce the following theorem regarding the KKT conditions and the solutions of the primal and dual problems.

**Theorem:**

- **Necessary conditions:** If  $\mathbf{x}^*$  and  $\{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$  are solutions for the respective primal and dual problems with strong duality, then they satisfy all KKT conditions.
- **Sufficient conditions:** If  $\mathbf{x}^*$  and  $\{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$  satisfy all KKT conditions, then they are solutions for the respective primal and dual problems, and strong duality holds, when the primal objective function  $f_p(\mathbf{x}^*)$  and the inequality constraints  $g_j(\mathbf{x})$  are convex, and equality constraints  $h_j(\mathbf{x})$  are linear.

**Proof:**

- **Necessary conditions:** If  $\mathbf{x}^*$  and  $\{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$  are respectively the primal and dual solutions, then the primal and dual feasibilities must be satisfied, and so is stationarity as  $\mathbf{x}^*$  as the optimal solution satisfies Eq. (2.183). In addition, we also have  $\mathbf{h}(\mathbf{x}^*) = \mathbf{0}$ . To show complementarity also holds, consider

$$\begin{aligned} d^* &= f_d(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \\ &= \inf_{\mathbf{x}} [f_p(\mathbf{x}) - (\boldsymbol{\lambda}^*)^T \mathbf{h}(\mathbf{x}) - (\boldsymbol{\mu}^*)^T \mathbf{g}(\mathbf{x})] \\ &\leq f_p(\mathbf{x}^*) - (\boldsymbol{\lambda}^*)^T \mathbf{h}(\mathbf{x}^*) - (\boldsymbol{\mu}^*)^T \mathbf{g}(\mathbf{x}^*) \\ &= f_p(\mathbf{x}^*) - (\boldsymbol{\mu}^*)^T \mathbf{g}(\mathbf{x}^*) \\ &\leq f_p(\mathbf{x}^*) = p^* \end{aligned} \quad (2.193)$$

The last inequality is due to Eq. (2.186). But as  $d^* = p^*$  due to the strong duality, the inequalities need to be equalities, i.e.,  $(\boldsymbol{\mu}^*)^T \mathbf{g}(\mathbf{x}^*) = 0$  and complementarity holds.

- **Sufficient conditions:** If  $\mathbf{x}^*$  and  $\{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$  satisfy the KKT conditions, then they are feasible for the respective primal and dual problems. Also, as  $f_p(\mathbf{x})$  and all  $g_j(\mathbf{x})$  are convex and  $h_j(\mathbf{x})$  are linear, Lagrangian in Eq. (2.182) as a sum of convex functions is also convex (see Section A.24), and stationarity in Eq. (2.189) indicates  $\mathbf{x}^*$  is a global minimum, therefore

$$\begin{aligned} d^* &= f_d(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = f_p(\mathbf{x}^*) - (\boldsymbol{\lambda}^*)^T \mathbf{h}(\mathbf{x}^*) - (\boldsymbol{\mu}^*)^T \mathbf{g}(\mathbf{x}^*) \\ &= f_p(\mathbf{x}^*) = p^* \end{aligned} \quad (2.194)$$

where the second line is due to the KKT conditions in Eqs. (2.190) and (2.192).

In summary, we see that KKT conditions are necessary and satisfied by all solutions of constrained optimization problems, and they can also be sufficient

for the solution of such problems if the primal problem further satisfies some additional conditions of convexity. In such a case, strong duality holds and the solution of the dual problem is the tightest lower bound of the primal, same as the solution of the primal.

Consider, as an example of the inequality constrained minimization problems, the algorithm of *support vector machine (SVM)* for binary classification (Section 13). To find the decision boundary that optimally separates two classes, we need to solve the following minimization problem constrained by a set of inequalities imposed by the data samples  $\{(\mathbf{x}_n, y_n), n = 1, \dots, N\}$  in the training set:

$$\begin{cases} \min: & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s. t.}: & y_n(\mathbf{x}_n^T \mathbf{w} + b) \geq 1 \quad (n = 1, \dots, N) \end{cases} \quad (2.195)$$

where  $\mathbf{w} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n$ . We note that the quadratic objective function is convex and so are the affine inequality constraints, i.e., this is a convex problem. The solution  $\mathbf{w}$  with minimum norm  $\|\mathbf{w}\|^2$  subject to the inequality constraints can be found by solving the dual problem in terms of the Lagrange multipliers in  $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_N]^T$ , subject to certain inequality constraints (detailed derivation in Section 13):

$$\begin{cases} \max: & \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m (\mathbf{x}_n^T \mathbf{x}_m) \\ \text{s. t.}: & \sum_{n=1}^N \alpha_n y_n = \mathbf{y}^T \boldsymbol{\alpha} = 0, \quad \alpha_n \geq 0 \quad (n = 1, \dots, N) \end{cases} \quad (2.196)$$

As the dual function can be written a quadratic form:

$$\boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{Q} \boldsymbol{\alpha} \quad (2.197)$$

where  $\mathbf{Q}$  is an  $N \times N$  symmetric matrix of which the component in the  $m$ th row and  $n$ th column is  $Q(m, n) = y_m y_n \mathbf{x}_m^T \mathbf{x}_n$ , this is a quadratic programming problem, which can be solved by the interior point algorithm to be discussed in Section 2.2.7.

#### 2.2.4 Linear Programming (LP)

The basic problem in linear programming (LP) is to minimize/maximize a linear objective function  $f(x_1, \dots, x_N) = \sum_{i=1}^N c_i x_i$  of  $N$  variables  $x_1, \dots, x_N$  under a set of  $M$  linear constraints:

$$\begin{aligned} \max: & & f(x_1, \dots, x_N) &= \sum_{i=1}^N c_i x_i \\ \text{s. t.}: & & \begin{cases} \sum_{i=1}^N a_{1i} x_i \leq b_1 \\ \dots \dots \dots \\ \sum_{i=1}^N a_{Mi} x_i \leq b_M \\ x_1 \geq 0, \dots, x_N \geq 0 \end{cases} & & (2.198) \end{aligned}$$

The LP problem can be more concisely represented in matrix form:

$$\begin{aligned} \max: & \quad \mathbf{c}^T \mathbf{x} \\ \text{s. t.}: & \quad \begin{cases} \mathbf{A}\mathbf{x} - \mathbf{b} \leq \mathbf{0} \\ \mathbf{x} \geq \mathbf{0} \end{cases} \end{aligned} \quad (2.199)$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_M \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{M1} & \cdots & a_{MN} \end{bmatrix} \quad (2.200)$$

and the smaller or greater than sign in a vector inequality should be understood as an element-wise relationship between the corresponding elements of the vectors on both sides.

For example, the objective function could be the total profit, the constraints could be some limited resources. Specifically,  $x_1, \dots, x_N$  may represent the quantities of  $N$  different types of products,  $c_1, \dots, c_N$  represent their unit prices, and  $a_{ij}$  represent the consumption of the  $i$ th resource by the  $j$ th product, the goal is to maximize the profit  $f(x_1, \dots, x_N)$  subject to the constraints imposed by the limited resources  $b_1, \dots, b_M$ .

Here all variables are assumed to be non-negative. If there exists a variable that is not restricted, it can be eliminated. Specifically, we can solve one of the constraining equations for the variable and use the resulting expression of the variable to replace all its appearances in the problem. For example:

$$\begin{aligned} \max: & \quad f(x_1, x_2, x_3) = 2x_1 - x_2 + 3x_3 \\ \text{s. t.}: & \quad \begin{cases} x_1 - 2x_2 + x_3 = 3 \\ 3x_1 - x_2 + 4x_3 = 10 \\ x_2 \geq 0, \quad x_3 \geq 0 \end{cases} \end{aligned} \quad (2.201)$$

Solving the first constraining equation for the free variable  $x_1$ , we get  $x_1 = 3 + 2x_2 - x_3$ . Substituting this into the objective function and the other constraint, we get  $2x_1 - x_2 + x_3 = 3x_2 + x_3 + 6$  and  $5x_2 + x_3 = 1$ , the problem can be reformulated as:

$$\begin{aligned} \max: & \quad f(x_2, x_3) = 3x_2 + x_3 + 6 \\ \text{s. t.}: & \quad \begin{cases} 5x_2 + x_3 = 1 \\ x_2 \geq 0, \quad x_3 \geq 0 \end{cases} \end{aligned} \quad (2.202)$$

Given the primal LP problem above, we can further find its dual problem. We first construct the Lagrangian of primal problem:

$$L(\mathbf{x}) = \mathbf{c}^T \mathbf{x} - \mathbf{y}^T (\mathbf{A}\mathbf{x} - \mathbf{b}) = (\mathbf{c} - \mathbf{A}^T \mathbf{y})^T \mathbf{x} + \mathbf{y}^T \mathbf{b} \quad (2.203)$$

where  $\mathbf{y} = [y_1, \dots, y_M]^T$  contains the Lagrange multipliers for the inequality constraints  $\mathbf{A}\mathbf{x} - \mathbf{b} \leq \mathbf{0}$ , and  $\mathbf{y} \geq \mathbf{0}$  according to Table 2.167. We define the dual

function as the maximum of  $L(\mathbf{x})$  over  $\mathbf{x}$ :

$$f_d(\mathbf{y}) = \max_{\mathbf{x}} L(\mathbf{x}) = \max_{\mathbf{x}} [\mathbf{c}^T \mathbf{x} - \mathbf{y}^T (\mathbf{A}\mathbf{x} - \mathbf{b})] = \max_{\mathbf{x}} [(\mathbf{c} - \mathbf{A}^T \mathbf{y})^T \mathbf{x} + \mathbf{y}^T \mathbf{b}] \quad (2.204)$$

To find the value of  $\mathbf{x}$  that maximizes  $L(\mathbf{x})$ , we set its gradient to zero and get:

$$\nabla_{\mathbf{x}} L(\mathbf{x}) = \nabla_{\mathbf{x}} [(\mathbf{c} - \mathbf{A}^T \mathbf{y})^T \mathbf{x} + \mathbf{y}^T \mathbf{b}] = \mathbf{c} - \mathbf{A}^T \mathbf{y} = \mathbf{0} \quad (2.205)$$

Substituting this into  $L_d(\mathbf{y}) = \max_{\mathbf{x}} L(\mathbf{x})$  we get:  $f_d(\mathbf{y}) = \mathbf{y}^T \mathbf{b}$ , which is the upper bound of  $f_p(\mathbf{x})$ , under the constraint that  $\mathbf{c} - \mathbf{A}^T \mathbf{y} \leq \mathbf{0}$  so that  $(\mathbf{c} - \mathbf{A}^T \mathbf{y})^T \mathbf{x} \leq 0$  contributes negatively to  $L(\mathbf{x})$  (as  $\mathbf{x} \geq 0$ ):

$$f_d(\mathbf{y}) = \mathbf{b}^T \mathbf{y} \geq L(\mathbf{x}) = \mathbf{c}^T \mathbf{x} - \mathbf{y}^T (\mathbf{A}\mathbf{x} - \mathbf{b}) \geq \mathbf{c}^T \mathbf{x} = f_p(\mathbf{x}) \quad (2.206)$$

This upper bound needs to be minimized with respect to  $\mathbf{y}$ , to become the tightest upper bound, i.e., the original primal problem of maximization in Eq. (2.199) has now been converted into the following dual problem of minimization:

$$\begin{cases} \max : & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} - \mathbf{b} \leq \mathbf{0}, \mathbf{x} \geq \mathbf{0} \end{cases} \iff \begin{cases} \min & \mathbf{b}^T \mathbf{y} \\ \text{s.t.} & \mathbf{A}^T \mathbf{y} - \mathbf{c} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0} \end{cases} \quad (2.207)$$

Following the same approach, we can also find the dual of a minimization primal problem:

$$\begin{cases} \min : & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} - \mathbf{b} \leq \mathbf{0}, \mathbf{x} \geq \mathbf{0} \end{cases} \iff \begin{cases} \max & \mathbf{b}^T \mathbf{y} \\ \text{s.t.} & \mathbf{A}^T \mathbf{y} - \mathbf{c} \leq \mathbf{0}, \mathbf{y} \leq \mathbf{0} \end{cases} \\ \iff \begin{cases} \max & -\mathbf{b}^T \mathbf{y} \\ \text{s.t.} & \mathbf{A}^T \mathbf{y} + \mathbf{c} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0} \end{cases} \quad (2.208)$$

Moreover, we can show that the dual of a dual problem is the original primal problem.

If either the primal or the dual is feasible and bounded, so is the other, and they form a strong duality, the solution  $d^*$  of the dual problem is the same as the solution  $p^*$  of the primal problem. Also, as the primal and dual problems are completely symmetric.

An inequality constrained LP problem can be converted to an equality constrained problem by introducing *slack variables*:

$$\sum_{i=1}^N a_i x_i \leq b \implies \sum_{i=1}^N a_i x_i + s = b, \quad (s \geq 0) \quad (2.209)$$

so that it can be reformulated to take the *standard form*:

$$\begin{aligned} \max: & \quad z = f(\mathbf{x}) = \mathbf{c}^T \mathbf{x} = \sum_{j=1}^N c_j x_j \\ \text{s. t.}: & \quad \begin{cases} \sum_{i=1}^N a_{1i} x_i + s_1 & & & & & = b_1 \\ \sum_{i=1}^N a_{2i} x_i & + s_2 & & & & = b_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \sum_{i=1}^N a_{Mi} x_i & & & & + s_M & = b_M \\ x_1 \geq 0, & \dots, & x_N \geq 0, & s_1 \geq 0, & \dots & s_M \geq 0 \end{cases} \end{aligned} \quad (2.210)$$



We further redefine the following:

- $\mathbf{x}$  is an  $N + M$  dimensional augmented variable vector that includes the  $M$  slack variables  $\{s_1, \dots, s_M\}$  as well as the  $N$  original variables  $\{x_1, \dots, x_N\}$ :

$$\mathbf{x} = [x_1, x_2, \dots, x_N, s_1, s_2, \dots, s_M]^T \quad (2.211)$$

- $\mathbf{A}$  is redefined as an  $M \times (N + M)$  augmented coefficient matrix that includes coefficients for both types of variables:

$$\mathbf{A} = \left[ \begin{array}{cccc|cccc} a_{11} & a_{12} & \cdots & a_{1N} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2N} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MN} & 0 & \cdots & 0 & 1 \end{array} \right]_{M \times (N+M)} = [ \mathbf{A}_{M \times N} \mid \mathbf{I}_{M \times M} ] \quad (2.212)$$

where  $\mathbf{A}_{M \times N}$  are the coefficients for the  $N$  original variables  $\{x_1, \dots, x_N\}$ , and the identity matrix  $\mathbf{I}_{M \times M} = [\mathbf{e}_1, \dots, \mathbf{e}_M]$  is for the unity coefficients of the  $M$  slack variables  $\{s_1, \dots, s_M\}$ , each of which appears in the  $M$  constraints only once.

Now the LP problem can be expressed in the standard form (original form on the left, with  $\mathbf{x}$  and  $\mathbf{A}$  redefined on the right):

$$\begin{array}{ll} \max: & \mathbf{c}^T \mathbf{x} \\ \text{s. t.} & \begin{cases} \mathbf{A}\mathbf{x} + \mathbf{s} = \mathbf{b} \\ \mathbf{x} \geq \mathbf{0}, \mathbf{s} \geq \mathbf{0} \end{cases} \end{array} \quad \text{or} \quad \begin{array}{ll} \max: & \mathbf{c}^T \mathbf{x} \\ \text{s. t.} & \begin{cases} \mathbf{A}\mathbf{x} = \mathbf{b} \\ \mathbf{x} \geq \mathbf{0} \end{cases} \end{array} \quad (2.213)$$

An LP problem can be viewed geometrically. If normalize  $\mathbf{c}$  so that  $\|\mathbf{c}\| = 1$ , then the objective function  $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$  becomes the projection of  $\mathbf{x}$  onto  $\mathbf{c}$ . Each of the  $M$  equality constraints in  $\mathbf{A}\mathbf{x} = \mathbf{b}$  represents a hyper-plane in the  $N$ -D vector space perpendicular to its normal direction  $\mathbf{a}_j = [a_{j1}, \dots, a_{jN}]^T$ :

$$\sum_{j=1}^N a_{ij} x_j = \mathbf{a}_j^T \mathbf{x} = b_i, \quad (i = 1, \dots, M) \quad (2.214)$$

Also, each of the  $N$  non-negativity conditions  $x_j \geq 0$  in  $\mathbf{x} \geq \mathbf{0}$  corresponds to a hyper-plane perpendicular to the  $j$ th standard basis vector  $\mathbf{e}_j$ . In general, in an  $N$ -D space, if none of the hyper-planes is parallel to any others, then no more than  $N$  hyper-planes intersect at one point. For example, in a 2-D or 3-D space, two straight lines or three surfaces intersect at a point. Here for the LP problem in the  $N$ -D space, the total number of intersection points formed by the  $N + M$  hyper-planes is

$$C_{M+N}^N = C_{M+N}^M = \frac{(M+N)!}{N! M!} \quad (2.215)$$

The polytope enclosed by these  $N + M$  hyper-planes is called the feasible region, in which the optimal solution must lie. The vertices of the polytopic feasible region are a subset of the  $C_{M+N}^N$  intersections.

### Fundamental theorem of linear programming

The optimal solution  $\mathbf{x}^*$  of a linear programming problem formulated above is either a vertex of the polytopical feasible region  $P$ , or lies on a hyper-surface of the polytope, on which all points are optimal solutions.

#### Proof:

Assume the optimal solution  $\mathbf{x}^* \in P$  is interior to the polytopical feasible region  $P$ , then there must exist some  $\epsilon > 0$  such that the hyper-sphere of radius  $\epsilon$  centered at  $\mathbf{x}^*$  is inside  $P$ . Evaluate the objective function at a point on the hyper-sphere

$$\mathbf{x}' = \mathbf{x}^* + \epsilon \mathbf{c}/\|\mathbf{c}\| \in P \quad (2.216)$$

we get

$$\begin{aligned} f(\mathbf{x}') &= \mathbf{c}^T \mathbf{x}' = \mathbf{c}^T (\mathbf{x}^* + \epsilon \mathbf{c}/\|\mathbf{c}\|) \\ &= \mathbf{c}^T \mathbf{x}^* + \epsilon \mathbf{c}^T \mathbf{c}/\|\mathbf{c}\| = \mathbf{c}^T \mathbf{x}^* + \epsilon \|\mathbf{c}\| > \mathbf{c}^T \mathbf{x}^* = f(\mathbf{x}^*) \end{aligned} \quad (2.217)$$

i.e.,  $\mathbf{x}^*$  cannot be the optimal solution as assumed. This contradiction indicates that an optimal solution  $\mathbf{x}^*$  must be on the surface of  $P$ , either at one of its vertices or on one of its surfaces. Q.E.D.

Based on this theorem, the optimization of a linear programming problem could be solved by exhaustively checking each of the  $C_{N+M}^N$  intersections formed by  $N$  of the  $N + M$  hyper-planes to find (a) whether it is feasible (satisfying all  $N + M$  constraints), and, if so, (b) whether the objective function  $f(\mathbf{x})$  is maximized at the point. Specifically, we choose  $N$  of the  $N + M$  equations for the constraints and solve this  $N$ -equation and  $N$ -unknown linear system to get the intersection point of  $N$  corresponding hyper-planes. This brute-force method is most straight forward, but the computational complexity is high when  $N$  and  $M$ , and therefore  $C_{N+M}^N$ , become large.

#### Example:

$$\begin{aligned} \max: \quad & f_p(x_1, x_2) = 2x_1 + 3x_2 \\ \text{s. t.}: \quad & \begin{cases} 2x_1 + x_2 \leq 18 \\ 6x_1 + 5x_2 \leq 60 \\ 2x_1 + 5x_2 \leq 40 \\ x_1 \geq 0, \quad x_2 \geq 0 \end{cases} \quad \text{or} \quad \begin{cases} \max: \quad f_p(\mathbf{x}) = \mathbf{c}^T \mathbf{x} \\ \text{s. t.}: \quad \begin{cases} \mathbf{A}\mathbf{x} - \mathbf{b} \leq \mathbf{0} \\ \mathbf{x} \geq \mathbf{0} \end{cases} \end{cases} \end{aligned}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 18 \\ 60 \\ 40 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 2 & 1 \\ 6 & 5 \\ 2 & 5 \end{bmatrix}$$

This problem has  $N = 2$  variables with the same number of non-negativity constraints and  $M = 3$  linear inequality constraints. The  $n + m = 5$  straight lines form  $C_{N+M}^N = C_5^2 = 10$  intersections, out of which 5 are the vertices of the polygonal feasible region satisfying all constraints. The value of the objective

function  $f_p(x_1, x_2) = \mathbf{c}^T \mathbf{x} = c_1x_1 + c_2x_2$  is proportional to the projection of the 2-D variable vector  $\mathbf{x} = [x_1, x_2]^T$  onto the coefficient vector  $\mathbf{c} = [2, 3]^T$ . The goal of this linear programming problem is to find a point  $\mathbf{x}$  inside the polygonal feasible region with maximum  $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$  (the projection of  $\mathbf{x}$  onto vector  $\mathbf{c}$  if  $\|\mathbf{c}\| = 1$ ). Geometrically, this can be understood as to push the hyper-plane  $\mathbf{c}^T \mathbf{x} = z$ , here a line in 2-D space, its normal direction  $\mathbf{c}$ , away from the origin along to eventually arrive at the vertex of the polytopic feasible region farthest away from the origin.

This LP problem can be converted into the standard form:

$$\begin{aligned} \max: \quad & f_p(x_1, x_2) = 2x_1 + 3x_2 \\ \text{s. t.}: \quad & \begin{cases} 2x_1 + x_2 \leq 18 \\ 6x_1 + 5x_2 \leq 60 \\ 2x_1 + 5x_2 \leq 40 \\ x_1 \geq 0, x_2 \geq 0 \end{cases} \end{aligned} \implies \begin{aligned} \max: \quad & f_p(x_1, x_2) = 2x_1 + 3x_2 \\ \text{s. t.}: \quad & \begin{cases} 2x_1 + x_2 + s_1 = 18 \\ 6x_1 + 5x_2 + s_2 = 60 \\ 2x_1 + 5x_2 + s_3 = 40 \\ x_1 \geq 0, x_2 \geq 0, s_1 \geq 0, s_2 \geq 0, s_3 \geq 0 \end{cases} \end{aligned}$$

Listed in the table below are the  $C_{M+N}^N = C_5^2 = 10$  intersections  $\mathbf{x}$ , called basic solutions, together with the corresponding slack variables  $s_1, s_2, s_3$  and the objective function value  $f(\mathbf{x})$  (proportional to the projection of  $\mathbf{x}$  onto  $\mathbf{c}$ ). We see that out of the ten basic solutions, five are feasible (vertices of the polytopic feasible region) with all  $M + N = 5$  variables taking non-negative values, while the remaining five are not feasible, as some of the slack variables are negative, and thereby violating the constraints. Out of the five feasible solutions, the one at  $\mathbf{x}^* = (5, 6)$  is optimal with maximum objective function value  $f_p(x_1, x_2) = 2x_1 + 3x_2 = 2 \times 5 + 3 \times 6 = 28$ .

	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	feasibility	$\mathbf{c}^T \mathbf{x} / \ \mathbf{c}\  = 1$
1	5	6	2	0	0	feasible	28/7.77
2	6.25	5.5	0	-5	0	infeasible	29/8.04
3	7.5	3	0	0	10	feasible	24/6.66
4	20	0	-2	-60	20	infeasible	40/11.1
5	10	0	-2	0	20	infeasible	20/5.55
6	9	0	0	6	22	feasible	18/4.10
7	0	8	10	20	0	feasible	24/6.66
8	0	12	6	0	-20	infeasible	36/9.98
9	0	18	0	-30	-50	infeasible	54/14.99
10	0	0	18	60	40	feasible	0/0.00

This primal maximization problem can be converted into the dual minimization problem:

$$\begin{aligned} \min: \quad & f_d(y_1, y_2, y_3) = 18y_1 + 60y_2 + 40y_3 \\ \text{s. t.}: \quad & \begin{cases} 2y_1 + 6y_2 + 2y_3 \geq 2 \\ y_1 + 5y_2 + 5y_3 \geq 3 \\ y_1 \geq 0, y_2 \geq 0, y_3 \geq 0 \end{cases} \end{aligned} \quad \text{or} \quad \begin{aligned} \min: \quad & f_d(\mathbf{y}) = \mathbf{b}^T \mathbf{y} \\ \text{s. t.}: \quad & \begin{cases} \mathbf{A}^T \mathbf{y} - \mathbf{c} \geq \mathbf{0} \\ \mathbf{y} \geq \mathbf{0} \end{cases} \end{aligned}$$

The five planes,  $m = 2$  from the constraining inequalities, and  $n = 3$  for the non-negative constraints  $y_1 \leq 0$ ,  $y_2 \leq 0$ , and  $y_3 \leq 0$  for the three variables, form 10 intersection points:

	$(y_1, y_2, y_3)$	feasibility	objective function
1	(1, 0, 0)	infeasible	18
2	(3, 0, 0)	feasible	54
3	(0, 1/3, 0)	infeasible	20
4	(0, 3/5, 0)	feasible	36
5	(0, 0, 1)	feasible	40
6	(0, 0, 3/5)	infeasible	24
7	(0, 1/5, 2/5)	feasible	28
8	(1/2, 0, 1/2)	feasible	29
9	(-2, 1, 0)	infeasible	24
10	(0, 0, 0)	infeasible	0

We see that at the feasible point  $(0, 1/5, 2/5)$ , the dual function  $f_d(y_1, y_2, y_3) = 18y_1 + 60y_2 + 40y_3$  is minimized to be 28, the same as the maximum of the primal function  $f_p(x_1, x_2) = 2x_1 + 3x_2$  at  $(5, 6)$ .

**Homework:**

Develop the code to find all  $C_{N+M}^N$  intersections formed by  $N + M$  given hyper-planes in an  $N$ -D space in terms of their equations  $\mathbf{A}_{M \times N} \mathbf{x}_{N \times 1} \leq \mathbf{b}_{M \times 1}$  and  $\mathbf{x} \geq \mathbf{0}$ , identify which of them are vertices of the polytope surrounded by these hyper-planes, and find the vertex corresponding to the optimal solution that maximizes  $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$  for a given  $\mathbf{c}$ .

### 2.2.5 The Simplex Algorithm

The simplex algorithm finds the optimal solution of a LP problem by an iterative process that traverses along a sequence of edges of the polytopical feasible region, starting at the origin and through a sequence of vertices  $\mathbf{x}$  with progressively greater objective value  $f(\mathbf{x})$ , until eventually reaching the optimal solution. By doing so, it avoids checking exhaustively all vertices of the feasible region for optimality.

We first consider the equality constraint  $\mathbf{A}\mathbf{x} = [\mathbf{A}_{M \times N} \mid \mathbf{I}_{M \times M}]\mathbf{x} = \mathbf{b}$  of the standard LP problem. This is an under-determined linear system of  $N + M$  variables but only  $M$  equations, only  $M$  of its  $N + M$  columns of the coefficient matrix  $\mathbf{A}$  are independent (assuming  $\text{rank}(\mathbf{A}) = M$ ). Initially we choose the  $M$  columns of the identity coefficient matrix  $\mathbf{I}$  as the independent columns, but subsequently we can choose any other  $M$  columns of  $\mathbf{A}$  as the independent columns and convert them into a standard basis vector by Gauss-Jordan elimination together with the corresponding variables in  $\mathbf{x}$ . The resulting equation  $\mathbf{A}\mathbf{x} = \mathbf{b}$  remains equivalent to the original one, i.e., the  $M$  equality constraints are always preserved and never violated in the process.

For convenience of discussion and without loss of generality, we could reorder

(not actually carried out) the  $N + M$  columns in  $\mathbf{A}$ , together with the corresponding variables in  $\mathbf{x}$ , so that the constraining equation would always takes the following form:

$$\mathbf{A}\mathbf{x} = [\mathbf{A}_n \ \mathbf{I}] \begin{bmatrix} \mathbf{x}_n \\ \mathbf{x}_b \end{bmatrix} = \mathbf{A}_n \mathbf{x}_n + \mathbf{I} \mathbf{x}_b = \mathbf{b} \quad (2.218)$$

Now the variables in the M-D vector  $\mathbf{x}_b$  corresponding to the  $M$  independent columns in  $\mathbf{I}$  are the *basic variables*, while the N-D vector  $\mathbf{x}_n$  corresponding to the remaining  $N$  dependent columns, now denoted by  $\mathbf{A}_n$ , are the *non-basic variables*. This equation will always hold if  $\mathbf{x}_n = \mathbf{0}$  and  $\mathbf{x}_b = \mathbf{b}$ . Such a solution is called a *basic solution* of the linear system  $\mathbf{A}\mathbf{x} = \mathbf{b}$ :

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_n \\ \mathbf{x}_b \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix} \quad (2.219)$$

Initially,  $\mathbf{A}_n$  is the coefficient matrix in the inequality constraint  $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ , and the corresponding non-basic variables  $\mathbf{x}_n = \mathbf{0}$  is actually the origin. But through the iteration, we will keep converting some other columns of  $\mathbf{A}$  into standard basis vectors  $\{\mathbf{e}_1, \dots, \mathbf{e}_M\}$ , the column vectors of  $\mathbf{I}$ , by Gauss-Jordan elimination. The variables in  $\mathbf{x}$  corresponding to the new standard basis vectors become the basic variables, while those corresponding to columns that become non-standard basis vectors become non-basis variables.

The basic solutions corresponding the  $C_{N+M}^N$  ways to choose  $M$  of the  $N + M$  columns of  $\mathbf{A}$  as independent are actually the intersections formed by any  $N$  of the  $N + M$  constraining hyper-planes in the N-D space. As noted before, only a subset of these  $C_{N+M}^N$  basic solutions satisfy the  $M$  constraints, and they are called *basic feasible solutions (BFS)*. The goal of the iterative process of the simplex algorithm is to find the optimal basic feasible solution that maximizes  $f(\mathbf{x})$  without exhausting all  $C_{N+M}^N$  possibilities. This is done by selecting the columns in such a way that the value of the objective function will always be maximally increased, until eventually we find the optimal solution  $\mathbf{x}^*$  at one of the vertex of the polytopical feasible region.

Specially, the implementation of the simplex algorithm is based on a tableau with  $N + M + 1$  columns and  $M + 1$  rows, initialized as below:

- The first  $M$  rows are for the coefficients in the equality constraints  $\sum_{j=1}^N a_{ij}x_j + s_i = b_i$  ( $i = 1, \dots, M$ ). An additional  $(M + 1)$ st row at the bottom is for the coefficients in the objective function  $f(\mathbf{x}) = \sum_{j=1}^N c_j x_j$ , which is zero initially at the origin  $x_1 = \dots = x_N = 0$ , i.e.,  $z - c_1 x_1 - \dots - c_N x_N = 0$ .
- The first  $N + M$  columns contain the coefficients for the  $N$  original variables  $\{x_1, \dots, x_N\}$  and the  $M$  slack variables  $\{s_1, \dots, s_M\}$ , and an additional  $(N + M + 1)$ st column is for the constants  $\mathbf{b} = [b_1, \dots, b_M]^T$  on the right-hand side of the  $M$  constraint equations. The last element of the column is the right-hand side of the objective equation  $z = f(\mathbf{x})$ , which is initially zero.

basic variables	$x_1$	$x_2$	$\cdots$	$x_N$	$s_1$	$s_2$	$\cdots$	$s_M$	basic solution
$s_1$	$a_{11}$	$a_{12}$	$\cdots$	$a_{1N}$	1	0	$\cdots$	0	$b_1$
$s_2$	$a_{21}$	$a_{22}$	$\cdots$	$a_{2N}$	0	1	$\cdots$	0	$b_2$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$s_M$	$a_{M1}$	$a_{M2}$	$\cdots$	$a_{MN}$	0	0	$\cdots$	1	$b_M$
$z$	$-c_1$	$-c_2$	$\cdots$	$-c_M$	0	0	$\cdots$	0	0

(2.220)

At this initial stage of the iteration, the  $N$  original variables  $\mathbf{x}_n = [x_1, \dots, x_N]^T$  are the non-basic variables, while the  $M$  slack variables  $\mathbf{x}_b = [s_1, \dots, s_M]^T$  are the basic variables and their coefficients form an  $M \times M$  identity matrix  $\mathbf{I}$ , composed of the standard basis vectors. The corresponding feasible basic solution is simply

$$\mathbf{x}_b = [s_1, \dots, s_M]^T = \mathbf{b}, \quad \mathbf{x}_n = [x_1, \dots, x_N]^T = \mathbf{0} \quad (2.221)$$

that satisfies all the constraints  $\mathbf{Ax} = \mathbf{A}_n \mathbf{x}_n + \mathbf{I} \mathbf{x}_b = \mathbf{b}$ .

In each of the subsequent iterations, we will select one of the non-basic variables, called the *entering variable*, to replace one of the basic variables, called the *leaving variable*, in such a way that the value of the objective function value  $z = f(\mathbf{x})$  in the last row will be maximally increased, while all constraints remain satisfied. Here are the steps in each iteration:

- **Selection of the entering variable**

This section is based on the maximization of  $z = f(\mathbf{x})$ . We select  $x_j$  in the  $j$ th column of the tableau if  $-c_j$  is most negative, i.e.,  $x_j$  is most heavily weighted by  $c_j = \max\{c_1, \dots, c_N\}$ , so that it will increase  $z = \sum_{j=1}^N c_j x_j$  more than any other  $x_k$  ( $k \neq j$ ).

- **Selection of the leaving variable**

This selection is based on the constraints imposed on the selected entering variable  $x_j$ . In general, the restriction on  $x_j$  set by the  $k$ th constraint  $\sum_{j=1}^N a_{kj} x_j \leq b_k$  is  $x_j \leq b_k/a_{kj}$  when  $a_{kj} > 0$ . If  $b_i/a_{ij} = \min\{b_k/a_{1j}, \dots, b_k/a_{Mj}\}$ , then the  $i$ th constraint is most restrictive on  $x_j$ , we will therefore select the corresponding basic variable  $s_i$  as the leaving variable, i.e., it becomes a non-basic variable to be set to zero, so that  $x_j$  can be maximally increased without violating the constraints. If all  $a_{kj} < 0$  for all  $k = 1, \dots, M$ , variable  $x_j$  is not bounded.

- **Gauss-Jordan elimination based on Pivoting**

To convert the entering variable  $x_j$  to a basic variable to replace the leaving variable  $s_i$ , we need to turn the corresponding  $j$ th column into a standard basis vector  $\mathbf{e}_i$ . This is realized by pivoting on  $a_{ij}$  in the following steps:

1. Divide the pivot row  $\mathbf{r}_i$  by  $a_{ij}$ :  $\mathbf{r}_i \leftarrow \mathbf{r}_i/a_{ij}$ . Now  $a_{ij} = 1$ .

2. Subtract  $a_{kj} \mathbf{r}_i$  from the  $k$ th row:  $\mathbf{r}_k \leftarrow \mathbf{r}_k - a_{kj} \mathbf{r}_i$  so that  $a_{kj} = 0$  for all  $k = 1, \dots, M+1$ ,  $k \neq i$ , including the last row  $\mathbf{r}_{M+1}$ .

Now the  $j$ th column corresponding to the entering variable  $x_j$  becomes a standard basis vector  $\mathbf{e}_i$ , i.e.,  $x_j$  becomes a basic variable that takes the value of  $b_i$ , and the column corresponding to the leaving variable  $s_i$  is no longer a standard basis vector, and  $s_i = 0$  as it is now a non-basic variable.

As Gauss-Jordan elimination converts the linear equations to a set of equivalent equations, the constraints remain satisfied through out the process. Although the membership of the basic and non-basic variable groups keeps changing in the iteration, so long as  $\mathbf{x}_b = \mathbf{b}$  and  $\mathbf{x}_n = \mathbf{0}$ , the  $M$  constraint equations  $\mathbf{Ax} = \mathbf{A}_n\mathbf{x}_n + \mathbf{Ix}_b = \mathbf{b}$  always holds.

This iterative process keeps replacing the slack variables  $\{s_1, \dots, s_M\}$  (the basic variables initially) by the original variables  $\{x_1, \dots, x_N\}$  (the non-basic variables initially), one at a time, until all original variables become basic variables and all elements in the last row are non-negative.

The final result can be read out directly from the tableau. The variables corresponding to the  $M$  standard basis vectors  $\mathbf{e}_i$  ( $i = 1, \dots, M$ ) are the final basic variables that take the values in  $\mathbf{b}$  in the right-most column of the tableau. They form the optimal basic solution, with the maximum of the objective function  $z = f(\mathbf{x})$  given by the last element also in the last column. The remaining  $n$  variables corresponding to non-standard columns are non-basic variables that take the value zero. When  $M > N$  (more constraints than variables), some of the slack variables may remain in the basic variable group taking non-zero values; when  $M < N$ , some of the original variable may be in the non-basic group taking the value zero.

**Example:**

We re-solve the LP problem considered in the previous examples, now in the standard form:

$$\begin{aligned} \text{max: } & f(\mathbf{x}) = 2x_1 + 3x_2 \\ \text{s. t.: } & \begin{cases} 2x_1 + x_2 + s_1 = 18 \\ 6x_1 + 5x_2 + s_2 = 60 \\ 2x_1 + 5x_2 + s_3 = 40 \\ x_1 \geq 0, x_2 \geq 0, s_1 \geq 0, s_2 \geq 0, s_3 \geq 0 \end{cases} \end{aligned}$$

The standard form is further converted to a tableau as shown below. The left-most column indicates the basic variables, the next  $N = 2$  columns are for the variables  $x$  and  $y$ , the next  $M = 3$  columns are for the slack variables  $u$ ,  $v$ , and  $w$ , the right most column is for the constants  $b_1$ ,  $b_2$ , and  $b_3$  on the right-hand side of the equations.

	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	$b$
$s_1$	2	1	1	0	0	18
$s_2$	6	5	0	1	0	60
$s_3$	2	5	0	0	1	40
$z$	-2	-3	0	0	0	0

In this initial state, the basic variables are  $s_1 = 18$ ,  $s_2 = 60$ , and  $s_3 = 40$ , the non-basic variables are  $x_1 = x_2 = 0$ . The corresponding basic feasible solution is at the origin.

- Select column  $j = 2$  as the pivot column, as  $-c_2 = -3 < -c_1$  is the most negative coefficient.
- Select  $\mathbf{r}_3$  as the pivot row, as  $b_3/a_{32} = 40/5 = 8 = \min\{18/1 = 18, 60/5 = 12, 40/5 = 8\}$ .
- Divide pivot row  $\mathbf{r}_i = \mathbf{r}_3 = [2, 5, 0, 0, 1, 40]$  by the pivot element  $a_{ij} = 5$  to get  $\mathbf{r}_i = \mathbf{r}_3 = [0.4, 1, 0, 0, 0.2, 8]$ .
- Subtract  $a_{kj}\mathbf{r}_k = a_{k2}\mathbf{r}_k$  from row  $\mathbf{r}_k$ , so that  $a_{kj} = a_{k2} = 0$ , ( $k = 1, \dots, M + 1 = 4, k \neq i$ ), including the last row  $\mathbf{r}_4$ .

	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	$b$
$s_1$	1.6	0	1	0	-0.2	10
$s_2$	4	0	0	1	-1	20
$x_2$	0.4	1.0	0	0	0.2	8
$z$	-0.8	0	0	0	0.6	24

The entering variable  $x_j = x_2$  becomes a basic variable, and the leaving variable  $s_i = s_3$  becomes a non-basic variable, replaced by new basic variable  $x_2$ . The corresponding basic feasible solution is at  $x_1 = 0$ ,  $x_2 = b_3 = 8$ , with  $s_1 = 10$ ,  $s_2 = 20$ ,  $s_3 = 0$ . The objective function value is  $z = \mathbf{c}^T \mathbf{x} = 24$ .

- Select column  $j = 1$  as the pivot column, as  $-c_1 = -0.8$  is the most (only) negative coefficient in the last row.
- Select row  $i = 2$  as the pivot row, as the ratio  $b_2/a_{21} = 20/4 = 5 = \min\{10/1.6 = 6.25, 20/4 = 5, 8/0.4 = 20\}$ .
- Divide pivot row  $\mathbf{r}_i = \mathbf{r}_2 = [4, 0, 0, 1, -1, 20]$  by the pivot element  $a_{ij} = a_{21} = 4$  to get  $\mathbf{r}_i = \mathbf{r}_2 = [1, 0, 0, 0.25, -0.25, 5]$ .
- Subtract  $a_{kj}\mathbf{r}_k = a_{k1}\mathbf{r}_k$  from row  $\mathbf{r}_k$ , so that  $a_{kj} = a_{k1} = 0$ , ( $k = 1, \dots, M + 1 = 4$ ), including the last row  $\mathbf{r}_4$ .

	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	$b$
$s_1$	0	0	1	-0.4	0.2	2
$x_1$	1	0	0	0.25	-0.25	5
$x_2$	0	1	0	-0.1	0.3	6
$z$	0	0	0	0.2	0.4	28

The entering variable  $x_1$  becomes a basic variable, and the leaving variable  $s_1$  becomes a non-basic variable, replaced by the new basic variable  $x_1$ . The corresponding basic feasible solution is at  $x_1 = b_2 = 5$ ,  $x_2 = b_3 = 6$ , with  $s_1 = 2$ ,  $s_2 = s_3 = 0$ . The objective function value is  $z = \mathbf{c}^T \mathbf{x} = 28$ . Now that both  $x_1$  and  $x_2$  have become basic variables, the optimal solution has been obtained.



This problem has  $C_{N+M}^N = C_5^2 = 10$  basic solutions, corresponding to the same number of intersections, out of which five are feasible, as previously obtained. The simplex method finds three of these feasible solutions, starting from  $x_1 = x_2 = 0$  at the origin with  $z = 0$ , through the vertex at  $x_1 = 0$  and  $x_2 = 8$  with  $z = 24$ , to the optimal solution at  $x_1 = 5$  and  $x_2 = 6$  with  $z = 28$ .

The Matlab code for implementing the simplex algorithm is listed below. The function takes input including matrix  $\mathbf{A}$  as well as the vectors  $\mathbf{b}$  and  $\mathbf{c}$ , and generates the optimal solution  $\mathbf{x}^*$ , the corresponding maximum value of the objective function  $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}^*$ , together with the values of the slack variable  $\mathbf{s}$ .

```
function [x s z] = MySimplex(A,b,c)
    [m n]=size(A)           % coefficient matrix for variables x
    x=zeros(n,1);          % initial zero values for x
    s=zeros(m,1);
    A=[A eye(m) b; -c zeros(1,m+1)]; % initialization of tableau

    [cmin, pc]=min( A(m+1,1:n+m) ); % pc, index of the first pivot column
    it=0;
    while cmin<0           % iteration until all c's in last row are zero
        it=it+1;
        rmin=9e9;
        for i=1:m          % find pivot row
            if A(i,n+m+1)~=0 & A(i,pc)>0
                w=A(i,n+m+1)./A(i,pc);
                if w<rmin
                    pr=i;          % pr, index of the pivot row
                    rmin=w;
                end
            end
        end
        p=A(pr,pc);        % get the pivot element
        A(pr,:)=A(pr,+)/p; % modify pivot row
        for i=1:m+1        % modify all k+1 rows
            if i~=pr
                A(i,:)=A(i,:)-A(pr,:)*A(i,pc);
            end
        end
        z=A(m+1,n+m+1);    % maximum value of objective function so far

        [cmin, pc]=min( A(m+1,1:n+m) ); % pivot column of next iteration
    end
end
end
```

### 2.2.6 Quadratic Programming (QP)

A quadratic programming (QP) problem is to minimize a quadratic function subject to some equality and/or inequality constraints:

$$\begin{aligned} \text{min: } & f(\mathbf{x}) = \frac{1}{2}[\mathbf{x} - \mathbf{m}]^T \mathbf{Q}[\mathbf{x} - \mathbf{m}] = \frac{1}{2}\mathbf{x}^T \mathbf{Q}\mathbf{x} + \mathbf{c}^T \mathbf{x} + c \\ \text{s. t.: } & \mathbf{A}\mathbf{x} \leq \mathbf{b} \end{aligned} \quad (2.222)$$

where  $\mathbf{Q} = \mathbf{Q}^T$  is an  $N \times N$  positive definite matrix,  $\mathbf{A}$  an  $M \times N$  matrix, and  $\mathbf{b}$  is an M-D vector. Also,  $\mathbf{c} = -\mathbf{Q}\mathbf{m}$ ,  $c = \mathbf{m}^T \mathbf{Q}\mathbf{m}/2$ . Here the scalar constant  $c$  can be dropped as it does not play any role in the optimization. Note that  $\mathbf{x}^T \mathbf{Q}\mathbf{x}$  is a hyper elliptic paraboloid with the minimum at point  $\mathbf{m}$  in the N-D space.

We first consider the special case where the QP problem is only subject to equality constraints and we assume  $m \leq N$ , i.e., the number of constraints is smaller than the number of unknowns in  $\mathbf{x}$ . Then the solution  $\mathbf{x}$  has to satisfy  $\mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$ , i.e., it has to be on  $M$  hyper planes in the N-D space.

The Lagrangian function of the QP problem is:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T (\mathbf{A}\mathbf{x} - \mathbf{b}) = \frac{1}{2}\mathbf{x}^T \mathbf{Q}\mathbf{x} + \mathbf{c}^T \mathbf{x} + \boldsymbol{\lambda}^T (\mathbf{A}\mathbf{x} - \mathbf{b}) \quad (2.223)$$

To find the optimal solution, we first equate the derivatives of the Lagrangian with respect to both  $\mathbf{x}$  and  $\boldsymbol{\lambda}$  to zero:

$$\begin{aligned} \nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}) &= \mathbf{Q}\mathbf{x} + \mathbf{c} + \mathbf{A}^T \boldsymbol{\lambda} = \mathbf{0} \\ \nabla_{\boldsymbol{\lambda}} L(\mathbf{x}, \boldsymbol{\lambda}) &= \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0} \end{aligned} \quad (2.224)$$

These two equations can be combined and expressed in matrix form as:

$$\begin{bmatrix} \mathbf{Q} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{c} \\ \mathbf{b} \end{bmatrix} \quad (2.225)$$

We then solve this system of  $m + N$  equations to get the optimal solution  $\mathbf{x}^*$  and the corresponding  $\boldsymbol{\lambda}^*$ .

#### Example

$$\begin{aligned} \text{min: } & f(x_1, x_2) = x_1^2 + x_2^2 = \frac{1}{2}[x_1, x_2] \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{2}\mathbf{x}^T \mathbf{Q}\mathbf{x} \\ \text{s. t.: } & \mathbf{A}\mathbf{x} = [1, 1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1 + x_2 = \mathbf{b} = 1 \end{aligned}$$

where

$$\mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad \mathbf{A} = [1, 1], \quad \mathbf{c} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mathbf{b} = 1;$$

$$\begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Solving this equation we get the solution  $x_1^* = x_2^* = 0.5$  and  $\lambda^* = -1$ , at which the function  $f(x_1^*, x_2^*) = 0.5$  is minimized subject to  $x_1 + x_2 = 1$ .

If  $M = N$ , i.e., the number of equality constraints is the same as the number of variables, then the variable  $\mathbf{x}$  is uniquely determined by the linear system  $\mathbf{Ax} = \mathbf{b}$ , as the intersect of  $M = N$  hyper planes, independent of the objective function  $f(\mathbf{x})$ . Further if  $M > N$ , i.e., the system  $\mathbf{Ax} = \mathbf{b}$  is over constrained, and its solution does not exist in general. It is therefore more interesting to consider QP problems subject to both equality and inequality constraints:

$$\begin{array}{ll} \text{min:} & f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{s. t.:} & \mathbf{Ax} \leq \mathbf{b} \end{array}$$

### 2.2.7 Interior Point Methods

Consider the a general constrained optimization problem:

$$\begin{array}{ll} \text{min:} & f(\mathbf{x}) \\ \text{s. t.:} & \begin{cases} \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \text{ or } \mathbf{g}(\mathbf{x}) \geq \mathbf{0} \\ \mathbf{x} \geq \mathbf{0} \end{cases} \end{array} \quad (2.226)$$

By introducing slack variables  $\mathbf{s} \geq \mathbf{0}$ , all inequality constraints can be converted into equality constraints:

$$\mathbf{g}(\mathbf{x}) \leq \mathbf{0} \implies \mathbf{g}(\mathbf{x}) + \mathbf{s} = \mathbf{0}, \quad \mathbf{g}(\mathbf{x}) \geq \mathbf{0} \implies \mathbf{g}(\mathbf{x}) - \mathbf{s} = \mathbf{0} \quad (2.227)$$

which can then be combined with the equality constraints and still denoted by  $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ , while the slack variables  $\mathbf{s}$  are also combined with the original variables and still denoted by  $\mathbf{x}$ . We assume there are in total  $M$  equality constraints and  $N$  variables. Now the optimization problem can be reformulated as:

$$\begin{array}{ll} \text{min:} & f(\mathbf{x}) \\ \text{s. t.:} & \begin{cases} \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ \mathbf{x} \geq \mathbf{0} \end{cases} \end{array} \quad (2.228)$$

The Lagrangian is

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) - \boldsymbol{\mu}^T \mathbf{x} \quad (2.229)$$

The KKT conditions are

$$\begin{cases} \nabla_{\mathbf{x}}L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{g}_f(\mathbf{x}) + \mathbf{J}_h^T(\mathbf{x})\boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{0} & \text{(stationarity)} \\ \mathbf{h}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{x} \geq \mathbf{0} & \text{(primal feasibility)} \\ \boldsymbol{\mu} \geq \mathbf{0} & \text{(dual feasibility)} \\ \mu_j x_j = 0, \quad (j = 1, \dots, N) \text{ or } \mathbf{XM}\mathbf{1} = \mathbf{0} & \text{(complementarity)} \end{cases} \quad (2.230)$$

where

$$\mathbf{X} = \begin{bmatrix} x_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & x_n \end{bmatrix}_{N \times N}, \quad \mathbf{M} = \begin{bmatrix} \mu_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mu_N \end{bmatrix}_{N \times N} \quad (2.231)$$

and  $\mathbf{J}_f(\mathbf{x})$  is the Jacobian matrix of  $\mathbf{h}(\mathbf{x})$ :

$$\mathbf{J}_h(\mathbf{x}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \cdots & \frac{\partial h_m}{\partial x_n} \end{bmatrix}_{M \times N} \quad (2.232)$$

To simplify the problem, the non-negativity constraints can be removed by introducing an indicator function:

$$I(x) = \begin{cases} 0 & x \geq 0 \\ \infty & x < 0 \end{cases} \quad (2.233)$$

which, when used as a cost function, penalizes any violation of the non-negativity constraint  $x \geq 0$ . Now the optimization problem can be reformulated as shown below without the non-negative constants  $\mathbf{x} \geq \mathbf{0}$ :

$$\begin{aligned} \min: & f(\mathbf{x}) + \sum_{i=1}^N I(x_i) \\ \text{s. t.}: & \mathbf{h}(\mathbf{x}) = \mathbf{0} \end{aligned} \quad (2.234)$$

However, as the indicator function is not smooth and therefore not differentiable, it is approximated by the *logarithmic barrier function* which approaches  $I(x)$  when the parameter  $t > 0$  approaches infinity:

$$-\frac{1}{t} \ln(x) \xrightarrow{t \rightarrow \infty} I(x) \quad (2.235)$$

Now the optimization problem can be written as:

$$\begin{aligned} \min: & f(\mathbf{x}) - \frac{1}{t} \sum_{i=1}^N \ln x_i \\ \text{s. t.}: & \mathbf{h}(\mathbf{x}) = \mathbf{0} \end{aligned} \quad (2.236)$$

The Lagrangian is:

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) - \frac{1}{t} \sum_{j=1}^N \ln x_j \quad (2.237)$$

and its gradient is:

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{g}_f(\mathbf{x}) + \mathbf{J}_h^T(\mathbf{x}) \boldsymbol{\lambda} - \frac{1}{t} \mathbf{X}^{-1} \mathbf{1} \quad (2.238)$$

We define

$$\boldsymbol{\mu} = \frac{\mathbf{x}}{t} = \frac{1}{t} \mathbf{X}^{-1} \mathbf{1}, \quad \text{i.e.} \quad \mathbf{X} \boldsymbol{\mu} = \mathbf{X} \mathbf{M} \mathbf{1} = \frac{\mathbf{1}}{t} \quad (2.239)$$

and combine this equation with the KKT conditions of the new optimization problem to get

$$\begin{cases} \mathbf{g}_f(\mathbf{x}) + \mathbf{J}_h^T(\mathbf{x})\boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{0} & \text{(stationarity)} \\ \mathbf{h}(\mathbf{x}) = \mathbf{0} & \text{(primal feasibility)} \\ \mathbf{X}\mathbf{M}\mathbf{1} - \mathbf{1}/t = \mathbf{0} & \text{(complementarity)} \end{cases} \quad (2.240)$$

The third condition is actually from the definition of  $\boldsymbol{\mu}$  given above, but here it is labeled as complementarity, so that these conditions can be compared with the KKT conditions of the original problem. We see that the two sets of KKT conditions are similar to each other, with the following differences:

- The non-negativity  $\mathbf{x} \geq \mathbf{0}$  in the primal feasibility of the original KKT is dropped;
- Consequently the inequality  $\boldsymbol{\mu} \geq \mathbf{0}$  in the dual feasibility of the original KKT conditions is also dropped;
- There is an extra term  $\mathbf{1}/t$  in complementarity which will vanish when  $t \rightarrow \infty$ .

The optimal solution that minimizes the objective function  $f(\mathbf{x})$  can be found by solving the simultaneous equations in the modified KKT conditions (with no inequalities) given above. To do so, we first express the equations in the KKT conditions above as a nonlinear equation system  $\mathbf{F}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{0}$ , and find its Jacobian matrix  $\mathbf{J}_F$  composed of the partial derivatives of the function  $\mathbf{F}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$  with respect to each of the three variables  $\mathbf{x}$ ,  $\boldsymbol{\lambda}$ , and  $\boldsymbol{\mu}$ :

$$\mathbf{F}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \begin{bmatrix} \mathbf{g}_f(\mathbf{x}) + \mathbf{J}_h^T(\mathbf{x})\boldsymbol{\lambda} - \boldsymbol{\mu} \\ \mathbf{h}(\mathbf{x}) \\ \mathbf{X}\mathbf{M}\mathbf{1} - \mathbf{1}/t \end{bmatrix}, \quad \mathbf{J}_F = \begin{bmatrix} \mathbf{W}(\mathbf{x}) & \mathbf{J}_h^T(\mathbf{x}) & -\mathbf{I} \\ \mathbf{J}_h(\mathbf{x}) & \mathbf{0} & \mathbf{0} \\ \mathbf{M} & \mathbf{0} & \mathbf{X} \end{bmatrix} \quad (2.241)$$

where

$$\begin{aligned} \mathbf{W}(\mathbf{x}) &= \nabla_{\mathbf{x}} [\mathbf{g}_f(\mathbf{x}) + \mathbf{J}_h^T(\mathbf{x})\boldsymbol{\lambda} - \boldsymbol{\mu}] \\ &= \nabla_{\mathbf{x}} \mathbf{g}_f(\mathbf{x}) - \nabla_{\mathbf{x}} \left[ \sum_{i=1}^M \lambda_i \frac{\partial h_i}{\partial x_1}, \dots, \sum_{i=1}^M \lambda_i \frac{\partial h_i}{\partial x_N} \right]^T = \mathbf{H}_f(\mathbf{x}) - \sum_{i=1}^M \lambda_i \mathbf{H}_{h_i}(\mathbf{x}) \end{aligned}$$

is symmetric as the Hessian matrices  $\mathbf{H}_f$  and  $\mathbf{H}_{h_i}$ , ( $i = 1, \dots, M$ ) are symmetric.

We can now use the Newton-Raphson method to find the solution of the nonlinear equation  $\mathbf{F}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{0}$  iteratively:

$$\begin{bmatrix} \mathbf{x}_{n+1} \\ \boldsymbol{\lambda}_{n+1} \\ \boldsymbol{\mu}_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_n \\ \boldsymbol{\lambda}_n \\ \boldsymbol{\mu}_n \end{bmatrix} + \alpha \begin{bmatrix} \delta \mathbf{x}_n \\ \delta \boldsymbol{\lambda}_n \\ \delta \boldsymbol{\mu}_n \end{bmatrix} \quad (2.243)$$

where  $\alpha$  is a parameter that controls the step size and  $[\delta \mathbf{x}_n, \delta \boldsymbol{\lambda}_n, \delta \boldsymbol{\mu}_n]^T$  is the search direction (*Newton direction*), which can be found by solving the following

equation:

$$\begin{aligned} \mathbf{J}_F(\mathbf{x}_n, \boldsymbol{\lambda}_n, \boldsymbol{\mu}_n) \begin{bmatrix} \delta \mathbf{x}_n \\ \delta \boldsymbol{\lambda}_n \\ \delta \boldsymbol{\mu}_n \end{bmatrix} &= \begin{bmatrix} \mathbf{W}(\mathbf{x}_n) & \mathbf{J}_h^T(\mathbf{x}_n) & -\mathbf{I} \\ \mathbf{J}_h(\mathbf{x}_n) & \mathbf{0} & \mathbf{0} \\ \mathbf{M}_n & \mathbf{0} & \mathbf{X}_n \end{bmatrix} \begin{bmatrix} \delta \mathbf{x}_n \\ \delta \boldsymbol{\lambda}_n \\ \delta \boldsymbol{\mu}_n \end{bmatrix} \\ = -\mathbf{F}(\mathbf{x}_n, \boldsymbol{\lambda}_n, \boldsymbol{\mu}_n) &= - \begin{bmatrix} \mathbf{g}_f(\mathbf{x}_n) + \mathbf{J}_h^T(\mathbf{x}_n)\boldsymbol{\lambda}_n - \boldsymbol{\mu}_n \\ \mathbf{h}(\mathbf{x}_n) \\ \mathbf{X}_n \mathbf{M}_n \mathbf{1} - \mathbf{1}/t \end{bmatrix} \end{aligned} \quad (2.244)$$

To get the initial values for the iteration, we first find any point inside the feasible region  $\mathbf{x}_0$  and then  $\boldsymbol{\mu}_0 = \mathbf{x}_0/t$ , based on which we further find  $\boldsymbol{\lambda}_0$  by solving the first equation in the KKT conditions in Eq. (2.240).

Actually this equation system above can be separated into two subsystems, which are easier to solve. Pre-multiplying  $\mathbf{X}^{-1}$  to the third equation:

$$\mathbf{M}\delta \mathbf{x} + \mathbf{X}\delta \boldsymbol{\mu} = -\mathbf{X}\mathbf{M}\mathbf{1} + \mathbf{1}/t \quad (2.245)$$

we get

$$\mathbf{X}^{-1}\mathbf{M}\delta \mathbf{x} + \delta \boldsymbol{\mu} = -\mathbf{M}\mathbf{1} + \mathbf{X}^{-1}\mathbf{1}/t = -\boldsymbol{\mu} + \mathbf{X}^{-1}\mathbf{1}/t \quad (2.246)$$

Adding this to the first equation:

$$\mathbf{W}\delta \mathbf{x} + \mathbf{J}_h^T(\mathbf{x})\delta \boldsymbol{\lambda} - \delta \boldsymbol{\mu} = -\mathbf{g}_f(\mathbf{x}) - \mathbf{J}_h^T(\mathbf{x})\boldsymbol{\lambda} + \boldsymbol{\mu} \quad (2.247)$$

we get

$$(\mathbf{W} + \mathbf{X}^{-1}\mathbf{M})\delta \mathbf{x} + \mathbf{J}_h^T(\mathbf{x})\delta \boldsymbol{\lambda} = -\mathbf{g}_f(\mathbf{x}) - \mathbf{J}_h^T(\mathbf{x})\boldsymbol{\lambda} + \mathbf{X}^{-1}\mathbf{1}/t = -\nabla_{\mathbf{x}}L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \quad (2.248)$$

Combining this equation with the second one, we get an equation system of two vector variables  $\delta \mathbf{x}$  and  $\delta \boldsymbol{\lambda}$  with symmetric coefficient matrix:

$$\begin{bmatrix} \mathbf{W} + \mathbf{X}^{-1}\mathbf{M} & \mathbf{J}_h^T(\mathbf{x}) \\ \mathbf{J}_h(\mathbf{x}) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \delta \mathbf{x} \\ \delta \boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \nabla_{\mathbf{x}}L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \\ \mathbf{h}(\mathbf{x}) \end{bmatrix} \quad (2.249)$$

Solving this equation system we get  $\delta \mathbf{x}$  and  $\delta \boldsymbol{\lambda}$ , and we can further find  $\delta \boldsymbol{\mu}$  by solving the third equation

$$\mathbf{M}\delta \mathbf{x} + \mathbf{X}\delta \boldsymbol{\mu} = -\mathbf{X}\mathbf{M}\mathbf{1} - \mathbf{1}/t \quad (2.250)$$

to get

$$\delta \boldsymbol{\mu} = \mathbf{X}^{-1}\mathbf{1}/t - \mathbf{X}^{-1}\mathbf{M}\delta \mathbf{x} - \boldsymbol{\mu} \quad (2.251)$$

We now consider the two special cases of linear and quadratic programming:

- **Linear programming:**

$$\begin{aligned} \min: & \quad f(\mathbf{x}) = \mathbf{c}^T \mathbf{x} \\ \text{s. t.}: & \quad \mathbf{h}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}, \quad \mathbf{x} \geq \mathbf{0} \end{aligned} \quad (2.252)$$

We have

$$-\mathbf{g}_f(\mathbf{x}) = \nabla_{\mathbf{x}}f(\mathbf{x}) = \mathbf{c}$$

- $\mathbf{J}_h(\mathbf{x}) = \nabla_{\mathbf{x}}(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{A}$ .
- $\mathbf{W}(\mathbf{x}) = \nabla_{\mathbf{x}}^2 L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{0}$ .

The Lagrangian is

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{c}^T \mathbf{x} + \boldsymbol{\lambda}^T (\mathbf{A}\mathbf{x} - \mathbf{b}) - \boldsymbol{\mu}^T \mathbf{x} \quad (2.253)$$

The KKT conditions are:

$$\begin{cases} \nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{g}_f(\mathbf{x}) + \mathbf{J}_h^T(\mathbf{x})\boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{c} + \mathbf{A}^T \boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{0} \\ \mathbf{h}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0} \\ \mathbf{X}\mathbf{M}\mathbf{1} - \mathbf{1}/t = \mathbf{0} \end{cases}$$

The search direction of the iteration can be found by solving the this equation system:

$$\begin{bmatrix} \mathbf{0} & \mathbf{A}^T & -\mathbf{I} \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{M} & \mathbf{0} & \mathbf{X} \end{bmatrix} \begin{bmatrix} \delta \mathbf{x} \\ \delta \boldsymbol{\lambda} \\ \delta \boldsymbol{\mu} \end{bmatrix} = - \begin{bmatrix} \mathbf{c} + \mathbf{A}^T \boldsymbol{\lambda} - \boldsymbol{\mu} \\ \mathbf{A}\mathbf{x} - \mathbf{b} \\ \mathbf{X}\mathbf{M}\mathbf{1} - \mathbf{1}/t \end{bmatrix} \quad (2.254)$$

By solving the equation  $\mathbf{c} + \mathbf{A}^T \boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{0}$  we get the initial value  $\boldsymbol{\lambda}$ . The Matlab code for the interior point method for the LP problem is listed below:

```
function x=InteriorPointLP(A,c,b,x)
    % given A,b,c of the LP problem and initial value of x,
    % find optimal solution x that minimizes c'*x
    [m n]=size(A);      % m constraints, n variables
    z1=zeros(n);
    z2=zeros(m);        % zero matrices in coefficient matrix
    z3=zeros(m,n);
    I=eye(n);           % identity matrix in coefficient matrix
    y=ones(n,1);
    t=9;                % initial value for parameter t
    alpha=0.3;          % small enough not to exceed boundary
    mu=x./t;
    lambda=pinv(A')*(mu-c); % initial value of lambda
    w=[x; lambda; mu];  % initial values for all three
    B=[c+A'*lambda-mu; A*x-b; x.*mu-y/t];
    p=[x(1) x(2)];      % initial guess of solution
    while norm(B)>10^(-7)
        t=t*9;          % increase parameter t
        X=diag(x);
        M=diag(mu);
        C=[z1 A' -I; A z2 z3; M z3' X];
        B=[c+A'*lambda-mu; A*x-b; x.*mu-y/t];
        dw=-inv(C)*B;   % find search direction
        w=w+alpha*dw;   % step forward
        x=w(1:n);
```

```

        lambda=w(n+1:n+m);
        mu=w(n+m+1:length(w));
        p=[p; x(1), x(2)];
    end
    scatter(p(:,1),p(:,2)); % plot trajectory of solution
end

```

**Example:**

A given linear programming problem shown below is first converted into the standard form:

$$\begin{array}{ll} \max: & f(x_1, x_2) = 2x_1 + 3x_2 \\ \text{s. t.}: & \begin{cases} 2x_1 + x_2 \leq 18 \\ 6x_1 + 5x_2 \leq 60 \\ 2x_1 + 5x_2 \leq 40 \\ x_1 \geq 0, x_2 \geq 0 \end{cases} \end{array} \quad \Rightarrow \quad \begin{array}{ll} \min: & f(x_1, x_2, x_3, x_4, x_5) = -2x_1 - 3x_2 \\ \text{s. t.}: & \begin{cases} 2x_1 + x_2 + x_3 = 18 \\ 6x_1 + 5x_2 + x_4 = 60 \\ 2x_1 + 5x_2 + x_5 = 40 \\ x_i \geq 0, (i = 1, \dots, 5) \end{cases} \end{array}$$

or in matrix form:

$$\begin{array}{ll} \min: & f(\mathbf{x}) = \mathbf{c}^T \mathbf{x} \\ \text{s. t.}: & \mathbf{h}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}, \quad \mathbf{x} \geq \mathbf{0} \end{array}$$

with

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} -2 \\ -3 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 \\ 6 & 5 & 0 & 1 & 0 \\ 2 & 5 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 18 \\ 60 \\ 40 \end{bmatrix}$$

where  $x_3, x_4, x_5$  are the slack variables.

We choose an initial value  $\mathbf{x}_0 = [1, 2, 1, 1, 1]^T$ , and an initial parameter  $t = 9$ , which is scaled up by a factor of 9 in each iteration. We also used full Newton step size of  $\alpha = 1$ . After 8 iterations, the algorithm converged to the optimal solution  $\mathbf{x}^* = [5, 6]^T$  corresponding to the maximum function value  $f(\mathbf{x}^*) = 2x_1 + 3x_2 = 28$ :

	$(x_1$	$x_2)$	$f(\mathbf{x})$	error
1	(1.000000e + 00	2.000000e + 00)	-8.000000	52.413951
2	(4.654514e + 00	6.346354e + 00)	-28.348090	3.080204
3	(5.040828e + 00	5.946973e + 00)	-27.922575	0.213509
4	(4.997282e + 00	6.001764e + 00)	-27.999856	0.004262
5	(4.999906e + 00	5.999962e + 00)	-27.999697	0.000303
6	(4.999989e + 00	5.999996e + 00)	-27.999966	0.000034
7	(4.999999e + 00	6.000000e + 00)	-27.999996	0.000004
8	(5.000000e + 00	6.000000e + 00)	-28.000000	0.000000

At the end of the iteration, we also get the values of the slack variables:  $x_3 = 2, x_4 = x_5 = 0$ .



- Quadratic programming:

$$\begin{aligned} \min: & f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{s. t.}: & \mathbf{h}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}, \quad \mathbf{x} \geq \mathbf{0} \end{aligned} \quad (2.255)$$

We have

$$\begin{aligned} - & \mathbf{g}_f(\mathbf{x}) = \nabla_{\mathbf{x}}f(\mathbf{x}) = \mathbf{Q}\mathbf{x} + \mathbf{c} \\ - & \mathbf{J}_h(\mathbf{x}) = \nabla_{\mathbf{x}}(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{A}. \\ - & \mathbf{W}(\mathbf{x}) = \nabla_{\mathbf{x}}^2L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{Q}. \end{aligned}$$

The Lagrangian is

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} + \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) - \boldsymbol{\mu}^T\mathbf{x} \quad (2.256)$$

The KKT conditions are:

$$\begin{cases} \nabla_{\mathbf{x}}L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{g}_f(\mathbf{x}) + \mathbf{J}_h^T(\mathbf{x})\boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{Q}\mathbf{x} + \mathbf{c} + \mathbf{A}^T\boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{0} \\ \mathbf{h}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0} \\ \mathbf{X}\mathbf{M}\mathbf{1} - \mathbf{1}/t = \mathbf{0} \end{cases}$$

The equation system for the Newton-Raphson method is:

$$\begin{bmatrix} \mathbf{Q} & \mathbf{A}^T & -\mathbf{I} \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{M} & \mathbf{0} & \mathbf{X} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\boldsymbol{\lambda} \\ \delta\boldsymbol{\mu} \end{bmatrix} = - \begin{bmatrix} \mathbf{Q}\mathbf{x} + \mathbf{c} + \mathbf{A}^T\boldsymbol{\lambda} - \boldsymbol{\mu} \\ \mathbf{A}\mathbf{x} - \mathbf{b} \\ \mathbf{X}\mathbf{M}\mathbf{1} - \mathbf{1}/t \end{bmatrix} \quad (2.257)$$

The Matlab code for the interior point method for the QP problem is listed below:

```
function [x mu lambda]=InteriorPointQP(Q,A,c,b,x)
n=length(c); % n variables
m=length(b); % m constraints
z2=zeros(m);
z3=zeros(m,n);
I=eye(n);
y=ones(n,1);
t=9; % initial value for parameter t
alpha=0.1; % stepsize, small enough not to exceed boundary
mu=x./t;
lambda=pinv(A')*(mu-c-Q*x); % initial value of lambda
w=[x; lambda; mu]; % initial value
B=[Q*x+c+A'*lambda-mu; A*x-b; x.*mu-y/t];
while norm(B)>10^(-7)
t=t*9; % increase parameter t
X=diag(x);
M=diag(mu);
C=[Q A' -I; A z2 z3; M z3' X];
B=[Q*x+c+A'*lambda-mu; A*x-b; x.*mu-y/t];
dw=-inv(C)*B; % find search direction
```

```

w=w+alpha*dw; % step forward
x=w(1:n);
lambda=w(n+1:n+m);
mu=w(n+m+1:length(w));
end
end

```

**Example:**

A 2-D quadratic programming problem shown below is to minimize four different quadratic functions under the same linear constraints as the linear programming problem in the previous example. In general, the quadratic function is given in the matrix form:

$$\begin{aligned} \text{min: } & f(\mathbf{x}) = [\mathbf{x} - \mathbf{m}]^T \mathbf{Q} [\mathbf{x} - \mathbf{m}] = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{c}^T \mathbf{x} + d \\ \text{s. t.: } & \mathbf{h}(\mathbf{x}) = \mathbf{A} \mathbf{x} - \mathbf{b} = \mathbf{0}, \quad \mathbf{x} \geq \mathbf{0} \end{aligned}$$

with following four different sets of function parameters:

$$\mathbf{Q} = \begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}, \quad \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}$$

$$\mathbf{m} = \begin{bmatrix} 4 \\ 10 \end{bmatrix}, \quad \begin{bmatrix} 9 \\ 5 \end{bmatrix}, \quad \begin{bmatrix} 7 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} -1 \\ 6 \end{bmatrix}$$

As shown in the four panels in the figure below, the iteration of the interior point algorithm brings the solution from the same initial position at  $\mathbf{x}_0 = [2, 1]^T$  to the final position for each of the four objective functions  $f(\mathbf{x})$ , the optimal solution, which is on the boundary of the feasible region in all cases except the third one, where the optimal solution is inside the feasible region, i.e., the optimization problem is not constrained. Also note that the trajectories of the solution during the iteration go straightly from the initial guess to the final optimal solution by following the negative gradient direction of the quadratic function in all cases except in the last one, where it makes a turn while approaching the vertical boundary, due obviously to the significantly greater value of the log barrier function close to the boundary.

We also note that the dual problem in the SVM algorithm in Eq. (2.196) is in the same form as in Eq. (2.255), i.e., it is a quadratic programming problem that can be solved by the interior point algorithm.

# 3 Review of Probability and Statistics

---

## 3.1 Univariate and Multivariate Random Variables

### 3.1.1 Univariate Variables

- **Random Experiment and its Sample Space**

A *random experiment* is a procedure that can be repeated an infinite number of times and has a set of possible outcomes. The *sample space*  $\Omega$  of a certain random experiment is the totality of all its possible outcomes.  $\Omega$  may be either countable (discrete) or uncountable (continuous).

For example, “Randomly pick a card from a deck of 10 cards labeled  $0, 1, \dots, 9$ ” is a random experiment. The sample space is a set of the 10 possible outcomes:  $\Omega = \{0, 1, \dots, 9\}$ .

- **Random Event and its Probability**

A *random event*  $A \subset \Omega$  is a subset of  $\Omega$ , which can be a null set (empty set)  $\emptyset$ , a proper subset, e.g.,  $A = \{0, 2, 4, 6, 8\} \subset \Omega = \{0, 1, \dots, 9\}$ , or the entire  $\Omega$ .

Event  $A$  occurs if the outcome  $s \in A$  is a member of  $A$ . The probability  $P(A)$  of an event  $A$  is a real-valued function that maps  $A$  to a real number  $0 \leq P(A) \leq 1$ . In particular,  $P(\emptyset) = 0$ , and  $P(\Omega) = 1$ .

For example, “The randomly chosen card has a number smaller than 4” is a random event, which is represented by a subset  $A = \{0, 1, 2, 3\} \subset \Omega$ . The probability of this event  $A$  is  $P(A) = 4/10 = 0.4$ . Event  $A$  occurs if the outcome is one of the member of  $A$  (e.g., 2).

- **Probability Space**

The triple  $(\Omega, A, P)$  of sample space  $\Omega$ , events  $A$  and probability  $P$  is called the probability space.

- **Random Variables**

A random variable  $x = x(s)$  is a real-valued function that maps any outcome  $s \in \Omega$  into a real number  $x$ , which can be either continuous or discrete.

- **Cumulative and Density Distribution Functions**

The *cumulative distribution function (cdf)* of a random variable  $x$  is defined as

$$F_x(\xi) = P(x < \xi) \tag{3.1}$$

and we have

$$P(a \leq x < b) = F_x(b) - F_x(a) \quad (3.2)$$

The *probability density function (pdf)* of  $x$  is defined as

$$p_x(x) = \frac{d}{d\xi} F_x(\xi) \quad (3.3)$$

and we have

$$F_x(\xi) = \int_{-\infty}^{\xi} p(x) dx, \quad \text{and} \quad P(a \leq x < b) = F_x(b) - F_x(a) = \int_a^b p(x) dx \quad (3.4)$$

and

$$F_x(\infty) = \int_{-\infty}^{\infty} p(x) dx = 1 \quad (3.5)$$

- **Discrete Random Variables**

If a random variable  $x$  can only take one of a set of finite number of discrete values  $\{x_i \ i = 1, \dots, n\}$ , then its *probability distribution* is

$$P(x = x_i) = P(x_i) = P_i \quad (i = 1, \dots, n) \quad (3.6)$$

and we have

$$0 \leq P_i \leq 1, \quad \sum_{i=1}^n P_i = 1 \quad (3.7)$$

The cumulative distribution function is

$$F_x(\xi) = P(x < \xi) = \sum_{x_i < \xi} P(x_i) = \sum_{i=1}^k P_i \quad (3.8)$$

where we assumed  $x_1 < \dots < x_n$  and  $\xi = x_{k+1}$ .

- **Expectation**

The *expectation* (or *mean value, average*) of a random variable  $x$  is defined as the average of all possible values of  $x$  weighted by their corresponding probabilities:

$$\mu_x = E(x) = \sum_{i=1}^n x_i P(x_i) = \sum_{i=1}^n x_i P_i \quad (3.9)$$

if  $x$  is discrete, or

$$\mu_x = E(x) = \int_{-\infty}^{\infty} x p(x) dx \quad (3.10)$$

If  $x$  is continuous.

When the probability  $p$  or  $P$  is not available, we can estimate the expectation  $E(x)$  based on a set of samples  $\{x_1, \dots, x_k\}$  of a discrete random variable  $x$ :

$$\hat{\mu}_x = \bar{x} = \frac{1}{k} \sum_{i=1}^k x_i \quad (3.11)$$

Sometimes for convenience the mean of a random variable can be Given a random variable  $x$  with mean  $\mu_x$ , we can define another random variable  $y$  as

$$y = x - \mu_x, \quad E(y) = E(x - \mu_x) = E(x) - \mu_x = 0 \quad (3.12)$$

Variable  $y$  has the same variance  $\sigma_y^2 = \sigma_x^2$  as  $x$ , and can be considered as a translated (shifted) version of  $x$ . This conversion is carried out often for mathematical convenience without loss of generality.

- **Variance**

The *variance* of a random variable  $x$  is defined as

$$\sigma^2 = Var(x) \triangleq E[(x - \mu)^2] = \int_{-\infty}^{\infty} (x - \mu)^2 p(x) dx \geq 0 \quad (3.13)$$

if  $x$  is continuous, or

$$\sigma^2 = Var(x) = E[(x - \mu)^2] = \sum_{i=1}^n (x_i - \mu)^2 P_i \geq 0 \quad (3.14)$$

if  $x$  is discrete. We note that  $\sigma^2$  is non-negative, and  $\sigma^2 = 0$  only in the special case when  $x = \mu_x$  is no longer random.

The *standard deviation* of  $X$  is defined as

$$\sigma = \sqrt{Var(x)} = \sqrt{\sigma^2} \quad (3.15)$$

We have

$$\begin{aligned} \sigma^2 = Var(x) &= E[(x - \mu)^2] = E(x^2 - 2\mu x + \mu^2) \\ &= E(x^2) - 2\mu E(x) + \mu^2 = E(x^2) - \mu^2 \end{aligned} \quad (3.16)$$

When the probability  $p$  or  $P$  is not available, we can estimate the variance  $\sigma^2(x)$  based on a set of samples  $\{x_1, \dots, x_k\}$  of a discrete random variable  $x$ :

$$\hat{\sigma}^2 = \frac{1}{k-1} \sum_{i=1}^k (x_i - \mu)^2 \quad (3.17)$$

- **The Median and Mode**

The median of a random variable  $x$  is defined as the middle value of its distribution. If  $x$  is discrete, then its median  $\tilde{x}$  is defined as

$$\tilde{x} = \begin{cases} x_{(n+1)/2} & \text{if } n \text{ is odd} \\ \frac{x_{n/2} + x_{n/2+1}}{2} & \text{if } n \text{ is even} \end{cases} \quad (3.18)$$

If  $x$  is continuous, then the median  $\tilde{x}$  is defined such that

$$\int_{-\infty}^{\tilde{x}} p(x)dx = \int_{\tilde{x}}^{\infty} p(x)dx = 0.5 \quad (3.19)$$

The *mode*: the value that appears most frequently among  $\{x_1, \dots, x_n\}$ .

$$\tilde{x} = \arg \max_x p(x), \quad \text{i.e.,} \quad p(\tilde{x}) = \max_x p(x) \quad (3.20)$$

The mean, median, and mode are the same if  $p(x)$  is unimodal (e.g., a normal distribution), but they are all different from each other if  $p(x)$  is asymmetric.

For example, given  $n = 8$  values in  $\{1, 2, 3, 3, 4, 4, 4, 5\}$ , we can find their mean as

$$\bar{x} = \frac{1}{8}(1+2+3+3+4+4+4+5) = 1 \times \frac{1}{8} + 2 \times \frac{1}{8} + 3 \times \frac{2}{8} + 4 \times \frac{3}{8} + 5 \times \frac{1}{8} = \frac{26}{8} = 3.25 \quad (3.21)$$

The median and mode are respectively 3.5 and 4.

- **Intersection and Union of Events**

Let  $A$  and  $B$  be two events defined over  $\Omega$ .

- The *Union* (OR) of events  $A$  and  $B$  is an event whose outcomes belong to either  $A$  or  $B$ ,  $A \cup B$ .

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \quad (3.22)$$

In particular, if  $A$  and  $B$  are *mutually exclusive*, i.e.,

$$A \cap B = \emptyset \quad (3.23)$$

then

$$P(A \cup B) = P(A) + P(B) \quad (3.24)$$

- The *Intersection* (AND) of events  $A$  and  $B$  is an event whose outcomes belong to both  $A$  and  $B$ ,  $A \cap B$ .

$$P(A, B) = P(A \cap B) = P(B|A)P(A) = P(A|B)P(B) \quad (3.25)$$

- **Independent Variables**

Two random variables  $A$  and  $B$  are *independent* if and only if

$$P(A \cap B) = P(A)P(B) \quad (3.26)$$

Similarly, a set of  $N$  variables are independent iff

$$p(x_1, \dots, x_N) = p(x_1)p(x_2) \cdots p(x_N) \quad (3.27)$$

- **Conditional Probability**

The *conditional probability*  $P(A|B)$  is the probability of event  $A$  given that event  $B$  has already occurred. Given  $P(B)$  as well as  $P(A|B)$ , we get

$$P(A, B) = P(A|B)P(B) \quad (3.28)$$

We have

$$P(A|B) = \frac{P(A, B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}, \quad (3.29)$$

We further have

$$\left. \begin{aligned} P(A|B) P(B|C) &= P(B|A) P(A|C) \\ P(A|B, C) P(B|C) &= P(B|A, C) P(A|C) \end{aligned} \right\} = P(A, B|C) \quad (3.30)$$

If events  $A$  and  $B$  are *independent*, i.e.,  $P(A, B) = P(A)P(B)$ , then

$$P(A|B) = \frac{P(A, B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A) \quad (3.31)$$

- **Marginal Probability** of  $A$  is the probability of event  $A$  averaging over information about  $B$  (the value of  $B$  is unknown):

$$p(x) = \int p(x, y) dy, \quad p(y) = \int p(x, y) dx \quad (3.32)$$

- **Partition Theorem (Law of Total Probability)**

Let  $\{A_1, \dots, A_n\}$  be a set of *mutually exclusive* events that partition the sample space  $\Omega$ , i.e.,

$$\bigcup_{i=1}^n A_i = \Omega, \quad \text{and} \quad A_i \cap A_j = \emptyset \quad \forall i \neq j \quad (3.33)$$

then for any event  $B \subset \Omega$ , we have

$$P(B) = \sum_{i=1}^n P(B, A_i) = \sum_{i=1}^n P(B|A_i)P(A_i) \quad (3.34)$$

**Proof:**

$$\begin{aligned} P(B) &= P(B \cap \Omega) = P(B \cap (\bigcup_{i=1}^n A_i)) = P(\bigcup_{i=1}^n (B \cap A_i)) \\ &= \sum_{i=1}^n P(B \cap A_i) = \sum_{i=1}^n P(B|A_i)P(A_i) \end{aligned} \quad (3.35)$$

- **Bayes' Theorem**

$$P(A_i|B) = \frac{P(B|A_i) P(A_i)}{P(B)} = \frac{P(B|A_i) P(A_i)}{\sum_{i=1}^n P(B|A_i)P(A_i)} \quad (3.36)$$

Based on Bayes' theorem, Bayesian inference is of essential importance and is widely applied in many fields, and it plays an essential role in machine learning, as some most important algorithms are based on it.

- **Normal (Gaussian) Distribution**

Random variable  $X$  has a *normal distribution* if its pdf is:

$$\mathcal{N}(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) \quad (3.37)$$

It can be shown that

$$\int_{-\infty}^{\infty} \mathcal{N}(x, \mu, \sigma) dx = 1 \quad (3.38)$$

$$E(x) = \int_{-\infty}^{\infty} x \mathcal{N}(x, \mu, \sigma) dx = \mu \quad (3.39)$$

and

$$Var(x) = \int_{-\infty}^{\infty} (x - \mu)^2 \mathcal{N}(x, \mu, \sigma) dx = \sigma^2 \quad (3.40)$$

- **Function of Random Variable**

A function  $y = f(x)$  of a random variable  $x$  with pdf  $p_X(x)$  is also a random variable. If the function is differentiable and invertible, then we can find its pdf. We first consider its cdf in the following two cases. First, if  $y = f(x)$  is a monotonically increasing function ( $dy/dx > 0$ ), then we have

$$F_Y(y) = P(Y < y) = P(f(X) < y) = P(X < f^{-1}(y)) = \int_{-\infty}^x p_X(u) du \Big|_{x=f^{-1}(y)} \quad (3.41)$$

and its pdf is:

$$p_Y(y) = \frac{d}{dy} F_Y(y) = \frac{d}{dy} \int_{-\infty}^x p_X(u) du \Big|_{x=f^{-1}(y)} = \left[ p_X(x) \frac{dx}{dy} \right]_{x=f^{-1}(y)} \quad (3.42)$$

Next, if  $y = f(x)$  is monotonically decreasing ( $dy/dx < 0$ ), then its cdf is:

$$F_Y(y) = P(Y < y) = P(f(X) < y) = P(X > f^{-1}(y)) = \int_x^{\infty} p_X(u) du \Big|_{x=f^{-1}(y)} \quad (3.43)$$

and its pdf is:

$$p_Y(y) = \frac{d}{dy} F_Y(y) = \frac{d}{dy} \int_x^{\infty} p_X(u) du \Big|_{x=f^{-1}(y)} = \left[ -p_X(x) \frac{dx}{dy} \right]_{x=f^{-1}(y)} \quad (3.44)$$

Combining both cases, we get

$$p_Y(y) = \left[ p_X(x) \left| \frac{dx}{dy} \right| \right]_{x=f^{-1}(y)} \quad (3.45)$$

Alternatively, consider the probability for  $a \leq x \leq b$ , same as the probability for  $f(a) \leq y = f(x) \leq f(b)$ , i.e.,

$$\int_a^b p_X(x) dx = \int_{f(a)}^{f(b)} p_Y(y) dy \quad (3.46)$$

If the interval  $b - a$  is infinitesimally small and so is  $f(b) - f(a)$ , then  $p_X(x)$  and  $p_Y(y)$  become constant, we have

$$p_X(x) dx = p_Y(y) dy, \quad \text{i.e.,} \quad p_Y(y) = \left[ p_X(x) \left| \frac{dx}{dy} \right| \right]_{x=f^{-1}(y)} \quad (3.47)$$



where the absolute value  $|dx/dy| \geq 0$  guarantees that  $p_Y(y) > 0$  as well as  $p_X(x) > 0$ .

### 3.1.2 Multivariate random vectors

- **Multiple Random Variables**

An outcome of a random experiment may need to be described by a set of  $n > 1$  random variables  $\{x_1, \dots, x_n\}$ , which can be more concisely represented as a *random vector*  $\mathbf{x} = [x_1, \dots, x_n]^T$ . For example, in signal processing  $\mathbf{x}$  can be a set of time samples  $x_i = x(t_i)$  of a random signal  $x(t)$  (a random or stochastic process), and in pattern recognition  $\mathbf{x}$  can represent a *pattern*, an object of interest, in terms of its *features* that describe the pattern.

- **Joint Distribution Function and Density Function**

The *joint distribution function* of a random vector  $\mathbf{x}$  is defined as

$$\begin{aligned} F_X(u_1, \dots, u_n) &= P(x_1 < u_1, \dots, x_n < u_n) \\ &= \int_{-\infty}^{u_1} \cdots \int_{-\infty}^{u_n} p(\xi_1, \dots, \xi_n) d\xi_1 \cdots d\xi_n \end{aligned} \quad (3.48)$$

where  $p(\xi_1, \dots, \xi_n)$  is the *joint density function* of the random vector  $\mathbf{x}$  in the  $n$ -dimensional vector space. In particular, if all these random variables are independent, then  $p(x_1, \dots, x_n) = p(x_1) \cdots p(x_n) = \prod_{i=1}^n p(x_i)$ .

- **Mean Vector**

The *expectation* or *mean* of a random variable  $x_i$  is defined as

$$\mu_i = E(x_i) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \xi_i p(\xi_1, \dots, \xi_n) d\xi_1 \cdots d\xi_n \quad (3.49)$$

The *mean vector* of a random vector  $\mathbf{x}$  is defined as

$$\mathbf{m} = E(\mathbf{x}) = [E(x_1), \dots, E(x_n)]^T = [\mu_1, \dots, \mu_n]^T \quad (3.50)$$

which can be interpreted as the center of gravity of an  $n$ -dimensional object with  $p(x_1, \dots, x_n)$  being the density function.

- **Covariance Matrix**

The *covariance* of random variables  $x_i$  and  $x_j$  is defined as

$$\begin{aligned} \sigma_{ij}^2 &= Cov(x_i, x_j) = E[(x_i - \mu_i)(x_j - \mu_j)] \\ &= E[x_i x_j] - E[x_i] \mu_j - \mu_i E[x_j] + \mu_i \mu_j = E[x_i x_j] - \mu_i \mu_j \\ &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \xi_i \xi_j p(\xi_1, \dots, \xi_n) d\xi_1 \cdots d\xi_n - \mu_i \mu_j \end{aligned} \quad (3.51)$$

Note that  $\sigma_{ij}^2 = \sigma_{ji}^2$  can be either positive or negative. Specially when  $i = j$ ,  $\sigma_{ii}^2 = E[(x_i - \mu_i)^2] = E(x_i^2) - \mu_i^2 \geq 0$  is the variance of variable  $x_i$ , which can be interpreted as the amount of dynamic energy (or information) contained in  $x_i$ .

If an off-diagonal component  $\sigma_{ij}^2 = 0$  is zero, the two variables  $x_i$  and  $x_j$  are *uncorrelated* or *decorrelated*, and

$$E(x_i x_j) = E(x_i)E(x_j) = \mu_i \mu_j \quad (3.52)$$

If all off-diagonal components  $\sigma_{ij} = 0$  are zero, then  $\mathbf{x}$  is uncorrelated. If  $x_i$  ( $i = 1, \dots, n$ ) are independent ( $p(x_1, \dots, x_n) = p(x_1) \cdots p(x_n)$ ), then they are also uncorrelated. But uncorrelated variables are not necessarily independent. If  $\mathbf{x}$  is normally distributed, then its components are independent if and only they are uncorrelated.

The *covariance matrix* of a random vector  $\mathbf{x}$  is defined as

$$\Sigma_{\mathbf{x}} = E[(\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T] = E(\mathbf{x}\mathbf{x}^T) - \mathbf{m}\mathbf{m}^T = \begin{bmatrix} \sigma_{11}^2 & \cdots & \sigma_{1n}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{n1}^2 & \cdots & \sigma_{nn}^2 \end{bmatrix}, \quad (3.53)$$

The  $i$ th diagonal component  $\sigma_{ii}^2 = E[(x_i - \mu_i)^2]$  is the variance of  $x_i$ , and the trace can be interpreted as the total energy or information contained in  $\mathbf{x}$ :

$$\text{tr } \Sigma = \sum_{i=1}^n \sigma_i^2 = \sum_{i=1}^n \lambda_i \quad (3.54)$$

where  $\lambda_1, \dots, \lambda_n$  are the eigenvalues of  $\Sigma$ .

As  $\sigma_{ij}^2 = \sigma_{ji}^2$ ,  $\Sigma = \Sigma^T$  is symmetric. Also,  $\Sigma$  is positive semi-definite (Section ?? in Appendix A), i.e., the following holds for any  $\mathbf{z} \neq \mathbf{0}$ :

$$\begin{aligned} \mathbf{z}^T \Sigma \mathbf{z} &= \mathbf{z}^T E[(\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T] \mathbf{z} = E[\mathbf{z}^T (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T \mathbf{z}] \\ &= E[(\mathbf{z}^T (\mathbf{x} - \mathbf{m}))^2] \geq 0 \end{aligned} \quad (3.55)$$

All eigenvalues of this positive semi-definite matrix are non-negative, and they can be sorted so that  $\lambda_1 \geq \dots \geq \lambda_n \geq 0$ . Consequently, we have

$$\text{tr } \Sigma = \sum_{i=1}^N \lambda_i \geq 0, \quad \det \Sigma = \prod_{i=1}^N \lambda_i \geq 0 \quad (3.56)$$

The equality holds only in the special case when  $\mathbf{x} = \mathbf{m}$ , i.e.,  $\mathbf{x}$  is deterministic. If this is not the case, the inequality always holds.

The *Pearson correlation coefficient* between two random variables  $x_i$  and  $x_j$  is defined as the normalized covariance  $\sigma_{ij}^2$ , when it is divided by the standard deviations  $\sigma_i$  and  $\sigma_j$ :

$$\rho_{ij} = \frac{\sigma_{ij}^2}{\sqrt{\sigma_i^2 \sigma_j^2}} = \frac{\sigma_{ij}^2}{\sigma_i \sigma_j} \quad (3.57)$$

As the covariance can be considered as an inner product of  $x_i$  and  $x_j$ ,

$\sigma_{ij}^2 = \langle x_i, x_j \rangle$ , the Cauchy-Schwarz inequality (Section 1 in Appendix A) applies:

$$|\langle x_i, x_j \rangle|^2 = \sigma_{ij}^4 \leq \langle x_i, x_i \rangle \langle x_j, x_j \rangle = \sigma_i^2 \sigma_j^2, \quad \text{or} \quad \left( \frac{\sigma_{ij}^2}{\sigma_i \sigma_j} \right)^2 = \rho^2 \leq 1 \quad (3.58)$$

i.e.,

$$-1 \leq \rho \leq 1 \quad (3.59)$$

- **Sample correlation coefficient**

The sample correlation coefficient of two random variables  $x$  and  $y$  can be found based on a set of samples of the two variables  $\{(x_i, y_i), \quad i = 1, \dots, k\}$ :

$$r_{xy} = \frac{\hat{\sigma}_{xy}^2}{\hat{\sigma}_x \hat{\sigma}_y} = \frac{\sum_{i=1}^k (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\left(\sum_{i=1}^k (x_i - \bar{x})^2\right) \left(\sum_{i=1}^k (y_i - \bar{y})^2\right)}} \quad (3.60)$$

- **Correlation Matrix**

The *correlation* matrix of  $\mathbf{x}$  is defined as

$$\mathbf{R}_x = E(\mathbf{x}\mathbf{x}^T) = \mathbf{\Sigma}_x + \mathbf{m}\mathbf{m}^T = \begin{bmatrix} r_{11}^2 & \cdots & r_{1n}^2 \\ \vdots & \ddots & \vdots \\ r_{n1}^2 & \cdots & r_{nn}^2 \end{bmatrix} \quad (3.61)$$

where

$$r_{ij} = E(x_i x_j) = \sigma_{ij}^2 + \mu_i \mu_j \quad (i, j = 1, \dots, n) \quad (3.62)$$

If  $\mathbf{m} = \mathbf{0}$ , then  $\mathbf{\Sigma} = \mathbf{R}$ .

- **Mean and Covariance under Orthogonal Transforms**

Given an orthogonal matrix  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n]$  composed of  $n$  orthogonal column vectors satisfying  $\mathbf{a}_i^T \mathbf{a}_j = \delta_{ij}$ , i.e.,  $\mathbf{A}^T = \mathbf{A}^{-1}$ , an orthogonal transform of  $\mathbf{x}$  can be defined as

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \mathbf{A}^T \mathbf{x} = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix} \mathbf{x}, \quad \text{i.e.} \quad y_i = \mathbf{a}_i^T \mathbf{x}, \quad (i = 1, \dots, n) \quad (3.63)$$

The resulting  $\mathbf{y}$  is also a random vector. The inverse transform is:

$$\mathbf{x} = \mathbf{A}\mathbf{y} = [\mathbf{a}_1, \dots, \mathbf{a}_n] \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^n y_i \mathbf{a}_i \quad (3.64)$$

Consider the squared norm of vector  $\mathbf{x}$ , the total amount of energy contained in  $\mathbf{x}$ :

$$\|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x} = \left( \sum_{i=1}^n y_i \mathbf{a}_i \right)^T \left( \sum_{j=1}^n y_j \mathbf{a}_j \right) = \sum_{i=1}^n \sum_{j=1}^n y_i y_j \mathbf{a}_i^T \mathbf{a}_j = \sum_{i=1}^n y_i^2 = \|\mathbf{y}\|^2 \quad (3.65)$$

This is Parseval's theorem, indicating that the total energy is conserved under any orthogonal transform.

The mean vector  $\mathbf{m}_y$  and the covariance matrix  $\Sigma_y$  of  $\mathbf{y}$  are related to the  $\mathbf{m}_x$  and  $\Sigma_x$  of  $\mathbf{x}$  by:

$$\mathbf{m}_y = E(\mathbf{y}) = E(\mathbf{A}^T \mathbf{x}) = \mathbf{A}^T E(\mathbf{x}) = \mathbf{A}^T \mathbf{m}_x \quad (3.66)$$

and

$$\begin{aligned} \Sigma_y &= E(\mathbf{y}\mathbf{y}^T) - \mathbf{m}_y \mathbf{m}_y^T = E(\mathbf{A}^T \mathbf{x} \mathbf{x}^T \mathbf{A}) - \mathbf{A}^T \mathbf{m}_x \mathbf{m}_x^T \mathbf{A} \\ &= \mathbf{A}^T E(\mathbf{x} \mathbf{x}^T) \mathbf{A} - \mathbf{A}^T \mathbf{m}_x \mathbf{m}_x^T \mathbf{A} = \mathbf{A}^T [E(\mathbf{x} \mathbf{x}^T) - \mathbf{m}_x \mathbf{m}_x^T] \mathbf{A} = \mathbf{A}^T \Sigma_x \mathbf{A} \end{aligned} \quad (3.67)$$

The orthogonal transform does not change the trace of  $\Sigma$ : Due to the commutativity of trace:  $tr(\mathbf{A}\mathbf{B}) = tr(\mathbf{B}\mathbf{A})$ , we have:

$$tr \Sigma_y = tr(\mathbf{A}^T \Sigma_x \mathbf{A}) = tr(\mathbf{A}^T \mathbf{A} \Sigma_x) = tr \Sigma_x \quad (3.68)$$

i.e., the total amount of dynamic energy or information contained in  $\mathbf{x}$  is conserved by the orthogonal transform  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$ .

- **Estimation of  $\mathbf{m}$  and  $\Sigma$**

When  $p(\mathbf{x}) = p(x_1, \dots, x_n)$  is not known,  $\mathbf{m}$  and  $\Sigma$  cannot be found based on their definitions. However, they can be estimated if a set of outcomes (samples)  $\mathbf{x}_i$ , ( $i = 1, \dots, k$ ) of the random experiment in question can be observed.

The *sample mean*  $\hat{\mathbf{m}}$  can be obtained as

$$\hat{\mathbf{m}} = \frac{1}{k} \sum_{i=1}^k \mathbf{x}_i \quad (3.69)$$

The unbiased *sample covariance*  $\hat{\Sigma}$  can be obtained as

$$\begin{aligned} \hat{\Sigma} &= \frac{1}{k-1} \sum_{i=1}^k (\mathbf{x}_i - \hat{\mathbf{m}})(\mathbf{x}_i - \hat{\mathbf{m}})^T = \frac{1}{k-1} \sum_{i=1}^k (\mathbf{x}_i \mathbf{x}_i^T - \mathbf{x}_i \hat{\mathbf{m}}^T - \hat{\mathbf{m}} \mathbf{x}_i^T + \hat{\mathbf{m}} \hat{\mathbf{m}}^T) \\ &= \frac{1}{k-1} \left( \sum_{i=1}^k \mathbf{x} \mathbf{x}^T - k \hat{\mathbf{m}} \hat{\mathbf{m}}^T \right) = \hat{\mathbf{R}} - \frac{k}{k-1} \hat{\mathbf{m}} \hat{\mathbf{m}}^T \end{aligned} \quad (3.70)$$

where

$$\hat{\mathbf{R}} = \frac{1}{k-1} \sum_{i=1}^k \mathbf{x}_i \mathbf{x}_i^T \quad (3.71)$$

If  $k \gg 1$ , then  $k - 1 \approx k$  and the above can be approximated as

$$\hat{\Sigma} \approx \frac{1}{k} \sum_{i=1}^k \mathbf{x}_i \mathbf{x}_i^T - \hat{\mathbf{m}} \hat{\mathbf{m}}^T, \quad \hat{\mathbf{R}} \approx \frac{1}{k} \sum_{i=1}^k \mathbf{x}_i \mathbf{x}_i^T \quad (3.72)$$

If all  $k$  samples in the dataset  $\mathbf{X}$  are independent, then the rank of the  $n \times n$  estimated covariance matrix  $\hat{\Sigma}$  is at most  $k - 1$ , due to the constraining equation:

$$\sum_{i=1}^k (\mathbf{x}_i - \hat{\mathbf{m}}) = \sum_{i=1}^k \mathbf{x}_i - k\hat{\mathbf{m}} = \mathbf{0} \quad (3.73)$$

If  $k - 1 < n$ , then  $\hat{\Sigma}$  does not have a full rank and is therefore noninvertible, i.e., its inverse does not exist.

- **Function of Random Vector**

Eq. (3.45) for the pdf  $p_Y(y)$  of a function  $y = f(x)$  of a single variable  $x$  can be generalized to the pdf of a functions of a random vector. Given a random vector  $\mathbf{x} = [x_1, \dots, x_n]^T$  with pdf  $p_x(\mathbf{x})$ , we can define another random vector  $\mathbf{y} = [y_1, \dots, y_m]^T$  of which  $y_j = f_j(x_1, \dots, x_n)$  for all  $j = 1, \dots, m$ . This function can be written in vector form  $\mathbf{y} = \mathbf{f}(\mathbf{x})$ , or  $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{y})$ . Then the pdf of  $\mathbf{y}$  is

$$p_Y(\mathbf{y}) = [p_X(\mathbf{x}) |\det \mathbf{J}_{f^{-1}}|]_{\mathbf{x}=\mathbf{f}^{-1}(\mathbf{y})} \quad (3.74)$$

where  $|\det \mathbf{J}_{f^{-1}}|$  is the absolute value of the determinant of the Jacobian matrix:

$$\mathbf{J}_{f^{-1}} = \begin{bmatrix} \frac{\partial x_1}{\partial y_1} & \dots & \frac{\partial x_1}{\partial y_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_m}{\partial y_1} & \dots & \frac{\partial x_m}{\partial y_m} \end{bmatrix}_{\mathbf{x}=\mathbf{f}^{-1}(\mathbf{y})} \quad (3.75)$$

Specially, for a linear function  $\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$ ,  $\mathbf{J}_f = \mathbf{A}$  and  $\mathbf{J}_{f^{-1}} = \mathbf{A}^{-1}$ , and

$$p_Y(\mathbf{y}) = [|\det \mathbf{A}^{-1}| p_X(x)]_{\mathbf{x}=\mathbf{A}^{-1}\mathbf{y}} \quad (3.76)$$

### 3.1.3 The Normal Distribution

The normal or Gaussian distribution is the most important pdf widely used to model observed data set drawn from an unknown distribution, due to two of its properties:

- **Central limit theorem**

The distribution of the sum (average or linear combination) of  $n$  independent random variables approaches Gaussian as  $n \rightarrow \infty$ .

For example, the face value of a dice has a uniform distribution from 1 to 6. But the distribution of the sum of a pair of dice is no longer uniform. It has a maximum probability at the mean of 7. As the number of dice

increases, the distribution of the sum of the face values will be better approximated by a Gaussian. This is illustrated as in the figure below showing  $n = 1, 3, 5, 7$ . We see that when

- **Maximization of entropy**

Among all possible probability density functions (pdfs) with the same variance, the normal distribution has the maximum entropy or uncertainty.

The proof of this property will be given later.

The univariate normal probability distribution is

$$p(x) = \mathcal{N}(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad (3.77)$$

In multivariate case, the joint pdf of  $n$  normally distributed random variables  $\mathbf{x} = [x_1, \dots, x_n]^T$  is

$$\begin{aligned} p(\mathbf{x}) &= \mathcal{N}(\mathbf{x}, \mathbf{m}, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{m})^T \Sigma^{-1} (\mathbf{x} - \mathbf{m})\right] \\ &= \exp\left(-\frac{1}{2}(n \log(2\pi) + \log |\Sigma|)\right) \exp\left[-\frac{1}{2}(\mathbf{x}^T \Sigma^{-1} \mathbf{x} - 2\mathbf{m}^T \Sigma^{-1} \mathbf{x} + \mathbf{m}^T \Sigma^{-1} \mathbf{m})\right] \\ &= \exp\left(-\frac{1}{2}(n \log(2\pi) + \log |\Sigma| + \mathbf{m}^T \Sigma^{-1} \mathbf{m})\right) \exp\left[-\frac{1}{2}(\mathbf{x}^T \Sigma^{-1} \mathbf{x} - 2\mathbf{m}^T \Sigma^{-1} \mathbf{x} + \mathbf{m}^T \Sigma^{-1} \mathbf{m})\right] \end{aligned}$$

Here  $\mathbf{m} = E(\mathbf{x})$  and  $\Sigma = Cov(\mathbf{x})$  are respectively the mean vector and covariance matrix of  $\mathbf{x}$ . The covariance matrix  $\Sigma = \Sigma^T$  is symmetric, and positive semi-definite. We further define the log-normal distribution as:

$$\psi(\mathbf{x}) = \log \mathcal{N}(\mathbf{x}, \mathbf{m}, \Sigma) = -\frac{1}{2}(\mathbf{x} - \mathbf{m})^T \Sigma^{-1} (\mathbf{x} - \mathbf{m}) - \frac{n}{2} \log(2\pi) - \frac{1}{2} \log |\Sigma| \quad (3.79)$$

and get its gradient vector and Hessian matrix:

$$\mathbf{g}_\psi(\mathbf{x}) = \frac{d}{d\mathbf{x}} \psi(\mathbf{x}) = -\Sigma^{-1} (\mathbf{x} - \mathbf{m}) \quad (3.80)$$

$$\mathbf{H}_\psi(\mathbf{x}) = \frac{d^2}{d\mathbf{x}^2} \psi(\mathbf{x}) = -\Sigma^{-1} \quad (3.81)$$

From these expressions we can find the mean  $\mathbf{m}$  at which  $\mathcal{N}(\mathbf{m}, \Sigma)$  reaches maximum can be found by solving the equation

$$\mathbf{g}_\psi(\mathbf{x}) = -\Sigma^{-1} (\mathbf{x} - \mathbf{m}) = \mathbf{0} \quad (3.82)$$

to get  $\mathbf{x} = \mathbf{m}$ , and the covariance  $\Sigma$  can be directly found from the second equation:

$$\Sigma = -\mathbf{H}_\psi^{-1}(\mathbf{x}) \quad (3.83)$$

The shape of this normal distribution in the  $n$ -dimensional space can be represented by the iso-hypersurface in the space determined by equation

$$\mathcal{N}(\mathbf{x}, \mathbf{m}, \Sigma) = c_0 \quad (3.84)$$

where  $c_0$  is a constant. Or, equivalently, this equation can be written as

$$(\mathbf{x} - \mathbf{m})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{m}) = c_1 \quad (3.85)$$

where  $c_1$  is another constant related to  $c_0$ ,  $\mathbf{m}$  and  $\boldsymbol{\Sigma}$ .

In particular, when  $n = 2$ ,  $\mathbf{x} = [x_1, x_2]^T$ , and we assume

$$\boldsymbol{\Sigma}^{-1} = \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix} \quad (3.86)$$

then the equation above becomes

$$\begin{aligned} (\mathbf{x} - \mathbf{m})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{m}) &= [x_1 - \mu_1, x_2 - \mu_2] \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix} \\ &= A(x_1 - \mu_1)^2 + B(x_1 - \mu_1)(x_2 - \mu_2) + C(x_2 - \mu_2)^2 \\ &= Ax_1^2 + Bx_1x_2 + Cx_2^2 - (2A\mu_1 + B\mu_2)x_1 - (2C\mu_2 + B\mu_1)x_2 + (A\mu_1^2 + B\mu_1\mu_2 + C\mu_2^2) \end{aligned} \quad (3.87)$$

As  $\boldsymbol{\Sigma}$  is positive semi-definite, and so is  $\boldsymbol{\Sigma}^{-1}$ , we have

$$|\boldsymbol{\Sigma}^{-1}| = AC - B^2/4 \geq 0, \quad \text{i.e.} \quad \Delta = B^2 - 4AC < 0 \quad (3.88)$$

i.e., the quadratic equation above represents an ellipse (instead of other quadratic curves such as hyperbola and parabola) centered at  $\mathbf{m} = [\mu_1, \mu_2]^T$ .

Recall in general that the discriminant  $\Delta = B^2 - 4AC$  of a quadratic equation

$$f(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \quad (3.89)$$

determines the shape of the corresponding conic section in a plane:

$$f(x, y) \text{ is } \begin{cases} \text{an ellipse} & \text{if } \Delta < 0 \\ \text{a parabola} & \text{if } \Delta = 0 \\ \text{a hyperbola} & \text{if } \Delta > 0 \end{cases} \quad (3.90)$$

When  $n = 3$ , the quadratic equation represents an ellipsoid. In general when  $n > 3$ , the equation  $\mathcal{N}(\mathbf{x}, \mathbf{m}, \boldsymbol{\Sigma}) = c_0$  represents a hyper ellipsoid in the  $n$ -dimensional space. The center and spatial distribution of this ellipsoid are determined by  $\mathbf{m}$  and  $\boldsymbol{\Sigma}$ , respectively.

Specially if  $\mathbf{x} = [x_1, \dots, x_n]^T$  is decorrelated, i.e.,  $\sigma_{ij} = 0$  for all  $i \neq j$ ,  $\boldsymbol{\Sigma}$  becomes a diagonal matrix

$$\boldsymbol{\Sigma} = \text{diag}[\sigma_1^2, \dots, \sigma_n^2] = \begin{bmatrix} \sigma_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_n^2 \end{bmatrix} \quad (3.91)$$

and equation  $\mathcal{N}(\mathbf{x}, \mathbf{m}, \boldsymbol{\Sigma}) = c_0$  can be written as

$$(\mathbf{x} - \mathbf{m})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{m}) = \sum_{i=1}^n \frac{(x_i - \mu_i)^2}{\sigma_i^2} = c_1 \quad (3.92)$$

which represents a standard ellipsoid with all its axes parallel to those of the coordinate system.

### Properties of Gaussian distribution

- Sum of normal distributions:

If  $p(\mathbf{x}_i) = \mathcal{N}(\mathbf{m}_i, \boldsymbol{\Sigma}_i)$  ( $i = 1, 2$ ), then  $p(\mathbf{x}_1 + \mathbf{x}_2) = \mathcal{N}(\mathbf{m}_1 + \mathbf{m}_2, \boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)$

- Affine transformation of normal distributions:

If  $p(\mathbf{x}) = \mathcal{N}(\mathbf{m}_x, \boldsymbol{\Sigma}_x)$ , then  $p(\mathbf{y}) = p(\mathbf{A}\mathbf{x} + \mathbf{b}) = \mathcal{N}(\mathbf{m}_y, \boldsymbol{\Sigma}_y)$ , where

$$\mathbf{m}_y = \mathbf{A}\mathbf{m}_x + \mathbf{b}, \quad \boldsymbol{\Sigma}_y = \mathbf{A}\boldsymbol{\Sigma}_x\mathbf{A}^T \quad (3.93)$$

**Proof:**

$$\mathbf{m}_y = E(\mathbf{A}\mathbf{x} + \mathbf{b}) = \mathbf{A} E(\mathbf{x}) + \mathbf{b} = \mathbf{A}\mathbf{m}_x + \mathbf{b} \quad (3.94)$$

$$\begin{aligned} \boldsymbol{\Sigma}_y &= E[(\mathbf{y} - \mathbf{m}_y)(\mathbf{y} - \mathbf{m}_y)^T] \\ &= E[(\mathbf{A}\mathbf{x} - \mathbf{b}) - (\mathbf{A}\mathbf{m}_x - \mathbf{b})][(\mathbf{A}\mathbf{x} - \mathbf{b}) - (\mathbf{A}\mathbf{m}_x - \mathbf{b})]^T] \\ &= E[(\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{m}_x)(\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{m}_x)^T] = \mathbf{A}E[(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^T]\mathbf{A}^T \\ &= \mathbf{A}\boldsymbol{\Sigma}_x\mathbf{A}^T \end{aligned} \quad (3.95)$$

Some special cases:

– If  $\mathbf{A} = \mathbf{I}$ , then  $p(\mathbf{x} + \mathbf{b}) = \mathcal{N}(\mathbf{m}_x + \mathbf{b}, \boldsymbol{\Sigma}_x)$

– If  $\mathbf{b} = \mathbf{0}$ , then  $p(\mathbf{A}\mathbf{x}) = \mathcal{N}(\mathbf{m}_x, \mathbf{A}\boldsymbol{\Sigma}_x\mathbf{A}^T)$

– If  $p(\mathbf{x}) = \mathcal{N}([\mu_1, \dots, \mu_n]^T, \text{diag}[\sigma_1^2, \dots, \sigma_n^2])$ , then  $p(\sum_{i=1}^n x_i) = \mathcal{N}(\sum_{i=1}^n \mu_i, \sum_{i=1}^n \sigma_i^2)$

- Marginal distributions:

$$p(x_1, x_2) = p(\mathbf{x}) = p\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_2^2 \end{bmatrix}\right) \quad (3.96)$$

Applying  $\mathbf{A}_1 = [1, 0]$  and  $\mathbf{A}_2 = [0, 1]$  to  $\mathbf{x} = [x_1, x_2]^T$ , we get the two marginal distributions for  $\mathbf{A}_1\mathbf{x} = x_1$  and  $\mathbf{A}_2\mathbf{x} = x_2$ , respectively:

$$p(x_1) = \mathcal{N}(\mu_1, \sigma_1^2), \quad p(x_2) = \mathcal{N}(\mu_2, \sigma_2^2) \quad (3.97)$$

- Product of Gaussians: Consider two Gaussians:

$$\begin{aligned} p_i(\mathbf{x}) &= \mathcal{N}(\mathbf{x}, \mathbf{m}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{n/2} |\boldsymbol{\Sigma}_i|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{m}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \mathbf{m}_i)\right] \\ &= \frac{1}{(2\pi)^{n/2} |\boldsymbol{\Sigma}_i|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}^T \boldsymbol{\Sigma}_i^{-1} \mathbf{x} - 2\mathbf{x}^T \boldsymbol{\Sigma}_i^{-1} \mathbf{m}_i + \mathbf{m}_i^T \boldsymbol{\Sigma}_i^{-1} \mathbf{m}_i)\right] \\ &= \exp\left[-\frac{1}{2}[n \log(2\pi) + \log |\boldsymbol{\Sigma}_i| + \mathbf{m}_i^T \boldsymbol{\Sigma}_i^{-1} \mathbf{m}_i]\right] \exp\left[-\frac{1}{2}(\mathbf{x}^T \boldsymbol{\Sigma}_i^{-1} \mathbf{x} - 2\mathbf{x}^T \boldsymbol{\Sigma}_i^{-1} \mathbf{m}_i)\right] \\ &= \exp\left[-\frac{1}{2}[\mathbf{x}^T \boldsymbol{\Sigma}_i^{-1} \mathbf{x} - 2\mathbf{x}^T \boldsymbol{\Sigma}_i^{-1} \mathbf{m}_i + c_i]\right] \quad (i = 1, 2) \end{aligned} \quad (3.98)$$

where

$$c_i = n \log(2\pi) + \log |\boldsymbol{\Sigma}_i| + \mathbf{m}_i^T \boldsymbol{\Sigma}_i^{-1} \mathbf{m}_i \quad (3.99)$$



The product of two Gaussians is

$$\begin{aligned}
p_1(\mathbf{x})p_2(\mathbf{x}) &= \mathcal{N}(\mathbf{x}, \mathbf{m}_1, \Sigma_1)\mathcal{N}(\mathbf{x}, \mathbf{m}_2, \Sigma_2) \\
&= \exp\left[-\frac{1}{2}[\mathbf{x}^T \Sigma_1^{-1} \mathbf{x} - 2\mathbf{x}^T \Sigma_1^{-1} \mathbf{m}_1 + c_1 + \mathbf{x}^T \Sigma_2^{-1} \mathbf{x} - 2\mathbf{x}^T \Sigma_2^{-1} \mathbf{m}_2 + c_2]\right] \\
&= \exp\left[-\frac{1}{2}[\mathbf{x}^T (\Sigma_1^{-1} + \Sigma_2^{-1}) \mathbf{x} - 2\mathbf{x}^T (\Sigma_1^{-1} \mathbf{m}_1 + \Sigma_2^{-1} \mathbf{m}_2) + c_1 + c_2]\right] \\
&= \exp\left[-\frac{1}{2}[\mathbf{x}^T \Sigma^{-1} \mathbf{x} - 2\mathbf{x}^T \Sigma^{-1} \mathbf{m} + c + (c_1 + c_2 - c)]\right] \\
&= \mathcal{N}(\mathbf{x}, \mathbf{m}, \Sigma) \exp(c_1 + c_2 - c) \propto \mathcal{N}(\mathbf{x}, \mathbf{m}, \Sigma)
\end{aligned} \tag{3.100}$$

Here we have defined

$$c = n \log(2\pi) + \log |\Sigma| + \mathbf{m}^T \Sigma^{-1} \mathbf{m} \tag{3.101}$$

$$\Sigma = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} \tag{3.102}$$

and

$$\Sigma^{-1} \mathbf{m} = \Sigma_1^{-1} \mathbf{m}_1 + \Sigma_2^{-1} \mathbf{m}_2 \tag{3.103}$$

i.e.,

$$\mathbf{m} = \Sigma(\Sigma_1^{-1} \mathbf{m}_1 + \Sigma_2^{-1} \mathbf{m}_2) = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} (\Sigma_1^{-1} \mathbf{m}_1 + \Sigma_2^{-1} \mathbf{m}_2) \tag{3.104}$$

We see that the product of Gaussians is also a scaled Gaussian.

- **Marginal and conditional distributions of multivariate normal distribution**

A random vector  $\mathbf{x} = [\mathbf{x}_1^T, \mathbf{x}_2^T]^T$  with a normal distribution  $\mathcal{N}(\mathbf{x}, \mathbf{m}, \Sigma)$  can be partitioned into two subvectors  $\mathbf{x}_1$  and  $\mathbf{x}_2$ :

$$p\left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}\right) = p(\mathbf{x}) = \mathcal{N}(\mathbf{x}, \mathbf{m}, \Sigma) = N\left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}, \begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right) \tag{3.105}$$

where  $\Sigma_{ij} = \Sigma_{ji}^T$ ,  $\Sigma_{ii} = \Sigma_{ii}^T$ , ( $i, j = 1, 2, i \neq j$ ). Given this normal joint distribution of  $\mathbf{x}_1$  and  $\mathbf{x}_2$ , we can find the normal conditional distribution of  $\mathbf{x}_i$  given  $\mathbf{x}_j$ :

$$p(\mathbf{x}_i/\mathbf{x}_j) = \mathcal{N}(\mathbf{x}_i, \mathbf{m}_{i/j}, \Sigma_{i/j}) \tag{3.106}$$

in terms of the mean vector and the covariance matrix:

$$\begin{cases} \mathbf{m}_{i/j} &= \mathbf{m}_i + \Sigma_{ij} \Sigma_{jj}^{-1} (\mathbf{x}_j - \mathbf{m}_j) \\ \Sigma_{i/j} &= \Sigma_{ii} - \Sigma_{ij} \Sigma_{jj}^{-1} \Sigma_{ji} \end{cases} \tag{3.107}$$

**Proof:** The joint distribution of  $\mathbf{x}$  can be written as:

$$\begin{aligned}
p(\mathbf{x}) &= p(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{m})^T \Sigma^{-1} (\mathbf{x} - \mathbf{m})\right] \\
&= \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}Q(\mathbf{x}_1, \mathbf{x}_2)\right]
\end{aligned} \tag{3.108}$$

Here we have defined:

$$\begin{aligned}
Q(\mathbf{x}_1, \mathbf{x}_2) &= (\mathbf{x} - \mathbf{m})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{m}) \\
&= [(\mathbf{x}_1 - \mathbf{m}_1)^T, (\mathbf{x}_2 - \mathbf{m}_2)^T] \begin{bmatrix} \boldsymbol{\Sigma}^{11} & \boldsymbol{\Sigma}^{12} \\ \boldsymbol{\Sigma}^{21} & \boldsymbol{\Sigma}^{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 - \mathbf{m}_1 \\ \mathbf{x}_2 - \mathbf{m}_2 \end{bmatrix} \\
&= (\mathbf{x}_1 - \mathbf{m}_1)^T \boldsymbol{\Sigma}^{11} (\mathbf{x}_1 - \mathbf{m}_1) + 2(\mathbf{x}_1 - \mathbf{m}_1)^T \boldsymbol{\Sigma}^{12} (\mathbf{x}_2 - \mathbf{m}_2) + (\mathbf{x}_2 - \mathbf{m}_2)^T \boldsymbol{\Sigma}^{22} (\mathbf{x}_2 - \mathbf{m}_2)
\end{aligned}$$

where we have assumed

$$\boldsymbol{\Sigma}^{-1} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{\Sigma}^{11} & \boldsymbol{\Sigma}^{12} \\ \boldsymbol{\Sigma}^{21} & \boldsymbol{\Sigma}^{22} \end{bmatrix} \quad (3.110)$$

According to the Woodbury matrix identify (Section ?? in Appendix A), we have

$$\begin{aligned}
\boldsymbol{\Sigma}^{11} &= (\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} \boldsymbol{\Sigma}_{21})^{-1} = \boldsymbol{\Sigma}_{11}^{-1} + \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12} (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1} \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \\
\boldsymbol{\Sigma}^{22} &= (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1} = \boldsymbol{\Sigma}_{22}^{-1} + \boldsymbol{\Sigma}_{22}^{-1} \boldsymbol{\Sigma}_{21} (\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} \boldsymbol{\Sigma}_{21})^{-1} \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} \\
\boldsymbol{\Sigma}^{12} &= -\boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12} (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1} = (\boldsymbol{\Sigma}^{12})
\end{aligned} \quad (3.111)$$

Substituting the second expression for  $\boldsymbol{\Sigma}^{11}$ , the first expression for  $\boldsymbol{\Sigma}^{22}$ , and  $\boldsymbol{\Sigma}^{12}$  into  $Q(\mathbf{x}_1, \mathbf{x}_2)$  we get:

$$\begin{aligned}
Q(\mathbf{x}_1, \mathbf{x}_2) &= (\mathbf{x}_1 - \mathbf{m}_1)^T [\boldsymbol{\Sigma}_{11}^{-1} + \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12} (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1} \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1}] (\mathbf{x}_1 - \mathbf{m}_1) \\
&\quad - 2(\mathbf{x}_1 - \mathbf{m}_1)^T [\boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12} (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1}] (\mathbf{x}_2 - \mathbf{m}_2) \\
&\quad + (\mathbf{x}_2 - \mathbf{m}_2)^T [(\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1}] (\mathbf{x}_2 - \mathbf{m}_2) \\
&= (\mathbf{x}_1 - \mathbf{m}_1)^T \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1) \\
&\quad + (\mathbf{x}_1 - \mathbf{m}_1)^T \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12} (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1} \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1) \\
&\quad - 2(\mathbf{x}_1 - \mathbf{m}_1)^T \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12} (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1} (\mathbf{x}_2 - \mathbf{m}_2) \\
&\quad + (\mathbf{x}_2 - \mathbf{m}_2)^T (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1} (\mathbf{x}_2 - \mathbf{m}_2) \\
&= (\mathbf{x}_1 - \mathbf{m}_1)^T \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1) \\
&\quad + [(\mathbf{x}_2 - \mathbf{m}_2) - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1)]^T (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1} [(\mathbf{x}_2 - \mathbf{m}_2) - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1)]
\end{aligned}$$

The last equal sign is due to the following equations for any vectors  $u$  and  $v$  and a symmetric matrix  $A = A^T$ :

$$\begin{aligned}
&u^T A u - 2u^T A v + v^T A v = u^T A u - u^T A v - u^T A v + v^T A v \\
&= u^T A (u - v) - (u - v)^T A v = u^T A (u - v) - v^T A (u - v) \\
&= (u - v)^T A (u - v) = (v - u)^T A (v - u)
\end{aligned} \quad (3.113)$$

We further define

$$\begin{cases} \mathbf{b} = \mathbf{m}_2 + \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1) \\ \mathbf{A} = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12} \end{cases} \quad (3.114)$$

and

$$\begin{cases} Q_1(\mathbf{x}_1) &= (\mathbf{x}_1 - \mathbf{m}_1)^T \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1) \\ Q_2(\mathbf{x}_1, \mathbf{x}_2) &= [(\mathbf{x}_2 - \mathbf{m}_2) - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1)]^T (\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12})^{-1} [(\mathbf{x}_2 - \mathbf{m}_2) - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1)] \\ &= (\mathbf{x}_2 - \mathbf{b})^T \mathbf{A}^{-1} (\mathbf{x}_2 - \mathbf{b}) \end{cases} \quad (3.115)$$

then the expression above for  $Q(\mathbf{x}_1, \mathbf{x}_2)$  can be written as:

$$Q(\mathbf{x}_1, \mathbf{x}_2) = Q_1(\mathbf{x}_1) + Q_2(\mathbf{x}_1, \mathbf{x}_2) \quad (3.116)$$

and the joint distribution can be written as:

$$\begin{aligned} p(\mathbf{x}) &= p(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{(2\pi)^{n/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left[ -\frac{1}{2} Q(\mathbf{x}_1, \mathbf{x}_2) \right] \\ &= \frac{1}{(2\pi)^{n/2} |\boldsymbol{\Sigma}_{11}|^{1/2} |\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12}|^{1/2}} \exp \left[ -\frac{1}{2} Q_1(\mathbf{x}_1) \right] \exp \left[ -\frac{1}{2} Q_2(\mathbf{x}_1, \mathbf{x}_2) \right] \\ &= \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}_{11}|^{1/2}} \exp \left[ -\frac{1}{2} (\mathbf{x}_1 - \mathbf{m}_1)^T \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1) \right] \frac{1}{(2\pi)^{q/2} |\mathbf{A}|^{1/2}} \exp \left[ -\frac{1}{2} (\mathbf{x}_2 - \mathbf{b})^T \mathbf{A}^{-1} (\mathbf{x}_2 - \mathbf{b}) \right] \\ &= \mathcal{N}(\mathbf{x}_1, \mathbf{m}_1, \boldsymbol{\Sigma}_{11}) \mathcal{N}(\mathbf{x}_2, \mathbf{b}, \mathbf{A}) \end{aligned} \quad (3.117)$$

where we have used theorem 2 in Section 7 in Appendix A:

$$|\boldsymbol{\Sigma}| = |\boldsymbol{\Sigma}_{11}| |\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12}| \quad (3.118)$$

The marginal distribution of  $\mathbf{x}_1$  is

$$\begin{aligned} p_1(\mathbf{x}_1) &= \int p(\mathbf{x}_1, \mathbf{x}_2) d\mathbf{x}_2 = \mathcal{N}(\mathbf{x}_1, \mathbf{m}_1, \boldsymbol{\Sigma}_{11}) \int \mathcal{N}(\mathbf{x}_2, \mathbf{b}, \mathbf{A}) d\mathbf{x}_2 \\ &= \mathcal{N}(\mathbf{x}_1, \mathbf{m}_1, \boldsymbol{\Sigma}_{11}) = \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}_{11}|^{1/2}} \exp \left[ -\frac{1}{2} (\mathbf{x}_1 - \mathbf{m}_1)^T \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1) \right] \end{aligned} \quad (3.119)$$

and the conditional distribution of  $\mathbf{x}_2$  given  $\mathbf{x}_1$  is

$$p_{2|1}(\mathbf{x}_2 | \mathbf{x}_1) = \frac{p(\mathbf{x}_1, \mathbf{x}_2)}{p_1(\mathbf{x}_1)} = \mathcal{N}(\mathbf{x}_2, \mathbf{b}, \mathbf{A}) = \frac{1}{(2\pi)^{q/2} |\mathbf{A}|^{1/2}} \exp \left[ -\frac{1}{2} (\mathbf{x}_2 - \mathbf{b})^T \mathbf{A}^{-1} (\mathbf{x}_2 - \mathbf{b}) \right] \quad (3.120)$$

with

$$\begin{cases} \mathbf{b} &= \mathbf{m}_{2/1} = \mathbf{m}_2 + \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \mathbf{m}_1) \\ \mathbf{A} &= \boldsymbol{\Sigma}_{2/1} = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12} \end{cases} \quad (3.121)$$

### Drawing samples from Gaussian distributions

We first consider drawing samples from a random variable  $x$  with a normal distribution  $p(x) = \mathcal{N}(0, 1)$ , i.e.,  $E(x) = 0$  and  $Var(x) = E(x^2) = 1$ . As shown in the figure, a random variable uniformly distributed in the range  $(0, 1)$  (vertical) can be mapped to another random variable  $x$  in the range  $(-\infty, \infty)$  (horizontal) with a Gaussian distribution  $p(x) = \mathcal{N}(0, 1)$ , by the inverse function of its accumulative  $F(x) = \int_{-\infty}^x p(u) du$ . This variable  $x$  can be further converted into another variable  $x' = \mu + \sigma x$ , whose distribution is a generic Gaussian

$p(x') = \mathcal{N}(\mu, \sigma^2)$ , with the mean and variance

$$\begin{aligned} E(x') &= E(\mu + \sigma x) = \mu + \sigma E(x) = \mu \\ \text{Var}(x') &= [(x' - \mu)^2] = E(\sigma^2 x) = E(\sigma^2 x^2) = \sigma^2 E(x^2) = \sigma^2 \end{aligned} \quad (3.122)$$

Using this method, we can generate a set of  $n$  normally distributed and independent random variables  $x_1, \dots, x_n$ , each with  $E(x_i) = 0$ ,  $\text{Var}(x_i) = 1$ , and  $\text{Cov}(x_i, x_j) = \delta_{ij}$ , and, based on these variables, we can further construct a random vector  $\mathbf{x} = [x_1, \dots, x_n]^T$  with a Gaussian joint distributed  $p(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ .

Given such an  $\mathbf{x}$  with  $\mathbf{m}_x = \mathbf{0}$  and  $\mathbf{\Sigma}_x = \mathbf{I}$ , we can generate another random vector with a Gaussian distribution  $\mathcal{N}(\mathbf{m}, \mathbf{\Sigma})$ . To do so, consider the affine transformation  $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{m}$ , which also has a Gaussian distribution:

$$\begin{aligned} \mathbf{m}_y &= E(\mathbf{y}) = \mathbf{W}\mathbf{m}_x + \mathbf{m} = \mathbf{m} \\ \mathbf{\Sigma}_y &= \mathbf{W}\mathbf{\Sigma}_x\mathbf{W}^T = \mathbf{W}\mathbf{W}^T \end{aligned} \quad (3.123)$$

While  $\mathbf{m}_y = \mathbf{m}$  as desired, we still need to make sure that  $\mathbf{\Sigma}_y = \mathbf{W}\mathbf{W}^T = \mathbf{\Sigma}$  also as desired. There exist infinite matrices all satisfying the equation above. Typically, such a  $\mathbf{W}$  can be found in either of the following two methods:

- The Cholesky decomposition: which can find an upper triangular matrix  $\mathbf{L}$  that satisfies  $\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^T$  (if  $\mathbf{\Sigma}$  is nonsingular).
- The eigenvalue decomposition  $\mathbf{\Sigma} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$  (even if  $\mathbf{\Sigma}$  is singular), where  $\mathbf{V}$  and  $\mathbf{\Lambda}$  are respectively the eigenvector and eigenvalue matrices of  $\mathbf{\Sigma}$ . As  $\mathbf{\Sigma}^T = \mathbf{\Sigma}$  is symmetric and positive definite, its eigenvalues take positive real values and its eigenvectors are orthogonal to each other, i.e.,  $\mathbf{V}^T = \mathbf{V}^{-1}$  is orthogonal. If we let  $\mathbf{W} = \mathbf{V}\mathbf{\Lambda}^{1/2}$ , then we have

$$\mathbf{W}\mathbf{W}^T = (\mathbf{V}\mathbf{\Lambda}^{1/2})(\mathbf{V}\mathbf{\Lambda}^{1/2})^T = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T = \mathbf{\Sigma} \quad (3.124)$$

We note that although  $\mathbf{L}\mathbf{L}^T = \mathbf{W}\mathbf{W}^T = \mathbf{\Sigma}$ ,  $\mathbf{L}$  found by the Cholesky decomposition method is upper triangular but  $\mathbf{W}$  is not,  $\mathbf{L} \neq \mathbf{W}$ . Also, we note that the transform based on eigenvalue decomposition is optimal in certain way, as to be discussed later.

Having found either  $\mathbf{L}$  or  $\mathbf{W}$ , we get  $\mathbf{y} = \mathbf{L}\mathbf{x} + \mathbf{m}$  or  $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{m}$  with  $p(\mathbf{y}) = \mathcal{N}(\mathbf{m}, \mathbf{\Sigma})$  as desired.

### 3.1.4 Function of Random Variables

We first find the mean and variance of  $y = cx$  where  $x$  is a random variable and  $c$  is the constant:

$$\begin{aligned} \mu_y &= E(y) = E(cx) = cE(x) = c\mu_x \\ \sigma_y &= E(y^2) - \mu_y^2 = E[(cx)^2] - (c\mu_x)^2 = c^2 E(x^2) - c^2 \mu_x^2 = c^2(E(x^2) - \mu_x^2) = c^2\sigma_x^2 \end{aligned}$$

We next consider the mean and variance of  $y = x_1 + \dots + x_n$  as the sum of  $n$

random variables:

$$\mu_y = E(y) = E\left(\sum_{i=1}^n x_i\right) = \sum_{i=1}^n E(x_i) = \sum_{i=1}^n \mu_{x_i} \quad (3.126)$$

We also have

$$E(y^2) = E\left(\sum_{i=1}^n x_i\right)^2 = E\left(\sum_{i=1}^n \sum_{j=1}^n x_i x_j\right) = \sum_{i=1}^n \sum_{j=1}^n E(x_i x_j) \quad (3.127)$$

and

$$(E(y))^2 = \left(\sum_{i=1}^n E(x_i)\right)^2 = \sum_{i=1}^n \sum_{j=1}^n E(x_i)E(x_j) \quad (3.128)$$

then we get

$$Var(y) = E(y^2) - (E(y))^2 = \sum_{i=1}^n \sum_{j=1}^n E(x_i x_j) - \sum_{i=1}^n \sum_{j=1}^n E(x_i)E(x_j) = \sum_{i=1}^n \sum_{j=1}^n Cov(x_i, x_j) \quad (3.129)$$

Only if all  $x_i$  are uncorrelated (or more specially independent), i.e.,  $Cov(x_i, x_j) = 0$  for  $i \neq j$ , do we have

$$\sigma_y^2 = Var(y) = \sum_{i=1}^n \sum_{j=1}^n Cov(x_i, x_j) = \sum_{i=1}^n Cov(x_i, x_i) = \sum_{i=1}^n Var(x_i) = \sum_{i=1}^n \sigma_{x_i}^2 \quad (3.130)$$

We further consider in general  $y = f(x)$  as a nonlinear function of a random variable  $x$  is also a random variable, which can be approximated based on its Taylor expansion near the mean  $\mu_x$  of  $x$ :

$$y = f(x) = f(\mu_x) + f'(\mu_x)(x - \mu_x) + \frac{f''(\mu_x)}{2}(x - \mu_x)^2 + \dots \approx f(\mu_x) + f'(\mu_x)(x - \mu_x) \quad (3.131)$$

The approximation is due to the assumption that  $|x - \mu_x|$  is small. Taking the mean on both side we get

$$\mu_y = E(y) \approx E[f(\mu_x) + f'(\mu_x)(x - \mu_x)] = f(\mu_x) \quad (3.132)$$

Taking the variance on both sides we get

$$Var(y) \approx Var[f(\mu_x) + f'(\mu_x)(x - \mu_x)] = Var[f'(\mu_x)(x - \mu_x)] = f'^2(\mu_x)Var(x) = f'^2(\mu_x)\sigma_x^2 \quad (3.133)$$

Finally to find the mean and variance of  $y = f(x_1, \dots, x_n) = f(\mathbf{x})$  as a nonlinear function of  $n$  random variables represented as an  $n$ -D vector  $\mathbf{x} = [x_1, \dots, x_n]^T$ , consider its Taylor expansion in the neighborhood of the mean

vector  $\mathbf{m}_x = [\mu_{x_1}, \dots, \mu_{x_n}]^T$ :

$$\begin{aligned} f(x_1, \dots, x_n) &= f(\mu_{x_1}, \dots, \mu_{x_n}) + \sum_{i=1}^n \frac{\partial f}{\partial x_i} (x_i - \mu_{x_i}) + \sum_{i=1}^n \sum_{j=1}^n \frac{\partial^2 f}{\partial x_i \partial x_j} (x_i - \mu_{x_i})(x_j - \mu_{x_j}) + \dots \\ &\approx f(\mu_{x_1}, \dots, \mu_{x_n}) + \sum_{i=1}^n \frac{\partial f}{\partial x_i} (x_i - \mu_{x_i}) \end{aligned} \quad (3.134)$$

or in matrix form:

$$f(\mathbf{x}) = f(\mathbf{m}_x) + \mathbf{g}(\mathbf{m}_x)^T (\mathbf{x} - \mathbf{m}_x) + \frac{1}{2} (\mathbf{x} - \mathbf{m}_x)^T \mathbf{H}(\mathbf{m}_x) (\mathbf{x} - \mathbf{m}_x) + \dots \approx f(\mathbf{m}_x) + \mathbf{g}(\mathbf{m}_x)^T (\mathbf{x} - \mathbf{m}_x) \quad (3.135)$$

where

$$\mathbf{g}(\mathbf{m}_x) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}_{\mathbf{m}_x}, \quad \mathbf{H}(\mathbf{m}_x) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \dots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \dots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix}_{\mathbf{m}_x} \quad (3.136)$$

are the gradient vector and Hessian matrix of  $f(\mathbf{x})$  evaluated at  $\mathbf{m}_x$ . Taking the mean of the expression above

$$\begin{aligned} \mu_y = E f(\mathbf{x}) &\approx E \left[ f(\mu_{x_1}, \dots, \mu_{x_n}) + \sum_{i=1}^n \frac{\partial f}{\partial x_i} (x_i - \mu_{x_i}) \right] \\ &= f(\mu_{x_1}, \dots, \mu_{x_n}) + \sum_{i=1}^n \frac{\partial f}{\partial x_i} E(x_i - \mu_{x_i}) = f(\mu_{x_1}, \dots, \mu_{x_n}) \end{aligned} \quad (3.137)$$

Taking the variance we get

$$\begin{aligned} \sigma_y^2 &= Var [f((x_1, \dots, x_n))] \approx Var \left[ f(\mu_{x_1}, \dots, \mu_{x_n}) + \sum_{i=1}^n \frac{\partial f}{\partial x_i} (x_i - \mu_{x_i}) \right] \\ &= Var \left[ \sum_{i=1}^n \frac{\partial f}{\partial x_i} (x_i - \mu_{x_i}) \right] = \sum_{i=1}^n \left( \frac{\partial f}{\partial x_i} \right)^2 \sigma_{x_i}^2 \end{aligned} \quad (3.138)$$

or

$$\sigma_y = \sqrt{\sum_{i=1}^n \left( \frac{\partial f}{\partial x_i} \right)^2 \sigma_{x_i}^2} \quad (3.139)$$

## 3.2 Entropy and Information

### 3.2.1 Entropy

Entropy is a measurement of the uncertainty of the outcome of a random event, which also measures the amount of information obtained in terms of the reduction of uncertainty (due to certain communication or xperiment). If the uncertainty

is reduced to zero by a communication, then the amount of information obtained is exactly the same as the uncertainty before the communication.

A random event of  $n$  outcomes can be represented by a discrete random variable  $x$  with  $N$  possible values  $\{x_1, \dots, x_n\}$  with the corresponding probabilities  $P_i = P(x_i)$  satisfying  $\sum_{i=1}^n P_i = 1$ . For example,  $n = 26$  letters in the English alphabet, with different probabilities to be used (e.g.,  $P(x = 'a') > P(x = 'z')$ ).

To quantitatively measure the *uncertainty* of random variable  $x$  denoted by  $H(x)$ , we define the uncertainty or information content (surprise) of each outcome  $x_i$  as

$$H(x_i) = \log \frac{1}{P_i} = -\log P_i, \quad (i = 1, \dots, n) \quad (3.140)$$

so that the desired properties listed below are satisfied:

- Uncertainty is inversely related to the probability, i.e., a more probable outcome is less uncertain than a less probable outcome. If  $P_i > P_j$ , then

$$H(x_i) = -\log P_i < H(x_j) = -\log P_j \quad (3.141)$$

- Specially, an certain outcome  $x$  with  $P(x_i) = 1$  has zero uncertainty, and an impossible outcome  $x$  with  $P(x_j) = 0$  has infinite uncertainty:

$$H(x_i) = -\log P_i = -\log 1 = 0, \quad H(x_j) = -\log P_j = -\log 0 = \infty \quad (3.142)$$

- The total uncertainty of  $x$  is the expectation of the uncertainties of the  $n$  outcomes:

$$H(x) = E[H(x_i)] = E[-\log P_i] = -\sum_{i=1}^n P_i \log P_i \quad (3.143)$$

The total uncertainty  $H(x)$  defined above, called the *entropy*, measures the total uncertainty of the random event represented by  $x$ , which is also the total amount of information contained in  $x$ . Once the outcome of the random event is known, the information contained in  $x$  is gained, and the uncertainty is reduced to zero.

The  $n$  outcomes of  $x$  can be binary encoded so that each outcome  $x_i$  is represented by a sequence of 0's and 1's. To minimize the total number of bits needed to encode  $x$ , we encode  $x_i$  with  $P_i$  by  $H(x_i) = -\log_2 P_i$  bits, so that a more probable outcome  $x_i$  with greater  $P_i$  (less uncertain and containing less information) is encoded by fewer bits, and a less probable outcome  $x_j$  with smaller  $P_j$  (more uncertain and containing more information) is encoded by more bits. Then the expected number of bits needed to encode  $x$  is minimized.

We can show that a uniform distribution of  $n = 2^m$  equally likely outcomes with  $p_i = 1/n$  ( $i = 1, \dots, n$ ) has the maximum entropy. This is a constrained

maximization problem which can be solved by Lagrange multiplier method:

$$\begin{aligned} \frac{\partial}{\partial P_j} \left[ -\sum_{i=1}^n P_i \log P_i + \lambda \left( 1 - \sum_{i=0}^n P_i \right) \right] &= 0 \\ = -\frac{\partial}{\partial P_j} P_j \log P_j - \lambda &= -(\log P_j + 1) - \lambda = 0 \end{aligned} \quad (3.144)$$

Solving this we get

$$P_j = 2^{-\lambda-1} \quad (j = 1, \dots, N) \quad (3.145)$$

which must satisfy

$$\sum_{j=1}^n P_j = \sum_{j=1}^n 2^{-\lambda-1} = 2^{-\lambda-1} n = 1 \quad (3.146)$$

Solving for  $\lambda$  we get

$$\lambda = \log n - 1 \quad (3.147)$$

Substituting back into  $P_j$  we get

$$P_j = 2^{-\lambda-1} = 2^{-\log n} = \frac{1}{n}, \quad (j = 1, \dots, n) \quad (3.148)$$

The corresponding maximum entropy can be obtained as

$$H_{max}(E_1, \dots, E_n) = -\sum_{i=1}^n \frac{1}{n} \log_2 \frac{1}{n} = \log_2 n = m \quad (3.149)$$

In this case, each outcome  $x_i$  is encoded by

$$\log_2 \frac{1}{P_i} = \log_2 \left( \frac{1}{1/2^n} \right) = \log_2 2^n = n \quad (3.150)$$

bits, and the total number of bits needed to encode this random event is

$$-\sum_{i=1}^n P_i \log_2 P_i = -\sum_{i=1}^n \frac{1}{n} \log_2 \frac{1}{2^n} = \sum_{i=1}^n \frac{1}{n} \log_2 2^n = m \quad (3.151)$$

If  $n = 2^m = 2^3 = 8$ ,  $P_i = 1/2^3 = 1/8$ ,  $n = 3$  bits are needed to encode the  $n = 8$  possible outcomes.

For example, in the special case of  $n = 2$  outcomes:

- If  $P_1 = 1$  and  $P_2 = 0$ , we have the minimum uncertainty zero ( $x \log_2 x|_{x=0} = 0$ ), i.e., 0 bit is needed to encode a sure event:

$$H = -1 \log_2 1 - 0 \log_2 0 = 0 \quad (3.152)$$

- If  $P_1 = 0.1$ ,  $P_2 = 0.9$ , the uncertainty is:

$$H = -(0.1 \log_2 0.1 + 0.9 \log_2 0.9) = 0.47 \quad (3.153)$$

- If  $P_1 = 0.2$ ,  $P_2 = 0.8$ , the uncertainty is:

$$H = -(0.2 \log_2 0.2 + 0.8 \log_2 0.8) = 0.72 \quad (3.154)$$



- If  $P_1 = 0.3$ ,  $P_2 = 0.7$ , the uncertainty is:

$$H = -(0.3 \log_2 0.3 + 0.7 \log_2 0.7) = 0.88 \quad (3.155)$$

- If  $P_1 = 0.4$ ,  $P_2 = 0.6$ , the uncertainty is:

$$H = -(0.4 \log_2 0.4 + 0.6 \log_2 0.6) = 0.97 \quad (3.156)$$

- If  $P_1 = P_2 = 0.5 = 1/2$ , we have the maximum uncertainty  $H = 1$ , i.e., 1 bit (0 or 1) is needed to encode the event:

$$H = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = \frac{1}{2} + \frac{1}{2} = 1 \quad (3.157)$$

The discussion above for a discrete random variable  $x$  can be extended to a continuous random variable  $x$  with pdf  $p(x)$ , of which the *differential entropy* is defined as

$$H(p) = - \int p(x) \log p(x) dx \quad (3.158)$$

The *perplexity* of a discrete probability distribution  $P$  is simply defined as  $2^{H(p)}$ .

We consider the pdf  $p(x)$  that maximizes the entropy under each of the following two conditions:

- The  $p(x)$  is non-zero only inside a closed region  $x \in [a, b]$ .

To find the pdf  $p(x)$  that maximizes  $H(p)$  under the condition  $\int_a^b p(x) dx = 1$ , we construct its Lagrangian function

$$L(p) = H(p) + \lambda \left( 1 - \int_a^b p(x) dx \right) = \int_a^b [-p(x) \log p(x) - \lambda p(x)] dx + \lambda \quad (3.159)$$

and set its derivative with respect to  $p(x)$  (with a specific value of  $x$ ) to zero:

$$\frac{dL(p)}{dp(x)} = -(\log p(x) + 1) - \lambda = 0 \quad (3.160)$$

Solving for  $p(x)$  we get

$$p(x) = e^{-\lambda-1} \quad (3.161)$$

which must satisfy

$$\int_{-\infty}^{\infty} p(x) dx = \int_a^b e^{-\lambda-1} dx = e^{-\lambda-1}(b-a) = 1 \quad (3.162)$$

i.e.,

$$p(x) = e^{-\lambda-1} = \frac{1}{b-a} \quad (3.163)$$

This is a uniform distribution over  $[a, b]$ . The corresponding maximum entropy is

$$H(p) = - \int_a^b \frac{1}{b-a} \log \left( \frac{1}{b-a} \right) = \log(b-a) \quad (3.164)$$

- The random variable  $x \in (-\infty, \infty)$  with a known variance

$$\sigma^2 = \int_{-\infty}^{\infty} (x - \mu)^2 p(x) dx \quad (3.165)$$

To find the pdf  $p(x)$  with  $\sigma^2$  that maximizes  $H(p)$ , we consider the following Lagrangian function

$$L(p) = - \int_{-\infty}^{\infty} p(x) \log p(x) dx + \lambda_1 \left( 1 - \int_{-\infty}^{\infty} p(x) dx \right) + \lambda_2 \left( \sigma^2 - \int_{-\infty}^{\infty} (x - \mu)^2 p(x) dx \right) \quad (3.166)$$

and set its derivative with respect to  $p(x)$  (evaluated at certain  $x$ ) to zero:

$$\frac{dL(p)}{dp(x)} = -(\log p(x) + 1) - \lambda_1 - \lambda_2(x - \mu)^2 = 0 \quad (3.167)$$

Solving for  $p(x)$  we get

$$p(x) = e^{-(\lambda_1 + \lambda_2(x - \mu)^2 + 1)} = e^{-(\lambda_1 + 1)} e^{-\lambda_2(x - \mu)^2} \quad (3.168)$$

which must satisfy the two constraints:

$$1 = \int_{-\infty}^{\infty} p(x) dx = e^{-\lambda_1 - 1} \int_{-\infty}^{\infty} e^{-\lambda_2(x - \mu)^2} dx = e^{-(\lambda_1 + 1)} \sqrt{\frac{\pi}{\lambda_2}} \quad (3.169)$$

and

$$\begin{aligned} \sigma^2 &= \int_{-\infty}^{\infty} (x - \mu)^2 p(x) dx = e^{-\lambda_1 - 1} \int_{-\infty}^{\infty} e^{-\lambda_2(x - \mu)^2} (x - \mu)^2 dx \\ &= e^{-\lambda_1 - 1} \frac{1}{2} \sqrt{\frac{\pi}{\lambda_2^3}} = e^{-\lambda_1 - 1} \frac{1}{2\lambda_2} \sqrt{\frac{\pi}{\lambda_2}} \end{aligned} \quad (3.170)$$

Comparing the two equations above, we get

$$\lambda_2 = \frac{1}{2\sigma^2} \quad (3.171)$$

Substituting this into Eq. (3.169), we get

$$e^{-(\lambda_1 + 1)} \sqrt{2\pi\sigma^2} = 1, \quad \text{i.e.} \quad e^{-(\lambda_1 + 1)} = \frac{1}{\sqrt{2\pi\sigma^2}} \quad (3.172)$$

Substituting this together with  $\lambda_2 = 1/2\sigma^2$  into  $p(x)$  we get

$$p(x) = e^{-(\lambda_1 + 1)} e^{-\lambda_2(x - \mu)^2} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x - \mu)^2}{2\sigma^2}} = \mathcal{N}(x, \mu, \sigma^2) \quad (3.173)$$

This is the normal or Gaussian distribution and the corresponding maxi-

mum entropy is

$$\begin{aligned}
H(g) &= - \int_{-\infty}^{\infty} g(x) \log g(x) dx = - \int_{-\infty}^{\infty} g(x) \log \left( \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \right) dx \\
&= - \int_{-\infty}^{\infty} g(x) \left( \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{(x-\mu)^2}{2\sigma^2} \right) dx \\
&= \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \int_{-\infty}^{\infty} (x-\mu)^2 g(x) dx \\
&= \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sigma^2 = \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2} \log(e) \\
&= \frac{1}{2} \log(2\pi e \sigma^2) = \frac{1}{2} \log(2\pi e) + \log \sigma
\end{aligned} \tag{3.174}$$

The uncertainty of a distribution  $p(x)$  can be measured by both its variance  $\sigma^2 = \int (x - \mu)^2 p(x) dx$  and its entropy  $H = - \int p(x) \log p(x) dx$ . However, they are also different in that  $\sigma^2$  represents the spread of  $p(x)$  while  $H$  captures the shape of  $p(x)$ . In multimodal distribution (with two or more modes),  $H$  is a better measurement as it increases due to the multimodes while  $\sigma^2$  is not sensitive to them.

In data analysis and modeling, if no additional knowledge is available regarding the observed data other than the variance (or covariance for multivariate data), the normal distribution with maximum entropy should be used as least-informative default to model the data, as it will impose the least amount of unsupported constraint and thereby causing minimum bias. This is the *principle of maximum entropy*.

- **Joint Entropy**

The joint entropy of two discrete random variables  $x$  and  $y$  is defined as:

$$H(x, y) = -E[\log P(x_i, y_j)] = - \sum_i \sum_j P(x_i, y_j) \log P(x_i, y_j) \tag{3.175}$$

More generally, the joint entropy of  $n$  discrete random variables in  $\mathbf{x} = [x_1, \dots, x_n]^T$  is defined as:

$$\begin{aligned}
H(\mathbf{x}) &= H(x_1, \dots, x_n) = -E[\log P(x_1, \dots, x_n)] \\
&= - \sum_{x_1} \dots \sum_{x_n} P(x_1, \dots, x_n) \log P(x_1, \dots, x_n) \\
&= - \sum_{x_1} \dots \sum_{x_n} P(\mathbf{x}) \log P(\mathbf{x})
\end{aligned} \tag{3.176}$$

The differential entropy is

$$\begin{aligned}
H(\mathbf{x}) &= H(x_1, \dots, x_n) = - \int \dots \int p(x_1, \dots, x_n) \log P(x_1, \dots, x_n) dx_1 \dots dx_n \\
&= - \int p(\mathbf{x}) \log P(\mathbf{x}) d\mathbf{x}
\end{aligned} \tag{3.177}$$

- **Entropy of function of random variables**

Given the entropy  $H(\mathbf{x})$  of  $\mathbf{x}$ , we can further find the entropy of a function  $\mathbf{y} = \mathbf{f}(\mathbf{x})$  of  $\mathbf{x}$ . We first recall that the pdf of  $\mathbf{y}$  can be expressed in terms of that of  $\mathbf{x}$  in Eq. (3.74):  $p_Y(\mathbf{y}) = |\det \mathbf{J}_{f^{-1}}| p_X(\mathbf{x})$ . We can then find  $H(\mathbf{y})$  directly as:

$$\begin{aligned}
 H(\mathbf{y}) &= - \int p_Y(\mathbf{y}) \log p_Y(\mathbf{y}) d\mathbf{y} \\
 &= - \int |\det \mathbf{J}_{f^{-1}}| p_X(\mathbf{x}) \log (|\det \mathbf{J}_{f^{-1}}| p_X(\mathbf{x})) |\det \mathbf{J}_f| d\mathbf{x} \\
 &= - \int p_X(\mathbf{x}) [\log |\det \mathbf{J}_{f^{-1}}| + \log p_X(\mathbf{x})] d\mathbf{x} \\
 &= - \int p_X(\mathbf{x}) \log p_X(\mathbf{x}) d\mathbf{x} - \int p_X(\mathbf{x}) \log |\det \mathbf{J}_{f^{-1}}| d\mathbf{x} \\
 &= H(\mathbf{x}) + E[\log |\det \mathbf{J}_f|] \tag{3.178}
 \end{aligned}$$

Here we have used integration by substitution and also the inverse function theorem stating that given the Jacobian  $\mathbf{J}_f$  of a function  $\mathbf{y} = \mathbf{f}(\mathbf{x})$ , the Jacobian of the inverse function  $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{y})$  is  $\mathbf{J}_{f^{-1}} = \mathbf{J}_f^{-1}$ . Also as in general  $\det(\mathbf{A}^{-1}) = (\det(\mathbf{A}))^{-1}$ , we have  $|\det \mathbf{J}_{f^{-1}}| |\det \mathbf{J}_f| = 1$ .

Specially for a linear function  $\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$ , we have  $\mathbf{J}_f = d\mathbf{f}(\mathbf{x})/d\mathbf{x} = \mathbf{A}$  and  $\mathbf{J}_{f^{-1}} = \mathbf{A}^{-1}$ , then we have

$$H(\mathbf{y}) = H(\mathbf{x}) + \log |\det \mathbf{A}| \tag{3.179}$$

- **Conditional Entropy**

The *conditional entropy*  $H(y|x)$  measures the uncertainty of a random variable  $y$  given the value of another random variable  $x$ , or the amount of information gained once the outcome of  $y$  is known, given the outcome of  $x$ .

Let  $H(y|x = x_i)$  be the entropy of  $y$  conditioned on  $x = x_i$ :

$$H(y|x = x_i) = - \sum_j P(y_j|x_i) \log P(y_j|x_i) \tag{3.180}$$

Then the conditional entropy  $H(y|x)$  is defined as the average of  $H(y|x = x_i)$  over all outcomes of  $x$ :

$$\begin{aligned}
 H(y|x) &= E_x[H(y|x = x_i)] = \sum_i P(x_i) H(y|x = x_i) \\
 &= - \sum_i P(x_i) \sum_j P(y_j|x_i) \log P(y_j|x_i) = - \sum_i \sum_j P(x_i) P(y_j|x_i) \log P(y_j|x_i) \\
 &= - \sum_i \sum_j P(x_i, y_j) \log P(y_j|x_i) \tag{3.181}
 \end{aligned}$$

The last equality is due to  $P(x, y) = P(y|x)P(x)$ .

**Theorem:**

$$\begin{aligned}
H(y|x) &= - \sum_i \sum_j P(x_i, y_j) \log P(y_j|x_i) = \sum_i \sum_j P(x_i, y_j) \log \frac{P(x_i)}{P(x_i, y_j)} \\
&= - \sum_i \sum_j P(x_i, y_j) \log P(x_i, y_j) + \sum_i \sum_j P(x_i, y_j) \log P(x_i) \\
&= H(x, y) + \sum_i P(x_i) \log P(x_i) = H(x, y) - H(x) \quad (3.182)
\end{aligned}$$

Similarly, we also have  $H(x|y) = H(x, y) - H(y)$ . We therefore have

$$H(x, y) = H(y|x) + H(x) = H(x|y) + H(y) \quad (3.183)$$

Substituting this into the previous equation we get *Bayes's rule* for conditional entropy:

$$H(y|x) = H(x|y) + H(y) - H(x) \quad (3.184)$$

### 3.2.2 Mutual Information and Kullback-Leibler Divergence

- **Mutual Information**

Mutual information  $I(x, y)$  measures the information shared by random variables  $x$  and  $y$ , i.e., how much knowing one of the two variables reduces uncertainty of the other.

$$\begin{aligned}
I(x, y) &= \sum_i \sum_j P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)P(y_j)} \\
&= \sum_i \sum_j P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)} - \sum_i \sum_j P(x_i, y_j) \log P(y_j) \\
&= \sum_i \sum_j P(y_j|x_i)P(x_i) \log P(y_j|x_i) - \sum_j \log P(y_j) \left( \sum_i P(x_i, y_j) \right) \\
&= \sum_i P(x_i) \sum_j P(y_j|x_i) \log P(y_j|x_i) - \sum_j P(y_j) \log P(y_j) \\
&= -H(y|x) + H(y) = H(y) - H(y|x) \\
&= H(x, y) - H(y|x) - H(x|y) = H(x) + H(y) - H(x, y) \quad (3.185)
\end{aligned}$$

The last two equalities are due to  $H(x, y) = H(x|y) + H(y)$  above.

If  $P(x, y) = P(x)P(y)$ , then  $\log P(x, y)/P(x)P(y) = \log 1 = 0$  and  $I(x, y) = 0$ , i.e., the mutual information of two independent random variables is zero.

These relationships can be summarized in the following diagram:

The concept of mutual information can be generalized from two variables

to multiple variables:

$$\begin{aligned} I(\mathbf{x}) &= I(x_1, \dots, x_n) = \sum_{i_1} \cdots \sum_{i_n} p(\mathbf{x}) \log \frac{\prod_{j=1}^n p(x_j)}{p(\mathbf{x})} \\ &= \sum_{i=1}^n H(x_i) - H(x_1, \dots, x_n) = \sum_{i=1}^n H(x_i) - H(\mathbf{x}) \end{aligned} \quad (3.186)$$

- **Cross Entropy**

When the true distribution  $P$  is unknown, the encoding of  $x$  can be based on another distribution  $Q$  as a model that approximates  $P$ . Then the average of the total number of bits needed is called the *cross-entropy*:

$$H(P, Q) = \sum_i P_i \log_2 \frac{1}{Q_i} = - \sum_i P_i \log Q_i = -E_P[\log Q] \quad (3.187)$$

which can be considered as the expectation of the logarithmic model probability  $\log Q_i$  with respect to the unknown ground truth probability  $P_i$ .

*Gibbs' inequality.* The cross entropy  $H(P, Q)$  is no smaller than the entropy  $H(P)$ :

$$H(P) = - \sum_i P_i \log P_i \leq H(P, Q) = - \sum_i P_i \log Q_i \quad (3.188)$$

The equality holds  $H(P, Q) = H(P)$  if and only if  $Q = P$ . For the model  $Q$  to be as close to the ground truth  $P$  as possible, we need to minimize the cross entropy  $H(P, Q)$ .

**Proof:** Based on inequality  $\log x \leq x - 1$ , we have

$$\sum_i P_i \log \left( \frac{Q_i}{P_i} \right) \leq \sum_i P_i \left( \frac{Q_i}{P_i} - 1 \right) = \sum_i Q_i - \sum_i P_i = 0 \quad (3.189)$$

i.e.,

$$\sum_i P_i \log \left( \frac{Q_i}{P_i} \right) = \sum_i P_i \log \left( \frac{1}{P_i} \right) + \sum_i P_i \log Q_i = H(P) - H(P, Q) \leq 0 \quad (3.190)$$

We see that the encoding of  $x$  based on model distribution  $Q$  always requires more bits than that based on the true distribution  $P$ .

- **Kullback-Leibler Divergence (Relative Entropy)**

The Kullback-Leibler (KL) divergence or relative entropy is the difference between the cross entropy  $H(P, Q)$  and the entropy  $H(P)$ :

$$\begin{aligned} D_{KL}(P||Q) &= H(P, Q) - H(P) = -E_P[\log Q] - E_P[\log P] \\ &= - \sum_i P_i \log Q_i - \left( - \sum_i P_i \log P_i \right) \\ &= \sum_i P_i (\log P_i - \log Q_i) = \sum_i P_i \log \left( \frac{P_i}{Q_i} \right) \geq 0 \end{aligned} \quad (3.191)$$

The KL-divergence represents the number of extra bits needed to encode  $x$

based on  $Q$  instead of  $P$ , or a measure of the error of using  $Q$  to approximate  $P$ , in terms of the amount of information lost, due to the inaccuracy of the model. In order to obtain the best model  $Q$  that optimally approximates  $P$ , we need to minimize  $D_{KL}(P||Q)$ .

In general,  $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ , the KL-divergence is not a distance metric as it is not symmetric (hence the name divergence rather than distance). The KL-divergence can be symmetrized to become the Jensen–Shannon (JS) divergence:

$$JS(P||Q) = \frac{1}{2} [D_{KL}(P||M) + D_{KL}(Q||M)], \quad \text{where } M = \frac{P + Q}{2} \tag{3.192}$$

As JS divergence is symmetric, it can be used as a distance measure for the similarity between two distributions  $P$  and  $Q$ .

The mutual information defined above can be expressed as the following KL-divergence:

$$I(x_1, \dots, x_n) = \sum_{i_1} \dots \sum_{i_n} P(x_1, \dots, x_n) \log \frac{P(x_1, \dots, x_n)}{\prod_{j=1}^n P(x_j)} = D_{KL}(P(\mathbf{x}) || \prod_{j=1}^n P(x_j)) \tag{3.193}$$

i.e., the mutual information is the error of using  $P(x)P(y)$  to model the joint probability  $P(x, y)$ . When  $x$  and  $y$  are independent of each other, i.e.,

$$P(x, y) = P(x)P(y), \quad P(x|y) = P(x), \quad P(y|x) = P(y) \tag{3.194}$$

their mutual information is equal to zero.

All discussions above for discrete variables can be generalized to continuous variables with summations replaced by integrals.

In particular, consider the cross-entropy of a Gaussian  $g(x)$  and an arbitrary pdf  $p(x)$  both with the same variance  $\sigma^2$ :

$$\begin{aligned} H(p, g) &= - \int p(x) \log g(x) dx = - \int p(x) \left( \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{(x - \mu)^2}{2\sigma^2} \right) dx \\ &= \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \int (x - \mu)^2 p(x) dx = \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sigma^2 \\ &= \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2} = \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2} \log e = \frac{1}{2} \log(2\pi e\sigma^2) \end{aligned} \tag{3.195}$$

We see that the cross-entropy  $H(p, g)$  of  $g(x)$  and  $p(x)$  is the same as the entropy of  $g(x)$ . But as we also know  $H(p, g) \geq H(p)$ , we have  $H(g) = H(p, g) \geq H(p)$ . We therefore conclude that among all pdfs  $p(x)$  with the same variance  $\sigma^2$ , the Gaussian  $N(x, \mu, \sigma^2)$  has the maximum entropy.

### 3.2.3 Entropy Maximization

There may be the need to find an unknown distribution  $p(x)$  of a random variable  $x$  based on the observed data as the averages of set observed experiment results

$f_i(x)$ :

$$E[f_i(x)] = \int_{-\infty}^{\infty} p(x) f_i(x) dx = c_i \quad (i = 1, \dots, m) \quad (3.196)$$

Among the infinite number of density functions all satisfying these equations as well as  $\int_{-\infty}^{\infty} p(x) dx = 1$ , the one with the maximum entropy  $H(p)$  is of the most interest. In the following we show that such a distribution has to take the following form:

$$p(x) = \exp\left(\lambda_0 + \sum_{i=1}^m \lambda_i f_i(x)\right) = A \exp\left(\sum_{i=1}^m \lambda_i f_i(x)\right) \quad (3.197)$$

where  $\lambda_0, \dots, \lambda_m$  and  $A = e^{\lambda_0}$  are constants with values for  $p(x)$  to satisfy Eq. (3.196).

To find this distribution  $p(x)$  with maximal  $H(p)$  and satisfying Eq. (3.196) as constraints, as well as  $\int_{-\infty}^{\infty} p(x) dx = 1$  required of all probability density distributions, we need to solve a constrained optimization problem with the Lagrangian:

$$L(p) = \int_{-\infty}^{\infty} p(x) \log p(x) dx + \lambda_0 \left( \int_{-\infty}^{\infty} p(x) dx - 1 \right) + \sum_{i=1}^m \lambda_i \left( \int_{-\infty}^{\infty} p(x) f_i(x) dx - 1 \right) \quad (3.198)$$

We then take its derivative with respect to  $p(x)$  corresponding to a specific value of  $x$  (i.e.,  $x$  is treated as constant and the integral is ignored), and set it to zero:

$$\begin{aligned} \frac{d}{dp} L(p) &= \frac{d}{dp} (p \log p) + \lambda_0 \frac{d}{dp} (p - 1) + \sum_{i=1}^m \lambda_i \frac{d}{dp} (p f_i(x) - 1) \\ &= \log p(x) + 1 - \lambda_0 - \sum_{i=1}^m \lambda_i f_i(x) = 0 \end{aligned} \quad (3.199)$$

Solving this equation we get the optimal distribution:

$$p(x) = \exp\left(\lambda_0 - 1 + \sum_{i=1}^m \lambda_i f_i(x)\right) \quad (3.200)$$

which is indeed in the form of Eq. (3.197).

Alternatively, the same result can also be obtained by comparing the entropies of an arbitrary density function  $q(x)$ , and another distribution  $p(x)$  in the form of Eq. (3.197), both assumed to satisfy Eq. (3.196), i.e.,

$$\int p(x) f_i(x) dx = \int q(x) f_i(x) dx = c_i \quad (i = 1, \dots, m) \quad (3.201)$$



We then consider  $H(q(x))$  to be compared to  $H(p(x))$ :

$$\begin{aligned}
 H(q) &= - \int q(x) \log q(x) dx = - \int q(x) \log \left( \frac{q(x)}{p(x)} p(x) \right) dx \\
 &= -D_{KL}(q||p) - \int q(x) \log p(x) dx \leq - \int q(x) \log p(x) dx \\
 &= - \int q(x) \left( \lambda_0 + \sum_{i=1}^m \lambda_i f_i(x) \right) dx = - \int p(x) \left( \lambda_0 + \sum_{i=1}^m \lambda_i f_i(x) \right) dx \\
 &= - \int p(x) \log p(x) dx = H(p) \tag{3.202}
 \end{aligned}$$

Here we have used the fact  $D_{KL}(q||p) \geq 0$  and the result in Eq. (3.201). This result again indicates that the distribution with maximal  $H(p)$  and satisfying Eq. (3.196) has to take the form in Eq. (3.197). The specific values of the  $m + 1$  constants  $\lambda_0, \dots, \lambda_m$  can be found by solving the  $m + 1$  equations including Eq. (3.196) and  $\int_{-\infty}^{\infty} p(x) dx = 1$ .

### 3.3 Bayesian Inference

#### 3.3.1 Bayes' Theorem

Bayesian inference is widely used in various machine learning algorithms, and it is based on Bayes' theorem:

$$p(A|B) = \frac{p(A, B)}{p(B)} = \frac{p(B|A) p(A)}{p(B)} \tag{3.203}$$

If  $\{A_1, \dots, A_n\}$  is a partition of sample space  $A$ , then

$$P(B) = \sum_{i=1}^n P(B|A_i) P(A_i), \quad P(A_i|B) = \frac{P(B|A_i) P(A_i)}{P(B)} = \frac{P(B|A_i) P(A_i)}{\sum_{i=1}^n P(B|A_i) P(A_i)} \quad (i = 1, \dots, n) \tag{3.204}$$

For example, students take one of  $n$  elective courses  $\{A_1, \dots, A_n\}$  each with probability  $P(A_i)$ . The probability to get a B grade in course  $A_i$  is known to be  $P(B|A_i)$ . Given that a student gets a B, the probability for him/her to have taken course  $A_i$  is  $P(A_i|B)$ .

The method of *Bayesian inference* deduces a hypothesis  $H$ , a model of the probability distribution of an observed dataset  $D$ , in terms of the form and/or parameters of the distribution. The method of Bayesian inference can be an iterative process, that updates the model parameters as more data becomes available. Based on Bayes' theorem, the Bayesian inference can be formulated in words:

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} \tag{3.205}$$

or in mathematics:

$$p(H|D) = \frac{p(D|H)p(H)}{p(D)} \propto p(D|H)p(H) \quad (3.206)$$

where  $p(H)$  is the *prior probability* of  $H$  based on some prior knowledge without observation of any data,  $p(H|D)$  is the *posterior probability* of  $H$  based on the observation of some data  $D$ ,  $p(D|H)$  is the *likelihood* of  $H$ , proportional to the probability of observing  $D$  given  $H$ , and  $p(D)$  is the probability of observing data  $D$ , which is independent of  $H$  in question, and typically ignored.

Specifically, the data  $D$  and hypothesis  $H$  take the following forms:

- Data  $D$ : a collection of data points in a  $d$ -dimensional *feature space*, represented in a data matrix  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  containing a set of  $N$  independent samples of a random vector  $\mathbf{x} = [x_1, \dots, x_d]^T$ , randomly and uniformly drawn from its probability density distribution (*independent and identically distributed, or i.i.d.*).
- Hypothesis  $H$ : a specific probabilistic model  $p(\mathbf{x}|\boldsymbol{\theta})$  for the dataset, parameterized by  $\theta_1, \dots, \theta_M$  represented as a vector  $\boldsymbol{\theta} = [\theta_1, \dots, \theta_M]^T$ , to be estimated based on the observed dataset  $\mathbf{X}$ .

Now Bayes inference can be formulated to estimate the model parameter  $\boldsymbol{\theta}$  given the observed dataset  $\mathbf{X}$ , i.e., Eq. (3.206) can be written as:

$$p(\boldsymbol{\theta}|\mathbf{X}) = \frac{p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{X})} = \frac{p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\sum_{\boldsymbol{\theta}} p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})} \propto p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \quad (3.207)$$

where

- $p(\boldsymbol{\theta})$ : the *a priori* probability, or the simply *prior*, of parameter  $\boldsymbol{\theta}$  based on some prior knowledge before any data  $\mathbf{X}$  are observed.
- $L(\boldsymbol{\theta}|\mathbf{X}) \propto p(\mathbf{X}|\boldsymbol{\theta})$ : the *likelihood* of the statistical model parameters in  $\boldsymbol{\theta}$  (hypothesis), given the observed data  $\mathbf{X}$ , representing how probable  $\mathbf{X}$  is for different values of  $\boldsymbol{\theta}$ . If the data samples in  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  are independently drawn from the same pdf, i.e., they are *independent and identically distributed (i.i.d.)*, then

$$p(\mathbf{X}|\boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\theta}) \quad (3.208)$$

- $p(\mathbf{X}) = \sum_{\boldsymbol{\theta}} p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ : the distribution of the data while the parameter  $\boldsymbol{\theta}$  is marginalized. As  $p(\mathbf{X})$  is a constant independent of the parameter  $\boldsymbol{\theta}$  in question, and it is needed only for normalizing  $p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ , it is often dropped in the process of maximization for convenience.
- $p(\boldsymbol{\theta}|\mathbf{X})$ : the *a posteriori* probability, or simply the *posterior* of the parameter  $\boldsymbol{\theta}$ , after data  $\mathbf{X}$  are observed.

There are two methods typically used for the estimation of the parameters in  $\boldsymbol{\theta}$  based on the observed data  $\mathbf{X}$ :

- *Maximum Likelihood Estimation (MLE)* seeks to find  $\boldsymbol{\theta}$  that maximizes the likelihood:

$$\boldsymbol{\theta}_{MLE} = \arg \max_{\boldsymbol{\theta}} p(\mathbf{X}|\boldsymbol{\theta}) \quad (3.209)$$

- *Maximum A Posteriori (MAP)* seeks to find  $\boldsymbol{\theta}$  that maximizes the posterior:

$$\boldsymbol{\theta}_{MAP} = \arg \max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathbf{X}) \propto p(\mathbf{X}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) \quad (3.210)$$

The difference between the two methods is whether certain prior knowledge is available regarding the distribution  $p(\boldsymbol{\theta})$  of the parameters before observing any data. If no such prior knowledge is available, i.e., all possible values of the parameters are equally likely with  $p(\boldsymbol{\theta}) = \text{constant}$  (uniform distribution), then the two methods will produce the same result. On the other hand, if we have some prior knowledge that the parameters obey certain distribution  $p(\boldsymbol{\theta})$ , e.g., they are normally distributed with certain mean and covariance, then the MAP method may produce more accurate estimation.

When we maximize either the likelihood or the posterior as a function of  $\boldsymbol{\theta}$ , we can equivalently maximize the logarithm (a monotonic function) of the function for mathematical convenience, the two methods above can therefore be expressed as

- MLE:

$$\begin{aligned} \boldsymbol{\theta}_{MLE} &= \arg \max_{\boldsymbol{\theta}} \log p(\mathbf{X}|\boldsymbol{\theta}) = \arg \max_{\boldsymbol{\theta}} \log \left( \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\theta}) \right) \\ &= \arg \max_{\boldsymbol{\theta}} \left( \sum_{n=1}^N \log p(\mathbf{x}_n|\boldsymbol{\theta}) \right) \end{aligned} \quad (3.211)$$

- MAP:

$$\begin{aligned} \boldsymbol{\theta}_{MAP} &= \arg \max_{\boldsymbol{\theta}} \log (p(\mathbf{X}|\boldsymbol{\theta}) p(\boldsymbol{\theta})) \\ &= \arg \max_{\boldsymbol{\theta}} \left( \sum_{n=1}^N \log p(\mathbf{x}_n|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \right) \end{aligned} \quad (3.212)$$

Both MLE and MAP seek to find some specific values for the parameters in  $\boldsymbol{\theta}$  that maximize certain function of the parameters, such as the likelihood or the posterior of  $\boldsymbol{\theta}$ , or their logarithm. In certain situations, we can instead find the values of such parameters in  $\boldsymbol{\theta}$  that maximizes the expectation of the function to be maximized, now denoted by  $Q(\boldsymbol{\theta})$ , by the method called *expectation-maximization (EM)*, in an iteration of two steps:

- The E-step: find the expectation  $Q(\boldsymbol{\theta})$  based on the previous parameter values  $\boldsymbol{\theta}_n$ ;
- The M-step: find the updated parameter values  $\boldsymbol{\theta}_{n+1}$  that maximize the expectation  $Q(\boldsymbol{\theta})$ .

### 3.3.2 Maximum Likelihood Estimation of Model Parameters

Given an observed dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  containing  $N$  independent and identically distributed (i.i.d.) sample data points, we can build a statistical model  $p(\mathbf{x}|\boldsymbol{\theta})$ , of which the parameter  $\boldsymbol{\theta}$  containing a set of model parameters is to be estimated based on the observed dataset  $\mathbf{X}$  by method such as *maximum likelihood estimation (MLE)*.

Specifically, we let  $\boldsymbol{\theta}'$  be the current estimation of the true model parameter  $\boldsymbol{\theta}$ , and find its likelihood and log-likelihood:

$$L(\boldsymbol{\theta}'|\mathbf{x}) \propto p(\mathbf{x}|\boldsymbol{\theta}'), \quad l(\boldsymbol{\theta}'|\mathbf{x}) = \log p(\mathbf{x}|\boldsymbol{\theta}') \quad (3.213)$$

We further consider the expectation of the log-likelihood with respect to the true distribution  $p(\mathbf{x}|\boldsymbol{\theta})$  is

$$\begin{aligned} E_{\boldsymbol{\theta}}[l(\boldsymbol{\theta}'|\mathbf{x})] &= \int p(\mathbf{x}|\boldsymbol{\theta})l(\boldsymbol{\theta}'|\mathbf{x})d\mathbf{x} = \int p(\mathbf{x}|\boldsymbol{\theta}) \log p(\mathbf{x}|\boldsymbol{\theta}')d\mathbf{x} \\ &= \int p(\mathbf{x}|\boldsymbol{\theta}) \log \left( p(\mathbf{x}|\boldsymbol{\theta}) \frac{p(\mathbf{x}|\boldsymbol{\theta}')}{p(\mathbf{x}|\boldsymbol{\theta})} \right) d\mathbf{x} \\ &= \int p(\mathbf{x}|\boldsymbol{\theta}) \log p(\mathbf{x}|\boldsymbol{\theta})d\mathbf{x} - \int p(\mathbf{x}|\boldsymbol{\theta}) \log \frac{p(\mathbf{x}|\boldsymbol{\theta}')}{p(\mathbf{x}|\boldsymbol{\theta})} d\mathbf{x} \\ &= -H(p(\mathbf{x}|\boldsymbol{\theta})) - D_{KL}(p(\mathbf{x}|\boldsymbol{\theta})||p(\mathbf{x}|\boldsymbol{\theta}')) \end{aligned} \quad (3.214)$$

where  $H(p(\mathbf{x}|\boldsymbol{\theta}))$  is the entropy of  $p(\mathbf{x}|\boldsymbol{\theta})$ , which is independent of the estimated parameter  $\boldsymbol{\theta}'$ , and  $D_{KL}(p(\mathbf{x}|\boldsymbol{\theta})||p(\mathbf{x}|\boldsymbol{\theta}'))$  is the KL-divergence between  $p(\mathbf{x}|\boldsymbol{\theta})$  and  $p(\mathbf{x}|\boldsymbol{\theta}')$ , which is zero when the estimated parameter becomes the same as the true parameter  $\boldsymbol{\theta}' = \boldsymbol{\theta}$ .

Now we see that the optimal model parameter that maximizes the likelihood is also the one that minimizes the KL-divergence:

$$\boldsymbol{\theta}_{MLE} = \arg \max_{\boldsymbol{\theta}'} L(\boldsymbol{\theta}'|\mathbf{x}) = \arg \min_{\boldsymbol{\theta}'} D_{KL}(p(\mathbf{x}|\boldsymbol{\theta})||p(\mathbf{x}|\boldsymbol{\theta}')) \quad (3.215)$$

Consider the simple example of the MLE method applied to estimate the two parameters in  $\boldsymbol{\theta} = \{\mathbf{m}, \boldsymbol{\Sigma}\}$  of the multivariate Gaussian distribution

$$\mathcal{N}(\mathbf{x}, \mathbf{m}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left[ -\frac{1}{2}(\mathbf{x} - \mathbf{m})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{m}) \right] \quad (3.216)$$

Given an observed dataset containing  $N$  i.i.d. data samples  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ , we can get the likelihood function

$$L(\boldsymbol{\theta}|\mathbf{X}) = L(\mathbf{m}, \boldsymbol{\Sigma}|\mathbf{X}) = p(\mathbf{X}|\mathbf{m}, \boldsymbol{\Sigma}) = \prod_{i=1}^N \mathcal{N}(\mathbf{x}_i|\mathbf{m}, \boldsymbol{\Sigma}) \quad (3.217)$$

and the log-likelihood function:

$$\begin{aligned} l(\mathbf{m}, \mathbf{\Sigma} | \mathbf{X}) &= \log L(\mathbf{m}, \mathbf{\Sigma} | \mathbf{X}) = \log \left[ \prod_{i=1}^N \mathcal{N}(\mathbf{x}_i | \mathbf{m}, \mathbf{\Sigma}) \right] = \sum_{i=1}^N \log \mathcal{N}(\mathbf{x}_i | \mathbf{m}, \mathbf{\Sigma}) \\ &= \sum_{i=1}^N \left[ -\frac{d}{2} \log 2\pi - \frac{1}{2} \log |\mathbf{\Sigma}| - \frac{1}{2} (\mathbf{x}_i - \mathbf{m})^T \mathbf{\Sigma}^{-1} (\mathbf{x}_i - \mathbf{m}) \right] \end{aligned} \quad (3.218)$$

To find the optimal parameters  $\mathbf{m}$  and  $\mathbf{\Sigma}$  that maximize this log-likelihood, we set its derivative with respect to  $\mathbf{m}$  and  $\mathbf{\Sigma}$  to zero and solve the resulting equations:

- Find  $\mathbf{m}$

$$\begin{aligned} \frac{\partial}{\partial \mathbf{m}} l(\mathbf{m}, \mathbf{\Sigma} | \mathbf{X}) &= -\frac{1}{2} \sum_{i=1}^N \frac{\partial}{\partial \mathbf{m}} ((\mathbf{x}_i - \mathbf{m})^T \mathbf{\Sigma}^{-1} (\mathbf{x}_i - \mathbf{m})) \\ &= -\frac{1}{2} \sum_{i=1}^N \mathbf{\Sigma}^{-1} (\mathbf{x}_i - \mathbf{m}) = \mathbf{0} \end{aligned} \quad (3.219)$$

Pre-multiplying  $-2\mathbf{\Sigma}$  on both sides, we get

$$\sum_{i=1}^N (\mathbf{x}_i - \mathbf{m}) = \mathbf{0} \quad (3.220)$$

solving which we get the optimal estimation of  $\mathbf{m}$ :

$$\hat{\mathbf{m}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \quad (3.221)$$

- Find  $\mathbf{\Sigma}$

$$\begin{aligned} \frac{\partial}{\partial \mathbf{\Sigma}} l(\mathbf{m}, \mathbf{\Sigma} | \mathbf{X}) &= -\frac{N}{2} \frac{\partial}{\partial \mathbf{\Sigma}} \log |\mathbf{\Sigma}| - \frac{1}{2} \sum_{i=1}^N \frac{\partial}{\partial \mathbf{\Sigma}} ((\mathbf{x}_i - \mathbf{m})^T \mathbf{\Sigma}^{-1} (\mathbf{x}_i - \mathbf{m})) \\ &= -\frac{N}{2} \mathbf{\Sigma}^{-1} + \frac{1}{2} \sum_{i=1}^N \mathbf{\Sigma}^{-1} (\mathbf{x}_i - \mathbf{m}) (\mathbf{x}_i - \mathbf{m})^T \mathbf{\Sigma}^{-1} = \mathbf{0} \end{aligned} \quad (3.222)$$

Pre and post multiplying  $\mathbf{\Sigma}$  on both sides and solving the resulting equation we get the optimal estimation of  $\mathbf{\Sigma}$ :

$$\hat{\mathbf{\Sigma}} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \hat{\mathbf{m}}) (\mathbf{x}_i - \hat{\mathbf{m}})^T \quad (3.223)$$

Here we have used the following facts:

$$\frac{d}{d\mathbf{A}} \log |\mathbf{A}| = (\mathbf{A}^{-1})^T, \quad \frac{d}{d\mathbf{A}} (\mathbf{a}^T \mathbf{A}^{-1} \mathbf{b}) = -(\mathbf{A}^{-1})^T \mathbf{a} \mathbf{b}^T (\mathbf{A}^{-1})^T \quad (3.224)$$

Note that the rank of  $\hat{\mathbf{\Sigma}}$  in Eq. (3.223) is  $N - 1$ , representing its degrees of

freedom due to the  $N$  samples, assumed to be independent, and the extra constraint in Eq. (3.221). When  $d > N - 1$ , the matrix  $\hat{\Sigma}$  does not have a full rank and is therefore non-invertible.

For more details regarding derivatives of a scalar function with respect to vector and matrix variables, see *Matrix Cookbook* here: [https://www.math.uwaterloo.ca/~h](https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf)

### 3.3.3 Natural gradient

The gradient method is commonly used in optimization to minimize or maximize an objective function  $J(\mathbf{x})$  in Euclidean space based on the following iteration:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}^* \quad (3.225)$$

where  $\Delta\mathbf{x}^*$  is the optimal increment among all those  $\Delta\mathbf{x} = \mathbf{x}_{n+1} - \mathbf{x}_n$  within a small neighborhood of the current  $\mathbf{x}_n$ :

$$\Delta\mathbf{x}^* = \arg \max_{\|\Delta\mathbf{x}\| < C} J(\mathbf{x} + \Delta\mathbf{x}) \quad (3.226)$$

The solution of this constrained optimization problem can be found by maximizing the following Lagrangian function

$$L(\Delta\mathbf{x}) = L(\mathbf{x} + \Delta\mathbf{x}) + \mu(\|\Delta\mathbf{x}\|^2 - C) \quad (3.227)$$

The optimal  $\Delta\mathbf{x}^*$  can be found by setting the derivative of  $L(\Delta\mathbf{x})$  to zero

$$\frac{d}{d\Delta\mathbf{x}} L(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{g}_L(\mathbf{x}) + \frac{\mu}{2}\Delta\mathbf{x} = 0 \quad (3.228)$$

and solving the resulting equation to get

$$\Delta\mathbf{x}^* = -\frac{2}{\mu}\mathbf{g}_J(\mathbf{x}) \quad (3.229)$$

Now the optimal  $\mathbf{x}^*$  can be found iteratively

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_n = \mathbf{x}_n - \lambda\mathbf{g}_J(\mathbf{x}_n) \quad (3.230)$$

where the step size  $\lambda = 2/\mu$  can be adjusted as needed.

However, in some cases the function to be optimized is not in the Euclidean space spanned by a set of orthonormal basis vector in which an inner product is defined and the Euclidean distance is used to measure the difference between two points in the space, but in a *Riemannian manifold* (a smooth manifold locally similar to a Euclidean space), in which a small neighboring area of a point can be approximated by its tangent space spanned by a set of vectors  $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$  not necessarily orthogonal to each other, and the squared length of an infinitesimal arc  $d\mathbf{x} = [dx_1, \dots, dx_d]^T$  on the manifold can be expressed in quadratic form as:

$$\begin{aligned} \|d\mathbf{x}\|^2 &= \left( \sum_{i=1}^n dx_i \mathbf{e}_i \right)^T \left( \sum_{j=1}^n dx_j \mathbf{e}_j \right) = \sum_{i=1}^n \sum_{j=1}^n dx_i dx_j \mathbf{e}_i^T \mathbf{e}_j \\ &= d\mathbf{x}^T \mathbf{G} d\mathbf{x} \end{aligned} \quad (3.231)$$

Here  $\mathbf{G}$ , called the *metric tensor*, is a positive-definite matrix based on the inner products of the basis vectors in the tangent space:

$$\mathbf{G} = \begin{bmatrix} \mathbf{e}_1^T \mathbf{e}_1 & \cdots & \mathbf{e}_1^T \mathbf{e}_n \\ \vdots & \ddots & \vdots \\ \mathbf{e}_n^T \mathbf{e}_1 & \cdots & \mathbf{e}_n^T \mathbf{e}_n \end{bmatrix} \quad (3.232)$$

In the special case when the Riemannian space is a Euclidean space spanned by orthonormal basis vectors satisfying  $\mathbf{e}_i^T \mathbf{e}_j = \delta_{ij}$ , we have  $\mathbf{G} = \mathbf{I}$ , and the squared arc length above becomes the conventional Euclidean squared norm based on inner product:  $\|\mathbf{dx}\|^2 = \mathbf{dx}^T \mathbf{dx} = \sum_i dx_i^2$ .

To minimize an objective function  $J(\mathbf{x})$  in a Riemannian manifold iteratively, we can still find the optimal increment step  $\Delta \mathbf{x}$  as given in Eq. (3.226), that maximizes the Lagrangian in Eq. (3.227), but now the squared norm is given in Eq. (3.231):

$$L(\Delta \mathbf{x}) = L(\mathbf{x} + \Delta \mathbf{x}) + \mu(\Delta \mathbf{x}^T \mathbf{G} \Delta \mathbf{x} - C) \quad (3.233)$$

Now we solve the equation

$$\begin{aligned} \frac{d}{d \Delta \mathbf{x}} L(\mathbf{x} + \Delta \mathbf{x}) &= \frac{d}{d \Delta \mathbf{x}} L(\mathbf{x} + \Delta \mathbf{x}) + \mu \frac{d}{d \Delta \mathbf{x}} (\Delta \mathbf{x}^T \mathbf{G} \Delta \mathbf{x} - C) \\ &= \mathbf{g}_L(\mathbf{x}) + \frac{\mu}{2} \mathbf{G} \Delta \mathbf{x} \end{aligned} \quad (3.234)$$

to get the optimal increment:

$$\Delta \mathbf{x}^* = -\frac{\mu}{2} \mathbf{G}^{-1} \mathbf{g}_L(\mathbf{x}) = -\frac{\mu}{2} \tilde{\mathbf{g}}_L(\mathbf{x}) \quad (3.235)$$

where  $\tilde{\mathbf{g}}_L(\mathbf{x}) = \mathbf{G}^{-1} \mathbf{g}_L(\mathbf{x})$  is the *natural gradient*, and the iteration becomes

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta \mathbf{x}_n = \mathbf{x}_n - \delta \tilde{\mathbf{g}}_L(\mathbf{x}) \quad (3.236)$$

### 3.3.4 Fisher Information and Natural Gradient

The *Fisher information* measures the amount of information carried by an observed random  $\mathbf{x}$  about the unknown parameter  $\boldsymbol{\theta}$  of a model distribution  $p(\mathbf{x}|\boldsymbol{\theta})$  of  $\mathbf{x}$ .

The likelihood function of the parameter  $\boldsymbol{\theta}$  for a model  $p(\mathbf{x}|\boldsymbol{\theta})$  given observed data  $\mathbf{x}$  is

$$L(\boldsymbol{\theta}|\mathbf{x}) \propto p(\mathbf{x}|\boldsymbol{\theta}) \quad (3.237)$$

and the log-likelihood is

$$l(\boldsymbol{\theta}|\mathbf{x}) = \log L(\boldsymbol{\theta}|\mathbf{x}) = \log p(\mathbf{x}|\boldsymbol{\theta}) \quad (3.238)$$

The gradient of the log-likelihood is

$$\mathbf{g}_l(\boldsymbol{\theta}) = \frac{d}{d \boldsymbol{\theta}} \log p(\mathbf{x}|\boldsymbol{\theta}) = \frac{d p(\mathbf{x}|\boldsymbol{\theta}) / d \boldsymbol{\theta}}{p(\mathbf{x}|\boldsymbol{\theta})} = \frac{\mathbf{g}_p(\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} \quad (3.239)$$

where  $\mathbf{g}_p(\boldsymbol{\theta}) = dp(\mathbf{x}|\boldsymbol{\theta})/d\boldsymbol{\theta}$  is the gradient of  $p(\mathbf{x}|\boldsymbol{\theta})$ . The expectation of  $\mathbf{g}_l$  is zero:

$$\begin{aligned} E_{\boldsymbol{\theta}}[\mathbf{g}_l(\boldsymbol{\theta})] &= \int \left( \frac{\mathbf{g}_p(\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} \right) p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x} = \int \mathbf{g}_p(\boldsymbol{\theta}) d\mathbf{x} \\ &= \int \frac{d}{d\boldsymbol{\theta}} p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x} = \frac{d}{d\boldsymbol{\theta}} \int p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x} = \frac{d}{d\boldsymbol{\theta}} 1 = \mathbf{0} \end{aligned} \quad (3.240)$$

and the covariance of  $\mathbf{g}_l$  is defined as the *Fisher information*:

$$\mathbf{F}(\boldsymbol{\theta}) = E_{\boldsymbol{\theta}}[\mathbf{g}_l(\boldsymbol{\theta})\mathbf{g}_l^T(\boldsymbol{\theta})] \quad (3.241)$$

which can be estimated based on an observed dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ :

$$\mathbf{F}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \mathbf{g}_l(\boldsymbol{\theta})\mathbf{g}_l^T(\boldsymbol{\theta}) \quad (3.242)$$

The Hessian of the log-likelihood  $l(\boldsymbol{\theta}|\mathbf{x})$  is:

$$\begin{aligned} \mathbf{H}_l(\boldsymbol{\theta}) &= \frac{d^2}{d\boldsymbol{\theta}^2} l(\boldsymbol{\theta}|\mathbf{x}) = \frac{d}{d\boldsymbol{\theta}} \mathbf{g}_l(\boldsymbol{\theta}) = \frac{d}{d\boldsymbol{\theta}} \left[ \frac{\mathbf{g}_p(\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} \right] \\ &= \frac{\mathbf{H}_p(\boldsymbol{\theta})p(\mathbf{x}|\boldsymbol{\theta}) - \mathbf{g}_p(\boldsymbol{\theta})\mathbf{g}_p^T(\boldsymbol{\theta})}{p^2(\mathbf{x}|\boldsymbol{\theta})} = \frac{\mathbf{H}_p(\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} - \mathbf{g}_l(\boldsymbol{\theta})\mathbf{g}_l^T(\boldsymbol{\theta}) \end{aligned} \quad (3.243)$$

The expectation of  $\mathbf{H}_l$  is

$$E_{\boldsymbol{\theta}}[\mathbf{H}_l(\boldsymbol{\theta})] = E_{\boldsymbol{\theta}} \left[ \frac{\mathbf{H}_p(\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} \right] - E_{\boldsymbol{\theta}} [\mathbf{g}_l(\boldsymbol{\theta})\mathbf{g}_l^T(\boldsymbol{\theta})] \quad (3.244)$$

The first term is

$$\begin{aligned} E_{\boldsymbol{\theta}} \left[ \frac{\mathbf{H}_p(\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} \right] &= \int \left[ \frac{\mathbf{H}_p(\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} \right] p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x} = \int \mathbf{H}_p(\boldsymbol{\theta}) d\mathbf{x} = \int \frac{d^2}{d\boldsymbol{\theta}^2} p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x} \\ &= \frac{d^2}{d\boldsymbol{\theta}^2} \int p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x} = \frac{d^2}{d\boldsymbol{\theta}^2} 1 = \mathbf{0} \end{aligned} \quad (3.245)$$

and the second term is the Fisher information defined in Eq. (3.241), and we have:

$$\mathbf{F}(\boldsymbol{\theta}) = -E_{\boldsymbol{\theta}}[\mathbf{H}_l(\boldsymbol{\theta})] = E_{\boldsymbol{\theta}} [\mathbf{g}_l(\boldsymbol{\theta})\mathbf{g}_l^T(\boldsymbol{\theta})] \quad (3.246)$$

of which the component in the  $i$ th row and  $j$ th column is expected product of the  $i$ th and  $j$ th components of  $\mathbf{g}_l(\boldsymbol{\theta})$ :

$$F_{ij} = E[g_i g_j] = E \left[ \frac{\partial \log p(\mathbf{x}|\boldsymbol{\theta})}{\partial \theta_i} \frac{\partial \log p(\mathbf{x}|\boldsymbol{\theta})}{\partial \theta_j} \right] \quad (3.247)$$

We note that  $g_i = \partial \log p(\mathbf{x}|\boldsymbol{\theta})/\partial \theta_i$  represents how much variation in the pdf  $p(\mathbf{x}|\boldsymbol{\theta})$  of  $\mathbf{x}$  will be caused by parameter  $\theta_i$ , or how much information about  $\theta_i$  is carried by  $\mathbf{x}$ . If in the extreme case when  $g_i = 0$ , then  $\mathbf{x}$  carries no information about  $\theta_i$  as it is irrelevant to  $\theta_i$ . Therefore the Fisher information containing all  $F_{ij} = E[g_i g_j]$  represents the amount of information about  $\boldsymbol{\theta}$  carried by  $\mathbf{x}$ .

While building a statistical model  $p(\mathbf{x}|\boldsymbol{\theta})$  of the observed dataset, we need to



approximate the unknown parameter  $\boldsymbol{\theta}$  by the estimated  $\boldsymbol{\theta}'$  based on the samples of variable  $\mathbf{x}$ , so that the KL-divergence between  $p(\mathbf{x}|\boldsymbol{\theta}')$  and the true  $p(\mathbf{x}|\boldsymbol{\theta})$  (Eq. (3.191)) is minimized:

$$D_{KL}(p(\mathbf{x}|\boldsymbol{\theta})||p(\mathbf{x}|\boldsymbol{\theta}')) = E_{\boldsymbol{\theta}}[\log p(\mathbf{x}|\boldsymbol{\theta}) - \log p(\mathbf{x}|\boldsymbol{\theta}')] \quad (3.248)$$

Consider its first and second order derivatives with respect to  $\boldsymbol{\theta}'$ :

- The gradient:

$$\begin{aligned} \mathbf{g}_{KL}(\boldsymbol{\theta}') &= E_{\boldsymbol{\theta}} \left[ \frac{d}{d\boldsymbol{\theta}'} [\log p(\mathbf{x}|\boldsymbol{\theta}) - \log p(\mathbf{x}|\boldsymbol{\theta}')] \right] \\ &= -E_{\boldsymbol{\theta}} \left[ \frac{d}{d\boldsymbol{\theta}'} \log p(\mathbf{x}|\boldsymbol{\theta}') \right] = -E_{\boldsymbol{\theta}}[\mathbf{g}_l(\boldsymbol{\theta}')] \end{aligned} \quad (3.249)$$

- The Hessian:

$$\mathbf{H}_{KL}(\boldsymbol{\theta}') = -E_{\boldsymbol{\theta}} \left[ \frac{d}{d\boldsymbol{\theta}'} \mathbf{g}_l(\boldsymbol{\theta}') \right] = -E_{\boldsymbol{\theta}}[\mathbf{H}_l(\boldsymbol{\theta}')] \quad (3.250)$$

Evaluating both  $\mathbf{g}_{KL}(\boldsymbol{\theta}')$  and  $\mathbf{H}_{KL}(\boldsymbol{\theta}')$  at  $\boldsymbol{\theta}' = \boldsymbol{\theta}$ , we get (Eqs. (3.240) and (3.246)):

$$\mathbf{g}_{KL}(\boldsymbol{\theta}) = -E_{\boldsymbol{\theta}}[\mathbf{g}_l(\boldsymbol{\theta})] = \mathbf{0}, \quad \text{and} \quad \mathbf{H}_{KL}(\boldsymbol{\theta}) = \mathbf{F}(\boldsymbol{\theta}) \quad (3.251)$$

Now the KL-divergence above can be approximated by the first three terms of its Taylor expansion in the neighborhood of  $\boldsymbol{\theta}$  in terms of  $\Delta\boldsymbol{\theta} = \boldsymbol{\theta}' - \boldsymbol{\theta}$ :

$$\begin{aligned} D_{KL}(p(\mathbf{x}|\boldsymbol{\theta})||p(\mathbf{x}|\boldsymbol{\theta}')) &\approx D_{KL}(p(\mathbf{x}|\boldsymbol{\theta})||p(\mathbf{x}|\boldsymbol{\theta})) + \mathbf{g}_{KL}^T(\boldsymbol{\theta})\Delta\boldsymbol{\theta} \\ &\quad + \frac{1}{2}\Delta\boldsymbol{\theta}^T \mathbf{H}_{KL}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} = \frac{1}{2}\Delta\boldsymbol{\theta}^T \mathbf{F}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} \end{aligned} \quad (3.252)$$

Comparing this with Eq. (3.231), we see that we can treat each component  $g_i = \partial \log p(\mathbf{x}|\boldsymbol{\theta})/\partial \theta_j$  of  $\mathbf{g}_l(\boldsymbol{\theta})$  as a basis vector in the Riemannian space for all distributions  $p(\mathbf{x}|\boldsymbol{\theta})$ , then the KL-divergence above becomes the squared distance between the two distributions:

$$D_{KL}(p(\mathbf{x}|\boldsymbol{\theta})||p(\mathbf{x}|\boldsymbol{\theta}')) \approx \frac{1}{2}\Delta\boldsymbol{\theta}^T \mathbf{F}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} = \frac{1}{2} \sum_i \sum_j \Delta\theta_i \Delta\theta_j g_i g_j \quad (3.253)$$

Now the optimal model parameter  $\boldsymbol{\theta}$  that maximizes the likelihood or equivalently minimizes the KL-divergence (Eq. (3.215)) can be found iteratively by the natural gradient method (Eq. (3.236)):

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \Delta\boldsymbol{\theta}_n = \boldsymbol{\theta}_n + \delta \tilde{\mathbf{g}}_L(\boldsymbol{\theta}_n) \quad (3.254)$$

### 3.4 Bias-Variance Tradeoff

The *mean squared error* as defined below is a common way to measure the error of the estimation of some parameter, or, more generally, the performance of a supervised learning algorithm (for either regression or classification). Let  $\hat{\boldsymbol{\theta}}$  be an

estimator of the parameter  $\theta$  of a data model. Then *mean squared error (MSE)* of this estimator is

$$\begin{aligned}
 \text{MSE}(\hat{\theta}) &= E[(\hat{\theta} - \theta)^2] \\
 &= E[(\hat{\theta} - E(\hat{\theta}) + E(\hat{\theta}) - \theta)^2] \\
 &= E[(\hat{\theta} - E(\hat{\theta}))^2] + 2E[(\hat{\theta} - E(\hat{\theta}))(E(\hat{\theta}) - \theta)] + E[(E(\hat{\theta}) - \theta)^2] \\
 &= E[(\hat{\theta} - E(\hat{\theta}))^2] + E[(E(\hat{\theta}) - \theta)^2] \\
 &= \text{Var}(\hat{\theta}) + \text{Bias}(\hat{\theta})
 \end{aligned} \tag{3.255}$$

Note that the middle term disappears as

$$E[\hat{\theta} - E(\hat{\theta})] = E(\hat{\theta}) - E(\hat{\theta}) = 0 \tag{3.256}$$

The equation above indicates that the MSE is composed of two parts:

- *variance error*  $\text{Var}(\hat{\theta}) = E[(\hat{\theta} - E(\hat{\theta}))^2]$ :  
caused by the oversensitivity of a learning algorithm to small fluctuations in the training dataset due to random noise rather than the intended property of the data, a problem called *overfitting*.
- *bias error*  $\text{Bias}(\hat{\theta}) = E[(E(\hat{\theta}) - \theta)^2]$ :  
caused by the under-sensitivity of a learning algorithm to the dominating variation in the training dataset representing the essential relationship in the training dataset, a problem called *underfitting*.

In general, a proper bias-variance tradeoff needs to be made for an algorithm to avoid either overfitting or underfitting, so that it can learn the essential relationship in the data while not affected by the inevitable noise in the data.

### 3.5 Principal Component Analysis (PCA)

Previously we discussed converting a random vector  $\mathbf{x}$  of zero mean  $\mathbf{m}_x = \mathbf{0}$  and identity covariance  $\Sigma_x = \mathbf{I}$  to  $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{m}$  with desired  $\mathbf{m}$  and  $\Sigma$  in Eq. (3.123) by the affine transformation. Here we consider the opposite operation to convert any random  $\mathbf{x}$  with  $\mathbf{m}_x = \mathbf{0}$  and  $\Sigma_x$  to  $\mathbf{y}$  with  $\Sigma_y = \mathbf{I}$ , so that the components in  $\mathbf{y}$  are completely decorrelated with zero covariance  $\sigma_{ij}^2 = \delta_{ij}$ . This general process, called *whitening*, can be easily realized also by the Cholesky decomposition of  $\Sigma_x$ :

$$\Sigma_x^{-1} = \mathbf{W}^T \mathbf{W}, \quad \text{i.e.} \quad \Sigma_x = (\mathbf{W}^T \mathbf{W})^{-1} \tag{3.257}$$

Then the resulting vector by the whitening transform  $\mathbf{y} = \mathbf{W}\mathbf{x}$  will have the same zero mean

$$\mathbf{m}_y = E[\mathbf{y}] = E[\mathbf{W}\mathbf{x}] = \mathbf{W}E[\mathbf{x}] = \mathbf{W}\mathbf{m}_x = \mathbf{0} \tag{3.258}$$

but a whitened covariance:

$$\begin{aligned}\boldsymbol{\Sigma}_y &= E[\mathbf{y}\mathbf{y}^T] = E[(\mathbf{W}\mathbf{x})(\mathbf{W}\mathbf{x})^T] = \mathbf{W}E[\mathbf{x}\mathbf{x}^T]\mathbf{W}^T = \mathbf{W}\boldsymbol{\Sigma}_x\mathbf{W}^T \\ &= \mathbf{W}(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T = \mathbf{W}\mathbf{W}^{-1}(\mathbf{W}^T)^{-1}\mathbf{W}^T = \mathbf{I}\end{aligned}\quad (3.259)$$

There exist infinite number of matrices  $\mathbf{W}$  all satisfying Eq. (3.257), such as those found by the Cholesky decomposition and eigenvalue decomposition of the covariance matrix  $\boldsymbol{\Sigma}_x$ , as discussed before. Here we revisit the method based on eigenvalue decomposition and show it is optimal in some sense.

consider the eigenequations of the covariance matrix  $\boldsymbol{\Sigma}$  of an N-D random vector  $\mathbf{x} = [x_1, \dots, x_N]^T$ , now treated as  $N$  samples of a time signal:

$$\boldsymbol{\Sigma}_x\phi_n = \lambda_n\phi_n \quad (n = 1, \dots, N) \quad (3.260)$$

As  $\boldsymbol{\Sigma}$  is symmetric and positive semi-definite, its eigenvalues are real and non-negative, and the corresponding eigenvectors are orthogonal to each other:

$$\lambda_N \geq \dots \geq \lambda_1 \geq 0, \quad \phi_i^T\phi_j = \delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \quad (i, j = 1, \dots, N) \quad (3.261)$$

i.e.,  $\boldsymbol{\Phi}^T = \boldsymbol{\Phi}^{-1}$  is an orthogonal matrix. The eigenequations above can be expressed in matrix form:

$$\boldsymbol{\Sigma}\boldsymbol{\Phi} = \boldsymbol{\Phi}\boldsymbol{\Lambda}, \quad \text{or} \quad \boldsymbol{\Phi}^{-1}\boldsymbol{\Sigma}\boldsymbol{\Phi} = \boldsymbol{\Phi}^T\boldsymbol{\Sigma}\boldsymbol{\Phi} = \boldsymbol{\Lambda} \quad (3.262)$$

where

$$\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_N), \quad \boldsymbol{\Phi} = [\phi_1, \dots, \phi_N] \quad (3.263)$$

are the eigenvalue and eigenvector matrix.

Further consider the following orthogonal transform of  $\mathbf{x}$ , called *Karhunen-Loeve transform (KLT)*, *Hotelling Transform* or *eigenvector Transform* based on the eigenvector matrix  $\boldsymbol{\Phi}$ :

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \boldsymbol{\Phi}^T\mathbf{x} = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_N^T \end{bmatrix} \mathbf{x} \quad (3.264)$$

where the  $n$ th component of  $\mathbf{y}$  is  $y_n = \phi_n^T\mathbf{x}$  is the projection of  $\mathbf{x}$  onto the  $n$ th eigenvector  $\phi_n$ . Left multiplying both sides by  $\boldsymbol{\Phi}$ , we get the inverse KLT:

$$\mathbf{x} = \boldsymbol{\Phi}\mathbf{y} = [\phi_1, \dots, \phi_N] \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \sum_{n=1}^N y_n\phi_n \quad (3.265)$$

We see that  $\mathbf{x}$  is expressed as a linear combination of  $\phi_1, \dots, \phi_N$  treated as an orthogonal basis that spans the N-D space. The KLT has two most favorable properties:

1. It completely decorrelate the signal

2. It maximally compact the signal energy

To see the first property, we consider covariance matrix of  $\mathbf{y}$ :

$$\Sigma_{\mathbf{y}} = \Phi^T \Sigma_{\mathbf{x}} \Phi = \Phi^{-1} \Sigma_{\mathbf{x}} \Phi = \Lambda \quad (3.266)$$

i.e.,

$$\begin{bmatrix} \sigma_{11}^2 & \cdots & \sigma_{1N}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{N1}^2 & \cdots & \sigma_{NN}^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{bmatrix} \quad (3.267)$$

The covariance of  $y_i$  and  $y_j$  of the random vector  $\mathbf{y} = \Phi^T \mathbf{x}$  is

$$\sigma_{ij}^2 = \begin{cases} \lambda_i & i = j \\ 0 & i \neq j \end{cases} \quad (3.268)$$

As  $\sigma_{ij}^2 = 0$  for any  $i \neq j$ , the components of  $\mathbf{y}$  are completely decorrelizd. after the KLT.

We next show that the KLT redistributes the dynamic energy contained in the signal so that it is maximally compacted into any  $M < N$  components, while conserving the total energy. Let  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_N]$  be an arbitrary orthogonal matrix, based on which an orthogonal transform of  $\mathbf{x}$  can be defined as

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \mathbf{A}^T \mathbf{x} = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_N^T \end{bmatrix} \mathbf{x} \quad (3.269)$$

where  $y_i = \mathbf{a}_i^T \mathbf{x}$ . The inverse transform is:

$$\mathbf{x} = \mathbf{A} \mathbf{y} = [\mathbf{a}_1, \dots, \mathbf{a}_N] \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \sum_{i=1}^N y_i \mathbf{a}_i \quad (3.270)$$

As shown before,  $\Sigma_{\mathbf{y}} = \mathbf{A}^T \Sigma_{\mathbf{x}} \mathbf{A}$  and

$$\text{tr} \Sigma_{\mathbf{y}} = \sum_{i=1}^N \sigma_{y_i}^2 = \sum_{i=1}^N \sigma_{x_i}^2 = \text{tr} \Sigma_{\mathbf{x}} \quad (3.271)$$

i.e., the total dynamic energy (or information) contained in the signal  $\mathbf{x}$  is conserved by the orthogonal transform. However, the energy distribution among all  $N$  components can be very different before and after the KLT transform. To see this, define the energy contained in the first  $M < N$  components after the transform  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$ , treated as an function of the transform matrix  $\mathbf{A}$ :

$$\varepsilon_M(\mathbf{A}) = \sum_{i=1}^M \sigma_{y_i}^2 = \sum_{i=1}^M \mathbf{a}_i^T \Sigma_{\mathbf{x}} \mathbf{a}_i \quad (3.272)$$

The optimal transform matrix  $\mathbf{A}$  that maximizes  $\varepsilon_M(\mathbf{A})$  can be found by solving the following constrained maximization problem:

$$\begin{cases} \text{maximize: } \varepsilon_M(\mathbf{A}) = \sum_{i=1}^M \mathbf{a}_i^T \boldsymbol{\Sigma}_x \mathbf{a}_i \\ \text{subject to: } \mathbf{a}_j^T \mathbf{a}_j = 1 \quad (j = 1, \dots, N) \end{cases} \quad (3.273)$$

The constraint  $\mathbf{a}_j^T \mathbf{a}_j = 1$  is to guarantee that the column vectors in  $\mathbf{A}$  are normalized. Based on the Lagrange multiplier method, we let:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{a}_i} \left[ S_M(\mathbf{A}) - \sum_{j=1}^M \lambda_j (\mathbf{a}_j^T \mathbf{a}_j - 1) \right] &= \frac{\partial}{\partial \mathbf{a}_i} \left[ \sum_{j=1}^M (\mathbf{a}_j^T \boldsymbol{\Sigma}_x \mathbf{a}_j - \lambda_j \mathbf{a}_j^T \mathbf{a}_j + \lambda_j) \right] \\ &= \frac{\partial}{\partial \mathbf{a}_i} (\mathbf{a}_i^T \boldsymbol{\Sigma}_x \mathbf{a}_i - \lambda_i \mathbf{a}_i^T \mathbf{a}_i) \stackrel{*}{=} 2 (\boldsymbol{\Sigma}_x \mathbf{a}_i - \lambda_i \mathbf{a}_i) = 0, \quad (i = 1, \dots, N) \end{aligned} \quad (3.274)$$

For this to hold,  $\mathbf{a}_i$  must be an eigen-vector  $\phi_i$  of  $\boldsymbol{\Sigma}_x$ , and we have thus proved that the optimal transform is indeed KLT. To maximize  $\varepsilon_M(\mathbf{A})$ , we must have  $\mathbf{A} = \Phi$ :

$$\varepsilon_M(\mathbf{A}) = \varepsilon_M(\Phi) = \sum_{i=1}^M \phi_i^T \boldsymbol{\Sigma}_x \phi_i = \sum_{i=1}^M \lambda_i \quad (3.275)$$

To maximize this,  $\phi_i$ 's need to be the eigenvectors of  $\boldsymbol{\Sigma}_x$  corresponding to  $M$  greatest eigenvalues of the  $N$ :  $\lambda_1 \geq \dots \geq \lambda_M \geq \dots \geq \lambda_N$ .

Due to KLT's properties of signal decorrelation and energy compaction, it is widely used for data compression by reducing the dimensionality of the data set.

## 3.6 Student's t-test

Let  $x$  be a random variable with a normal distribution  $\mathcal{N}(\mu, \sigma^2)$ , and  $\{x_1, \dots, x_N\}$  be a set of  $N$  independent and identically distributed (i.i.d.) samples drawn from this distribution. Then the true mean of  $x$  can be estimated by its *sample mean*:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \sim \mathcal{N}(\mu, \sigma^2/N) \quad (3.276)$$

i.e.,  $\bar{x}$  is a normally distributed random variable with the same mean  $\mu$  as that of the original distribution and variance  $\sigma^2/N$ , which is the same as that of the original distribution if  $N = 1$ , but approaches zero when  $N \rightarrow \infty$ . We can further define another random variable  $z$  as below with a standard normal distribution (of zero mean and unit variance):

$$z = \frac{\bar{x} - \mu}{\sigma/\sqrt{N}} \sim \mathcal{N}(0, 1) \quad (3.277)$$

If the standard deviation  $\sigma$  is not known, we can replace it by the estimated *sample standard deviation* based on the estimated *sample variance*:

$$S^2 = \frac{1}{N-1} \sum_{i=1}^N e_i^2 = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2 = \frac{1}{N-1} \left[ \sum_{i=1}^N x_i - N\bar{x} \right] \quad (3.278)$$

Now random variable  $Z$  can be replaced by another random variable  $T$  based on  $S$  instead of  $\sigma$ :

$$t = \frac{\bar{x} - \mu}{S/\sqrt{N}} \sim T_\nu(t) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})} \left(1 + \frac{t^2}{\nu}\right)^{-(\nu+1)/2} \quad (3.279)$$

where  $\Gamma(n) = (n-1)!$  is the *gamma function*. This is a random variable with Student's t-distribution of  $\nu = N-1$  degrees of freedom. Specially

$$T_1(t) = \frac{1}{\pi(1+t^2)}, \quad T_2(t) = \frac{1}{(2+t^2)^{3/2}}, \quad T_3(t) = \frac{6\sqrt{3}}{\pi(3+t^2)^2}, \quad T_\infty(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2} \quad (3.280)$$

When  $\nu \rightarrow \infty$ , the t-distribution becomes the normal distribution (the blue curve in the plots).

The *confidence interval* of the mean  $\mu$  is defined as the interval between two limiting values such that the probability for  $\mu$  to fall inside the range is  $(1-\alpha)100\%$ :

$$P(L \leq \mu \leq U) = (1-\alpha)100\% \quad (3.281)$$

where  $\alpha$  is the *significant level*. For example, if  $\alpha = 0.05$ , then  $P(L \leq \mu \leq U) = 95\%$ .

To find the limits of such a confidence interval corresponding to a given  $\alpha$ , we first consider the interval  $[-z_{\alpha/2}, z_{\alpha/2}]$  so that

$$P(-z_{\alpha/2} \leq z \leq z_{\alpha/2}) = \int_{z_{-\alpha/2}}^{z_{\alpha/2}} \mathcal{N}(z, 0, 1) dz = 1 - \alpha, \quad (3.282)$$

i.e.,

$$\int_{-\infty}^{z_{-\alpha/2}} \mathcal{N}(z, 0, 1) dz = \int_{z_{\alpha/2}}^{\infty} \mathcal{N}(z, 0, 1) dz = \alpha/2 \quad (3.283)$$

For example, if  $\alpha = 0.05$ , we can find  $z_{\alpha/2} = 1.96$  from the *z-Table* (<https://math.arizona.edu/~r>

Now we can further find the lower and upper limits of the confidence interval for  $\mu$  by noting that the following events are equivalent:

$$\begin{aligned} P(-z_{\alpha/2} \leq z \leq z_{\alpha/2}) &= P\left(-z_{\alpha/2} \leq \frac{\bar{x} - \mu}{\sigma/\sqrt{N}} \leq z_{\alpha/2}\right) \\ &= P\left(\bar{x} - z_{\alpha/2} \frac{\sigma}{\sqrt{N}} \leq \mu \leq \bar{x} + z_{\alpha/2} \frac{\sigma}{\sqrt{N}}\right) = 1 - \alpha \end{aligned} \quad (3.284)$$

i.e., the lower and upper limits of the confidence interval for  $\mu$  corresponding to  $1-\alpha$  is:

$$L = \bar{x} - z_{\alpha/2} \frac{\sigma}{\sqrt{N}}, \quad U = \bar{x} + z_{\alpha/2} \frac{\sigma}{\sqrt{N}} \quad (3.285)$$

so that

$$P(L \leq \mu \leq U) = 1 - \alpha \quad (3.286)$$

The *standard error* of the estimated mean is defined as  $\sigma/\sqrt{N}$ , and the upper and lower limits of the confidence interval given above can be expressed as:

$$\bar{x} \pm z_{\alpha/2} \frac{\sigma}{\sqrt{N}} = \bar{x} \pm z_{\alpha/2} SE \quad (3.287)$$

When  $\sigma$  is not known, it can be approximated by the *estimated standard error*  $ESE = S/\sqrt{N}$  based on the estimated sample standard deviation  $S$ . We can follow the same derivation above, but replacing  $z$  with standard normal distribution by  $t$  with t-distribution, and find the limits of the confidence interval as:

$$\bar{x} \pm t_{\alpha/2} \frac{S}{\sqrt{N}} = \bar{x} \pm t_{\alpha/2} ESE \quad (3.288)$$

where  $t_{\alpha/2}$  are the limits satisfying the following:

$$P(-t_{\alpha/2} \leq t \leq t_{\alpha/2}) = \int_{-t_{\alpha/2}}^{t_{\alpha/2}} T_{\nu}(t) dt = 1 - \alpha \quad (3.289)$$

which can be found by the *t-Table* (<http://www.sjsu.edu/faculty/gerstman/StatPrimer/t-table.pdf>), given a significant level  $\alpha$  and the degrees of freedom  $\nu$ .

For example, assume  $\bar{x}$  is obtained based on  $N = 11$  samples:

$$\bar{x} = \frac{1}{11} \sum_{i=1}^{11} x_i \quad (3.290)$$

then the degrees of freedom is  $\nu = N - 1 = 10$ . For a significant level  $\alpha = 0.05$ , we can find  $t_{0.05/2} = 2.228$  from the t-Table. Note that this value is different from  $t_{0.05/2} = 1.96$ . However, when the sample size  $N$  becomes larger,  $T_{\nu}(t) = T_{N-1}(t)$  will approach to the standard normal distribution  $N(z, 0, 1)$ , and  $t_{\alpha/2} \rightarrow z_{\alpha/2}$ .

Confidence interval for linear regression....

## 3.7 Hypergeometric Distribution

If  $n$  samples in a population of  $N$  samples have certain property of interest while the rest do not, the probability of the event that  $k$  samples in a subset of  $K$  samples of the population have the property is

$$P(k) = \frac{\binom{n}{k} \binom{N-n}{K-k}}{\binom{N}{K}}, \quad (k = 0, \dots, K) \quad (3.291)$$

where  $\binom{n}{k} = n!/(k!(n-k)!)$  is the number of ways to choose  $k$  samples out of  $n$  samples. We have

$$\sum_{k=0}^K P(k) = 1 \quad (3.292)$$

We are interested in finding out whether the ratio of the samples with the property out of all samples in the subset is significantly different from the same ratio of the total population, and if so, whether it is significantly higher or lower. Here the term *significance* can be intuitively understood as the occurrence of a highly unlikely event with very low probability (e.g., 5% = 0.05).

We first define the *p-value* as

$$p(k) = \sum_{i=k}^K P(i) = 1 - \sum_{i=0}^{k-1} P(i) \quad (3.293)$$

based on which

- $p < 0.05$ : the ratio of the subset is significantly higher than that of the total population
- $p > 0.95$ : the ratio of the subset is significantly lower than that of the total population
- $0.05 < p < 0.95$ : the ratio of the subset is not significantly different from that of the total population

**Example:** In a class of  $N = 100$  students of which  $K = 45$  are female,  $n = 15$  students get grade A, of which  $k$  are female. Determine whether the number of female students who get an A is significantly lower or higher than the total population of the class based on the p-value  $p(k)$  of the following values:

- $p(6) = 0.9625 < 0.95$ , significantly lower
- $p(7) = 0.8963 < 0.95$ , not significantly different
- $p(12) = 0.1047 > 0.05$ , not significantly different
- $p(13) = 0.0394 < 0.05$ , significantly higher



## Part II

---

# Regression



# 4 Linear Regression

---

## 4.1 Regression as Optimization

The goal of regression analysis is to model the relationship between a dependent variable  $y$  and a set of independent variables (predictors) as components of a vector  $\mathbf{x} = [x_1, \dots, x_d]^T$ , a point in the  $d$ -dimensional space, given a set of observed sample points in the training set  $\mathcal{D} = \{(\mathbf{x}_n, y_n), (n = 1, \dots, N)\}$ , in which each sample  $\mathbf{x}_n$  of the independent variables is labeled by the value of the corresponding dependent variable  $y_n$ . Alternatively, the training set can also be denoted by an  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$  in terms of a  $d \times N$  matrix  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  and an  $N$ -D vector  $\mathbf{y} = [y_1, \dots, y_N]^T$ .

Specifically, the model is a *regression function*  $\hat{y} = f(\mathbf{x}, \boldsymbol{\theta})$ , which can be either deterministic or probabilistic depending on the specific algorithm. Here  $\boldsymbol{\theta}$  is a vector containing a set of parameters as its components. Typically the regression function is hypothesized to take certain form (e.g., linear, polynomial, or exponential) based on some prior knowledge, while the parameters in  $\boldsymbol{\theta}$  are to be determined by the regression algorithm based on the training set  $\mathcal{D}$ , so that the predicted value  $\hat{y}$  by the model function matches the ground truth  $y$  optimally in certain sense, as discussed below.

Geometrically, the regression function  $\hat{y} = f(\mathbf{x}, \boldsymbol{\theta})$  represents a hypersurface when  $d > 2$  as generally assumed, otherwise a surface if  $d = 2$ , or a curve if  $d = 1$ ). The goal to match the model to the training set is to fit the hypersurface to all training samples as points in the  $d + 1$  dimensional space (spanned by the dependent variable  $y$  as well as the  $d$  independent variables as components in  $\mathbf{x}$ ) optimally in certain sense.

Regression analysis can also be interpreted as *system modeling/identification*, when the independent variable  $\mathbf{x}$  and the dependent variable  $y$  are treated respectively as the input (stimuli) and output (responses) of a system, and the behavior of the system in terms of the relationship between such input and output is modeled by the regression function  $y = f(\mathbf{x})$ .

While the regression algorithm tries to fit the regression function to the dataset, it should not be affected by the inevitable observation noise in the data but still catches the essential characteristics present in the data. In other words, the regression analysis should neither *overfit* (overly affected by noise) nor *underfit* (not matching valid data sufficiently closely) the observed data. This issue of

making a proper tradeoff between overfitting and underfitting is of great importance in all supervised learning methods in general, to be addressed repeatedly in the future.

Regression analysis can be considered as a process of *supervised learning* during which the regression algorithm finds the unknown model parameters in  $\theta$  by learning from the training set  $\mathcal{D}$ . It is similar to a class of supervised learning methods in *pattern recognition/classification*, also formulated in terms of the independent variables  $\mathbf{x}$  treated as a set of *features* that characterize some *patterns* or *objects* of interest, and the corresponding dependent variable  $y$  treated as the categorical labeling indicating to which of a set of  $K$  classes  $\{C_1, \dots, C_K\}$  a pattern  $\mathbf{x}$  belongs. In this case, the modeling process for the relationship between  $\mathbf{x}$  and  $y$  becomes pattern classification or recognition, to be discussed in Chapter ??.

Due to this similarity between the two different types of supervised learning methods for regression and classification, all following discussion in this section for regression algorithms also applies in general to the classification algorithms to be considered in the future chapters.

The problem of regression analysis can be addressed based on different philosophical viewpoints. From the *frequentist* point of view, the unknown model parameters in  $\theta$  are fixed deterministic variables and their values are to be estimated based on the training set. A typical method based on this viewpoint is the *least squares (LS)* method for both linear and nonlinear regressions.

Alternatively, from the Bayesian inference point of view (Section ??), the model parameters in  $\theta$  are random variables. Their *prior probability distribution*  $p(\theta)$  before any data are observed can be estimated based on some prior knowledge. If no such prior knowledge is available,  $p(\mathbf{w})$  can be simply a uniform distribution, i.e., all possible values of  $\mathbf{w}$  are equally likely. Once the data samples in the training set  $\mathbf{X}$  are available and observed, the *posterior probability* of the model parameters can be estimated based on Bayes' theorem:

$$p(\theta|\mathbf{X}) = \frac{p(\mathbf{X}|\theta)p(\theta)}{p(\mathbf{X})} \propto p(\mathbf{X}|\mathbf{w})p(\theta) \quad (4.1)$$

The posterior  $p(\theta|\mathbf{X})$  is in general a narrower distribution than the prior  $p(\theta)$  due to the additional information gained from the observed data, and it is therefore a more accurate description of the model parameters. The output  $\hat{y} = f(\mathbf{x}, \theta)$  of such a regression model based Bayesian inference is no longer a deterministic value, but a random variable described by its probability distribution in terms of its mean and variance. Typical methods based on this point of view include Bayesian linear regression, logistic regression and Gaussian process regression.

Based on either of these two viewpoints, different methods for regression can be used to find the parameters  $\theta$  for the model function  $\hat{y} = f(\mathbf{x}, \theta)$ , such as those listed below.

- *Least squares (LS) method:*

This frequentist method measures how well the regression function models the training data by the *residual* defined as  $r(\boldsymbol{\theta}) = y - \hat{y}(\boldsymbol{\theta})$ , defined as the difference between the model prediction  $\hat{y}(\boldsymbol{\theta}) = f(\mathbf{x}, \boldsymbol{\theta})$  and the ground truth labeling value  $y$  corresponding to  $\mathbf{x}$ , for each of the  $N$  data points in the observed data:

$$\mathbf{r}(\boldsymbol{\theta}) = \begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix} = \begin{bmatrix} y_1 - \hat{y}_1 \\ \vdots \\ y_N - \hat{y}_N \end{bmatrix} = \begin{bmatrix} y_1 - f(\mathbf{x}_1, \boldsymbol{\theta}) \\ \vdots \\ y_N - f(\mathbf{x}_N, \boldsymbol{\theta}) \end{bmatrix} = \mathbf{y} - \hat{\mathbf{y}} \quad (4.2)$$

Ideally, for the residual to be zero,  $\mathbf{r}(\boldsymbol{\theta}) = \mathbf{0}$ , we find the  $M$  model parameters as components of  $\boldsymbol{\theta}$  by solving this nonlinear equation system of  $N$  equations of  $M$  unknowns in  $\boldsymbol{\theta}$ . However, as in general  $N \gg M$ , this system is overconstrained without a solution. We therefore can only use least squares method to find the optimal solution that minimizes the *sum of squared error (SSE)*:

$$\varepsilon(\boldsymbol{\theta}) = \frac{1}{2} \|\mathbf{r}\|^2 = \frac{1}{2} \mathbf{r}^T \mathbf{r} = \frac{1}{2} \sum_{n=1}^N r_n^2 = \frac{1}{2} \sum_{n=1}^N (y_n - f(\mathbf{x}_n, \boldsymbol{\theta}))^2 \quad (4.3)$$

Here the coefficient  $1/2$  is included for mathematical convenience. When divided by the total number of samples  $N$ , this SSE becomes the average of all  $N$  squared errors, and the error becomes the *mean squared error (MSE)*. Minimizing either SSE or MSE produces the same results.

- *Maximum likelihood estimate (MLE)*:

This Bayesian inference method measures how well the regression function models the training data in terms of the likelihood  $L(\boldsymbol{\theta}|\mathcal{D})$  of the model parameter  $\boldsymbol{\theta}$  based on the observed dataset  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ , which is proportional to the conditional probability of  $\mathcal{D}$  given  $\boldsymbol{\theta}$ :

$$L(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta}) = p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \quad (4.4)$$

The optimal model parameters in  $\boldsymbol{\theta}$  can be found as those that maximize  $L(\boldsymbol{\theta}|\mathcal{D})$  or equivalently  $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$ .

We assume the residual  $r = y - f(\mathbf{x}, \boldsymbol{\theta})$  is a zero-mean random variable (without loss of generality) with a normal probability density function (pdf):

$$\begin{aligned} p(r) &= p(y - f(\mathbf{x}, \boldsymbol{\theta})) = \mathcal{N}(r, 0, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - f(\mathbf{x}, \boldsymbol{\theta}))^2}{2\sigma^2}\right) \\ &= \mathcal{N}(y, f(\mathbf{x}, \boldsymbol{\theta}), \sigma^2) = p(y|\mathbf{x}, \boldsymbol{\theta}) \end{aligned} \quad (4.5)$$

The justification for this assumed normal pdf is that it has the greatest entropy (Section 3.2.1), or maximum uncertainty among all possible pdfs with the same variance, i.e., given  $\sigma^2$ , the assumed normal pdf imposes least amount of constraint and thereby minimum bias.

The zero-mean pdf of  $r = y - f(\mathbf{x}, \boldsymbol{\theta})$  can also be considered as the pdf

of  $y$  with mean  $f(\mathbf{x}, \boldsymbol{\theta})$ , the conditional pdf  $p(y|\mathbf{x}, \boldsymbol{\theta})$  of  $y$  given  $\boldsymbol{\theta}$  as well as  $\mathbf{x}$ , based on the assumption that  $y$  and  $\mathbf{x}$  are related by  $y = f(\mathbf{x}, \boldsymbol{\theta})$ . Then we can find the likelihood of  $\boldsymbol{\theta}$  given the data samples in the training set, all assumed to be *independent and identically distributed (i.i.d.)*:

$$\begin{aligned} L(\boldsymbol{\theta}|\mathcal{D}) &= p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{n=1}^N p(y = y_n|\mathbf{x}_n, \boldsymbol{\theta}) \\ &= \prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - f(\mathbf{x}_n, \boldsymbol{\theta}))^2}{2\sigma^2}\right) \end{aligned} \quad (4.6)$$

or, equivalently and for mathematical convenience, minimize the negative log likelihood function:

$$-l(\boldsymbol{\theta}|\mathcal{D}) = -\log L(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) = -\sum_{n=1}^N \log p(y = y_n|\mathbf{x}_n, \boldsymbol{\theta}) \quad (4.7)$$

$$\begin{aligned} &= -\sum_{n=1}^N \log \left[ \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - f(\mathbf{x}_n, \boldsymbol{\theta}))^2}{2\sigma^2}\right) \right] \\ &= \frac{N}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sum_{n=1}^N \|\mathbf{r}\|^2 \end{aligned} \quad (4.8)$$

The first term and the coefficient  $1/\sigma^2$  in the second term are both independent of  $\boldsymbol{\theta}$  and can therefore be dropped. We note that this result is equivalent to that in Eq. (4.3), i.e., we can find the optimal  $\boldsymbol{\theta}$  by either maximizing the likelihood  $L(\boldsymbol{\theta}|\mathcal{D})$ , or equivalently minimizing the SSE  $\varepsilon(\boldsymbol{\theta})$ .

We further note that negative log likelihood  $-l(\boldsymbol{\theta}|\mathcal{D}) = -\log L(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y})$  in the equation above can also be interpreted differently as the cross entropy (Section 3.186) between an unknown ground truth probability  $P$  and the model probability  $Q$ :

$$H(P, Q) = -\sum_i P_i \log Q_i = -E_P[\log Q] \quad (4.9)$$

We rewrite Eq. (4.7) as the average of the logarithm of the model distribution  $Q = p(y|\mathbf{x}, \boldsymbol{\theta})$  over all  $N$  observed samples in the dataset  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ :

$$-l(\boldsymbol{\theta}|\mathcal{D}) = -\sum_{n=1}^N \log p(y = y_n|\mathbf{x}_n, \boldsymbol{\theta}) = -E_{(X, y)}[\log p(y = y_n|\mathbf{x}_n, \boldsymbol{\theta})] \quad (4.10)$$

and realize that this is the cross entropy  $H(P, Q)$  estimated based on the  $N$  samples, which can be minimized for model  $Q$  to match the ground truth  $P$  as closely as possible.

- *Maximum A Posteriori (MAP)*

This is also a Bayesian inference method that measures how well the regression function models the training data by the posterior probability of the parameter  $\boldsymbol{\theta}$  given in Eq. (4.1), proportional to the product of the

likelihood  $L(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})$  and the prior  $p(\boldsymbol{\theta})$ . If no prior knowledge about  $\boldsymbol{\theta}$  is available, then  $p(\boldsymbol{\theta})$  is a uniform distribution, then MAP is equivalent to MLE. However, if certain prior knowledge regarding  $\boldsymbol{\theta}$  does exist, and the prior is not uniform, then the posterior better characterizes  $\boldsymbol{\theta}$  than the likelihood and MAP may produce better result than MLE.

We conclude that regression analysis, and in general many supervised learning algorithms to be considered in the following chapters, can be treated as an over-constrained optimization problem with many more observed data points in the training set than the number of unknown parameters in the model. Following different philosophical view points, this optimization problem can be addressed differently, such as minimizing the sum of squared errors, maximizing the likelihood of the model parameters, or minimizing the negative likelihood, i.e., the cross entropy between the model and unknown ground truth distributions of the given dataset, to be discussed in details in the following sections.

More generally, some of the concepts and methods in regression analysis discussed in the following sections are also shared by many supervised learning algorithms for classification. For example, the general method of neural network is conceptually based on some ideas very different from regression, but both of these two methods can be described mathematically as a function  $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta})$  that models the relationship between the independent variable  $\mathbf{x}$  and dependent variable  $\mathbf{y}$ . While this function is called the regress function in this chapter, it is also a way to describe the complex behavior of a multilayer deep learning network to be discussed in Chapter ??

## 4.2 Linear Least Squares Regression

In linear regression, the relationship between the dependent variable  $y$  and  $d$  independent variables in  $\mathbf{x} = [x_1, \dots, x_d]^T$  is modeled by a linear hypothesis function:

$$\hat{y} = f(\mathbf{x}, \mathbf{w}) = b + \sum_{i=1}^d w_i x_i = \sum_{i=0}^d w_i x_i = \mathbf{x}^T \mathbf{w} \quad (4.11)$$

where  $\mathbf{x} = [x_0 = 1, x_1, \dots, x_d]^T$  is redefined as an augmented  $d + 1$  dimensional vector containing  $x_0 = 1$  as well as the  $d$  independent variables, and  $\mathbf{w} = [w_0 = b, w_1, \dots, w_d]^T$  is an augmented  $d + 1$  dimensional vector containing  $w_0 = b$  as well as the  $d$  weights as the parameters, to be determined based on the given dataset.

Geometrically, this linear regression function  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  is a straight line if  $d = 1$ , a plane if  $d = 2$ , or a hyperplane if  $d > 2$ , in the  $d + 1$  dimensional space spanned by  $y$  as well as  $\{x_1, \dots, x_d\}$ . The function has an intercept  $f(\mathbf{0}) = w_0 = b$  along the  $y$  dimension and a normal direction  $[w_1, \dots, w_d, -1]^T$ . When thresholded by the plane  $y = 0$ , the regression function becomes an equation

$f(\mathbf{x}) = \mathbf{x}^T \mathbf{w} = 0$ , representing a point if  $d = 1$ , a straight line if  $d = 2$ , a plane or hyperplane if  $d \geq 3$ , in the  $d$ -dimensional space spanned by  $\{x_1, \dots, x_d\}$ , as shown in the figure below for  $d = 2$ .

We denote by  $\mathbf{X}$  the matrix containing the  $N$  augmented data points  $\mathbf{x}_n = [x_{0n} = 1, x_{1n}, \dots, x_{dn}]^T$  ( $n = 1, \dots, N$ ) as its column vectors

$$\mathbf{X} = \begin{bmatrix} 1 & \cdots & 1 \\ x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{d1} & \cdots & x_{dN} \end{bmatrix}_{(d+1) \times N} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \quad (4.12)$$

and its transpose can be written as:

$$\mathbf{X}^T = \begin{bmatrix} 1 & x_{11} & \cdots & x_{d1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1N} & \cdots & x_{dN} \end{bmatrix}_{N \times (d+1)} = [\mathbf{1}, \underline{\mathbf{x}}_1, \dots, \underline{\mathbf{x}}_d] \quad (4.13)$$

where  $\mathbf{1} = [1, \dots, 1]^T$ , and  $\underline{\mathbf{x}}_i = [x_{i1}, \dots, x_{iN}]^T$  ( $i = 1, \dots, d$ ) is an  $N$ -dimensional vectors containing the  $i$ th components of all  $N$  data vectors in the dataset.

Now the linear regression problem is simply to find the model parameter, here the weight vector  $\mathbf{w}$ , so that the model prediction

$$\hat{\mathbf{y}} = \mathbf{X}^T \mathbf{w} \quad (4.14)$$

matches the ground truth labeling  $\mathbf{y}$ . In other words, the residual of the linear model defined as  $\mathbf{r} = \mathbf{y} - \mathbf{X}^T \mathbf{w}$  need to be zero:

$$\mathbf{r} = \mathbf{y} - \mathbf{X}^T \mathbf{w} = \mathbf{0} \quad (4.15)$$

This is a linear system containing  $N$  equations of  $M$  unknowns.

If the number of data samples is equal to the number of unknown parameters, i.e.,  $N = d + 1$ , then  $\mathbf{X}$  is a square and invertible matrix (assuming all  $N$  samples are independent and therefore  $\mathbf{X}$  has a full rank), and the equation above can be solved to get the desired weight vector:

$$\mathbf{w} = (\mathbf{X}^T)^{-1} \mathbf{y} \quad (4.16)$$

However, as typically  $N \gg d + 1$ , i.e., there are many more data samples than the unknown parameters,  $\mathbf{X}$  is non-invertible and  $\mathbf{X}^T \mathbf{w} = \mathbf{y}$  is over-determined linear system without a solution, i.e., the prediction  $\hat{\mathbf{y}} = \mathbf{X}^T \mathbf{w}$  can never match the ground truth labeling  $\mathbf{y}$ . We need to find an optimal solution  $\mathbf{w}^*$  that minimizes the SSE (proportional to MSE):

$$\begin{aligned} \varepsilon(\mathbf{w}) &= \frac{1}{2} \sum_{n=1}^B r_n^2 = \frac{1}{2} \|\mathbf{r}\|^2 = \frac{1}{2} \|\mathbf{y} - \mathbf{X}^T \mathbf{w}\|^2 \\ &= \frac{1}{2} \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}^T \mathbf{w} + \frac{1}{2} \mathbf{w}^T \mathbf{X} \mathbf{X}^T \mathbf{w} \end{aligned} \quad (4.17)$$



To find the optimal  $\mathbf{w}$  that minimizes  $\varepsilon(\mathbf{w})$ , we set the gradient of  $\varepsilon$  to zero and get the *normal equation*:

$$\mathbf{g}_\varepsilon(\mathbf{w}) = \frac{d}{d\mathbf{w}}\varepsilon(\mathbf{w}) = \mathbf{X}\mathbf{X}^T\mathbf{w} - \mathbf{X}\mathbf{y} = \mathbf{0}, \quad \text{i.e.} \quad \mathbf{X}\mathbf{X}^T\mathbf{w} = \mathbf{X}\mathbf{y} \quad (4.18)$$

Solving for  $\mathbf{w}$ , we get the optimal weight vector:

$$\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y} = (\mathbf{X}^T)^{-}\mathbf{y} \quad (4.19)$$

where  $(\mathbf{X}^T)^- = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}$  is the  $(d+1) \times N$  pseudo-inverse (Section A.15) of the  $N \times (d+1)$  matrix  $\mathbf{X}^T$ . This method for estimating the regression model parameters is called *ordinary least squares (OLS)*.

Having found the parameter  $\mathbf{w}$  for the linear regression model  $\hat{y} = \mathbf{x}^T\mathbf{w}$ , we can predict the outputs corresponding to any unlabeled test dataset  $\mathbf{X}_* = [\mathbf{x}_{1*}, \dots, \mathbf{x}_{M*}]$ :

$$\hat{\mathbf{y}}_* = \mathbf{f}_* = \mathbf{X}_*^T\mathbf{w}^* = [\mathbf{1}, \underline{\mathbf{x}}_1, \dots, \underline{\mathbf{x}}_N] \begin{bmatrix} w_0^* \\ w_1^* \\ \vdots \\ w_d^* \end{bmatrix} = w_0^*\mathbf{1} + \sum_{i=1}^d w_i^*\underline{\mathbf{x}}_i \quad (4.20)$$

which is a linear combination of  $\mathbf{1}$  and  $\{\underline{\mathbf{x}}_1, \dots, \underline{\mathbf{x}}_d\}$

We further consider the evaluation of the linear regression result in terms of how well it fits the training data. We first rewrite the normal equation in Eq. (4.18) as:

$$\begin{aligned} \mathbf{X}\mathbf{X}^T\mathbf{w} - \mathbf{X}\mathbf{y} &= \mathbf{X}(\mathbf{X}^T\mathbf{w} - \mathbf{y}) = \mathbf{X}(\hat{\mathbf{y}} - \mathbf{y}) \\ &= \mathbf{X}\mathbf{r} = \begin{bmatrix} 1 & \cdots & 1 \\ x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{d1} & \cdots & x_{dN} \end{bmatrix} \mathbf{r} = \begin{bmatrix} \mathbf{1}^T \\ \underline{\mathbf{x}}_1^T \\ \vdots \\ \underline{\mathbf{x}}_d^T \end{bmatrix} \mathbf{r} = \mathbf{0} \end{aligned} \quad (4.21)$$

The last equality indicate that the residual vector  $\mathbf{r}$  is perpendicular to each of the  $d+1$  vectors  $\{\mathbf{1}, \underline{\mathbf{x}}_1, \dots, \underline{\mathbf{x}}_d\}$ :

$$\mathbf{1}^T\mathbf{r} = \sum_{n=1}^N r_n = 0, \quad \underline{\mathbf{x}}_1^T\mathbf{r} = \dots = \underline{\mathbf{x}}_d^T\mathbf{r} = 0 \quad (4.22)$$

We see that the residual  $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}^T\mathbf{w}^*$  based on the optimal  $\mathbf{w}^*$  represents the shortest distance between  $\mathbf{y}$  and any point in the space spanned by  $\{\mathbf{1}, \underline{\mathbf{x}}_1, \dots, \underline{\mathbf{x}}_d\}$ , i.e., any other estimate  $\hat{\mathbf{y}}' = \mathbf{X}^T\mathbf{w}$  also in the space but based on a non-optimal  $\mathbf{w} \neq \mathbf{w}^*$  must have a greater residual  $\mathbf{r}' = \mathbf{y} - \hat{\mathbf{y}}' > \mathbf{r}$ , i.e.,  $\mathbf{w}^*$  is indeed the optimal parameter based on which the sum of squared error  $\|\mathbf{r}\|^2$  of the model  $\hat{\mathbf{y}} = \mathbf{X}^T\mathbf{w}^*$  is minimized.

We further consider some quantitative measurement for evaluating how well the linear regression model  $y = \mathbf{w}^T\mathbf{x}$  fits the given dataset. To do so, we

first derive some properties of the model based on its output  $\hat{\mathbf{y}} = \mathbf{X}^T \mathbf{w} = \mathbf{X}^T (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{y}$ :

$$\begin{aligned} \|\hat{\mathbf{y}}\|^2 &= \hat{\mathbf{y}}^T \hat{\mathbf{y}} = \left( \mathbf{X}^T (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{y} \right)^T \left( \mathbf{X}^T (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{y} \right) \\ &= \left( \mathbf{y}^T \mathbf{X}^T (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X} \right) \left( \mathbf{X}^T (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{y} \right) \\ &= \mathbf{y}^T \mathbf{X}^T (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{y} = \mathbf{y}^T \hat{\mathbf{y}} \end{aligned} \quad (4.23)$$

based on which we also get

$$\mathbf{r}^T \hat{\mathbf{y}} = (\mathbf{y} - \hat{\mathbf{y}})^T \hat{\mathbf{y}} = \mathbf{y}^T \hat{\mathbf{y}} - \hat{\mathbf{y}}^T \hat{\mathbf{y}} = 0 \quad (4.24)$$

$$\begin{aligned} \|\mathbf{r}\|^2 &= \mathbf{r}^T \mathbf{r} = (\mathbf{y} - \hat{\mathbf{y}})^T (\mathbf{y} - \hat{\mathbf{y}}) = \mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T \hat{\mathbf{y}} + \hat{\mathbf{y}}^T \hat{\mathbf{y}} \\ &= \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \hat{\mathbf{y}} = \mathbf{y}^T (\mathbf{y} - \hat{\mathbf{y}}) = \mathbf{y}^T \mathbf{r} \end{aligned} \quad (4.25)$$

and

$$\|\hat{\mathbf{y}}\|^2 + \|\mathbf{r}\|^2 = \mathbf{y}^T \hat{\mathbf{y}} + \mathbf{y}^T \mathbf{r} = \mathbf{y}^T (\hat{\mathbf{y}} + \mathbf{r}) = \mathbf{y}^T \mathbf{y} = \|\mathbf{y}\|^2 \quad (4.26)$$

We further find the mean of  $\{y_1, \dots, y_N\}$

$$\bar{y} = \frac{1}{N} \sum_{n=1}^N y_n = \frac{1}{N} \mathbf{y}^T \mathbf{1} \quad (4.27)$$

and define a vector  $\bar{\mathbf{y}} = \bar{y} \mathbf{1}$ . The three vectors  $\mathbf{y} - \hat{\mathbf{y}}$ ,  $\mathbf{y} - \bar{\mathbf{y}}$  and  $\hat{\mathbf{y}} - \bar{\mathbf{y}}$  are all perpendicular to vector  $\mathbf{1}$ :

$$\begin{aligned} (\mathbf{y} - \bar{\mathbf{y}})^T \mathbf{1} &= \sum_{n=1}^N (y_n - \bar{y}) = N(\bar{y} - \bar{y}) = 0 \\ (\mathbf{y} - \hat{\mathbf{y}})^T \mathbf{1} &= \sum_{n=1}^N (y_n - \hat{y}_n) = \sum_{n=1}^N r_n = 0 \quad Eq.(4.22) \\ (\hat{\mathbf{y}} - \bar{\mathbf{y}})^T \mathbf{1} &= (\hat{\mathbf{y}} - \mathbf{y} + \mathbf{y} - \bar{\mathbf{y}})^T \mathbf{1} = (\hat{\mathbf{y}} - \mathbf{y})^T \mathbf{1} + (\mathbf{y} - \bar{\mathbf{y}})^T \mathbf{1} = 0 \end{aligned} \quad (4.28)$$

Based on these three vectors, we further define the following *sums of squares*:

- *The total sum of squares (TSS)* for the total variation in data:

$$TSS = \|\mathbf{y} - \bar{\mathbf{y}}\|^2 = \sum_{n=1}^N (y_n - \bar{y})^2 \quad (4.29)$$

- *The explained sum of squares (ESS)* for the variation of data explained by the regression model:

$$ESS = \|\hat{\mathbf{y}} - \bar{\mathbf{y}}\|^2 = \sum_{n=1}^N (\hat{y}_n - \bar{y})^2 \quad (4.30)$$

- *Residual sum of squares (RSS)* for the variation of data not explained by the data (due to noise or discrepancy between the data and the model):

$$RSS = \|\mathbf{r}\|^2 = \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \sum_{n=1}^N (y_n - \hat{y}_n)^2 \quad (4.31)$$

We can show that the total sum of squares is the sum of the explained sum of squares and the residual sum of squares:

$$\begin{aligned} TSS &= \|\mathbf{y} - \bar{\mathbf{y}}\|^2 = \|\mathbf{y} - \bar{\mathbf{y}} + \hat{\mathbf{y}} - \hat{\mathbf{y}}\|^2 = \|(\hat{\mathbf{y}} - \bar{\mathbf{y}}) + (\mathbf{y} - \hat{\mathbf{y}})\|^2 \\ &= \|\hat{\mathbf{y}} - \bar{\mathbf{y}}\|^2 + 2(\mathbf{y} - \hat{\mathbf{y}})^T(\hat{\mathbf{y}} - \bar{\mathbf{y}}) + \|\mathbf{y} - \hat{\mathbf{y}}\|^2 \\ &= \|\hat{\mathbf{y}} - \bar{\mathbf{y}}\|^2 + 2\mathbf{r}^T(\hat{\mathbf{y}} - \bar{\mathbf{y}}) + \|\mathbf{r}\|^2 \\ &= ESS + 2\mathbf{r}^T\hat{\mathbf{y}} - 2\mathbf{r}^T\bar{\mathbf{y}} + RSS = ESS + RSS \end{aligned} \quad (4.32)$$

The last equality is due to the fact that the two middle terms are both zero:

$$\begin{cases} \mathbf{r}^T\bar{\mathbf{y}} = \sum_{n=1}^N r_n\bar{y} = \bar{y}\sum_{n=1}^N r_n = 0 & \text{Eq.(4.22),} \\ \mathbf{r}^T\hat{\mathbf{y}} = (\mathbf{y} - \hat{\mathbf{y}})^T\hat{\mathbf{y}} = \mathbf{y}^T\hat{\mathbf{y}} - \hat{\mathbf{y}}^T\hat{\mathbf{y}} = 0 & \text{Eq.(4.24)} \end{cases} \quad (4.33)$$

We can now measure the goodness of the model by the *coefficient of determination*, denoted by  $R^2$  (*R-squared*), defined as the percentage of variance explained by the model:

$$R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS} \quad (4.34)$$

Given the total sum of squares TSS, if the model residual RSS is small, then ESS is large, indicating most of the variation in the data can be explained by the model, and  $R^2$  is large, i.e., the model is a good fit of the data.

In the special case of 1-dimensional dataset  $\{(x_n, y_n) \ (n = 1, \dots, N)\}$ , how closely the two variables  $x$  and  $y$  are correlated to each other can be measured by *correlation coefficient* defined as:

$$\rho = \frac{\sigma_{xy}^2}{\sqrt{\sigma_x^2 \sigma_y^2}} = \frac{\sigma_{xy}}{\sigma_x \sigma_y} \quad (4.35)$$

where

$$\bar{x} = \frac{1}{N} \sum_{n=1}^N x_n, \quad \bar{y} = \frac{1}{N} \sum_{n=1}^N y_n$$

and

$$\sigma_x^2 = \frac{1}{N} \sum_{n=1}^N x_n^2 - \bar{x}^2, \quad \sigma_y^2 = \frac{1}{N} \sum_{n=1}^N y_n^2 - \bar{y}^2, \quad \sigma_{xy}^2 = \frac{1}{N} \sum_{n=1}^N x_n y_n - \bar{x}\bar{y}$$

We can further find the linear regression model  $y = w_0 + w_1 x$  in terms of the

model parameters:

$$\begin{aligned}
\mathbf{w} &= \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y} \\
&= \left( \begin{bmatrix} 1 & \cdots & 1 \\ x_1 & \cdots & x_N \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & \cdots & 1 \\ x_1 & \cdots & x_N \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \\
&= \begin{bmatrix} N & \sum_{n=1}^N x_n \\ \sum_{n=1}^N x_n & \sum_{n=1}^N x_n^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{n=1}^N y_n \\ \sum_{n=1}^N x_n y_n \end{bmatrix} = \begin{bmatrix} 1 & \bar{x} \\ \bar{x} & \sigma_x^2 + \bar{x}^2 \end{bmatrix}^{-1} \begin{bmatrix} \bar{y} \\ \sigma_{xy}^2 + \bar{x}\bar{y} \end{bmatrix} \\
&= \frac{1}{\sigma_x^2} \begin{bmatrix} \sigma_x^2 + \bar{x}^2 & -\bar{x} \\ -\bar{x} & 1 \end{bmatrix} \begin{bmatrix} \bar{y} \\ \sigma_{xy}^2 + \bar{x}\bar{y} \end{bmatrix} = \begin{bmatrix} \bar{y} - \sigma_{xy}^2/\sigma_x^2 & \bar{x} \\ \sigma_{xy}^2/\sigma_x^2 & \end{bmatrix} \quad (4.36)
\end{aligned}$$

Based on these model parameters  $w_1 = \sigma_{xy}^2/\sigma_x^2$  and  $w_0 = \bar{y} - \bar{x}w_1$ , we get the linear regression model:

$$\hat{y} = w_0 + w_1 x = \bar{y} - \frac{\sigma_{xy}^2}{\sigma_x^2} \bar{x} + \frac{\sigma_{xy}^2}{\sigma_x^2} x \quad \text{or} \quad \frac{\sigma_{xy}^2}{\sigma_x^2} = \frac{y - \bar{y}}{x - \bar{x}} \quad (4.37)$$

The coefficient of determination  $R^2$  for the goodness of the regression is closely related to the correlation coefficient  $\rho$  for correlation of the given data (but independent of the model), as one would intuitively expect. Consider

$$\begin{aligned}
\frac{1}{N}RSS &= \frac{1}{N} \|y - \hat{y}\|^2 = \frac{1}{N} \sum_{n=1}^N (y_n - w_0 - w_1 x_n)^2 \\
&= \frac{1}{N} \sum_{n=1}^N (y_n - (\bar{y} - \bar{x}w_1) - w_1 x_n)^2 = \frac{1}{N} \sum_{n=1}^N [(y_n - \bar{y}) - w_1(x_n - \bar{x})]^2 \\
&= \frac{1}{N} \sum_{n=1}^N [(y_n - \bar{y})^2 - 2w_1(y_n - \bar{y})(x_n - \bar{x}) + w_1^2(x_n - \bar{x})^2] \\
&= \sigma_y^2 - 2w_1\sigma_{xy}^2 + w_1^2\sigma_x^2 = \sigma_y^2 - 2\frac{\sigma_{xy}^2}{\sigma_x^2}\sigma_{xy}^2 + \frac{\sigma_{xy}^4}{\sigma_x^4}\sigma_x^2 = \sigma_y^2 - \frac{\sigma_{xy}^4}{\sigma_x^2} \\
&= \sigma_y^2 \left( 1 - \frac{\sigma_{xy}^4}{\sigma_x^2\sigma_y^2} \right) = \sigma_y^2(1 - \rho^2) \quad (4.38)
\end{aligned}$$

We see that if the two variables  $x$  and  $y$  are highly correlated and  $\rho$  is close to 1, then the error of the model is small, i.e.,  $RSS = \|\mathbf{r}\|^2$  is small and  $R^2$  is large, indicating the LS linear model fits the data well.

#### Example 1:

Find a linear regression function  $y = w_0 + w_1 x$  to fit the following  $N = 6$  points:

$x$	-3.4	-2.1	-0.8	0.3	1.7	2.5
$y$	-0.76	-1.04	1.75	1.82	3.17	3.15

Based on the data, we get

$$\bar{y} = 1.348, \quad \sigma_x^2 = 4.22, \quad \sigma_y^2 = 2.85, \quad \sigma_{xy}^2 = 3.27, \quad \rho = \frac{\sigma_{xy}^2}{\sigma_x \sigma_y} = 0.94$$

The augmented data array is:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -3.4 & -2.1 & -0.8 & 0.3 & 1.7 & 2.5 \end{bmatrix}$$

Solving the over-constrained equation system

$$\mathbf{y} = \begin{bmatrix} -0.76 \\ -1.04 \\ 1.75 \\ 1.82 \\ 3.17 \\ 3.15 \end{bmatrix} = \mathbf{X}^T \mathbf{w} = \begin{bmatrix} 1 & -3.4 \\ 1 & -2.1 \\ 1 & -0.8 \\ 1 & 0.3 \\ 1 & 1.7 \\ 1 & 2.5 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

we get

$$\mathbf{w} = \mathbf{X}^- \mathbf{y} = (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X} \mathbf{y} = \begin{bmatrix} 0.13 & 0.15 & 0.16 & 0.17 & 0.19 & 0.20 \\ -0.12 & -0.07 & -0.02 & 0.02 & 0.08 & 0.11 \end{bmatrix} \begin{bmatrix} -0.76 \\ -1.04 \\ 1.75 \\ 1.82 \\ 3.17 \\ 3.15 \end{bmatrix} = \begin{bmatrix} 1.58 \\ 0.78 \end{bmatrix}$$

and the linear regression model function  $y = w_0 + w_1 x = 1.58 + 0.78x$ , a straight line with intercept  $w_0 = 1.58$ , slope  $w_1 = 0.78$ , and normal direction  $[0.78, -1]^T$ . The LS error is  $\|\mathbf{r}\| = \|\mathbf{y} - \hat{\mathbf{y}}\| = 1.38$ . We can further get  $\bar{\mathbf{y}} = [1.348, 1.348, 1.348, 1.348, 1.348, 1.348]^T$ ,  $\hat{\mathbf{y}} = \mathbf{X}^T \mathbf{w} = [-1.054, -0.047, 0.961, 1.813, 2.898, 3.518]^T$ , and

$$ESS = 15.21, \quad RSS = 1.91, \quad TSS = 17.12, \quad R^2 = 0.89$$

We can also check that

$$\frac{1}{N} RSS = \sigma_y^2 (1 - \rho^2) = 0.318$$

### Example 2:

$x$	-3.0	-2.5	-2.0	-1.5	-1.0	-0.5	0.0	0.5	1.0	1.5	2.0	2.5	3.0
$y_1$	-2.67	-3.76	-0.64	-0.93	-0.01	-1.42	3.38	-0.47	1.77	3.67	6.40	4.49	8.82
$y_2$	2.73	-9.38	-1.44	-9.86	-2.82	-7.29	8.31	0.98	10.18	8.58	10.86	0.77	0.20

	$w_0$	$w_1$	$ESS$	$RSS$	$TSS$	$R^2$	$\rho$
dataset1	1.43	1.72	134.71	30.18	164.89	0.817	0.904
dataset2	0.91	1.92	166.98	443.92	610.89	0.273	0.523

**Example 3:**

The figure below shows a set of  $N = 30$  data points  $\{(x_i, y_i), i = 1, \dots, 30\}$  fitted by a 1-D linear regression function  $y = w_0 + w_1x$ , a straight line, with  $w_0 = 1.81$ ,  $w_1 = 3.22$ . The correlation coefficient is  $\rho = 0.98$ , and  $ESS = 734.78$ ,  $RSS = 32.98$ ,  $TSS = 767.77$ ,  $R^2 = 0.96$ .

**Example 4:**

A set of  $N = 90$  data points  $\{(\mathbf{x}_i, y_i), i = 1, \dots, 90\}$  fitted by a 2-D linear regression function  $y = w_0 + w_1x_1 + w_2x_2$ , with  $w_0 = 2.82$ ,  $w_1 = 1.85$ ,  $w_2 = 4.31$ . This is a plane as shown in the figure below from two perspective view angles. Here  $ESS = 677.00$ ,  $RSS = 124.48$ ,  $TSS = 802.47$  and  $R^2 = 0.84$ .

Finally we note that regression analysis can be considered as a binary classification, when the regression function  $y = f(\mathbf{x})$ , as a hypersurface in a  $d + 1$  dimensional space spanned by  $\{x_1, \dots, x_d, y\}$ , is thresholded by a constant  $C$  (e.g.,  $C = 0$ ). The resulting equation  $f(\mathbf{x}) = C$  defines a hypersurface in the  $d$  dimensional space spanned by  $\{x_1, \dots, x_d\}$ , which partitions the space into two parts in such a way that all points on one side of the hypersurface satisfy  $f(\mathbf{x}) > C$ , while all points on the other side satisfy  $f(\mathbf{x}) < C$ . In other words, the regression function is a binary classifier that separates every point  $\mathbf{x}$  in the  $d$  dimensional space into two classes  $C_+$  and  $C_-$ , depending on whether  $f(\mathbf{x})$  is greater or smaller than  $C$ . Now each  $y_n$  corresponding to  $\mathbf{x}_n$  in the given dataset  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$  can be treated as a label indicating  $\mathbf{x}_n$  belong to class  $C_+$  if  $y_n > C$ , or  $C_-$  if  $y_n < C$ , and the regression problem becomes a binary classification problem.

### 4.3 Bias-Variance Tradeoff and Ridge Regression

In a generic regression problem, the goal is to model the unknown relationship  $y = f(\mathbf{x})$  between the dependent variable  $y$  and the  $d$  independent variables  $\mathbf{x} = [x_1, \dots, x_d]^T$ , by a regression function  $\hat{f}(\mathbf{x})$ , based on a set of given training data  $\mathcal{D} = \{(\mathbf{x}_n, y_n) \mid n = 1, \dots, N\}$  containing a set of observed data samples inevitably contaminated by some random noise  $e$ . Now corresponding to a noisy  $\mathbf{x}$  the observed value  $y = f(\mathbf{x}) + e$  is also random and so is the regression model  $\hat{f}(\mathbf{x})$ .

We assume the random noise  $e$  has zero mean and is independent of  $\hat{f}$ :

$$E[e] = 0, \quad \text{Var}[e] = E[(e - E[e])^2] = E[e^2] = \sigma^2 E[\hat{f}] = E[e] E[\hat{f}] \quad (4.39)$$

and measure how well the model  $\hat{f}(\mathbf{x})$  fits the noisy data  $y = f(\mathbf{x}) + e$  by the

mean squared error (MSE):

$$\begin{aligned}
\mathbb{E}[(y - \hat{f})^2] &= \mathbb{E}[(f + e - \hat{f})^2] = \mathbb{E}\left[\left[(f - E\hat{f}) + (E\hat{f} - \hat{f}) + e\right]^2\right] \\
&= \mathbb{E}\left[(f - E\hat{f})^2 + (E\hat{f} - \hat{f})^2 + e^2 + 2[e(f - E\hat{f}) + e(E\hat{f} - \hat{f}) + (f - E\hat{f})(E\hat{f} - \hat{f})]\right] \\
&= \mathbb{E}[(f - E\hat{f})^2] + \mathbb{E}[(E\hat{f} - \hat{f})^2] + \mathbb{E}(e^2) \\
&\quad + 2\mathbb{E}[e(f - E\hat{f})] + 2\mathbb{E}[e(E\hat{f} - \hat{f})] + 2\mathbb{E}[(E\hat{f} - \hat{f})(f - E\hat{f})] \\
&= \mathbb{E}[(f - E\hat{f})^2] + \mathbb{E}[(E\hat{f} - \hat{f})^2] + \mathbb{E}(e^2)
\end{aligned} \tag{4.40}$$

The last equality is due to the fact that all three cross terms are zero:

$$\begin{cases} \mathbb{E}[e(f - E\hat{f})] = \mathbb{E}[e] \mathbb{E}[f - E\hat{f}] = 0 \\ \mathbb{E}[e(E\hat{f} - \hat{f})] = \mathbb{E}[e] \mathbb{E}[E\hat{f} - \hat{f}] = 0 \\ \mathbb{E}[(E\hat{f} - \hat{f})(f - E\hat{f})] = (E\hat{f} - E\hat{f})(f - E\hat{f}) = 0 \end{cases} \tag{4.41}$$

The three remaining terms represent three different types of error:

- *Irreducible Error*  $\mathbb{E}[e^2] = \sigma^2$ , due to the inevitable observation error.
- *Bias Error*:  $\mathbb{E}[(f - E\hat{f})^2] = (f - E\hat{f})^2$ , the expected squared bias, the difference between the estimated function and the true function, measuring how well the estimated function  $\hat{f}$  models the true function  $f$ ;
- *Variance Error*:  $\mathbb{E}[\hat{f} - E\hat{f}]^2 = \text{Var}[\hat{f}]$ , the variance of the estimated function  $\hat{f}$ , measuring the variation of the model due to the noise in the training data.

We therefore get the following general relationship in terms of the three types of error:

$$\text{Mean Squared Error} = \text{Variance} + \text{Bias Error} + \text{Irreducible Error} \tag{4.42}$$

The variance error and bias error are illustrated below:

Given the total MSE and the irreducible error  $\sigma^2$ , the bias error and the variance error are complementary, i.e., if one error is higher, the other is lower, depending on the model function  $\hat{f}$  based on a specific algorithm. This relationship is called the *bias-variance tradeoff*, which is related to the major issue of overfitting vs. underfitting in supervised learning in general:

- *Overfitting*:

In order to match the training data closely, a complex model may be required to take into consideration detailed variations in the data which may be caused by noise. Such a model tends to have a low bias error, but may suffer from high variance error as it may be overly affected by the random noise (possibly some outliers) and does not generalize well across different datasets contaminated by varying noise.

- *Underfitting*:

In order to avoid being overly affected by the random noise, a simple model is used with the possible consequence that the variation in the data

is not closely match. Such a model tends to have low variance error as it is insensitive to the varying random noise accross datasets, but it may suffer from high bias error as it may not match the valid signal variations.

These two possibilities are summarized in the table and the plot below.

	Overfit	Underfit
Variance error	High	Low
Bias error	Low	High

(4.43)

As both overfitting and underfitting result in poor prediction capability, they need to be avoided by making a proper tradeoff between bias-error/underfitting and variance-error/overfitting. This is an important issue not only in the context of regression, but also in general in all supervised learning algorithms, for the purpose of predicting either a value as in regression, or a categorical labeling in classification, based on the output  $\hat{y} = f(\mathbf{x})$  as a function of the given data point  $\mathbf{x}$ . A general framework for all such algorithms is shown below, where the algorithm in the box is to come up with a model  $f(\mathbf{x}, \boldsymbol{\theta})$  of which the output  $\hat{y}$  is to match the labeling  $y$  of the data point  $\mathbf{x}$  in the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$  so that the residual  $r$  is minimized.

The issue of overfitting versus underfitting in supervised learning is further illustrated below, for regression on the left and classification on the right.

We see that in regression, the data points are modeled by two different regression functions, while in classification, the 2-D space is partitioned into two regions corresponding to two classes by two different *decision boundaries*. The issue is, which of the two models, in either regression or classification, fits the data better, in terms of underfitting versus overfitting:

- Underfitting:  
The simple linear regression function or decision boundary (red) may underft the data as it may miss some legitimate variations in the data.
- Overfitting:  
The complex regression model, e.g., a high order polynomial, or decision boundary (blue) may overfit the data as it may be overly affected by the variation due to the observation noise.

In general, based on a single set of training data points, it is impossible to distinguish between variations due to valid signal and random noise,, and to judge whether a learning algorithm either over or under fits the data. However, based on multiple datasets contaminated by different random noise, overfitting problem can be detected if the algorithm produces significantly different results when trained by different datasets. Based on this idea, the method of *cross-validation* can be used to train a learning algorithm by one subset of the training dataset and then test the algorithm by another subset of the data. if the residual error during testing is large, then the algorithm may suffer from overfitting.



Many regression and classification algorithms address the issue of overfitting versus underfitting by a process called *regularization* to prevent overfitting, especially when the problem is ill-conditioned, so that the model is less sensitive to noise in the training data and therefore more generalizable, while still able to capture the essential nature and behavior of the signal. Typically this is done by adjusting some hyperparameter of the model to make a proper tradeoff between the two extremes of over and underfitting.

As an example, consider the method of *ridge regression* for LS linear regression. If the  $N$  data points in the training set are barely independent, then matrix  $\mathbf{X}\mathbf{X}^T$  has a full rank but may be very close to singularity, i.e., its smallest eigenvalue  $\lambda_{min}$ , the smallest singular value  $\sigma_{min} = \sqrt{\lambda_{min}}$  of  $\mathbf{A}$ , is close to zero, and the normal equation  $\mathbf{X}\mathbf{X}^T\mathbf{w} = \mathbf{X}\mathbf{y}$  in Eq. (4.18) is ill-conditioned. Now the solution  $\mathbf{w} = (\mathbf{X}^T)^{-1}\mathbf{y}$  based on the pseudo-inverse is unstable and prone to noise. Any minor change due to noise in the dataset (in either  $\mathbf{X}$  or  $\mathbf{y}$ ) may cause a major change in  $\mathbf{w}$ , i.e., the resulting  $\mathbf{w}$  has a large variance error and the regression model overfits the data and is therefore poor generalizable.

In this case, the ridge regression method can be used to regularize the ill-conditioned problem by constructing an objective function  $J(\mathbf{w})$  that contains a regularization or penalty term  $\lambda\|\mathbf{w}\|^2/2$  to discourage large weights while minimizing the error  $\varepsilon = \|\mathbf{r}\|^2/2$ :

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}^T\mathbf{w}\|^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2 \quad (4.44)$$

This method is in general called *weight decay* and the hyperparameter  $\lambda$  is called the *weight decay parameter*. Solving the equation

$$\frac{d}{d\mathbf{w}}J(\mathbf{w}) = \mathbf{X}\mathbf{X}^T\mathbf{w} - \mathbf{X}\mathbf{y} + \lambda\mathbf{w} = (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})\mathbf{w} - \mathbf{X}\mathbf{y} = \mathbf{0} \quad (4.45)$$

we get

$$\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{X}\mathbf{y} \quad (4.46)$$

Now a proper tradeoff between accuracy and stability can be made by adjusting the hyperparameter  $\lambda$ :

- Small  $\lambda$ : the solution is more accurate but also more prone to noise and therefore less stable, i.e., large variance error but small bias error, or overfitting;
- Large  $\lambda$ : the solution is more stable as it is less affected by noise, but it may be less accurate, i.e., small variance error but large bias error, or underfitting.

To prevent overfitting, regularization by means of including a penalty term  $\|\mathbf{w}\|^2$  in the objective function is commonly used in many supervised learning algorithms to discourage large values in the weight vector  $\mathbf{w}$  as a parameter of the model for the dataset, such as in various regression algorithms in this chapter, and some popular classification algorithms in later chapters, including support vector machine in Chapter ?? and back propagation neural networks in Chapter ??.

#### 4.4 Regression Based on Basis Functions

The method of linear regression considered previously can be generalized to model nonlinear relationships between the dependent variable  $y$  and the independent variables in  $\mathbf{x} = [x_1, \dots, x_d]^T$  by a regression function as a linear combination of  $K$  nonlinear basis functions of  $\mathbf{x}$ :

$$\hat{y} = f(\mathbf{x}) = \sum_{k=1}^K w_k \varphi_k(\mathbf{x}, \boldsymbol{\theta}) = \boldsymbol{\varphi}(\mathbf{x})^T \mathbf{w} \quad (4.47)$$

where  $\varphi_k(\mathbf{x}, \boldsymbol{\theta})$ , ( $k = 1, \dots, K$ ) is one of the basis function parameterized by  $\boldsymbol{\theta}$  that span the function space in which  $\hat{y}$  resides, and  $\boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x}), \dots, \varphi_K(\mathbf{x})]^T$  is a K-D vector. Some of the commonly used basis functions are listed below:

- Power function:

$$\varphi_k(x) = x^{k-1} \quad (4.48)$$

The regression function  $f(x)$  is  $(K - 1)$ th order polynomial.

- Sinusoidal function:

$$\varphi_k(x, \omega) = \sin((k - 1)\omega x), \quad \text{or} \quad \varphi_k(x, \omega) = \cos((k - 1)\omega x) \quad (4.49)$$

The regression function  $f(x)$  is similar to the sine or cosine series expansion of the given function.

- Gaussian function:

$$\varphi_k(\mathbf{x}, \mathbf{c}_k, a) = \exp(a\|\mathbf{x} - \mathbf{c}_k\|^2) \quad (4.50)$$

This function is one member of a family of functions called *radial basis functions (RBFs)*, all of which are centered at point  $\mathbf{c}_k$ .

- Sigmoid function:

$$\varphi_k(\mathbf{x}, \mathbf{c}_k, a) = \frac{1}{1 + e^{-a\|\mathbf{x} - \mathbf{c}_k\|}} \quad (4.51)$$

These four types of basis functions are shown in the plot below for  $K = 7$ . In practice, choosing a proper type of basis functions may be a trial-and-error process depending on the specific dataset.

The original  $d$ -dimensional data space spanned by  $x_1, \dots, x_d$  is now converted to a  $K$ -dimensional space spanned by the  $K$  basis functions  $\varphi_1(\mathbf{x}), \dots, \varphi_K(\mathbf{x})$ . Applying such a regression model to each data point  $\mathbf{x}_n$ , we get

$$\hat{y}_n = \sum_{k=1}^K w_k \varphi_k(\mathbf{x}_n) = \boldsymbol{\varphi}(\mathbf{x}_n)^T \mathbf{w} \quad (n = 1, \dots, N) \quad (4.52)$$

These  $N$  equations can be written in matrix form:

$$\begin{aligned} \hat{\mathbf{y}} &= \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} \varphi_1(\mathbf{x}_1) & \cdots & \varphi_K(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ \varphi_1(\mathbf{x}_N) & \cdots & \varphi_K(\mathbf{x}_N) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix} \\ &= \begin{bmatrix} \varphi^T(\mathbf{x}_1) \\ \vdots \\ \varphi_K^T(\mathbf{x}_N) \end{bmatrix} \mathbf{w} = \Phi^T \mathbf{w} \end{aligned} \quad (4.53)$$

where  $\Phi$  is a  $K \times N$  matrix:

$$\Phi = \begin{bmatrix} \varphi_1(\mathbf{x}_1) & \cdots & \varphi_1(\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \varphi_K(\mathbf{x}_1) & \cdots & \varphi_K(\mathbf{x}_N) \end{bmatrix} = [\varphi(\mathbf{x}_1), \dots, \varphi(\mathbf{x}_N)] \quad (4.54)$$

Typically there are many more sample data points than the basis functions, i.e.,  $N \gg K$ .

If, specially,  $\varphi(u) = u$  is an identity function, then Eq. (4.47) becomes the same as Eq. (4.11) for linear regression. Also, as the nonlinear model  $\hat{\mathbf{y}} = \Phi^T \mathbf{w}$  in Eq. (4.53) is in the same form as the linear model  $\hat{\mathbf{y}} = \mathbf{X}^T \mathbf{w}$  in Eq. (4.11), it can be solved the same way as in Eq. (4.19), by simply replacing  $\mathbf{X}$  by  $\Phi$ :

$$\mathbf{w}^* = (\Phi \Phi^T)^{-1} \Phi \mathbf{y} = (\Phi^T)^{-} \mathbf{y} \quad (4.55)$$

where  $(\Phi^T)^{-} = (\Phi \Phi^T)^{-1} \Phi$  is the  $K \times N$  pseudo-inverse of the  $N \times K$  matrix  $\Phi^T$ . Now we get the output of the regression model

$$\hat{\mathbf{y}} = \Phi^T \mathbf{w} = \Phi^T (\Phi \Phi^T)^{-1} \Phi \mathbf{y} = \Phi^T (\Phi \Phi^T)^{-1} \Phi \mathbf{y} \quad (4.56)$$

The Matlab code segment below shows the essential part of the algorithm, where  $x(n)$  is the  $n$ th data sample  $\mathbf{x}_n$  out of the total of  $N$ ,  $y(x)$  is the vector  $\mathbf{y} = [y_1, \dots, y_N]^T$  with  $y_n = f(\mathbf{x}_n)$ , and  $c(k)$  contains the parameters for the  $k$ th basis function  $\phi_k(\mathbf{x}) = \phi(\mathbf{x}, c(k))$  represented by `phi` in the code.

```
for n=1:N
    for k=1:K
        Phi(k,n)=phi(x(n),c(k));
    end
end
w=pinv(Phi')*y(x);    % weight vector by LS method
yhat=Phi'*w;          % reconstructed function
```

#### Example 1:

The plots below show the approximations of a given function based on a set of  $N$  samples using linear regression based on the four different types of basis functions. We see that as the number of basis functions  $K$  increases, the approximation becomes more accurate.

more accurate

**Example 2:**

The linear regression method based on basis functions is used to approximate a 2-D function  $f(\mathbf{x}) = f(x_1, x_2)$  shown in the figure below (left), based on the function values in  $\{y_1, \dots, y_N\}$  at a set  $N$  sample points  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  taken from a grid in the 2-D space (right). The regression is based on  $K$  2-D Gaussian basis functions centered at different locations in the 2-D space shown in the following figure for  $K = 9$ .

The figure below shows the results based on different numbers of basis functions. Again, we see that the approximation becomes more accurate as the value of  $K$  increases.

In addition to the different types of basis functions considered above, we further consider the following basis functions:

$$\varphi_k(\mathbf{x}, \mathbf{w}_k, b_k) = g(\mathbf{w}_k^T \mathbf{x} + b_k) = g\left(\sum_{i=1}^d w_{ki} x_i + b_k\right) = g\left(\sum_{i=0}^d w_{ki} x_i\right) \quad (4.57)$$

where  $g(x)$ , called *activation function*, can take some different forms such as a sigmoid or rectified linear function (rectified linear unit or ReLU),

$$g(x) = \begin{cases} 1/(1 + e^{-x}) & \text{sigmoid function} \\ \max(0, x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} & \text{ReLU} \end{cases} \quad (4.58)$$

Now the regression function in Eq. (4.47) can be written as below for multiple outputs :

$$\hat{y}_j = \sum_{k=1}^K w_{jk}^o \varphi_k(\mathbf{x}, \boldsymbol{\theta}_k) = \sum_{k=1}^K w_{jk}^o g\left(\sum_{i=0}^d w_{ki}^h x_i\right) \quad (j = 1, \dots, m) \quad (4.59)$$

which can be interpreted as the output of a three-layer *neural network* shown below, containing an input layer of  $d$  nodes, a middle or hidden layer of  $K$  nodes with weights  $w_{ki}^h$ , ( $i = 0, \dots, d$ ,  $k = 1, \dots, K$ ), and an output layer of  $m$  nodes with weights  $w_{jk}^o$ , ( $k = 1, \dots, K$ ,  $j = 1, \dots, m$ ). These nodes are also called *neurons*, mimicking the biological neurons in the brain.

As shown in figure, the nodes of the input layer take the  $d$  components of a data point  $\mathbf{x}$  from the dataset one at a time, while each of the  $K$  nodes of the hidden layer calculates their linear combination  $\mathbf{w}_k^T \mathbf{x}$ , and, together with a bias term  $b_k$ , generates the basis function  $\varphi_k(\mathbf{x}) = g(\mathbf{w}_k^T \mathbf{x} + b_k)$  as its output. Finally, each node of the output layer takes these  $K$  basis function from the  $K$  hidden layer and generates as its output the regression function  $\hat{y} = f(\mathbf{x}) = \sum_{k=1}^K w_k \varphi_k(\mathbf{x})$  as the model of the dataset.

This network depends on two sets of weights, those represented by  $w_{ki}^h$  for the hidden nodes, and those represented by  $w_{jk}^o$  (found in Eq. (4.55)) for the output nodes. When there are multiple nodes in the output layer, they can each

approximate a different function. This three-layer network is of particular interest in neural network algorithms, to be considered in Chapter 9.///

**Example 3:**

This method of regression based on basis functions is used to approximate the same 2-D function  $f(\mathbf{x}) = f(x_1, x_2)$  in the previous example, as a linear combination of  $K$  basis functions  $\varphi_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k$  with each parameterized by  $\mathbf{w}_k$  and  $b_k$  (randomly generated). The Matlab code segment below shows the essential part of the program for this example, where  $f$  is an arbitrary function to be approximated.

```

a=@(x) x.*(x>0);           % ReLU activation function, or
a=@(x) 1/(1+exp(-x));      % Sigmoid activation function
g=@(x,w,b)a(w'*x+b);      % the basis function

[X,Y]=meshgrid(xmin:1:xmax, ymin:1:ymin); % define 2-D points
nx=size(X,2);              % number of samples in first dimension
ny=size(Y,1);              % number of samples in second dimension
N=nx*ny;                   % total number of data samples
Phi=zeros(N,K);           % K basis functions evaluated at N sample points
W=1-2*rand(2,K);          % random initialization of weights
b=1-2*rand(1,K);          % and biases for K basis functions
n=0;
for i=1:nx
    x(1)=X(1,i);          % first component
    for j=1:ny
        x(2)=Y(j,1);      % second component
        n=n+1;            % of the nth sample
        Y(i,j)=func(x);   % function evaluated at the sample
        for k=1:K          % basis functions evaluated at the sample
            Phi(k,n)=g(x,W(:,k),b(k));
        end
    end
end
end
y=reshape(Y,N,1);         % convert function values to vector
w=pinv(Phi')*y;          % find weights by LS method
yhat=Phi'*w;             % reconstructed function
Yhat=reshape(yhat,m,n);  % convert vector y to 2-D to compare w/ Y

```

The figures below show the basis functions based on both the sigmoid and ReLU activation functions, and the corresponding regression results. We note that as more basis functions are used, the more accurately the function is approximated.

- Regression based on sigmoid function:
- Regression based on ReLU function:

## 4.5 Bayesian Linear Regression

We revisit the linear regression problem but from the viewpoint of Bayesian inference. Now the parameter  $\mathbf{w}$  in the linear regression model  $\hat{y} = f(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T \mathbf{w}$  is assumed to be a random vector, and, as a function of  $\mathbf{w}$ , the regression function  $f(\mathbf{x}, \mathbf{w})$  is also random, and so is the residual based on the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ :

$$\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}^T \mathbf{w}, \quad \text{i.e.} \quad \mathbf{y} = \hat{\mathbf{y}} + \mathbf{r} = \mathbf{f}(\mathbf{X}) + \mathbf{r} = \mathbf{X}^T \mathbf{w} + \mathbf{r} \quad (4.60)$$

We further assume  $\mathbf{w}$  has a zero-mean normal prior pdf:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}, \mathbf{0}, \Sigma_w) \propto \exp\left(-\frac{1}{2} \mathbf{w}^T \Sigma_w^{-1} \mathbf{w}\right), \quad (4.61)$$

based on the desire that the weights in  $\mathbf{w}$  should take values around zero, instead of large values on either positive or negative, to avoid potential overfitting. We also assume the residual  $\mathbf{r}$  has a zero-mean normal pdf with a diagonal covariance matrix, i.e., all components of  $\mathbf{r}$  are independent of each other:

$$p(\mathbf{r}) = \mathcal{N}(\mathbf{r}, \mathbf{0}, \sigma_r^2 \mathbf{I}) \propto \exp\left(-\frac{1}{2\sigma_r^2} \mathbf{r}^T \mathbf{r}\right) = \exp\left(-\frac{\|\mathbf{r}\|^2}{2\sigma_r^2}\right) \quad (4.62)$$

As a linear transformation of  $\mathbf{w}$ , the regression function  $\hat{\mathbf{y}} = \mathbf{X}^T \mathbf{w}$  also has a zero-mean Gaussian distribution, and so is  $\mathbf{y} = \mathbf{X}^T \mathbf{w} + \mathbf{r}$  as the sum of two Gaussians.

To obtain the model parameter  $\mathbf{w}$ , we consider its *likelihood*  $L(\mathbf{w}|\mathcal{D})$  based on the observed data  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ :

$$\begin{aligned} L(\mathbf{w}|\mathcal{D}) &\propto p(\mathcal{D}|\mathbf{w}) = p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = p(\mathbf{X}^T \mathbf{w} + \mathbf{r}) = \mathcal{N}(\mathbf{y}, \mathbf{X}^T \mathbf{w}, \sigma_r^2 \mathbf{I}) \\ &\propto \exp\left(-\frac{1}{2\sigma_r^2} (\mathbf{y} - \mathbf{X}^T \mathbf{w})^T (\mathbf{y} - \mathbf{X}^T \mathbf{w})\right) = \exp\left(-\frac{1}{2\sigma_r^2} \|\mathbf{r}\|^2\right) \end{aligned} \quad (4.63)$$

This is the same Gaussian as  $p(\mathbf{r}) = \mathcal{N}(\mathbf{r}, \mathbf{0}, \sigma_r^2 \mathbf{I})$ , but shifted by  $\mathbf{X}^T \mathbf{w}$ . Now we can find the optimal model parameter  $\mathbf{w}^*$  by maximizing the likelihood  $L(\mathbf{w}|\mathcal{D})$ . This method is called *maximum likelihood estimation (MLE)*. But as maximizing  $L(\mathbf{w}|\mathcal{D})$  is equivalent to minimizing the SSE  $\|\mathbf{r}\|^2$ , the MLE method is equivalent to the least squares method considered before.

Maximizing the likelihood above is equivalent to minimizing the negative log likelihood:

$$-l(\mathbf{w}|\mathbf{X}, \mathbf{y}) = -\log L(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{\|\mathbf{r}\|^2}{2\sigma_r^2} \propto \|\mathbf{r}\|^2 \quad (4.64)$$

which is the same as the squared error  $\varepsilon(\mathbf{w})$  defined in Eq. (4.17) to be minimized by the LS method previously considered. We therefore see that the least squares method and the maximum likelihood method are equivalent to each other in this case.

We can further find the posterior  $p(\mathbf{w}|\mathcal{D})$  of the model parameter  $\mathbf{w}$  given

the observed data  $\mathcal{D}$ , as the product of prior  $p(\mathbf{w})$  and likelihood  $L(\mathbf{w}|\mathcal{D}) \propto p(\mathbf{y}|\mathbf{X}, \mathbf{w})$ :

$$\begin{aligned}
p(\mathbf{w}|\mathcal{D}) &= p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})} = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})} \\
&\propto p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w}) = \mathcal{N}(\mathbf{y}, \mathbf{X}^T \mathbf{w}, \sigma_r^2 \mathbf{I}) \mathcal{N}(\mathbf{w}, \mathbf{0}, \Sigma_w) \\
&\propto \exp \left[ -\frac{1}{2} \left( \frac{1}{\sigma_r^2} (\mathbf{y} - \mathbf{X}^T \mathbf{w})^T (\mathbf{y} - \mathbf{X}^T \mathbf{w}) + \mathbf{w}^T \Sigma_w^{-1} \mathbf{w} \right) \right] \\
&\propto \exp \left[ -\frac{1}{2} \left( \frac{1}{\sigma_r^2} \mathbf{y}^T \mathbf{y} - \frac{2}{\sigma_r^2} (\mathbf{X}\mathbf{y})^T \mathbf{w} + \frac{1}{\sigma_r^2} \mathbf{w}^T \mathbf{X}\mathbf{X}^T \mathbf{w} + \mathbf{w}^T \Sigma_w^{-1} \mathbf{w} \right) \right] \\
&\propto \exp \left[ -\frac{1}{2} \left( \mathbf{w}^T \left( \frac{1}{\sigma_r^2} \mathbf{X}\mathbf{X}^T + \Sigma_w^{-1} \right) \mathbf{w} - \frac{2}{\sigma_r^2} (\mathbf{X}\mathbf{y})^T \mathbf{w} \right) \right] \quad (4.65)
\end{aligned}$$

where all constant scaling factors (irrelevant to  $\mathbf{w}$ ) are dropped for simplicity. As both the prior and the likelihood are Gaussian, this posterior, when normalized, is also Gaussian in terms of the mean  $\mathbf{m}_{w/d}$  and covariance  $\Sigma_{w/d}$ :

$$\begin{aligned}
p(\mathbf{w}|\mathcal{D}) &= \mathcal{N}(\mathbf{w}, \mathbf{m}_{w/d}, \Sigma_{w/d}) \propto \exp \left[ -\frac{1}{2} \left( (\mathbf{w} - \mathbf{m}_{w/d})^T \Sigma_{w/d}^{-1} (\mathbf{w} - \mathbf{m}_{w/d}) \right) \right] \\
&\propto \exp \left[ -\frac{1}{2} \left( \mathbf{w}^T \Sigma_{w/d}^{-1} \mathbf{w} - 2\mathbf{m}_{w/d}^T \Sigma_{w/d}^{-1} \mathbf{w} \right) \right] \quad (4.66)
\end{aligned}$$

with the following covariance and mean:

$$\begin{aligned}
\Sigma_{w/d} &= \left( \frac{1}{\sigma_r^2} \mathbf{X}\mathbf{X}^T + \Sigma_w^{-1} \right)^{-1} \\
\mathbf{m}_{w/d} &= \frac{1}{\sigma_r^2} \Sigma_{w/d} \mathbf{X}\mathbf{y} = (\mathbf{X}\mathbf{X}^T + \sigma_r^2 \Sigma_w^{-1})^{-1} \mathbf{X}\mathbf{y} \quad (4.67)
\end{aligned}$$

Now we can find the optimal model parameter  $\mathbf{w}^* = \mathbf{m}_{w/d}$  equal to the mean of the Gaussian posterior where it is maximized. This solution is called the *maximum a posteriori (MAP)* estimate of the model parameter  $\mathbf{w}$ .

We can now predict the function value of any unlabeled test data point  $\mathbf{x}_*$  based on the conditional probability distribution  $p(f_*|\mathbf{x}_*, \mathcal{D})$  of the regression model  $\hat{y}_* = f_* = f(\mathbf{x}_*) = \mathbf{x}_*^T \mathbf{w}$ , which, as a linear transformation of the random parameters in  $\mathbf{w}$ , is also Gaussian (Section 3.1.3).

$$p(f_*|\mathbf{x}_*, \mathcal{D}) = p(\mathbf{x}_*^T \mathbf{w}|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(f_*, \mathbf{x}_*^T \mathbf{m}_{w/d}, \mathbf{x}_*^T \Sigma_{w/d} \mathbf{x}_*) \quad (4.68)$$

with the following mean and variance:

$$E[f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}] = \mathbf{x}_*^T \mathbf{m}_{w/d} = \mathbf{x}_*^T (\mathbf{X}\mathbf{X}^T + \sigma_r^2 \Sigma_w^{-1})^{-1} \mathbf{X}\mathbf{y} \quad (4.69)$$

$$\text{Var}[f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}] = \mathbf{x}_*^T \Sigma_{w/d} \mathbf{x}_* = \mathbf{x}_*^T \left( \frac{1}{\sigma_r^2} \mathbf{X}\mathbf{X}^T + \Sigma_w^{-1} \right)^{-1} \mathbf{x}_* \quad (4.70)$$

Furthermore, given a set of  $M$  unlabeled test data points in  $\mathbf{X}_* = [\mathbf{x}_{1*}, \dots, \mathbf{x}_{M*}]^T$ ,

we can find

$$\hat{\mathbf{y}}_* = \mathbf{f}_* = \mathbf{f}(\mathbf{x}_*) = \begin{bmatrix} f(\mathbf{x}_{1*}) \\ \vdots \\ f(\mathbf{x}_{M*}) \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{1*}^T \\ \vdots \\ \mathbf{x}_{M*}^T \end{bmatrix} \mathbf{w} = [\mathbf{x}_{1*}, \dots, \mathbf{x}_{M*}]^T \mathbf{w} = \mathbf{X}_*^T \mathbf{w} \quad (4.71)$$

As a linear combination of Gaussian  $\mathbf{w}$ ,  $\mathbf{f}_* = \mathbf{X}_*^T \mathbf{w}$  is also Gaussian:

$$p(\mathbf{f}_* | \mathbf{X}_*, \mathcal{D}) = p(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{f}_*, \mathbf{X}_*^T \mathbf{m}_{w/d}, \mathbf{X}_*^T \boldsymbol{\Sigma}_{w/d} \mathbf{X}_*) \quad (4.72)$$

with the following mean and covariance:

$$\begin{aligned} E[\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}] &= E[\mathbf{X}_*^T \mathbf{w}] = \mathbf{X}_*^T E[\mathbf{w}] = \mathbf{X}_*^T \mathbf{m}_{w/d} = \mathbf{X}_*^T (\mathbf{X} \mathbf{X}^T + \sigma_r^2 \boldsymbol{\Sigma}_w^{-1})^{-1} \mathbf{X} \mathbf{y} \\ Cov(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}) &= E[\mathbf{f}_* \mathbf{f}_*^T] = E[\mathbf{X}_*^T \mathbf{w} \mathbf{w}^T \mathbf{X}_*] = \mathbf{X}_*^T E[\mathbf{w} \mathbf{w}^T] \mathbf{X}_* = \mathbf{X}_*^T \boldsymbol{\Sigma}_{w/d} \mathbf{X}_* \end{aligned} \quad (4.74)$$

The mean  $E[\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}]$  above can be considered as the estimated function value  $\hat{\mathbf{y}}_* = \mathbf{f}_* = \mathbf{X}_*^T \mathbf{w}$  at the test points in  $\mathbf{X}_*$ , while the variance  $\text{Var}[\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}]$  as the confidence or certainty of the estimation.

Such a result produced by the Bayesian method in terms of the mean and covariance of  $\mathbf{f}_* = \hat{\mathbf{y}}_*$  as a random vector is quite different from the result produced by a frequentist method such as the linear least square regression in terms of an estimated value of  $\hat{\mathbf{y}}_*$  as a deterministic variable. However, in the special case of  $\sigma_r = 0$ , i.e., the residual  $\mathbf{r}$  is no longer treated as a random variable, then Eq. (4.73) above becomes the same as Eq. (4.20) produced by linear least square regression, in other words, the linear method method can be considered as a special case of the Bayesian regression when the residual is deterministic.

The method considered above is based on the linear regression model  $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ , but it can also be generalized to nonlinear functions, same as in the case of regression based on basis functions discussed in Section ???. To do so, the regression function is assumed to be a linear combination of a set of  $K$  nonlinear mapping functions  $\varphi_k(\mathbf{x})$  ( $k = 1, \dots, K$ ):

$$y = f(\mathbf{x}) + e = \sum_{k=1}^K w_k \varphi_k(\mathbf{x}) + r = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + r = \boldsymbol{\phi}^T(\mathbf{x}) \mathbf{w} + r \quad (4.75)$$

where

$$\mathbf{w} = [w_1, \dots, w_K]^T, \quad \boldsymbol{\phi}(\mathbf{x}) = [\varphi_1(\mathbf{x}), \dots, \varphi_K(\mathbf{x})]^T \quad (4.76)$$

For example, some commonly used mapping functions are listed in Eq. (4.57).

Applying this regression model to each of the  $N$  observed data points in  $\mathcal{D} = \{(\mathbf{x}_n, y_n), (n = 1, \dots, N)\}$ , we get  $\hat{y}_n = \sum_{k=1}^K w_k \varphi_k(x_n)$  with residual  $r_n =$



$y_n - \hat{y}_n$ , which can be written in matrix form:

$$\begin{aligned} \mathbf{y} &= \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}_{N \times 1} = \begin{bmatrix} \varphi_1(\mathbf{x}_1) & \cdots & \varphi_K(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ \varphi_1(\mathbf{x}_N) & \cdots & \varphi_K(\mathbf{x}_N) \end{bmatrix}_{N \times K} \begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix}_{K \times 1} + \begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix}_{N \times 1} \\ &= \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_N^T \end{bmatrix} \mathbf{w} + \mathbf{r} = \mathbf{\Phi}^T \mathbf{w} + \mathbf{r} \end{aligned} \quad (4.77)$$

where, as previously defined in Eq. (4.54),

$$\phi_n = \phi(\mathbf{x}_n) = \begin{bmatrix} \varphi_1(\mathbf{x}_n) \\ \vdots \\ \varphi_K(\mathbf{x}_n) \end{bmatrix}, \quad \mathbf{\Phi} = [\phi_1, \cdots, \phi_N] \quad (4.78)$$

As this nonlinear model  $\hat{\mathbf{y}} = \mathbf{\Phi}^T \mathbf{w}$  is in the same form as the linear model  $\hat{\mathbf{y}} = \mathbf{X}^T \mathbf{w}$ , all previous discussion for the linear Bayesian regression is still valid for this nonlinear model, if  $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]_{(d+1) \times N}$  is replaced by  $\mathbf{\Phi} = [\phi_1, \cdots, \phi_N]_{K \times N}$ . We can get the covariance matrix and mean vector of the posterior  $p(\mathbf{w}|\mathcal{D})$  in the  $K$ -dimensional space based on the observed data  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ :

$$\begin{aligned} \Sigma_{w/d} &= \left( \frac{1}{\sigma_r^2} \mathbf{\Phi} \mathbf{\Phi}^T + \Sigma_w^{-1} \right)^{-1} \\ \mathbf{m}_{w/d} &= \frac{1}{\sigma_r^2} \Sigma_{w/d} \mathbf{\Phi} \mathbf{y} = \left( \mathbf{\Phi} \mathbf{\Phi}^T + \sigma_r^2 \Sigma_w^{-1} \right)^{-1} \mathbf{\Phi} \mathbf{y} \end{aligned} \quad (4.79)$$

The function  $\hat{\mathbf{y}}_* = \mathbf{f}(\mathbf{X}_*)$  of the multiple test points in  $\mathbf{X}_* = [\mathbf{x}_{*1}, \cdots, \mathbf{x}_{*M}]^T$  is a linear transformation of  $\mathbf{w}$ :

$$\hat{\mathbf{y}}_* = \mathbf{f}_* = \mathbf{f}(\mathbf{x}_*) = \begin{bmatrix} f_1(\mathbf{x}_*) \\ \vdots \\ f_M(\mathbf{x}_*) \end{bmatrix} = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_M^T \end{bmatrix}_* \mathbf{w} = [\phi_1, \cdots, \phi_M]^T \mathbf{w} = \mathbf{\Phi}_*^T \mathbf{w} \quad (4.80)$$

and it also has a Gaussian joint distribution:

$$p(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{\Phi}_*^T \mathbf{m}_{w/d}, \mathbf{\Phi}_*^T \Sigma_{w/d} \mathbf{\Phi}_*) \quad (4.81)$$

with the mean and covariance:

$$\begin{aligned} \mathbb{E}[\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}] &= \mathbb{E}[\mathbf{\Phi}_*^T \mathbf{w}] = \mathbf{\Phi}_*^T \mathbb{E}[\mathbf{w}] = \mathbf{\Phi}_*^T \mathbf{m}_{w/d} \\ \text{Cov}(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}) &= \mathbb{E}[\mathbf{f}_* \mathbf{f}_*^T] = \mathbb{E}[\mathbf{\Phi}_*^T \mathbf{w} \mathbf{w}^T \mathbf{\Phi}_*] = \mathbf{\Phi}_*^T \mathbb{E}[\mathbf{w} \mathbf{w}^T] \mathbf{\Phi}_* = \mathbf{\Phi}_*^T \Sigma_{w/d} \mathbf{\Phi}_* \end{aligned} \quad (4.82)$$

They can be reformulated to become:

$$\begin{aligned}
\mathbb{E}[\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}] &= \mathbf{\Phi}_*^T \mathbf{m}_{w/d} = \mathbf{\Phi}_*^T (\mathbf{\Phi} \mathbf{\Phi}^T + \sigma_r^2 \mathbf{\Sigma}_w^{-1})^{-1} \mathbf{\Phi} \mathbf{y} \\
&= \mathbf{\Phi}_*^T (\mathbf{\Sigma}_w \mathbf{\Phi}) (\mathbf{\Sigma}_w \mathbf{\Phi})^{-1} (\mathbf{\Phi} \mathbf{\Phi}^T + \sigma_r^2 \mathbf{\Sigma}_w^{-1})^{-1} (\mathbf{\Phi}^{-1})^{-1} \mathbf{y} \\
&= \mathbf{\Phi}_*^T \mathbf{\Sigma}_w \mathbf{\Phi} [\mathbf{\Phi}^{-1} (\mathbf{\Phi} \mathbf{\Phi}^T + \sigma_r^2 \mathbf{\Sigma}_w^{-1}) \mathbf{\Sigma}_w \mathbf{\Phi}]^{-1} \mathbf{y} \\
&= \mathbf{\Phi}_*^T \mathbf{\Sigma}_w \mathbf{\Phi} (\mathbf{\Phi}^T \mathbf{\Sigma}_w \mathbf{\Phi} + \sigma_r^2 \mathbf{I})^{-1} \mathbf{y}
\end{aligned} \tag{4.83}$$

and

$$\begin{aligned}
Cov(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}) &= \mathbf{\Phi}_*^T \mathbf{\Sigma}_{w/d} \mathbf{\Phi}_* = \mathbf{\Phi}_*^T \left( \frac{1}{\sigma_r^2} \mathbf{\Phi} \mathbf{\Phi}^T + \mathbf{\Sigma}_w^{-1} \right)^{-1} \mathbf{\Phi}_* \\
&= \mathbf{\Phi}_*^T [\mathbf{\Sigma}_w - \mathbf{\Sigma}_w \mathbf{\Phi} (\mathbf{\Phi}^T \mathbf{\Sigma}_w \mathbf{\Phi} + \sigma_r^2 \mathbf{I})^{-1} \mathbf{\Phi}^T \mathbf{\Sigma}_w] \mathbf{\Phi}_* \\
&= \mathbf{\Phi}_*^T \mathbf{\Sigma}_w \mathbf{\Phi}_* - \mathbf{\Phi}_*^T \mathbf{\Sigma}_w \mathbf{\Phi} (\mathbf{\Phi}^T \mathbf{\Sigma}_w \mathbf{\Phi} + \sigma_r^2 \mathbf{I})^{-1} \mathbf{\Phi}^T \mathbf{\Sigma}_w \mathbf{\Phi}_*
\end{aligned} \tag{4.84}$$

where we have used Woodbury identity in Section ??.

We note that in these expressions the mapping function  $\varphi(\mathbf{x})$  only shows up in the quadratic form of  $\mathbf{\Phi}^T \mathbf{\Sigma}_w \mathbf{\Phi}$ ,  $\mathbf{\Phi}^T \mathbf{\Sigma}_w \mathbf{\Phi}_*$ , or  $\mathbf{\Phi}_*^T \mathbf{\Sigma}_w \mathbf{\Phi}_*$ . This suggests that we may consider using *kernel method* (Section ??, by which some significant advantage may be gained (such as in the support-vector machines (Section 13)). Specifically, if we replace the mn-th component  $\phi^T(\mathbf{x}_m) \mathbf{\Sigma}_w \phi(\mathbf{x}_n)$  of  $\mathbf{\Phi}^T \mathbf{\Sigma}_w \mathbf{\Phi}$  by a *kernel function* only in terms of  $\mathbf{x}_m$  and  $\mathbf{x}_n$ :

$$\phi^T(\mathbf{x}_m) \mathbf{\Sigma}_w \phi(\mathbf{x}_n) = k(\mathbf{x}_m, \mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{x}_m) \tag{4.85}$$

then the mapping function  $\varphi(\mathbf{x})$  no longer needs to be explicitly specified. This idea is called the *kernel trick*. Now we have

$$\mathbf{\Phi}^T \mathbf{\Sigma}_w \mathbf{\Phi} = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_N^T \end{bmatrix} \mathbf{\Sigma}_w [\phi_1, \dots, \phi_N] = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} = \mathbf{K} \tag{4.86}$$

and similarly we also have  $\mathbf{\Phi}^T \mathbf{\Sigma}_w \mathbf{\Phi}_* = \mathbf{K}_*$  and  $\mathbf{\Phi}_*^T \mathbf{\Sigma}_w \mathbf{\Phi} = \mathbf{K}_*^T$ . These kernel functions are symmetric, same as all covariance matrices. Now the mean and covariance of  $\mathbf{f}_*$  given above can be expressed as:

$$\begin{aligned}
\mathbb{E}[\mathbf{f}_*] &= \mathbf{K}_*^T (\mathbf{K} + \sigma_r^2 \mathbf{I})^{-1} \mathbf{y} \\
Cov(\mathbf{f}_*) &= \mathbf{K}_{**} - \mathbf{K}_*^T (\mathbf{K} + \sigma_r^2 \mathbf{I})^{-1} \mathbf{K}_*
\end{aligned} \tag{4.87}$$

As a general summary of the method discussed above, we note that it is essentially different from the frequentist approach, in the sense that the estimated result produced is no longer a set specific values of some unknown variables in question (function values, parameters of regression models, etc.), but the probability distribution of these variables in terms of the expectation and covariance. Consequently, we can not only treat the expectation as the estimated values, but also use the variance as the confidence or certainty of such an estimation.

Below is a segment of Matlab code for estimating the mean and covariance

of weight vector  $\mathbf{w}$  by the Bayes linear regression. Here  $\mathbf{X}$  and  $\mathbf{y}$  are for  $\mathbf{X}$  and  $\mathbf{y}$  of the training data  $\mathcal{D}$ , and  $\mathbf{Xstar}$ ,  $\Phi$  and  $\Phi_{\text{star}}$  are for  $\mathbf{X}_*$ ,  $\Phi$  and  $\Phi_*$ , respectively.

```

SigmaP=eye(d); % prior variance of r in original space
SigmaQ=eye(D); % prior variance of r in feature space
Sigmax1=inv(X*X'/sigmaN^2+inv(SigmaP)); % covariance of w by Bayes regression
Sigmax2=inv(Phi0*Phi0'/sigmaN^2+inv(SigmaQ)); % covariance of w with basis function

w0=inv(X*X')*X*y % weight vector w by LS method
w1=Sigmax1*X*y/sigmaN^2 % mean of w by Bayes regression
w2=Sigmax2*Phi0*y/sigmaN^2; % mean of w vector by Bayes regression with basis funct

y0=w0'*Xstar; % y by least squares method
y1=w1'*Xstar; % y by Bayes regression
y2=w2'*Phistar0; % y by Bayes regression with basis function

for i=1:M
    sgm1(i)=sqrt(Xstar(:,i)'*Sigmax1*Xstar(:,i)); % variance of estimated y
    sgm2(i)=sqrt(Phistar(:,i)'*Sigmax2*Phistar(:,i));
end

```

### Example

Consider a 2-D linear function

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} = [w_0, w_1] \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = w_0 x_0 + w_1 x_1 = 0.5 + 0.5 x_1 \quad (4.88)$$

where we have assumed  $x_0 = 1$ ,  $w_0 = -1/2$  and  $w_1 = 1/2$ , i.e., the function is actually a straight line  $f(x) = (-1 + x)/2$  with intercept  $w_0 = -1/2$  and slope  $w_1 = 1/2$ . The distribution of the additive noise (residual) is  $p(e) = \mathcal{N}(0, \sigma_r = 0.8)$ . The prior distributions of the weights  $\mathbf{w} = [w_0, w_1]^T$  is assumed to be  $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ , and posterior  $p(\mathbf{w}|\mathcal{D})$  can be obtained based on the observed data  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ , both shown in the figure below. We see that the posterior is concentrated around  $\hat{w}_1 = -0.28$  and  $\hat{w}_2 = 0.39$ , the estimates of the true weights  $w_1 = -0.5$  and  $w_2 = 0.5$ , based on the observed noisy data.

Various regression methods including both the least-squared method and the Bayesian linear regression methods, and in feature space spanned by a set of basis functions  $\phi_i(\mathbf{x})$ , ( $i = 1, \dots, D = 5$ ) based on each of the three functions given above are carried out with their results shown in the figure below:

The LS errors for these methods are listed below:

Method	LS Error	$\mathbf{w} = [w_1, w_2]^T$
Pseudo inverse	0.4730	$[-0.28, 0.39]^T$
Bayesian linear regression	0.4742	$[-0.22, 0.38]^T$
$\phi = \exp(a\ \mathbf{x} - \mathbf{c}\ ^2)$	0.6196	
$\phi = \ \mathbf{x} - \mathbf{c}\ $	0.4429	
$\phi = \exp(-a\ \mathbf{x} - \mathbf{c}\ ^2)$	0.4459	

(4.89)

We see that the linear methods (the pseudo inverse and the Bayesian regression) produce similar estimated weights error, but smaller error can be achieved by the same method of Bayesian linear regression if proper nonlinear mapping is used (e.g., the last two mapping functions).

# 5 Nonlinear Regression

---

## 5.1 Nonlinear Least Squares Regression

The goal of nonlinear least squares (LS) regression is to model the nonlinear relationship between the dependent variable  $y$  and independent variables in  $\mathbf{x} = [x_1, \dots, x_d]^T$ , based on the given training dataset  $\mathcal{D} = \{(\mathbf{x}_n, y_n), n = 1, \dots, N\}$ , by a nonlinear regression function parameterized by a set of  $M$  parameters represented by  $\boldsymbol{\theta} = [\theta_1, \dots, \theta_M]^T$ :

$$\hat{y} = f(x_1, \dots, x_d, \theta_1, \dots, \theta_M) = f(\mathbf{x}, \boldsymbol{\theta}) \quad (5.1)$$

While the specific form of the nonlinear function is typically determined based on certain prior knowledge, we need to find the parameter  $\boldsymbol{\theta}$  for the model to fit the training dataset  $\mathcal{D}$ , so that, its output  $\hat{\mathbf{y}}$  matches the ground truth labeling  $\mathbf{y}$  in the training set, i.e., the residual  $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$  is ideally zero:

$$\mathbf{r}(\boldsymbol{\theta}) = \begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} f_1(\boldsymbol{\theta}) \\ \vdots \\ f_N(\boldsymbol{\theta}) \end{bmatrix} = \mathbf{y} - \hat{\mathbf{y}}(\boldsymbol{\theta}) = \mathbf{y} - \mathbf{f}(\mathbf{X}, \boldsymbol{\theta}) = \mathbf{0} \quad (5.2)$$

where  $\hat{y}(\mathbf{x}_n, \boldsymbol{\theta}) = f_n(\boldsymbol{\theta}) = f(\mathbf{x}_n, \boldsymbol{\theta})$  is the predicted output by the regression function based on the  $n$ th data point  $\mathbf{x}_n$ . We see that the regression problem is equivalent to solving a nonlinear equation system of  $N$  equations for the  $M$  unknown parameters in  $\boldsymbol{\theta}$ . As typically there are many more data samples than the unknown parameters, i.e.,  $N \gg M$ , the equation  $\mathbf{r} = \mathbf{0}$  is over-overconstrained without an exact solution. We therefore consider the optimal parameter  $\boldsymbol{\theta}^*$  that minimizes the sum-of-squared error based on the residual  $\mathbf{r}$ :

$$\varepsilon(\boldsymbol{\theta}) = \frac{1}{2} \|\mathbf{r}\|^2 = \frac{1}{2} \sum_{n=1}^N r_n^2 = \frac{1}{2} \sum_{n=1}^N [y_n - f_n(\boldsymbol{\theta})]^2 \quad (5.3)$$

To do so, we first consider the gradient descent method based on the gradient of this error function:

$$\begin{aligned} \mathbf{g}_\varepsilon(\boldsymbol{\theta}) &= \frac{d}{d\boldsymbol{\theta}} \varepsilon(\boldsymbol{\theta}) = \frac{d}{d\boldsymbol{\theta}} \left( \frac{1}{2} \sum_{n=1}^N r_n^2 \right) = \sum_{n=1}^N \frac{dr_n}{d\boldsymbol{\theta}} r_n \\ &= - \sum_{n=1}^N \frac{d}{d\boldsymbol{\theta}} f_n(\boldsymbol{\theta}) [y_n - f_n(\boldsymbol{\theta})] = -\mathbf{J}_f^T \mathbf{r} \end{aligned} \quad (5.4)$$

where  $\mathbf{J}_f$  is the  $N \times M$  Jacobian matrix  $\mathbf{J}_f$  of  $\mathbf{f}(\mathbf{X}, \boldsymbol{\theta})$ . The  $m$ th component of the gradient vector is:

$$\begin{aligned} g_m(\boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_m} \varepsilon(\boldsymbol{\theta}) = \frac{1}{2} \frac{\partial}{\partial \theta_m} \|\mathbf{r}\|^2 = \frac{1}{2} \sum_{n=1}^N \frac{\partial}{\partial \theta_m} r_n^2 = \sum_{n=1}^N \frac{\partial r_n}{\partial \theta_m} r_n \\ &= \sum_{n=1}^N \frac{\partial f_n(\boldsymbol{\theta})}{\partial \theta_m} [f_n(\boldsymbol{\theta}) - y_n] = \sum_{n=1}^N J_{nm} [f_n(\boldsymbol{\theta}) - y_n] \sum_{n=1}^N J_{nm} r_n \quad (m = 1, \dots, M) \end{aligned} \quad (5.5)$$

where  $J_{nm} = \partial f_n(\boldsymbol{\theta}) / \partial \theta_m$  is the element in the  $n$ th row and  $m$ th column of  $\mathbf{J}_f$ , which is simply the negative version of Jacobian of  $\mathbf{r} = \mathbf{y} - \mathbf{f}(\mathbf{X}, \boldsymbol{\theta})$ :

$$\mathbf{J}_r(\boldsymbol{\theta}) = \frac{d}{d\boldsymbol{\theta}} \mathbf{r}(\boldsymbol{\theta}) = \frac{d}{d\boldsymbol{\theta}} [\mathbf{y} - \mathbf{f}(\mathbf{X}, \boldsymbol{\theta})] = -\frac{d}{d\boldsymbol{\theta}} \mathbf{f}(\mathbf{X}, \boldsymbol{\theta}) = -\mathbf{J}_f(\boldsymbol{\theta}) \quad (5.6)$$

The optimal parameters in  $\boldsymbol{\theta}$  that minimizes  $\varepsilon(\boldsymbol{\theta})$  can now be found iteratively by the gradient descent method

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \delta_n \mathbf{g}_\varepsilon(\boldsymbol{\theta}_n) = \boldsymbol{\theta}_n - \delta_n \mathbf{J}_f^T \mathbf{r} \quad (5.7)$$

Setting the value for the step size  $\delta_n$  properly may be tricky, as it depends on the specific landscape of the error hypersurface  $\varepsilon(\boldsymbol{\theta})$ . If  $\delta_n$  is too large, a minimum may be skipped over, but if  $\delta_n$  is too small, the convergence of the iteration may be too slow.

This gradient method is also called *batch gradient descent (BGD)*, as the gradient in Eq. (5.4) is calculated based on the entire batch of all  $N$  data points in the dataset  $\mathcal{D}$  in each iteration. When the size of the dataset is large, BGD may be slow due to the high computational complexity. In this case the method of *stochastic gradient descent (SGD)* can be used, which approximates the gradient based on only one of the  $N$  data points uniformly randomly chosen from the dataset in each iteration, i.e., the summation in Eq. (5.4) contains only one term corresponding to the chosen sample  $\mathbf{x}_n$ :

$$\mathbf{g}_n(\boldsymbol{\theta}) = \frac{d}{d\boldsymbol{\theta}} \frac{1}{2} r_n^2 = r_n \frac{dr_n}{d\boldsymbol{\theta}} \quad (5.8)$$

The expectation of  $\mathbf{g}_n$  based on one data point is the same as the true gradient (scaled by  $1/N$ ) based on all  $N$  data points in the dataset

$$E[\mathbf{g}_n(\boldsymbol{\theta})] = \sum_{n=1}^N P(\mathbf{x}_n) \mathbf{g}_n(\boldsymbol{\theta}) = \sum_{n=1}^N \frac{1}{N} \mathbf{g}_n(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \frac{dr_n}{d\boldsymbol{\theta}} r_n = \frac{1}{N} \mathbf{g}(\boldsymbol{\theta}) \quad (5.9)$$

As the data are typically noisy, the SGD gradient based on only one data point may be in a direction very different from the true gradient based on the mean of all  $N$  data points, and the resulting search path in the M-D space may be in a very jagged zig-zag shape, and the iteration may converge slowly. However, as the computational complexity in each iteration is much reduced, we can afford to carry out a large number of iterations with an overall search path generally coinciding with that of the BGD method.

Some trade-off between BGD and SGD can be made so that the gradient is

estimated based on more than one but less than all  $N$  data points in the dataset, the resulting method, called *mini-batch gradient descent*, can take advantage of both methods. The methods of stochastic and mini-batch gradient descent are widely used in machine learning, such as in the method of artificial neural networks, which can be treated as a nonlinear regression problem, considered in Chapter ??.

Alternatively, the optimal parameter  $\boldsymbol{\theta}^*$  of the nonlinear regression function  $f(\mathbf{x}, \boldsymbol{\theta})$  can also be found by the *Gauss-Newton method*, which directly solves the nonlinear over-determined equation  $\mathbf{r}(\boldsymbol{\theta}) = \mathbf{y} - \mathbf{f}(\mathbf{X}, \boldsymbol{\theta}) = \mathbf{0}$  in Eq. (5.2), based on the iterative Newton-Raphson method in Eq. (1.110) in Chapter 1:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \mathbf{J}_r^- \mathbf{r} = \boldsymbol{\theta}_n - (\mathbf{J}_r^T \mathbf{J}_r)^{-1} \mathbf{J}_r^T \mathbf{r} = \boldsymbol{\theta}_n + (\mathbf{J}_f^T \mathbf{J}_f)^{-1} \mathbf{J}_f^T \mathbf{r} \quad (5.10)$$

The last equality is due to the fact that  $\mathbf{J}_r = -\mathbf{J}_f$ . Here  $\mathbf{J}_f^T \mathbf{J}_f$  in the pseudo-inverse is actually an approximation of the Hessian  $\mathbf{H}_\varepsilon$  of the error function  $\varepsilon(\boldsymbol{\theta})$ , as shown in the example in Chapter 2, and  $-\mathbf{J}_f^T \mathbf{r} = \mathbf{g}_\varepsilon(\boldsymbol{\theta})$  is the gradient of the error function given in Eq. (5.4). So the iteration above is essentially:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \mathbf{J}_r^- \mathbf{r} \approx \boldsymbol{\theta}_n - \mathbf{H}_\varepsilon^{-1}(\boldsymbol{\theta}_n) \mathbf{g}_\varepsilon(\boldsymbol{\theta}_n) \quad (5.11)$$

same as the Newton-Raphson method based on the second order Hessian as well as the first order gradient. We therefore see that the Gauss-Newton method is actually for minimizing the sum-of-squares error is essentially the same as the Newton-Raphson method, with the only difference that the Hessian  $\mathbf{H}_\varepsilon \approx \mathbf{J}_f^T \mathbf{J}_f$  as the second order derivative of the error function  $\varepsilon(\boldsymbol{\theta})$  is approximated by the Jacobian  $\mathbf{J}_f \boldsymbol{\theta}$  as the first order derivative of the regression function  $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$ , making the implementation computationally easier.

In the special case of linear regression function  $\mathbf{f}(\mathbf{w}) = \mathbf{x}^T \mathbf{w}$ , we have:

$$\hat{\mathbf{y}} = \mathbf{f}(\mathbf{X}, \mathbf{w}) = \begin{bmatrix} f(\mathbf{x}_1, \mathbf{w}) \\ \vdots \\ f(\mathbf{x}_N, \mathbf{w}) \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \mathbf{w} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^T \mathbf{w} = \mathbf{X}^T \mathbf{w} \quad (5.12)$$

where  $\mathbf{w} = [w_0, w_1, \dots, w_d]^T$  and  $\mathbf{x}_i = [1, x_{i1}, \dots, x_{id}]^T$  are both augmented  $d+1$  dimensional vectors and  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ . The Taylor series is

$$f(\mathbf{X}, \mathbf{w} + \Delta \mathbf{w}) = \mathbf{X}^T (\mathbf{w} + \Delta \mathbf{w}) = \mathbf{X}^T \mathbf{w} + \mathbf{X}^T \Delta \mathbf{w} \quad (5.13)$$

and the Jacobian is

$$\mathbf{J}_f(\mathbf{w}) = \mathbf{X}^T \quad (5.14)$$

Now Eq. (5.10) becomes

$$\begin{aligned} \mathbf{w}_{n+1} &= \mathbf{w}_n + (\mathbf{X}^T)^- (\mathbf{y} - \mathbf{X}^T \mathbf{w}_n) = \mathbf{w}_n + (\mathbf{X} \mathbf{X}^T)^{-1} \mathbf{X} (\mathbf{y} - \mathbf{X}^T \mathbf{w}_n) \\ &= \mathbf{w}_n + (\mathbf{X} \mathbf{X}^T)^{-1} \mathbf{X} \mathbf{y} - (\mathbf{X} \mathbf{X}^T)^{-1} (\mathbf{X} \mathbf{X}^T) \mathbf{w}_n \\ &= (\mathbf{X} \mathbf{X}^T)^{-1} \mathbf{X} \mathbf{y} = (\mathbf{X}^T)^- \mathbf{y} \end{aligned} \quad (5.15)$$

i.e., the optimal parameter  $\mathbf{w}^*$  can be found in a single step without iteration, same as that in Eq. (4.19).

The Matlab code for the essential parts of the algorithm is listed below, where `data(:,1)` and `data(:,2)` are the observed data containing  $N$  samples  $x_1, \dots, x_N$  in the first column and their corresponding function values  $y_1, \dots, y_N$  in the second column. In the code segment below, the regression function (e.g., an exponential model) is symbolically defined, and then a function `NLregression` is called to estimate the function parameters:

```

sym x; % symbolic variables
a=sym('a',[3 1]); % symbolic parameters
f=@(x,a)a(1)*exp(a(2)*x)+a(3); % symbolic function
[a er]=NLregression(data(:,1),data(:,2),f,3); % call regression

```

The code below estimates the  $M$  parameters of the regression function based on Newton's method.

```

function [a er]=NLregression(x,y,f,M) % nonlinear regression
N=length(x); % number of data samples
a=sym('a',[M 1]); % M symbolic parameters
F=[];
for i=1:N
    F=[F; f(x(i),a)]; % evaluate function at N given data points
end
J=jacobian(F,a); % symbolic Jacobian
J=matlabFunction(J); % convert to a true function
F=matlabFunction(F);
a1=ones(M,1); % initial guesses for M parameters
n=0; % iteration index
da=1;
while da>tol % terminate if no progress made
    n=n+1;
    a0=a1; % update parameters
    J0=J(a0(1),a0(2)); % evaluate Jacobian based on a0
    Jinv=pinv(J0); % pseudo inverse of Jacobian
    r=y-F(a0(1),a0(2)); % residual
    a1=a0+Jinv*r; % Newton iteration
    er=norm(r); % residual error
    da=norm(a0-a1); % progress
end
end

```

where `tol` is a small value for tolerance, such as  $10^{-6}$ .

### Example 1



We first resolve the simple linear regression problem considered previously:

$$\mathbf{X} = [1 \ 2 \ 4], \quad \mathbf{y} = [1.6, 1.8, 3.2]^T$$

We have

$$\bar{x} = 2.33, \quad \bar{y} = 2.20, \quad \sigma_{xy}^2 = 0.87, \quad \sigma_x^2 = 1.56$$

and get

$$b = \frac{\sigma_{xy}^2}{\sigma_x^2} = 0.557, \quad a = \bar{y} - b\bar{x} = 0.9$$

which are the same as the results obtained previously by the linear regression pseudo inverse method.

Consider three sets of data (shown as the solid dots in the three columns in the plots below) are fitted by three different models (shown as the open circles in the plots):

1. Linear model:  $y = f_1(x) = ax + b$
2. quadratic model:  $y = f_2(x) = ax^2 + bx + c$
3. Exponential model:  $y = f_3(x) = a e^{bx} + c$

These are the parameters for the three models to fit each of the three datasets:

	Parameters :	a	b	c	error
Dataset 1	Model 1 :	30.337	51.652		7.4294
	Model 2 :	0.076	30.034	51.580	7.4223
	Model 3 :	5933.214	0.005	-5881.634	7.4221
Dataset 2	Model 1 :	24.519	20.089		21.0362
	Model 2 :	4.459	6.625	15.855	8.4559
	Model 3 :	29.791	0.346	-10.202	9.7402
Dataset 3	Model 1 :	32.608	22.008		53.0724
	Model 2 :	10.391	-9.089	12.140	28.3211
	Model 3 :	2.197	0.885	27.241	8.3393

We see that the first dataset can be fitted by all three models equally well with similar error (when  $a \approx 0$  in model 2, it is approximately a linear functions), the second dataset can be well fitted by either the second or the third model, while the third dataset can only be fitted by the third exponential model.

### Example 2

Based on the following three functions with four parameters  $a = 1$ ,  $b = 2$ ,  $c = 3$ ,  $d = 4$ , three sets of 2-D data points are generated (with some random noise added), shown in the figure below as the dots in the three rows.

$$\begin{aligned} f_1(\mathbf{x}) &= ax_1 + bx_2 - c \\ f_2(\mathbf{x}) &= \exp(ax_1 + bx_2) - cx_1x_2 + d \\ f_3(\mathbf{x}) &= ax_1^2 - bx_1x_2 + cx_2^2 + d \end{aligned} \tag{5.16}$$

Then each of three datasets are fitted by the three functions as models while the optimal parameters are found by the Gauss-Newton method.

	Parameters :	$a$	$b$	$c$	$d$	error
Dataset 1	Model 1 :	0.98	1.97	3.00		4.13
	Model 2 :	0.70	1.30	1.21	-4.42	6.87
	Model 3 :	-0.19	0.73	-0.12	-3.01	18.03
Dataset 2	Model 1 :	1.84	3.40	-6.09		14.70
	Model 2 :	0.99	2.01	2.97	3.98	3.75
	Model 3 :	0.37	1.13	2.66	4.87	32.45
Dataset 3	Model 1 :	-0.19	0.30	-5.34		16.82
	Model 2 :	-0.29	0.23	2.00	4.30	13.51
	Model 3 :	0.94	2.05	2.99	4.02	4.09

The *Levenberg-Marquardt algorithm (LMA)* or *damped least-squares method* is a method widely used to minimize a sum-of-squares error  $\varepsilon$  as in Eq. (5.3). The LMA is an interpolation or trade-off between the the gradient descent method based on the first order derivative of the error function to be minimized, and the Gauss-Newton method based on both the first and second order derivatives of the error function, both discussed above. As the LMA can take advantage of both methods and is therefore more robust. Even if the initial guess is far from the solution corresponding to the minimum of the objective function, the iteration can still converge toward the solution.

Specifically, the LMA modifies Eq. (5.10)

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \mathbf{J}_f^{-1}(\boldsymbol{\theta}_n)[\mathbf{y} - \mathbf{f}(\mathbf{X}, \boldsymbol{\theta}_n)] = \boldsymbol{\theta}_n + (\mathbf{J}_n^T \mathbf{J}_n)^{-1} \mathbf{J}_n^T [\mathbf{y} - \mathbf{f}(\mathbf{X}, \boldsymbol{\theta}_n)] \quad (5.17)$$

to include an extra term proportional to the identity matrix  $\mathbf{I}$ :

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + (\mathbf{J}_n^T \mathbf{J}_n + \lambda \mathbf{I})^{-1} \mathbf{J}_n^T [\mathbf{y} - \mathbf{f}(\mathbf{X}, \boldsymbol{\theta}_n)] \quad (5.18)$$

where  $\lambda \geq 0$  is the non-negative damping factor, which is to be adjusted at each iteration. We recognize that this is actually similar to Eq. (4.46) for ridge regression.

In each iterative step, if  $\lambda$  is small and the error function  $\varepsilon(\boldsymbol{\theta}_n)$  decreases quickly, indicating the Gauss-Newton is effective in the sense that the error function is well modeled as a quadratic function by the first and second order terms of the Taylor series, we will keep  $\lambda$  small. On the other hand, if  $\varepsilon$  decreases slowly, indicating the error function may not be well modeled as a quadratic function, due possibly to the fact that the current  $\boldsymbol{\theta}_n$  is far away from the minimum, then we let  $\lambda$  take a large value, so that the term  $\mathbf{J}_n^T \mathbf{J}_n$  becomes insignificant, and Eq. (5.18) approaches

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \lambda \mathbf{J}_n^T (\mathbf{y} - \mathbf{f}(\boldsymbol{\theta})) = \boldsymbol{\theta}_n - \alpha \mathbf{g}_\varepsilon(\boldsymbol{\theta}) \quad (5.19)$$

Now  $\boldsymbol{\theta}_n$  is updated along the negative direction of the gradient in Eq. (5.4):

$$\mathbf{g}_\varepsilon(\boldsymbol{\theta}) \propto \mathbf{J}^T (\mathbf{f}(\boldsymbol{\theta}) - \mathbf{y}) \quad (5.20)$$

with some step size  $\alpha$ .

By adjusting the parameter  $\lambda$ , the LMA can switch between either of the two methods for the same purpose of minimizing  $\varepsilon(\boldsymbol{\theta})$ . When the value of  $\lambda$  is small, the iteration is dominated by the Gauss-Newton method that approximate the error function by a quadratic function based on the second order derivatives. If the error reduces slowly, indicating the error function cannot be approximated by a quadratic function, a greater value of  $\lambda$  is used to switch to the gradient descent method.

## 5.2 Parameter Estimation by Natural Gradient Descent

In Section 4.1, we considered the MLE and MAP methods for estimating the parameter  $\boldsymbol{\theta}$  of a probability model  $p(\mathbf{x}|\boldsymbol{\theta})$  by maximizing the likelihood function  $L(\boldsymbol{\theta}|\mathbf{x})$  or the posterior  $p(\boldsymbol{\theta}|\mathbf{x})$ . In the previous section we also considered how the gradient method is used to estimate the model parameter  $\boldsymbol{\theta}$  by minimizing the squared error. Obviously we can also use the gradient method in MLE to maximize the likelihood. However, unlike the parameter  $\boldsymbol{\theta} = [\theta_1, \dots, \theta_m]^T$  which is treated as a vector in a Euclidean space spanned by a set of orthonormal basis vectors, the family of distributions  $p(\mathbf{x}|\boldsymbol{\theta})$  corresponding to all different parameters  $\boldsymbol{\theta}$  resides in a distribution space, an  $m$ -dimensional Riemannian manifold or space, in which no orthogonal basis is defined and methods based on concepts such as inner product and Euclidean distance are no longer valid.

In this case, the difference between two distributions  $p(\mathbf{x}|\boldsymbol{\theta})$  and  $p(\mathbf{x}|\boldsymbol{\theta}')$  in the space can be measured by their KL-divergence  $KL(p(\mathbf{x}|\boldsymbol{\theta})||p_p(\mathbf{x}|\boldsymbol{\theta} + \Delta\boldsymbol{\theta}))$ , and the gradient method can still be used to find iteratively the optimal parameter that maximize the likelihood  $L(\boldsymbol{\theta})$ , similar to the steps that lead to Eq. (2.177). Specifically, we desired to find the optimal  $\Delta\boldsymbol{\theta}$  that maximally increases the likelihood, which is also subject to the constraint  $\|\Delta\boldsymbol{\theta}\|^2 < C$ :

$$\Delta\boldsymbol{\theta}^* = \arg \max_{\|\Delta\boldsymbol{\theta}\|^2 < C} L(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \quad (5.21)$$

where  $\|\Delta\boldsymbol{\theta}\|^2$  is defined as Eq. (3.253):

$$\|\Delta\boldsymbol{\theta}\|^2 = \frac{1}{2}\Delta\boldsymbol{\theta}^T \mathbf{F}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} = KL(p(\mathbf{x}|\boldsymbol{\theta})||p_p(\mathbf{x}|\boldsymbol{\theta} + \Delta\boldsymbol{\theta})) < C \quad (5.22)$$

To solve this constrained optimization problem we set the derivative of the Lagrangian to zero

$$\frac{d}{d(\Delta\boldsymbol{\theta})} \left[ L(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) + \mu \left( \frac{1}{2}\Delta\boldsymbol{\theta}^T \mathbf{F}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} - C \right) \right] = \mathbf{g}_L(\boldsymbol{\theta}) + \mu \mathbf{F}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} = \mathbf{0} \quad (5.23)$$

Solving this equation we get the optimal increment:

$$\Delta\boldsymbol{\theta}^* = -\frac{1}{\mu} \mathbf{F}^{-1}(\boldsymbol{\theta})\mathbf{g}_L(\boldsymbol{\theta}) \quad (5.24)$$

and the optimal parameter can be found iteratively using this method of *natural gradient descent* (Eq. (3.3.4)):

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \Delta\boldsymbol{\theta}_n = \boldsymbol{\theta}_n + \lambda\mathbf{F}^{-1}\mathbf{g}_L(\boldsymbol{\theta}_n) = \boldsymbol{\theta}_n + \lambda\tilde{\mathbf{g}}_L(\boldsymbol{\theta}_n) \quad (5.25)$$

where  $\lambda = 1/\mu$  is the step size, and  $\tilde{\mathbf{g}}_L(\boldsymbol{\theta}_n)$  is the *natural gradient*:

$$\tilde{\mathbf{g}}_L(\boldsymbol{\theta}_n) = \mathbf{F}^{-1}\mathbf{g}_L(\boldsymbol{\theta}_n) \quad (5.26)$$

Comparing this iteration with that in Eq. (2.16), we see that the natural gradient method is similar to the Newton-Raphson method in the sense that it is based on not only the gradient  $\mathbf{g}_J(\boldsymbol{\theta})$  for the local first order variation of the likelihood function  $L(\boldsymbol{\theta})$  in the distribution space, but also the Fisher information  $\mathbf{F}(\boldsymbol{\theta})$  for the more global second order curvature in the space, by which the gradient direction is modified so that the iterative search process in the space can be carried out more efficiently.

# 6 Logistic and Softmax Regression

---

## 6.1 Logistic Regression and Binary Classification

All previously discussed regression methods can be considered as supervised binary classifiers, when the regression function  $f(\mathbf{x}, \boldsymbol{\theta})$  is thresholded by some constant  $C$ . Without loss of generality, we will always assume  $C = 0$  in the following. Once the model parameter  $\boldsymbol{\theta}$  is obtained based on the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ , every point  $\mathbf{x}$  in the  $d$ -dimensional feature space can be classified into either of two classes  $C_-$  and  $C_+$ :

$$\text{if } f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \begin{cases} < C \\ > C \end{cases} \quad \text{then } \mathbf{x} \in \begin{cases} C_- \\ C_+ \end{cases} \quad (6.1)$$

When the regression function is thresholded by  $C = 0$ , it becomes an equation  $f(\mathbf{x}) = 0$  representing a hypersurface in the  $d$ -dimensional space (a point if  $d = 1$ , a curve if  $d = 2$ , a surface or hypersurface if  $d \geq 3$ ), by which the space is partitioned into two regions. In this case,  $f(\mathbf{x})$  is called the *decision function* and the surface *decision boundary*. Now any point in the space is classified into either of the two classes, depending on whether it is on the positive side of the decision boundary ( $f(\mathbf{x}) > 0$ ) or the negative side ( $f(\mathbf{x}) < 0$ ).

This simple binary classifier suffers from the drawback that the distance between a point  $\mathbf{x}$  and the decision boundary  $f(\mathbf{x}) = 0$  is not taken into consideration, as all points on the same side of the boundary, regardless how far from the boundary, are equally classified into the same class. On the other hand, one would naturally feel more confident about such a classification for a point far away from the boundary, than a point very close to it.

This problem can be addressed by the method of *logistic regression*, which is similarly trained based on the training set with each sample  $\mathbf{x}_n$  labeled by a binary value  $y_n \in \{0, 1\}$  for either of the two classes  $C_-$  and  $C_+$ . While the logistic regression is based on the linear regression function  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ , which is not hard-thresholded by the unit step function  $u(x)$  to become either 0 or 1; instead, it is soft-thresholded by a sigmoid function  $\sigma(f(\mathbf{x})) = \sigma(\mathbf{w}^T \mathbf{x})$  to become a real value between 0 and 1, interpreted as the probability  $p(\mathbf{x} \in C_+)$  for  $\mathbf{x}$  to belong to  $C_+$ , and  $p(\mathbf{x} \in C_-) = 1 - p(\mathbf{x} \in C_+)$  is the probability for  $\mathbf{x}$  to belong to  $C_-$ .

Similar to linear and nonlinear regression methods considered before, the goal

of logistic regression is also to find the optimal model parameter  $\boldsymbol{\theta}$  for the regression function  $f(\mathbf{x}, \boldsymbol{\theta})$  to fit the training set optimally. But different from linear and nonlinear regression based on the least squares (LS) method that minimizes the squared error in Eq. (5.3), logistic regression is based on maximum likelihood (ML) estimation, which maximizes the *likelihood* of the model parameter, here  $\boldsymbol{\theta} = \mathbf{w}$ , as discussed below.

The sigmoid function can be either the error function (cumulative Gaussian, erf):

$$\phi(x) = \int_{-\infty}^x \mathcal{N}(u|0, 1) du = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}u^2\right) du \quad (6.2)$$

or the logistic function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} = \begin{cases} 1 & x \rightarrow \infty \\ 0.5 & x = 0 \\ 0 & x \rightarrow -\infty \end{cases} \quad (6.3)$$

In the following, we will only consider the logistic function for consistency and simplicity. Here is a set of properties of the logistic function:

$$\begin{aligned} \sigma(-x) &= \frac{1}{1 + e^x} = 1 - \frac{1}{1 + e^{-x}} = 1 - \sigma(x) \\ \frac{d}{dx} \sigma(x) &= \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)\sigma(-x) \\ x &= \ln\left(\frac{\sigma(x)}{1 - \sigma(x)}\right), \quad e^x = \frac{\sigma(x)}{1 - \sigma(x)} \end{aligned} \quad (6.4)$$

Moreover, a parameter  $w$  can be included in the logistic function to control its softness when used to threshold the variable  $x$  as shown below:

$$\sigma(wx) = \begin{cases} 1/2 & w = 0 \\ 1/(1 + e^{-wx}) & w > 0 \\ u(x) & w \rightarrow \infty \end{cases} \quad (6.5)$$

We see that the argument  $x$  is thresholded by the logistic function with variable softness controlled by parameter  $w$ , between the two extreme cases of  $\sigma(wx) = 1/2$  (no thresholding) when  $w = 0$ , and  $\sigma(wx) = u(x)$  when  $w \rightarrow \infty$  (hard thresholding).

Now the function value  $f(\mathbf{x}) \in (-\infty, \infty)$  is mapped to  $\sigma(f(\mathbf{x})) \in (0, 1)$ , treated as the probability  $p(\mathbf{x} \in C_+)$  for  $\mathbf{x} \in C_+$ . Specifically, consider the following cases:

- If  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \gg 0$ , i.e.,  $\mathbf{x}$  is far away from the plane  $f(\mathbf{x}) = 0$  on the positive side, then the probability  $p(\mathbf{x} \in C_+) = \sigma(f(\mathbf{x}))$  is close to 1;
- If  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \ll 0$ , i.e.,  $\mathbf{x}$  is far away from the plane  $f(\mathbf{x}) = 0$  on the negative side, then the probability  $p(\mathbf{x} \in C_+) = \sigma(f(\mathbf{x}))$  is close to 0, but  $p(\mathbf{x} \in C_-) = 1 - p(\mathbf{x} \in C_+)$  for  $\mathbf{x} \in C_-$  is close to 1;

- If  $f(\mathbf{x}) \approx 0$ , i.e.,  $\mathbf{x}$  is close to the plane  $f(\mathbf{x}) = 0$ , then  $p(\mathbf{x} \in C_+) = \sigma(f(\mathbf{x})) \approx 0.5$ , and  $p(\mathbf{x} \in C_-) = 1 - \sigma(f(\mathbf{x})) \approx 0.5$ , i.e., the probability for  $\mathbf{x}$  to belong to either  $C_+$  or  $C_-$  is about the same.

The value of this logistic function is interpreted as the conditional probability of output  $\hat{y} = 1$  given data point  $\mathbf{x}$  and model parameter  $\mathbf{w}$ :

$$p(\hat{y} = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} \quad (6.6)$$

and the conditional probability of  $\hat{y} = 0$  is:

$$p(\hat{y} = 0|\mathbf{x}, \mathbf{w}) = 1 - p(\hat{y} = 1|\mathbf{x}, \mathbf{w}) = 1 - \sigma(\mathbf{w}^T \mathbf{x}) = \sigma(-\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}^T \mathbf{x}}} \quad (6.7)$$

Combining these two cases, we get the conditional probability of  $\hat{y} = y_n$  (either 1 or 0), as the labeling of a sample  $\mathbf{x}_n$  in the training set:

$$p(\hat{y} = y_n|\mathbf{x}_n, \mathbf{w}) = p(\hat{y} = 1|\mathbf{x}_n, \mathbf{w})^{y_n} p(\hat{y} = 0|\mathbf{x}_n, \mathbf{w})^{1-y_n} = \begin{cases} p(\hat{y} = 1|\mathbf{x}_n, \mathbf{w}) & \text{if } y_n = 1 \\ p(\hat{y} = 0|\mathbf{x}_n, \mathbf{w}) & \text{if } y_n = 0 \end{cases} \quad (6.8)$$

We further get the likelihood function of  $\mathbf{w}$  given all  $N$  samples in  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$  (assumed i.i.d.):

$$\begin{aligned} L(\mathbf{w}|\mathcal{D}) &= p(\mathcal{D}|\mathbf{w}) = p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^N p(\hat{y} = y_n|\mathbf{x}_n, \mathbf{w}) \\ &= \prod_{n=1}^N p(\hat{y} = 1|\mathbf{x}_n)^{y_n} p(\hat{y} = 0|\mathbf{x}_n)^{1-y_n} = \prod_{n=1}^N \sigma(\mathbf{w}^T \mathbf{x}_n)^{y_n} (1 - \sigma(\mathbf{w}^T \mathbf{x}_n))^{1-y_n} \end{aligned} \quad (6.9)$$

Now the optimal model parameter  $\mathbf{w}$  can be defined as the one that maximizes the likelihood function  $L(\mathbf{w}|\mathcal{D})$ , or, equivalently, that minimizes the negative log likelihood as the objective function:

$$J(\mathbf{w}) = -l(\mathbf{w}|\mathcal{D}) = -\log L(\mathbf{w}|\mathcal{D}) = -\sum_{n=1}^N [y_n \log \sigma(\mathbf{w}^T \mathbf{x}_n) + (1 - y_n) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_n))] \quad (6.10)$$

This is the method of maximum likelihood estimation (MLE), and it produces the most probable estimation of certain parameter of a model based on the given data.

The optimal parameter  $\mathbf{w}$  can also be found by the method of maximum a posteriori (MAP), which maximizes the posterior probability:

$$p(\mathbf{w}|\mathcal{D}) = p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})} \propto p(\mathbf{y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w}) \quad (6.11)$$

where  $p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = L(\mathbf{w}|\mathcal{D})$  is the likelihood found above,  $p(\mathbf{w})$  is the prior probability of  $\mathbf{w}$  without observing the training data  $\mathcal{D}$ , and the denominator independent of the variable  $\mathbf{w}$  is dropped as a constant independent of  $\mathbf{w}$ .

Based on the assumption that all components of  $\mathbf{w}$  are independent of each

other and they have zero mean and the same variance, we let the prior probability be:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}, \mathbf{0}, \sigma^2 \mathbf{I}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{\|\mathbf{w}\|^2}{2\sigma^2}\right) \propto \exp\left(-\frac{\|\mathbf{w}\|^2}{2\sigma^2}\right) \quad (6.12)$$

where the normalizing factor  $1/(2\pi\sigma^2)^{N/2}$  is independent of  $\mathbf{w}$  and is therefore dropped. The assumed zero mean is due to the desire that the values of all weights in  $\mathbf{w}$  should be around zero instead of large values on either positive or negative side, to avoid overfitting, as discussed below.

Now the posterior can be written as

$$\begin{aligned} p(\mathbf{w}|\mathcal{D}) &\propto p(\mathbf{y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w}|\mathbf{X}) = \prod_{n=1}^N p(\hat{y}_n = y_n | \mathbf{x}_n, \mathbf{w}) \mathcal{N}(\mathbf{0}, \Sigma_w) \\ &\propto \prod_{n=1}^N \sigma(\mathbf{w}^T \mathbf{x}_n)^{y_n} (1 - \sigma(\mathbf{w}^T \mathbf{x}_n))^{1-y_n} \exp\left(-\frac{\|\mathbf{w}\|^2}{2\sigma^2}\right) \end{aligned} \quad (6.13)$$

We can alternatively find the optimal model parameter  $\mathbf{w}$  that maximizes this posterior  $p(\mathbf{w}|\mathcal{D})$  by the method of *maximum a posteriori (MAP)* estimation. To do so, we minimize the negative log posterior as the objective function:

$$\begin{aligned} J(\mathbf{w}) &= -\log p(\mathbf{w}|\mathcal{D}) = -\log p(\mathbf{y}|\mathbf{X}, \mathbf{w}) - \left(-\frac{\|\mathbf{w}\|^2}{2\sigma^2}\right) \\ &= -\sum_{n=1}^N y_n \log \sigma(\mathbf{w}^T \mathbf{x}_n) + (1 - y_n) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_n)) + \frac{\|\mathbf{w}\|^2}{2\sigma^2} \end{aligned} \quad (6.14)$$

We can define  $\lambda = 1/\sigma^2$  as a hyperparameter in the second term  $\lambda\|\mathbf{w}\|^2$  treated as a penalty term to discourage large values of  $\mathbf{w}$ , similar to Eq. (4.44) for ridge regression. A larger value of  $\lambda$  will result in smaller values for  $\mathbf{w}$ , and the logistic function will be more smooth for softer thresholding, and the classification is less affected by possible noisy outliers close to the decision boundary. In other words, by adjusting  $\lambda$ , we can make a proper tradeoff between overfitting and underfitting.

In particular, if  $\sigma \rightarrow \infty$  and  $\lambda \rightarrow 0$ , then  $\exp(-\lambda\|\mathbf{w}\|^2/2) \Rightarrow 1$ , i.e., the prior  $p(\mathbf{w})$  approaches a uniform distribution (all values of  $\mathbf{w}$  are equally likely), then the penalty term can be removed, and Eq. (6.14) becomes the same as Eq. (6.10), i.e., the MAP solution that maximizes the posterior  $p(\mathbf{w}|\mathcal{D})$  becomes the same as the MLE solution that maximizes the likelihood  $L(\mathbf{w}|\mathcal{D})$ .

Now we can find the optimal  $\mathbf{w}$  that minimizes the objective function  $J(\mathbf{w})$  in either Eq. (6.10) for the MLE method, or Eq. (6.14) for the MAP method. Here we consider the latter case which is more general (with an extra term



$\|\mathbf{w}\|^2/2\sigma^2$ ). We first find the gradient of objective function:

$$\begin{aligned}
 \mathbf{g}_J(\mathbf{w}) &= \frac{d}{d\mathbf{w}} J(\mathbf{w}) = \frac{d}{d\mathbf{w}} \left( -\log p(\mathbf{y}|\mathbf{X}, \mathbf{w}) + \frac{\|\mathbf{w}\|^2}{2\sigma^2} \right) \\
 &= -\sum_{n=1}^N \frac{d}{d\mathbf{w}} [y_n \log \sigma(\mathbf{w}^T \mathbf{x}_n) + (1 - y_n) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_n))] + \lambda \mathbf{w} \\
 &= -\sum_{n=1}^N \left[ \frac{y_n}{\sigma(\mathbf{w}^T \mathbf{x}_n)} - \frac{1 - y_n}{1 - \sigma(\mathbf{w}^T \mathbf{x}_n)} \right] \sigma(\mathbf{w}^T \mathbf{x}_n)(1 - \sigma(\mathbf{w}^T \mathbf{x}_n)) \mathbf{x}_n + \lambda \mathbf{w} \\
 &= \sum_{n=1}^N [\sigma(\mathbf{w}^T \mathbf{x}_n) - y_n] \mathbf{x}_n + \lambda \mathbf{w} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \begin{bmatrix} \sigma(\mathbf{w}^T \mathbf{x}_1) - y_1 \\ \vdots \\ \sigma(\mathbf{w}^T \mathbf{x}_N) - y_N \end{bmatrix} + \lambda \mathbf{w} \\
 &= \mathbf{X}\mathbf{r} + \lambda \mathbf{w} \tag{6.15}
 \end{aligned}$$

where we have defined a residual vector

$$\mathbf{r} = \mathbf{r}(\mathbf{w}) = \begin{bmatrix} r_1(\mathbf{w}) \\ \vdots \\ r_N(\mathbf{w}) \end{bmatrix} = \begin{bmatrix} \sigma(\mathbf{w}^T \mathbf{x}_1) - y_1 \\ \vdots \\ \sigma(\mathbf{w}^T \mathbf{x}_N) - y_N \end{bmatrix} \tag{6.16}$$

containing all  $N$  residuals  $r_n = \sigma(\mathbf{w}^T \mathbf{x}_n) - y_n$  ( $n = 1, \dots, N$ ), the difference between the model prediction  $0 < \sigma(\mathbf{w}^T \mathbf{x}_n) < 1$  and the ground truth labeling  $y_n \in \{0, 1\}$ . Based on  $\mathbf{g}_J(\mathbf{w})$  we can find the optimal  $\mathbf{w}$  that minimizes this objective function iteratively by the gradient descent method. We note that the gradient in Eq. (6.15) is based on all  $N$  training samples in the training set. If  $N$  is large, then the stochastic gradient descent method can be considered based on only one of the  $N$  samples in each iteration.

Below is a segment of Matlab code for estimating  $\mathbf{w}$  by the gradient descent method. Here  $\mathbf{X}$  contains the  $N$  samples  $\mathbf{x}_1, \dots, \mathbf{x}_N$  and  $\mathbf{y}$  contains the corresponding a binary labelings.

The following code segment estimates weight vector  $\mathbf{w}$  by the gradient descent method:

```

X=[ones(1,n); X];           % augmented X with x_0=1
w=ones(m+1,1);           % initialize weight vector
g=X*(sgm(w,X)-y)';       % gradient of log posterior
tol=10^(-6);             % tolerance
delta=0.1;               % step size
while norm(g)>tol         % terminate when gradient is small enough
    w=w-delta*g;         % update weight vector by gradient descent
    g=X*(sgm(w,X)-y)';  % update gradient
end

```

where

```
function s=sgm(w,X)
```

```

s=1./(1+exp(-w'*X));
end

```

Alternatively, this minimization problem for finding optimal  $\mathbf{w}$  can also be solved by the Newton's method based on the second order Hessian matrix  $\mathbf{H}_J(\mathbf{w})$  as well as the first order gradient  $\mathbf{g}_J(\mathbf{w})$ :

$$\begin{aligned}\mathbf{H}_J(\mathbf{w}) &= \frac{d}{d\mathbf{w}} \mathbf{g}_J(\mathbf{w}) = \frac{d}{d\mathbf{w}} (\mathbf{X}\mathbf{r}(\mathbf{w}) + \lambda \|\mathbf{w}\|) = \mathbf{X} \frac{d}{d\mathbf{w}} \mathbf{r}(\mathbf{w}) + \lambda \\ &= \mathbf{X} \frac{d}{d\mathbf{w}} \begin{bmatrix} r_1(\mathbf{w}) \\ \vdots \\ r_N(\mathbf{w}) \end{bmatrix} + \lambda = \mathbf{X}\mathbf{J}_r(\mathbf{w}) + \lambda\end{aligned}\quad (6.17)$$

where  $\mathbf{J}_r(\mathbf{w}) = d\mathbf{r}(\mathbf{w})/d\mathbf{w}$  is the  $N \times (d+1)$  Jacobian matrix of the vector function  $\mathbf{r}(\mathbf{w})$ , whose  $n$ th row is

$$\frac{dr_n(\mathbf{w})}{d\mathbf{w}} = \frac{d}{d\mathbf{w}} [\sigma(\mathbf{w}^T \mathbf{x}_n) - y_n] = \sigma(\mathbf{w}^T \mathbf{x}_n) [1 - \sigma(\mathbf{w}^T \mathbf{x}_n)] \mathbf{x}_n = z_n \mathbf{x}_n \quad (6.18)$$

where we have defined  $z_n = \sigma(\mathbf{w}^T \mathbf{x}_n)(1 - \sigma(\mathbf{w}^T \mathbf{x}_n))$ , which is always greater than zero as  $0 < \sigma(\mathbf{w}^T \mathbf{x}_n) < 1$ . Now the Jacobian can be written as

$$\mathbf{J}_r(\mathbf{w}) = \begin{bmatrix} dr_1(\mathbf{w})/d\mathbf{w} \\ \vdots \\ dr_N(\mathbf{w})/d\mathbf{w} \end{bmatrix} = \begin{bmatrix} z_1 \mathbf{x}_1 \\ \vdots \\ z_N \mathbf{x}_N \end{bmatrix} = \begin{bmatrix} z_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & z_N \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \mathbf{Z}\mathbf{X}^T \quad (6.19)$$

where  $\mathbf{Z} = \text{diag}(z_1, \dots, z_N)$  is a positive definite diagonal matrix (as  $z_n > 0$ ). Substituting  $\mathbf{J}_r$  back into the expression for  $\mathbf{H}_J(\mathbf{w})$  above, we get

$$\mathbf{H}_J = \mathbf{X}\mathbf{J}_r + \lambda = \mathbf{X}\mathbf{Z}\mathbf{X}^T + \lambda \quad (6.20)$$

Since  $\mathbf{Z}$  is positive definite, the following quadratic function is also positive, i.e., for any  $\mathbf{y}$ , we have

$$\mathbf{y}^T \mathbf{H}_J \mathbf{y} = \mathbf{y}^T \mathbf{X}\mathbf{Z}\mathbf{X}^T \mathbf{y} + \lambda \mathbf{y}^T \mathbf{y} = (\mathbf{X}^T \mathbf{y})^T \mathbf{Z} (\mathbf{X}^T \mathbf{y}) + \lambda \|\mathbf{y}\|^2 > 0 \quad (6.21)$$

indicating  $\mathbf{H}_J$  is also positive definite, and the objective function  $J(\mathbf{w})$  when approximated by the first three terms of its Taylor series has a global minimum.

Having found  $\mathbf{H}_J(\mathbf{w})$  as well as  $\mathbf{g}_J = J'(\mathbf{w})$  of the objective function  $J(\mathbf{w})$ , we can solve the minimization problem also by the Newton method:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \mathbf{H}_J(\mathbf{w}_n)^{-1} \mathbf{g}_J(\mathbf{w}_n) = \mathbf{w}_n - (\mathbf{X}\mathbf{Z}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{r} \quad (6.22)$$

The following code segment estimates weight vector  $\mathbf{w}$  by Newton method:

```

X=[ones(1,n); X];           % augmented X with x_0=1
w=ones(m+1,1);             % initialize weight vector
g=X*(sgm(w,X)-y)';        % gradient of log posterior
H=X*diag(s.*(1-s))*X';    % Hessian of log posterior
tol=10^(-6)

```

```

while norm(g)>tol           % terminate when gradient is small enough
    w=w-inv(H)*g;          % update weight vector by Newton-Raphson
    s=sgm(w,X);           % update sigma
    g=X*(s-y)';           % update gradient
    H=X*diag(s.*(1-s))*X'; % update Hessian
end

```

Once the model parameter  $\mathbf{w}$  is available, a given test sample  $\mathbf{x}_*$  of unknown class can be classified into either of the two classes  $C_1$  and  $C_0$ , based on  $p(y_* = 1|\mathbf{x}_*)$ :

$$\text{if } p(y_* = 1|\mathbf{x}_*) = \sigma(\mathbf{w}^T \mathbf{x}_*) \begin{cases} > 0.5 \\ < 0.5 \end{cases}, \quad \text{then } \mathbf{x}_* \in \begin{cases} C_1 \\ C_0 \end{cases} \quad (6.23)$$

In summary, we make a comparison between the linear and logistic regression methods both can be used for binary classification. In linear regression, the function value  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  is hard-thresholded by the unit step function, so that any data point  $\mathbf{x}$  is classified to class  $C_+$  if  $\mathbf{w}^T \mathbf{x} > 0$ , or class  $C_-$  if  $\mathbf{w}^T \mathbf{x} < 0$  in a deterministic fashion (with probability  $p(\mathbf{x} \in C_+) \in \{0, 1\}$ ). On the other hand, in logistic regression, the linear function  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  is soft-thresholded by the logistic function to become  $p(\mathbf{x} \in C_+) = \sigma(f(\mathbf{x})) \in (0, 1)$ , treated as the probability for  $\mathbf{x}$  to belong to class  $C_+$ . Although the same linear model  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  is used in both cases, the two methods obtain the model parameter  $\mathbf{w}$  by minimizing very different objective functions (Eqs. (4.17) and (6.14)).

## 6.2 Softmax Regression for Multiclass Classification

In a multiclass classification problem, an unlabeled data point  $\mathbf{x}$  is to be classified into one of  $K > 2$  classes  $\{C_1, \dots, C_K\}$ , based on the training set  $\mathcal{D} = \{(\mathbf{x}_n, y_n), n = 1, \dots, N\}$ , where  $y_n \in \{1, \dots, K\}$  is an integer indicating  $\mathbf{x}_n \in C_{y_n}$ .

Any binary classifier, such as the logistic regression considered above, can be used to solve such a multiclass classification problem in either of the following two ways:

- One-Vs-One (1V1): construct  $K(K-1)/2$  binary classifiers for every pair of the  $K$  classes based on the training set  $\{(C_i, C_j) \mid i = 2, \dots, K, j = 1, \dots, i-1\}$ , then an unlabeled sample  $\mathbf{x}$  is classified to one of the  $K$  classes with maximum votes out of this many binary classifications.
- One-Vs-Rest (1VR). Convert a  $K$ -class problem into  $K$  binary problems by regrouping the  $K$  classes into two classes  $C_+ = C_i$  and  $C_- = \{C_j \mid j = 1, \dots, K, j \neq i\}$ , and get the corresponding discriminant function  $f(\mathbf{x})$  of the binary problem:

$$\text{if } f_i(\mathbf{x}) \begin{cases} > 0 \\ < 0 \end{cases}, \quad \text{then } \begin{cases} \mathbf{x} \in C_+ = C_i \\ \mathbf{x} \in C_- \end{cases} \quad (6.24)$$

which represents quantitatively how much a given  $\mathbf{x}$  belongs to class  $C_+ = C_i$ , instead of  $C_-$  containing the rest  $K - 1$  classes. This process is repeated  $K$  times for all  $i = 1, \dots, K$  to get

$$f_k(\mathbf{x}) = \max\{f_1(\mathbf{x}), \dots, f_K(\mathbf{x})\}, \quad (k = 1, \dots, K) \quad (6.25)$$

which indicates  $\mathbf{x}$  belongs to class  $C_k$ .

Alternatively, a multiclass problem with  $K > 2$  can also be solved by *multinomial logistic* or *softmax regression*, which can be considered as a generalized version of the logistic regression method, based on the *softmax function* of  $K$  variables  $z_1, \dots, z_K$ :

$$s_k = s(z_k) = \frac{e^{z_k}}{\sum_{l=1}^K e^{z_l}}, \quad (k = 1, \dots, K) \quad (6.26)$$

with the following properties:

$$0 < s_k < 1, \quad \sum_{k=1}^K s_k = 1 \quad (6.27)$$

and

$$\begin{aligned} \frac{\partial s(z_i)}{\partial z_j} &= \frac{\delta_{ij} e^{z_i} \sum_{l=1}^K e^{z_l} - e^{z_i} e^{z_j}}{\left(\sum_{l=1}^K e^{z_l}\right)^2} \\ &= \delta_{ij} s(z_i) - s(z_i) s(z_j) = s(z_i) [\delta_{ij} - s(z_j)] \end{aligned} \quad (6.28)$$

where  $\delta_{ij}$  is the Dirac delta function which is 1 if  $i = j$ , but 0 otherwise.

Similar to how the logistic function is used to model the conditional probability  $p(\hat{y} = 1 | \mathbf{x}, \mathbf{w})$  for a given data point  $\mathbf{x}$  to belong to class  $C_+$  based on model parameter  $\mathbf{w}$  in Eq. (6.6), here the softmax function  $s_k$  defined above is used to model the conditional probability for  $\mathbf{x}$  to belong to class  $C_k$  based on some model parameter  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]$ :

$$p(\hat{y} = k | \mathbf{x}, \mathbf{W}) = s_k = s(\mathbf{w}_k^T \mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{l=1}^K e^{\mathbf{w}_l^T \mathbf{x}}}, \quad (k = 1, \dots, K) \quad (6.29)$$

where  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]$  is composed of  $K$  weight vectors each associated with one of the  $K$  classes  $\{C_1, \dots, C_K\}$ , to be determined in the training process based on training set  $\mathcal{D}$ . Same as in logistic regression, here both  $\mathbf{x}$  and  $\mathbf{w}$  are augmented  $d + 1$  dimensional vectors.

We note that the inner product  $\mathbf{w}^T \mathbf{x} = \|\mathbf{w}\| \|\mathbf{x}\| \cos \theta$  of vectors  $\mathbf{w}$  and  $\mathbf{x}$  is inversely related to the angle  $\theta$  between the two vectors, i.e., when compared with other classes, a data sample  $\mathbf{x}$  has a smaller angular distance to  $\mathbf{w}_k$ , then it has a larger inner product  $\mathbf{w}_k^T \mathbf{x}$  and thereby greater probability  $p(\hat{y} = k | \mathbf{x})$  to belong to class  $C_k$ . We therefore see that weight vectors  $\mathbf{w}_1, \dots, \mathbf{w}_K$  as the parameters of the softmax regression model actually represent the angular directions of the corresponding classes in the feature space.

Specially, when  $K = 2$ , the above becomes logistic functions:

$$\begin{aligned} s_1 &= \frac{e^{\mathbf{w}_1^T \mathbf{x}}}{e^{\mathbf{w}_0^T \mathbf{x}} + e^{\mathbf{w}_1^T \mathbf{x}}} = \frac{1}{1 + e^{-(\mathbf{w}_1 - \mathbf{w}_0)^T \mathbf{x}}} = \sigma(\mathbf{w}^T \mathbf{x}) \\ s_0 &= \frac{e^{\mathbf{w}_0^T \mathbf{x}}}{e^{\mathbf{w}_0^T \mathbf{x}} + e^{\mathbf{w}_1^T \mathbf{x}}} = \frac{1}{1 + e^{(\mathbf{w}_1 - \mathbf{w}_0)^T \mathbf{x}}} = \sigma(-\mathbf{w}^T \mathbf{x}) = 1 - \sigma(\mathbf{w}^T \mathbf{x}) \end{aligned} \quad (6.30)$$

where we have defined  $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_0$ , as the vector between the two classes of  $C_+$  and  $C_-$ , which is the normal direction of a decision plane separating the two classes. Any point  $\mathbf{x}$  of unknown class can be therefore classified into either of the two classes depending on whether its projection  $\mathbf{w}^T \mathbf{x}$  onto  $\mathbf{w}$  is positive or negative, i.e., on which side of the decision plane it is located.

For mathematical convenience, we also label each sample  $\mathbf{x}_n$  in the training set by a binary vector  $\mathbf{y}_n = [y_{n1}, \dots, y_{nK}]^T$ , in addition to its integer labeling  $y_n \in \{1, \dots, K\}$ . If  $y_n = k$  indicating  $\mathbf{x}_n \in C_k$ , then the  $k$ th component of  $\mathbf{y}_n$  is  $y_{nk} = 1$ , while other components are zero  $y_{nl} = 0$  for all  $l \neq k$ . Note that all components of  $\mathbf{y}_n$  add up to 1. The  $N$  training samples in  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  are also labeled by their corresponding binary labelings in  $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]$ .

We define the probability for  $\mathbf{x}_n \in C_k$  as the softmax function based on the linear function  $\mathbf{w}_k^T \mathbf{x}_n$ :

$$s_{nk} = p(\hat{y} = k | \mathbf{x}_n) = \frac{e^{\mathbf{w}_k^T \mathbf{x}_n}}{\sum_{l=1}^K e^{\mathbf{w}_l^T \mathbf{x}_n}}, \quad (k = 1, \dots, K, \quad n = 1, \dots, N) \quad (6.31)$$

and the probability for  $\mathbf{x}_n$  to be correctly classified into class  $C_{y_n}$  can be written as the following product of  $K$  factors (of which  $K - 1$  are equal to 1):

$$p(\hat{y} = y_n | \mathbf{x}_n, \mathbf{W}) = \prod_{k=1}^K p(\hat{y} = k | \mathbf{x}_n, \mathbf{W})^{y_{nk}} = \prod_{k=1}^K s_{nk}^{y_{nk}} \quad (6.32)$$

Our goal is to find these weight vectors in  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]$  as the parameter of the softmax model for it to optimally fit the  $N$  i.i.d. data points in the training set, so that that the following likelihood is maximized:

$$\begin{aligned} L(\mathbf{W} | \mathcal{D}) &\propto p(\mathcal{D} | \mathbf{W}) = \prod_{n=1}^N p(\hat{y} = y_n | \mathbf{x}_n, \mathbf{W}) = \prod_{n=1}^N \left( \prod_{k=1}^K s_{nk}^{y_{nk}} \right) \\ &= \prod_{n=1}^N \prod_{k=1}^K \left( \frac{e^{\mathbf{w}_k^T \mathbf{x}_n}}{\sum_{l=1}^K e^{\mathbf{w}_l^T \mathbf{x}_n}} \right)^{y_{nk}} \end{aligned} \quad (6.33)$$

Equivalently we can also minimize the negative log likelihood as the objective

function:

$$\begin{aligned}
J(\mathbf{W}) &= -l(\mathbf{W}|\mathcal{D}) = -\log L(\mathbf{w}|\mathcal{D}) = -\sum_{n=1}^N \sum_{k=1}^K y_{nk} \log \left( \frac{e^{\mathbf{w}_k^T \mathbf{x}_n}}{\sum_{l=1}^K e^{\mathbf{w}_l^T \mathbf{x}_n}} \right) \\
&= -\sum_{n=1}^N \sum_{k=1}^K y_{nk} \left( \mathbf{w}_k^T \mathbf{x}_n - \log \sum_{l=1}^K e^{\mathbf{w}_l^T \mathbf{x}_n} \right) \\
&= -\sum_{n=1}^N \left[ \sum_{k=1}^K y_{nk} \mathbf{w}_k^T \mathbf{x}_n - \sum_{k=1}^K y_{nk} \left( \log \sum_{l=1}^K e^{\mathbf{w}_l^T \mathbf{x}_n} \right) \right] \\
&= -\sum_{n=1}^N \left[ \sum_{k=1}^K y_{nk} \mathbf{w}_k^T \mathbf{x}_n - \log \sum_{l=1}^K e^{\mathbf{w}_l^T \mathbf{x}_n} \right] \quad (6.34)
\end{aligned}$$

The last equality is due to the fact that the last summation of  $K$  terms is independent of  $k$  and  $\sum_{k=1}^K y_{nk} = 1$ . Also, same as in the case of  $K = 2$ , to avoid overfitting, we could further include in this objective function extra regularization terms  $\lambda \|\mathbf{w}_k\|^2/2$ , ( $k = 1, \dots, K$ ) to penalize large weights for smoother thresholding:

$$J(\mathbf{W}) = -l(\mathbf{W}|\mathcal{D}) + \frac{\lambda}{2} \sum_{k=1}^K \|\mathbf{w}_k\|^2 \quad (6.35)$$

To find the optimal  $\mathbf{W}$  that minimize this objective function  $J(\mathbf{W})$ , we find its gradient vector with respect to each of its  $K$  columns  $\mathbf{w}_i$ , ( $i = 1, \dots, K$ ):

$$\begin{aligned}
\mathbf{g}_i(\mathbf{W}) &= \frac{d}{d\mathbf{w}_i} J(\mathbf{W}) = \frac{d}{d\mathbf{w}_i} \left[ -l(\mathbf{W}|\mathcal{D}) + \frac{\lambda}{2} \sum_{k=1}^K \|\mathbf{w}_k\|^2 \right] \\
&= -\sum_{n=1}^N \left[ \sum_{k=1}^K y_{nk} \frac{d}{d\mathbf{w}_i} (\mathbf{w}_k^T \mathbf{x}_n) - \frac{d}{d\mathbf{w}_i} \left( \log \sum_{l=1}^K e^{\mathbf{w}_l^T \mathbf{x}_n} \right) \right] + \lambda \mathbf{w}_i \\
&= -\sum_{n=1}^N \left[ \sum_{k=1}^K y_{nk} \delta_{ik} \mathbf{x}_n - \left( \frac{e^{\mathbf{w}_i^T \mathbf{x}_n}}{\sum_{l=1}^K e^{\mathbf{w}_l^T \mathbf{x}_n}} \right) \mathbf{x}_n \right] + \lambda \mathbf{w}_i = \sum_{n=1}^N (s_{in} - y_{in}) \mathbf{x}_n + \lambda \mathbf{w}_i \\
&= [\mathbf{x}_1, \dots, \mathbf{x}_N] \begin{bmatrix} s_{i1} - y_{i1} \\ \vdots \\ s_{iN} - y_{iN} \end{bmatrix} + \lambda \mathbf{w}_i = [\mathbf{x}_1, \dots, \mathbf{x}_N] \begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix} + \lambda \mathbf{w}_i \\
&= \mathbf{X} \mathbf{r}_i + \lambda \mathbf{w}_i \quad (6.36)
\end{aligned}$$

where  $r_n = s_{in} - y_{in}$  is the residual, the difference between the model output  $s_{in}$  and the binary labeling  $y_{in}$ , the  $i$ th component of  $\mathbf{y}_n$  that labels  $\mathbf{x}_n$  as well as  $y_n$ .

As  $s_{in}$  is a function of all  $K$  weight vectors in  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]$ , so is  $r_{in} = s_{in} - y_{in}$ , and  $\mathbf{r}_i$  can be explicitly expressed as  $\mathbf{r}_i(\mathbf{W})$ . We note that Eq. (6.36) takes the same form as Eq. (6.15) for  $K = 2$ . When specially  $K = 2$  with  $k \in \{0, 1\}$ , the two equations become the same. We therefore see that logistic regression is actually a special case of softmax regression.

We stack all  $K$  such  $d + 1$  dimensional gradient vectors  $\mathbf{g}_1, \dots, \mathbf{g}_K$  together, and get a  $(d + 1)K$  dimensional gradient vector of the objective function  $J(\mathbf{W})$  with respect to all  $K$  parameter vectors  $\mathbf{w}_1, \dots, \mathbf{w}_K$ :

$$\mathbf{g}_J = \begin{bmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_K \end{bmatrix} \quad (6.37)$$

based on which the optimal parameters in  $\mathbf{W}$  that minimize the objective function  $J(\mathbf{W})$  can be found by the gradient descent method.

We can also find the optimal  $\mathbf{W}$  by the Newton's method if we can further find the Hessian matrix  $\mathbf{H}_J$  as the second derivative of  $J(\mathbf{W})$ , by taking the derivative of the  $i$ th gradient vector  $\mathbf{g}_i(\mathbf{W})$  ( $i = 1, \dots, K$ ) with respect to the  $j$ th weight vector  $\mathbf{w}_j$  ( $j = 1, \dots, K$ ) to get the second order derivative of  $J(\mathbf{W})$  with respect to both  $\mathbf{w}_i$  and  $\mathbf{w}_j$ , ( $i, j = 1, \dots, K$ ):

$$\begin{aligned} \mathbf{H}_{ij} &= \frac{d^2}{d\mathbf{w}_j d\mathbf{w}_i} J(\mathbf{W}) = \frac{d}{d\mathbf{w}_j} \frac{d}{d\mathbf{w}_i} J(\mathbf{W}) = \frac{d}{d\mathbf{w}_j} \mathbf{g}_i = \frac{d}{d\mathbf{w}_j} [\mathbf{X}\mathbf{r}_i + \lambda\mathbf{w}_i] \\ &= \mathbf{X} \frac{d\mathbf{r}_i}{d\mathbf{w}_j} + \delta_{ij}\lambda = \mathbf{X}\mathbf{J}_{ij} + \delta_{ij}\lambda \end{aligned} \quad (6.38)$$

where  $\mathbf{J}_{ij} = d\mathbf{r}_i/d\mathbf{w}_j$  is the  $N \times (d + 1)$  Jacobian matrix of  $\mathbf{r}_i$  with respect to  $\mathbf{w}_j$ , of which the  $n$ th row is the transpose of the following vector:

$$\begin{aligned} \frac{d}{d\mathbf{w}_j} r_{in} &= \frac{d}{d\mathbf{w}_j} (s_{in} - y_{in}) = \frac{d}{d\mathbf{w}_j} s_{in} = \frac{d}{d\mathbf{w}_j} s(\mathbf{w}_i^T \mathbf{x}_n) \\ &= s_{in} (\delta_{ij} - s_{jn}) \frac{d}{d\mathbf{w}_j} (\mathbf{w}_j^T \mathbf{x}_n) = z_n \mathbf{x}_n \quad (n = 1, \dots, N) \end{aligned} \quad (6.39)$$

where we have used Eq. (6.28) and defined  $z_n = s_{in}(\delta_{ij} - s_{jn})$ . Now the Jacobian can be written as

$$\mathbf{J}_{ij} = \begin{bmatrix} dr_{i1}/d\mathbf{w}_j \\ \vdots \\ dr_{iN}/d\mathbf{w}_j \end{bmatrix} = \begin{bmatrix} z_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & z_N \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \mathbf{Z}\mathbf{X}^T \quad (6.40)$$

where  $\mathbf{Z} = \text{diag}(z_1, \dots, z_N)$  is an  $N \times N$  diagonal matrix. Substituting  $\mathbf{J}_r(\mathbf{W})$  back into the expression for  $\mathbf{H}_{ij}$ , we get:

$$\mathbf{H}_{ij} = \mathbf{X}\mathbf{J}_r + \delta_{ij}\lambda = \mathbf{X}\mathbf{Z}\mathbf{X}^T + \delta_{ij}\lambda = \sum_{n=1}^N s_{in}(\delta_{ij} - s_{jn})\mathbf{x}_n\mathbf{x}_n^T + \delta_{ij}\lambda \quad (6.41)$$

Here  $\mathbf{H}_{ij}$  is a matrix of dimension  $(d + 1) \times (d + 1)$ , corresponds to  $\mathbf{w}_i$  and  $\mathbf{w}_j$  ( $i, j = 1, \dots, K$ ). We further stack all  $K \times K$  such matrices  $\mathbf{H}_{ij}$  together to get the  $K(d + 1) \times K(d + 1)$  dimensional full Hessian matrix  $\mathbf{H}_J$  of  $J(\mathbf{W})$  with

respect to all  $K$  vectors in  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]$ :

$$\mathbf{H}_J = \begin{bmatrix} \mathbf{H}_{11} & \cdots & \mathbf{H}_{1K} \\ \vdots & \ddots & \vdots \\ \mathbf{H}_{K1} & \cdots & \mathbf{H}_{KK} \end{bmatrix} \quad (6.42)$$

Now we can find the optimal parameter  $\mathbf{W}$  iteratively by the Newton-Raphson method based on both  $\mathbf{H}_J$  and  $\mathbf{g}_J$ :

$$\begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_K \end{bmatrix}_{n+1} = \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_K \end{bmatrix}_n - \mathbf{H}_n^{-1} \mathbf{g}_n = \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_K \end{bmatrix}_n - \begin{bmatrix} \mathbf{H}_{11} & \cdots & \mathbf{H}_{1K} \\ \vdots & \ddots & \vdots \\ \mathbf{H}_{K1} & \cdots & \mathbf{H}_{KK} \end{bmatrix}_n^{-1} \begin{bmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_K \end{bmatrix}_n \quad (6.43)$$

Once  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]$  is available, any unlabeled sample  $\mathbf{x}$  of unknown class can be classified into one of the  $K$  classes with the maximum conditional probability given in Eq. (6.29):

$$\text{if } p(\hat{y} = k|\mathbf{x}) = \max_l p(\hat{y} = l|\mathbf{x}), \quad \text{then } \mathbf{x} \in C_k \quad (6.44)$$

Whether we should use softmax regression or  $K$  logistic regressions for a problem of  $K$  classes  $C_1, \dots, C_K$  depends on the nature of the classes. The method of softmax regression is suitable if the  $K$  classes are mutually exclusive and independent, as assumed by the method. Otherwise,  $K$  logistic regression binary classifiers are more suitable.

Below is a Matlab function for estimating the  $K$  weight vectors in  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]$  based on the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}$  and the hyperparameter  $\lambda$ :

```
function W=softmaxRegression(X,y,lambda)
    [d N]=size(X);
    K=length(unique(y));           % number of classes
    X=[ones(1,N); X];             % augmented data points
    d=d+1;
    Y=zeros(K,N);
    for n=1:N
        Y(y(n),n)=1;             % generate binary labeling Y
    end
    W=zeros(d*K,1);               % initial guess of K weight vectors
    I=eye(K);
    s=zeros(K,N);
    phi=zeros(K,N);               % softmax functions
    gi=zeros(d,1);                % ith gradient
    Hij=zeros(d,d);               % ij-th Hebbian
    er=9;
    tol=10^(-6);
    it=0;
    while er>tol
```



```

it=it+1;
W2=reshape(W,d,K);      % weight vectors in d x K 2-D array
g=[];                   % total gradient vector
for i=1:K
    for n=1:N
        xn=X(:,n);      % get the nth sample
        t=0;
        for k=1:K
            wk=W2(:,k); % get the kth weight vector
            s(k,n)=exp(wk'*xn);
            t=t+s(k,n);
        end
        phi(i,n)=s(i,n)/t; % softmax function
    end
    gi=X*(phi(i,:)-Y(i,:))'+lambda*W2(:,i); % ith gradient
    g=[g; gi];          % stack all gradients into a long gradient vector
end
H=[];                   % total Hessian
z=zeros(N,1);
for i=1:K               % for the ith block row
    Hi=[];
    for j=1:K           % for the jth block in ith row
        for n=1:N
            z(n)=phi(i,n)*((i==j)-phi(j,n));
        end
        Hij=X*diag(z)*X';
        Hi=[Hi Hij];   % append jth block
    end
    H=[H; Hi];        % append ith block row
end
H=H+lambda*eye(d*K); % include regulation term
W=W-inv(H)*g;        % update W by Newton's method
er=norm(g);
end
W=reshape(W,d,K);    % reshape weight vector into d x K array
end

```

Here is the function for the classification of the unlabeled data samples in matrix  $\mathbf{X}$  based on the weight vectors in  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]$ :

```

function yhat=softmaxClassify(W,X)
[d N]=size(X);          % dataset to be classified
[d K]=size(W);         % model parameters
X=[ones(1,N); X];     % augmented data points
yhat=zeros(N,1);

```



# 7 Gaussian Process Regression and Classification

---

## 7.1 Gaussian Process Regression

The *Gaussian process regression (GPR)* is yet another regression method that fits a regression function  $f(\mathbf{x})$  to the data samples in the given training set. Different from all previously considered algorithms that treat  $f(\mathbf{x})$  as a deterministic function with an explicitly specified form, the GPR treats the regression function  $f(\mathbf{x})$  as a stochastic process called *Gaussian process (GP)*, i.e., the function value  $f(\mathbf{x})$  at any point  $\mathbf{x}$  is assumed to be a random variable with a Gaussian distribution.

The joint probability distribution of the vector function  $\mathbf{f}(\mathbf{X}) = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)]^T$  for all  $N$  i.i.d. samples in the training set  $\mathbf{X}$  is also Gaussian:

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{m}_f, \Sigma_f) = \mathcal{N}(\mathbf{0}, \Sigma_f) \quad (7.1)$$

where, for convenience and without loss of generality, we have assumed a zero mean  $\mathbf{m}_f = \mathbf{0}$ . The ultimate goal is to estimate the regression function  $f(\mathbf{x})$  in terms of its Gaussian pdf  $p(\mathbf{f}|\mathbf{X})$ .

We also assume  $\mathbf{y} = \mathbf{f}(\mathbf{x}) + \mathbf{n}$  as the labeling of  $\mathbf{X}$  in the training set is contaminated by noise  $\mathbf{n}$ , with a zero-mean normal pdf

$$p(\mathbf{n}) = \mathcal{N}(\mathbf{n}, \mathbf{0}, \sigma_n^2 \mathbf{I}) \quad (7.2)$$

where the diagonal covariance matrix  $\sigma_n^2 \mathbf{I}$  is based on the assumption that all components of  $\mathbf{n}$  are independent of each other. We further assume noise  $\mathbf{n}$  is independent of  $\mathbf{f}$  with  $Cov(\mathbf{y}, \mathbf{n}) = \mathbf{0}$ . Now  $\mathbf{y} = \mathbf{f}(\mathbf{X}) + \mathbf{n} = \mathbf{f} + \mathbf{n}$ , as the sum of two zero-mean Gaussian variables, is also a zero mean Gaussian process.

The unknown covariance matrix  $\Sigma_f$  can be constructed based on the desired smoothness of the regression function  $f(\mathbf{x})$ , in the sense that any two function values  $f(\mathbf{x}_m)$  and  $f(\mathbf{x}_n)$  are likely to be more correlated if the distance  $\|\mathbf{x}_m - \mathbf{x}_n\|$  between the two corresponding points is small, but less so if the distance is large. Such a covariance can be modeled by a *squared exponential (SE) kernel* function:

$$Cov(f(\mathbf{x}_m), f(\mathbf{x}_n)) = \exp\left(-\frac{1}{a^2} \|\mathbf{x}_m - \mathbf{x}_n\|^2\right) = k(\mathbf{x}_m, \mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{x}_m) \quad (7.3)$$

where  $k(\mathbf{x}_m, \mathbf{x}_n)$  is a *kernel function* of  $\mathbf{x}_m$  and  $\mathbf{x}_n$ . This SE function may take any value between 1 when  $\|\mathbf{x}_m - \mathbf{x}_n\| = 0$  and  $f(\mathbf{x}_m) = f(\mathbf{x}_n)$  are maximally

correlated, and 0 when  $\|\mathbf{x}_m - \mathbf{x}_n\| = \infty$  and  $f(\mathbf{x}_m)$  and  $f(\mathbf{x}_n)$  are minimally correlated. We therefore see that the SE so constructed mimics the covariance of a smooth function  $f(\mathbf{x})$ .

The smoothness of the function  $f(\mathbf{x})$  plays a very important role as it determines whether the function overfits or underfits the sample points in the training data. If the function is not smooth enough, it may overfit as it may overly be affected by some noisy samples; on the other hand, if the function is too smooth, it may underfit as it may not follow the training samples closely enough.

The smoothness of the regression function can be controlled by the hyperparameter  $\alpha$  in Eq. (7.3). Consider two extreme cases. If  $a \rightarrow \infty$ , i.e., the value of the SE approaches 1, then  $f(\mathbf{x}_m)$  and  $f(\mathbf{x}_n)$  are highly correlated and  $f(\mathbf{x})$  is smooth; on the other hand, if  $a \rightarrow 0$ , i.e., SE approaches 0, then  $f(\mathbf{x}_m)$  and  $f(\mathbf{x}_n)$  are not correlated and  $f(\mathbf{x})$  is not smooth. Therefore by adjusting the value of  $a$ , a proper tradeoff between overfitting and underfitting can be made.

Given the covariance  $Cov(f(\mathbf{x}_m), f(\mathbf{x}_n)) = k(\mathbf{x}_m, \mathbf{x}_n)$  of function values at two different sample points  $\mathbf{x}_m$  and  $\mathbf{x}_n$ , we can represent the covariance matrix of the prior  $p(\mathbf{f}|\mathbf{X})$  as

$$\Sigma_f = Cov(\mathbf{f}, \mathbf{f}) = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} = K(\mathbf{X}, \mathbf{X}) = \mathbf{K} \quad (7.4)$$

Note that the value of all diagonal components are the same  $k(\mathbf{x}_n, \mathbf{x}_n) = 1$  ( $n = 1, \dots, N$ ); and the greater the distance  $\|\mathbf{x}_m - \mathbf{x}_n\|$ , the farther away the component  $k(\mathbf{x}_m, \mathbf{x}_n)$  is from the diagonal, and the lower value it takes.

Once the conditional probability  $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{0}, \Sigma_f)$  is estimated based on the training set  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , we can get the mean and covariance of  $\mathbf{f}_* = \mathbf{f}(\mathbf{X}_*)$  of a test dataset  $\mathbf{X}_* = [\mathbf{x}_{1*}, \dots, \mathbf{x}_{M*}]$  containing  $M$  unlabeled test samples. As both  $\mathbf{f}_*$  and  $\mathbf{f}$  are the same zero-mean Gaussian process, their joint distribution (given  $\mathbf{X}_*$  as well as  $\mathbf{X}$ ) is also a zero-mean Gaussian:

$$\begin{aligned} p\left(\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \middle| \mathbf{X}, \mathbf{X}_*\right) &= \mathcal{N}\left(\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix}, \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \Sigma_f & \Sigma_{ff_*} \\ \Sigma_{f_*f} & \Sigma_{f_*} \end{bmatrix}\right) \\ &= \mathcal{N}\left(\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix}, \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix}\right) \end{aligned} \quad (7.5)$$

where

$$\begin{cases} \Sigma_{ff_*} = Cov(\mathbf{f}, \mathbf{f}_*) = K(\mathbf{X}, \mathbf{X}_*) = \mathbf{K}_* \\ \Sigma_{f_*f} = Cov(\mathbf{f}_*, \mathbf{f}) = K(\mathbf{X}_*, \mathbf{X}) = \mathbf{K}_*^T \\ \Sigma_{f_*} = Cov(\mathbf{f}_*, \mathbf{f}_*) = K(\mathbf{X}_*, \mathbf{X}_*) = \mathbf{K}_{**} \end{cases} \quad (7.6)$$

can all be constructed based on the SE function, same as how  $K(\mathbf{X}, \mathbf{X}) = \mathbf{K}$  is constructed.

Having found the joint pdf  $p(\mathbf{f}, \mathbf{f}_*|\mathbf{X}, \mathbf{X}_*)$  of both  $\mathbf{f}$  and  $\mathbf{f}_*$  in Eq. (7.5), we can further find the conditional distribution  $p(\mathbf{f}_*|\mathbf{f}, \mathbf{X}, \mathbf{X}_*)$  of  $\mathbf{f}_*$  given  $\mathbf{f}$  (as well as

$\mathbf{X}$  and  $\mathbf{X}_*$ ), based on the properties of the Gaussian distributions discussed in Section 3.1.3:

$$p(\mathbf{f}_*|\mathbf{f}, \mathbf{X}, \mathbf{X}_*) = \mathcal{N}(\mathbf{m}_{(f_*|f)}, \boldsymbol{\Sigma}_{(f_*|f)}) \quad (7.7)$$

with the mean and covariance

$$\begin{cases} \mathbf{m}_{(f_*|f)} &= \mathbb{E}[\mathbf{f}_*|\mathbf{f}] = \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f} \\ \boldsymbol{\Sigma}_{(f_*|f)} &= \text{Cov}(\mathbf{f}_*|\mathbf{f}) = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_* \end{cases} \quad (7.8)$$

Similarly, we can also find the joint distribution of  $\mathbf{f}_*$  and  $\mathbf{y} = \mathbf{f} + \mathbf{n}$ , both of which are zero-mean Gaussian:

$$p(\mathbf{y}, \mathbf{f}_*|\mathbf{X}, \mathbf{X}_*) = \mathcal{N}\left(\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix}, \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_y & \boldsymbol{\Sigma}_{yf_*} \\ \boldsymbol{\Sigma}_{f_*y} & \boldsymbol{\Sigma}_{f_*} \end{bmatrix}\right) \quad (7.9)$$

where

$$\begin{aligned} \boldsymbol{\Sigma}_y &= \text{Cov}(\mathbf{y}, \mathbf{y}) = \text{Cov}(\mathbf{f} + \mathbf{n}, \mathbf{f} + \mathbf{n}) = \text{Cov}(\mathbf{f}, \mathbf{f}) + 2\text{Cov}(\mathbf{f}, \mathbf{n}) + \text{Cov}(\mathbf{n}, \mathbf{n}) \\ &= \text{Cov}(\mathbf{f}, \mathbf{f}) + \text{Cov}(\mathbf{n}, \mathbf{n}) = \boldsymbol{\Sigma}_f + \boldsymbol{\Sigma}_n = \mathbf{K} + \sigma_n^2 \mathbf{I} \\ \boldsymbol{\Sigma}_{yf_*} &= \text{Cov}(\mathbf{y}, \mathbf{f}_*) = \text{Cov}(\mathbf{f} + \mathbf{n}, \mathbf{f}_*) = \text{Cov}(\mathbf{f}, \mathbf{f}_*) + \text{Cov}(\mathbf{n}, \mathbf{f}_*) \\ &= \text{Cov}(\mathbf{f}, \mathbf{f}_*) = \boldsymbol{\Sigma}_{ff_*} = \mathbf{K}_* \end{aligned} \quad (7.10)$$

where  $\text{Cov}(\mathbf{f}, \mathbf{n}) = \text{Cov}(\mathbf{f}_*, \mathbf{n}) = \mathbf{0}$ . Now we have

$$\text{Cov}\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Sigma}_y & \boldsymbol{\Sigma}_{yf_*} \\ \boldsymbol{\Sigma}_{f_*y} & \boldsymbol{\Sigma}_{f_*} \end{bmatrix} = \begin{bmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix} \quad (7.11)$$

We can now further get the desired conditional distribution of  $\mathbf{f}_*$  given  $\mathbf{X}_*$  as well as the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ :

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathcal{D}) = p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{m}_{(f_*)}, \boldsymbol{\Sigma}_{(f_*)}) \quad (7.12)$$

with the mean and covariance

$$\begin{cases} \mathbf{m}_{(f_*)} &= \mathbb{E}[\mathbf{f}_*|\mathbf{y}] = \mathbf{K}_*^T (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} \\ \boldsymbol{\Sigma}_{(f_*)} &= \text{Cov}(\mathbf{f}_*|\mathbf{y}) = \mathbf{K}_{**} - \mathbf{K}_*^T (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{K}_* \end{cases} \quad (7.13)$$

We see that these expressions are the same as those in Eq. (4.87) previously obtained by Bayesian linear regression, i.e., the same results can be derived by two very different methods.

In the special noise-free case of  $\mathbf{n} = \mathbf{0}$  (with zero mean and zero covariance  $\sigma_n^2 = 0$ ), we have  $\mathbf{y} = \mathbf{f} + \mathbf{n} = \mathbf{f}$ , and the mean and covariance in Eq. (7.13) become the same as Eq. (7.8), and Eqs. (7.12) and (7.7) become the same:

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) = p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f}) \quad (7.14)$$

The Matlab code below for the Gaussian process regression algorithm, takes the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ , the test samples in  $\mathbf{X}_*$ , and the variance  $\sigma_n^2$  of the noise as the input, and generates as the output the posterior  $p(\mathbf{f}_*|\mathcal{D}, \mathbf{X}_*)$ , in terms of its mean as the GPR regression function, and its covariance matrix, of which the variances on the diagonal represent the confidence or certainty of the regression.

```

function [fmean fcov]=GPR(X,y,Xstar,sigmaN) % given training set D={X,y}, test set Dstar={Xstar,ystar}
% find regression function fmean and covariance fcov
Sy=Kernel(X,X)+sigmaN^2*eye(length(X)); % Covariance of training samples
Syf=Kernel(X,Xstar); % Covariance of training and test samples
Sf=Kernel(Xstar,Xstar); % Covariance of test samples
fmean=Syf'*inv(Sy)*y; % mean of posterior as the regression function
fcov=Sf-Syf'*inv(Sy)*Syf; % covariance of posterior
end

```

The following code is for a function that generates the squared exponential kernel matrix based on two sets of data samples in X1 and X2:

```

function K=Kernel(X1,X2)
a=1; % parameter that controls smoothness of f(x)
m=length(X1);
n=length(X2);
K=zeros(m,n);
for i=1:m
for j=1:n
d=norm(X1(:,i)-X2(:,j));
K(i,j)=exp(-(d/a)^2);
end
end
end

xstar=xmin:0.1:xmax; % test samples
n=length(xstar); % number of test samples
Kss=Kernel(xstar,xstar); % K(x*,x*), covariance of prior
[L p]=chol(Kss); % Cholesky decomposition
for i=1:k
plot(xstar,L'*randn(n,1)) % plot k sample curves from prior
hold on
end
hold off
x=xstar(randi(n,1,N)); % get N random samples from xstar
y=f(xstar)+randn(1,n)*sigma_n^2; % get their noisy labeling values
Kxx=Kernel(x,x)+sigma_n^2*eye(N); % K(x,x)
Kxs=Kernel(x,xstar); % K(x,x*)
Ksx=Kxs'; % K(x*,x)
fmean=Ksx*inv(Kxx)*y; % mean of posterior, regression function
fcov=Kss-Ksx*inv(Kxx)*Kxs; % covariance of posterior
plot(xstar,fmean,xstar,fmean+diag(fcov),xstar,fmean-diafcov));

```

```

[L p]=chol(fcov);           % Cholesky decomposition
for i=1:k
    plot(xstar,L'*randn(n,1)) % plot k sample curves drawn from posterior
    hold on
end
hold off

```

**Example:**

A sinusoidal function  $f(x) = \cos(2\pi fx)$  ( $f = 0.2$ ) contaminated by Gaussian noise  $e$  with  $p(r) = \mathcal{N}(0, \sigma_r^2)$  ( $\sigma_r = 0.2$ ) is sampled at  $N = 7$  random positions over the duration of the function  $(0, 0)$ . These samples form the training dataset  $\mathcal{D} = \{(x_n, y_n) (n = 1, \dots, N)\}$ .

Then the method of GPR is used to estimate the mean and variance of  $f_{i*} = f(x_{i*})$  at 91 test points  $x_{i*}$ , evenly distributed over the duration of the function between the limits  $x_{min} = 0$  and  $x_{max} = 9$ ,

The 8 panels of the figure below show the estimated mean  $E[f_*]$ , and the variance  $\pm \text{Var}[f_*]$ , both above and below the mean, to form a region of uncertainty of the estimates. The wider the region, the more uncertain the results are (less confidence).

The top left panel represents the prior probability  $p(f_*|x_*)$  of the estimated model output  $f_* = \hat{y}_*$  at point  $x_*$  with no knowledge based on the training set  $\mathbf{x}$  ( $N = 0$  training samples), which is assumed to have zero-mean  $E[f_*] = 0$  and unity variance  $\text{Var}[f(x_*)] = k(x_*, x_*) = 1$ . The subsequent seven panels show the mean and variance of the posterior  $p(f_* = \hat{y}_*|x_*, \mathcal{D})$  based on  $N = 1, \dots, 7$  training samples in the training set  $\mathcal{D}$ . Note that wherever a training sample  $x$  becomes available, the estimated mean  $\mu_{f_*|y}$  is modified to be close to the true function value  $f(x)$ , and the variance of the posterior around  $x$  is reduced, indicating the uncertainty of the estimate is reduced, or the confidence is much increased. When all  $N = 7$  training samples are used, the general shape of the original function is reasonably reconstructed.

The 8 panels in the figure below show 12 sample curves drawn from each posterior  $p(f_*|\mathbf{x}_*, \mathcal{D})$ . They can be treated as different interpolations for the  $N$  observed samples in training set  $\mathcal{D}$ , and any of them can be used to predict the outputs  $y_*$  at any input points  $x_*$ . The mean of these samples shown as the red curve is simply the same as  $E[f_*]$  shown in the previous figure.

We see that wherever a training sample  $f(x)$  is available at  $x$ , the variance and thereby the width of the region of uncertainty is reduced, and correspondingly the sample curves in the neighborhood of  $x$  are more tightly distributed. Specifically, we see that the sample curves drawn from the prior in the top left panel are very loosely distributed, while those samples drawn from the posterior obtained based on all  $N = 7$  training points in the bottom right panel are relatively tightly distributed. In particular, note that due to lack of training data in the interval

$5 < x < 8$ , the region of uncertainty is significantly wider than elsewhere, and the sample curves are much more loosely located.

The parameter  $\alpha$  of the kernel function plays an important role in the GPR results, as shown in the three panels in the figure below, corresponding respectively to  $a$  values of 0.5, 1.5, and 3.0. When  $a$  is too small (top panel), the model may overfit the data, as the regression function fits the training samples well but it is not smooth and it does not interpolate regions between the training samples well (with large variance); on the other hand, when  $a$  is too large (bottom panel), the model may underfit the data, as the regression curve is smooth, but it does not fit the training samples well. We therefore see that we need to choose the value of  $a$  carefully for a proper tradeoff between overfitting and underfitting (middle panel).

## 7.2 Gaussian Process Classifier – Binary

In both logistic regression and softmax regression considered previously, we convert the linear regression function  $\hat{y} = f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$  to the probability  $p(\hat{y} = k | \mathbf{x}, \mathbf{W})$  for  $\mathbf{x} \in C_k$ , by either the logistic function  $\sigma(f(\mathbf{x}))$  for binary classification, or the softmax function  $s(f(\mathbf{x}))$  for multiclass classification. Also, in Gaussian process regression (GPR), we treat the regression function  $f(\mathbf{x})$  as a Gaussian process. Now we consider the *Gaussian process classification (GPC)* based on the combination of both logistic/softmax regression and Gaussian process regression. We will consider binary GPC based on the logistic function in this section, and then multiclass GPC based on softmax function in the following section.

Here in binary GPC, we assume the training samples are labeled by 1 or -1 (instead of 0) for mathematical convenience, i.e.,  $\mathbf{x}_n \in C_+$  if  $y_n = 1$  or  $\mathbf{x}_n \in C_-$  if  $y_n = -1$ . Similar to logistic programming, here we also convert the regression function  $f(\mathbf{x})$ , now a Gaussian process, into the probability for  $\mathbf{x} \in C_k$  by the logistic function  $\sigma(f(\mathbf{x}))$ . However, as a function of this random argument,  $\sigma(f(\mathbf{x}))$  is also random. We therefore define  $p(y = 1 | \mathbf{x}, \mathcal{D})$  as the expectation of  $\sigma(f(\mathbf{x}))$  with respect to  $f(\mathbf{x})$ :

$$p(y = 1 | \mathbf{x}, \mathcal{D}) = E_f[\sigma(f(\mathbf{x}))] = \int \sigma(f(\mathbf{x})) p(f | \mathbf{x}, \mathcal{D}) df \quad (7.15)$$

As function  $f(\mathbf{x})$  is marginalized (averaged out) in the integral above, it is hidden instead of explicitly specified, and is therefore called a *latent function*.

For the training set  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , we define  $\mathbf{f}(\mathbf{X}) = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)]^T$ , and express the probability above in vector form:

$$p(\mathbf{y} = 1 | \mathcal{D}) = E_f[\sigma(\mathbf{f}(\mathbf{X}))] = \int \sigma(\mathbf{f}(\mathbf{X})) p(\mathbf{f} | \mathcal{D}) d\mathbf{f} \quad (7.16)$$

where  $p(\mathbf{f} | \mathcal{D})$  is the posterior which can be found in terms of the likelihood



$p(\mathbf{y}|\mathbf{f}) = L(\mathbf{f}|\mathbf{y})$  and the prior  $p(\mathbf{f}|\mathbf{X})$  based on Bayes' theorem

$$p(\mathbf{f}|\mathcal{D}) = p(\mathbf{f}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}, \mathbf{f}|\mathbf{X})}{p(\mathbf{y}|\mathbf{X})} = \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X})}{p(\mathbf{y}|\mathbf{X})} \propto p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X}) \quad (7.17)$$

As always, the denominator  $p(\mathbf{y}|\mathbf{X})$  independent of  $\mathbf{f}$  is dropped.

We first find the likelihood  $p(\mathbf{y}|\mathbf{f})$ . The Gaussian process  $f(\mathbf{x})$  is mapped by the logistic function into the probability for  $y = 1$  for  $\mathbf{x} \in C_+$ , or  $y = -1$  for  $\mathbf{x} \in C_-$ :

$$\begin{aligned} p(y = 1|f(\mathbf{x})) &= \sigma(f(\mathbf{x})) \\ p(y = -1|f(\mathbf{x})) &= 1 - p(y = 1|f(\mathbf{x})) = 1 - \sigma(f(\mathbf{x})) = \sigma(-f(\mathbf{x})) \end{aligned} \quad (7.18)$$

These two cases can be combined to get the conditional probability of  $y$  given  $f(\mathbf{x})$ :

$$p(y|f) = \sigma(y f(\mathbf{x})) = \sigma(y f) = \frac{1}{1 + e^{-yf}} = \frac{e^{yf}}{1 + e^{yf}} \quad (7.19)$$

The likelihood of  $\mathbf{f}$  for all  $N$  i.i.d. samples in the training set  $\mathbf{X}$  is:

$$p(\mathbf{y}|\mathbf{f}) = \prod_{n=1}^N p(y_n|f_n) = \prod_{n=1}^N \sigma(y_n f_n) \quad (7.20)$$

We then consider the prior  $p(\mathbf{f}|\mathbf{X})$ , which is assumed to be a zero-mean Gaussian  $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_f)$ . Here the covariance matrix  $\mathbf{\Sigma}_f$  can be constructed based on the training set  $\mathbf{X}$ , the same as in GPR. Specifically, the covariance between  $f(\mathbf{x}_m)$  and  $f(\mathbf{x}_n)$  in the  $m$ th row and  $n$ th column of  $\mathbf{\Sigma}_f$  is modeled by the squared exponential (SE) kernel:

$$\text{cov}(f(\mathbf{x}_m), f(\mathbf{x}_n)) = k(\mathbf{x}_m, \mathbf{x}_n) = \exp\left(-\frac{1}{a^2}\|\mathbf{x}_m - \mathbf{x}_n\|^2\right), \quad (m, n = 1, \dots, N) \quad (7.21)$$

Such a covariance matrix is desired for GPC as well as GPR, so that  $f(\mathbf{x}_m)$  and  $f(\mathbf{x}_n)$  are more correlated if  $\mathbf{x}_m$  and  $\mathbf{x}_n$  are close together, but less so if they are farther apart. However the justification for such a property is different. In GPR, this property is desired for the smoothness of the regression function; while here in GPC, this property is also desired so that function values  $f(\mathbf{x}_m)$  and  $f(\mathbf{x}_n)$  for two samples  $\mathbf{x}_m$  and  $\mathbf{x}_n$  close to each other in the feature space are more correlated and therefore the two samples are more likely to be classified into the same class, but less so if they are far apart.

Also, as discussed in GPR, here the parameter  $a$  in the SE controls the smoothness of the function. If  $a \rightarrow \infty$ , the value of the SE approaches 1, then  $f(\mathbf{x}_m)$  and  $f(\mathbf{x}_n)$  are highly correlated and  $f(\mathbf{x})$  is very smooth; but if  $a \rightarrow 0$ , SE approaches 0, then  $f(\mathbf{x}_m)$  and  $f(\mathbf{x}_n)$  are not correlated and  $f(\mathbf{x})$  is no longer smooth. By adjusting  $a$ , a proper tradeoff can be made between overfitting and underfitting.

Having found both the likelihood  $p(\mathbf{y}|\mathbf{f})$  and the prior  $p(\mathbf{f})$ , we can get the

posterior in Eq. (7.17):

$$\begin{aligned} p(\mathbf{f}|\mathcal{D}) &\propto p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|\mathbf{X}) = p(\mathbf{y}|\mathbf{f}) \mathcal{N}(\mathbf{0}, \mathbf{K}) = \prod_{n=1}^N \sigma(y_n f_n) \frac{1}{(2\pi)^{d/2} |\mathbf{K}|^{1/2}} \exp\left(-\frac{1}{2} \mathbf{f}^T \mathbf{K}^{-1} \mathbf{f}\right) \\ &\propto \prod_{n=1}^N \sigma(y_n f_n) \exp\left(-\frac{1}{2} \mathbf{f}^T \mathbf{K}^{-1} \mathbf{f}\right) \end{aligned} \quad (7.22)$$

Note that the likelihood  $p(\mathbf{y}|\mathbf{f})$  is not Gaussian, as the binary labeling  $\mathbf{y}$  of the training set  $\mathbf{X}$  is not continuous. Consequently, the posterior  $p(\mathbf{f}|\mathcal{D})$ , as a product of the Gaussian prior and non-Gaussian likelihood, is not Gaussian. However, we can carry out *Laplace approximation* and still approximate the posterior as a Gaussian:

$$p(\mathbf{f}|\mathcal{D}) \approx \mathcal{N}(\mathbf{m}_{f|D}, \mathbf{\Sigma}_{f|D}) \quad (7.23)$$

of which the mean  $\mathbf{m}_{f|D}$  and covariance  $\mathbf{\Sigma}_{f|D}$  are to be obtained in the following. We further get the log posterior denoted by  $\psi(\mathbf{f})$ :

$$\psi(\mathbf{f}) = \log p(\mathbf{f}|\mathcal{D}) = \sum_{n=1}^N \log \sigma(y_n f_n) - \frac{N}{2} \log(2\pi) - \frac{1}{2} \log |\mathbf{K}| - \frac{1}{2} \mathbf{f}^T \mathbf{K}^{-1} \mathbf{f} \quad (7.24)$$

The two middle terms are constant independent of  $\mathbf{f}$  and can therefore be dropped.

Now that the posterior  $p(\mathbf{f}|\mathcal{D})$  is approximated as a Gaussian, we can find the gradient vector  $\mathbf{g}_\psi(\mathbf{f})$  and Hessian matrix  $\mathbf{H}_\psi(\mathbf{f})$  of the log posterior:

$$\mathbf{g}_\psi(\mathbf{f}) = \frac{d}{d\mathbf{f}} \psi(\mathbf{f}) = -\mathbf{\Sigma}_{f|D}^{-1} (\mathbf{f} - \mathbf{m}_{f|D}) \quad (7.25)$$

$$\mathbf{H}_\psi(\mathbf{f}) = \frac{d^2}{d\mathbf{f}^2} \psi(\mathbf{f}) = \frac{d}{d\mathbf{f}} \mathbf{g}_\psi(\mathbf{f}) = -\mathbf{\Sigma}_{f|D}^{-1} \quad (7.26)$$

On the other hand, we can also find  $\mathbf{g}_\psi(\mathbf{f})$  and  $\mathbf{H}_\psi(\mathbf{f})$  by directly taking the first and second order derivatives of the log posterior::

$$\mathbf{g}_\psi(\mathbf{f}) = \frac{d}{d\mathbf{f}} \psi(\mathbf{f}) = \frac{d}{d\mathbf{f}} \log p(\mathbf{y}|\mathbf{f}) - \frac{d}{d\mathbf{f}} \left( \frac{1}{2} \mathbf{f}^T \mathbf{K}^{-1} \mathbf{f} \right) = \mathbf{w} - \mathbf{K}^{-1} \mathbf{f} \quad (7.27)$$

$$\mathbf{H}_\psi(\mathbf{f}) = \frac{d^2}{d\mathbf{f}^2} \psi(\mathbf{f}) = \frac{d}{d\mathbf{f}} \mathbf{g}_\psi(\mathbf{f}) = \frac{d}{d\mathbf{f}} (\mathbf{w} - \mathbf{K}^{-1} \mathbf{f}) = \mathbf{W} - \mathbf{K}^{-1} \quad (7.28)$$

where we have defined

$$\mathbf{w} = \frac{d}{d\mathbf{f}} \log p(\mathbf{y}|\mathbf{f}), \quad \mathbf{W} = \frac{d^2}{d\mathbf{f}^2} \log p(\mathbf{y}|\mathbf{f}) = \frac{d\mathbf{w}}{d\mathbf{f}} \quad (7.29)$$

are respectively the gradient vector and Hessian matrix of the log function

$$\log p(\mathbf{y}|\mathbf{f}) = \log \prod_{n=1}^N \sigma(y_n f_n) = \sum_{n=1}^N \log \sigma(y_n f_n) = \sum_{n=1}^N \log \left( \frac{1}{1 + \exp(-y_n f_n)} \right) \quad (7.30)$$

We first find the first and second order derivatives with respect to a single component  $f_n$  of  $\mathbf{f}$ :

$$\begin{aligned}\frac{d}{df_n} \log p(y_n|f_n) &= \frac{d}{df_n} \log \sigma(y_n f_n) = \frac{d}{df_n} \log \left( \frac{1}{1 + \exp(-y_n f_n)} \right) = \frac{y_n}{1 + \exp(y_n f_n)} \\ \frac{d^2}{df_n^2} \log p(y_n|f_n) &= \frac{d^2}{df_n^2} \log \sigma(y_n f_n) = \frac{d}{df_n} \left( \frac{y_n}{1 + \exp(y_n f_n)} \right) = \frac{-\exp(-y_n f_n) y_n}{(1 + \exp(-y_n f_n))^2}\end{aligned}$$

where we have used the facts that  $y_n = \pm 1$  and  $y_n^2 = 1$ . We then find

$$\mathbf{w} = \frac{d}{d\mathbf{f}} \log p(\mathbf{y}|\mathbf{f}) = \begin{bmatrix} d/df_1 \\ \vdots \\ d/df_N \end{bmatrix} \sum_{n=1}^N \log p(y_n|f_n) = \begin{bmatrix} d \log p(y_1|f_1)/df_1 \\ \vdots \\ d \log p(y_N|f_N)/df_N \end{bmatrix} = \begin{bmatrix} \frac{y_1}{1+e^{y_1 f_1}} \\ \vdots \\ \frac{y_N}{1+e^{y_N f_N}} \end{bmatrix} \quad (7.32)$$

and

$$\begin{aligned}\mathbf{W} &= \frac{d^2}{d\mathbf{f}^2} \log p(\mathbf{y}|\mathbf{f}) = \begin{bmatrix} \frac{\partial^2}{\partial f_1 \partial f_1} & \cdots & \frac{\partial^2}{\partial f_1 \partial f_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial f_N \partial f_1} & \cdots & \frac{\partial^2}{\partial f_N \partial f_N} \end{bmatrix} \sum_{n=1}^N \log p(y_n|f_n) \\ &= \text{diag} \left[ \frac{d^2}{df_1^2} \log p(y_1|f_1), \dots, \frac{d^2}{df_N^2} \log p(y_N|f_N) \right] = \text{diag} \left[ \frac{-e^{-y_1 f_1}}{(1 + e^{-y_1 f_1})^2}, \dots, \frac{-e^{-y_N f_N}}{(1 + e^{-y_N f_N})^2} \right]\end{aligned}$$

Equating the two expressions for  $\mathbf{H}_\psi(\mathbf{f})$  in Eqs. (7.26) and (7.28) we get:

$$\mathbf{H}_\psi(\mathbf{f}) = \mathbf{W} - \mathbf{K}^{-1} = -\Sigma_{f|D}^{-1}, \quad \text{i.e.,} \quad \Sigma_{f|D} = (\mathbf{K}^{-1} - \mathbf{W})^{-1} \quad (7.34)$$

Equating the two expressions for  $\mathbf{g}_\psi(\mathbf{f})$  in Eqs. (7.25) and (7.27) we get:

$$\mathbf{g}_\psi(\mathbf{f}) = -\Sigma_{f|D}^{-1}(\mathbf{f} - \mathbf{m}_{f|D}) = \mathbf{w} - \mathbf{K}^{-1}\mathbf{f} \quad \text{i.e.,} \quad \mathbf{m}_{f|D} = \mathbf{f} + \Sigma_{f|D}(\mathbf{w} - \mathbf{K}^{-1}\mathbf{f}) = \mathbf{f} - \mathbf{H}_\psi^{-1}\mathbf{g}_\psi \quad (7.35)$$

Substituting  $\Sigma_{f|D}$  given in Eq. (7.34) into the equation and solving it for  $\mathbf{m}_{f|D}$ , we get

$$\mathbf{m}_{f|D} = \mathbf{f} + (\mathbf{K}^{-1} + \mathbf{W})^{-1}(\mathbf{w} - \mathbf{K}^{-1}\mathbf{f}) \quad (7.36)$$

We recognize this is actually the iteration of Newton's method solving equation  $\mathbf{g}_\psi(\mathbf{f}) = \mathbf{0}$ , in consistence with the fact that when  $\mathbf{f} = \mathbf{m}_{f|D}$ , the log posterior  $\psi(\mathbf{f}) = \log p(\mathbf{f}|\mathcal{D})$ , assumed to be Gaussian, reaches maximum with  $\mathbf{g}_\psi(\mathbf{f}) = \mathbf{0}$ . However, we note that in the equation above,  $\mathbf{m}_{f|D}$  as a value of  $\mathbf{f}$  is expressed in terms of both  $\mathbf{w}$  and  $\mathbf{W}$ , which in turn are functions of  $\mathbf{f}$  as given in Eqs. (7.32) and (7.33), i.e.,  $\mathbf{m}_{f|D}$  given above is not in a closed form, as  $\mathbf{f}$  and parameters  $\mathbf{w}$  and  $\mathbf{W}$  are interdependent. We instead need to carry out an iteration during which both  $\mathbf{f}$  and the parameters  $\mathbf{w}$  and  $\mathbf{W}$  are updated alternatively:

$$\begin{aligned}\mathbf{f}_{n+1} &= \mathbf{f}_n - \mathbf{H}_\psi^{-1} \mathbf{g}_\psi = \mathbf{f}_n + (\mathbf{K}^{-1} - \mathbf{W})^{-1} (\mathbf{w} - \mathbf{K}^{-1}\mathbf{f}_n) \\ &= \mathbf{f}_n + (\mathbf{K}^{-1} - \mathbf{W})^{-1} [-(\mathbf{K}^{-1} - \mathbf{W})\mathbf{f}_n + \mathbf{w} - \mathbf{W}\mathbf{f}_n] \\ &= (\mathbf{K}^{-1} - \mathbf{W})^{-1} (\mathbf{w} - \mathbf{W}\mathbf{f}_n)\end{aligned} \quad (7.37)$$

This iterative process converges to  $\mathbf{f} = \mathbf{m}_{f|D}$ , at which  $p(\mathbf{f}|\mathcal{D})$  is maximized, and

$$\mathbf{g}_\psi(\mathbf{f}) = \frac{d}{d\mathbf{f}}\psi(\mathbf{f}) = \mathbf{w} - \mathbf{K}^{-1}\mathbf{f} = \mathbf{0}, \quad \text{i.e.,} \quad \mathbf{f} = \mathbf{m}_{f|D} = \mathbf{K} \mathbf{w} \quad (7.38)$$

Now that  $\mathbf{w}$  and  $\mathbf{W}$  as well as  $\mathbf{f} = \mathbf{m}_{f|D}$  are available, we can further obtain  $\Sigma_{f|D}$  in Eq. (7.34), and the posterior  $p(\mathbf{f}|\mathcal{D}) \approx \mathcal{N}(\mathbf{m}_{f|D}, \Sigma_{f|D})$ .

Having found  $p(\mathbf{f}|\mathcal{D})$ , we can proceed to carry out classification of any test set  $\mathbf{X}_*$  based on the probability (similar to Eq. (7.16)):

$$p(\mathbf{y}_* = 1|\mathbf{X}_*, \mathcal{D}) = E_f[\sigma(\mathbf{f}(\mathbf{X}_*))] = \int \sigma(\mathbf{f}(\mathbf{X}_*)) p(\mathbf{f}|\mathbf{X}_*, \mathcal{D}) d\mathbf{f} \quad (7.39)$$

We first approximate the posterior of  $\mathbf{f}_* = \mathbf{f}(\mathbf{X}_*)$  in the equation as a Gaussian:

$$p(\mathbf{f}|\mathbf{X}_*, \mathcal{D}) \approx \mathcal{N}(\mathbf{m}_{f_*}, \Sigma_{f_*}) \quad (7.40)$$

where the mean  $\mathbf{m}_{f_*}$  and covariance  $\Sigma_{f_*}$  can be obtained based on the fact that both  $\mathbf{f}_* = \mathbf{f}(\mathbf{X}_*)$  and  $\mathbf{f} = \mathbf{f}(\mathbf{X})$  are the same Gaussian process, i.e., their joint probability  $p(\mathbf{f}, \mathbf{f}_*)$  is a Gaussian. The method is therefore the same as what is discussed in the method of GPR. Specifically, we take the following steps:

- Find the mean  $\mathbf{m}_{f_*|f} = E(\mathbf{f}_*|\mathbf{f})$  and covariance  $\Sigma_{f_*|f} = Cov(\mathbf{f}_*|\mathbf{f})$  of  $p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f})$  conditioned on the latent function  $\mathbf{f}$ , same as in Eq. (7.8) for GPR:

$$\begin{cases} \mathbf{m}_{f_*|f} = \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f} \\ \Sigma_{f_*|f} = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_* \end{cases} \quad (7.41)$$

- Find  $\mathbf{m}_{f_*} = E_f(\mathbf{f}_*)$  as the expectation of  $\mathbf{m}_{f_*|f} = \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f}$  conditioned on  $\mathbf{f}$ , i.e. find the average of  $\mathbf{m}_{f_*|f}$  over  $\mathbf{f}$  based on  $p(\mathbf{f}|\mathcal{D})$  (marginalize over  $\mathbf{f}$ ):

$$\begin{aligned} \mathbf{m}_{f_*} &= E_f(\mathbf{m}_{f_*|f}) = E_f(\mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f}) = \mathbf{K}_*^T \mathbf{K}^{-1} E_f(\mathbf{f}) = \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{m}_f \\ &= \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K} \mathbf{w}_{m_f} = \mathbf{K}_*^T \mathbf{w}_{m_f} \end{aligned} \quad (7.42)$$

where  $\mathbf{m}_f = E_f(\mathbf{f}) = \mathbf{K} \mathbf{w}_{m_f}$  given in Eq. (7.38).

- Find the covariance  $\Sigma_{m_{f_*|f}} = E_f[(\mathbf{m}_{f_*|f} - \mathbf{m}_{f_*})^2]$ . Given the covariance  $\Sigma_{f|D} = (\mathbf{K}^{-1} - \mathbf{W})^{-1}$  of  $\mathbf{f}$  in Eq. (7.34), we can further find the covariance of  $\mathbf{m}_{f_*|f} = \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f}$  as a linear combination of  $\mathbf{f}$  (Recall if  $\mathbf{y} = \mathbf{A}\mathbf{x}$ , then  $\Sigma_y = \mathbf{A}\Sigma_x \mathbf{A}^T$ ):

$$\Sigma_{m_{f_*|f}} = \mathbf{K}_*^T \mathbf{K}^{-1} \Sigma_{f|D} \mathbf{K}^{-1} \mathbf{K}_* = \mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{K}^{-1} - \mathbf{W})^{-1} \mathbf{K}^{-1} \mathbf{K}_* \quad (7.43)$$

- Find the covariance  $\Sigma_{f_*}$  of  $\mathbf{f}_*$  as the sum of  $\Sigma_{f_*|f} = E[(\mathbf{f}_* - \mathbf{m}_{f_*|f})(\mathbf{f}_* - \mathbf{m}_{f_*|f})^T]$  for the variation of  $\mathbf{f}_*$  with respect to  $\mathbf{m}_{f_*|f}$ , and  $\Sigma_{m_{f_*|f}} = E_f[(\mathbf{m}_{f_*|f} - \mathbf{m}_{f_*})^2]$  for the variation of  $\mathbf{m}_{f_*|f}$  with respect to  $\mathbf{m}_{f_*}$ :

$$\begin{aligned} \Sigma_{f_*} &= \Sigma_{f_*|f} + \Sigma_{m_{f_*|f}} \\ &= (\mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_*) + (\mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{K}^{-1} - \mathbf{W})^{-1} \mathbf{K}^{-1} \mathbf{K}_*) \\ &= \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_* + \mathbf{K}_*^T \mathbf{K}^{-1} [\mathbf{K} - \mathbf{K}(\mathbf{K} - \mathbf{W}^{-1})^{-1} \mathbf{K}] \mathbf{K}^{-1} \mathbf{K}_* \\ &= \mathbf{K}_{**} - \mathbf{K}_*^T (\mathbf{K} - \mathbf{W}^{-1})^{-1} \mathbf{K}_* \end{aligned} \quad (7.44)$$

Here we have used the identity for  $(\mathbf{A} + \mathbf{B})^{-1}$  given in Section ??.

Now that the posterior  $p(\mathbf{f}|\mathbf{X}_*, \mathcal{D}) = \mathcal{N}(\mathbf{m}_{f_*}, \mathbf{\Sigma}_{f_*})$  is approximated as a Gaussian with mean and covariance given in Eqs. (7.42) and (7.44), we can finally carry out Eq. (7.16) to find the probability for the test points in  $\mathbf{X}_*$  to belong to class  $C_1$ :

$$\begin{aligned} p(y_* = 1|\mathbf{X}_*, \mathcal{D}) &= \int \sigma(\mathbf{f}_*) p(\mathbf{f}_*|\mathbf{X}_*, \mathcal{D}) d\mathbf{f}_* \\ &= E_f[\sigma(\mathbf{f}(\mathbf{X}_*))] = \sigma(E_f(\mathbf{f}(\mathbf{X}_*))) = \sigma(\mathbf{m}_{f_*}) \end{aligned} \quad (7.45)$$

Same as in GPR, the certainty or confidence of this classification result is represented by the variances on the diagonal of the covariance  $\mathbf{\Sigma}_{f_*}$ .

The Matlab code for the essential parts of this algorithm is listed below. Here  $\mathbf{X}$  and  $\mathbf{y}$  are for the training data  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ , and  $\mathbf{X}_s$  is an array composed of test vectors. First, the essential segment of the main program listed below takes in the training and test data, generates the covariance matrices  $\mathbf{K}$ ,  $\mathbf{K}_*$ , and  $\mathbf{K}_{**}$  represented by  $K$ ,  $K_s$ , and  $K_{ss}$ , respectively. The function `Kernel` is exactly the same as the one used for Gaussian process regression. This code segment then further calls a function `findPosteriorMean` which finds the mean and covariance of  $p(\mathbf{f}|\mathbf{X}, \mathbf{y})$  based on covariance of training data ( $\mathbf{K} = \text{Cov}(\mathbf{X})$ ) and  $\mathbf{y}$ , and computes the mean  $\mathbf{m}_{f_*|y}$  and covariance  $\mathbf{\Sigma}_{f_*|y}$  based on Eqs. (7.42) and (7.44), respectively. The sign function of  $\mathbf{m}_{f_*|y}$  indicates the classification of the test data points in  $\mathbf{X}_*$ .

```

K=Kernel(X,X);           % cov(f,f), covariance of prior p(f|X)
Ks=Kernel(X,Xs);        % cov(f_*,f)
Kss=Kernel(Xs,Xs);      % cov(f_*,f_*)
[Sigmaf W w]=findPosteriorMean(K,y);
                           % get mean/covariance of p(f|D), W, w
meanfD=Ks'*w;           % mean of p(f_*|X_*,D)
SigmafD=Kss-Ks'*inv(K-inv(W))*Ks; % covariance of p(f_*|X_*,D)
ys=sign(meanfD);        % binary classification of test data
p=1./(1+exp(-meanfD));  % p(y_*=1|X_*,D) as logistic function

```

The function `findPosteriorMean` listed below uses Newton's method to find the mean and covariance of the posterior  $p(\mathbf{f}|\mathcal{D})$  of the latent function  $\mathbf{f}$  based on the training data, and returns them in `meanf` and `covf`, respectively, together with `w` and `W` for the gradient and Hessian of the likelihood  $p(\mathbf{y}|\mathbf{f})$ , to be used for computing  $\mathbf{m}_{f_*|D}$  and  $\mathbf{\Sigma}_{f_*|D}$ .

```

function [meanf covf W w]=findPosteriorMean(K,y)
    % K: covariance of prior of p(f|X)
    % y: labeling of training data X
    % w: gradient vector of log p(y|f)
    % W: Hessian matrix of log p(y|f)
    % meanf: mean of p(f|X,y)
    % covf: covariance of p(f|X,y)

```

```

n=length(y); % number of training samples
f0=zeros(n,1); % initial value of latent function
f=f0;
er=1;
while er > 10^(-9) % Newton's method to get f that maximizes p(f|y)
    e=exp(-y.*f);
    w=y.*e./(1+e); % update w
    W=diag(-e./(1+e).^2); % update W
    f=inv(inv(K)-W)*(w-W*f0); % iteration to get f from previous f0
    er=norm(f-f0); % difference between two consecutive f's
    f0=f; % update f
end
meanf=f; % mean of f
covf=inv(inv(K)-W); % coviance of f
end

```

**Example 1** The GPC method is trained by the two classes shown in the figure below, represented by 100 red points and 80 blue points drawn from two Gaussian distributions  $\mathcal{N}(\mathbf{m}_0, \Sigma_0)$  and  $\mathcal{N}(\mathbf{m}_1, \Sigma_1)$ , where

$$\mathbf{m}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{m}_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad \Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix} \quad (7.46)$$

The 3-D plot of these two normalized Gaussian distributions is also shown in the figure.

Three classification results are shown in the figure below corresponding to different values used in the kernel function. On the left, the 2-D space is partitioned into red and blue regions corresponding to the two classes based on the sign function of  $\mathbf{m}_{f_*|y}$ ; on the right, the 3-D distribution plots of  $\sigma(\mathbf{m}_{f_*|y})$  representing the estimated probability for  $\mathbf{x}_*$  to belong to either class (not normalized) is shown, to be compared with the original Gaussian distributions from which the training samples were drawn.

We see that wherever there is evidence represented by the training samples of either class in red or blue, there are high probabilities for the neighboring points to belong to either  $C_1$  or  $C_0$  represented by the positive or negative peaks in the 3-D plots. Data points far away from any evidence will have low probability to belong to either class.

We make the following observations for three different values of the parameter  $\alpha$  in SE:

- $\alpha = 0.4$  (top row), the space is partitioned into three regions, with 20 out of 180 training points misclassified. The estimated distribution is smooth.
- $\alpha = 0.2$  (middle row), the space is fragmented into several more pieces for the two classes (blue islands inside the red region and vice versa), with 5 out of the 180 training points misclassified. The estimated distribution is jagged.

- $\alpha = 0.03$  (bottom row), the space is partitioned into still more pieces, with all 180 training points correctly classified. The estimated distribution is spiky.

Although the error rate is lowest when  $\alpha$  is small, the classification result is not necessarily the best as it may well be an overfitting of the noisy data. We conclude that by adjusting parameter  $\alpha$ , we can make proper tradeoff between error rate and overfitting.

### 7.3 Gaussian Process Classifier – Multi-Class

The binary GPC considered previously can be generalized to multi-class GPC based on softmax function, similar to the how binary classification based on logistic function is generalized to multi-class classification. First of all, we define the following variables for each class  $C_k$ , ( $k = 1, \dots, K$ ) of the  $K$  classes  $\{C_1, \dots, C_K\}$ :

- **Class labeling:**

Binary labeling  $\mathbf{y}_k = [y_{1k}, \dots, y_{Nk}]^T$  indicating whether each of the  $N$  samples in the training set  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  belongs to  $C_k$ :

$$y_{nk} = \begin{cases} 1 & \text{if } \mathbf{x}_n \in C_k \\ 0 & \text{if } \mathbf{x}_n \notin C_k \end{cases} \quad (7.47)$$

In other words, each training sample  $\mathbf{x}_n \in C_k$  is labeled by  $K$  binary variables  $y_{nk} = 1$  and  $y_{nl} = 0$  for all  $l = 1, \dots, K$ ,  $l \neq k$  (same as in softmax regression), as well as an integer labeling  $y_n = k$ .

- **Latent functions:**

Latent function  $\mathbf{f}_k = [f_{1k}, \dots, f_{Nk}]^T$ , of which the  $n$ th component  $f_{nk} = f_k(\mathbf{x}_n)$  ( $n = 1, \dots, N$ ) is  $k$ th Gaussian process associated with  $C_k$  evaluated at the  $n$ th training sample  $\mathbf{x}_n$  in the training set  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ .

- **Probability modeling:**

Probability  $\mathbf{p}_k = [p_{1k}, \dots, p_{Nk}]^T$ , of which the  $n$ th component  $p_{nk}$  ( $n = 1, \dots, N$ ) is the probability for  $\mathbf{x}_n \in C_k$ , modeled by the softmax function based on  $f_{nk}$  (same as in Eq. (6.31) in softmax regression):

$$p_{nk} = p(\hat{y} = k | \mathbf{x}_n) = \frac{e^{f_{nk}}}{\sum_{l=1}^K e^{f_{nl}}}, \quad (k = 1, \dots, K, \quad n = 1, \dots, N) \quad (7.48)$$

Note that if  $y_{nk} = 1$  then necessarily  $y_{nl} = 0$  for all  $l = 1, \dots, K$  but  $l \neq k$ , i.e., the  $K$  variables  $y_{n1}, \dots, y_{nK}$  are not independent. given  $p(y_{nk} = 1)$ , we do not need to consider  $p(y_{nl} = 0)$  for any  $l \neq k$ .

The probability for  $\mathbf{x}_n$  to be correctly classified into class  $C_{y_n}$  can be written as the following product of  $K$  factors (of which  $K - 1$  are equal to 1, same as

Eq. (6.32) in softmax regression):

$$p(\hat{y} = y_n | \mathbf{x}_n, \mathbf{f}) = \prod_{k=1}^K p(y_{nk} = 1 | \mathbf{x}_n, \mathbf{f})^{y_{nk}} = \prod_{k=1}^K p_{nk}^{y_{nk}} \quad (7.49)$$

Based on  $\mathbf{y}_k$ ,  $\mathbf{f}_k$ , and  $\mathbf{p}_k$  for all  $k = 1, \dots, K$ , we further define the following  $KN$  dimensional vectors:

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_K \end{bmatrix} = \begin{bmatrix} y_{11} \\ \vdots \\ y_{N1} \\ \vdots \\ y_{1K} \\ \vdots \\ y_{NK} \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_K \end{bmatrix} = \begin{bmatrix} f_{11} \\ \vdots \\ f_{N1} \\ \vdots \\ f_{1K} \\ \vdots \\ f_{NK} \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_K \end{bmatrix} = \begin{bmatrix} p_{11} \\ \vdots \\ p_{N1} \\ \vdots \\ p_{1K} \\ \vdots \\ p_{NK} \end{bmatrix} \quad (7.50)$$

The posterior of  $\mathbf{f}$  given the training set  $\mathcal{D}$  can be found based on the Bayesian theorem as (same as in Eq. (7.17) in the binary case):

$$p(\mathbf{f} | \mathcal{D}) = p(\mathbf{f} | \mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y} | \mathbf{f}) p(\mathbf{f} | \mathbf{X})}{p(\mathbf{y} | \mathbf{X})} \propto p(\mathbf{y} | \mathbf{f}) p(\mathbf{f} | \mathbf{X}) \quad (7.51)$$

We first find the likelihood based on all  $p_{nk}$  (same as Eq. (6.33) in softmax regression):

$$L(\mathbf{f} | \mathbf{y}) \propto p(\mathbf{y} | \mathbf{f}) = \prod_{n=1}^N p(\hat{y} = y_n | \mathbf{x}_n, \mathbf{f}) = \prod_{n=1}^N \left( \prod_{k=1}^K (p_{nk})^{y_{nk}} \right) = \prod_{n=1}^N \prod_{k=1}^K \left( \frac{e^{f_{nk}}}{\sum_{h=1}^K e^{f_{nh}}} \right)^{y_{nk}} \quad (7.52)$$

As each sample  $\mathbf{x}_n$  belongs to only one of the  $K$  classes, only one of  $\{y_{n1}, \dots, y_{nK}\}$  can be 1 while all others are 0, consequently, the product above contains only  $N$  probabilities raised to the power of  $y_{nk} = 1$ , each for one of the  $N$  samples belonging to a certain class, while all other probabilities raised to the power of  $y_{nk} = 0$  become 1 and not considered.

We next assume the prior probability of the latent function  $\mathbf{f}_k$  for each class  $C_k$  to be a zero-mean Gaussian process  $p(\mathbf{f}_k | \mathbf{X}) = \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_k)$ , where the covariance matrix  $\mathbf{\Sigma}_k$  is constructed based on the squared exponential (SE) for its  $m$ -th component:

$$\text{cov}(f_k(\mathbf{x}_m), f_k(\mathbf{x}_n)) = \text{cov}(f_{mk}, f_{nk}) = k(\mathbf{x}_m, \mathbf{x}_n) = \exp\left(-\frac{1}{a^2} \|\mathbf{x}_m - \mathbf{x}_n\|^2\right), \quad (m, n = 1, \dots, N) \quad (7.53)$$

Then the prior of  $\mathbf{f}$  containing all  $K$  such latent functions is also a Gaussian  $p(\mathbf{f} | \mathbf{X}) = \mathcal{N}(\mathbf{0}, \mathbf{K})$ , where  $\mathbf{0}$  is a  $KN$ -dimensional zero vector and  $\mathbf{K}$  is a  $KN \times KN$



block diagonal matrix

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{K}_K \end{bmatrix} \quad (7.54)$$

with  $\mathbf{K}_1 = \cdots = \mathbf{K}_K$  as defined in Eq. (7.4) on the diagonal. All off-diagonal blocks are zero as the latent functions of different classes are uncorrelated.

Having found both the likelihood  $p(\mathbf{y}|\mathbf{f})$  and prior  $p(\mathbf{f}|\mathbf{X})$ , we can write the posterior as

$$p(\mathbf{f}|\mathcal{D}) \propto p(\mathbf{f}|\mathbf{X}, \mathbf{y}) \propto p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|\mathbf{X}) = \prod_{n=1}^N \prod_{k=1}^K \left( \frac{e^{f_{nk}}}{\sum_{h=1}^K e^{f_{nh}}} \right)^{y_{nk}} \mathcal{N}(\mathbf{0}, \mathbf{K}) \quad (7.55)$$

We note that the likelihood  $p(\mathbf{y}|\mathbf{f})$  is not Gaussian, as  $y_{nk} \in \{0, 1\}$  is a binary labeling, instead of a continuous variable. As a product of the Gaussian prior and non-Gaussian likelihood, the posterior  $p(\mathbf{f}|\mathcal{D})$  is not a Gaussian process either. However, for convenience, we still assume it is approximately Gaussian  $p(\mathbf{f}|\mathcal{D}) \approx \mathcal{N}(\mathbf{f}, \mathbf{m}_{f|\mathcal{D}}, \mathbf{\Sigma}_{f|\mathcal{D}})$ , of which the mean  $\mathbf{m}_{f|\mathcal{D}}$  and covariance  $\mathbf{\Sigma}_{f|\mathcal{D}}$  are to be found.

Taking log of the posterior, we get

$$\begin{aligned} \psi(\mathbf{f}) &= \log p(\mathbf{f}|\mathcal{D}) = \log p(\mathbf{y}|\mathbf{f}) + \log p(\mathbf{f}|\mathbf{X}) = \sum_{n=1}^N \sum_{k=1}^K y_{nk} \left( f_{nk} - \log \sum_{h=1}^K e^{f_{nh}} \right) + \log \mathcal{N}(\mathbf{0}, \mathbf{K}) \\ &= \mathbf{y}^T \mathbf{f} - \sum_{n=1}^N \log \sum_{h=1}^K e^{f_{nh}} - \frac{NK}{2} \log(2\pi) - \frac{1}{2} \log |\mathbf{K}| - \frac{1}{2} \mathbf{f}^T \mathbf{K}^{-1} \mathbf{f} \end{aligned} \quad (7.56)$$

where we have used the fact that  $\sum_{k=1}^K y_{nk} = 1$ . The gradient of  $\psi(\mathbf{f})$  is

$$\begin{aligned} \mathbf{g}_\psi &= \frac{d}{d\mathbf{f}} \psi(\mathbf{f}) = \frac{d}{d\mathbf{f}} \left( \mathbf{y}^T \mathbf{f} - \sum_{n=1}^N \log \sum_{h=1}^K e^{f_{nh}} - \frac{NK}{2} \log(2\pi) - \frac{1}{2} \log |\mathbf{K}| - \frac{1}{2} \mathbf{f}^T \mathbf{K}^{-1} \mathbf{f} \right) \\ &= \mathbf{y} - \mathbf{p} - \mathbf{K}^{-1} \mathbf{f} \end{aligned} \quad (7.57)$$

where the second term  $\mathbf{p}$  comes from

$$\sum_{n=1}^N \frac{d}{d\mathbf{f}} \log \sum_{h=1}^K e^{f_{nh}} = \sum_{n=1}^N \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \frac{e^{f_{nk}}}{\sum_{h=1}^K e^{f_{nh}}} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \sum_{n=1}^N \begin{bmatrix} 0 \\ \vdots \\ p_{nk} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \mathbf{p} \quad (7.58)$$

Note that  $\mathbf{p} = \mathbf{p}(\mathbf{f})$  is a function of  $\mathbf{f}$ .

As the posterior  $p(\mathbf{f}|\mathcal{D}) \approx \mathcal{N}(\mathbf{m}_{f|D}, \boldsymbol{\Sigma}_{f|D})$  is approximated as a Gaussian, which reaches maximum at its mean  $\mathbf{m}_{f|D}$ , and so does the log prior  $\psi(\mathbf{f}) = \log p(\mathbf{f}|\mathcal{D})$ , and the gradient of  $\psi(\mathbf{f})$  is zero at  $\mathbf{f} = \mathbf{m}_{f|D}$ :

$$\mathbf{g}_\psi(\mathbf{f}) \Big|_{\mathbf{f}=\mathbf{m}_{f|D}} = \mathbf{y} - \mathbf{p} - \mathbf{K}^{-1}\mathbf{f} \Big|_{\mathbf{f}=\mathbf{m}_{f|D}} = \mathbf{y} - \mathbf{p}_{m_f} - \mathbf{K}^{-1}\mathbf{m}_{f|D} = \mathbf{0}, \quad (7.59)$$

i.e.,

$$\mathbf{m}_{f|D} = \mathbf{K}(\mathbf{y} - \mathbf{p}_{m_f}) \quad (7.60)$$

where  $\mathbf{p}_{m_f} = \mathbf{p}(\mathbf{m}_{f|D})$  is the vector defined above evaluated at  $\mathbf{f} = \mathbf{m}_{f|D}$ .

We further get the Hessian matrix of  $\psi(\mathbf{f})$ :

$$\mathbf{H}_\psi(\mathbf{f}) = \frac{d^2}{d\mathbf{f}^2}\psi(\mathbf{f}) = \frac{d}{d\mathbf{f}}\mathbf{g}_\psi = \frac{d}{d\mathbf{f}}(\mathbf{y} - \mathbf{K}^{-1}\mathbf{f} - \mathbf{p}) = -\mathbf{K}^{-1} - \mathbf{W} = -\boldsymbol{\Sigma}_{f|D}^{-1} \quad (7.61)$$

The last equality is due to the property of the Gaussian distribution given in Section 3.1.3, from which we get the KN by KN covariance matrix of  $p(\mathbf{f}|\mathcal{D})$ :

$$\boldsymbol{\Sigma}_{f|D} = -\mathbf{H}_\psi^{-1}(\mathbf{f}) = (\mathbf{K}^{-1} + \mathbf{W})^{-1} \quad (7.62)$$

Here  $\mathbf{W} = d\mathbf{p}/d\mathbf{f}$  is a  $KN \times KN$  Jacobian matrix of  $\mathbf{p}$ , of which the  $ijkl$ -th component ( $i, j = 1, \dots, N$ ,  $k, l = 1, \dots, K$ ) is:

$$\begin{aligned} \frac{\partial}{\partial f_{jl}} p_{ik} &= \frac{\partial}{\partial f_j^l} \left[ \frac{e^{f_{ik}}}{\sum_{h=1}^K e^{f_{ih}}} \right] = \frac{e^{f_{ik}} \left( \sum_{h=1}^K e^{f_{ih}} \right) \delta_{kl} - e^{f_{ik}} e^{f_{jl}}}{\left( \sum_{h=1}^K e^{f_{ih}} \right)^2} \delta_{ij} \\ &= \left[ \frac{e^{f_{ik}}}{\sum_{h=1}^K e^{f_{ih}}} \delta_{kl} - \frac{e^{f_{ik}} e^{f_{jl}}}{\left( \sum_{h=1}^K e^{f_{ih}} \right)^2} \right] \delta_{ij} = (p_{ik} \delta_{kl} - p_{ik} p_{jl}) \delta_{ij} \end{aligned} \quad (7.63)$$

and  $\mathbf{W}$  can be written also in two terms:

$$\mathbf{W} = \text{diag}(\mathbf{p}) - \mathbf{P}\mathbf{P}^T \quad (7.64)$$

where the first term  $\text{diag}(\mathbf{p})$  is a diagonal matrix containing all  $KN$  components of  $\mathbf{p}$  along the diagonal ( $k = l$  and  $i = j$ ), and  $\mathbf{P}$  in the second term is a  $KN$  by  $N$  matrix composed of  $K$   $N \times N$  diagonal matrices  $\text{diag}(\mathbf{p}_k)$ :

$$\mathbf{P} = \begin{bmatrix} \text{diag}(\mathbf{p}_1) \\ \vdots \\ \text{diag}(\mathbf{p}_K) \end{bmatrix}, \quad \text{diag}(\mathbf{p}_k) = \begin{bmatrix} p_{1k} & 0 & \cdots & 0 \\ 0 & p_{2k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & p_{Nk} \end{bmatrix}, \quad (k = 1, \dots, K) \quad (7.65)$$

so that

$$\mathbf{P}\mathbf{P}^T = \begin{bmatrix} \text{diag}(\mathbf{p}_1) \\ \vdots \\ \text{diag}(\mathbf{p}_K) \end{bmatrix} [\text{diag}(\mathbf{p}_1), \dots, \text{diag}(\mathbf{p}_K)] = \begin{bmatrix} \text{diag}(\mathbf{p}_1^2) & \cdots & \text{diag}(\mathbf{p}_1)\text{diag}(\mathbf{p}_K) \\ \vdots & \ddots & \vdots \\ \text{diag}(\mathbf{p}_K)\text{diag}(\mathbf{p}_1) & \cdots & \text{diag}(\mathbf{p}_K^2) \end{bmatrix} \quad (7.66)$$

with

$$\text{diag}(\mathbf{p}_k)\text{diag}(\mathbf{p}_l) = \begin{bmatrix} p_1^k p_1^l & 0 & \cdots & 0 \\ 0 & p_1^k p_1^l & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & p_{Nk} p_{Nl} \end{bmatrix} \quad (7.67)$$

Having found both the gradient  $\mathbf{g}_\psi = \mathbf{y} - \mathbf{p} - \mathbf{K}^{-1}\mathbf{f}$  and Hessian  $\mathbf{H}_\psi = -(\mathbf{K}^{-1} + \mathbf{W})$ , we can further find the mean  $\mathbf{m}_{f|D}$  at which  $\psi(\mathbf{f})$  achieves maximum by the following iteration of Newton's method:

$$\begin{aligned} \mathbf{f}_{n+1} &= \mathbf{f}_n - \mathbf{H}_\psi^{-1} \mathbf{g}_\psi = \mathbf{f}_n + (\mathbf{K}^{-1} + \mathbf{W})^{-1} (\mathbf{y} - \mathbf{K}^{-1} \mathbf{f}_n - \mathbf{p}) \\ &= \mathbf{f}_n + (\mathbf{K}^{-1} + \mathbf{W})^{-1} (-(\mathbf{K}^{-1} + \mathbf{W}) \mathbf{f}_n + \mathbf{W} \mathbf{f}_n + \mathbf{y} - \mathbf{p}) \\ &= (\mathbf{K}^{-1} + \mathbf{W})^{-1} (\mathbf{W} \mathbf{f}_n + \mathbf{y} - \mathbf{p}) \end{aligned} \quad (7.68)$$

We note that during the iteration, we need to update not only  $\mathbf{f}$ , but also  $\mathbf{p}$  as a functions of  $\mathbf{f}$  given in Eq. (7.58).

Now that we have got both  $\mathbf{m}_{f|D}$  and  $\Sigma_{f|D}$  of  $\mathbf{f}$ , we can further get  $\mathbf{m}_{f_*}$  and  $\Sigma_{f_*}$  of  $\mathbf{f}_*$ :

- Get mean  $\mathbf{m}_{f_*}$ :

Similar to Eq. (7.42) in the binary case, we have the following based on  $\mathbf{f} = \mathbf{K}(\mathbf{y} - \mathbf{p})$  in Eq. (7.60):

$$\begin{aligned} \mathbf{m}_{f_*} &= E_f(\mathbf{m}_{f_*|f}) = E_f(\mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f}) = \mathbf{K}_*^T \mathbf{K}^{-1} E_f(\mathbf{f}) \\ &= \mathbf{K}_*^T \mathbf{K}^{-1} [\mathbf{K}(\mathbf{y} - \mathbf{p}_{m_f})] = \mathbf{K}_*^T (\mathbf{y} - \mathbf{p}_{m_f}) \end{aligned} \quad (7.69)$$

As the  $K$  classes are uncorrelated, i.e.,  $\mathbf{K}$  is block-diagonal, the above can be separated into  $K$  equations each for one of the classes:

$$\mathbf{m}_{f_*^k} = (\mathbf{K}_*^k)^T (\mathbf{y}^k - \mathbf{p}^k), \quad (k = 1, \dots, K) \quad (7.70)$$

- Get covariance  $\Sigma_{f_*}$ :

Similar to Eq. (7.44) in the binary case, we have the following based on  $\Sigma_{f|D} = (\mathbf{K}^{-1} + \mathbf{W})^{-1}$  in Eq. (7.62):

$$\Sigma_{m_{f_*|f}} = \mathbf{K}_*^T \mathbf{K}^{-1} \Sigma_{f|D} \mathbf{K}^{-1} \mathbf{K}_* = \mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{K}^{-1} + \mathbf{W})^{-1} \mathbf{K}^{-1} \mathbf{K}_* \quad (7.71)$$

where  $\mathbf{W} = \text{diag}(\mathbf{p}) - \mathbf{P}\mathbf{P}^T$  is given in Eq. (7.64) with  $\mathbf{p}$  evaluated at  $\mathbf{m}_{f|D}$ . Then, similar to Eq. (7.44), we get

$$\Sigma_{f_*} = \mathbf{K}_{**} - \mathbf{K}_*^T (\mathbf{K} + \mathbf{W}^{-1})^{-1} \mathbf{K}_* \quad (7.72)$$

Now we can further get the probability for  $\mathbf{x}_*$  to belong to  $C_k$  based on the softmax function in Eq. (7.48)

$$p_*^k = \frac{e^{m_{f_*}^k}}{\sum_{l=1}^K e^{m_{f_*}^l}}, \quad (k = 1, \dots, K) \quad (7.73)$$

and classify  $\mathbf{x}_*$  to class  $C_k$  if  $p_*^k = \max\{p_*^1, \dots, p_*^K\}$ .

Moreover, the certainty or confidence of this classification result can be found from  $\Sigma_{f_*}$ .

The Matlab code for the essential parts of the algorithm is listed below. First, the following code segment carries out the classification of  $n$  given test samples based on the  $N$  training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ .

```

K=Kernel(X,X); % covariance of prior of p(f|X)
Ks=Kernel(X,Xs);
Kss=Kernel(Xs,Xs);
[meanf Sigmaf p W]=findPosteriorMeanMC(K,y,C);
% find mean and covariance of p(f|D), and
Sigmaf=Kss-Ks'*inv(K+inv(W))*Ks; % covariance of p(f|D)
p=reshape(p,N,C);
y=reshape(y,N,C);
for k=1:C
    meanfD(:,k)=Ks'*(y(:,k)-p(:,k)); % mean of p(f_*|X_*,D) for kth class
end
for i=1:n % for each of n test samples
    d=sum(exp(meanfD(i,:))); % denominator of softmax function
    [pmax k]=max(exp(meanfD(i,:))/d); % find class with max probability
    ys(i)=k; % label ith sample as member of kth class
    pr(i,k)=pmax; % probability of ith sample belonging to k
end

```

The code segment above calls the following function which computes the mean and covariance of  $p(\mathbf{f}|\mathbf{X}, \mathbf{y})$  by Newton's method:

```

function [meanf Sigmaf p W]=findPosteriorMeanMC(K,y,C)
% find mean and covariance of p(f|X,y) by Newton's method, based on D={X,y}
n=length(y); % number of training samples
f0=zeros(n,1); % initial value of latent function
f=zeros(n,1);
er=1;
k=0;
while er > 10^(-9) % find f that maximizes p(f|X,y)
    k=k+1;
    [p W]=findp(f,N,C); % call function findp to find vector p and mat
    f=inv(inv(K)+W)*(W*f0+y-p); % iteratively update value of f
    er=norm(f-f0); % difference between consecutive iterations
end

```

```

        f0=f;                % update f
    end
    meanf=f;
    Sigmaf=inv(inv(K)+W);
end

```

The following function called by the previous function finds vector  $\mathbf{p}$  and matrix  $\mathbf{W} = \text{diag}(\mathbf{p}) - \mathbf{P}\mathbf{P}^T$ :

```

function [p W]=findp(f,N,C)    % find vector p and matrix W=diag(p)-P*P'
    F=reshape(f,N,C)';        % kth row contains N samples of class k
    p=zeros(C,N);            % initialize p
    for n=1:N                  % for each of N training samples
        d=sum(exp(F(:,n)));    % sum of all C terms in denominator
        for k=1:C              % for all C classes
            p(k,n)=exp(F(k,n))/d;
        end
    end
    P=[];
    for k=1:C                   % generate P
        P=[P; diag(p(k,:))];  % stack C diagonal matrices
    end
    p=reshape(p',N*C,1);       % convert p into a column vector
    W=diag(p)-P*P';           % generate W matrix
end

```

#### Example 1:

This example shows the classification result of the same dataset of three classes used before. The top two panels show the distributions of the three classes in the training data set. The bottom two panels show the classification results in terms of the partitioning of the feature space (bottom left) and the posterior distribution  $p(\mathbf{f}_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*)$  (bottom right), which can be compared with the distribution of the training set (top right). The confusion matrix of the classification result is shown below, with the error rate  $30/600 = 0.045$ :

$$\begin{bmatrix} 197 & 3 & 0 \\ 0 & 195 & 5 \\ 1 & 21 & 178 \end{bmatrix} \quad (7.74)$$

#### Example 2:

This example shows the classification of the XOR data set. The confusion matrix of the classification result is shown below, with the error rate  $13/400 = 0.0325$ :

$$\begin{bmatrix} 193 & 7 \\ 6 & 194 \end{bmatrix} \quad (7.75)$$

We see that in both examples, the error rates of the GPC method are lower than those of the naive Bayesian method. However, the naive Bayesian method does not have the overfitting problem, while in the method of GPC, we may need to carefully adjust the parameter of squared exponential for the kernel functions to make proper tradeoff between overfitting and error rate.

## **Part III**

---

# **Feature Selection**





# 8 Feature Selection

---

## Feature Selection

### 8.1 Dimensionality Reduction

The main goal of pattern classification is to classify a set of *patterns*, objects of interest each described by  $d$  *features* represented as a  $d$ -dimensional vector  $\mathbf{x} = [x_1, \dots, x_d]^T$  in the *feature space*, into one of some  $K$  classes  $\{C_1, \dots, C_K\}$ . In supervised classification, we are to partition the feature space into  $K$  regions each corresponding to one of the classes based on the provided training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ , of which each training sample in  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  is labeled by the corresponding component in  $\mathbf{y} = [y_1, \dots, y_N]^T$ . For example, if  $y_n = \{1, 2, \dots, K\}$ , then  $\mathbf{x}_n \in C_{y_n}$ . However, different types of labeling can be used depending on the specific classification algorithms, such as binary labeling used in certain regression algorithms considered previously.

When the dimensionality  $d$  of the feature space is high, there is the need to reduce it significantly to  $d' \ll d$  while still maintaining most of the information relevant to the task of classification, i.e., the separability of the classes. This is called *feature selection*, which can be done by either selecting  $d'$  features directly from the  $d$  original ones (there are  $C_d^{d'} = d!/d!(d-d)!$  ways to do so), or generating  $d'$  new features as certain linear combinations of the  $d$  original ones. In either case, the separability of the data are to be maximally maintained in the resulting space spanned by the  $d'$  features, so that the classification can be carried out both efficiently and effectively in the much lower dimensional feature space.

### 8.2 Distances and Separability Measurements

We first consider various distances widely used for feature selection in pattern classification.

- **Distance between two data points**

The distance between two points  $\mathbf{x}$  and  $\mathbf{y}$  in the  $n$ -dimensional feature

space can be measured by the p-norm of the difference  $\mathbf{x} - \mathbf{y}$ :

$$d_p(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p} \quad 1 \leq p \leq \infty \quad (8.1)$$

Specially, consider the following three cases:

–  $p = 1$ , *city block or Manhattan distance*:

$$d_1(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_1 = \sum_{i=1}^n |x_i - y_i| \quad (8.2)$$

–  $p = 2$ , *Euclidean distance*:

$$d_2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_{i=1}^n |x_i - y_i|^2} \quad (8.3)$$

–  $p = \infty$ , the *Chebychev distance*:

$$d_\infty(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_\infty = \max\{|x_1|, \dots, |x_n|\} \quad (8.4)$$

#### • Intra-class distance

An intra-class distance represents how widely (or narrowly) all samples in each class  $C_k$  are distributed in the space. It needs to be small for good separability.

– The *max diameter*:

$$d_{max}(C_k) = \max_{\mathbf{x}, \mathbf{y} \in C_k} d(\mathbf{x}, \mathbf{y}) \quad (8.5)$$

– The *average diameter*:

$$d_{average}(C_k) = \frac{1}{N_k(N_k - 1)} \sum_{\mathbf{x}, \mathbf{y} \in C_k, \mathbf{x} \neq \mathbf{y}} d(\mathbf{x}, \mathbf{y}) \quad (8.6)$$

– The average distance from each sample to the mean vector of the cluster:

$$\frac{1}{N_k} \sum_{\mathbf{x} \in C_k} d(\mathbf{x}, \mathbf{m}_k), \quad \text{where} \quad \mathbf{m}_k = \frac{1}{N_k} \sum_{\mathbf{x} \in C_k} \mathbf{x} \quad (8.7)$$

– The covariance of all samples in the class represents the tightness of the samples in the class:

$$\Sigma_k = \frac{1}{N_k} \sum_{\mathbf{x} \in C_k} (\mathbf{x} - \mathbf{m}_k)(\mathbf{x} - \mathbf{m}_k)^T \quad (8.8)$$

and its determinant or trace can be used as such a scalar measurement of the tightness:

$$\det \Sigma_k = \prod_{i=1}^n \lambda_i, \quad tr \Sigma_k = \sum_{i=1}^n \lambda_i \quad (8.9)$$

As  $\Sigma_k$  is positive semi-definite, all of its eigenvalues  $\lambda_i$  are non-negative and so are its determinant and trace.

- **Distance between a point and a cluster/class of points**

- The *max and min-distances*:

$$d_{max}(\mathbf{x}, C_k) = \max_{\mathbf{y} \in C_k} d(\mathbf{x}, \mathbf{y}), \quad d_{min}(\mathbf{x}, C_k) = \min_{\mathbf{y} \in C_k} d(\mathbf{x}, \mathbf{y}) \quad (8.10)$$

- The *centroid distance*:

$$d_{centroid}(\mathbf{x}, C_k) = d(\mathbf{x}, \mathbf{m}_k) \quad (8.11)$$

- The *Mahalanobis distance*

$$d_M(\mathbf{x}, C_k) = (\mathbf{x} - \mathbf{m}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \mathbf{m}_k) \quad (8.12)$$

- **Inter-class distance**

An intra-class distance measures the difference between two classes, it needs to be large for good separability.

- The *max- and min-distances*:

$$d_{max}(C_i, C_j) = \max_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}), \quad d_{min}(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}) \quad (8.13)$$

- The *average distance* between two clusters is the average of all pair-wise distances (e.g., Euclidean) between members of the two classes:

$$d_{average}(C_i, C_j) = \frac{1}{N_i N_j} \sum_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}) \quad (8.14)$$

- The *centroid distance* is the distance (e.g., Euclidean) between the centroids (mean vectors) of the two classes:

$$d_{centroid}(C_i, C_j) = d(\mathbf{m}_i, \mathbf{m}_j) \quad (8.15)$$

- The *Bhattacharyya distance*:

$$d_B(C_i, C_j) = \frac{1}{4} (\mathbf{m}_i - \mathbf{m}_j)^T \left( \frac{\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j}{2} \right)^{-1} (\mathbf{m}_i - \mathbf{m}_j) + \log \left[ \frac{|(\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j)/2|}{(|\boldsymbol{\Sigma}_i| |\boldsymbol{\Sigma}_j|)^{1/2}} \right] \quad (8.16)$$

The first term reflects mostly the difference between the two mean vectors of the two clusters, while the second term reflects the difference between the distributions of the two clusters, which is always positive due to the *AM-GM inequality* (algebraic mean vs geometric mean) of the arithmetic and geometric means:

$$\frac{1}{n} \sum_{i=1}^n x_i \geq \left( \prod_{i=1}^n x_i \right)^{1/n} \quad (8.17)$$

Note that even when the first term is zero due to  $\mathbf{m}_i = \mathbf{m}_j$ , the Bhattacharyya distance may still not zero due to the non-zero second term if  $\boldsymbol{\Sigma}_i \neq \boldsymbol{\Sigma}_j$ .

We also define a set of scatter matrices all related to the *separability* of the classes/clusters in the feature space, based on which different subsets of features can be compared and the best subset can be selected. We will also consider methods for *feature selection* or *feature extraction* to reduce the dimensionality of the feature space. Here we assume there are  $K$  classes (or clusters) in the dataset containing  $N$  data points, and the  $k$ th class contains  $N_k$  data points, i.e.,  $N_1 + \dots + N_K = N$ .

- Total scatter matrix:

The total scatter matrix is the same as the covariance matrix:

$$\mathbf{S}_T = \frac{1}{N} \sum_{\text{all } \mathbf{x}} (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T = \frac{1}{N} \sum_{k=1}^K \sum_{\mathbf{x} \in C_k} (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T \quad (8.18)$$

Note that  $\mathbf{S}_T \text{rank}(\mathbf{S}_T) \leq N - 1$ , due to the mean  $\mathbf{m} = \sum_{\text{all } \mathbf{x}} \mathbf{x} / N$  as a constraint.

- Within-class (intra-class) scatter matrix:

$$\mathbf{S}_W = \sum_{k=1}^K P_k \mathbf{\Sigma}_k = \sum_{k=1}^K \frac{N_k}{N} \mathbf{\Sigma}_k = \frac{1}{N} \sum_{k=1}^K \sum_{\mathbf{x} \in C_k} (\mathbf{x} - \mathbf{m}_k)(\mathbf{x} - \mathbf{m}_k)^T \quad (8.19)$$

Note that  $\text{rank}(\mathbf{S}_W) = \sum_{k=1}^K (N_k - 1) \leq N - K$ , due to the  $K$  means  $\mathbf{m}_k = \sum_{\mathbf{x} \in C_k} \mathbf{x} / N_k$ , ( $k = 1, \dots, K$ ) as  $K$  constraints.

- Between-class (inter-class) scatter matrix:

$$\mathbf{S}_B = \sum_{k=1}^K P_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T = \sum_{k=1}^K \frac{N_k}{N} (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T \quad (8.20)$$

Note that  $\text{rank}(\mathbf{S}_B) \leq K - 1$ , due to total mean  $\mathbf{m} = \sum_{k=1}^K N_k \mathbf{m}_k / N$  as a constraint.

We can show that  $\mathbf{S}_T = \mathbf{S}_W + \mathbf{S}_B$ , i.e., the total scatter matrix is the sum of within-class and between-class scatter matrices:

$$\begin{aligned} \mathbf{S}_T &= \frac{1}{N} \sum_{k=1}^K \sum_{\mathbf{x} \in C_k} (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T = \frac{1}{N} \sum_{k=1}^K \sum_{\mathbf{x} \in C_k} (\mathbf{x} - \mathbf{m}_k + \mathbf{m}_k - \mathbf{m})(\mathbf{x} - \mathbf{m}_k + \mathbf{m}_k - \mathbf{m})^T \\ &= \frac{1}{N} \sum_{k=1}^K \sum_{\mathbf{x} \in C_k} [(\mathbf{x} - \mathbf{m}_k)(\mathbf{x} - \mathbf{m}_k)^T + (\mathbf{x} - \mathbf{m}_k)(\mathbf{m}_k - \mathbf{m})^T + (\mathbf{m}_k - \mathbf{m})(\mathbf{x} - \mathbf{m}_k)^T + (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T] \\ &= \frac{1}{N} \sum_{k=1}^K \sum_{\mathbf{x} \in C_k} (\mathbf{x} - \mathbf{m}_k)(\mathbf{x} - \mathbf{m}_k)^T + \frac{1}{K} \sum_{k=1}^K \sum_{\mathbf{x} \in C_k} (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T \\ &= \sum_{k=1}^K \frac{N_k}{N} \mathbf{\Sigma}_k + \sum_{k=1}^K \frac{N_k}{N} (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T = \mathbf{S}_W + \mathbf{S}_B \end{aligned}$$

Obviously, for better separability, we want the within-class scatteredness to be small but the between-class scatteredness to be large. However, we cannot

use the scatter matrices directly as they cannot be compared in terms of their sizes. Instead, we use their traces (or determinants) as the scalar measurements for the separability:

$$\begin{aligned} J_B &= \text{tr } \mathbf{S}_B = \text{tr} \left[ \sum_{k=1}^C P_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T \right] \\ &= \sum_{i=1}^C P_k \text{tr} [(\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T] = \sum_{i=1}^C P_k \|\mathbf{m}_k - \mathbf{m}\|^2 \end{aligned} \quad (8.22)$$

and

$$\begin{aligned} J_W &= \text{tr } \mathbf{S}_W = \text{tr} \left[ \sum_{k=1}^K P_k \sum_{\mathbf{x} \in C_k} (\mathbf{x} - \mathbf{m}_k)(\mathbf{x} - \mathbf{m}_k)^T \right] \\ &= \sum_{k=1}^K P_k \sum_{\mathbf{x} \in C_k} \text{tr} [(\mathbf{x} - \mathbf{m}_k)(\mathbf{x} - \mathbf{m}_k)^T] = \sum_{k=1}^K P_k \sum_{\mathbf{x} \in C_k} \|\mathbf{x} - \mathbf{m}_k\|^2 \end{aligned} \quad (8.23)$$

We see that  $J_B$  is the weighted sum of the Euclidean distances between  $\mathbf{m}_k$  and  $\mathbf{m}$  for all  $C$  classes, and  $J_W$  is the weighted sum of the sums of Euclidean distances between all samples  $\mathbf{x} \in C_k$  to the mean  $\mathbf{m}_k$  for all  $K$  classes  $C_1, \dots, C_K$ .

To achieve the best separability, we maximize  $\text{tr} \mathbf{S}_B$  while at the same time minimize  $\text{tr} \mathbf{S}_W$ . In the same space with a constant  $\mathbf{S}_T$ , maximizing  $\text{tr} \mathbf{S}_B$  is equivalent to minimizing  $\text{tr} \mathbf{S}_W$ , due to the relationship  $\mathbf{S}_B + \mathbf{S}_W = \mathbf{S}_T$ . To measure the separability across different spaces,  $\mathbf{S}_B$  can still be used but now it needs to be normalized by either  $\mathbf{S}_T$  or equivalently  $\mathbf{S}_W$ :

$$J_{B/W} = \text{tr}(\mathbf{S}_W^{-1} \mathbf{S}_B) = \text{tr}(\mathbf{S}_B \mathbf{S}_W^{-1}), \quad J_{B/T} = \text{tr}(\mathbf{S}_T^{-1} \mathbf{S}_B) = \text{tr}(\mathbf{S}_B \mathbf{S}_T^{-1}) \quad (8.24)$$

The separability measured by the scatter matrices can be used as the criterion for selecting  $m < n$  of the  $n$  original features to reduce the computational cost while still keeping the effectiveness of the classification. Moreover, we can also generate  $m$  new features as the linear combination of the  $n$  original ones by a linear transform:

$$\mathbf{y}_{m \times 1} = (\mathbf{A}_{n \times m})^T \mathbf{x}_{n \times 1} \quad (8.25)$$

We desire to find the  $n \times m$  transform matrix  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_m]$  so that the separability in the original  $n$ -D space is maximally conserved in the new  $m$ -D space of lower dimensionality.

- **Optimal transformation for maximizing  $\text{tr}(\mathbf{S}_B)$**

The separability criterion  $\text{tr}(\mathbf{S}_B^{(y)})$  in new  $m$ -D space can be expressed

as:

$$\begin{aligned} J^{(y)}(\mathbf{A}) &= \text{tr}(\mathbf{S}_B^{(y)}) = \text{tr}(\mathbf{A}^T \mathbf{S}_B^x \mathbf{A}) = \text{tr} \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_m^T \end{bmatrix} \mathbf{S}_B^x [\mathbf{a}_1, \dots, \mathbf{a}_m] \\ &= \text{tr} \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_m^T \end{bmatrix} [\mathbf{S}_B^x \mathbf{a}_1, \dots, \mathbf{S}_B^x \mathbf{a}_m] = \sum_{i=1}^m \mathbf{a}_i^T \mathbf{S}_B^x \mathbf{a}_i \end{aligned} \quad (8.26)$$

To find the optimal transform matrix  $\mathbf{A}$  that maximizes  $\text{tr}(\mathbf{S}_B^{(y)})$ , we consider the following constrained maximization problem:

$$\begin{cases} \text{maximize} & J(\mathbf{A}) = \text{tr}(\mathbf{A}^T \mathbf{S}_B^x \mathbf{A}) = \sum_{i=1}^m \mathbf{a}_i^T \mathbf{S}_B^x \mathbf{a}_i \\ \text{subject to} & \|\mathbf{a}_j\|^2 = \mathbf{a}_j^T \mathbf{a}_j = 1 \quad (j = 1, \dots, m) \end{cases} \quad (8.27)$$

Here the constraint is to guarantee that the column vectors of  $\mathbf{A}$  are normalized. Same as the PCA problem discussed in Section 3.5, this constrained optimization problem can be solved by Lagrange multiplier method:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{a}_i} \left[ J(\mathbf{A}) - \sum_{j=1}^m \lambda_j (\mathbf{a}_j^T \mathbf{a}_j - 1) \right] &= \frac{\partial}{\partial \mathbf{a}_i} \left[ \sum_{j=1}^m (\mathbf{a}_j^T \mathbf{S}_B^x \mathbf{a}_j - \lambda_j \mathbf{a}_j^T \mathbf{a}_j + \lambda_j) \right] \\ &= \frac{\partial}{\partial \mathbf{a}_i} [\mathbf{a}_i^T \mathbf{S}_B^x \mathbf{a}_i - \lambda_i \mathbf{a}_i^T \mathbf{a}_i] = 2\mathbf{S}_B^x \mathbf{a}_i - 2\lambda_i \mathbf{a}_i = 0, \quad \text{i.e. } \mathbf{S}_B \mathbf{a}_i = \lambda_i \mathbf{a}_i \quad (i = 1, \dots, m) \end{aligned} \quad (8.28)$$

We see that the optimal feature selection transform is the PCA transform which compacts most of the energy/information (representing separability in this context) into  $m$  components. The column vectors of  $\mathbf{A}$  must be the orthogonal eigenvectors of the symmetric matrix  $\mathbf{S}_B^x$  corresponding to the  $m$  greatest eigenvalues, and the  $m$  new features can be obtained by

$$\mathbf{y}_{m \times 1} = \mathbf{A}_{m \times n}^T \mathbf{x}_{n \times 1} = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_m^T \end{bmatrix}_{m \times n} \mathbf{x}_{n \times 1}, \quad \text{or } y_i = \phi_i^T \mathbf{x}, \quad (i = 1, \dots, m) \quad (8.29)$$

and

$$J(\mathbf{A}) = J(\Phi) = \sum_{i=1}^m \phi_i^T \mathbf{S}_B \phi_i = \sum_{i=1}^m \lambda_i \quad (8.30)$$

where the eigenvalues of  $\mathbf{S}_B$  are sorted in descending order  $\lambda_1 \geq \dots \geq \lambda_m \geq \dots \geq \lambda_n$ .

- **Optimal transformation for maximizing  $\text{tr}(\mathbf{S}_{B/T})$**

The previous method only maximizes  $\mathbf{S}_B$  without taking into consideration  $\mathbf{S}_W$  or, equivalently,  $\mathbf{S}_T = \mathbf{S}_B + \mathbf{S}_W$ . If in the  $m$ -D space after the transform  $\mathbf{S}_W^{(y)}$  has also changed as well as  $\mathbf{S}_B^{(y)}$ , then we need to maximize  $\mathbf{S}_B$  while at the same time also minimize  $\mathbf{S}_W$ , in order to maximize

the separability. Or, equivalently, we need to maximize  $\mathbf{S}_B^{(y)}$  normalized by the total scatteredness  $\mathbf{S}_T^{(y)}$ . We can therefore use the trace of  $\mathbf{S}_T^{-1}\mathbf{S}_B$  (or  $\mathbf{S}_W^{-1}\mathbf{S}_B$ ) as the measurement of the separability in the new m-D space:

$$\begin{aligned}\mathbf{S}_{B/T}^{(y)} &= (\mathbf{S}_T^{(y)})^{-1} \mathbf{S}_B^{(y)} = (\mathbf{A}^T \mathbf{S}_T^x \mathbf{A})^{-1} (\mathbf{A}^T \mathbf{S}_B^x \mathbf{A}) \\ &= \mathbf{A}^{-1} (\mathbf{S}_T^x)^{-1} (\mathbf{A}^T)^{-1} \mathbf{A}^T \mathbf{S}_B^x \mathbf{A} = \mathbf{A}^{-1} (\mathbf{S}_T^x)^{-1} \mathbf{S}_B^x \mathbf{A} = \mathbf{A}^{-1} \mathbf{S}_B^x \mathbf{A}\end{aligned}\quad (8.31)$$

We need to find  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_m]$  that maximizes the trace of the matrix above in the m-D space of  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$ . As  $\mathbf{S}_{B/T}^{(y)}$  (a product of two symmetric matrices) is not a symmetric matrix, the PCA method above cannot be used, we will find the optimal matrix  $\mathbf{A}$  in some other way.

We first let  $m = 1$ , so that  $\mathbf{A} = \mathbf{a}$  is an  $n \times 1$  vector that maximizes the following objective function in the 1-D space  $y = \mathbf{a}^T \mathbf{x}$ :

$$(\mathbf{S}_T^{(y)})^{-1} \mathbf{S}_B^{(y)} = (\mathbf{a}^T \mathbf{S}_T^x \mathbf{a})^{-1} (\mathbf{a}^T \mathbf{S}_B^x \mathbf{a}) = \frac{s_B^{(y)}}{s_T^{(y)}} = \frac{\mathbf{a}^T \mathbf{S}_B^x \mathbf{a}}{\mathbf{a}^T \mathbf{S}_T^x \mathbf{a}} = R(\mathbf{a}) \quad (8.32)$$

This function  $R(\mathbf{a})$  is the *Rayleigh quotient* (Section A.11) of the two symmetric matrices  $\mathbf{S}_B^x$  and  $\mathbf{S}_T^x$ . The optimal transform vector  $\mathbf{a}$  that maximizes this  $R(\mathbf{a})$  can be found by solving the corresponding *generalized eigenvalue problem* (Section A.10):

$$\mathbf{S}_B^x \phi_i = \lambda_i \mathbf{S}_T^x \phi_i, \quad (i = 1, \dots, n) \quad (8.33)$$

where  $\lambda_i = R(\mathbf{a})$  is an eigenvalue and  $\phi_i$  the corresponding eigenvector of  $\mathbf{S}_{B/T} = (\mathbf{S}_T^x)^{-1} \mathbf{S}_B^x$ . Obviously, the transform vector  $\mathbf{a}$  that maximizes  $R(\mathbf{a})$  is the eigenvector corresponding to the greatest eigenvalue  $\lambda_1 = \max\{\lambda_i (i = 1, \dots, n)\}$ .

We next generalize the method above to the case of  $m > 1$ . The matrix form of the eigenequation above is:

$$\mathbf{S}_B^x \Phi = \mathbf{S}_T^x \Phi \Lambda \quad (8.34)$$

where  $\Lambda$  is the eigenvalue matrix of  $(\mathbf{S}_T^x)^{-1} \mathbf{S}_B^x$ , and  $\Phi = [\phi_1, \dots, \phi_n]$  the eigenvector matrix (no longer orthogonal in general). This generalized eigenvalue problem can be solved by finding the matrix  $\Phi$  that diagonalizes both  $\mathbf{S}_T^x$  and  $\mathbf{S}_B^x$  at the same time:

$$\begin{cases} \mathbf{S}_B^x = \Phi^T \mathbf{S}_B^x \Phi = \Lambda \\ \mathbf{S}_T^x = \Phi^T \mathbf{S}_T^x \Phi = \mathbf{I} \end{cases} \quad (8.35)$$

Left multiplying the inverse of the second equation to the first, we get

$$\mathbf{S}_{B/T}^{(y)} = (\mathbf{S}_T^x)^{-1} \mathbf{S}_B^x = \Lambda \quad (8.36)$$

The optimal transform matrix is composed of the eigenvectors  $\mathbf{A} = [\phi_1, \dots, \phi_m]$  corresponding to the  $m$  greatest of all  $n$  eigenvalues  $\lambda_1 \geq \dots \geq \lambda_m \geq \dots \geq \lambda_n$ , so that

- the signal components in  $\mathbf{y} = [y_1, \dots, y_m]^T$  are completely decorrelated, each component  $y_i = \phi_i^T \mathbf{x}$  carries certain separability information  $s_B^{(y_i)} / s_T^{(y_i)} = \lambda_i$  ( $i = 1, \dots, m$ ), independent of others;
- the total separability contained in the m-D space, as the sum of the  $m$  greatest eigenvalues  $\sum_{i=1}^m \lambda_i$ , is maximized.



# 9 Principal Component Analysis

## Principal Component Analysis

### 9.1 Covariance and Correlation

In *principal component analysis (PCA)*, each pattern  $\mathbf{x}$  is treated as a random vector of which each component  $x_i$  is a random variable with mean and variance

$$\begin{aligned}\mu_i &= E[\mathbf{x}_i] = \int x_i p(x_i) dx_i \\ \sigma_i^2 &= E[(x_i - \mu_i)^2] = E[x_i^2] - \mu_i^2 = \int x_i^2 p(x_i) dx_i - \mu_i^2, \quad (i = 1, \dots, d)\end{aligned}\quad (9.1)$$

and the *covariance* of  $x_i$  and  $x_j$  is:

$$\begin{aligned}\sigma_{ij}^2 &= E[(x_i - \mu_i)(x_j - \mu_j)] = E[x_i x_j] - \mu_i \mu_j \\ &= \int \int x_i x_j p(x_i, x_j) dx_i dx_j - \mu_i \mu_j \quad (i, j = 1, \dots, d)\end{aligned}\quad (9.2)$$

then the mean vector and covariance matrix of  $\mathbf{x}$  are:

$$\begin{aligned}\mathbf{m}_x &= E[\mathbf{x}] = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_d \end{bmatrix} \\ \Sigma_x &= E[(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^T] = E[\mathbf{x}\mathbf{x}^T] - \mathbf{m}_x \mathbf{m}_x^T = \begin{bmatrix} \sigma_{11}^2 & \cdots & \sigma_{1d}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{d1}^2 & \cdots & \sigma_{dd}^2 \end{bmatrix}\end{aligned}\quad (9.3)$$

Usually the joint probability density function  $p(\mathbf{x}) = p(x_1, \dots, x_d)$  of the random vector  $\mathbf{x}$  is unknown. In this case, the mean vector  $\mathbf{m}_x$  and covariance matrix  $\Sigma_x$  of  $\mathbf{x}$  can be estimated by the method of *maximum likelihood estimation (MLE)* (Section 3.3.2) based on a set of observed data samples  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ :

$$\hat{\mathbf{m}}_x = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n, \quad \hat{\Sigma} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \hat{\mathbf{m}})(\mathbf{x}_n - \hat{\mathbf{m}})^T = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T - \hat{\mathbf{m}} \hat{\mathbf{m}}^T \quad (9.4)$$

Note that the rank of the  $d \times d$  estimated covariance matrix  $\hat{\Sigma}$  is at most

$N - 1$ , due to the  $N$  samples in the dataset  $\mathbf{X}$ , assumed to be independent, and the additional constraint:

$$\sum_{n=1}^N (\mathbf{x}_n - \hat{\mathbf{m}}) = \sum_{n=1}^N \mathbf{x}_n - N\hat{\mathbf{m}} = \mathbf{0} \quad (9.5)$$

The variance  $\sigma_i^2 = E[(x_i - \mu_{x_i})^2]$  can be treated as the dynamic energy contained in  $x_i$ , or the amount of information carried by  $x_i$ , while the trace  $\text{tr}\Sigma_x = \sum_{i=1}^d \sigma_i^2$  can be considered as the total amount of dynamic energy contained in  $\mathbf{x}$ . Also, the covariance  $\sigma_{ij}^2 = E[(x_i - \mu_{x_i})(x_j - \mu_{x_j})]$  can be considered as the *mutual energy*, a measure of the correlation between  $x_i$  and  $x_j$ . By normalizing the covariance  $\sigma_{ij}^2$ , we get the *correlation coefficient* between  $x_i$  and  $x_j$ :

$$r_{ij} = \frac{\sigma_{ij}^2}{\sqrt{\sigma_i^2 \sigma_j^2}} = \frac{\sigma_{ij}^2}{\sigma_i \sigma_j} \quad (9.6)$$

that measures how the two random variables  $x_i$  and  $x_j$  are correlated.

- $r_{ij} = \pm 1$  or  $|r_{ij}| = 1$ :  $x_i$  and  $x_j$  are maximally correlated. The information contained in the two variables is completely redundant, given the value of one of them, the value of the other is known.
- $-1 < r_{ij} < 1$  or  $|r_{ij}| < 1$ :  $x_i$  and  $x_j$  are correlated to different extents. The information they each carry has certain redundancy.
- $r_{ij} = 0$ :  $x_i$  and  $x_j$  are uncorrelated. They each carry their own independent information with no redundancy.

We see that the correlation coefficient  $r_{ij}$  that measures how much two random variables are correlated also measures the redundancy of the information they carry. When there exists some data redundancy in the data, it is possible to carry out data compression by some method such as the principal component analysis based on the covariance  $\Sigma$  to reduce the data redundancy, so that the data size can be significantly reduced while the information (dynamic energy) contained in the data is still mostly preserved.

#### Examples

Six normally distributed 2-D datasets are generated with zero mean and the following covariance matrices:

$$\Sigma_1 = \begin{bmatrix} 1.0 & 0.95 \\ 0.95 & 1 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1.0 & -0.95 \\ -0.95 & 1.0 \end{bmatrix}, \quad \Sigma_3 = \begin{bmatrix} 1.0 & 0.7 \\ 0.7 & 1.0 \end{bmatrix}$$

$$\Sigma_4 = \begin{bmatrix} 1.0 & -0.7 \\ -0.7 & 1.0 \end{bmatrix}, \quad \Sigma_5 = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}, \quad \Sigma_6 = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}$$

These data points are plotted below, together with the correlation coefficient on top of each dataset.

## 9.2 Karhunen-Loeve Transformation

A data point in a  $d$ -dimensional space is represented by a vector  $\mathbf{x} = [x_1, \dots, x_d]^T$ , of which the components are the coordinates along the  $d$  standard orthonormal basis vectors  $\{\mathbf{e}_1, \dots, \mathbf{e}_d\}$  that span the space:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} + \dots + x_d \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} = \sum_{i=1}^d x_i \mathbf{e}_i \\ &= \begin{bmatrix} \mathbf{e}_1 & \dots & \mathbf{e}_d \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} = \mathbf{I} \mathbf{x} \end{aligned} \quad (9.7)$$

where  $\mathbf{I} = [\mathbf{e}_1, \dots, \mathbf{e}_d]$  is the identity matrix.

The space can also be spanned by any other orthonormal basis  $\{\mathbf{a}_1, \dots, \mathbf{a}_d\}$  satisfying

$$\mathbf{a}_i^T \mathbf{a}_j = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad (9.8)$$

and the same vector  $\mathbf{x}$  can also be represented as a weighted vector sum of these basis vectors:

$$\mathbf{x} = \sum_{i=1}^d y_i \mathbf{a}_i = \begin{bmatrix} \mathbf{a}_1 & \dots & \mathbf{a}_d \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix} = \mathbf{A} \mathbf{y} \quad (9.9)$$

where  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_d]$  composed of  $d$  orthonormal column vectors is an orthogonal matrix satisfying  $\mathbf{A} \mathbf{A}^T = \mathbf{A}^T \mathbf{A} = \mathbf{I}$ , and  $\mathbf{y} = [y_1, \dots, y_d]^T$  is a vector containing all  $d$  coordinates in the directions of the basis vector  $\{\mathbf{a}_1, \dots, \mathbf{a}_d\}$ , which can be found by premultiplying  $\mathbf{A}^T = \mathbf{A}^{-1}$  on both sides of the equation above  $\mathbf{A} \mathbf{y} = \mathbf{x}$ :

$$\mathbf{A}^T \mathbf{A} \mathbf{y} = \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix} = \mathbf{A}^T \mathbf{x} = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_d^T \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{a}_1^T \mathbf{x} \\ \vdots \\ \mathbf{a}_d^T \mathbf{x} \end{bmatrix} \quad (9.10)$$

of which the  $i$ th coordinate  $y_i = \mathbf{a}_i^T \mathbf{x}$  is the projection of  $\mathbf{x}$  onto the  $i$ th basis vector  $\mathbf{a}_i$ .

The basis vectors in  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_d]$  can be considered as a rotated version of the standard basis in  $\mathbf{I} = [\mathbf{e}_1, \dots, \mathbf{e}_d]$ , and the norm or length of  $\mathbf{x}$  before and after the transform remain the same:

$$\|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x} = \left( \sum_{i=1}^d y_i \mathbf{a}_i \right)^T \left( \sum_{j=1}^d y_j \mathbf{a}_j \right) = \sum_{i=1}^d \sum_{j=1}^d y_i y_j \mathbf{a}_i^T \mathbf{a}_j = \sum_{i=1}^d y_i^2 = \|\mathbf{y}\|^2 \quad (9.11)$$

This identity can be considered as the generalized version of Parseval's identity in the Fourier transform, as a special form of unitary transform (complex version of orthogonal transform).

Summarizing the above, we can define an orthogonal transform based on any orthogonal matrix  $\mathbf{A}$ :

$$\begin{cases} \mathbf{y} = \mathbf{A}^T \mathbf{x} & \text{forward transform} \\ \mathbf{x} = \mathbf{A} \mathbf{y} & \text{inverse transform} \end{cases} \quad (9.12)$$

by which any vector  $\mathbf{x}$  under the implicit standard basis, the column vectors of  $\mathbf{I} = [\mathbf{e}_1, \dots, \mathbf{e}_d]$ , is transformed into  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$  under the explicit basis, the column vectors of  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_d]$ .

Any orthogonal transform  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$  is actually a rotation of the standard basis  $\{\mathbf{e}_1, \dots, \mathbf{e}_d\}$  into another orthonormal basis  $\{\mathbf{a}_1, \dots, \mathbf{a}_d\}$  spanning the same space, while  $\mathbf{x}$  and  $\mathbf{y}$  are just the coordinates or coefficients of the same vector under these two different coordinate systems.

If  $\mathbf{x}$  is treated as a random vector, then the linear transform  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$ , is also a random vector, and its mean vector and covariance can be found as:

$$\begin{aligned} \mathbf{m}_y &= E[\mathbf{y}] = E[\mathbf{A}^T \mathbf{x}] = \mathbf{A}^T E[\mathbf{x}] = \mathbf{A}^T \mathbf{m}_x \\ \boldsymbol{\Sigma}_y &= E[\mathbf{y}\mathbf{y}^T] - \mathbf{m}_y \mathbf{m}_y^T = E[\mathbf{A}^T \mathbf{x} \mathbf{x}^T \mathbf{A}] - \mathbf{A}^T \mathbf{m}_x \mathbf{m}_x^T \mathbf{A} \\ &= \mathbf{A}^T [E[\mathbf{x}\mathbf{x}^T] - \mathbf{m}_x \mathbf{m}_x^T] \mathbf{A} = \mathbf{A}^T \boldsymbol{\Sigma}_x \mathbf{A} \end{aligned} \quad (9.13)$$

In particular, the *Karhunen-Loeve Transform (KLT)* is just one of such orthogonal transforms in the form of  $\mathbf{y} = \mathbf{V}^T \mathbf{x}$ , where the orthogonal transform matrix  $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_d]$  is the eigenvector matrix of the covariance matrix  $\boldsymbol{\Sigma}_x$  of  $\mathbf{x}$ , composed of the  $d$  normalized eigenvectors of  $\boldsymbol{\Sigma}_x$ . As in general the covariance matrix  $\boldsymbol{\Sigma}_x$  is symmetric and positive definite, its eigenvalues  $\{\lambda_1, \dots, \lambda_d\}$  are real and positive, and its eigenvectors are orthogonal, i.e., its eigenvector matrix  $\mathbf{V}$  is indeed an orthogonal matrix satisfying  $\mathbf{V}^T = \mathbf{V}^{-1}$  or  $\mathbf{V}^T \mathbf{V} = \mathbf{I}$ .

The  $d$  eigenvalues  $\{\lambda_1, \dots, \lambda_d\}$  and the corresponding eigenvectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_d\}$  can then be found by solving the eigenequations:

$$\boldsymbol{\Sigma}_x \mathbf{v}_i = \lambda_i \mathbf{v}_i \quad (i = 1, \dots, d) \quad (9.14)$$

which can be combined to be expressed in matrix form:

$$\boldsymbol{\Sigma}_x \mathbf{V} = \boldsymbol{\Sigma}_x [\mathbf{v}_1, \dots, \mathbf{v}_d] = [\mathbf{v}_1, \dots, \mathbf{v}_d] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_d \end{bmatrix} = \mathbf{V} \boldsymbol{\Lambda} \quad (9.15)$$

where  $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_d)$  is a diagonal matrix. Premultiplying  $\mathbf{V}^T = \mathbf{V}^{-1}$  on both sides, we get

$$\mathbf{V}^T \boldsymbol{\Sigma}_x \mathbf{V} = \mathbf{V}^{-1} \mathbf{V} \boldsymbol{\Lambda} = \boldsymbol{\Lambda} \quad (9.16)$$

Taking inverse on both sides, we also get

$$(\mathbf{V}^T \boldsymbol{\Sigma}_x \mathbf{V})^{-1} = \mathbf{V}^{-1} \boldsymbol{\Sigma}_x^{-1} (\mathbf{V}^T)^{-1} = \mathbf{V}^T \boldsymbol{\Sigma}_x^{-1} \mathbf{V} = \boldsymbol{\Lambda}^{-1} \quad (9.17)$$

We see that both  $\Sigma_x$  and its inverse  $\Sigma_x^{-1}$  are diagonalized by the orthogonal eigenvector matrix  $\mathbf{V}$  to become  $\Lambda$  and  $\Lambda^{-1}$  respectively.

Based on the orthogonal eigenvector matrix  $\mathbf{V}$ , the KLT is defined as:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix} = \mathbf{V}^T \mathbf{x} = \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_d^T \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{v}_1^T \mathbf{x} \\ \vdots \\ \mathbf{v}_d^T \mathbf{x} \end{bmatrix} \quad (9.18)$$

of which the  $i$ th component  $y_i = \mathbf{v}_i^T \mathbf{x}$  is the projection of  $\mathbf{x}$  onto the  $i$ th eigenvector  $\mathbf{v}_i$ . Premultiplying  $\mathbf{V} = (\mathbf{V}^T)^{-1}$  on both sides of the forward transform  $\mathbf{y} = \mathbf{V}^T \mathbf{x}$ , we get the inverse KLT transform, by which vector  $\mathbf{x}$  is represented as a linear combination of the eigenvectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_d\}$  as the basis vectors:

$$\mathbf{x} = \mathbf{V} \mathbf{y} = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_d \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix} = \sum_{i=1}^d y_i \mathbf{v}_i \quad (9.19)$$

**Example:**

$$\Sigma_1 = \begin{bmatrix} 2.964 & 0.544 \\ 0.544 & 4.687 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 5.036 & 3.083 \\ 3.083 & 2.830 \end{bmatrix}, \quad \Sigma_3 = \begin{bmatrix} 3.043 & -3.076 \\ -3.076 & 3.930 \end{bmatrix} \quad (9.20)$$

$$r_1 = 0.146, \quad r_2 = 0.817, \quad r_3 = -0.889 \quad (9.21)$$

$$\lambda_{1,2} = (4.845, 2.807), \quad \lambda_{1,2} = (7.207, 0.659), \quad \lambda_{1,2} = (6.594, 0.379) \quad (9.22)$$

### 9.3 Optimality of KLT

The KLT is the optimal orthogonal transform among all possible orthogonal transforms  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$  based on any orthogonal matrix  $\mathbf{A}$  satisfying  $\mathbf{A}^T \mathbf{A} = \mathbf{I}$ , in the following sense:

- KLT completely decorrelates the signal.
- KLT maximally compacts the energy (information) contained in the signal.

#### KLT Completely Decorrelates the Signal

The mean vector and covariance matrix of the random vector  $\mathbf{y} = \mathbf{V}^T \mathbf{x}$  after

the KLT transform can be found as

$$\mathbf{m}_y = E[\mathbf{y}] = E[\mathbf{V}^T \mathbf{x}] = \mathbf{V}^T E[\mathbf{x}] = \mathbf{V}^T \mathbf{m}_x \quad (9.23)$$

$$\begin{aligned} \Sigma_y &= E[(\mathbf{y} - \mathbf{m}_y)(\mathbf{y} - \mathbf{m}_y)^T] = E[\mathbf{V}^T (\mathbf{x} - \mathbf{m}_x)] [\mathbf{V}^T (\mathbf{x} - \mathbf{m}_x)]^T \\ &= \mathbf{V}^T E[(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^T] \mathbf{V} = \mathbf{V}^T \Sigma_x \mathbf{V} = \Lambda \\ &= \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_d \end{bmatrix} = \begin{bmatrix} \sigma_{y_1}^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{y_d}^2 \end{bmatrix} \end{aligned} \quad (9.24)$$

Based on the fact that the covariance matrix  $\Sigma_y$  is a diagonal matrix, we have the following two observations:

- The zero off-diagonal component  $\sigma_{ij}^2 = 0$  ( $i \neq j$ ) for the covariance between any two components  $y_i$  and  $y_j$  indicates they are completely decorrelated after the KLT, i.e., each component carries its own independent information;
- The nonzero diagonal component for the variance  $\lambda_i = \sigma_i^2 = E[(y_i - \mu_{y_i})^2]$  ( $i = 1, \dots, c$ ) of the  $i$ th component  $y_i$  represents the dynamic energy or amount of information contained in  $y_i$ . As  $\Sigma_x$  and  $\Sigma_y = \Lambda$  share the same eigenvalues  $\{\lambda_1, \dots, \lambda_d\}$ , and therefore have the same trace

$$\mathcal{E}_x = \text{tr} \Sigma_x = \sum_{i=1}^d \lambda_i = \text{tr} \Sigma_y = \mathcal{E}_y \quad (9.25)$$

We see that the total amount of energy or information contained in the signal  $\mathbf{x}$  before the KLT and  $\mathbf{y}$  after the KLT remains the same, i.e., the energy or information contained in the signal is conserved by the KLT. This is generally true for all orthogonal transforms  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$  (such as the Fourier transform), due to the fact that the eigenvalues remain the same under any orthogonal transform  $\mathbf{B} = \mathbf{R}^T \mathbf{A} \mathbf{R}$ , i.e., matrices  $\mathbf{A}$  and  $\mathbf{B}$  share the same eigenvalues, and thereby the same trace and determinant.

### KLT Optimally Compacts Signal Energy

While the total energy contained in the signal is conserved by any orthogonal transform, the distribution of this energy among the  $d$  components before and after the transform may be very different. We can show that the KLT is optimal in the sense that it redistributes the total energy in such a way that it is maximally compacted into a subset of components of  $\mathbf{y} = \mathbf{V}^T \mathbf{x}$ .

Consider a generic orthogonal transform  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$  based on an arbitrary orthogonal matrix  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_d]$  satisfying  $\mathbf{A}^T = \mathbf{A}^{-1}$ :

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix} = \mathbf{A}^T \mathbf{x} = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_d^T \end{bmatrix} \mathbf{x} \quad (9.26)$$

We also have

$$y_i = \mathbf{a}_i^T \mathbf{x}, \quad \mu_{y_i} = E(y_i) = \mathbf{a}_i^T E(\mathbf{x}) = \mathbf{a}_i^T \mathbf{m}_x, \quad (i = 1, \dots, d) \quad (9.27)$$

We then consider the energy contained in  $d' < d$  out of the  $d$  components of  $\mathbf{y} = \mathbf{A}^T \mathbf{x}$  expressed as a function of the transform matrix  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_d]$ :

$$\begin{aligned} \mathcal{E}'(\mathbf{A}) &= \sum_{i=1}^{d'} \sigma_{y_i}^2 = \sum_{i=1}^{d'} E[(y_i - \mu_{y_i})^2] = \sum_{i=1}^{d'} E[\mathbf{a}_i^T (\mathbf{x} - \mathbf{m}_{x_i}) (\mathbf{x} - \mathbf{m}_{x_i})^T \mathbf{a}_i] \\ &= \sum_{i=1}^{d'} \mathbf{a}_i^T E[(\mathbf{x} - \mathbf{m}_{x_i})(\mathbf{x} - \mathbf{m}_{x_i})^T] \mathbf{a}_i = \sum_{i=1}^{d'} \mathbf{a}_i^T \boldsymbol{\Sigma}_x \mathbf{a}_i \end{aligned} \quad (9.28)$$

We can find the optimal orthogonal transform matrix  $\mathbf{A}$  that maximizes this partial energy  $\mathcal{E}'$  by solving the following optimization problem with the constraint that the column vectors of  $\mathbf{A}$  are normalized with  $\mathbf{a}_j^T \mathbf{a}_j = 1$ :

$$\begin{cases} \text{maximize: } \mathcal{E}'(\mathbf{A}) \\ \text{subject to: } \mathbf{a}_j^T \mathbf{a}_j = 1 \quad (j = 1, \dots, d') \end{cases} \quad (9.29)$$

The Lagrange function of this constrained optimization problem is

$$\mathcal{L}(\mathbf{A}) = \mathcal{E}'(\mathbf{A}) - \sum_{j=1}^{d'} \lambda_j (\mathbf{a}_j^T \mathbf{a}_j - 1) \quad (9.30)$$

To find the optimal  $\mathbf{A}$ , we set partial derivative with respect to each  $\mathbf{a}_i$  ( $i = 1, \dots, d'$ ) to zero:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{a}_i} \mathcal{L}(\mathbf{A}) &= \frac{\partial}{\partial \mathbf{a}_i} \left[ \sum_{j=1}^{d'} (\mathbf{a}_j^T \boldsymbol{\Sigma}_x \mathbf{a}_j - \lambda_j \mathbf{a}_j^T \mathbf{a}_j + \lambda_j) \right] \\ &= \frac{\partial}{\partial \mathbf{a}_i} (\mathbf{a}_i^T \boldsymbol{\Sigma}_x \mathbf{a}_i - \lambda_i \mathbf{a}_i^T \mathbf{a}_i) = 2\boldsymbol{\Sigma}_x \mathbf{a}_i - 2\lambda_i \mathbf{a}_i = \mathbf{0} \end{aligned} \quad (9.31)$$

The resulting equation  $\boldsymbol{\Sigma}_x \mathbf{a}_i = \lambda_i \mathbf{a}_i$  happens to be the eigenequation of the covariance matrix  $\boldsymbol{\Sigma}_x$ , i.e., the column vectors of the optimal transform matrix  $\mathbf{A}$  must be the eigenvectors of  $\boldsymbol{\Sigma}_x$  satisfying:

$$\boldsymbol{\Sigma}_x \mathbf{a}_i = \lambda_i \mathbf{a}_i, \quad \text{i.e.} \quad \mathbf{a}_i^T \boldsymbol{\Sigma}_x \mathbf{a}_i = \lambda_i, \quad (i = 1, \dots, d') \quad (9.32)$$

We therefore see that the optimal transform is indeed the KLT transform

$$\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_d] = \mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_d] \quad (9.33)$$

and

$$\mathcal{E}'(\mathbf{V}) = \sum_{i=1}^{d'} \mathbf{v}_i^T \boldsymbol{\Sigma}_x \mathbf{v}_i = \sum_{i=1}^{d'} \lambda_i \geq \sum_{i=1}^d \lambda_i = \mathcal{E}_y = \mathcal{E}_x \quad (9.34)$$

This partial energy  $\mathcal{E}'(\mathbf{V})$  is maximized if we choose those eigenvectors  $\mathbf{v}_i$  corresponding to the  $d'$  greatest eigenvalues of  $\boldsymbol{\Sigma}_x$ :  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{d'} \geq \dots \geq \lambda_d$ .

## 9.4 Geometric Interpretation of the KLT

The optimality of the KLT discussed above can be demonstrated geometrically, based on the assumption that the random vector  $\mathbf{x} = [x_1, \dots, x_d]^T$  has a normal probability density function:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}, \mathbf{m}_x, \mathbf{\Sigma}_x) = \frac{1}{(2\pi)^{d/2} |\mathbf{\Sigma}_x|^{1/2}} \exp \left[ -\frac{1}{2} (\mathbf{x} - \mathbf{m}_x)^T \mathbf{\Sigma}_x^{-1} (\mathbf{x} - \mathbf{m}_x) \right] \quad (9.35)$$

with mean vector  $\mathbf{m}_x$  and covariance matrix  $\mathbf{\Sigma}_x$ . The shape of this normal distribution in the d-dimensional space can be represented by the iso-hypersurface in the space determined by the equation

$$\mathcal{N}(\mathbf{x}, \mathbf{m}_x, \mathbf{\Sigma}_x) = c_0 \quad (9.36)$$

where  $c_0$  is some constant. This equation can be converted into an equivalent equation:

$$(\mathbf{x} - \mathbf{m}_x)^T \mathbf{\Sigma}_x^{-1} (\mathbf{x} - \mathbf{m}_x) = c_1 \quad (9.37)$$

where  $c_1$  is another constant related to  $c_0$ . As  $\mathbf{\Sigma}^{-1}$  is positive definite as well as  $\mathbf{\Sigma}$ , this equation represents an hyper ellipsoid in the d-dimensional space. In particular, when  $d = 2$ ,  $\mathbf{x} = [x_1, x_2]^T$ , with positive definite  $\mathbf{\Sigma}_x^{-1}$ :

$$\mathbf{\Sigma}_x^{-1} = \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix} \quad \text{and} \quad |\mathbf{\Sigma}_x^{-1}| = AC - B^2/4 > 0 \quad (9.38)$$

then the quadratic equation above becomes

$$\begin{aligned} (\mathbf{x} - \mathbf{m}_x)^T \mathbf{\Sigma}_x^{-1} (\mathbf{x} - \mathbf{m}_x) &= [x_1 - \mu_{x_1}, x_2 - \mu_{x_2}] \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix} \begin{bmatrix} x_1 - \mu_{x_1} \\ x_2 - \mu_{x_2} \end{bmatrix} \\ &= A(x_1 - \mu_{x_1})^2 + B(x_1 - \mu_{x_1})(x_2 - \mu_{x_2}) + C(x_2 - \mu_{x_2})^2 = c_1 \end{aligned}$$

representing an ellipse (instead of other quadratic curves such as hyperbola and parabola) centered at  $\mathbf{m}_x = [\mu_1, \mu_2]^T$ . When  $d = 3$ , the quadratic equation represents an ellipsoid. In general when  $d > 3$ , the equation  $\mathcal{N}(\mathbf{x}, \mathbf{m}_x, \mathbf{\Sigma}_x) = c_0$  represents a hyper-ellipsoid in the d-dimensional space.

Substituting  $\mathbf{x} = \mathbf{V}\mathbf{y}$  and  $\mathbf{m}_x = \mathbf{V}\mathbf{m}_y$  into the iso-surface equation Eq. (9.37), we get the equation for the hyper-ellipsoid after the KLT  $\mathbf{y} = \mathbf{V}^T \mathbf{x}$ :

$$\begin{aligned} (\mathbf{x} - \mathbf{m}_x)^T \mathbf{\Sigma}_x^{-1} (\mathbf{x} - \mathbf{m}_x) &= [\mathbf{V}(\mathbf{y} - \mathbf{m}_y)]^T \mathbf{\Sigma}_x \mathbf{V}(\mathbf{y} - \mathbf{m}_y) \\ &= (\mathbf{y} - \mathbf{m}_y)^T \mathbf{V}^T \mathbf{\Sigma}_x^{-1} \mathbf{V}(\mathbf{y} - \mathbf{m}_y) = (\mathbf{y} - \mathbf{m}_y)^T \mathbf{\Sigma}_y^{-1} (\mathbf{y} - \mathbf{m}_y) \\ &= (\mathbf{y} - \mathbf{m}_y)^T \mathbf{\Lambda}^{-1} (\mathbf{y} - \mathbf{m}_y) = \sum_{i=1}^d \frac{(y_i - \mu_{y_i})^2}{\lambda_i} = \sum_{i=1}^d \frac{(y_i - \mu_{y_i})^2}{\sigma_{y_i}^2} = c_1 \end{aligned} \quad (9.39)$$

This a standardized hyper-ellipsoid. We therefore see that the KLT transform  $\mathbf{y} = \mathbf{V}^T \mathbf{x}$  is actually a rotation of the coordinate system of the d-dimensional space, which is spanned by the standard basis  $\{\mathbf{e}_1, \dots, \mathbf{e}_d\}$  before the KLT and the eigenvectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_d\}$  as the basis vectors after the KLT.



As the result, the principal semi-axes of the ellipsoid representing the Gaussian distribution of the dataset become in parallel with the axes of the new coordinate system, i.e., the ellipsoid becomes standardized. Moreover, the length of the  $i$ th principal semi-axis is proportional to the standard deviation  $\sigma_{y_i} = \sqrt{\lambda_i}$  of the  $i$ th variable  $y_i$ . This is the reason why KLT possesses the two desirable properties: (a) the decorrelation of the signal components, and (b) redistribution and compaction of the energy or information contained in the signal, as illustrated in the figure below.

### Examples

## 9.5 Computation of the KLT

Due to KLT's properties of signal decorrelation and energy compaction, it can be used for data compression by reducing the dimensionality of the dataset. Specifically, we carry out the following steps:

1. Find the mean vector  $\mathbf{m}_x$  and the covariance matrix  $\mathbf{\Sigma}_x$  of the random vectors  $\mathbf{x}$  based on the given dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  containing  $N$  samples of  $\mathbf{x}$ .
2. Find the eigenvalues  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$  of  $\mathbf{\Sigma}_x$ , sorted in descending order, and their corresponding eigenvectors  $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_d\}$ .
3. Determine the reduced dimensionality  $d' < d$  so that the percentage of energy contained after the transform is no less than a given threshold (e.g., 95%):

$$\frac{\sum_{i=1}^{d'} \lambda_i}{\sum_{i=1}^d \lambda_i} = \frac{\sum_{i=1}^{d'} \sigma_{y_i}^2}{\sum_{i=1}^d \sigma_{y_i}^2} \geq 0.95 \quad (9.40)$$

4. Construct an  $d \times d'$  transform matrix composed of the  $d$  eigenvectors corresponding to the  $d$  greatest eigenvalues of  $\mathbf{\Sigma}_x$ :

$$\mathbf{V}' = [\mathbf{v}_1, \dots, \mathbf{v}_{d'}]_{d \times d'} \quad (9.41)$$

and carry out the KLT based on  $\mathbf{V}'$ :

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{d'} \end{bmatrix}_{d' \times 1} = \mathbf{V}'^T \mathbf{x} = \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_{d'}^T \end{bmatrix}_{d' \times d} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}_{d \times 1} \quad (9.42)$$

This is a lossy compression by which the dimensionality is reduced from  $d$  to  $d' < d$ , with the following error representing the percentage of information lost:

$$\sum_{i=d'+1}^d \lambda_i / \sum_{i=1}^d \lambda_i \quad (9.43)$$

But as  $\lambda_{d'+1}, \dots, \lambda_d$  are the smallest  $d - d'$  eigenvalues, the error is minimum (e.g., 5%).

5. Reconstruct  $\mathbf{x}$  by inverse KLT:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_d \end{bmatrix}_{d \times 1} = \mathbf{V}' \mathbf{y} = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_{d'} \end{bmatrix}_{d \times d'} \begin{bmatrix} y_1 \\ \vdots \\ y_{d'} \end{bmatrix}_{d' \times 1} = \sum_{i=1}^{d'} y_i \mathbf{v}_i \quad (9.44)$$

The rank of the  $d \times d$  estimated covariance matrix  $\hat{\Sigma}_x$  in Eq. (9.4) based on the dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  is no greater than  $N - 1$ . When  $N \leq d$ , then the rank of  $\hat{\Sigma}$  is at most  $N - 1 < d$ , i.e., there are no more than  $N - 1$  non-zero eigenvalues while all remaining  $d - (N - 1) = d - N + 1$  eigenvalues are zero. Also, if the dimensionality  $d \gg N$  is high, the computational complexity  $O(d^3)$  for solving this eigenvalue problem of  $\hat{\Sigma}_x$  may be expensive. In this case, we can find the  $N - 1$  non-zero eigenvalues and their corresponding eigenvectors of  $\hat{\Sigma}_x$  by solving the eigenvalue problem of an  $N \times N$  matrix with much lower complexity of  $O(N^3)$ .

Specifically, for convenience and without loss of generality, we assume that the data have zero mean  $\mathbf{m}_x = \mathbf{0}$ , so that the covariance matrix of the given dataset  $\mathbf{X}$  can be estimate as

$$\hat{\Sigma}_x = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \mathbf{m}_x)(\mathbf{x}_n - \mathbf{m}_x)^T = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T = \frac{1}{N} [\mathbf{x}_1, \dots, \mathbf{x}_N] \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \frac{1}{N} \mathbf{X} \mathbf{X}^T$$

and the eigenequation of this  $d \times d$  matrix can be written as

$$\hat{\Sigma}_x \mathbf{v}_i = \frac{1}{N} \mathbf{X} \mathbf{X}^T \mathbf{v}_i = \lambda_i \mathbf{v}_i \quad (9.45)$$

Pre-multiplying  $\mathbf{X}^T$  on both sides, we get the eigenequation of an  $N \times N$  matrix  $\mathbf{X}^T \mathbf{X} / N$ :

$$\frac{1}{N} \mathbf{X}^T \mathbf{X} (\mathbf{X}^T \mathbf{v}_i) = \lambda_i (\mathbf{X}^T \mathbf{v}_i) \quad \text{i.e.} \quad \frac{1}{N} \mathbf{X}^T \mathbf{X} \mathbf{u}_i = \lambda_i \mathbf{u}_i \quad (9.46)$$

where  $\mathbf{u}_i = \mathbf{X}^T \mathbf{v}_i$  is the eigenvector of  $\mathbf{X}^T \mathbf{X} / N$ . We see that  $\mathbf{X} \mathbf{X}^T / N$  and  $\mathbf{X}^T \mathbf{X} / N$  share the same non-zero eigenvalues. When  $N < d$ , we can solve the eigenvalue problem in Eq. (9.46) with complexity  $O(N^3)$ , instead that of the original problem in Eq. (9.45) with  $O(d^3)$ , to get the  $N - 1$  non-zero eigenvalues  $\lambda_i$ , ( $i = 1, \dots, N - 1$ ), and the corresponding eigenvectors  $\mathbf{u}_i = \mathbf{X}^T \mathbf{v}_i$ . To find the eigenvectors  $\mathbf{v}_i$  of the original matrix  $\hat{\Sigma}_x = \mathbf{X} \mathbf{X}^T / N$ , we pre-multiply  $\mathbf{X}$  on both sides of the eigenequation above and get

$$\frac{1}{N} \mathbf{X} \mathbf{X}^T (\mathbf{X} \mathbf{u}_i) = \lambda_i (\mathbf{X} \mathbf{u}_i) \quad (9.47)$$

Comparing this with Eq. (9.45), we see that  $\mathbf{X}\mathbf{u}_i = \mathbf{X}\mathbf{X}^T\mathbf{v}_i = N\lambda_i\mathbf{v}_i$  is proportional to the original eigenvector  $\mathbf{v}_i$  corresponding to  $\lambda_i$ , and they become the same when normalized.

## 9.6 Comparison with Other Orthogonal Transforms

Here we compare the KLT, in terms of signal decorrelation and energy compaction, with other orthogonal transforms such as discrete cosine transform (DCT) and Fourier transform (DFT), as well as identity transform IT (no transform), based on two images of cloud and sand as shown below.

We treat each column of the image as an observation (instantiation) of a 1-D random vector  $\mathbf{x}$ , and apply an orthogonal transform  $\mathbf{y} = \mathbf{A}^T\mathbf{x}$  to  $\mathbf{x}$  based on the covariance matrix  $\mathbf{\Sigma}_x$ , and compare the corresponding covariance matrices  $\mathbf{\Sigma}_y$  after the transform to see how well each transform decorrelates the signal and compacts its energy.

The figure below shows the original images (left) and the covariance matrices in image form (pixel values proportional to the values in the covariance matrix) after three orthogonal transforms of IT, DCT, and KLT.

- Sand image

In the second panel showing the covariance after IT, the off-diagonal pixels are dark, indicating the pixels in the original image are not highly correlated at all. In the third panel, the covariance after DCT looks similar to that before the transform, indicating that the DCT has little effect in terms of signal decorrelation and energy compaction. Finally, in the right-most panel showing the covariance after KLT, all off diagonal elements are zero, i.e., the signal components are completely decorrelated.

- Cloud image

In the second panel for the covariance after IT, there exist some bright areas off the diagonal, indicating that many signal components close to each other are highly correlated. In the third panel for the covariance after DCT, the values of the off diagonal elements are significantly reduced, indicating that the signal components are significantly decorrelated. Finally, in the right-most panel showing the covariance after KLT, all off diagonal elements are zero, i.e., the signal components are completely decorrelated.

The effect of energy redistribution and compaction of these different transforms are also shown in the figure below, where the variances of signal components are sorted and plotted.

We see that due to the physical nature of the clouds and sand grains, the textures in the corresponding images are very different. In the cloud image, the neighboring pixels are highly correlated, while in the sand image, they are not much correlated at all. Correspondingly, the DCT can effectively decorrelate the cloud image, but much less so in the sand image. But in either case, whether the

pixels in the image are highly correlated or not, the KLT can always completely decorrelate the signal and optimally compact the energy.

The table below further demonstrate the effect of energy compaction quantitatively in terms of the number of components out of the total of 256 components needed after the transform in order to keep a certain percentage of the total dynamic energy (information). For example, if it is desired to keep 95% of the total energy contained in the original signal, 230 components are needed with no transform, 22 are needed after DCT, and only 13 are needed after KLT. Note that DCT's performance is reasonably close to that of the optimal KLT in this case.

Percentage:	90%	95%	99%	100%
IT:	209	230	250	256
DCT:	10	22	97	256
KLT:	7	13	55	256

(9.48)

Although KLT is optimal in terms of signal decorrelation and energy compaction, it is not as convenient as other transforms for two reasons. First, the KLT is data-specific, i.e., the transform matrix is the eigenvector matrix of the covariance matrix of the dataset, which needs to be estimated based on sufficient amount of data samples, while all other orthogonal transforms are data independent. Second, the computational cost of the KLT is significantly higher than that of other transforms, due to the  $O(d^3)$  complexity for solving the eigenvalue problem of the  $d \times d$  matrix covariance matrix  $\Sigma_x$ , while for most other popular orthogonal transforms fast algorithm exist with  $O(d \log_2 d)$  complexity instead of  $O(d^2)$  required by the KLT.

## 9.7 Application to Image Data

The KLT can be applied to a set of  $N$  images for various purposes such as data compression and feature extraction. There are two alternative ways to carry out the KLT on the  $N$  images each containing  $K$  pixels, depending on how the random vector  $\mathbf{x}$  is defined based on the image data which can be represented in the form of an  $N \times K$  2-D array.

- A  $K$ -dimensional vector  $\mathbf{x}_n$  can be formed by all  $K$  pixels (by concatenating the rows or columns) of each of the  $N$  images. These  $N$  vectors each for one of the  $N$  images are represented by a  $K \times N$  data array  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , and treated as the samples of a  $K$ -dimensional random vector  $\mathbf{x}$ , its covariance matrix can be estimated (assuming with zero mean) as:

$$\Sigma = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T = \frac{1}{N} [\mathbf{x}_1, \dots, \mathbf{x}_N] \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \frac{1}{N} (\mathbf{X} \mathbf{X}^T)_{K \times K} \quad (9.49)$$

- An  $N$ -dimensional vector can be formed by the pixels at the same position (in the  $i$ th row and  $j$ th column of the image array) from all  $N$  images. There are  $K$  such vectors each for one of the  $K$  pixels in the images. These vectors are the rows of the  $\mathbf{X}$  defined above, or the columns of  $\mathbf{X}^T$ . The covariance matrix of these  $N$  dimensional vectors can be estimated as:

$$\Sigma' = \frac{1}{K} (\mathbf{X}^T \mathbf{X})_{N \times N}. \quad (9.50)$$

As shown previously, these two different covariance matrices share the same eigenvalues. The eigenequations for  $\mathbf{X}^T \mathbf{X}$  and  $\mathbf{X} \mathbf{X}^T$  (with the constant coefficients  $1/K$  and  $1/N$  neglected) are:

$$\mathbf{X}^T \mathbf{X} \mathbf{v} = \lambda \mathbf{v}, \quad \mathbf{X} \mathbf{X}^T \mathbf{u} = \mu \mathbf{u}. \quad (9.51)$$

Pre-multiplying  $\mathbf{X}^T$  on both sides of the second equation we get

$$\mathbf{X}^T \mathbf{X} [\mathbf{X}^T \mathbf{u}] = \mu [\mathbf{X}^T \mathbf{u}]. \quad (9.52)$$

which is actually the first eigenequation with the same eigenvalue  $\mu = \lambda$  and eigenvector  $\mathbf{v} = \mathbf{X}^T \mathbf{u}$  when normalized. The two covariance matrices  $\Sigma$  and  $\Sigma'$  have the same rank  $R = \min(N, K)$  (if  $\mathbf{X}$  is not degenerate) and therefore the same number of non-zero eigenvalues. Consequently, the KLT can be carried out based on either matrix with the same effects in terms of the signal decorrelation and energy compaction. As the number of pixels in the image is typically much greater than the number of images,  $K > N$ , we will take the second approach above to treat the pixels in the same position in all  $N$  images as a sample of an  $N$ -dimensional random signal vector and carry out the KLT based on the  $N \times N$  covariance matrix  $\hat{\Sigma}$ .

We can now carry out the KLT to each of the  $K$   $N$ -dimensional vectors  $\mathbf{x}$  corresponding to each pixel of the  $N$  images to obtain another  $N$ -dimensional vector  $\mathbf{y} = \mathbf{v}^* \mathbf{x}$  for the same pixel of a set of  $N$  *eigen-images*, as shown below. After the KLT, most of the energy/information contained in the  $N$  images, representing the variations among all  $N$  images, is concentrated in the first few eigen-images corresponding to the greatest eigenvalues, while the remaining eigen-images can be omitted without losing much energy/information. This is the foundation for various KLT-based image compression and feature extraction algorithms. The subsequent operations such as image recognition and classification can all be carried out in a much lower dimensional space.

We now consider some of such applications.

- **Remote sensing data compression**

In remote sensing, images of the surface of either the Earth or other planets such as Mars are taken by a multispectral camera system on board satellite, for various studies (e.g., geology, geography, etc.). The camera system has an array of  $N$  sensors, typically a few tens or even over a hundred, each sensitive to a different wavelength band in the visible and

infrared range of the electromagnetic spectrum. Depending on the number of sensors, the data are referred to as either multi or hyper-spectral images.

These sensors will produce a set of  $N$  images covering the same surface area on the ground. For the same position in these images, there are  $N$  pixel values each from one wavelength band representing the spectral profile that characterizes the material on the surface area corresponding to the pixel. A typical application of the multi or hyper-spectral image data is to classify the pixels into different types of materials (different types of rocks, vegetations, pollutants, etc.). When  $N$  is large, KLT can be used to reduce the dimensionality without loss of much information. Specifically, we consider the  $N$  values associated with each pixel form an  $N$ -dimensional random vector, and then carry out KLT to reduce its dimensionality. All classification can be carried out in this low dimensional space, thereby significantly reducing the computational complexity.

- **Video image compression**

A sequence of  $N = 8$  frames of a video of a moving escalator and their eigen-images are shown respectively in the upper and lower parts of the figure below.

It is interesting to observe that the first eigen-image corresponding to the greatest eigenvalue (left panel of the third row of the figure) represents mostly the static scene of the image frames representing the main variations in the image (carrying most of the energy), while the subsequent eigen-images represent mostly the motion in the video, the variation between the frames. For example, the motion of the people riding on the escalator is mostly reflected by the first few eigen-images following the first one, while the motion of the escalator stairs is mostly reflected in the subsequent eigen-images.

The  $8 \times 8$  covariance matrix and the energy distribution among the eight components plot before and after the KLT are shown below.

We see that due to the spatial correlation between nearby pixels, the covariance matrix before the KLT (left) can be modeled by the squared exponential function, while the covariance matrix after the KLT (middle) is completely decorrelated and the energy is highly compacted into a small number of principal components (here the first component), as also clearly shown in the comparison of the energy distribution before and after the KLT (right).

- **Eigenface and face recognition**

Twenty images of faces ( $N = 20$ ):

The eigen-images after KLT:

The table below shows the percentage of energy contained in each of the

components:

components	1	2	3	4	5	6	7	8	9	10	11	12	13
percentage energy	48.5	11.6	6.1	4.6	3.8	3.7	2.6	2.5	1.9	1.9	1.8	1.6	1.5
accumulative energy	48.5	60.1	66.2	70.8	74.6	78.3	81.0	83.5	85.4	87.3	89.	90.7	92.2

(9.53)

Reconstructed faces using 95% of the total information (15 out of 20 components):

- **Hand-written number recognition**

The goal here is to recognize hand-written number from 0 to 9 in an image, as those in the figure below, containing  $N = 2240$  samples for the  $K = 10$  numbers. Each sample in the  $16 \times 16$  image can be represented by a  $d = 16 \times 16 = 256$  dimensional vector by concatenating all columns (or rows) of the image. The KLT can be carried out to significantly reduce the dimensionality of the vectors from  $d = 256$  to some  $d' \ll d$ , based on either the covariance matrix  $\Sigma_x = \mathbf{S}_T$  of all  $N = 2240$  sample vectors representing the over all distribution of these data points, or the between-class scatter matrix  $\mathbf{S}_B$  previously considered representing the separability of the ten classes.

Specifically, we use the eigenvectors corresponding to the  $d'$  greatest eigenvalues of the covariance matrix or the between-class scatter matrix to form a  $d'$  by  $d$  transform matrix. After the KLT transform by the data, certain classification algorithm can be carried out in the much reduced  $d'$  dimensional space.

The energy distribution over all  $d = 256$  signal components is plotted below for the original signal (top), after the KLT based on  $\mathbf{S}_B$  (middle), and after the KLT based on  $\mathbf{S}_T$ .

For the KLT based on  $\mathbf{S}_T$  with rank  $N - 1$ ,  $d' = 79$  components are needed to keep 95.1% of the total energy, in comparison to the KTL based on  $\mathbf{S}_B$  with rank  $\text{rank}(\mathbf{S}_B) = K - 1 = 9$ , requiring are only  $d' = 9$  principal components corresponding to the same number of non-zero eigenvalues of  $\mathbf{S}_B$  to keep 100% of the total energy representing the separability information. The percentage energy contained in these non-zero eigenvalues are: 28.6%, 22.7%, 14.4%, 10.9%, 7.1%, 7.0%, 3.8%, 3.2%, 2.4%.

The corresponding  $d = 256$  dimensional eigenvectors can be visualized when converted to  $16 \times 16$  eigenimages, representing the basis by which any original images can be represented as a linear combination of such eigenimages, as shown in the figure below. The 10th eigenimage corresponding to a zero eigenvalue contains only some random noise.

If we only keep the first two or three principal components (corresponding to the greatest eigenvalues) after the KLT, the dataset can be visualized as shown in the figure below. The sample points in each of the ten different classes are color-coded. It can be seen that even when the dimensions are

much reduced from  $N = 256$  to  $d = 3$  or even  $d = 2$ , it is still possible to separate the ten classes reasonably well.



# 10 Variations of PCA

---

## Variations of PCA

### 10.1 Kernel Methods

In the kernel method, all data points as a vector  $\mathbf{x} = [x_1, \dots, x_d]^T$  in the original  $d$ -dimensional feature space are mapped by a *kernel function*  $\mathbf{z} = \phi(\mathbf{x})$  into a higher, possibly infinite, dimensional space

$$\mathbf{x} \implies \mathbf{z} = \phi(\mathbf{x}) \quad (10.1)$$

The presumption of the method is that the data points only appear in the form of inner product  $\mathbf{x}_m^T \mathbf{x}_n$  in the algorithm, then based on the *kernel trick*, the kernel function  $\mathbf{z} = \phi(\mathbf{x})$  never needs to be actually carried out. In fact, the form of the kernel function  $\phi(\mathbf{x})$  and the dimensionality of the higher dimensional space do not need to be explicitly specified or known.

The motivation for such a kernel mapping is that the relevant operations such as classification and clustering may be carried out much more effectively once the dataset is mapped to the higher dimensional space. For example, classes not linearly separable in the original  $d$ -dimensional feature space may be trivially separable in a higher dimensional space, as illustrated by the following examples.

#### Example 1

In 1-D space, classes  $C_- = \{x | (a \leq x \leq b)\}$  and  $C_+ = \{x | (x \leq a) \text{ or } (x \geq b)\}$  are not linearly separable. By the following mapping from 1-D space to 2-D space:

$$\mathbf{z} = \phi(x) = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} x \\ (x - (a + b)/2)^2 \end{bmatrix} \quad (10.2)$$

the two classes can be separated by a threshold in the second dimension of the 2-D space.

#### Example 2:

The method above can be generalized to higher dimensional spaces such as mapping from 2-D to 3-D space. Consider two classes in a 2-D space that are not linearly separable:  $C_- = \{\mathbf{x}, \|\mathbf{x}\| < D\}$  and  $C_+ = \{\mathbf{x}, \|\mathbf{x}\| > D\}$ . However,

by the following mapping from 2-D space to 3-D space:

$$\mathbf{z} = \phi(\mathbf{x}) = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix} \quad (10.3)$$

the two classes can be trivially separated linearly by thresholding in the third dimension of the 3-D space.

**Example 3:**

In 2-D space, in the exclusive OR dataset, the two classes of  $C_-$  containing points in quadrants I and III, and  $C_+$  containing points in quadrants II and IV are not linearly separable. However, by mapping the data points to a 3-D space:

$$\mathbf{z} = \phi(\mathbf{x}) = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix} \quad (10.4)$$

the two classes can be separated by simply thresholding in the third dimension of the 3-D space.

**Definition:** A kernel is a function that takes two vectors  $\mathbf{x}_m$  and  $\mathbf{x}_n$  as arguments and returns the inner product of their images  $\mathbf{z}_m = \phi(\mathbf{x}_m)$  and  $\mathbf{z}_n = \phi(\mathbf{x}_n)$ :

$$K(\mathbf{x}_m, \mathbf{x}_n) = \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n) = \mathbf{z}_m^T \mathbf{z}_n \quad (10.5)$$

The kernel function takes as input some two vectors  $\mathbf{x}_m$  and  $\mathbf{x}_n$  in the original feature space, and returns a scalar value as the inner product  $\mathbf{z}_m$  and  $\mathbf{z}_n$  in some higher dimensional space. If the data points in the original space only appear in the form of inner product, then the kernel function  $\mathbf{z} = \phi(\mathbf{x})$ , called the kernel-induced *implicit* mapping, never needs to be explicitly specified, and the dimension of the new does not even need to be known.

The following is a set of commonly used kernel functions  $K(\mathbf{x}, \mathbf{x}')$ , which can be represented as an inner product of two vectors  $\mathbf{z} = \phi(\mathbf{x})$  and  $\mathbf{z}' = \phi(\mathbf{x}')$  in a higher dimensional space.

- **linear kernel (no kernel mapping)**

Assume  $\mathbf{x} = [x_1, \dots, x_d]^T$ ,  $\mathbf{x}' = [x'_1, \dots, x'_d]^T$ ,

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}' = \sum_{i=1}^d x_i x'_i \quad (10.6)$$

- **polynomial kernels**

The binomial theorem states:

$$(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k = \sum_{k=0}^n \frac{n!}{k!(n-k)!} x^{n-k} y^k \quad (10.7)$$

where the binomial coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{10.8}$$

is the number of ways to distribute  $n$  items into two bins ( $k$  in one and  $n - k$  in the other). This result can be generalized to the multinomial case:

$$(x_1 + \dots + x_d)^n = \sum_{\sum_{i=1}^d k_i = n} \binom{n}{k_1, \dots, k_d} x_1^{k_1} \dots x_d^{k_d} = \sum_{\sum_{i=1}^d k_i = n} \frac{n!}{k_1! \dots k_d!} x_1^{k_1} \dots x_d^{k_d} \tag{10.9}$$

where the multinomial coefficient

$$\binom{n}{k_1, \dots, k_d} = \frac{n!}{k_1! \dots k_d!} \tag{10.10}$$

is the number of ways to distribute  $n$  balls into  $d$  bins with  $k_i$  balls in the  $i$ th bin, and the summation is over all possible ways to get  $d$  non-negative integers  $k_1, \dots, k_d$  that add up to  $n$ .

Now consider the homogeneous polynomial kernel for  $d$ -dimensional vectors  $\mathbf{x} = [x_1, \dots, x_d]^T$  defined as

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^T \mathbf{x}')^n = (x_1 x'_1 + \dots + x_d x'_d)^n \\ &= \sum_{\sum_{i=1}^d k_i = n} \frac{n!}{k_1! \dots k_d!} ((x_1 x'_1)^{k_1} \dots (x_d x'_d)^{k_d}) = \phi(\mathbf{x})^T \phi(\mathbf{x}') \end{aligned}$$

where

$$\mathbf{z} = \phi(\mathbf{x}) = \left[ \sqrt{\frac{n!}{k_1! \dots k_d!}} (x_1^{k_1} \dots x_d^{k_d}), \left( k_i \geq 0, \sum_{i=1}^d k_i = n \right) \right]^T \tag{10.12}$$

In particular, when  $d = 2$  and  $n = 2$ , the polynomial kernel defined over 2-D vectors  $\mathbf{x} = [x_1, x_2]^T$  is:

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2 = (x_1 x'_1 + x_2 x'_2)^2 = (x_1 x'_1)^2 + 2x_1 x'_1 x_2 x'_2 + (x_2 x'_2)^2 = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \mathbf{z}^T \mathbf{z}' \tag{10.13}$$

where  $\mathbf{z} = \phi(\mathbf{x}) = [x_1^2, \sqrt{2}x_1 x_2, x_2^2]$  is a mapping from  $\mathbf{x}$  in 2-D space to  $\mathbf{z}$  in 3-D space.

A non-homogeneous polynomial kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^n \tag{10.14}$$

- **The radial basis function (RBF) kernel**

The RBF kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = e^{-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2} = e^{-\gamma \|\mathbf{x} - \mathbf{x}'\|^2} \tag{10.15}$$

where  $\gamma = 1/2\sigma^2$  is a parameter that can be adjusted to fit each specific

dataset. This kernel can be written as the inner product of two infinite dimensional vectors (for simplicity, we assume  $\sigma = 1$ ):

$$\begin{aligned}
K(\mathbf{x}, \mathbf{x}') &= e^{-\|\mathbf{x}-\mathbf{x}'\|^2/2} = e^{-\|\mathbf{x}\|^2/2} e^{-\|\mathbf{x}'\|^2/2} e^{\mathbf{x}^T \mathbf{x}'} = e^{-\|\mathbf{x}\|^2/2} e^{-\|\mathbf{x}'\|^2/2} \sum_{n=0}^{\infty} \frac{(\mathbf{x}^T \mathbf{x}')^n}{n!} \\
&= e^{-\|\mathbf{x}\|^2/2} e^{-\|\mathbf{x}'\|^2/2} \sum_{n=0}^{\infty} \left[ \frac{1}{n!} \sum_{\sum_{i=1}^d k_i = n} \frac{n!}{k_1! \cdots k_d!} ((x_1 x'_1)^{k_1} \cdots (x_d x'_d)^{k_d}) \right] \\
&= \sum_{n=0}^{\infty} \sum_{\sum_{i=1}^d k_i = n} \left( e^{-\|\mathbf{x}\|^2/2} \frac{x_1^{k_1} \cdots x_d^{k_d}}{\sqrt{k_1! \cdots k_d!}} \right) \left( e^{-\|\mathbf{x}'\|^2/2} \frac{x'_1{}^{k_1} \cdots x'_d{}^{k_d}}{\sqrt{k_1! \cdots k_d!}} \right) \\
&= \phi(\mathbf{x})^T \phi(\mathbf{x}') = \mathbf{z}^T \mathbf{z}' \tag{10.16}
\end{aligned}$$

where

$$\mathbf{z} = \phi(\mathbf{x}) = \left[ e^{-\|\mathbf{x}\|^2/2} \frac{x_1^{k_1} \cdots x_d^{k_d}}{\sqrt{k_1! \cdots k_d!}}, \left( n = 0, \dots, \infty, \sum_{k=1}^n k_i = n \right) \right]^T \tag{10.17}$$

is a vector in an infinite dimensional space. In particular, when  $d = 1$  we have

$$\begin{aligned}
K(x, x') &= e^{-(x-x')^2/2} = e^{-x^2/2} e^{-x'^2/2} e^{xx'} = e^{-x^2/2} e^{-x'^2/2} \sum_{n=0}^{\infty} \frac{(xx')^n}{n!} \\
&= \sum_{n=0}^{\infty} (e^{-x^2/2} x^n / \sqrt{n!}) (e^{-x'^2/2} x'^n / \sqrt{n!}) \tag{10.18}
\end{aligned}$$

where  $\mathbf{z} = \phi(x) = \left[ e^{-x^2/2} x^n / \sqrt{n!}, (n = 0, \dots, \infty) \right]^T$  is a kernel function that maps a 1-D space into an infinite dimensional space.

## 10.2 Kernel PCA

In kernel PCA, the PCA method, which maps the dataset to a low dimensional space, and the kernel method, which maps the dataset to a high dimensional space, are combined to take advantage of both the high dimensional space (e.g., better separability) and the low dimensional space (e.g., computational efficiency).

Specifically, we assume the given dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  in the original  $d$ -dimensional feature space is mapped into  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$  in a high dimensional space. Correspondingly, the inner product  $\mathbf{x}_m^T \mathbf{x}_n$  in the original space is mapped into an inner product in the high dimensional space:

$$\mathbf{z}_m^T \mathbf{z}_n = K(\mathbf{x}_m, \mathbf{x}_n) = k_{mn} \quad (m, n = 1, \dots, N) \tag{10.19}$$

Given  $N$  data points in the dataset  $\mathbf{X}$ , we can further define a kernel matrix:

$$\mathbf{K} = \mathbf{Z}^T \mathbf{Z} = \begin{bmatrix} \mathbf{z}_1^T \\ \vdots \\ \mathbf{z}_N^T \end{bmatrix} [\mathbf{z}_1, \dots, \mathbf{z}_N] = \begin{bmatrix} \mathbf{z}_1^T \mathbf{z}_1 & \cdots & \mathbf{z}_1^T \mathbf{z}_N \\ \vdots & \ddots & \vdots \\ \mathbf{z}_N^T \mathbf{z}_1 & \cdots & \mathbf{z}_N^T \mathbf{z}_N \end{bmatrix} = \begin{bmatrix} k_{11} & \cdots & k_{1N} \\ \vdots & \ddots & \vdots \\ k_{N1} & \cdots & k_{NN} \end{bmatrix} \quad (10.20)$$

To carry out KLT in the high dimensional space, we need to estimate the covariance matrix of  $\mathbf{z} = \phi(\mathbf{x})$ :

$$\Sigma_z = \frac{1}{N} \sum_{n=1}^N (\mathbf{z}_n - \mathbf{m}_z)(\mathbf{z}_n - \mathbf{m}_z)^T = \frac{1}{N} \sum_{n=1}^N \mathbf{z}_n \mathbf{z}_n^T - \mathbf{m}_z \mathbf{m}_z^T \quad (10.21)$$

We first assume the samples in  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$  have a zero mean  $\mathbf{m}_z = \mathbf{0}$ , which will be justified later, and get (same as Eq. (9.45)):

$$\Sigma_z = \frac{1}{N} \sum_{n=1}^N (\mathbf{z}_n - \mathbf{m}_z)(\mathbf{z}_n - \mathbf{m}_z)^T = \frac{1}{N} \sum_{n=1}^N \mathbf{z}_n \mathbf{z}_n^T = \frac{1}{N} [\mathbf{z}_1, \dots, \mathbf{z}_N] \begin{bmatrix} \mathbf{z}_1^T \\ \vdots \\ \mathbf{z}_N^T \end{bmatrix} = \frac{1}{N} \mathbf{Z} \mathbf{Z}^T \quad (10.22)$$

and its eigenequations:

$$\Sigma_z \mathbf{v}_n = \left( \frac{1}{N} \mathbf{Z} \mathbf{Z}^T \right) \mathbf{v}_n = \lambda_n \mathbf{v}_n \quad (n = 1, 2, \dots) \quad (10.23)$$

We then premultiply  $\mathbf{Z}^T$  on both sides of the eigenequation above to get (same as Eq. (9.46)):

$$\frac{1}{N} \mathbf{Z}^T \mathbf{Z} (\mathbf{Z}^T \mathbf{v}_n) = \frac{1}{N} \mathbf{K} \mathbf{u}_n = \lambda_n (\mathbf{Z}^T \mathbf{v}_n) = \lambda_n \mathbf{u}_n \quad (10.24)$$

where  $\mathbf{u}_n = \mathbf{Z}^T \mathbf{v}_n$ . Solving this eigenequation of the  $N \times N$  matrix  $\mathbf{Z}^T \mathbf{Z} / N = \mathbf{K} / N$ , we get its eigenvectors  $\mathbf{u}_n = \mathbf{Z}^T \mathbf{v}_n$  ( $n = 1, \dots, N$ ), corresponding to the eigenvalues  $\{\lambda_1, \dots, \lambda_N\}$ , which are the same as the non-zero eigenvalues of  $\Sigma_z = \mathbf{Z} \mathbf{Z}^T / N$ . The eigenvectors  $\mathbf{v}_n$  of  $\Sigma_z$  in Eq. (10.23) can now be written as:

$$\mathbf{v}_n = \frac{1}{\lambda_n N} \mathbf{Z} (\mathbf{Z}^T \mathbf{v}_n) = \frac{1}{\lambda_n N} \mathbf{Z} \mathbf{u}_n \quad (10.25)$$

which can be further normalized by imposing the condition  $\|\mathbf{v}_n\|^2 = 1$ :

$$\begin{aligned} \|\mathbf{v}_n\|^2 &= \mathbf{v}_n^T \mathbf{v}_n = \frac{1}{(\lambda_n N)^2} (\mathbf{Z} \mathbf{u}_n)^T (\mathbf{Z} \mathbf{u}_n) = \frac{1}{(\lambda_n N)^2} \mathbf{u}_n^T \mathbf{Z}^T \mathbf{Z} \mathbf{u}_n \\ &= \frac{1}{(\lambda_n N)^2} \mathbf{u}_n^T \mathbf{K} \mathbf{u}_n = \frac{1}{\lambda_n N} \mathbf{u}_n^T \mathbf{u}_n = \frac{1}{\lambda_n N} \|\mathbf{u}_n\|^2 = 1 \end{aligned} \quad (10.26)$$

We see that if the eigenvectors  $\mathbf{u}_n$  of  $\mathbf{K}$  are rescaled to satisfy  $\|\mathbf{u}_n\|^2 = \lambda_n N$ , then the eigenvectors  $\mathbf{v}_n$  of  $\Sigma_z$  are normalized to satisfy  $\|\mathbf{v}_n\|^2 = 1$ . Now we can carry out PCA in the high dimensional space by

$$\mathbf{y} = [\mathbf{v}_1, \dots, \mathbf{v}_N]^T \mathbf{z} = \mathbf{V}^T \mathbf{z} \quad (10.27)$$

of which each component  $y_n = \mathbf{v}_n^T \mathbf{z}$  is the projection of  $\mathbf{z} = \phi(\mathbf{x})$  onto one of the eigenvectors  $\mathbf{v}_n$  of  $\Sigma_z$ :

$$\begin{aligned} y_n &= \mathbf{z}^T \mathbf{v}_n = \mathbf{z}^T \left( \frac{1}{\lambda_n N} \mathbf{Z} \mathbf{u}_n \right) = \frac{1}{\lambda_n N} \mathbf{z}^T \mathbf{Z} \mathbf{u}_n = \frac{1}{\lambda_n N} \mathbf{z}^T [\mathbf{z}_1, \dots, \mathbf{z}_N] \mathbf{u}_n \\ &= \frac{1}{\lambda_n N} [\mathbf{z}^T \mathbf{z}_1, \dots, \mathbf{z}^T \mathbf{z}_N] \mathbf{u}_n = \frac{1}{\lambda_n N} [k_1, \dots, k_N] \mathbf{u}_n = \frac{1}{\lambda_n N} \mathbf{k}^T \mathbf{u}_n, \end{aligned} \quad (n = 1, \dots, N) \quad (10.28)$$

where we have defined  $k_n = \mathbf{z}^T \mathbf{z}_n$  and  $\mathbf{k} = [k_1, \dots, k_N]^T$ . Although neither  $\mathbf{z}$  nor  $\mathbf{v}_n = \mathbf{Z} \mathbf{u}_n / \lambda_n N$  is available, their inner product  $y_n = \mathbf{z}^T \mathbf{v}_n$  can still be obtained based on the kernel function  $k_n = \mathbf{z}^T \mathbf{z}_n$ . We see that all data points in the high dimensional space appear only in the form of an inner product in the equation above, as well as in the eigenequation Eq. (10.24), the kernel mapping  $\mathbf{z} = \phi(\mathbf{x})$  never needs to be explicitly carried out.

The discussion above is based on the assumption that the data in the high dimensional space have zero mean  $\mathbf{m}_z = \mathbf{0}$ . This cannot be achieved by subtracting the mean from the data to get  $\tilde{\mathbf{z}} = \mathbf{z} - \mathbf{m}_z$ , as neither  $\mathbf{z}$  nor  $\mathbf{m}_z$  is available without carrying out the kernel mapping  $\mathbf{z} = \phi(\mathbf{x})$ . However, we can find the inner product of two zero-mean data points in the high dimensional space as

$$\begin{aligned} \tilde{k}_{mn} &= \tilde{\mathbf{z}}_m^T \tilde{\mathbf{z}}_n = (\mathbf{z}_m - \mathbf{m}_z)^T (\mathbf{z}_n - \mathbf{m}_z) = \left( \mathbf{z}_m - \frac{1}{N} \sum_{k=1}^N \mathbf{z}_k \right)^T \left( \mathbf{z}_n - \frac{1}{N} \sum_{l=1}^N \mathbf{z}_l \right) \\ &= \mathbf{z}_m^T \mathbf{z}_n - \mathbf{z}_m^T \left( \frac{1}{N} \sum_{l=1}^N \mathbf{z}_l \right) - \left( \frac{1}{N} \sum_{k=1}^N \mathbf{z}_k^T \right) \mathbf{z}_n + \frac{1}{N^2} \sum_{k=1}^N \sum_{l=1}^N \mathbf{z}_k^T \mathbf{z}_l \\ &= k_{mn} - \frac{1}{N} \sum_{l=1}^N k_{ml} - \frac{1}{N} \sum_{k=1}^N k_{kn} + \frac{1}{N^2} \sum_{k=1}^N \sum_{l=1}^N k_{kl}, \end{aligned} \quad (m, n = 1, \dots, N) \quad (10.29)$$

The kernel matrix composed of all  $N \times N$  such inner products can be written as:

$$\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N \mathbf{K} - \mathbf{K} \mathbf{1}_N + \mathbf{1}_N \mathbf{K} \mathbf{1}_N \quad (10.30)$$

where

$$\mathbf{1}_N = \frac{1}{N} \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}_{N \times N} \quad (10.31)$$

If we replace the kernel matrix  $\mathbf{K}$  for  $\mathbf{z}$  with non-zero mean in the algorithm considered above by  $\tilde{\mathbf{K}}$  for  $\tilde{\mathbf{z}}$  with zero mean, then the assumption that  $\mathbf{z}$  has zero mean  $\mathbf{m}_z$  becomes valid. In summary, here are the steps of the kernel PCA algorithm:

- Given the dataset  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ , obtain  $\mathbf{K}$  based on kernel mapping  $k_{mn} = k(\mathbf{x}_m, \mathbf{x}_n)$  for all  $m, n = 1, \dots, N$ , and then find  $\tilde{\mathbf{K}}$ , as in Eq. (10.30);

- Solve eigenvalue problem of  $\tilde{\mathbf{K}}$  to get the eigenvalue  $\lambda_n$  and the corresponding eigenvector  $\mathbf{u}_n$  for all  $i = 1, \dots, N$ ;
- Project each data point  $\mathbf{z} = \phi(\mathbf{x})$  (implicit) onto the  $n$ th eigenvector  $\mathbf{v}_n$  to get the principal component:  $y_n = \mathbf{z}^T \mathbf{v}_n = \mathbf{k}^T \mathbf{u}_n / \lambda_n N$  for each  $n = 1, \dots, N$ , as in Eq. (10.28).
- Carry out whatever operation (e.g., feature selection, classification) in the subspace spanned by the first  $M < N$  principal components  $\{y_1, \dots, y_M\}$  in the high dimensional feature space, containing most of the information.

Note that in kernel PCA methods, the complexity for solving the eigenvalue problem of the  $N \times N$  kernel matrix  $\mathbf{K}$  is  $O(N^3)$ , instead of  $O(d^3)$  for the linear PCA based on the  $d \times d$  covariance matrix  $\Sigma_x$ .

Shown below is a Matlab function for the kernel PCA algorithm which takes the dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  as input and generates the eigenvalues in  $\mathbf{d}$  and the dataset in the transform space  $\mathbf{Y}$ . The function `Kernel(X,X)` calculates the kernel matrix  $\mathbf{K}$ .

```
function [Y d]=KPCA(X)
    [D N]=size(X);
    K=Kernel(X,X);           % kernel matrix of data X
    one=ones(N)/N;
    Kt=K-one*K-K*one+one*K*one; % Kernel matrix of data with zero mean
    [U D]=eig(Kt);          % solve eigenvalue problem for K
    [d idx]=sort(diag(D),'descend'); % sort eigenvalues in descending order
    U=U(:,idx);
    D=D(:,idx);
    for n=1:N
        U(:,n)=U(:,n)/d(n)/N;
    end
    Y=(K*U)';
end
```

#### Examples:

Visualization of a 2-D dataset of four classes, linear plot on the left and kernel PCA (based on RBF kernel) on the right:

Visualization of a 256-D dataset of 10 classes of hand-written digits:

Visualization of the 4-D iris dataset in 3-D space spanned by the first 3 principal components of the linear and kernel PCA transforms:

## 10.3 Factor Analysis and Expectation Maximization

The method of *factor analysis (FA)* models a set of  $d$  observed *manifest variables* in  $\mathbf{x} = [x_1, \dots, x_d]^T$  as a linear combination of a set of  $d' < d$  unobserved hidden *latent variables* or *common factors* in  $\mathbf{z} = [z_1, \dots, z_{d'}]^T$ , to explain and reveal the

variability and dependency among the  $d$  observed variables, typically correlated, in terms of the latent variables, assumed to be independent and therefore uncorrelated. The method of FA is therefore considered as a method for dimensionality reduction.

Specifically, we assume each of the observed variables in  $\mathbf{x}$  is a linear combination of the  $d'$  factors in  $\mathbf{z}$

$$x_i = \sum_{j=1}^{d'} w_{ij} z_j + e_i = [w_{i1}, \dots, w_{id'}] \begin{bmatrix} z_1 \\ \vdots \\ z_{d'} \end{bmatrix} + e_i, \quad (i = 1, \dots, d) \quad (10.32)$$

or in matrix form

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{1d'} \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ w_{d1} & \cdots & w_{dd'} \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_{d'} \end{bmatrix} + \begin{bmatrix} e_1 \\ \vdots \\ \vdots \\ e_d \end{bmatrix} = \mathbf{W}\mathbf{z} + \mathbf{e} \quad (10.33)$$

where  $\mathbf{W}$  is a  $d \times d'$  *factor loading matrix*, and  $e_i$  is the noise associated with  $x_i$ . Also, for simplicity and without loss of generality, we assume the dataset has a zero mean. If  $\mathbf{W}$  were available, the  $d'$  factors in  $\mathbf{z}$  can be found by solving this over-determined linear equation system of  $d$  equations but  $d' < d$  unknowns by the least-squares method (with minimum squared error  $\|\mathbf{e}\|^2$ ):

$$\hat{\mathbf{z}} = \mathbf{W}^- \mathbf{x} = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T \mathbf{x} \quad (10.34)$$

where  $\mathbf{W}^- = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T$  is the left pseudo-inverse of  $\mathbf{W}$ .

However, as  $\mathbf{W}$  is unavailable, it needs to be estimated together with  $\mathbf{z}$  at the same time, based on the given dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , typically  $N \gg d$ . This can be done by the general method of *expectation-maximization (EM)*, an iterative algorithm that maximizes the expectation of the likelihood (ML) or maximum a posteriori (MAP) estimate of some model parameters, widely used in machine learning.

Specifically, we treat both  $\mathbf{z}$  and  $\mathbf{e}$  as random vectors, and make the following assumptions:

- The latent variables in  $\mathbf{z}$  are of zero mean, independent of each other, and with unity variance:

$$\mathbf{m}_z = E[\mathbf{z}] = \mathbf{0}, \quad \Sigma_z = Cov[\mathbf{z}] = E[\mathbf{z}\mathbf{z}^T] = \mathbf{I} \quad (10.35)$$

and they are normally distributed:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (10.36)$$

- The noise components in  $\mathbf{e}$  have zero mean, and they are independent of each



other with a diagonal covariance matrix:

$$\mathbf{m}_e = E[\mathbf{e}] = \mathbf{0}, \quad \Sigma_e = Cov[\mathbf{e}] = E[\mathbf{e}\mathbf{e}^T] = \Psi = \begin{bmatrix} \psi_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \psi_d \end{bmatrix} \quad (10.37)$$

and they are also normally distributed:

$$p(\mathbf{e}) = \mathcal{N}(\mathbf{0}, \Psi) \quad (10.38)$$

- The latent variables and the noise are independent of each other:

$$\Sigma_{ze} = Cov[\mathbf{z}, \mathbf{e}] = E[(\mathbf{z} - \mathbf{m}_z)(\mathbf{e} - \mathbf{m}_e^T)] = E[\mathbf{z}\mathbf{e}^T] = \mathbf{0} \quad (10.39)$$

The two matrices  $\mathbf{W}$  and  $\Psi$  defined above as the parameters of the FA model are denoted by  $\theta = \{\mathbf{W}, \Psi\}$ .

Based on the assumptions above, we desire to find the conditional pdf  $p(\mathbf{z}|\mathbf{x})$  of the latent variable  $\mathbf{z}$  given the observed variable  $\mathbf{x}$ . To do so, we first find the pdf of  $\mathbf{x} = \mathbf{W}\mathbf{z} + \mathbf{e}$ , which, as a linear combination of the two normally distributed random vectors  $\mathbf{z}$  and  $\mathbf{e}$ , is also normally distributed with  $p(\mathbf{x}) = \mathcal{N}(\mathbf{m}_x, \Sigma_x)$ , where

$$\mathbf{m}_x = E[\mathbf{x}] = E[\mathbf{W}\mathbf{z} + \mathbf{e}] = \mathbf{W}E[\mathbf{z}] + E[\mathbf{e}] = \mathbf{0} \quad (10.40)$$

$$\begin{aligned} \Sigma_x = Cov[\mathbf{x}] &= E[(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x^T)] = E[\mathbf{x}\mathbf{x}^T] = E[(\mathbf{W}\mathbf{z} + \mathbf{e})(\mathbf{W}\mathbf{z} + \mathbf{e})^T] \\ &= \mathbf{W}E[\mathbf{z}\mathbf{z}^T]\mathbf{W}^T - E[\mathbf{e}\mathbf{z}^T]\mathbf{W}^T - \mathbf{W}E[\mathbf{z}\mathbf{e}^T] + E[\mathbf{e}\mathbf{e}^T] \\ &= \mathbf{W}\mathbf{W}^T + \Psi \end{aligned} \quad (10.41)$$

As both  $\mathbf{m}_x$  and  $\Sigma_x$  are based on the model parameter  $\theta = \{\mathbf{W}, \Psi\}$ , the pdf of  $\mathbf{x}$  is conditional on  $\theta$ :

$$p(\mathbf{x}|\theta) = \mathcal{N}(\mathbf{m}_x, \Sigma_x) = \mathcal{N}(\mathbf{0}, \mathbf{W}\mathbf{W}^T + \Psi) \quad (10.42)$$

The joint distribution  $p(\mathbf{z}, \mathbf{x})$  is also Gaussian with a zero mean

$$\mathbf{m} = E\left(\begin{bmatrix} \mathbf{z} \\ \mathbf{x} \end{bmatrix}\right) = \begin{bmatrix} \mathbf{m}_z \\ \mathbf{m}_x \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \quad (10.43)$$

and a covariance  $\Sigma$  composed of four submatrices:

$$\Sigma = \begin{bmatrix} \Sigma_{zz} & \Sigma_{zx} \\ \Sigma_{xz} & \Sigma_{xx} \end{bmatrix} \quad (10.44)$$

where

$$\begin{aligned} \Sigma_{zz} &= \mathbf{I} \\ \Sigma_{xx} &= \mathbf{W}\mathbf{W}^T + \Psi \\ \Sigma_{zx} &= \Sigma_{zx}^T = E[(\mathbf{z} - \mathbf{m}_z)(\mathbf{x} - \mathbf{m}_x)^T] = E[\mathbf{z}\mathbf{x}^T] = E[\mathbf{z}(\mathbf{W}\mathbf{z} + \mathbf{e})^T] \\ &= E[\mathbf{z}\mathbf{z}^T]\mathbf{W}^T + E[\mathbf{z}\mathbf{e}^T] = \mathbf{I}\mathbf{W}^T + \mathbf{0} = \mathbf{W}^T \end{aligned} \quad (10.45)$$

The normal distribution  $p(\mathbf{z}, \mathbf{x}|\theta)$  can now be expressed as:

$$\begin{aligned} p(\mathbf{x}, \mathbf{z}|\theta) &= p\left(\begin{bmatrix} \mathbf{z} \\ \mathbf{x} \end{bmatrix}\right) = \mathcal{N}(\mathbf{m}, \Sigma) = \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \Sigma_{zz} & \Sigma_{zx} \\ \Sigma_{xz} & \Sigma_{xx} \end{bmatrix}\right) \\ &= \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{I} & \mathbf{W}^T \\ \mathbf{W} & \mathbf{W}\mathbf{W}^T + \Psi \end{bmatrix}\right) \end{aligned} \quad (10.46)$$

Based on this joint pdf of both  $\mathbf{x}$  and  $\mathbf{z}$ , we can further find the desired conditional pdfs of both  $p(\mathbf{x}|\mathbf{z})$  and  $p(\mathbf{z}|\mathbf{x})$  (Section 3.1.3):

- $p(\mathbf{x}|\mathbf{z}, \theta) = \mathcal{N}(\mathbf{m}_{x|z}, \Sigma_{x|z})$ , with

$$\begin{cases} \mathbf{m}_{x|z} &= \mathbf{m}_x + \Sigma_{xz}\Sigma_{zz}^{-1}(\mathbf{z} - \mathbf{m}_z) = \mathbf{W}\mathbf{z} \\ \Sigma_{x|z} &= \Sigma_{xx} - \Sigma_{xz}\Sigma_{zz}^{-1}\Sigma_{zx} = \mathbf{W}\mathbf{W}^T + \Psi - \mathbf{W}\mathbf{W}^T = \Psi \end{cases} \quad (10.47)$$

- $p(\mathbf{z}|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{m}_{z|x}, \Sigma_{z|x})$ , with

$$\begin{cases} \mathbf{m}_{z|x} &= \mathbf{m}_z + \Sigma_{zx}\Sigma_{xx}^{-1}(\mathbf{x} - \mathbf{m}_x) = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T + \Psi)^{-1}\mathbf{x} = \mathbf{B}\mathbf{x} \\ \Sigma_{z|x} &= \Sigma_{zz} - \Sigma_{zx}\Sigma_{xx}^{-1}\Sigma_{xz} = \mathbf{I} - \mathbf{W}^T(\mathbf{W}\mathbf{W}^T + \Psi)^{-1}\mathbf{W} = \mathbf{I} - \mathbf{B}\mathbf{W} \end{cases} \quad (10.48)$$

where we have defined

$$\mathbf{B} = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T + \Psi)^{-1} \quad (10.49)$$

Note that while  $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$  has zero mean and diagonal covariance, the conditional distribution  $p(\mathbf{z}|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{m}_{z|x}, \Sigma_{z|x})$  has non-zero mean  $\mathbf{m}_{z|x}$  and non-diagonal covariance  $\Sigma_{z|x}$ .

The computational complexity for the inversion of the  $N \times N$  matrix  $\mathbf{W}\mathbf{W}^T + \Psi$  is  $O(N^3)$ . However, by applying the Woodbury matrix identity (Section A.6):

$$(\Psi + \mathbf{W}\mathbf{W}^T)^{-1} = \Psi^{-1} - \Psi^{-1}\mathbf{W}(\mathbf{I} + \mathbf{W}^T\Psi^{-1}\mathbf{W})^{-1}\mathbf{W}^T\Psi^{-1} \quad (10.50)$$

where  $\Psi$  as a diagonal matrix can be easily inverted, and  $\mathbf{I} + \mathbf{W}^T\Psi^{-1}\mathbf{W}$  is an  $d' \times d'$  matrix that can be inverted with complexity  $O(d'^3) \ll O(N^3)$ .

The model parameter  $\theta = \{\mathbf{W}, \Psi\}$  can now be estimated based on the given dataset  $\mathbf{X}$  by the EM algorithm, an iterative process of the following two steps:

- **The E-step:**

Find the expectation of the log-likelihood function of the model parameter  $\theta$ , to be maximized in the following M-step.

Find the likelihood function of  $\theta$  based on the observed dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  (all samples assumed to be i.i.d.):

$$L(\theta|\mathbf{X}, \mathbf{z}) = p(\mathbf{X}, \mathbf{z}|\theta) = \prod_{n=1}^N p(\mathbf{x}_n, \mathbf{z}|\theta) = \prod_{n=1}^N p(\mathbf{x}_n|\mathbf{z}, \theta) p(\mathbf{z}|\theta) \quad (10.51)$$

and the log-likelihood:

$$\log L(\theta|\mathbf{X}, \mathbf{z}) = \sum_{n=1}^N (\log p(\mathbf{x}_n|\mathbf{z}, \theta) + \log p(\mathbf{z})) \quad (10.52)$$

The second term can be dropped as  $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$  is independent of the model parameter  $\theta$  and therefore irrelevant to the maximization of the log-likelihood with respect to  $\theta$ .

We then find the expectation of the log-likelihood function, denoted by  $Q$ , with respect to the latent variable  $\mathbf{z}$  based on Eq. (10.47):

$$\begin{aligned}
Q &= E_{z|x} [\log L(\theta|\mathbf{X}, \mathbf{z})] = \sum_{n=1}^N E_{z|x_n} [\log p(\mathbf{x}_n|\mathbf{z}, \theta)] \\
&= \sum_{n=1}^N E_{z|x_n} \log \left[ \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}_{x|z}|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x}_n - \mathbf{m}_{x|z})^T \boldsymbol{\Sigma}_{x|z}^{-1} (\mathbf{x}_n - \mathbf{m}_{x|z}) \right) \right] \\
&= \sum_{n=1}^N E_{z|x_n} \left[ -\frac{d}{2} \log(2\pi) - \frac{1}{2} \log |\boldsymbol{\Psi}| - \frac{1}{2} (\mathbf{x}_n - \mathbf{Wz})^T \boldsymbol{\Psi}^{-1} (\mathbf{x}_n - \mathbf{Wz}) \right] \\
&= -\frac{dN}{2} \log(2\pi) - \frac{N}{2} \log |\boldsymbol{\Psi}| - \frac{1}{2} \sum_{n=1}^N E_{z|x_n} [(\mathbf{x}_n - \mathbf{Wz})^T \boldsymbol{\Psi}^{-1} (\mathbf{x}_n - \mathbf{Wz})] \\
&= C - \frac{N}{2} \log |\boldsymbol{\Psi}| - \frac{1}{2} \sum_{n=1}^N E_{z|x_n} [\mathbf{x}_n^T \boldsymbol{\Psi}^{-1} \mathbf{x}_n - 2\mathbf{x}_n^T \boldsymbol{\Psi}^{-1} \mathbf{Wz} + \mathbf{z}^T \mathbf{W}^T \boldsymbol{\Psi}^{-1} \mathbf{Wz}]
\end{aligned}$$

where we have used the fact that

$$\mathbf{x}_n^T \boldsymbol{\Psi}^{-1} (\mathbf{Wz}) = (\mathbf{Wz})^T \boldsymbol{\Psi}^{-1} \mathbf{x}_n \quad (10.54)$$

as both  $\boldsymbol{\Psi}^{-1}$  and  $\boldsymbol{\Psi}$  are diagonal and therefore also symmetric. Here the constant  $C = -dN \log(2\pi)/2$  can be dropped.

- **The M-step:**

Find the optimal model parameter  $\theta = \{\mathbf{W}, \boldsymbol{\Psi}\}$  that maximizes the expectation of the log-likelihood  $Q$  obtained in the E-step.

We set to zero the derivative of  $Q$  with respect to each of the two parameters in  $\theta = \{\mathbf{W}, \boldsymbol{\Psi}\}$  and solve the resulting equations.

– Find  $\mathbf{W}$ :

$$\begin{aligned}
\frac{\partial Q}{\partial \mathbf{W}} &= \frac{\partial}{\partial \mathbf{W}} \sum_{n=1}^N E_{z|x_n} [-\mathbf{x}_n^T \boldsymbol{\Psi}^{-1} \mathbf{x}_n + 2\mathbf{x}_n^T \boldsymbol{\Psi}^{-1} \mathbf{Wz} - \mathbf{z}^T \mathbf{W}^T \boldsymbol{\Psi}^{-1} \mathbf{Wz}] \\
&= \sum_{n=1}^N E_{z|x_n} \left[ \frac{\partial}{\partial \mathbf{W}} (2\mathbf{x}_n^T \boldsymbol{\Psi}^{-1} \mathbf{Wz} - \mathbf{z}^T \mathbf{W}^T \boldsymbol{\Psi}^{-1} \mathbf{Wz}) \right] \\
&= \sum_{n=1}^N E_{z|x_n} [2\boldsymbol{\Psi}^{-1} \mathbf{x}_n \mathbf{z}^T - 2\boldsymbol{\Psi}^{-1} \mathbf{Wz} \mathbf{z}^T] \\
&= 2\boldsymbol{\Psi}^{-1} \sum_{n=1}^N (\mathbf{x}_n E_{z|x_n} [\mathbf{z}^T] - \mathbf{W} E_{z|x_n} [\mathbf{z} \mathbf{z}^T]) = \mathbf{0} \quad (10.55)
\end{aligned}$$

Solving for  $\mathbf{W}$  we get

$$\mathbf{W} = \left( \sum_{n=1}^N \mathbf{x}_n E_{z|x_n}[\mathbf{z}^T] \right) \left( \sum_{n=1}^N E_{z|x_n}[\mathbf{z}\mathbf{z}^T] \right)^{-1} \quad (10.56)$$

where  $E_{z|x_n}[\mathbf{z}]$  and  $E_{z|x_n}[\mathbf{z}\mathbf{z}^T]$  can be found in Eq. (10.48):

$$\begin{cases} E_{z|x_n}[\mathbf{z}] &= \mathbf{m}_{z|x_n} = \mathbf{B}\mathbf{x}_n \\ E_{z|x_n}[\mathbf{z}\mathbf{z}^T] &= \mathbf{\Sigma}_{z|x_n} + \mathbf{m}_{z|x_n}\mathbf{m}_{z|x_n}^T = \mathbf{I} - \mathbf{B}\mathbf{W} + \mathbf{B}\mathbf{x}_n\mathbf{x}_n^T\mathbf{B}^T \end{cases} \quad (10.57)$$

The second equation is due to the fact that  $\mathbf{\Sigma}_z = E[\mathbf{z}\mathbf{z}^T] - \mathbf{m}_z\mathbf{m}_z^T$ . Here  $E_{z|x_n}[\mathbf{z}]$  can be considered as the estimation of the  $\mathbf{z}$ , while  $E_{z|x_n}[\mathbf{z}\mathbf{z}^T]$  the uncertainty of the estimation.

– Find  $\mathbf{\Psi}$ :

$$\begin{aligned} \frac{\partial Q}{\partial \mathbf{\Psi}^{-1}} &= \frac{\partial}{\partial \mathbf{\Psi}^{-1}} \left[ N \log |\mathbf{\Psi}| + \sum_{n=1}^N E_{z|x_n} [\mathbf{x}_n^T \mathbf{\Psi}^{-1} \mathbf{x}_n - 2\mathbf{x}_n^T \mathbf{\Psi}^{-1} \mathbf{W}\mathbf{z} + \mathbf{z}^T \mathbf{W}^T \mathbf{\Psi}^{-1} \mathbf{W}\mathbf{z}] \right] \\ &= -N\mathbf{\Psi} + \sum_{n=1}^N E_{z|x_n} [\mathbf{x}_n\mathbf{x}_n^T - 2\mathbf{x}_n\mathbf{z}^T\mathbf{W}^T + \mathbf{W}(\mathbf{z}\mathbf{z}^T)\mathbf{W}^T] \\ &= -N\mathbf{\Psi} + \sum_{n=1}^N \mathbf{x}_n\mathbf{x}_n^T - 2 \sum_{n=1}^N \mathbf{x}_n E_{z|x_n}[\mathbf{z}]^T \mathbf{W}^T + \mathbf{W} \sum_{n=1}^N E_{z|x_n}[\mathbf{z}\mathbf{z}^T] \mathbf{W}^T = \mathbf{0} \end{aligned} \quad (10.58)$$

Solving for  $\mathbf{\Psi}$  we get

$$\mathbf{\Psi} = \frac{1}{N} \text{diag} \left( \sum_{n=1}^N \mathbf{x}_n\mathbf{x}_n^T - 2 \left( \sum_{n=1}^N \mathbf{x}_n E_{z|x_n}[\mathbf{z}]^T \right) \mathbf{W}^T + \mathbf{W} \left( \sum_{n=1}^N E_{z|x_n}[\mathbf{z}\mathbf{z}^T] \right) \mathbf{W}^T \right) \quad (10.59)$$

where  $\text{diag}(\mathbf{A})$  is the operation that sets all off-diagonal elements of matrix  $\mathbf{A}$  to zero, so that the resulting matrix is indeed diagonal as what  $\mathbf{\Psi}$  should be. We further replace  $\mathbf{W}$  in front of the last term above by that in Eq. (10.56) and get

$$\mathbf{\Psi} = \frac{1}{N} \text{diag} \left( \sum_{n=1}^N \mathbf{x}_n\mathbf{x}_n^T - \sum_{n=1}^N \mathbf{x}_n E_{z|x_n}[\mathbf{z}]^T \mathbf{W}^T \right) \quad (10.60)$$

In summary, here are the steps of the EM method:

1. Initialize parameters  $\theta_{old} = \{\mathbf{W}_{old}, \mathbf{\Psi}_{old}\}$ ;
2. E-step:  
Find  $E_{z|x}(\mathbf{z})$  and  $E_{z|x}(\mathbf{z}\mathbf{z}^T)$  in Eq. (10.57) based on  $\mathbf{m}_{z|x_n}$  and  $\mathbf{\Sigma}_{z|x_n}$  in Eq. (10.48), which in turn is based on  $\theta_{old} = \{\mathbf{W}_{old}, \mathbf{\Psi}_{old}\}$ ;
3. M-step:  
Find  $\theta_{new} = \{\mathbf{W}_{new}, \mathbf{\Psi}_{new}\}$  in Eqs. (10.56) and (10.60), based on  $E_{z|x}(\mathbf{z})$  and  $E_{z|x}(\mathbf{z}\mathbf{z}^T)$ ;
4. Terminate if convergence criterion is satisfied, otherwise replace  $\theta_{old}$  by  $\theta_{new}$  and return to the E-step.

As the E-step and M-step of the EM algorithm for FA are interdependent on each other, they need to be carried out iteratively based on certain initial guess of the parameters in  $\theta$ , as show in the steps below:

## 10.4 Probabilistic PCA

The method of *probabilistic PCA (PPCA)* can be considered as a special case of factor analysis based on the model  $\mathbf{x} = \mathbf{W}\mathbf{z} + \mathbf{e}$ , when the random noise  $\mathbf{e}$  is assumed to have an isotropic normal distributions with covariance matrix  $\Psi = \varepsilon\mathbf{I}$ . More specially, if we further let  $\varepsilon \rightarrow 0$  approach zero, the iteration of the EM algorithm for the PPCA becomes extremely simple, as we will see below.

Same as in Eqs. (10.36) and (10.38) for FA, here we also have the normal distribution of both  $\mathbf{z}$  and  $\mathbf{e}$ :

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad p(\mathbf{e}) = \mathcal{N}(\mathbf{0}, \varepsilon\mathbf{I}) \quad (10.61)$$

and the normal distribution of the observed data  $\mathbf{x} = \mathbf{W}\mathbf{z} + \mathbf{e}$ , same as in Eq. (10.42):

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \mathbf{W}\mathbf{W}^T + \varepsilon\mathbf{I}) \quad (10.62)$$

The goal here is to estimate  $\varepsilon$  and  $\mathbf{W}$  as the model parameter, based on the observed data dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ .

Now Eqs. (10.48) through (10.50) are rewritten as:

$$\begin{cases} \mathbf{m}_{z|x_n} = E_{z|x_n}(\mathbf{z}) = \mathbf{B}\mathbf{x}_n \\ \Sigma_{z|x_n} = Cov_{z|x_n}(\mathbf{z}) = \mathbf{I} - \mathbf{B}\mathbf{W} \end{cases} \quad (10.63)$$

$$\mathbf{B} = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T + \varepsilon\mathbf{I})^{-1} \quad (10.64)$$

$$(\mathbf{W}\mathbf{W}^T + \varepsilon\mathbf{I})^{-1} = \varepsilon^{-1}\mathbf{I} - \varepsilon^{-1}\mathbf{W}(\varepsilon\mathbf{I} + \mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T \quad (10.65)$$

The two steps of the EM algorithm become:

- E-step: Get the expectation of the log-likelihood by replacing  $\Psi$  in Eq. (10.53) by  $\varepsilon\mathbf{I}$ :

$$Q = -\frac{dN}{2} \log \varepsilon - \frac{1}{2\varepsilon} \sum_{n=1}^N E_{z|x_n} [\mathbf{x}_n^T \mathbf{x}_n - 2\mathbf{x}_n^T \mathbf{W}\mathbf{z} + \mathbf{z}^T \mathbf{W}^T \mathbf{W}\mathbf{z}] \quad (10.66)$$

- M-step: Find the optimal model parameters  $\mathbf{W}$  and  $\varepsilon$  that maximize the expectation of the log-likelihood  $Q$  above, by setting the derivative of  $Q$  with respect to the parameters to zero and solve the resulting equations. For  $\mathbf{W}$ , the result is the same as Eq. (10.56):

$$\mathbf{W}_{new} = \left( \sum_{n=1}^N \mathbf{x}_n E_{z|x_n}[\mathbf{z}^T] \right) \left( \sum_{n=1}^N E_{z|x_n}[\mathbf{z}\mathbf{z}^T] \right)^{-1} \quad (10.67)$$

where

$$\begin{cases} E_{z|x_n}[\mathbf{z}] &= \mathbf{m}_{z|x_n} = \mathbf{B}\mathbf{x}_n \\ E_{z|x_n}[\mathbf{z}\mathbf{z}^T] &= \mathbf{\Sigma}_{z|x_n} + \mathbf{m}_{z|x_n}\mathbf{m}_{z|x_n}^T \end{cases} \quad (10.68)$$

To find  $\varepsilon$ , we solve the equation

$$\frac{\partial Q}{\partial \varepsilon} = -\frac{dN}{2\varepsilon} + \frac{1}{2\varepsilon^2} \sum_{n=1}^N E_{z|x_n} [\mathbf{x}_n^T \mathbf{x}_n - 2\mathbf{x}_n^T \mathbf{W}\mathbf{z} + \mathbf{z}^T \mathbf{W}^T \mathbf{W}\mathbf{z}] = 0 \quad (10.69)$$

and get

$$\begin{aligned} \varepsilon &= \frac{1}{dN} \sum_{n=1}^N [\mathbf{x}_n^T \mathbf{x}_n - 2\mathbf{x}_n^T \mathbf{W} E_{z|x_n}(\mathbf{z}) + \text{tr}(\mathbf{W}^T E_{z|x_n}(\mathbf{z}\mathbf{z}^T) \mathbf{W})] \\ &= \frac{1}{dN} \sum_{n=1}^N [ \|\mathbf{x}_n\|^2 - 2E_{z|x_n}(\mathbf{z}^T) \mathbf{W}^T \mathbf{x}_n + \text{tr}(E_{z|x_n}(\mathbf{z}\mathbf{z}^T) \mathbf{W}^T \mathbf{W}) ] \end{aligned} \quad (10.70)$$

where we have used the fact that  $\mathbf{a}^T \mathbf{b} = \text{tr}(\mathbf{a}\mathbf{b}^T)$ .

In summary, here are the steps in the EM algorithm for PPCA:

1. Initialize parameters  $\theta_{old} = \{\mathbf{W}_{old}, \varepsilon_{old}\}$ ;
2. E-step: Based on  $\theta_{old}$ , find  $\mathbf{m}_{z|x_n}$  and  $\mathbf{\Sigma}_{z|x_n}$  in Eq. (10.63), and  $E_{z|x_n}(\mathbf{z})$  and  $E_{z|x_n}(\mathbf{z}\mathbf{z}^T)$  in Eq. (10.68);
3. M-step: Find  $\theta_{new} = \{\mathbf{W}_{new}, \varepsilon_{new}\}$  by evaluating Eqs. (10.67) and (10.70);
4. Terminate if convergence criterion is satisfied, otherwise replace  $\theta_{old}$  by  $\theta_{new}$  and return to step 2.

Specially, if we further assume  $\varepsilon \rightarrow 0$  and the latent variables in  $\mathbf{z}$  are deterministic, i.e.,  $E_{z|x}(\mathbf{z}) = \mathbf{z}$ . At the limit where  $\varepsilon = 0$ ,  $\mathbf{x}$  is simply a linear combination of the latent variables in  $\mathbf{z}$ :

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \varepsilon \mathbf{I} \xrightarrow{\varepsilon \rightarrow 0} \mathbf{W}\mathbf{z} = [\mathbf{w}_1, \dots, \mathbf{w}_{d'}] \begin{bmatrix} z_1 \\ \vdots \\ z_{d'} \end{bmatrix} = \sum_{i=1}^{d'} z_i \mathbf{w}_i \quad (10.71)$$

Comparing this expression of  $\mathbf{x}$  with Eq. (9.44)

$$\mathbf{x} = \mathbf{V}'\mathbf{y} = \sum_{i=1}^{d'} y_i \mathbf{v}_i \quad (10.72)$$

we see that the column vectors in  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_{d'}]$  for the PPCA are similar to the column vectors of  $\mathbf{V}$ , the eigenvector matrix of  $\mathbf{\Sigma}_x$ , in the linear PCA, as both of them can be viewed as the principal directions, an alternative set of basis vectors that also span the same  $d$ -dimensional feature space in which  $\mathbf{x}$  resides. When  $\mathbf{x}$  is expressed as a linear combination of these basis vectors, it can be approximated by only  $d' < d$  of the  $d$  such basis vectors weighted by the  $d'$  principal components in  $\mathbf{y} = [y_1, \dots, y_{d'}]^T$  in linear PCA, or the  $d'$  factors in  $\mathbf{z} = [z_1, \dots, z_{d'}]^T$  in PPCA.

Here are the two EM steps for the PPCA:

- E-step: If  $\varepsilon \rightarrow 0$ , Eq. (10.63) can be written as:

$$\begin{aligned} \mathbf{m}_{z|x_n} &= E_{z|x_n}(\mathbf{z}) = \mathbf{z}_n = \mathbf{B}\mathbf{x}_n = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T + \varepsilon\mathbf{I})^{-1}\mathbf{x}_n \\ &\xrightarrow{\varepsilon \rightarrow 0} \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{x}_n \quad (n = 1, \dots, N) \end{aligned} \quad (10.73)$$

All  $N$  such equations can be written in matrix form as

$$\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N] = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}[\mathbf{x}_1, \dots, \mathbf{x}_N] = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{X} \quad (10.74)$$

As  $\mathbf{W}$  is a  $d \times d'$  matrix ( $d' < d$ ), the rank of the  $d \times d$  matrix  $\mathbf{W}\mathbf{W}^T$  is at most  $d'$ , i.e., the inverse of  $\mathbf{W}\mathbf{W}^T$  does not exist and we cannot find  $\mathbf{z}_n = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{x}_n$ . In fact the transformation matrix  $\mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}$  is the *right pseudo inverse* (Section A.15) of the  $d \times d'$  matrix  $\mathbf{W}$ , which exists only if  $d' \geq d$ .

However, we note that the model  $\mathbf{x} = \mathbf{W}\mathbf{z}$  is an over-constrained linear system of  $d$  equations but only  $d' < d$  variables in  $\mathbf{z} = [z_1, \dots, z_{d'}]^T$ , and the least-squares solution that minimizes the squared error  $\|\mathbf{x} - \mathbf{W}\mathbf{z}\|^2$  can be found by the *left pseudo inverse* (Section A.15) of  $\mathbf{W}$ :

$$\mathbf{z}_n = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\mathbf{x}_n = \mathbf{W}^-\mathbf{x}_n \quad (n = 1, \dots, N) \quad (10.75)$$

and we can get the latent variables in  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$  given all  $N$  data points in  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ :

$$\mathbf{Z} = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\mathbf{X} = \mathbf{W}^-\mathbf{X} \quad (10.76)$$

which is based on the old value of  $\mathbf{W}$  previously estimated.

- M-step: Now Eq. (10.67) can be written as:

$$\begin{aligned} \mathbf{W} &= \left( \sum_{n=1}^N \mathbf{x}_n E_{z|x_n}(\mathbf{z}^T) \right) \left( \sum_{n=1}^N E_{z|x_n}(\mathbf{z}\mathbf{z}^T) \right)^{-1} \\ &= [\mathbf{x}_1, \dots, \mathbf{x}_N] \begin{bmatrix} \mathbf{z}_1^T \\ \vdots \\ \mathbf{z}_N^T \end{bmatrix} \left( \begin{bmatrix} \mathbf{z}_1, \dots, \mathbf{z}_N \end{bmatrix} \begin{bmatrix} \mathbf{z}_1^T \\ \vdots \\ \mathbf{z}_N^T \end{bmatrix} \right)^{-1} = \mathbf{X}\mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T)^{-1} \quad (\neq \mathbf{1}\mathbf{X}\mathbf{Z}) \end{aligned}$$

by which  $\mathbf{W}$  is updated based on  $\mathbf{Z}$  found previously in the E-step. We note that  $\mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T)^{-1}$  is the right pseudo inverse of the  $d \times N$  matrix  $\mathbf{Z}$ , which exists if  $d \leq N$ .

Again, as the E-step and M-steps are interdependent on each other, they need to be carried out iteratively:

$$\begin{aligned} \text{E-step:} & \quad \mathbf{Z} = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\mathbf{X} = \mathbf{W}^-\mathbf{X} \\ \text{M-Step:} & \quad \mathbf{W} = \mathbf{X}\mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T)^{-1} = \mathbf{X}\mathbf{Z}^- \end{aligned} \quad (10.78)$$

We note that given the dataset  $\mathbf{X}$ , the FA model  $\mathbf{x} = \mathbf{W}\mathbf{z}$  can be expressed as an over-determined matrix equation

$$\mathbf{X}_{d \times N} = [\mathbf{x}_1, \dots, \mathbf{x}_N] = \mathbf{W}_{d \times d'} [\mathbf{z}_1, \dots, \mathbf{z}_N] = \mathbf{W}_{d \times d'} \mathbf{Z}_{d' \times N} \quad (10.79)$$

or, taking transpose of the equation, we get

$$\mathbf{X}_{N \times d}^T = \mathbf{Z}_{N \times d'}^T \mathbf{W}_{d' \times d}^T \quad (10.80)$$

Given  $\mathbf{X}$ , this equation can be interpreted in two ways:

- If  $\mathbf{W}$  is available, this is a system of  $d$  equations but  $d' < d$  unknowns in each of the  $N$  columns of  $\mathbf{Z}$ , which is solved as in the E-step above;
- If  $\mathbf{Z}$  is available, this is a system of  $N$  equations but  $d' < N$  unknowns in each of the  $d$  rows of  $\mathbf{W}$ , which is solved as in the M-step above.

In either step, the pseudo inverse method can be used to minimize the squared error  $\|\mathbf{X} - \mathbf{W}\mathbf{Z}\|^2$ .

When compared to the regular PCA method that finds all  $d$  eigenvalues and the corresponding eigenvectors all at once by solving the eigenequation  $\Sigma_x \mathbf{w} = \lambda \mathbf{w}$ , the PPCA method only finds the  $d'$  column vectors of matrix  $\mathbf{W}$  as the basis vectors, as the basis vectors that span a subspace with much reduced dimensionality but containing the most essential information in the data. However, unlike the PCA, the basis vectors in PPCA are not necessarily orthogonal to each other.

The implementation of the PPCA is extremely simple, as shown in the Matlab code below. Based on some initialized  $\mathbf{W}$ , the EM iteration is composed of the E-step and M-step:

```

while er>tol
    Z=inv(W'*W)*W'*X;           % find Z given W in E-step
    Wnew=X*Z'*inv(Z*Z');        % find W given Z in M-step
    er=norm(Wnew-W);            % the LS error
    W=Wnew;
end

```

#### Example 1:

The figure below shows the first few iterations of the PPCA based on the EM method presented above, for a set of  $N = 1000$  data points in  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  the  $d = 2$  dimensional space. The straight line in the plots indicates the direction of  $\mathbf{w}_1$ , which is initially set along a random direction, but quickly approaches the direction corresponding to the principal component obtained by the PCA method, along which the data points are most widely spread.

#### Example 2:

The figure below shows the first few iterations of the PPCA EM algorithm for a set of data points in  $d = 3$  dimensional space. The straight lines in the plots indicates the direction of the principal components when  $d' = 2$ .

#### Example 3:



The figure below shows the visualization of the dataset for the hand-written numbers from 0 to 9, containing 2240 data points in a 256-D space, but now linearly mapped to a 3-D space spanned by  $d' = 3$  factors found by the PPCA method.

## 10.5 Classical Multidimensional Scaling

The goal of *multidimensional scaling (MDS)* is to reduce the dimensionality of the dataset representing a set of  $N$  objects of interest, each described by a set of  $d$  features and represented as a vector  $\mathbf{x} = [x_1, \dots, x_d]^T$  in a  $d$ -dimensional space, while the pairwise similarity relationship between any two of these objects are preserved. MDS can be used to visualize datasets in a high dimensional space. Specially, here we consider the *classical MDS (cMDS)*, also known as *principal coordinates analysis (PCoA)*, as one of the methods in MDS.

Specifically, we represent the pairwise similarity of a set of  $N$  objects  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  by the  $N \times N$  similarity matrix  $\mathbf{D}_x = [d_{ij}^x]$ , of which the  $ij$ -th component  $d_{ij}^x$  is certain measurement of the similarity between  $\mathbf{x}_i$  and  $\mathbf{x}_j$ . The goal is to map the dataset  $\mathbf{X}$  in the  $d$ -dimensional space into  $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]$  in a lower  $d'$ -dimensional space, in which the similarity matrix  $\mathbf{D}_x$  is optimally approximated by  $\mathbf{D}_y = [d_{ij}^y]$ . When  $d' \leq 3$ , these data points in  $\mathbf{Y}$  can be visualized. MDS can be considered as a method for dimension reduction. In some cases only the pairwise similarities  $d_{ij}^x$  ( $i, j = 1, \dots, N$ ) are given, while the  $N$  objects in  $\mathbf{X}$  are not explicitly given, and the dimensionality  $d$  may be unknown.

We assume the given pairwise similarity is the Euclidean distance  $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$  between  $\mathbf{x}_i$  and  $\mathbf{x}_j$ :

$$d_{ij}^2 = \|\mathbf{x}_i - \mathbf{x}_j\|^2 = (\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i^T \mathbf{x}_j + \mathbf{x}_j^T \mathbf{x}_j \quad (10.81)$$

and get

$$\mathbf{D}_x^2 = \begin{bmatrix} d_{ij}^2 \\ \vdots \\ d_{ij}^2 \end{bmatrix}_{N \times N} = \begin{bmatrix} \mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i^T \mathbf{x}_j + \mathbf{x}_j^T \mathbf{x}_j \\ \vdots \\ \mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i^T \mathbf{x}_j + \mathbf{x}_j^T \mathbf{x}_j \end{bmatrix}_{N \times N} = \mathbf{X}_r - 2\mathbf{X}^T \mathbf{X} + \mathbf{X}_c \quad (10.82)$$

where

$$\mathbf{X}_r = \begin{bmatrix} \|\mathbf{x}_1\|^2 & \cdots & \|\mathbf{x}_1\|^2 \\ \vdots & \ddots & \vdots \\ \|\mathbf{x}_N\|^2 & \cdots & \|\mathbf{x}_N\|^2 \end{bmatrix} = \mathbf{X}_c^T$$

$$\mathbf{X}^T \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} [\mathbf{x}_1, \dots, \mathbf{x}_N] = \begin{bmatrix} \mathbf{x}_1^T \mathbf{x}_1 & \cdots & \mathbf{x}_1^T \mathbf{x}_N \\ \vdots & \ddots & \vdots \\ \mathbf{x}_N^T \mathbf{x}_1 & \cdots & \mathbf{x}_N^T \mathbf{x}_N \end{bmatrix} \quad (10.83)$$

Our goal is to find  $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]$  in a lower dimensional space so that its pairwise similarity matrix  $\mathbf{D}_y$  matches  $\mathbf{D}_x$  optimally, in the sense that the following objective function is minimized:

$$J(\mathbf{Y}) = \|\mathbf{D}_y^2 - \mathbf{D}_x^2\| \quad (10.84)$$

We note that such a solution  $\mathbf{Y}$  is not unique, as any translated version of  $\mathbf{Y}$  has the same pairwise similarity matrix and is also a solution. We therefore seek to find a unique solution  $\mathbf{Y}$  in which all data points are centralized, i.e, the mean of all points in  $\mathbf{Y}$  is zero:

$$\bar{\mathbf{y}} = \frac{1}{N} \sum_{n=1}^N \mathbf{y}_n = \mathbf{0} \quad (10.85)$$

To do so, we introduce an  $N \times N$  symmetric centering matrix  $\mathbf{C} = \mathbf{I} - \mathbf{1}/N = \mathbf{C}^T$ , where  $\mathbf{1} = \mathbf{e}\mathbf{e}^T$  is an  $N \times N$  matrix with all components equal 1. Premultiplying  $\mathbf{C}$  to a column vector  $\mathbf{a} = [a_1, \dots, a_N]^T$  removes the mean  $\bar{a} = \sum_{i=1}^N a_i/N$  of  $\mathbf{a}$  from each of its component:

$$\mathbf{C}\mathbf{a} = \mathbf{I}\mathbf{a} - \frac{1}{N}\mathbf{1}\mathbf{a} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix} - \frac{1}{N} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} a_1 - \bar{a} \\ \vdots \\ a_N - \bar{a} \end{bmatrix} \quad (10.86)$$

Taking transpose of the equation above, we get

$$(\mathbf{C}\mathbf{a})^T = \mathbf{a}^T \mathbf{C} = [a_1 - \bar{a}, \dots, a_N - \bar{a}] \quad (10.87)$$

i.e., postmultiplying  $\mathbf{C}$  to a row vector  $\mathbf{a}^T = [a_1, \dots, a_N]$  removes the mean  $\bar{a}$  from each of its component.

We can now postmultiply  $\mathbf{C}$  to  $\mathbf{X}$  to remove the mean of each row of  $\mathbf{X}$ :

$$\mathbf{X}\mathbf{C} = [\mathbf{x}_1, \dots, \mathbf{x}_N]\mathbf{C} = [\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_N] = \bar{\mathbf{X}} \quad (10.88)$$

i.e., the  $n$ th column becomes:

$$\bar{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}, \quad \text{where} \quad \bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \quad (10.89)$$

and premultiply  $\mathbf{C}$  to  $\mathbf{X}^T$  to remove the mean of each column of  $\mathbf{X}^T$ :

$$\mathbf{C}\mathbf{X}^T = \bar{\mathbf{X}}^T \quad (10.90)$$

We further apply double centering to  $\mathbf{D}_x^2$  by pre and post-multiplying  $\mathbf{C}$  and get:

$$\begin{aligned} \mathbf{C}\mathbf{D}_x^2\mathbf{C} &= \mathbf{C}(\mathbf{X}_r - 2\mathbf{X}^T\mathbf{X} + \mathbf{X}_c)\mathbf{C} = \mathbf{C}\mathbf{X}_r\mathbf{C} - 2\mathbf{C}(\mathbf{X}^T\mathbf{X})\mathbf{C} + \mathbf{C}\mathbf{X}_c\mathbf{C} \\ &= -2\mathbf{C}(\mathbf{X}^T\mathbf{X})\mathbf{C} = -2(\mathbf{X}\mathbf{C})^T(\mathbf{X}\mathbf{C}) = -2\bar{\mathbf{X}}^T\bar{\mathbf{X}} \end{aligned}$$

Note that the first term  $\mathbf{C}\mathbf{X}_r\mathbf{C} = \mathbf{C}(\mathbf{X}_r\mathbf{C}) = \mathbf{C}\mathbf{0} = \mathbf{0}$ , as all components of each row of  $\mathbf{X}_r$  are the same as their mean, and removing the mean results in a zero

row vector,  $\mathbf{X}_r \mathbf{C} = \mathbf{0}$ ; and the last term  $\mathbf{C} \mathbf{X}_c \mathbf{C} = (\mathbf{C} \mathbf{X}_c) \mathbf{C} = \mathbf{0} \mathbf{C} = \mathbf{0}$ , as all components of each column of  $\mathbf{X}_c$  are the same as their mean, and removing the mean results in a zero column vector.

Now the similarity matrix  $\mathbf{D}_x^2$  of all centralized data points in  $\bar{\mathbf{X}}$  with zero mean can be represented by the *Gram matrix* of  $\bar{\mathbf{X}}$ :

$$\mathbf{G}_{\bar{x}} = \bar{\mathbf{X}}^T \bar{\mathbf{X}} = -\frac{1}{2} \mathbf{C} \mathbf{D}_x^2 \mathbf{C} \quad (10.91)$$

We further assume all data points in  $\mathbf{Y}$  are also centralized with zero mean and their corresponding similarity matrix is also represented by  $\mathbf{G}_{\bar{y}}$ , then the objective function can be expressed as:

$$J(\mathbf{Y}) = \|\mathbf{G}_{\bar{y}} - \mathbf{G}_{\bar{x}}\| = \|\bar{\mathbf{Y}}^T \bar{\mathbf{Y}} - \mathbf{G}_{\bar{x}}\| \quad (10.92)$$

To find  $\bar{\mathbf{Y}}$  that minimizes  $J(\mathbf{Y})$ , we first consider the equation  $\bar{\mathbf{Y}}^T \bar{\mathbf{Y}} - \mathbf{G}_{\bar{x}} = \mathbf{0}$  so that  $J(\mathbf{Y}) = 0$ . In other words, we desire to express  $\mathbf{G}_{\bar{x}}$  as the inner product of some matrix with itself. To do so, we carry out eigenvalue decomposition of  $\mathbf{G}_x$  to find its eigenvalue matrix  $\Lambda$  and the orthogonal eigenvector matrix  $\mathbf{V}$  satisfying  $\mathbf{G}_x \mathbf{V} = \mathbf{V} \Lambda$ , i.e.,

$$\begin{aligned} \mathbf{G}_{\bar{x}} &= \mathbf{V} \Lambda \mathbf{V}^{-1} = \mathbf{V} \Lambda \mathbf{V}^T = [\mathbf{v}_1, \dots, \mathbf{v}_N] \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_N \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_N^T \end{bmatrix} \\ &= (\mathbf{V} \Lambda^{1/2}) (\Lambda^{1/2} \mathbf{V}^T) = (\Lambda^{1/2} \mathbf{V}^T)^T (\Lambda^{1/2} \mathbf{V}^T) \end{aligned}$$

Note that as  $\mathbf{G}_{\bar{x}}$  is symmetric, the eigenvalues are real and the eigenvectors are orthogonal, i.e.,  $\mathbf{V}^T = \mathbf{V}^{-1}$  is orthogonal matrix. Now we can find the desired  $\mathbf{Y}$  as:

$$\mathbf{Y} = \Lambda^{1/2} \mathbf{V}^T = \begin{bmatrix} \sqrt{\lambda_1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sqrt{\lambda_N} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_N^T \end{bmatrix} = \begin{bmatrix} \sqrt{\lambda_1} \mathbf{v}_1^T \\ \vdots \\ \sqrt{\lambda_N} \mathbf{v}_N^T \end{bmatrix} \quad (10.93)$$

Each column  $\mathbf{y}_i$  in  $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]$  is an N-D vector. If all  $N$  dimensions of these  $\mathbf{y}_n$  are used,  $\mathbf{Y}^T \mathbf{Y} = \mathbf{G}_x$  and  $J(\mathbf{Y}) = 0$ . To reduce the dimensionality to  $d' < d$ , we construct  $\mathbf{Y}'$  by the first  $d'$  rows of  $\mathbf{Y}$  corresponding to the  $d'$  greatest eigenvalues of  $\lambda_1 \geq \dots \geq \lambda_{d'} \geq \dots \geq \lambda_N$  and their corresponding eigenvectors  $\mathbf{v}_1, \dots, \mathbf{v}_{d'}$ . The error is:

$$J(\mathbf{Y}') = \|\mathbf{Y}'^T \mathbf{Y}' - \mathbf{Y}^T \mathbf{Y}\| = \left\| \sum_{i=1}^{d'} \lambda_i \mathbf{v}_i \mathbf{v}_i^T - \sum_{i=1}^N \lambda_i \mathbf{v}_i \mathbf{v}_i^T \right\| = \left\| \sum_{i=d'+1}^N \lambda_i \mathbf{v}_i \mathbf{v}_i^T \right\| \quad (10.94)$$

In summary, here is the PCoA algorithm:

- Given the pairwise similarity, an  $N \times N$  matrix  $\mathbf{D}_x = [d_{ij}]$ , construct the squared proximity matrix  $\mathbf{D}_x^2 = [d_{ij}^2]$ ;
- Apply double centering to get  $\mathbf{G}_x = -\mathbf{C} \mathbf{D}_x^2 \mathbf{C} / 2$ ;

- Find the  $d'$  greatest eigenvalues  $\lambda_1, \dots, \lambda_{d'}$  of  $\mathbf{G}_x$ , and the corresponding eigenvectors  $\mathbf{v}_1, \dots, \mathbf{v}_{d'}$ ;
- Get the map

$$\mathbf{Y}_{d' \times N} = \mathbf{\Lambda}_{d'}^{1/2} \mathbf{V}_{d'} = \begin{bmatrix} \sqrt{\lambda_1} \mathbf{v}_1^T \\ \vdots \\ \sqrt{\lambda_{d'}} \mathbf{v}_{d'}^T \end{bmatrix} \quad (10.95)$$

The Matlab functions for the PCA and PCoA are listed below:

```
function Y=PCA(X)
    [V D]=eig(cov(X'));           % solve eigenequation for covariance of data
    [d idx]=sort(diag(D),'descend'); % sort eigenvalues in descend order
    V=V(:,idx);                   % reorder eigenvectors
    Y=V'*X;                       % carry out KLT
end
```

```
function Y=PCoA(X)
    N=size(X,2);
    D=zeros(N);                  % pairwise similarity matrix
    C=eye(N)-ones(N)/N;         % centering matrix
    for i=1:N
        for j=1:N
            D(i,j)=(norm(X(:,i)-X(:,j)))^2;
        end
    end
    B=-C*D*C/2;
    [V D]=eig(B);                % solve eigenequation for matrix B
    [d idx]=sort(diag(D),'descend'); % sort eigenvalues in descend order
    V=V(:,idx);                  % reorder eigenvectors
    Y=sqrt(D)*V';                % carry out PCoA
end
```

### Example 1:

In the figure below, the top panel shows the locations of some major cities in North America, and the bottom panel shows the reconstructed city locations based on the pairwise distance of these cities. Note that the reconstructed map is a rotated and centralized version of the original map.

### Example 2:

Both PCA and PCoA are applied to the Iris dataset of 150 4-D data points, and the figure below shows the data points in 2-D and 3-D spaces corresponding to the first 2 and 3 eigenvalues.

### Example 3:

Both PCA and PCoA are applied to the dataset of handwritten numbers of

2240 256-D data points, and the figure below shows the data points in 2-D and 3-D spaces corresponding to the first 2 and 3 eigenvalues.

## 10.6 t-Distributed Stochastic Neighbor Embedding (t-SNE)

It is impossible to reduce the dimensionality of a given dataset which is intrinsically high-dimensional (high-D), while still preserving all the pairwise distances in the resulting low-dimensional (low-D) space, compromise will have to be made to sacrifice certain aspects of the dataset when the dimensionality is reduced.

The PCA method as a linear, dimensionality reduction algorithm,

is unable to interpret complex nonlinear relationship between features. In a dimensionality preserving method, the PCA emphasizes the importance of the PCA dimensions with great variances, but neglects small distance variations. It places distant and dissimilar data points far apart in the low-D space, while ignoring the importance of similar datapoints close together in the high-D, which need to be also placed close together in the low-D space. Therefore the PCA method tends to preserve large scale global structure of the dataset, while ignoring the small scale local structure.

But in some cases the local structures are of equal or even more importance than the global ones, such as the typical example of “Swiss roll”.

Also, when the dimensionality is much reduced from a high-D space to a low-D space, the crowding problem occurs. In a high-D space, a data point may have a large number of similar neighbors of roughly equal distance; however, in a low-D space, the number of equal-distance neighbors is significantly reduced (8 in a 3-D space, 4 in a 2-D space, 2 in a 1-D space). Thus the space available for the neighbors of a point is much reduced. As the result, when a large number of neighbors in the high-D are mapped to a low-D space, they are forced to spread out, some are getting closer to those that are farther away in the original space, thereby reducing the gaps between potential clusters of similar points.

In contrast, the t-SNE method is a nonlinear method that is based on probability distributions of the data points being neighbors, and it attempts to preserve the structure at all scales, but emphasizing more on the small scale structures, by mapping nearby points in high-D space to nearby points in low-D space.

The similarity between any two distinct points  $\mathbf{x}_i$  and  $\mathbf{x}_j$  in the high-D space is defined as a probability based on the Gaussian kernel:

$$p_{ij} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)}{\sum_k \sum_{l \neq k} \exp(-\|\mathbf{x}_k - \mathbf{x}_l\|^2/2\sigma^2)}, \quad \sum_i \sum_j p_{i/j} = 1 \quad (10.96)$$

However, if  $\mathbf{x}_i = \mathbf{x}_j$ ,  $p_{ij} = 0$  is defined to be zero, as it never needed in the algorithm. Note that  $p_{ij}$  is a probability measurement as it is normalized and add up to 1.

If  $\mathbf{x}_j$  is close to  $\mathbf{x}_i$ ,  $\|\mathbf{x}_i - \mathbf{x}_j\|$  is small and  $p_{ij}$  is large, indicating  $\mathbf{x}_j$  is similar to  $\mathbf{x}_i$ ; but if  $\mathbf{x}_j$  is far away from  $\mathbf{x}_i$ ,  $\|\mathbf{x}_i - \mathbf{x}_j\|$  is large and  $p_{ij}$  is small, indicating  $\mathbf{x}_j$

is dissimilar to  $\mathbf{x}_i$ . Also, as  $\sum_i \sum_j p_{ij} = 1$ ,  $p_{ij}$  can be considered as a probability that any two points  $\mathbf{x}_j$  and  $\mathbf{x}_i$  are neighbors in the feature space and are therefore similar to each other.

Similarly, we could also define the similarity  $q_{ij}$  in the low-D space also based on the Gaussian kernel. However, for reason explained below, we prefer to define the similarity  $q_{in}$  in the low-D space based on Student's t-distribution of degree 1, instead of Gaussian distribution:

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}} \quad (10.97)$$

As shown in the figure below, when compared with the Gaussian distribution, the t-distribution has a lower peak in the center but higher tails on the sides. It is because of this property, the t-distribution is used to address the crowding problem. Let  $\mathbf{x}_i$  be the point of interest (black, in the center of the Gaussian kernel), then another point  $\mathbf{x}_j$  (red) is mapped to a different position, which is closer to  $\mathbf{x}_i$  if they are initially close (similar), or farther away from each other if they are initially far apart (dissimilar).

How well the points  $\mathbf{y}_i$  in the low-D space represent the similarity relationship among the points  $\mathbf{x}_i$  in the original high-D space can be measured by the discrepancy between the two sets of probabilities, measured by the cost function, defined as the Kullback-Leibler divergence (Section 3.2.2):

$$C = D_{KL}(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} = \sum_i \sum_j (p_{ij} \log p_{ij} - p_{ij} \log q_{ij}) \quad (10.98)$$

We desire to find the optimal map points  $\mathbf{y}_i$  in the low-D space that minimize the discrepancy  $C = D_{KL}(P||Q)$ .

The KL-divergence is not symmetric as  $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ , therefore it can not be viewed as a distance between the two sets of probabilities. This asymmetry is actually a desired feature. Consider the following two cases:

- If  $p_{ij}$  is large but  $q_{ij}$  is small, the cost  $C$  is very large, i.e., and such a mapping is heavily penalized. As the result, similar points that are close to each other in the high-D space tend to be mapped to points in the low-D space that are also close to each other.
- If  $p_{ij}$  is small but  $q_{ij}$  is large, the cost  $C$  is small, i.e., such a mapping is not heavily penalized. As the result, dissimilar points that are far apart in the high-D space are allowed to be mapped to points in the low-D space that are closer to each other.

We therefore see that minimizing the KL-divergence has the effect of preserving the local structure of the dataset, but not necessarily the global structure.

Before proceeding to consider the minimization of the cost function  $C$ , we still need to resolve two additional issues:

- One potential problem with  $p_{ij}$  defined above is that when a point  $\mathbf{x}_i$  is an outlier with large distances  $\|\mathbf{x}_i - \mathbf{x}_j\|$  to every other point  $\mathbf{x}_j$  in the dataset,

$p_{ij}$  is small and insignificant in the cost function  $C$ , and the location of  $\mathbf{y}_i$  in the low-D space can not be well determined. To resolve this issue, we redefine  $p_{ij}$  as the average of two conditional probabilities:

$$p_{ij} = \frac{p_{i/j} + p_{j/i}}{2n} \quad (10.99)$$

where  $n$  is the total number of data points and

$$p_{i/j} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}, \quad \sum_i p_{j/i} = 1 \quad (10.100)$$

is the conditional probability that any point  $\mathbf{x}_j$  is a neighboring point similar to a given point  $\mathbf{x}_i$ . Note that  $p_{j/i}$  is properly normalized and add up to 1:

$$\sum_{j=1}^n \sum_{i=1}^n p_{i/j} = \frac{1}{2n} \sum_{i=1}^n \left( \sum_{j=1}^n p_{j/i} \right) + \frac{1}{2n} \sum_{j=1}^n \left( \sum_{i=1}^n p_{i/j} \right) = \frac{1}{2n} \sum_{i=1}^n 1 + \frac{1}{2n} \sum_{j=1}^n 1 = \frac{1}{2n} (n+n) = 1 \quad (10.101)$$

i.e., it is a probability measurement.

- Whether two points are close (therefore similar) to each other needs to be measured relatively, independent of the local data density, which may vary in different local regions of the feature space. We can therefore use different variance  $\sigma_i$  for  $p_{j/i}$  depending on the local data density in the neighborhood of  $\mathbf{x}_i$ . Specifically, we choose  $\sigma_i$  in such a way that the *perplexity* (Section 3.2.1)  $2^{H(p_i)}$  is the same for all data points, i.e., the entropy  $H(p_i) = -\sum_j p_{j/i} \log_2 p_{j/i}$  representing the uncertainty is the same, independent of the local data density. As shown in Section 3.2.1, the entropy of a Gaussian  $p(x, \mu, \sigma)$  is  $\log(2\pi e\sigma^2)/2$ , a monotonic function of its variance  $\sigma$ .

We can now find the optimal map points  $\mathbf{y}_i$  that minimize the cost function  $C = D_{KL}(P||Q)$ , by gradient descent method.

To find the gradient of  $\partial C/\partial \mathbf{y}_i$ , we first introduce a set of intermediate variables

$$d_{kl} = \|\mathbf{y}_k - \mathbf{y}_l\| = [(\mathbf{y}_k - \mathbf{y}_l)^T (\mathbf{y}_k - \mathbf{y}_l)]^{1/2} \quad (k, l = 1, \dots, n), \quad Z = \sum_k \sum_{l \neq k} (1 + d_{kl}^2)^{-1} \quad (10.102)$$

so that

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}} = \frac{(1 + d_{ij}^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + d_{kl}^2)^{-1}} = \frac{(1 + d_{ij}^2)^{-1}}{Z} \quad (10.103)$$

Note that

$$\frac{\partial Z}{\partial d_{ij}} = \frac{\partial}{\partial d_{ij}} \left[ \sum_k \sum_{l \neq k} (1 + d_{kl}^2)^{-1} \right] = -2d_{ij} (1 + d_{ij}^2)^{-2} \quad (10.104)$$

We now find  $\partial C/\partial \mathbf{y}_i$  based on chain-rule. Note that when  $\mathbf{y}_i$  changes, only  $d_{ij}$  and  $d_{ji}$  (for all  $j$ ) change, we therefore have

$$\frac{\partial C}{\partial \mathbf{y}_i} = \sum_j \frac{\partial C}{\partial d_{ij}} \frac{\partial d_{ij}}{\partial \mathbf{y}_i} + \sum_j \frac{\partial C}{\partial d_{ji}} \frac{\partial d_{ji}}{\partial \mathbf{y}_i} \quad (10.105)$$

But as

$$\frac{\partial d_{ij}}{\partial \mathbf{y}_i} = \frac{\partial d_{ji}}{\partial \mathbf{y}_i} = \frac{\partial}{\partial \mathbf{y}_i} [(\mathbf{y}_i - \mathbf{y}_j)^T (\mathbf{y}_i - \mathbf{y}_j)]^{1/2} = \frac{1}{2} [(\mathbf{y}_i - \mathbf{y}_j)^T (\mathbf{y}_i - \mathbf{y}_j)]^{-1/2} 2(\mathbf{y}_i - \mathbf{y}_j) = \frac{\mathbf{y}_i - \mathbf{y}_j}{d_{ij}} \quad (10.106)$$

we have

$$\frac{\partial C}{\partial \mathbf{y}_i} = \sum_j \left( \frac{\partial C}{\partial d_{ij}} + \frac{\partial C}{\partial d_{ji}} \right) \frac{\mathbf{y}_i - \mathbf{y}_j}{d_{ij}} = 2 \sum_j \frac{\partial C}{\partial d_{ij}} \frac{\mathbf{y}_i - \mathbf{y}_j}{d_{ij}} \quad (10.107)$$

We next find  $\partial C/\partial d_{ij}$ . Note that only  $q_{kl}$  may be a function of  $d_{ij}$ , while all  $p_{kl}$  are constants, we therefore have

$$\begin{aligned} \frac{\partial C}{\partial d_{ij}} &= \frac{\partial}{\partial d_{ij}} \left[ \sum_k \sum_{l \neq k} (p_{kl} \log p_{kl} - p_{kl} \log q_{kl}) \right] = - \sum_k \sum_{l \neq k} p_{kl} \frac{\partial}{\partial d_{ij}} \log q_{kl} \\ &= - \sum_k \sum_{l \neq k} p_{kl} \frac{\partial}{\partial d_{ij}} (\log q_{kl} Z - \log Z) = \sum_k \sum_{l \neq k} p_{kl} \left( -\frac{1}{q_{kl} Z} \frac{\partial}{\partial d_{ij}} (1 + d_{kl}^2)^{-1} + \frac{1}{Z} \frac{\partial Z}{\partial d_{ij}} \right) \\ &= p_{ij} \left( \frac{1}{q_{ij} Z} (1 + d_{ij}^2)^{-2} 2d_{ij} \right) - \left( \sum_k \sum_{l \neq k} p_{kl} \right) \frac{1}{Z} (1 + d_{ij}^2)^{-2} 2d_{ij} \\ &= 2d_{ij} \left[ \frac{p_{ij} (1 + d_{ij}^2)^{-2}}{q_{ij} Z} - \frac{(1 + d_{ij}^2)^{-2}}{Z} \right] \end{aligned} \quad (10.108)$$

The second to the last equality is due to  $\partial(1 + d_{kl}^2)^{-1}/\partial d_{ij} = 0$  when  $i \neq k$  or  $j \neq l$ , and the last equality is due to  $\sum_k \sum_{l \neq k} p_{kl} = 1$ . But as

$$\frac{(1 + d_{ij}^2)^{-2}}{Z} = q_{ij} (1 + d_{ij}^2)^{-1} \quad (10.109)$$

The above becomes

$$\frac{\partial C}{\partial d_{ij}} = 2d_{ij} \left( \frac{p_{ij}}{q_{ij}} - 1 \right) \frac{(1 + d_{ij}^2)^{-2}}{Z} = 2d_{ij} (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1} \quad (10.110)$$

We can now finally get the gradient vector of the cost function  $C$  with respect to each of the map points  $\mathbf{y}_i$ :

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{y}_i} &= 2 \sum_j \frac{\partial C}{\partial d_{ij}} \frac{\mathbf{y}_i - \mathbf{y}_j}{d_{ij}} = 4(p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1} (\mathbf{y}_i - \mathbf{y}_j) \\ &= 4 \sum_j (p_{ij} - q_{ij}) (1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1} (\mathbf{y}_i - \mathbf{y}_j) \end{aligned} \quad (10.111)$$



---

Given the gradient vector, we can iteratively approach the optimal data points in the low-D space that minimize the discrepancy  $C = D_{KL}(P||Q)$ :

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \delta \frac{\partial C}{\partial \mathbf{y}} + \alpha_n (\mathbf{y}_n - \mathbf{y}_{n-1}) \quad (10.112)$$

where  $\delta$  is the step size (learning rate) and the second term is momentum term.

# 11 Independent Component Analysis

---

## Independent Component Analysis

### 11.1 Independence and Non-Gaussianity

The goal of *independent component analysis (ICA)* is to reconstruct a set of latent source signals denoted by  $\mathbf{s} = [s_1, \dots, s_m]^T$  and assumed to be independent of each other, thereby the term *independent components*, from the observed variables  $\mathbf{x} = [x_1, \dots, x_n]^T$  as the linear combinations of the source signals,

$$\mathbf{x} = \mathbf{A}\mathbf{s} = \sum_{i=1}^m \mathbf{a}_i s_i \quad (11.1)$$

where  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_m]$  is an  $n \times m$  *mixing matrix* of unknown values. We assume  $n \geq m$  to avoid under determination. Given a set of observed data samples in dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  with  $\mathbf{x}_t = \mathbf{A}\mathbf{s}_t$ , the ICA is to recover the independent source signals  $\mathbf{s}_1, \dots, \mathbf{s}_N$ , based on a linear reconstruction model

$$\mathbf{y} = \mathbf{W}\mathbf{x} = \mathbf{W}\mathbf{A}\mathbf{s} \quad (11.2)$$

where  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_n]$  is a  $m \times n$  reconstruction matrix, and  $\mathbf{y} = [y_1, \dots, y_m]^T$  contains the  $m$  reconstructed signals to match the independent source signals in  $\mathbf{s}$ . Here  $i$ th component of  $\mathbf{y}$  is  $y_i = \mathbf{w}_i^T \mathbf{x}$  where  $\mathbf{w}_i^T$  is the  $i$ th row vector of  $\mathbf{W}$ .

As ICA is an unsupervised method without training data for the ground truth regarding either  $\mathbf{A}$  or  $\mathbf{s}$ , we cannot find the reconstruction matrix  $\mathbf{W}$  in closed form by the least squares method to minimize the squared error  $\|\mathbf{y} - \mathbf{s}\|^2$ . Instead we have to find the optimal  $\mathbf{W}$  based only on the desired requirement that the components in  $\mathbf{y} = \mathbf{W}\mathbf{x}$  need to be as independent of each other as possible.

The method of ICA can be compared with the PCA. Given the observed data  $\mathbf{x}$ , both methods find a matrix  $\mathbf{W}$  so that each data point as a vector  $\mathbf{x}$  in the space is mapped to another vector  $\mathbf{y} = \mathbf{W}\mathbf{x}$ . In PCA, the transform matrix  $\mathbf{W} = \mathbf{V}^T$  (Eq. (9.18)) is the eigenvector matrix of the covariance matrix of  $\mathbf{x}$ , i.e., the data points are projected onto the eigenvectors  $\mathbf{v}_i$  as the orthogonal basis of the space, and components  $y_i = \mathbf{v}_i^T \mathbf{x}$  of  $\mathbf{y}$  are decorrelated. (Only in the special case when  $\mathbf{y} = [y_1, \dots, y_n]$  is gaussian, are its components also independent.) While in ICA,  $\mathbf{W}$  is in general not orthogonal, and the components are independent and

necessarily also uncorrelated. We therefore see that the ICA is a more powerful method in the sense that it satisfies a stronger requirement for  $\mathbf{W}$ .

A typical example of the problems that can be addressed by the ICA is the *blind source separation (BSS)* problem, i.e., to separate and reconstruct the linearly mixed source signals. The word “blind” means that no prior knowledge about sources or the mixing process is assumed available, except that the source signals are statistically independent. Although this BSS problem seems severely under constrained, the method of ICA can find nearly unique solutions satisfying certain properties. A typical example of the BSS problem is the “cocktail party problem”, where the goal is to reconstruct the individual voices of  $m$  speakers from  $n$  microphones ( $n \geq m$ ) recording all the conversations.

Here are some additional facts about the source signal  $\mathbf{s}$  and the observed data  $\mathbf{x} = \mathbf{A}\mathbf{s}$ :

- The number  $n$  for the observed variables in  $\mathbf{x}$  must be no fewer than the number  $m$  for the independent source signals in  $\mathbf{s}$ . We assume  $m = n$  in the following.
- The source components are stochastically independent and therefore uncorrelated, and they have to be non-Gaussian.
- The estimation of  $\mathbf{A}$  and  $\mathbf{s}$  is up to a scaling factor. Let

$$\mathbf{C} = \text{diag}(c_1, \dots, c_n), \quad \mathbf{C}^{-1} = \text{diag}(1/c_1, \dots, 1/c_n) \quad (11.3)$$

and  $\mathbf{A}' = \mathbf{A}\mathbf{C}^{-1}$  and  $\mathbf{s}' = \mathbf{C}\mathbf{s}$ , we have

$$\mathbf{x} = \mathbf{A}\mathbf{s} = [\mathbf{A}\mathbf{C}^{-1}][\mathbf{C}\mathbf{s}] = \mathbf{A}'\mathbf{s}' \quad (11.4)$$

- Due to the arbitrary scaling factor  $C$ , we can assume all the independent components have unit variance  $\sigma_i^2 = E[s_i^2] = 1$ . As they are independent and necessarily also uncorrelated, we have  $\sigma_{ij}^2 = E[s_i s_j] = \delta_{ij}$ , i.e.,

$$E(\mathbf{s}\mathbf{s}^T) = \mathbf{I} \quad (11.5)$$

- The estimated independent components are not in any particular order. When the order of the corresponding elements in both  $\mathbf{s}$  and  $\mathbf{A}$  is rearranged,  $\mathbf{x} = \mathbf{A}\mathbf{s}$  still holds.

The theoretical foundation of ICA algorithms is the *central limit theorem*, stating that the distribution of the sum (average or linear combination) of  $N$  independent random variables approaches Gaussian as  $N \rightarrow \infty$ . For example, the face value of a dice has a uniform distribution from 1 to 6. But the distribution of the sum of a pair of dice is no longer uniform. It has a maximum probability at the mean of 7. As the number  $N$  of dice increases, the distribution of the sum of the face values will approach a Gaussian. More generally, let  $x_1, \dots, x_N$  be  $N$  random variables independently drawn from an arbitrary distribution with mean  $\mu$  and variance  $\sigma^2$ . Then the distribution of the mean  $x = \sum_{i=1}^N x_i/N$  approaches a Gaussian with mean  $\mu$  and variance  $\sigma^2/N$ .

To solve the BSS problem, we want to find a matrix  $\mathbf{W}$  so that  $\mathbf{y} = \mathbf{W}\mathbf{x} =$

$\mathbf{W}\mathbf{A}\mathbf{s}$  is as close to the independent components in the sources  $\mathbf{s}$  as possible. This can be seen as the reverse process of the central limit theorem above. Consider the  $j$ th component  $y_j = \mathbf{w}_j^T \mathbf{x}_j$  of  $\mathbf{y}$ , where  $\mathbf{w}_j^T$  is the  $j$ th row of  $\mathbf{W}$ . As a linear combination of all components of  $\mathbf{s}$ ,  $y_i$  is necessarily more Gaussian than any of the  $m$  source components  $\{s_1, \dots, s_m\}$ . Therefore for  $\mathbf{y}$  to be a good estimate of  $\mathbf{s}$ , we desire to find  $\mathbf{W}$  that maximizes the *non-Gaussianity* of  $\mathbf{y} = \mathbf{W}\mathbf{x}$  so that  $\mathbf{y}$  is least Gaussian and most independent. The idea that non-Gaussianity is independence is in the core of the ICA method. Obviously if all source variables are Gaussian, the ICA method will not work.

All ICA algorithms can be considered as an optimization process that finds the matrix  $\mathbf{W}$  that maximizes certain objective function (to be discussed below) that measures the non-Gaussianity of  $\mathbf{y} = \mathbf{W}\mathbf{x}$ , and thereby the independence of its components.

## 11.2 Preprocessing

Before considering specifically how to find the desired transform matrix  $\mathbf{W}$  to reconstruct  $\mathbf{y} = \mathbf{W}\mathbf{x}$  to approximate the independent source signal  $\mathbf{s}$ , we first consider some preprocessing process which is required for the FastICA algorithm discussed below.

The purpose of the preprocessing process is to convert the observed data  $\mathbf{x}$  so that its components are of zero mean, unit variance, and independent of each other, i.e., its covariance matrix should be  $E[\mathbf{x}\mathbf{x}^T] = E[\mathbf{s}\mathbf{s}^T] = \mathbf{I}$ , same as the source signal  $\mathbf{s}$ .

- **Centering**

Subtract the mean  $E[\mathbf{x}]$  from the observed variable  $\mathbf{x} = \mathbf{A}\mathbf{s}$  so it has zero mean. By doing so, the sources  $\mathbf{s}$  also become zero mean because  $E[\mathbf{x}] = \mathbf{A} E[\mathbf{s}] = 0$  and  $E[\mathbf{s}]$  can be estimated to be  $\mathbf{A}^{-1}E[\mathbf{x}]$ .

- **Whitening**

Transform the observed variable  $\mathbf{x}$  so that its components are uncorrelated and have unit variance. Do do so, we first find the eigenvalue matrix  $\mathbf{\Lambda}$  and eigenvector matrix  $\mathbf{\Phi}$  of the covariance matrix  $E[\mathbf{x}\mathbf{x}^T]$  so that

$$\mathbf{\Phi}^T E[\mathbf{x}\mathbf{x}^T] \mathbf{\Phi} = \mathbf{\Lambda}, \quad \text{i.e.,} \quad \mathbf{\Lambda}^{-1/2} \mathbf{\Phi}^T E[\mathbf{x}\mathbf{x}^T] \mathbf{\Phi} \mathbf{\Lambda}^{-1/2} = \mathbf{I} \quad (11.6)$$

As  $E[\mathbf{x}\mathbf{x}^T]$  is symmetric,  $\mathbf{\Phi}^T = \mathbf{\Phi}^{-1}$  is orthogonal. We now carry out a linear transform of the observed  $\mathbf{x}$  so that its components are decorrelated (same as the KLT transform) and normalized with unity variance:

$$\mathbf{x}' = (\mathbf{\Lambda}^{-1/2} \mathbf{\Phi}^T) \mathbf{x}, \quad \text{i.e.,} \quad \mathbf{x} = (\mathbf{\Lambda}^{-1/2} \mathbf{\Phi}^T)^{-1} \mathbf{x}' = (\mathbf{\Phi} \mathbf{\Lambda}^{1/2}) \mathbf{x}' \quad (11.7)$$

so that the covariance matrix of  $\mathbf{x}'$  becomes

$$\begin{aligned} E[\mathbf{x}' \mathbf{x}'^T] &= E[(\mathbf{\Lambda}^{-1/2} \mathbf{\Phi}^T \mathbf{x})(\mathbf{\Lambda}^{-1/2} \mathbf{\Phi}^T \mathbf{x})^T] = E[\mathbf{\Lambda}^{-1/2} \mathbf{\Phi}^T \mathbf{x} \mathbf{x}^T \mathbf{\Phi} \mathbf{\Lambda}^{-1/2}] \\ &= \mathbf{\Lambda}^{-1/2} \mathbf{\Phi}^T E[\mathbf{x}\mathbf{x}^T] \mathbf{\Phi} \mathbf{\Lambda}^{-1/2} = \mathbf{I} \end{aligned} \quad (11.8)$$

i.e.,  $\mathbf{x}'$  is whitened with covariance  $\sigma_{ij}^2 = \delta_{ij}$ .

Once  $\mathbf{x}$  is centered and whitened to become  $\mathbf{x}' = (\mathbf{\Lambda}^{-1/2}\mathbf{\Phi}^T)\mathbf{x}$ , the reconstruction becomes

$$\mathbf{y} = \mathbf{W}\mathbf{x} = \mathbf{W}(\mathbf{\Phi}\mathbf{\Lambda}^{1/2})\mathbf{x}' = \mathbf{W}'\mathbf{x}' \quad (11.9)$$

where  $\mathbf{W}' = \mathbf{W}\mathbf{\Phi}\mathbf{\Lambda}^{1/2}$  is the reconstruction matrix for whitened  $\mathbf{x}'$ . We also have

$$E[\mathbf{x}'\mathbf{x}'^T] = E[(\mathbf{A}\mathbf{s})(\mathbf{A}\mathbf{s})^T] = \mathbf{A}E[\mathbf{s}\mathbf{s}^T]\mathbf{A}^T = \mathbf{A}\mathbf{A}^T = \mathbf{I} \quad (11.10)$$

Also as  $\mathbf{y} = \mathbf{W}'\mathbf{x}'$  is to match  $\mathbf{s}$ , we assume they have the same covariance  $E[\mathbf{y}\mathbf{y}^T] = E[\mathbf{s}\mathbf{s}^T] = \mathbf{I}$  (Eq. (11.5)), and further get

$$\mathbf{I} = E[\mathbf{y}\mathbf{y}^T] = E[(\mathbf{W}'\mathbf{x}')(\mathbf{W}'\mathbf{x}')^T] = \mathbf{W}'E[\mathbf{x}'\mathbf{x}'^T]\mathbf{W}'^T = \mathbf{W}'\mathbf{W}'^T \quad (11.11)$$

after whitening,  $\mathbf{W}'$  becomes an orthogonal matrix.

We see that after whitening, both the mixing matrix  $\mathbf{A}$  and the reconstruction matrix  $\mathbf{W}'$  becomes orthogonal and can therefore be interpreted as rotation matrices.

For convenience, in the following discussion we will still denote the whitened data samples by  $\mathbf{x}$  and the reconstruction matrix by  $\mathbf{W}$ .

## 11.3 Non-Gaussianity Maximization

In ICP, we desire to find the optimal  $\mathbf{W}$  that maximizes the non-Gaussianity of the reconstructed  $\mathbf{y} = \mathbf{W}\mathbf{x}$ . In general, the non-Gaussianity of a random  $y$  with density function  $p(y)$  can be measured by the *negentropy*, defined as the difference between its entropy  $H(y)$  and the entropy of a Gaussian variable  $y_G$  with the same variance:

$$J(y) = H(y_G) - H(y) = C - H(y) \geq 0 \quad (11.12)$$

where  $C = H(y_G)$  is a constant independent of  $y$ . We note that  $J(y)$  is non-negative as Gaussian distribution has the maximum entropy among all distributions with the same variance (Section 3.2.1). We see that maximizing non-Gaussianity measured by the negentropy  $J(y)$  is equivalent to minimizing the entropy  $H(y)$ . This negentropy defined for a scalar variable  $y$  can be generalized to a random vector  $\mathbf{y} = [y_1, \dots, y_m]^T$ :

$$J(\mathbf{y}) = H(\mathbf{y}_G) - H(\mathbf{y}) = C - H(\mathbf{y}) \geq 0 \quad (11.13)$$

The non-Gaussianity of  $\mathbf{y} = \mathbf{W}\mathbf{x}$  can also be measured in terms of the mutual information (Eq. (3.186)) shared by its components  $\{y_1, \dots, y_m\}$ . We desire to find the optimal  $\mathbf{W}$  that maximizes the independence or minimizes the mutual information of the components of  $\mathbf{y} = \mathbf{W}\mathbf{x}$ :

$$I(\mathbf{y}) = \sum_{i=1}^n H(y_i) - H(\mathbf{y}) = \sum_{i=1}^n H(y_i) - \log |\det \mathbf{W}| - H(\mathbf{x}) \quad (11.14)$$

where  $H(\mathbf{y}) = H(\mathbf{x}) + \log |\det \mathbf{W}|$  is the entropy of  $\mathbf{y} = \mathbf{W}\mathbf{x}$  as a linear function of  $\mathbf{x}$  (Eq. (3.179)).

As after preprocessing  $\mathbf{W}$  is orthogonal with  $\det \mathbf{W} = \pm 1$  independent of  $\mathbf{W}$ , and so is  $H(\mathbf{x})$ , the mutual information can be written as

$$I(\mathbf{y}) = \sum_{i=1}^n H(y_i) + C \quad (11.15)$$

where  $C$  is a constant representing the last two terms in the equation.

We see that minimizing all entropies  $H(y_i) = H(\mathbf{w}_i^T \mathbf{x})$  is equivalent to minimizing both Gaussianity (i.e., maximizing the negentropy  $J(\mathbf{y})$ ), and the mutual information  $I(\mathbf{y})$ . If the mutual information of a set of variables is decreased indicating the variables are less dependent, then the negentropy is increased, and components  $y_i$  are less Gaussian.

## 11.4 The FastICA Algorithm

The negentropy can be generalized to  $J(\mathbf{y}) = H(\mathbf{y}_G) - H(\mathbf{y})$  for a random vector  $\mathbf{y} = [y_1, \dots, y_m]^T$ . In ICA, we desire to find matrix  $\mathbf{W}$  so that  $\mathbf{y} = \mathbf{W}\mathbf{x}$  has the maximum negentropy and therefore is most non-Gaussian. The exact  $J(\mathbf{y})$  is difficult to find without the specific distribution  $p(\mathbf{y})$ , but it can be approximated by

$$J(y) \approx \sum_i k_i (E[G_i(y)] - E[G_i(g)])^2 \quad (11.16)$$

where  $k_i$  are some positive constants,  $y$  is assumed to have zero mean and unit variance due to whitening, and  $g$  is a Gaussian variable also with zero mean and unit variance.  $G_i$  are some non-quadratic functions such as

$$G_1(u) = \frac{1}{a} \log \cosh(au), \quad G_2(u) = -\exp(-u^2/2) \quad (11.17)$$

where  $1 \leq a \leq 2$  is some suitable constant. Although this approximation may not be accurate, it is always greater than zero except when  $x$  is Gaussian. In particular, when only a single term is used in Eq. (11.16), we have

$$J(y) = (E[G(y)] - E[G(g)])^2 \quad (11.18)$$

Since the second term is a constant, we simply need to maximize  $E[G(y)]$ .

The algorithm FastICA is to find the transform matrix  $\mathbf{W}$  so that  $\mathbf{y} = \mathbf{W}\mathbf{x}$  maximizes the following as an objective function representing the negentropy in Eq. (11.18) with the constant second term removed:

$$\sum_{i=1}^m E[G(y_i)] = \sum_{i=1}^m E[G(\mathbf{w}_i^T \mathbf{x})] \quad (11.19)$$

We first consider computing  $\mathbf{w}_i$  for  $y_i$  as one of the  $m$  components (with the subscript  $i$  dropped for now). This is a constrained optimization problem of

finding the optimal vector  $\mathbf{w}$  that maximizes the above treated as an objective function but subject to the constraint that it needs to be normalized. To solve this problem, we first construct the Lagrangian

$$L(\mathbf{w}) = E[G(\mathbf{w}^T \mathbf{x})] - \frac{\mu}{2}(\mathbf{w}^T \mathbf{w} - 1) \quad (11.20)$$

and then set to zero the derivative of  $L(\mathbf{w})$  with respect to  $\mathbf{w}$ :

$$\mathbf{f}(\mathbf{w}) = \frac{d}{d\mathbf{w}}L(\mathbf{w}) = E[\mathbf{x} \frac{d}{d\mathbf{w}}(\mathbf{w}^T \mathbf{x})] - \lambda \mathbf{w} = E[\mathbf{x} g(\mathbf{w}^T \mathbf{x})] - \mu \mathbf{w} = 0 \quad (11.21)$$

where  $g(z) = dG(z)/dz$  is the derivative of function  $G(z)$ . This equation system can be solved for  $\mathbf{w}$  iteratively by the Newton-Raphson method:

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{J}_f^{-1}(\mathbf{w})\mathbf{f}(\mathbf{w}) \quad (11.22)$$

where  $\mathbf{J}(\mathbf{w})$  is the Jacobian matrix of  $\mathbf{f}(\mathbf{w})$

$$\mathbf{J}_f(\mathbf{w}) = \frac{d}{d\mathbf{w}}\mathbf{f}(\mathbf{w}) = E[\mathbf{x}\mathbf{x}^T g'(\mathbf{w}^T \mathbf{x})] - \lambda \mathbf{I} \quad (11.23)$$

where  $g'(z)$  is the derivative of  $g(z)$  or second derivative of  $G(z)$ . We further approximate the first term on the right to get

$$E[\mathbf{x}\mathbf{x}^T g'(\mathbf{w}^T \mathbf{x})] \approx E[\mathbf{x}\mathbf{x}^T] E[g'(\mathbf{w}^T \mathbf{x})] = E[g'(\mathbf{w}^T \mathbf{x})]\mathbf{I} \quad (11.24)$$

where  $E[\mathbf{x}\mathbf{x}^T] = \mathbf{I}$  as the dataset is assumed to be whitened, then the Jacobian becomes diagonal

$$J_F(\mathbf{w}) = (E[g'(\mathbf{w}^T \mathbf{x})] - \lambda) \mathbf{I} \quad (11.25)$$

and the Newton-Raphson iteration becomes:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \frac{1}{E[g'(\mathbf{w}_n^T \mathbf{x})] - \mu} (E[\mathbf{x} g(\mathbf{w}_n^T \mathbf{x})] - \mu \mathbf{w}_n) \quad (11.26)$$

Multiplying both sides by the scalar  $\lambda - E[g'(\mathbf{w}^T \mathbf{x})]$ , we get

$$\mathbf{w}_{n+1} = E[\mathbf{x}g(\mathbf{w}_n^T \mathbf{x})] - E[g'(\mathbf{w}_n^T \mathbf{x})]\mathbf{w}_n \quad (11.27)$$

where the scaled version of  $\mathbf{w}_{n+1}$  is still denoted by  $\mathbf{w}$  as it is to be normalized right away anyway.

This algorithm can then be repeated to get all row vectors in  $\mathbf{W}$  each for one of the  $m$  row vectors  $\mathbf{w}_i$ . However, if this process is carried out independently, the resulting vectors may not be orthogonal to each other. To address this issue, the Gram-Schmidt method in Section A.2 can be used. Specifically, during each iteration of the computation for  $\mathbf{w}_i$  for  $i > 1$ , before it is normalized, we also subtract its projection onto each of the previously found  $\mathbf{w}_1, \dots, \mathbf{w}_{i-1}$  (Eq. ??GramSchmidtIteration)), so that it is guaranteed to be orthogonal to all of them. All vectors such computed will guarantee that the final matrix  $\mathbf{W}$  is orthogonal satisfying  $\mathbf{W}^T = \mathbf{W}^{-1}$  as required.

Now we have the following FastICA algorithm that is carried out  $m$  times each for one of the vectors  $\mathbf{w}_i$ , ( $i = 1, \dots, m$ ):

1. Choose an initial random guess for  $\mathbf{w}_i$
2. Iterate:

$$\mathbf{w} = E[\mathbf{x}g(\mathbf{w}^T \mathbf{x})] - E[g'(\mathbf{w}^T \mathbf{x})]\mathbf{w} \quad (11.28)$$

where the expectation is approximated by the average of all  $N$  samples.

- 3.

$$\mathbf{w} \leftarrow \mathbf{w} - \sum_{j=1}^{i-1} (\mathbf{w}^T \mathbf{w}_j) \mathbf{w}_j \quad (11.29)$$

4. Normalize:

$$\mathbf{w} \leftarrow \mathbf{w} / \|\mathbf{w}\| \quad (11.30)$$

5. If not converged, go back to step 2.

The Matlab code segment below shows the essential part of the FastICA algorithm for ICA problem, where `Imax` is the maximum number of iterations, and `tol` is the tolerance, e.g.,  $10^{-6}$ .

```

syms u
f=-exp(-u^2/2);
f=log(cosh(u));
g=matlabFunction(diff(f));
dg=matlabFunction(diff(diff(f)));
[d N]=size(X);           % size of dataset
[X T]=whiten(X);        % whiten dataset
W=[];                    % initialize matrix W
one=ones(N,1);
for i=1:d                % for each vector w
    w=rand(d,1);         % random initialization
    for i=1:Imax
        wold=w;
        wx=w'*X;
        w=X*g(wx)'/N-dg(wx)*one*w/N;
        if i>1
            w=w-W*W'*w;
        end
        w=w/norm(w);
        if norm(w-wold)<tol
            break
        end
    end
end
W=[W w];
end
Y=W'*X;

```



```
W=W*T;
```

Here is the whitening function that takes the dataset  $\mathbf{X}$  as input and generates the whitened data and the whitening transform matrix as the output:

```
function [X T]=whitening(X)
    [d N]=size(X);
    X=X-repmat(mean(X,2),1,N);
    [U S V]=svd(X);
    T=sqrt(N)*diag(1./(diag(S(1:d,1:d))))*U';
    X=T*X;
end
```

## 11.5 Likelihood Maximization

The ICA problem can also be addressed by the method of maximum likelihood estimation. We first consider the following two ways to estimate the unknown mixing matrix  $\mathbf{A}$ .

- Matrix  $\mathbf{A}$  can be estimated so that  $p(\mathbf{x}|\mathbf{A})$  of  $\mathbf{x} = \mathbf{A}\mathbf{s}$  conditioned on  $\mathbf{A}$  matches the actual  $p(\mathbf{x})$  based on the observed data, i.e., the KL-divergence  $D_{KL}(p(\mathbf{x})||p(\mathbf{x}|\mathbf{A}))$  is minimized. Here  $p(\mathbf{x}|\mathbf{A})$  is related to  $p(\mathbf{s}) = p(s_1, \dots, s_m) = \prod_{i=1}^m p_i(s_i)$  of the independent source signals by (Eq. (3.76)):

$$p(\mathbf{x}|\mathbf{A}) = |\det \mathbf{A}^{-1}| p(\mathbf{s}) = |\det \mathbf{A}^{-1}| \prod_{i=1}^m p_i(s_i) \quad (11.31)$$

Now the KL-divergence to be minimized can be obtained as

$$\begin{aligned} D_{KL}(p(\mathbf{x})||p(\mathbf{x}|\mathbf{A})) &= \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{p(\mathbf{x}|\mathbf{A})} d\mathbf{x} \\ &= \int p(\mathbf{x}) \log p(\mathbf{x}) d\mathbf{x} - \int p(\mathbf{x}) \log \left( |\det \mathbf{A}^{-1}| \prod_{i=1}^m p_i(s_i) \right) d\mathbf{x} \\ &= H(\mathbf{x}) - \log |\det \mathbf{A}^{-1}| \int p(\mathbf{x}) d\mathbf{x} - \int p(\mathbf{x}) \sum_{i=1}^m \log p_i(s_i) d\mathbf{x} \\ &= H(\mathbf{x}) - \log |\det \mathbf{A}^{-1}| - E_x \left[ \sum_{i=1}^m \log p_i(s_i) \right] \end{aligned} \quad (11.32)$$

where  $s_i$  is the  $i$ th component of  $\mathbf{s} = \mathbf{A}^{-1}\mathbf{x}$  and the expectation is with respect to observed data  $\mathbf{x}$  and can be estimated as the average of the samples in the dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ . Here the first term  $H(\mathbf{x})$  independent of  $\mathbf{A}$  can be dropped.

- Matrix  $\mathbf{A}$  can also be estimated by maximizing its likelihood given the observed dataset  $\mathbf{X}$  containing  $N$  i.i.d. samples:

$$\begin{aligned} L(\mathbf{A}|\mathbf{X}) &= p(\mathbf{X}|\mathbf{A}) = \prod_{n=1}^N p(\mathbf{x}_n|\mathbf{A}) = \prod_{n=1}^N |\det \mathbf{A}^{-1}| \prod_{i=1}^m p_i(s_{in}) \\ &= |\det \mathbf{A}^{-1}|^N \prod_{n=1}^N \prod_{i=1}^m p_i(s_{in}) \end{aligned} \quad (11.33)$$

where  $p(\mathbf{x}_n|\mathbf{A})$  is given in Eq. (11.31). The log-likelihood is:

$$\begin{aligned} l(\mathbf{A}|\mathbf{X}) &= \log L(\mathbf{A}|\mathbf{X}) = \log \left[ |\det \mathbf{A}^{-1}|^N \prod_{n=1}^N \prod_{i=1}^m p_i(s_{in}) \right] \\ &= N \left[ \log |\det \mathbf{A}^{-1}| + \frac{1}{N} \sum_{n=1}^N \sum_{i=1}^m p_i(s_{in}) \right] \\ &\propto \log |\det \mathbf{A}^{-1}| + E_x \left[ \sum_{i=1}^m \log p_i(s_i) \right] \end{aligned} \quad (11.34)$$

Here in the last step the average over all  $N$  data samples in the second term is treated as the estimated expectation.

Comparing these two cases, we note that maximizing the log-likelihood in Eq. (11.34) is equivalent to minimizing the KL-divergence in Eq. (11.32), as the two expressions are negation of each other.

To find the optimal  $\mathbf{W}$  so that the reconstructed signal  $\mathbf{y}$  ideally matches the source signal  $\mathbf{s}$ :

$$\mathbf{y} = \mathbf{W}\mathbf{x} = \mathbf{W}\mathbf{A}\mathbf{s} = \mathbf{s} \quad (11.35)$$

We can simply let  $\mathbf{W} = \mathbf{A}^{-1}$ , i.e.,  $y_i = \mathbf{w}_i^T \mathbf{x} = s_i$  for all  $i = 1, \dots, m$ . Then either Eq. (11.32) or (11.34) can be used as the objective function for the optimal  $\mathbf{W}$ , when  $\mathbf{A}^{-1}$  is replaced by  $\mathbf{W}$  and  $s_i$  is replaced by  $\mathbf{w}_i^T \mathbf{x}$ :

$$J(\mathbf{W}) = \log |\det \mathbf{W}| + E \left[ \sum_{i=1}^m \log p_i(\mathbf{w}_i^T \mathbf{x}) \right] \quad (11.36)$$

Its gradient with respect to  $\mathbf{W}$  is:

$$\mathbf{g}_J(\mathbf{W}) = \frac{d}{d\mathbf{W}} J(\mathbf{W}) = \frac{d}{d\mathbf{W}} \log |\det \mathbf{W}| + E \left[ \frac{d}{d\mathbf{W}} \sum_{i=1}^m \log p_i(\mathbf{w}_i^T \mathbf{x}) \right] \quad (11.37)$$

The first term is

$$\frac{d}{d\mathbf{W}} \log |\det \mathbf{W}| = \mathbf{W}^{-T} \quad (11.38)$$

where  $\mathbf{W}^{-T} = (\mathbf{W}^T)^{-1} = (\mathbf{W}^{-1})^T$ . For each term inside the summation of the

second term, we find its derivative with respect to the component  $w_{ki}$  in the  $k$ th row and  $i$ th column of  $\mathbf{W}$ :

$$\frac{d}{dw_{ki}} \log p_i(\mathbf{w}_i^T \mathbf{x}) = \left( \frac{p'_i}{p_i} \right) x_k = \phi_i x_k \quad (11.39)$$

where function  $\phi(u)$  is defined as

$$\phi(u) = \frac{d}{du} \log p(u) = \frac{1}{p(u)} \frac{d}{du} p(u) \quad (11.40)$$

based on which the matrix form of the second term can be written as  $\phi \mathbf{x}^T$ , where  $\phi = [\phi_1, \dots, \phi_m]^T$ . Now we have

$$\mathbf{g}_J(\mathbf{W}) = \mathbf{W}^{-T} + E[\phi \mathbf{x}^T] \quad (11.41)$$

and the optimal  $\mathbf{W}$  that maximizes  $J(\mathbf{W})$  for the log-likelihood can be found iteratively based on the gradient ascend method:

$$\mathbf{W}_{n+1} = \mathbf{W}_n + \delta \mathbf{g}_J(\mathbf{W}_n) = \mathbf{W}_n + \delta [\mathbf{W}^{-T} + E[\phi \mathbf{x}^T]] \quad (11.42)$$

By post-multiplying  $\mathbf{W}^T \mathbf{W}$  the gradient  $\mathbf{g}_J(\mathbf{W})$  we get the *natural gradient* along a modified search direction in each step of the iteration:

$$\mathbf{W}_{n+1} = \mathbf{W}_n + \delta \mathbf{g}_J(\mathbf{W}_n) = \mathbf{W}_n + \delta [\mathbf{I} + E[\phi \mathbf{W} \mathbf{x}^T]] \mathbf{W} \quad (11.43)$$

which becomes more efficient and converges more quickly.

If the stochastic gradient method is used, the iterative update of  $\mathbf{W}$  is based on only one of the  $N$  samples at a time, instead of the average of all  $N$  of them, i.e., the expectation symbol  $E$  can be dropped.

Various forms of  $p(u)$  and  $\phi(u) = p'(u)/p(u)$  have been suggested including the following:

$$p(u) = \frac{1}{\cosh(u)}, \quad \phi(u) = \frac{1}{p(u)} \frac{d}{du} p(u) = -\tanh(u) \quad (11.44)$$

The Matlab code segment below shows the essential part of the MLE method for ICA problem, where `Imax` is the maximum number of iterations, `tol` is the tolerance, e.g.,  $10^{-6}$ . We note that whitening is not required for the algorithm and is therefore optional.

```
[d N]=size(X);      % size of the dataset
T=eye(d);           % whitening transform matrix
if whiten
    [X T]=whitening(X);
end
W=eye(d);           % initialize W
for i=1:Imax
    Y=W*X;
    Wnew=W+eta*(W-tanh(Y)*(Y'*W)/N);
```

```
        if mean(abs(Wnew(:)-W(:))) < tol
            break
        end
        W=Wnew;
    end
W=W*T;
```

## **Part IV**

---

### **Classification**



# 12 Statistic Classification

---

## 12.1 Discriminative vs. Generative Methods for Classification

As one of the most important tasks in *machine learning*, *pattern classification* is to classify some objects of interest, generically referred to as *patterns* and described by a set of  $d$  *features* or *attributes* that characterizes the patterns, to one of some  $K$  *classes* or *categories*. Each pattern is represented by a vector (or a point)  $\mathbf{x} = [x_1, \dots, x_d]^T$  in a  $d$ -dimensional *feature space*, where  $x_i$  ( $i = 1, \dots, d$ ) is a variable for the measurement of the  $i$ th feature. Symbolically, the  $K$  classes can be denoted  $\{C_1, \dots, C_K\}$ , and a pattern  $\mathbf{x}$  belonging to the  $k$ th class is denoted by  $\mathbf{x} \in C_k$ . Pattern classification can therefore be considered as the process by which the  $d$ -dimensional feature space is partitioned into  $K$  regions each corresponding to one of the  $K$  classes. The boundaries between these regions, called *decision boundaries*, are to be determined by the specific algorithm, called a classifier, used for the classification.

Pattern classification can be carried out as either a *supervised* or *unsupervised learning* process, depending on the availability of a *training set* containing patterns of known class identities. Specifically, the training set contains a set of  $N$  patterns in  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , labeled respectively by the corresponding component in  $\mathbf{y} = [y_1, \dots, y_N]^T$  representing the class identities of the corresponding patterns in some way. For example, we can use  $y_k \in \{1, \dots, K\}$  to indicate  $\mathbf{x}_k \in C_{y_k}$ . In the special case when  $K = 2$ , there are only two classes  $C_1 = C_+$  and  $C_0 = C_-$ , and the classifier becomes binary based on training pattern  $\mathbf{x}_j$  ( $j = 1, \dots, N$ ), each labeled by  $y_j = 1$  if  $\mathbf{x}_j \in C_+$  or  $y_j = -1$  if  $\mathbf{x}_j \in C_-$ .

We assume there are  $N_k$  training samples  $\{\mathbf{x}_1^{(k)}, \dots, \mathbf{x}_{N_k}^{(k)}\}$  all labeled to belong to  $C_k$ , ( $k = 1, \dots, K$ ), and in total  $N = \sum_{k=1}^K N_k$  samples in the training set. If the training set is a fair representation of all patterns of different classes in the entire dataset, then  $P_k = N_k/N$  can be treated as an estimate of the *a priori* probability that any randomly selected pattern  $\mathbf{x}$  happens to belong to class  $C_k$ , without any prior knowledge of the pattern.

Once a classifier is properly trained according to a specific algorithm based on the training set, the feature space is partitioned into regions corresponding to the different classes and any unlabeled pattern of unknown class as a vector  $\mathbf{x}$  in the feature space can be classified into one of the  $K$  classes.

Supervised classification can be considered as a process of establishing the corresponding relationship between the patterns  $\mathbf{x}_1, \dots, \mathbf{x}_N$  treated as the independent or input variables to the classifier, and the classes  $C_1, \dots, C_K$  the input patterns belong, treated as the dependent or output variables. Therefore regression and classification can be considered as the same supervised learning process: modeling the relationship between the data points in  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  and their corresponding labelings (or targets) in  $\{y_1, \dots, y_N\}$ . This process is regression when the labelings take continuous real values, but it is classification when they are discrete categorical representing different classes. Some methods in the previous chapter on regression analysis are actually used as classifiers, such as logistic and softmax regressions, and the method of Gaussian process can also be used for classification.

If the training data of labeled patterns are unavailable, various *unsupervised learning* methods can be used to assign each unlabeled patterns into one of the  $K$  different groups, called *clusters*, according to its position in the feature space, based on the overall spatial structure and distribution of the data set in the feature space. This process is called *clustering analysis* or simply *clustering*.

There exist a variety of methods for learning, including both regression and classification, based on different models assumed. One way to characterize these methods is to put them all in a probabilistic framework, in terms of the probabilities of the given dataset including data points  $\mathbf{X}$  and the corresponding labeling  $\mathbf{y}$ . Now a method can be categorized into either of the following two groups:

- *Discriminative model (conditional or backward model)*

A discriminative method establishes a model that maps a data point  $\mathbf{x}$  to a class labeling  $y$ . Such a model is either a pure or traditional discriminative model if it is deterministic and aims to fit the training set in some optimal way, or a conditional model if it is probabilistic in nature, such as the conditional probability  $p(y|\mathbf{x})$ . The model parameters in  $\boldsymbol{\theta}$  are obtained in some optimal way based on the training set. Then prediction can be made for any unlabeled  $\mathbf{x}$  in terms of the corresponding  $y$ . As a discriminative method aims at finding the decision boundaries between different classes based on the training set, only those data samples that are close to the boundaries play an important role, while all other samples farther away from the boundaries are mostly ignored.

Typical discriminative methods include:

- K-nearest neighbors algorithm
- Linear and logistic regressions
- Support vector machines
- Perceptrons and Neural networks
- Decision trees and random forests

- *Generative model (forward model)*

A generative method first assumes certain probabilistic model for the underlying structure of the observed data, such the joint probability  $p(\mathbf{x}, y|\boldsymbol{\theta})$



based on all data samples available. It then estimates the parameter  $\theta$  of the model based on the training dataset, and obtains the conditional probability  $p(y|\mathbf{x})$  (by Bayes' theorem) based on which a prediction can be made for any unlabeled  $\mathbf{x}$  to find the corresponding  $y$ . As in general the generative method is based on some probabilistic model of the data, it can be used for unsupervised learning as well as supervised.

Typical generative methods include:

- Naive Bayes classifiers
- Gaussian mixture classifiers
- Hidden Markov model

Here are some comparisons between the two approaches:

- The discriminative methods find the decision boundary in the feature space directly based on the data points in the training set, in general they
  - are simpler than the generative approach requiring problem-specific knowledge for building models of the data,
  - are effective in producing accurate result when the dataset is large
  - provide no insight or interpretation regarding the data and no uncertainty estimate
- The generative methods first establish a probabilistic model for the underlying structure of the data as an effort to explain how the data was generated, and then finds the decision boundary based the model. In general, they
  - allow the use of problem-specific knowledge for building the model,
  - can provide explanation and interpretation of the data,
  - may be less prone to overfitting than a discriminative method,
  - can provide uncertainty estimate
  - may not be as accurate as the discriminative methods if the model does not fit the dataset well.

## 12.2 K Nearest Neighbor and Minimum Distance Classifiers

Here we first consider a set of simple supervised classification algorithms that assign an unlabeled sample  $\mathbf{x}$  to one of the  $K$  known classes based on set  $N$  of *training samples*  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , where each sample  $\mathbf{x}_n$  is labeled by  $y_n = k \in \{1, \dots, K\}$ , indicating it belongs to class  $C_k$ .

- **k Nearest neighbors (k-NN) Classifier**

Given an unlabeled pattern  $\mathbf{x}$ , we first find its  $k$  *nearest neighbors* in the training dataset, and then assign  $\mathbf{x}$  to one of the  $K$  classes by a majority vote of the  $k$  neighbors based on their class labelings. The voting can be weighted so that closer neighbors are more heavily weighted than those that are farther away. In particular, when  $k = 1$ ,  $\mathbf{x}$  is assigned to the class of its closest neighbor.

While the k-NN method is simple and straight forward, its computational cost is high as classifying any unlabeled pattern  $\mathbf{x}$  requires computing distances to all data points in the training set.

- **Minimum Distance Classifier**

Given a set of training data points  $\{\mathbf{x}_1, \dots, \mathbf{x}_{N_k}\}$  all belonging to the  $k$ th class  $C_k$  ( $k = 1, \dots, K$ ), we can find their mean and covariance to represent the class:

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{n=1}^{N_k} \mathbf{x}_n, \quad \mathbf{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N_k} (\mathbf{x}_n - \mathbf{m})^T (\mathbf{x}_n - \mathbf{m}) \quad (12.1)$$

Any unlabeled data point  $\mathbf{x}$  can be classified to one of the  $K$  classes based on certain distance  $d(\mathbf{x}, C_k)$  between  $\mathbf{x}$  and each of class  $C_k$ :

$$\text{if } d(\mathbf{x}, C_k) = \min\{d(\mathbf{x}, C_i) \mid i = 1, \dots, C\} \text{ then } \mathbf{x} \in C_k \quad (12.2)$$

We could simply use the Euclidean distance  $d_E(\mathbf{x}, \mathbf{m}_k)$  between  $\mathbf{x}$  and  $\mathbf{m}_k$ . But such a classification may not be reliable as Euclidean distance does not take into consideration the covariance  $\mathbf{\Sigma}_k$  representing how the  $N_k$  samples are distributed in the feature space, as illustrated by the following example.

**Example 1:** As illustrated in the figure below (left plot), a point  $x = 1$  in 1-D space is to be classified into one of the two classes represented by their corresponding Gaussian pdfs:

$$C_1 \sim \mathcal{N}(5, 1.2^2), \quad C_2 \sim \mathcal{N}(-5, 3^2) \quad (12.3)$$

If only the two means  $m_1 = 5$  and  $m_2 = -5$  are considered, we have  $d_E(x, m_1) = 4 < d_E(x, m_2) = 6$ , i.e.,  $x = 1$  is closer to  $m_1$  than  $m_2$  and therefore should be classified to class  $C_1$ . However, as shown in the plot,  $x$  should be classified to class  $C_2$ , if the variances  $\sigma_1 = 1.2$  and  $\sigma_2 = 3$  are also taken into consideration.

We see the distance  $d(\mathbf{x}, C_k)$  should be positively related to  $d_E(x, m_k)$ , but inversely related to  $\sigma_k$ , i.e., we can define  $d(x, C_k) = (x - m_k)^2 / \sigma_k^2$ . Based on this distance, we find  $d(x_1, m_1) = 11.11 > d(x_1, m_2) = 4.0$ , i.e.,  $x = 1$  should be classified to  $C_2$ .

In a higher dimensional feature space, we can carry out classification based on the more generally defined *Mahalanobis distance* between a point  $\mathbf{x}$  and a distribution represented by  $\mathbf{m}_k$  and  $\mathbf{\Sigma}_k$ :

$$d_M(\mathbf{x}, C_k) = (\mathbf{x} - \mathbf{m}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x} - \mathbf{m}_k) \quad (12.4)$$

**Example 2:** As illustrated in the above figure (right plot), two samples  $x_1 = 1$  and  $x_2 = 3$  are to be classified into either of two classes:

$$C_1 \sim \mathcal{N}(0, 1.2^2), \quad C_2 \sim \mathcal{N}(0, 3^2) \quad (12.5)$$

As the two means  $m_1 = m_2 = 0$  are the same,  $|x_i - m_1|^2 = |x_i - m_2|^2$

for both samples  $x_1$  and  $x_2$ , they are both classified into  $C_2$  with a greater variance  $\sigma_2 > \sigma_1$  therefore smaller Mahalanobis distances:

$$\begin{cases} d_M(x_1, C_1) = \frac{(x_1 - m_1)^2}{\sigma_1^2} = \frac{1}{1.2^2} = 0.69, & d_M(x_1, C_2) = \frac{(x_1 - m_1)^2}{\sigma_2^2} = \frac{1}{9} = 0.11 \\ d_M(x_2, C_1) = \frac{(x_2 - m_1)^2}{\sigma_1^2} = \frac{3^2}{1.2^2} = 6.25, & d_M(x_2, C_2) = \frac{(x_2 - m_1)^2}{\sigma_2^2} = \frac{3^2}{9} = 1.00 \end{cases} \quad (12.6)$$

However, as shown in the plot,  $x_1 = 1$  should be classified to  $C_1$ . We therefore see that sometimes the Mahalanobis distance is not reliable for classification, and some better method need to be considered, as discussed later.

## 12.3 Naive Bayes Classification

The method of naive Bayes (NB) classification is a classical supervised classification algorithm, which is first trained by a training set of  $N$  samples  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  and their corresponding labelings  $\mathbf{y} = [y_1, \dots, y_N]^T$ , and then used to classify any unlabeled sample  $\mathbf{x}$  into class  $C_k$  with the maximum *posterior probability*. As indicated by the name, naive Bayes classification is based on Bayes' theorem:

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}, C_k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_k)P(C_k)}{p(\mathbf{x})} \quad (12.7)$$

where

- $P(C_k)$  is the *prior probability* that any data sample belongs to class  $C_k$  without observing its values, more briefly denoted by  $P_k$ , which can be estimated by

$$P_k = \frac{N_k}{N} = \frac{N_k}{\sum_{l=1}^K N_l}, \quad (k = 1, \dots, K) \quad \text{satisfying} \quad \sum_{k=1}^K P_k = 1 \quad (12.8)$$

where  $N_k$  is the number of data samples in the training set labeled to belong to class  $C_k$ . This estimation is based on the assumption that the training samples are evenly drawn from the entire population, and are therefore a fair representation of all  $K$  classes.

- $p(\mathbf{x}|C_k) = L(C_k|\mathbf{x})$  is the *likelihood* for any observed  $\mathbf{x}$  to belong to class  $C_k$ , which is the conditional probability of  $\mathbf{x}$  given that  $\mathbf{x} \in C_k$ , assumed to be a Gaussian in terms of the mean vector  $\mathbf{m}_k$  and covariance matrix  $\mathbf{\Sigma}_k$ :

$$p(\mathbf{x}|C_k) = N(\mathbf{x}, \mathbf{m}_k, \mathbf{\Sigma}_k) = \frac{1}{(2\pi)^{d/2} |\mathbf{\Sigma}_k|^{1/2}} \exp \left[ -\frac{1}{2} (\mathbf{x} - \mathbf{m}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x} - \mathbf{m}_k) \right] \quad (12.9)$$

This assumption is based on the fact that the Gaussian distribution has the maximum entropy (uncertainty) among all probability density functions

with the same covariance, i.e., it imposes the least amount of unsupported constraint on the model for the dataset.

- $p(\mathbf{x}, C_k)$  is the joint probability of  $\mathbf{x}$  and  $C_k$ :

$$p(\mathbf{x}, C_k) = p(\mathbf{x}|C_k)P(C_k) = P(C_k|\mathbf{x})p(\mathbf{x}) \quad (12.10)$$

- $p(\mathbf{x})$  is the distribution of any data sample in the dataset, independent of its classe identity, the weighted sum of all likelihood  $p(\mathbf{x}|C_k)$  for  $k = 1, \dots, K$ , i.e., the joint probability  $p(\mathbf{x}, C_k)$  marginalized over all  $K$  classes:

$$p(\mathbf{x}) = \sum_{k=1}^K p(\mathbf{x}, C_k) = \sum_{k=1}^K P_k p(\mathbf{x}|C_k) = \sum_{k=1}^K P_k \left[ \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{m}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \mathbf{m}_k)\right) \right] \quad (12.11)$$

- $P(C_k|\mathbf{x})$  is the posterior probability that a data sample  $\mathbf{x}$  belongs to class  $C_k$  based on the observed values in  $\mathbf{x} = [x_1, \dots, x_d]^T$ .

Based on Bayes' theorem discussed above, the naive Bayes method classifies an unlabeled data sample  $\mathbf{x}$  to class  $C_k$  with the maximum posterior probability (*maximum a posteriori (MAP)*):

$$\text{if } P(C_k|\mathbf{x}) = \max\{P(C_l|\mathbf{x}), (l = 1, \dots, K)\}, \quad \text{then } \mathbf{x} \in C_k \quad (12.12)$$

As  $p(\mathbf{x})$  is common to all  $K$  classes (independent of  $k$ ), it plays no role in the relative comparison among the  $K$  classes, and can therefore be dropped, i.e.,  $\mathbf{x}$  is classified to  $C_k$  with maximal  $p(\mathbf{x}|C_k) P_k$ .

The naive Bayes classifier is an optimal classifier in the sense that the classification error is minimum. To illustrate this, consider an arbitrary boundary between two classes  $C_1$  and  $C_2$  that partitions the 1-D feature space into two regions  $R_1$  and  $R_2$ , as shown below. The probability of a misclassification is the joint probability  $P(\mathbf{x} \in C_i \cap \mathbf{x} \in R_j)$  for a sample  $\mathbf{x} \in C_i$  but falling in  $R_j$ . The total probability of error due to misclassification can be expressed as:

$$\begin{aligned} P(\text{error}) &= P((\mathbf{x} \in R_2) \cap (\mathbf{x} \in C_1)) + P((\mathbf{x} \in R_1) \cap (\mathbf{x} \in C_2)) \\ &= P(\mathbf{x} \in R_2/C_1) P_1 + P(\mathbf{x} \in R_1/C_2) P_2 \\ &= P_1 \int_{R_2} p(\mathbf{x}/C_1) d\mathbf{x} + P_2 \int_{R_1} p(\mathbf{x}/C_2) d\mathbf{x} \end{aligned} \quad (12.13)$$

It is obvious that the Bayes classifier is indeed optimal, due to the fact that its boundary  $p(\mathbf{x}|C_i)P_i = p(\mathbf{x}|C_j)P_j$  between classes  $C_i$  and  $C_j$  (the bottom plot) guarantees the classification error (shaded area) to be minimum.

To find the likelihood function  $p(\mathbf{x}|C_k) = \mathcal{N}(\mathbf{x}, \mathbf{m}_k, \boldsymbol{\Sigma}_k)$ , we need to find  $\mathbf{m}_k$  and  $\boldsymbol{\Sigma}_k$  based on the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ , based on the  $N_k$  training samples in class  $C_k$  by the method of maximum likelihood estimation

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{i=1}^{N_k} \mathbf{x}_i, \quad \boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{i=1}^{N_k} (\mathbf{x}_i - \mathbf{m}_k)(\mathbf{x}_i - \mathbf{m}_k)^T, \quad (\mathbf{x}_i \in C_k) \quad (12.14)$$

Once both  $P_k = N_k/N$  and  $p(\mathbf{x}|C_k) = \mathcal{N}(\mathbf{x}, \mathbf{m}_k, \boldsymbol{\Sigma}_k)$  are available, the classifier is trained, and the posterior  $P(C_k|\mathbf{x})$  can be calculated for the classification in Eq. (12.12).

As the classification is based on the relative comparison of the posterior  $P(C_k|\mathbf{x})$ , any monotonic mapping of the posterior can be equivalently used to simplify the computation, such as the logarithmic function:

$$\begin{aligned} \log P(C_k|\mathbf{x}) &= \log [p(\mathbf{x}|C_k)P_k/p(\mathbf{x})] = \log p(\mathbf{x}|C_k) + \log P_k - \log p(\mathbf{x}) \\ &= -\frac{1}{2}(\mathbf{x} - \mathbf{m}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \mathbf{m}_k) - \frac{N}{2} \log(2\pi) - \frac{1}{2} \log |\boldsymbol{\Sigma}_k| + \log P_k - \log p(\mathbf{x}) \end{aligned} \quad (12.15)$$

Dropping the constant terms  $\log p(\mathbf{x})$  and  $N \log(2\pi)/2$  that are independent of the index  $k$  and therefore play no role in the comparison above, we get the *quadratic discriminant function*:

$$D_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mathbf{m}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \mathbf{m}_k) - \frac{1}{2} \log |\boldsymbol{\Sigma}_k| + \log P_k \quad (12.16)$$

Now the classification can be represented by

$$\text{if } D_l(\mathbf{x}) = \max\{D_k(\mathbf{x}), (k = 1, \dots, K)\}, \quad \text{then } \mathbf{x} \in C_l \quad (12.17)$$

Equivalently, we can also treat the negative of this discriminant function as a distance measurement:

$$d_k(\mathbf{x}) = (\mathbf{x} - \mathbf{m}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \mathbf{m}_k) + \log |\boldsymbol{\Sigma}_k| - 2 \log P_k \quad (12.18)$$

for minimum distance classification. We note that the first term is the Mahalanobis distance defined previously. However, the additional terms in the expression contribute to better classification performance.

Geometrically, the feature space is partitioned into regions corresponding to the  $K$  classes by the *decision boundaries* between every pair of classes  $C_i$  and  $C_j$ , represented by the equation  $D_i(\mathbf{x}) = D_j(\mathbf{x})$ , i.e.,

$$\begin{aligned} &-\frac{1}{2}(\mathbf{x} - \mathbf{m}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \mathbf{m}_i) - \frac{1}{2} \log |\boldsymbol{\Sigma}_i| + \log P_i \\ &= -\frac{1}{2}(\mathbf{x} - \mathbf{m}_j)^T \boldsymbol{\Sigma}_j^{-1}(\mathbf{x} - \mathbf{m}_j) - \frac{1}{2} \log |\boldsymbol{\Sigma}_j| + \log P_j \end{aligned} \quad (12.19)$$

which can be written as

$$\mathbf{x}^T \mathbf{W} \mathbf{x} + \mathbf{w}^T \mathbf{x} + w = 0 \quad (12.20)$$

where

$$\begin{aligned} \mathbf{W} &= -\frac{1}{2}(\boldsymbol{\Sigma}_i^{-1} - \boldsymbol{\Sigma}_j^{-1}) \\ \mathbf{w} &= \boldsymbol{\Sigma}_i^{-1} \mathbf{m}_i - \boldsymbol{\Sigma}_j^{-1} \mathbf{m}_j \\ w &= -\frac{1}{2}(\mathbf{m}_i^T \boldsymbol{\Sigma}_i^{-1} \mathbf{m}_i - \mathbf{m}_j^T \boldsymbol{\Sigma}_j^{-1} \mathbf{m}_j) - \frac{1}{2} \log \frac{|\boldsymbol{\Sigma}_i|}{|\boldsymbol{\Sigma}_j|} + \log \frac{P_i}{P_j} \end{aligned} \quad (12.21)$$

In general, this decision boundary is a quadratic hypersurface (hypersphere, hyperellipsoid, hyperparabola, or hyperhyperbola), by which an unlabeled data

sample  $\mathbf{x}$  is classified into either of the two classes  $C_i$  and  $C_j$  based on whether it is on the positive or negative side of the surface::

$$\text{if } \mathbf{x}^T \mathbf{W} \mathbf{x} + \mathbf{w}^T \mathbf{x} + w \begin{cases} > 0 \\ < 0 \end{cases}, \quad \text{then } \mathbf{x} \in \begin{cases} C_i \\ C_j \end{cases} \quad (12.22)$$

We further consider several special cases:

- All classes have the same prior:

$$P_i = P_j \quad (i, j = 1, \dots, K) \quad (12.23)$$

then the last term of  $D_i(\mathbf{x})$  is zero, and we have

$$D_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mathbf{m}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \mathbf{m}_i) - \frac{1}{2} \log |\boldsymbol{\Sigma}_i| \quad (12.24)$$

We can reconsider Example 2 in the previous section but now based on the negative of the discriminant function above treated as a distance  $d(\mathbf{x}, C_i) = (\mathbf{x} - \mathbf{m}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \mathbf{m}_i) + \log |\boldsymbol{\Sigma}_i|$ , and get

$$\begin{cases} d(x_1, C_1) = 0.69 + \log(1.2^2) = 1.06, & d_M(x_1, C_2) = 0.11 + \log(3^2) = 2.31 \\ d_M(x_2, C_1) = 6.25 + \log(1.2^2) = 6.61, & d_M(x_2, C_2) = 1 + \log(3^2) = 3.20 \end{cases} \quad (12.25)$$

Now we see that  $x_1$  is classified into class  $C_1$ , while  $x_2$  into  $C_2$ , as desired. The misclassification of  $x_1$  to  $C_2$  by the Mahalanobis distance, the first term, is corrected, due to the addition of the second term  $\log(\sigma^2)$ . The plot below shows the partitioning of the 1-D space for the two classes  $C_1$  and  $C_2$ :

- All classes have the same covariance matrix  $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$ , the discriminant function becomes:

$$D_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mathbf{m}_i)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{m}_i) + \log P_i \quad (12.26)$$

Moreover, if all  $P_i$  are the same and the second term is dropped,  $D_i(\mathbf{x})$  becomes the negative *Mahalanobis distance*.

The boundary equation  $D_i(\mathbf{x}) = D_j(\mathbf{x})$  between  $C_i$  and  $C_j$  becomes

$$-\frac{1}{2}(\mathbf{x} - \mathbf{m}_i)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{m}_i) + \log P_i = -\frac{1}{2}(\mathbf{x} - \mathbf{m}_j)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{m}_j) + \log P_j \quad (12.27)$$

As the quadratic terms  $\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x}$  on both sides of the equation are the same, it becomes a linear equation:

$$\mathbf{w}^T \mathbf{x} + w = 0 \quad (12.28)$$

where

$$\mathbf{w} = \boldsymbol{\Sigma}^{-1}(\mathbf{m}_i - \mathbf{m}_j) \quad (12.29)$$

and

$$w = -\frac{1}{2}(\mathbf{m}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{m}_i - \mathbf{m}_j^T \boldsymbol{\Sigma}^{-1} \mathbf{m}_j) + \log \frac{P_i}{P_j} \quad (12.30)$$



```

end

function yhat=NBtesting(X,M,S,P)           % naive Bayes testing
    [d N]=size(X);                         % dimensions of dataset
    K=length(P);                           % number of classes
    for k=1:K
        InvS(:, :,k)=inv(S(:, :,k));      % inverse of covariance matrix
        Det(k)=det(S(:, :,k));           % determinant of covariance matrix
    end
    for n=1:N                               % for each of N samples
        x=X(:,n);
        dmax=-inf;
        for k=1:K
            xm=x-M(:,k);
            d=-(xm'*InvS(:, :,k)*xm)/2;
            d=d-log(Det(k))/2+log(P(k)); % discriminant function
            if d>dmax
                yhat(n)=k;                 % assign nth sample to kth class
                dmax=d;
            end
        end
    end
end
end

```

The classification result in terms of the estimated class identity  $\hat{y}$  of the training samples compared with the ground truth class labeling  $\mathbf{y}$  can be represented in the form of a  $K \times K$  confusion matrix, of which the element in the  $i$ th row and  $j$ th column is the number of training samples labeled as a member of the  $i$ th class but actually classified into the  $j$ th class. For an ideal classifier that classifies all data samples correctly, this confusion matrix should be a diagonal matrix. But for a nonideal classifier, its error rate can be found as the percentage of the number of misclassified samples (sum of all off-diagonal elements) out of the total number of samples. The code below shows the function that generates such a confusion matrix based on the estimated class identity  $\hat{y}$  and the ground truth labeling  $\mathbf{y}$ :

```

function [Cm er]=ConfusionMatrix(yhat,ytrain)
    N=length(ytrain);                       % number of test samples
    K=length(unique(ytrain));               % number of classes
    Cm=zeros(K);                             % the confusion matrix
    for n=1:N
        i=ytrain(n);
        j=yhat(n);
        Cm(i,j)=Cm(i,j)+1;
    end
end

```



```

er=1-sum(diag(Cm))/N;           % error percentage
end

```

**Example 1:**

This example shows the classification of two cocentric classes with one surrounding the other. They are separated by an ellipse with 9 out of 200 samples misclassified, i.e., the error rate of  $9/200 = 4.5\%$ . The confusion matrix is shown below:

$$\begin{bmatrix} 92 & 8 \\ 1 & 99 \end{bmatrix} \quad (12.37)$$

indicating 8 of the 100 samples belonging to class 1 are misclassified to class 2, and 1 of the 100 samples belonging to class 2 is misclassified to class 1.

**Example 2:**

This example shows the classification of a 2-class exclusive-or (XOR) data set with significant overlap, in terms of the confusion matrix and the error rate of  $61/400 = 15.25\%$ . Note that the decision boundaries are a pair of hyperbolas. The confusion matrix is

$$\begin{bmatrix} 175 & 25 \\ 36 & 164 \end{bmatrix} \quad (12.38)$$

**Example 3:**

The first two panels in the figure below show three classes in the 2-D space, while the third one shows the partitioning of the space corresponding to the three classes. Note that the boundaries between the classes are all quadratic. The confusion matrix of the classification result is shown below, with the error rate  $49/600 = 8.17\%$ .

$$\begin{bmatrix} 196 & 3 & 1 \\ 0 & 191 & 9 \\ 3 & 33 & 164 \end{bmatrix} \quad (12.39)$$

**Example 4**

The figure below shows some sub-samples of ten sets digits from 0 to 9, each is hand-written 224 times by different students. Each hand-written digit is represented as an 16 by 16 image containing 256 pixels, treated as a 256 dimensional vector.

The dataset can be visualized by applying the KLT transform to map all data points in the 256-D space into a 3-D space spanned by the three eigenvectors corresponding to the three greatest eigenvalues of the covariance matrix of the dataset, as shown below:

The naive Bayes classifier is applied to the dataset of handwritten digits from 0 to 9, of which each data sample is a 256-dimensional vector composed of  $16 \times 16$  pixels of an image for the handwritten of a digit. The dataset contains 224 samples per class and  $N = 2240$  in total for all  $K = 10$  classes.

We note that the 256-dimensional covariance matrix  $\Sigma_k$  for each of the  $K = 10$  classes is estimated by the 224 samples in the class, i.e., the estimated covariance

matrix does not have a full rank and is not invertible. This issue can be resolved by applying KLT to the dataset to reduce its dimensionality from 256 to some value smaller than 224, such as 100.

The classifier is trained on half of the dataset of 1120 randomly selected samples and then tested on the remaining 1120 samples. The classification results are given below in terms of the confusion matrices of the training samples (left) and the testing samples (right). All 1120 training samples are correctly classified, while out of the remaining 1120 test samples, 198 are misclassified (17.7% error).

$$\begin{bmatrix} 107 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 108 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 105 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 118 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 110 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 120 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 128 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 104 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 109 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 111 \end{bmatrix}, \begin{bmatrix} 101 & 0 & 0 & 0 & 0 & 0 \\ 0 & 58 & 0 & 0 & 0 & 0 \\ 0 & 1 & 105 & 0 & 0 & 0 \\ 5 & 2 & 6 & 101 & 2 & 4 \\ 0 & 15 & 0 & 0 & 102 & 0 \\ 5 & 4 & 1 & 1 & 0 & 99 \\ 0 & 17 & 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 6 & 15 & 6 & 4 & 6 & 1 \\ 0 & 3 & 0 & 0 & 2 & 0 \end{bmatrix}, \quad (12.40)$$

## 12.4 AdaBoost

The *Adaptive boosting* (*AdaBoost*) is a supervised binary classification algorithm based on a training set  $\{(\mathbf{x}_i, y_i) \mid i = 1, \dots, N\}$ , where each sample  $\mathbf{x}_i$  is labeled by  $y_i \in \{-1, +1\}$ , indicating to which of the two classes  $C_-$  and  $C_+$  it belongs.

AdaBoost is an iterative algorithm. In the  $t$ -th iteration, each of the  $N$  training samples is classified into one of the two classes by a *weak classifier*  $h_t(\mathbf{x}_n) \in \{-1, +1\}$ , which can be considered as a hypothesis. The classifier is weak in the sense that its performance only needs to be better than chance, i.e., the error rate is less than 50%. If a sample  $\mathbf{x}_n$  is correctly classified,  $h_t(\mathbf{x}_n) = y_n = \pm 1$ , i.e.,  $y_n h_t(\mathbf{x}_n) = 1$ ; if it is misclassified,  $h_t(\mathbf{x}_n) = -y_n = \pm 1$ , i.e.,  $y_n h_t(\mathbf{x}_n) = -1$ . We need to find the best weak classifier that minimizes the weighted error rate defined as:

$$\varepsilon_t = \frac{\sum_{y_n h_t(\mathbf{x}_n) = -1} w_t(n)}{\sum_{n=1}^N w_t(n)} \quad (12.41)$$

where  $w_t(n)$  is the weight of the  $n$ th training samples at the  $t$ -th iteration. We can also get the correct rate:

$$1 - \varepsilon_t = 1 - \frac{\sum_{y_n h_t(\mathbf{x}_n) = -1} w_t(n)}{\sum_{n=1}^N w_t(n)} = \frac{\sum_{y_n h_t(\mathbf{x}_n) = 1} w_t(n)}{\sum_{n=1}^N w_t(n)} \quad (12.42)$$

At the initial stage with  $t = 0$ , all  $N$  samples are equally weighted by  $w_0(1) =$

$\dots = w_0(N) = 1$ , and the weighted error

$$\varepsilon_0 = \frac{1}{N} \sum_{y_n h_0(\mathbf{x}_n) = -1} w_0(n) = \frac{\text{number of misclassified samples}}{\text{total number of samples}} \quad (12.43)$$

is actually the probability for any sample  $\mathbf{x}$  in the training set to be misclassified by  $h_0$ . For the weak classifier the error rate defined above only needs to be lower than 50 percent, i.e.,  $\varepsilon_t < 1/2$ . When  $t > 0$ , this weight  $w_t(n)$  will be modified based on whether  $\mathbf{x}_n$  is classified correctly in the subsequent iterations, as we will see below.

At each iteration, a *strong or boosted classifier*  $H_t(\mathbf{x}_n)$  is also constructed based on the linear combination (boosting) of all previous weak classifiers  $h_1(\mathbf{x}_n), \dots, h_t(\mathbf{x}_n)$  for all  $n = 1, \dots, N$ :

$$\begin{aligned} F_t(\mathbf{x}_n) &= \alpha_t h_t(\mathbf{x}_n) + F_{t-1}(\mathbf{x}_n) = \alpha_t h_t(\mathbf{x}_n) + \alpha_{t-1} h_{t-1}(\mathbf{x}_n) + F_{t-2}(\mathbf{x}_n) \\ &= \dots = \sum_{i=1}^t \alpha_i h_i(\mathbf{x}_n) \end{aligned} \quad (12.44)$$

where the coefficient  $\alpha_i$  for the weak classifier  $h_i(\mathbf{x}_n)$  in the  $i$ -th iteration is obtained as discussed below. Taking the sign function of  $F_t(\mathbf{x})$  we get the strong classifier:

$$H_t(\mathbf{x}_n) = \text{sign}[F_t(\mathbf{x}_n)] = \begin{cases} +1 & F_t(\mathbf{x}_n) > 0 \\ -1 & F_t(\mathbf{x}_n) < 0 \end{cases} \quad (12.45)$$

which classifies a training sample  $\mathbf{x}_n$  into one of the two classes, while the magnitude  $|F_t(\mathbf{x}_n)|$  represents the confidence of the decision.

The weight  $w_t(n)$  of  $\mathbf{x}_n$  in the  $t$ -th iteration ( $t > 0$ ) is defined as

$$w_t(n) = e^{-y_n F_{t-1}(\mathbf{x}_n)} \begin{cases} > 1 & \text{if } y_n F_{t-1}(\mathbf{x}_n) < 0 \text{ (misclassification)} \\ < 1 & \text{if } y_n F_{t-1}(\mathbf{x}_n) > 0 \text{ (correct classification)} \end{cases} \quad (12.46)$$

The weight  $w_t(n)$  of the  $t$ -th iteration can be obtained recursively from  $w_{t-1}(n)$  of the previous iteration:

$$\begin{aligned} w_t(n) &= e^{-y_n F_{t-1}(\mathbf{x}_n)} = e^{-y_n [\alpha_{t-1} h_{t-1}(\mathbf{x}_n) + F_{t-2}(\mathbf{x}_n)]} \\ &= e^{-\alpha_{t-1} y_n h_{t-1}(\mathbf{x}_n)} e^{-y_n F_{t-2}(\mathbf{x}_n)} = e^{-\alpha_{t-1} y_n h_{t-1}(\mathbf{x}_n)} w_{t-1}(n) \end{aligned} \quad (12.47)$$

and this recursion can be carried out all the way back to the first iteration with  $w_1(k) = e^{-\alpha_0 y_n h_0(\mathbf{x}_n)} w_0(n)$ .

The performance of the strong classifier can be measured by the *exponential loss*, defined as the sum of all  $N$  weights:

$$E_{t+1} = \sum_{n=1}^N w_{t+1}(n) = \sum_{n=1}^N e^{-y_n F_t(\mathbf{x}_n)} = \sum_{n=1}^N e^{-\alpha_t y_n h_t(\mathbf{x}_n)} w_t(n) \quad (t > 1) \quad (12.48)$$

The coefficient  $\alpha_t$  in Eq. (12.44) can now be found as the one that minimizes the exponential loss  $E_{t+1}$ . To do so, we first separate the terms in the summation

for  $E_{t+1}$  into two parts, corresponding to the samples that are classified by  $h_t$  correctly and incorrectly:

$$E_{t+1} = \sum_{n=1}^N w_t(n) e^{-\alpha_t y_n h_t(\mathbf{x}_n)} = \sum_{y_n h_t(\mathbf{x}_n) = -1} w_t(n) e^{\alpha_t} + \sum_{y_n h_t(\mathbf{x}_n) = 1} w_t(n) e^{-\alpha_t} \quad (12.49)$$

and then set its derivative with respect to  $\alpha_t$  to zero:

$$\frac{dE_{t+1}}{d\alpha_t} = \sum_{y_n h_t(\mathbf{x}_n) = -1} w_t(n) e^{\alpha_t} - \sum_{y_n h_t(\mathbf{x}_n) = 1} w_t(n) e^{-\alpha_t} = 0 \quad (12.50)$$

Solving this equation we get  $\alpha_t$  as the optimal coefficient that minimizes  $E_{t+1}$ :

$$\alpha_t = \frac{1}{2} \ln \left( \frac{\sum_{y_n h_t(\mathbf{x}_n) = 1} w_t(n)}{\sum_{y_n h_t(\mathbf{x}_n) = -1} w_t(n)} \right) = \ln \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} > 0 \quad (12.51)$$

which is greater than zero because  $\varepsilon_t < 1/2$ ,  $1 - \varepsilon_t > 1/2$ , and

$$e^{\alpha_t} = \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} > 1, \quad e^{-\alpha_t} = \sqrt{\frac{\varepsilon_t}{1 - \varepsilon_t}} < 1 \quad (12.52)$$

We now see that if a weak classifier  $h_t$  has a small error  $\varepsilon_t$ , it will be weighted by a large  $\alpha_t$  and therefore contributes more to the strong classifier  $H_t$ ; on the other hand, if  $h_t$  has a large error  $\varepsilon_t$ , it will be weighted by a small  $\alpha_t$  and contributes less to  $H_t$ . As the strong classifier  $H_t$  takes advantage of all previous weak classifiers  $h_1, \dots, h_t$ , each of which may be more effective in a certain region of the N-D feature space than others,  $H_t$  can be expected to be a much more accurate classifier than any of the weak classifiers.

Replacing  $t$  by  $t + 1$  in Eq. (12.47), we now get

$$w_{t+1}(n) = w_t(n) e^{-\alpha_t y_n h_t(\mathbf{x}_n)} = \begin{cases} w_t(n) \sqrt{\frac{\varepsilon_t}{1 - \varepsilon_t}} < w_t(n) & \text{if } y_n h_t(\mathbf{x}_n) = 1 \\ w_t(n) \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} > w_t(n) & \text{if } y_n h_t(\mathbf{x}_n) = -1 \end{cases} \quad (12.53)$$

We see that if  $y_n h_t(\mathbf{x}_n) = 1$ , i.e.,  $\mathbf{x}_n$  is classified correctly by  $h_t$ , it will be weighted more lightly by  $w_{t+1}(n) < w_t(n)$  in the next iteration, but if  $y_n h_t(\mathbf{x}_n) = -1$ , i.e.,  $\mathbf{x}_n$  is classified incorrectly by  $h_t$ , it will be weighted more heavily by  $w_{t+1}(k) > w_t(k)$  in the next iteration, thereby it will be emphasized and have a better chance to be corrected to have more accurately classified by  $h_{t+1}$  in the next iteration.

We further consider the ratio of the exponential losses of two consecutive

iterations:

$$\begin{aligned}
\frac{E_{t+1}}{E_t} &= \frac{\sum_{n=1}^N w_{t+1}(n)}{\sum_{n=1}^N w_t(n)} = \frac{\sum_{n=1}^N w_t(n) e^{-\alpha_t y_n h_t(\mathbf{x}_n)}}{\sum_{n=1}^N w_t(n)} \\
&= \frac{\sum_{y_n h_t(\mathbf{x}_n) = -1} w_t(n)}{\sum_{n=1}^N w_t(n)} e^{\alpha_t} + \frac{\sum_{y_n h_t(\mathbf{x}_n) = 1} w_t(n)}{\sum_{n=1}^N w_t(n)} e^{-\alpha_t} \\
&= \varepsilon_t \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} + (1 - \varepsilon_t) \sqrt{\frac{\varepsilon_t}{1 - \varepsilon_t}} = 2\sqrt{\varepsilon_t(1 - \varepsilon_t)} \leq 1 \quad (12.54)
\end{aligned}$$

This ratio is twice the geometric average  $\sqrt{\varepsilon_t(1 - \varepsilon_t)} \leq 1$  of  $\varepsilon_t$  and  $1 - \varepsilon_t$ , which reaches its maximum when  $\varepsilon_t = 1 - \varepsilon_t = 1/2$ . However, as  $\varepsilon_t < 1/2$ , the ratio is always smaller than 1. We see that the exponential cost can be approximated as an exponentially decaying function from its initial value  $E_0 = \sum_{n=1}^N w_0(n) = N$  for some  $\varepsilon < 1/2$ :

$$\lim_{t \rightarrow \infty} E_t \approx \lim_{t \rightarrow \infty} \left(2\sqrt{\varepsilon(1 - \varepsilon)}\right)^t E_0 = 0 \quad (12.55)$$

We can therefore conclude that the exponential loss will always decrease, i.e., the error of AdaBoost will always converge to zero.

The weak classifier used in each iteration is typically implemented as a *decision stump*, a binary classifier that partitions the N-D feature space into two regions. Specifically, as a simple example, a coordinate descent method can be used by which all training samples are projected onto the  $i$ th dimension of the feature space  $\mathbf{d}_i$ :

$$x_n = P_{\mathbf{d}_i}(\mathbf{x}_n) = \frac{\mathbf{x}_n^T \mathbf{d}_i}{\|\mathbf{d}_i\|}, \quad (n = 1, \dots, N) \quad (12.56)$$

and then classified into two classes by a threshold  $T$  along the direction of  $\mathbf{d}_i$ :

$$h(x_n) = \begin{cases} +1 & \text{if } x_n < T \\ -1 & \text{if } x_n > T \end{cases} \quad (12.57)$$

The optimal decision stump is obtained when the following weighted error is minimized with respect to the threshold  $T$  along that direction of  $\mathbf{d}_i$ :

$$\varepsilon_t = \sum_{n=1}^N w_t(n) \delta(h_t(x_n) - y_n) = \sum_{h_t(x_n) \neq y_n} w_t(n) \quad (12.58)$$

Alternatively, the weak classifier can also be obtained based on the *principal component analysis (PCA)* by partitioning the N-D feature space along the directions of the eigenvectors of the between-class scatter matrix

$$\mathbf{S}_b = \frac{1}{N} [N_-(\mathbf{m}_{-1} - \mathbf{m})(\mathbf{m}_{-1} - \mathbf{m})^T + N_+(\mathbf{m}_{+1} - \mathbf{m})(\mathbf{m}_{+1} - \mathbf{m})^T] \quad (12.59)$$

where  $N_-$  and  $N_+$  are the numbers of samples in the two classes ( $N_- + N_+ = N$ ),

and

$$\mathbf{m}_{\pm} = \frac{1}{N_{\pm}} \sum_{y_n = \pm 1} w(n) \mathbf{x}_n, \quad \mathbf{m} = \frac{1}{N} \sum_{n=1}^N w(n) \mathbf{x}_n \quad (12.60)$$

are the weighted mean vectors of the two classes and the total weighted mean vector of all  $N$  samples in the training set. The training samples are likely to be better separated along these directions of the eigenvectors of  $\mathbf{S}_b$ , which measures the separability of the two classes. The eigenequations of  $\mathbf{S}_b$  is:

$$\mathbf{S}_b = \mathbf{\Phi} \mathbf{\Lambda} \mathbf{\Phi}^{-1} = \mathbf{\Phi} \mathbf{\Lambda} \mathbf{\Phi}^T \quad (12.61)$$

As  $\mathbf{S}_b$  is symmetric, the eigenvector matrix  $\mathbf{\Phi}$  is orthonormal and its columns can be used as an orthogonal basis that spans the feature space as well as the standard basis. The same binary threshold classification considered above for the weak classifiers can be readily applied along these PCA bases.

The Matlab code of the main iteration loop of the algorithm is listed below, followed by the essential functions called by the main loop. Here  $T$  is the maximum number of iteration, and  $N$  is the total number of training samples.

The algorithm can be carried out in the feature space based on either the PCA basis (columns of the eigenvector matrix  $\mathbf{\Phi}$ ), or the original standard basis (columns of the identity matrix  $\mathbf{I}$ ).

The *decision stump* is implemented by a binary classifier, which partitions all training samples projected onto a 1-D space into two groups by a threshold value, which needs to be optimal in terms of the number of misclassification. A parameter `plt` is used to indicate the polarity of the binary classification, i.e., which of the two class  $C_1$  and  $C_0$  is on the lower (or higher) side of the threshold.

```

Plt=[];           % polarity of binary classification
Tr=[];           % transformation of basis vectors
Th=[];           % threshold of binary classification
Alpha=[];        % alpha values
h=zeros(T,N);   % h functions
F=zeros(T,N);   % F functions
w=ones(T,N);    % weights for N training samples
Er=N;           % error initialized to N
t=0;            % iteration index
while t<T & Er>0           % the main iteration
    t=t+1;
    [tr th plt er]=WeakClassifier(X,y,w(t,:),PCA); % weak classifier
    alpha=log(sqrt((1-er)/er)); % update alpha
    Alpha=cat(1,Alpha,alpha); % record alpha
    Tr=cat(2,Tr,tr'); % record transform vector
    Th=cat(1,Th,th); % record threshold
    Plt=cat(1,Plt,plt); % record polarity
    x=tr*X; % carry out transform

```

```

c=sqrt(er/(1-er));
for n=1:N
    % update weights
    h(t,n)=h_function(x(n),th,plt); % find h function
    if h(t,n)*y(n)<0
        w(t+1,n)=w(t,n)/c; % update weights
    else
        w(t+1,n)=w(t,n)*c;
    end
end
F(t,:)=Alpha(t)*h(t,:); % get F functions
if t>1
    F(t,:)=F(t,:)+F(t-1,:);
end
Er=sum(sign(F(t,:).*y)==-1); % error of strong classifier, number of misclassification
fprintf('%d: %d/%d=%f\n',t,Er,N,Er/N)
end
end

```

Here are the functions called by the main iteration loop above:

```

function [Tr,Th Plt Er]=WeakClassifier(X,y,w,pca)
% X: N columns each for one of the N training samples
% y: labeling of X
% w: weights for N training samples
% pca: use PCA dimension if pca~=0
% Er: minimum error among all D dimensions
% Tr: transform vector (standard or PCA basis)
% Th: threshold value
% Plt: polarity
[D N]=size(X);
n0=sum(y>0); % number of samples in class C+
n1=sum(y<0); % number of samples in class C-
if pca % find PCA basis
    for i=1:D
        Y(i,:)=w.*X(i,:);
    end
    X0=Y(:,find(y>0)); % all samples in class C+
    X1=Y(:,find(y<0)); % all samples in class C-
    m0=mean(X0')'; % mean of C+
    m1=mean(X1')'; % mean of C-
    mu=(n0*m0+n1*m1)/N; % over all mean
    Sb=(n0*(m0-mu)*(m0-mu)'+n1*(m1-mu)*(m1-mu)')/N; % between-class scatter matrix
    [v d]=eig(Sb); % eigenvector and eigenvalue matrices of Sb
else
    v=eye(N); % standard basis
end
end

```

```

Er=9e9;
for i=1:D % for all D dimensions
    tr=v(:,i)'; % get transform vector from identity or PCA matrix
    x=tr*X; % rotate the vector
    [th plt er]=BinaryClassifier(x,y,w); % binary classify N samples in 1-D
    er=0;
    for n=1:N
        h(n)=h_function(x(n),th,plt); % h-function of nth sample
        if h(n)*y(n)<0 % if misclassified
            er=er+w(n); % add error
        end
    end
    er=er/sum(w); % total error of dimension d
    if Er>er % record info corresponding to min error
        Er=er; % min error
        Plt=plt; % polarity
        Th=th; % threshold
        Tr=tr; % transform vector
    end
end
end

function h=h_function(x, th, plt)
    if xor(x>th, plt)
        h=1;
    else
        h=-1;
    end
end

function [Th Plt Er]=BinaryClassifier(x,y,w)
    N=length(x);
    [x1 i]=sort(x); % sort 1-D data x
    y1=y(i); % reorder the targets
    w1=w(i); % reorder the weights
    Er=9e9;
    for n=1:N-1 % for N-1 ways of binary classification
        e0=sum(w1(find(y1(1:n)==1)))+sum(w1(n+find(y1(n+1:N)==-1)));
        e1=sum(w1(find(y1(1:n)~=1)))+sum(w1(n+find(y1(n+1:N)~-1)));
        if e1 > e0 % polarity: left -1, right +1
            plt=0; er=e0;
        else % polarity: left +1, right -1
            plt=1; er=e1;
        end
    end
end

```



```

        if Er > er                % update minimum error for kth dimension
            Er=er;                % minimum error
            Plt=plt;              % polarity
            Th=(x1(k)+x1(k+1))/2; % threshold
        end
    end
end
end
end

```

**Example 0** This example shows the classification of an XOR data set (used previously the test the naive Bayes method), containing two classes of  $N = 200$  samples each in 2-D space, represented by the red and blue dots in the figure below. The intermediate results after 4, 8, 16 and 32 iterations are shown in to illustrate the progress of the iteration, when the error rate continuously reduces from 134/400, 85/400, 81/400, 58/100.

After over 350 iterations the error rate eventually reduced to zero (guaranteed by the AdaBoost method). The sequence of binary participation Although all training samples are correctly classified eventually, the result suffers the problem of overfitting.

**Example 1** This example shows the classification of two classes of  $N = 500$  samples in 2-D space, which is partitioned along both the standard basis vectors (left) and the PCA directions (right). The PCA method performances significantly better in this example as its error reduces faster and it converges to zero after 61 iterations, while the error of the method based on the standard axes (in vertical and horizontal directions) does not go down to zero even after 120 iterations. This faster reduction of error of the PCA method can be easily understood as many different directions are used to partition the space into two regions, while in the coordinate descent method the partitioning of the space is limited to either of the two directions.

How the AdaBoost is iteratively trained can be seen in the figure below. First, the error rate, the ratio between the number of misclassified samples and the total number of samples, for both the training and testing samples, are plotted as the iteration progresses (top). Also, the weighted error  $\varepsilon_t$ , the exponential costs  $E_t$ , and the ratio  $E_{t+1}/E_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$  are all plotted (bottom). We see that  $\varepsilon_t < 0.5$  and  $E_{t+1}/E_t < 1$  for all steps, and the exponential cost  $E_t$  attenuates exponentially towards zero.

**Example 2** In this dataset, 100 training samples of two classes (50 each) forms four clusters arranged in the 2-D space as an XOR pattern as shown figure (top-left), and 100 testing samples of the same distribution are classified by the AdaBoost algorithm trained by both the coordinate descent and PCA methods.

We see that the coordinate descent method performs very poorly in both the slow convergence (more than 400 iterations) during training and high classification error rate (more than 1/3) during testing. The partitioning of the 2-D space is an obvious over fitting of the training set instead of reflecting the actual dis-

tribution of the two classes (four clusters). On the other hand, the PCA method converges quickly (10 iterations) and classifies the testing samples with very low error rate. The space is clearly partitioned into two regions corresponding to the two classes.

# 13 Support Vector machine

---

## 13.1 Maximal Margin and Support Vectors

The support vector machine (SVM) is a supervised binary classifier trained by a data set  $\{(\mathbf{x}_n, y_n), n = 1, \dots, N\}$  of which data sample  $\mathbf{x}_n$ , a vector in the  $d$ -dimensional feature space, belongs to either class  $C_+$  if labeled by  $y_n = 1$  or class  $C_-$  if labeled by  $y_n = -1$ . The result of the training process is the *decision boundaries* or *decision plane*  $H_0$  in the feature space, described by the linear *decision equation*:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^d x_i w_i + b = 0 \quad (13.1)$$

in terms of its normal vector  $\mathbf{w}$  and intercept  $b$ . Once the two parameters  $\mathbf{w}$  and  $b$  are determined based on the training set, any unlabeled sample  $\mathbf{x}$  can be classified into either of the two classes:

$$\text{if } f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b \begin{cases} > 0 \\ = 0 \\ < 0 \end{cases}, \text{ then } \begin{cases} \mathbf{x} \in C_+ \\ \mathbf{x} \in H_0 \\ \mathbf{x} \in C_- \end{cases} \quad (13.2)$$

The initial setup of the SVM algorithm seems the same as the method of linear regression as a linear binary classifier when the linear regression function  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$  is thresholded by zero to become  $f(\mathbf{x}) = 0$ . But here for the SVM, the parameters  $\mathbf{w}$  and  $b$  are determined in such a way that the corresponding decision plane separates the training samples belonging to the two classes optimally, so that its distance to the closest samples on either side, called *support vectors*, is maximized. Also, as kernel mapping can be applied to the SVM, it can also be used for nonlinear classification.

We rewrite the decision equation  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$  as

$$p_{\mathbf{w}}(\mathbf{x}) = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = -\frac{b}{\|\mathbf{w}\|} \quad (13.3)$$

and realize this is the projection of any point  $\mathbf{x} \in H_0$  on the decision plane  $H_0$  onto its normal direction  $\mathbf{w}$ , of which the absolute value is the distance between  $H_0$  and the origin:

$$d(H_0, \mathbf{0}) = |p_{\mathbf{w}}(\mathbf{x})| = \frac{|b|}{\|\mathbf{w}\|} \quad (13.4)$$

We further find the projection of any point  $\mathbf{x}' \notin H_0$  off the decision plane  $H_0$  onto  $\mathbf{w}$  as  $p_{\mathbf{w}}(\mathbf{x}') = \mathbf{w}^T \mathbf{x}' / \|\mathbf{w}\|$ , and its distance to  $H_0$  as:

$$d(H_0, \mathbf{x}') = \left| p_{\mathbf{w}}(\mathbf{x}') - p_{\mathbf{w}}(\mathbf{x}) \right| = \left| \frac{\mathbf{w}^T \mathbf{x}'}{\|\mathbf{w}\|} - \left( -\frac{b}{\|\mathbf{w}\|} \right) \right| = \frac{|\mathbf{w}^T \mathbf{x}' + b|}{\|\mathbf{w}\|} = \frac{|f(\mathbf{x}')|}{\|\mathbf{w}\|} \quad (13.5)$$

We desire to find the optimal decision plane  $H_0$  that separates the training samples belonging to the two different classes, assumed to be linearly separable, in such a way that its distance to the support vectors, denoted by  $\mathbf{x}_{sv}$ , is maximized:

$$d(H_0, \mathbf{x}_{sv} \in C_+) = d(H_0, \mathbf{x}_{sv} \in C_-) = d(H_0, \mathbf{x}_{sv}) = \frac{|f(\mathbf{x}_{sv})|}{\|\mathbf{w}\|} \quad (13.6)$$

We further consider two additional planes  $H_+$  and  $H_-$  that are in parallel with  $H_0$  and pass through the support vectors  $\mathbf{x}_{sv}$  on either side of  $H_0$ :

$$f_{\pm}(\mathbf{x}) = f(\mathbf{x}) \pm c = \mathbf{w}^T \mathbf{x} + b \pm c = \mathbf{w}^T \mathbf{x} + b \pm 1 = 0 \quad (13.7)$$

As these equations can be arbitrarily scaled, we can let  $c = 1$  for convenience. Based on Eq. (13.4), the distances from  $H_+$  or  $H_-$  to the origin can be written as

$$d(H_{\pm}, \mathbf{0}) = \frac{|b \pm c|}{\|\mathbf{w}\|} = \frac{|b \pm 1|}{\|\mathbf{w}\|} \quad (13.8)$$

and their distances to the decision plane  $H_0$ , called the decision *margin*, can be found as:

$$d(H_{\pm}, H_0) = \left| d(H_{\pm}, \mathbf{0}) - d(H_0, \mathbf{0}) \right| = \left| \frac{|b \pm c|}{\|\mathbf{w}\|} - \frac{|b|}{\|\mathbf{w}\|} \right| = \frac{|c|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \quad (13.9)$$

To maximize this margin, we need to minimize  $\|\mathbf{w}\|$ .

For these planes to correctly separate all samples in the training set  $\{(\mathbf{x}_n, y_n) \mid n = 1, \dots, N\}$ , they have to satisfy the following two conditions:

$$\begin{cases} f_-(\mathbf{x}_n) = \mathbf{w}^T \mathbf{x}_n + b - 1 \geq 0 & \text{if } y_n = 1 \\ f_+(\mathbf{x}_n) = \mathbf{w}^T \mathbf{x}_n + b + 1 \leq 0 & \text{if } y_n = -1 \end{cases} \quad (n = 1, \dots, N) \quad (13.10)$$

where the equality is satisfied by the support vectors on  $H_+$  or  $H_-$ , while the inequalities are satisfied by all other samples farther away from  $H_0$  behind  $H_+$  or  $H_-$ . The two conditions above can be combined to become:

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1, \quad (n = 1, \dots, N) \quad (13.11)$$

Now the task of finding the optimal decision plane  $H_0$  can be formulated as a constrained minimization problem:

$$\begin{aligned} \text{minimize:} & \quad \frac{1}{2} \|\mathbf{w}\|^2 = \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ \text{subject to:} & \quad y_n(\mathbf{x}_n^T \mathbf{w} + b) \geq 1, \quad (n = 1, \dots, N) \end{aligned} \quad (13.12)$$

**Example:**

The straight line in 2D space shown above, denoted by  $H_0$ , is described by the following linear equation

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = [w_1, w_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b = [1, 2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - 1 = x_1 + 2x_2 - 1 = 0 \quad (13.13)$$

The distance from  $H_0$  to the origin is:

$$d(H_0, \mathbf{0}) = \frac{|b|}{\|\mathbf{w}\|} = \frac{1}{\sqrt{w_1^2 + w_2^2}} = \frac{1}{\sqrt{1^2 + 2^2}} = \frac{1}{\sqrt{5}} = 0.447 \quad (13.14)$$

which is invariant with respect to any scaling of the equation. Consider three points in the space:

- $\mathbf{x}_0 = [0.5, 0.25]^T$ ,  $f(\mathbf{x}_0) = 0.5 + 2 \times 0.25 - 1 = 0$ , i.e.,  $\mathbf{x}_0$  is on the plane. Its distance to the plane is  $d(\mathbf{x}_0, H) = f(\mathbf{x}_0)/\|\mathbf{w}\| = 0$ .
- $\mathbf{x}_1 = [1, 0.25]^T$ ,  $f(\mathbf{x}_1) = 1 + 2 \times 0.25 - 1 = 0.5 > 0$ , i.e.,  $\mathbf{x}_1$  is above the straight line, its distance to the plane is  $d(\mathbf{x}_1, H) = f(\mathbf{x}_1)/\|\mathbf{w}\| = |0.5|/\sqrt{5} = 0.2235$ .
- $\mathbf{x}_2 = [0.5, 0]^T$ ,  $f(\mathbf{x}_2) = 0.5 + 2 \times 0 - 1 = -0.5 < 0$ , i.e.,  $\mathbf{x}_2$  is below the straight line, its distance to the plane is  $d(\mathbf{x}_2, H) = f(\mathbf{x}_2)/\|\mathbf{w}\| = |-0.5|/\sqrt{5} = 0.2235$ .

These two points  $\mathbf{x}_1 = [1, 0.25]^T$  and  $\mathbf{x}_2 = [0.5, 0]^T$  with equal distance to  $H_0$ , the straight line  $f(\mathbf{x}) = x_1 + 2x_2 - 1 = 0$ , can be assumed to be two support vectors  $\mathbf{x}_{sv}$  on either side of  $H_0$ , and

$$|f(\mathbf{x}_{sv})| = |\mathbf{w}^T \mathbf{x}_1 + b| = |\mathbf{w}^T \mathbf{x}_2 + b| = 0.5 \quad (13.15)$$

Multiplying 2 on both sides of the decision equation  $f(\mathbf{x}) = 0$ , it is scaled to become

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 2x_1 + 4x_2 - 2 = 0 \quad (13.16)$$

and the two equations describing  $H_+$  parallel to  $H_0$  and passing through  $\mathbf{x}_1$  and  $H_-$  passing through  $\mathbf{x}_2$  are

$$H_+ : 2x_1 + 4x_2 - 2 - 1 = 2x_1 + 4x_2 - 3 = 0$$

$$H_- : 2x_1 + 4x_2 - 2 + 1 = 2x_1 + 4x_2 - 1 = 0$$

Their distances to the origin are

$$d(H_+, \mathbf{0}) = \frac{|-3|}{\|\mathbf{w}\|} = 1.341, \quad d(H_-, \mathbf{0}) = \frac{|-1|}{\|\mathbf{w}\|} = 0.447 \quad (13.17)$$

and the distance between  $H_-$  and  $H_+$  is indeed  $2/\|\mathbf{w}\|$ :

$$d(H_-, H_+) = d(H_+, \mathbf{0}) - d(H_-, \mathbf{0}) = \frac{2}{\|\mathbf{w}\|} = 0.894 \quad (13.18)$$

twice the distance between  $H_0$  and the origin  $d(H_0, \mathbf{0}) = 0.447$  found previously.

For reasons to be discussed later, instead of directly solving the constrained

minimization problem in Eq. (13.12), now called the *primal problem*, we actually solve the *dual problem* (Section 2.2.3). Specifically, we first construct the Lagrangian function of the primal problem:

$$L_p(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{n=1}^N \alpha_n (1 - y_n (\mathbf{w}^T \mathbf{x}_n + b)) \quad (13.19)$$

where  $\alpha_1, \dots, \alpha_N$  are the *Lagrange multipliers*, which is called the *primal function*. Here for this minimization problem with non-positive constraints, the Lagrangian multipliers are required to be negative,  $\alpha_n$ , according to Table 2.167 in Section 2.2.2, if a minus sign is used for the second term. However, to be consistent with most SVM literatures, we use the positive sign for the second term and require  $\alpha_n \geq 0$ . Note that if  $\mathbf{x}_n$  is a support vector on either  $H_+$  or  $H_-$ , i.e., the equality constraint  $y_n (\mathbf{w}^T \mathbf{x}_n + b) = 1$  holds, then  $\alpha_n > 0$ ; but if  $\mathbf{x}_n$  is not a support vector, the equality constraint does not hold, and  $\alpha_n = 0$ .

We next find the minimum (or infimum) as the lower bound of the primal function  $L_p(\mathbf{w}, b)$  in Eq. (13.19), by setting to zero its partial derivatives with respect to both  $b$  and  $\mathbf{w}$ :

$$\frac{\partial}{\partial b} L_p(\mathbf{w}, b) = \frac{\partial}{\partial b} \left[ \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{n=1}^N \alpha_n (1 - y_n (\mathbf{w}^T \mathbf{x}_n + b)) \right] = \sum_{n=1}^N \alpha_n y_n = 0 \quad (13.20)$$

and

$$\frac{\partial}{\partial \mathbf{w}} L_p(\mathbf{w}, b) = \frac{\partial}{\partial \mathbf{w}} \left[ \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{n=1}^N \alpha_n (1 - y_n (\mathbf{w}^T \mathbf{x}_n + b)) \right] = \mathbf{w} - \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n = \mathbf{0}, \quad (13.21)$$

i.e.,

$$\mathbf{w} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n \quad (13.22)$$

Substituting these back into the primal function  $L_p(\mathbf{w}, b, \alpha)$ , we get its lower bound as a function of the Lagrange multipliers  $\{\alpha_1, \dots, \alpha_N\}$ , called the *dual function*:

$$\begin{aligned} L_d(\alpha) &= \inf_{\mathbf{w}, b} L_p(\mathbf{w}, b, \alpha) = \inf_{\mathbf{w}, b} \left[ \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{n=1}^N \alpha_n (1 - y_n (\mathbf{w}^T \mathbf{x}_n + b)) \right] \\ &= \inf_{\mathbf{w}, b} \left[ \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{n=1}^N \alpha_n - \mathbf{w}^T \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n - b \sum_{n=1}^N \alpha_n y_n \right] \\ &= \frac{1}{2} \left( \sum_{m=1}^N \alpha_m y_m \mathbf{x}_m \right)^T \left( \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n \right) + \sum_{n=1}^N \alpha_n - \left( \sum_{m=1}^N \alpha_m y_m \mathbf{x}_m \right)^T \left( \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n \right) \\ &= \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m (\mathbf{x}_n^T \mathbf{x}_m) \end{aligned} \quad (13.23)$$

To further find the greatest or tightest lower bound of the primal function, we need to maximize this dual function  $L_d(\alpha)$  with respect to the Lagrange multipliers, subject to the constraint imposed by Eq. (13.20):

$$\begin{aligned} \text{maximize: } L_d(\alpha) &= \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m (\mathbf{x}_n^T \mathbf{x}_m) = \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T \mathbf{Q} \alpha \\ \text{subject to: } \sum_{n=1}^N \alpha_n y_n &= \mathbf{y}^T \alpha = 0, \quad \alpha_n \geq 0 \quad (n = 1, \dots, N) \end{aligned} \quad (13.24)$$

where  $\mathbf{Q}$  is an  $N$  by  $N$  symmetric matrix of which the component in the  $m$ th row and  $n$ th column is  $Q(m, n) = y_m y_n \mathbf{x}_m^T \mathbf{x}_n$  ( $m, n = 1, \dots, N$ ). Now we have converted the primal problem of constrained minimization of the primal function  $L_p(\mathbf{w}, b, \alpha)$  with respect to  $\mathbf{w}$  and  $b$  to its dual problem of linearly constrained maximization of the dual function  $L_d(\alpha)$  with respect to  $\{\alpha_1, \dots, \alpha_N\}$ . Solving this quadratic programming (QP) problem (Section 2.2.6), we get all Lagrange multipliers  $\alpha_n \geq 0$ , ( $n = 1, \dots, N$ ). All training samples  $\mathbf{x}_n$  corresponding to positive Lagrange multipliers  $\alpha_n > 0$  are support vectors, while others corresponding to  $\alpha_n = 0$  are not support vectors.

We can now find the normal direction  $\mathbf{w}$  of the optimal decision plane based on Eq. (13.22):

$$\mathbf{w} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n = \sum_{n \in sv} \alpha_n y_n \mathbf{x}_n = \sum_{x_n \in C_+, n \in sv} \alpha_n \mathbf{x}_n - \sum_{x_n \in C_-, n \in sv} \alpha_n \mathbf{x}_n \quad (13.25)$$

Note that the normal vector  $\mathbf{w}$  is the difference between the weighted means of the support vectors in class  $C_+$  (first term) and class  $C_-$  (second term), and it is determined only by the support vectors.

Having found  $\{\alpha_1, \dots, \alpha_N\}$ , we can also find  $b$  based on the equality of Eq. (13.11) for any of the support vectors:

$$y_n (\mathbf{w}^T \mathbf{x}_n + b) = 1 \quad \text{or} \quad \mathbf{w}^T \mathbf{x}_n + b = y_n, \quad (n \in sv) \quad (13.26)$$

(recall  $y_n^2 = 1$ ). Solving the equation for  $b$  we get:

$$b = y_n - \mathbf{w}^T \mathbf{x}_n = y_n - \sum_{m \in sv} \alpha_m y_m (\mathbf{x}_m^T \mathbf{x}_n), \quad (n \in sv) \quad (13.27)$$

All support vectors should yield the same result. Computationally we simply get  $b$  as the average of the above for all support vectors.

We note that both  $\mathbf{w}$  and  $b$  depend only on the support vectors on planes  $H_+$  and  $H_-$ , corresponding to a positive  $\alpha_n > 0$ , while all remaining samples corresponding to  $\alpha_n = 0$  are farther away from the decision plane  $H_0$ , behind either  $H_+$  or  $H_-$ . We see that the SVM is solely based on those support vectors in the training set, once they are identified during the training process, all other non-support vectors are irrelevant.

Having obtained  $\mathbf{w}$  and  $b$  as shown above as the training process, any unlabeled

sample  $\mathbf{x}$  can be classified into either of the two classes depending on whether the following decision function is greater or smaller than zero:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \left( \sum_{n \in sv} \alpha_n y_n \mathbf{x}_n \right)^T \mathbf{x} + b = \sum_{n \in sv} \alpha_n y_n (\mathbf{x}_n^T \mathbf{x}) + b \quad \begin{cases} > 0 & \mathbf{x} \in C_+ \\ < 0 & \mathbf{x} \in C_- \end{cases} \quad (13.28)$$

We also get:

$$\begin{aligned} \|\mathbf{w}\|^2 &= \mathbf{w}^T \mathbf{w} = \left( \sum_{n \in sv} \alpha_n y_n \mathbf{x}_n^T \right) \left( \sum_{m \in sv} \alpha_m y_m \mathbf{x}_m \right) = \sum_{n \in sv} \alpha_n y_n \sum_{m \in sv} \alpha_m y_m (\mathbf{x}_n^T \mathbf{x}_m) \\ &= \sum_{n \in sv} \alpha_n y_n (y_n - b) = \sum_{n \in sv} \alpha_n (1 - y_n b) = \sum_{n \in sv} \alpha_n - b \sum_{n \in sv} \alpha_n y_n = \sum_{n \in sv} \alpha_n \end{aligned} \quad (13.29)$$

The last equality is due to equality constraints of the dual problem  $\sum_{n=1}^N \alpha_n y_n = 0$ . Now the decision margin can be written as:

$$\frac{1}{\|\mathbf{w}\|} = \left( \sum_{n \in sv} \alpha_n \right)^{-1/2} \quad (13.30)$$

The SVM algorithm for binary classification can now be summarized as the following steps:

- Find the Lagrange multipliers  $\{\alpha_1, \dots, \alpha_N\}$  by solving the QP problem in Eq. (13.24);
- Find the normal vector  $\mathbf{w}$  by Eq. (13.25);
- Find the bias  $b$  by Eq. (13.27);
- Classify unlabeled  $\mathbf{x}$  by Eq. (13.28).

As the last expression of the decision function in Eq. (13.28) depends only on  $\alpha_n$  as well as  $\mathbf{x}_n$  and  $y_n$  in the training set, the normal vector  $\mathbf{w}$  is never needed, and the second step above for calculating  $\mathbf{w}$  by Eq. (13.25) can be dropped.

We prefer to solve this dual problem not only because it is easier than the original primal problem, but also, more importantly, for the reason that all data points appear only in the form of an inner product  $\mathbf{x}_n^T \mathbf{x}_m$  in the dual problem, allowing the *kernel method* to be used, as discussed later.

The Matlab code for the essential part of the algorithm is listed below, where  $\mathbf{X}$  and  $\mathbf{y}$  are respectively the data array composed of  $N$  training vectors  $[\mathbf{x}_1, \dots, \mathbf{x}_N]$  and their corresponding labelings  $y_1, \dots, y_N$ , and `IPmethod` is a function that implements the *interior point method* (Section 2.2.7) for solving a general QP problem of minimizing  $\mathbf{x}^T \mathbf{Q} \mathbf{x} / 2 + \mathbf{c}^T \mathbf{x}$  subject to the linear equality constraints  $\mathbf{A} \mathbf{x} = \mathbf{b}$  and  $\mathbf{x} \geq \mathbf{0}$ .

```
[X y]=getData;           % get the training data
[m n]=size(X);
Q=(y'*y).*(X'*X);       % compute the quadratic matrix
c=-ones(n,1);           % coefficients of linear term
```



```

A=y; % coefficient matrix and
b=0; % constants of linear equality constraints
alpha=0.1*ones(n,1); % initial guess of solution

[alpha mu lambda]=IPmethod(Q,A,c,b,alpha); % solve QP to find alphas
% by interior point method

I=find(abs(alpha)>10^(-3)); % indecies of non-zero alphas
asv=alpha(I); % non-zero alphas
Xsv=X(:,I); % support vectors
ysv=y(:,I); % their labels
w=sum(repmat(ysv.*asv,m,1).*Xsv,2); % normal vector (not needed)
bias=mean(ysv-w'*Xsv); % bias
    
```

**Example:** The training set contains two classes of 2-D points with Gaussian distributions generated based on the following mean vectors and covariance matrices:

$$\mathbf{m}_- = \begin{bmatrix} 3.5 \\ 0 \end{bmatrix}, \quad \mathbf{S}_- = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}, \quad \mathbf{m}_+ = \begin{bmatrix} 0 \\ 3.5 \end{bmatrix}, \quad \mathbf{S}_+ = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{13.31}$$

The SVM algorithm finds the weight vector  $\mathbf{w}$  and constant  $b$  of the optimal boundary between the two classes based on three support vectors, all listed below. Note that decision boundary is completely dictated by the three support vectors and the margin distance between them is maximized.

$n$	$\alpha_n$	$\mathbf{x}_n = [x_1, x_2]$	$y_n$
40	0.52	$x = [4.43, 2.44]$	-1
52	3.11	$x = [1.17, 0.74]$	-1
103	3.64	$x = [1.30, 1.64]$	1

$$\mathbf{w} = \begin{bmatrix} -1.25 \\ 2.39 \end{bmatrix}, \quad b = -1.31 \tag{13.32}$$

## 13.2 Kernel Mapping

The SVM algorithm above converges only if the data points of the two classes in the training set are linearly separable. If this is not the case, we can consider the following *kernel method*.

In the kernel method, all data points as a vector  $\mathbf{x} = [x_1, \dots, x_d]^T$  in the original  $d$ -dimensional feature space are mapped by a *kernel function*  $\mathbf{z} = \phi(\mathbf{x})$  into a higher, possibly infinite, dimensional space

$$\mathbf{x} \implies \mathbf{z} = \phi(\mathbf{x}) \tag{13.33}$$

The presumption of the method is that the data points only appear in the form of inner product  $\mathbf{x}_m^T \mathbf{x}_n$  in the algorithm, and the kernel function  $\mathbf{z} = \phi(\mathbf{x})$  never needs to be actually carried out. In fact, the form of the kernel function  $\phi(\mathbf{x})$  and

the dimensionality of the higher dimensional space do not need to be explicitly specified or known.

The motivation for such a kernel mapping is that the relevant operations such as classification and clustering may be carried out much more effectively once the dataset is mapped to the higher dimensional space. For example, classes not linearly separable in the original  $d$ -dimensional feature space may be trivially separable in a higher dimensional space, as illustrated by the following examples.

**Example 1**

In 1-D space, classes  $C_- = \{x | (a \leq x \leq b)\}$  and  $C_+ = \{x | (x \leq a) \text{ or } (x \geq b)\}$  are not linearly separable. By the following mapping from 1-D space to 2-D space:

$$\mathbf{z} = \phi(x) = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} x \\ (x - (a + b)/2)^2 \end{bmatrix} \quad (13.34)$$

the two classes can be separated by a threshold in the second dimension of the 2-D space.

**Example 2:**

The method above can be generalized to higher dimensional spaces such as mapping from 2-D to 3-D space. Consider two classes in a 2-D space that are not linearly separable:  $C_- = \{\mathbf{x}, \|\mathbf{x}\| < D\}$  and  $C_+ = \{\mathbf{x}, \|\mathbf{x}\| > D\}$ . However, by the following mapping from 2-D space to 3-D space:

$$\mathbf{z} = \phi(\mathbf{x}) = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix} \quad (13.35)$$

the two classes can be trivially separated linearly by thresholding in the third dimension of the 3-D space.

**Example 3:**

In 2-D space, in the exclusive OR data set, the two classes of  $C_-$  containing points in quadrants I and III, and  $C_+$  containing points in quadrants II and IV are not linearly separable. However, by mapping the data points to a 3-D space:

$$\mathbf{z} = \phi(\mathbf{x}) = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix} \quad (13.36)$$

the two classes can be separated by simply thresholding in the third dimension of the 3-D space.

**Definition:** A kernel is a function that takes two vectors  $\mathbf{x}_m$  and  $\mathbf{x}_n$  as arguments and returns the inner product of their images  $\mathbf{z}_m = \phi(\mathbf{x}_m)$  and  $\mathbf{z}_n = \phi(\mathbf{x}_n)$ :

$$K(\mathbf{x}_m, \mathbf{x}_n) = \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n) = \mathbf{z}_m^T \mathbf{z}_n \quad (13.37)$$

The kernel function takes as input some two vectors  $\mathbf{x}_m$  and  $\mathbf{x}_n$  in the original feature space, and returns a scalar value as the inner product  $\mathbf{z}_m$  and  $\mathbf{z}_n$  in some higher dimensional space. If the data points in the original space only appear

in the form of inner product, then the kernel function  $\mathbf{z} = \phi(\mathbf{x})$ , called the kernel-induced *implicit* mapping, never needs to be explicitly specified, and the dimension of the new does not even need to be known.

The following is a set of commonly used kernel functions  $K(\mathbf{x}, \mathbf{x}')$ , which can be represented as an inner product of two vectors  $\mathbf{z} = \phi(\mathbf{x})$  and  $\mathbf{z}' = \phi(\mathbf{x}')$  in a higher dimensional space.

- **linear kernel (no kernel mapping)**

Assume  $\mathbf{x} = [x_1, \dots, x_d]^T$ ,  $\mathbf{x}' = [x'_1, \dots, x'_d]^T$ ,

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}' = \sum_{i=1}^d x_i x'_i \tag{13.38}$$

- **polynomial kernels**

The binomial theorem states:

$$(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k = \sum_{k=0}^n \frac{n!}{k!(n-k)!} x^{n-k} y^k \tag{13.39}$$

where the binomial coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{13.40}$$

is the number of ways to distribute  $n$  items into two bins ( $k$  in one and  $n - k$  in the other). This result can be generalized to the multinomial case:

$$(x_1 + \dots + x_d)^n = \sum_{\sum_{i=1}^d k_i = n} \binom{n}{k_1, \dots, k_d} x_1^{k_1} \dots x_d^{k_d} = \sum_{\sum_{i=1}^d k_i = n} \frac{n!}{k_1! \dots k_d!} x_1^{k_1} \dots x_d^{k_d} \tag{13.41}$$

where the multinomial coefficient

$$\binom{n}{k_1, \dots, k_d} = \frac{n!}{k_1! \dots k_d!} \tag{13.42}$$

is the number of ways to distribute  $n$  balls into  $d$  bins with  $k_i$  balls in the  $i$ th bin (<https://www.statlect.com/mathematical-tools/partitions>), and the summation is over all possible ways to get  $d$  non-negative integers  $k_1, \dots, k_d$  that add up to  $n$ .

Now consider the homogeneous polynomial kernel for  $d$ -dimensional vectors  $\mathbf{x} = [x_1, \dots, x_d]^T$  defined as

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^T \mathbf{x}')^n = (x_1 x'_1 + \dots + x_d x'_d)^n \\ &= \sum_{\sum_{i=1}^d k_i = n} \frac{n!}{k_1! \dots k_d!} ((x_1 x'_1)^{k_1} \dots (x_d x'_d)^{k_d}) = \phi(\mathbf{x})^T \phi(\mathbf{x}') \end{aligned} \tag{13.43}$$

where

$$\mathbf{z} = \phi(\mathbf{x}) = \left[ \sqrt{\frac{n!}{k_1! \cdots k_d!}} (x_1^{k_1} \cdots x_d^{k_d}), \left( k_i \geq 0, \sum_{i=1}^d k_i = n \right) \right]^T \quad (13.44)$$

In particular, when  $d = 2$  and  $n = 2$ , the polynomial kernel defined over 2-D vectors  $\mathbf{x} = [x_1, x_2]^T$  is:

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2 = (x_1 x'_1 + x_2 x'_2)^2 = (x_1 x'_1)^2 + 2x_1 x'_1 x_2 x'_2 + (x_2 x'_2)^2 = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \mathbf{z}^T \mathbf{z}' \quad (13.45)$$

where  $\mathbf{z} = \phi(\mathbf{x}) = [x_1^2, \sqrt{2}x_1 x_2, x_2^2]$  is a mapping from  $\mathbf{x}$  in 2-D space to  $\mathbf{z}$  in 3-D space.

A non-homogeneous polynomial kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^n \quad (13.46)$$

- **The radial basis function (RBF) kernel**

The RBF kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = e^{-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2} = e^{-\gamma \|\mathbf{x} - \mathbf{x}'\|^2} \quad (13.47)$$

where  $\gamma = 1/2\sigma^2$  is a parameter that can be adjusted to fit each specific dataset. This kernel can be written as the inner product of two infinite dimensional vectors (for simplicity, we assume  $\sigma = 1$ ):

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= e^{-\|\mathbf{x} - \mathbf{x}'\|^2 / 2} = e^{-\|\mathbf{x}\|^2 / 2} e^{-\|\mathbf{x}'\|^2 / 2} e^{\mathbf{x}^T \mathbf{x}'} = e^{-\|\mathbf{x}\|^2 / 2} e^{-\|\mathbf{x}'\|^2 / 2} \sum_{n=0}^{\infty} \frac{(\mathbf{x}^T \mathbf{x}')^n}{n!} \\ &= e^{-\|\mathbf{x}\|^2 / 2} e^{-\|\mathbf{x}'\|^2 / 2} \sum_{n=0}^{\infty} \left[ \frac{1}{n!} \sum_{\sum_{i=1}^d k_i = n} \frac{n!}{k_1! \cdots k_d!} ((x_1 x'_1)^{k_1} \cdots (x_d x'_d)^{k_d}) \right] \\ &= \sum_{n=0}^{\infty} \sum_{\sum_{i=1}^d k_i = n} \left( e^{-\|\mathbf{x}\|^2 / 2} \frac{x_1^{k_1} \cdots x_d^{k_d}}{\sqrt{k_1! \cdots k_d!}} \right) \left( e^{-\|\mathbf{x}'\|^2 / 2} \frac{x'_1{}^{k_1} \cdots x'_d{}^{k_d}}{\sqrt{k_1! \cdots k_d!}} \right) \\ &= \phi(\mathbf{x})^T \phi(\mathbf{x}') = \mathbf{z}^T \mathbf{z}' \end{aligned} \quad (13.48)$$

where

$$\mathbf{z} = \phi(\mathbf{x}) = \left[ e^{-\|\mathbf{x}\|^2 / 2} \frac{x_1^{k_1} \cdots x_d^{k_d}}{\sqrt{k_1! \cdots k_d!}}, \left( n = 0, \dots, \infty, \sum_{k=1}^d k_i = n \right) \right]^T \quad (13.49)$$

is a vector in an infinite dimensional space. In particular, when  $d = 1$  we have

$$\begin{aligned} K(x, x') &= e^{-(x-x')^2 / 2} = e^{-x^2 / 2} e^{-x'^2 / 2} e^{xx'} = e^{-x^2 / 2} e^{-x'^2 / 2} \sum_{n=0}^{\infty} \frac{(xx')^n}{n!} \\ &= \sum_{n=0}^{\infty} (e^{-x^2 / 2} x^n / \sqrt{n!}) (e^{-x'^2 / 2} x'^n / \sqrt{n!}) \end{aligned} \quad (13.50)$$

where  $\mathbf{z} = \phi(x) = \left[ e^{-x^2/2} x^n / \sqrt{n!}, (n = 0, \dots, \infty) \right]^T$  is a kernel function that maps a 1-D space into an infinite dimensional space.

The method of kernel mapping can be applied to the SVM algorithm consider previously as all data points appear in the algorithm are in the form of an inner product. Specifically, during the training process, we replace the inner product  $\mathbf{x}_m^T \mathbf{x}_n$  in both Eq. (13.24) and Eq. (13.27) by the kernel function  $K(\mathbf{x}_m, \mathbf{x}_n)$  to get

$$\begin{aligned} \text{maximize: } & L_d(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m K(\mathbf{x}_n, \mathbf{x}_m) \\ \text{subject to: } & \sum_{n=1}^N \alpha_n y_n = \mathbf{y}^T \boldsymbol{\alpha} = 0, \quad \alpha_n \geq 0 \quad (n = 1, \dots, N) \end{aligned} \quad (13.51)$$

and

$$b = y_n - \sum_{m \in sv} \alpha_m y_m K(\mathbf{x}_m, \mathbf{x}_n), \quad (n \in sv) \quad (13.52)$$

We also replace the inner product  $\mathbf{x}_n^T \mathbf{x}$  in Eq. (13.28) by  $K(\mathbf{x}_n, \mathbf{x})$  for the classification of any unlabeled point  $\mathbf{z} = \phi(\mathbf{x})$ :

$$f(\mathbf{z}) = \mathbf{w}^T \mathbf{z} + b = \sum_{n \in sv} \alpha_n y_n (\mathbf{z}_n^T \mathbf{z}) + b = \sum_{n \in sv} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + b \quad \begin{cases} > 0 & \mathbf{x} \in C_+ \\ < 0 & \mathbf{x} \in C_- \end{cases} \quad (13.53)$$

Again, we note that the normal vector  $\mathbf{w}$  in Eq. (13.25) never needs to be explicitly calculated. As now both the training and classification are carried out in some higher dimensional space  $\mathbf{z} = \phi(\mathbf{x})$ , in which the classes are more likely linearly separable, the classification can be more effectively. More generally, the kernel method can be applied to any algorithm, so long as the data always appear in the form of an inner product.

### 13.3 Soft Margin SVM

When the two classes  $C_-$  and  $C_+$  are not linearly separable, the condition for the optimal hyperplane in Eq. (13.11) can be relaxed by including an extra error term  $\xi_n \geq 0$ :

$$y_n (\mathbf{x}_n^T \mathbf{w} + b) \geq 1 - \xi_n, \quad (n = 1, \dots, N) \quad (13.54)$$

For better classification result, this error  $\xi_n$  needs to be minimized as well as  $\|\mathbf{w}\|$ . Now the primal problem in Eq. (13.12) can be modified to the following minimization problem with an objective function and two sets of inequality (non-

negative) constraints:

$$\begin{aligned} \text{minimize: } & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n \\ \text{subject to: } & y_n(\mathbf{x}_n^T \mathbf{w} + b) + \xi_n - 1 \geq 0, \quad \text{and } \xi_n \geq 0; \quad (n = 1, \dots, N) \end{aligned} \quad (13.55)$$

The value of the parameter  $C$  can be adjusted to make a proper trade-off between under fitting emphasizing the global distribution of the training samples, and over fitting emphasizing the local samples close to the decision boundary  $H_0$ . A large  $C$  value emphasizes the minimization of the error term in the objective function, and the resulting  $H_0$  is mostly dictated by a small number of local support vectors in the region between the class, including possibly some noisy outliers, thereby risking the problem of overfitting. On the other hand, a small  $C$  value de-emphasizes the error term and the classifier can tolerate greater errors, thereby allowing more data samples farther away from  $H_0$  to become support vectors and play a role in determining  $H_0$ . Consequently the classification trends to better reflect the overall distribution of the data samples, but with the risk of overfitting.

This constrained minimization problem can be converted to the minimization of the corresponding Lagrangian:

$$L_p(\mathbf{w}, b, \xi, \alpha, \mu) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n - \sum_{n=1}^N \alpha_n [y_n(\mathbf{w}^T \mathbf{x} + b) + \xi_n - 1] - \sum_{n=1}^N \beta_n \xi_n \quad (13.56)$$

where  $\beta_n$  as well as  $\alpha_n$  is the Lagrange multipliers for the two sets of constraints. As they are non-negative (instead of non-positive as in previous section), minus sign is used in front of the last two terms for the constraints, so that the Lagrange multipliers  $\alpha_n$  and  $\beta_n$  are still required to be positive (in consistent with Table 2.167). The KKT conditions (Eqs. (2.189) through (2.192)) of this minimization problem are listed below (for all  $n = 1, \dots, N$ ):

- Stationarity:

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^N y_n \alpha_n \mathbf{x}_n = 0, \quad \text{i.e.} \quad \mathbf{w} = \sum_{n=1}^N y_n \alpha_n \mathbf{x}_n; \quad (13.57)$$

$$\frac{\partial L_p}{\partial b} = \sum_{n=1}^N y_n \alpha_n = 0; \quad (13.58)$$

$$\frac{\partial L_p}{\partial \xi_n} = C - \alpha_n - \beta_n = 0; \quad (13.59)$$

- Primal feasibility:

$$y_n(\mathbf{w}^T \mathbf{x} + b) + \xi_n - 1 \geq 0, \quad \xi_n \geq 0, \quad (13.60)$$

- Complementarity:

$$\alpha_n [y_n(\mathbf{w}^T \mathbf{x} + b) + \xi_n - 1] = 0, \quad \beta_n \xi_n = 0, \quad (13.61)$$

- Dual feasibility:

$$\alpha_n \geq 0, \quad \beta_n \geq 0 \quad (13.62)$$

Depending on the value of  $\alpha_n$ , there exist three possible cases, due to the complementarity (Eq. (13.61)):

- If  $\alpha_n = 0$ , then  $\beta_n = C - \alpha_n = C > 0$ ,  $\xi_n = 0$ , and we have

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) - 1 \geq 0 \quad (13.63)$$

- If  $0 < \alpha_n < C$ , i.e.,  $\alpha_n \neq 0$ , then we have

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) + \xi_n - 1 = 0 \quad (13.64)$$

But as  $\beta_n = C - \alpha_n > 0$ ,  $\xi_n = 0$  and the equation above becomes

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) - 1 = 0 \quad (13.65)$$

- If  $\alpha_n = C \neq 0$ , then we have

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) + \xi_n - 1 = 0 \quad (13.66)$$

But as  $\beta_n = C - \alpha_n = 0$ ,  $\xi_n \geq 0$  and the equation above becomes

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) - 1 \leq 0 \quad (13.67)$$

In all three cases the expression  $y_n(\mathbf{w}^T \mathbf{x}_n + b) - 1$  can be rewritten as

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) - 1 = y_n(\mathbf{w}^T \mathbf{x}_n + b) - y_n^2 = y_n[(\mathbf{w}^T \mathbf{x}_n + b) - y_n] = y_n E_n \quad (13.68)$$

where  $E_n = (\mathbf{w}^T \mathbf{x}_n + b) - y_n$  is the error, or the difference between the actual and desired output. We summarize all three cases above to get:

$$y_n E_n \begin{cases} \geq 0 & \text{if } \alpha_n = 0 \\ = 0 & \text{if } 0 < \alpha_n < C \\ \leq 0 & \text{if } \alpha_n = C \end{cases} \quad (13.69)$$

These results derived from the KKT conditions in Eqs. (13.57) through (13.62) are an alternative form of the KKT conditions, which will be used in SMO algorithm to check if the KKT conditions are violated by any  $\alpha_n$ .

The dual of the primal problem in Eq. (13.56) can be obtained by substituting the first set of equations (stationarity) into the primal Lagrangian:

$$\begin{aligned} \text{maximize:} \quad L_d(\alpha) &= \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m \mathbf{x}_m^T \mathbf{x}_n \\ &+ C \sum_{n=1}^N \xi_n - \sum_{n=1}^N \alpha_n \xi_n - \sum_{n=1}^N \beta_n \xi_n \\ &= \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m \mathbf{x}_m^T \mathbf{x}_n \\ \text{subject to:} \quad &0 \leq \alpha_n \leq C, \quad \sum_{n=1}^N \alpha_n y_n = 0 \end{aligned} \quad (13.70)$$

Note that due to the condition  $C = \alpha_n + \beta_n$ , the Lagrangian multipliers  $\xi_n$  and  $\beta_n$  no longer appear in the Lagrangian  $L_d$  of the dual problem, which happens to be identical to that of the linearly separable problem discussed in previous section. Also, since  $\alpha_n \geq 0$ ,  $\beta_n \geq 0$  and  $C = \alpha_n + \beta_n$ , we have  $0 \leq \alpha_n = C - \beta_n \leq C$ . Having solved this QP problem for  $\alpha_1, \dots, \alpha_N$ , we get the optimal decision plane in terms of the normal vector of the decision plane

$$\mathbf{w} = \sum_{n=1}^N y_n \alpha_n \mathbf{x}_n = \sum_{n \in sv} y_n \alpha_n \mathbf{x}_n \quad (13.71)$$

based on Eq. (13.25), and the bias term  $b$  based on Eq. (13.27):

$$b = y_n - \sum_{i \in sv} y_i \alpha_i \mathbf{x}_i^T \mathbf{x}_n = y_n - \sum_{i \in sv} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}_n), \quad (n \in sv) \quad (13.72)$$

where the inner product  $\mathbf{x}_k^T \mathbf{x}$  is replaced by the kernel function  $K(\mathbf{x}_k, \mathbf{x})$  if it is nonlinear. Computationally, we take the average of such  $b$  values of all support vectors as the offset term.

Given  $\mathbf{w}$  and  $b$ , any unlabeled point  $\mathbf{x}$  can be classified into either of the two classes depending on whether it is on the positive or negative side of the decision plane:

$$\mathbf{w}^T \mathbf{x} + b = \sum_{i \in sv} y_i \alpha_i \mathbf{x}_i^T \mathbf{x} + b = \sum_{i \in sv} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b \begin{cases} > 0, & \mathbf{x} \in C_+ \\ < 0, & \mathbf{x} \in C_- \end{cases} \quad (13.73)$$

where the inner product is replaced by a kernel function if it is nonlinear.

## 13.4 Sequential Minimal Optimization Algorithm

The *sequential minimal optimization (SMO)* is a more efficient algorithm for solving the SVM problem, compared with the generic QP algorithms such as the internal-point method. The SMO algorithm can be considered as a method of decomposition, by which an optimization problem of multiple variables is decomposed into a series of subproblems each optimizing an objective function of a small number of variables, typically only one, while all other variables are treated as constants that remain unchanged in the subproblem. For example, the coordinate descent algorithm is just such a decomposition method, which solves a problem in a multi-dimensional space by converting it into a sequence of subproblems each in a one-dimensional space.

When applying the decomposition method to the soft margin SVM problem, we could consider optimizing only one variable  $\alpha_i$  at a time, while treating all remaining variables  $\alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_N$  as constants. However, due to the constraint  $\sum_{n=1}^N \alpha_n y_n = 0$ ,  $\alpha_i$  is a linear combination of  $N - 1$  constants and is therefore also a constant. We therefore need to consider two variables at a time, here assumed to be  $\alpha_i$  and  $\alpha_j$ , while treating the remaining  $N - 2$  variables as constants. The objective function of such a subproblem can be obtained by



dropping all constant terms independent of the two selected variables  $\alpha_i$  and  $\alpha_j$  in the original objective function in Eq. (13.70):

$$\begin{aligned}
\text{maximize: } \quad & L(\alpha_i, \alpha_j) = \alpha_i + \alpha_j - \frac{1}{2} (\alpha_i^2 \mathbf{x}_i^T \mathbf{x}_i + \alpha_j^2 \mathbf{x}_j^T \mathbf{x}_j + 2\alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j) \\
& - \alpha_i y_i \left( \sum_{n \neq i} \alpha_n y_n \mathbf{x}_n^T \right) \mathbf{x}_i - \alpha_j y_j \left( \sum_{n \neq j} \alpha_n y_n \mathbf{x}_n^T \right) \mathbf{x}_j \\
& = \alpha_i + \alpha_j - \frac{1}{2} (\alpha_i^2 K_{ii} + \alpha_j^2 K_{jj} + 2\alpha_i \alpha_j y_i y_j K_{ij}) \\
& - \alpha_i y_i \sum_{n \neq i, j} \alpha_n y_n K_{ni} - \alpha_j y_j \sum_{n \neq i, j} \alpha_n y_n K_{nj} \\
\text{subject to: } \quad & 0 \leq \alpha_i, \alpha_j \leq C, \quad \sum_{n=1}^N \alpha_n y_n = 0 \tag{13.74}
\end{aligned}$$

where, to use the kernel method, all inner products  $\mathbf{x}_m^T \mathbf{x}_n$  are replaced by the corresponding kernel function  $K_{mn} = K(\mathbf{x}_m, \mathbf{x}_n) = \mathbf{z}_m^T \mathbf{z}_n$  with  $\mathbf{z}_n = \phi(\mathbf{x}_n)$ .

This maximization problem can be solved iteratively. Out of all previous values  $\alpha_n^{old}$  ( $n = 1, \dots, N$ ), the two selected variables are updated to their corresponding new values  $\alpha_i^{new}$  and  $\alpha_j^{new}$  that maximize  $L(\alpha_i, \alpha_j)$ , subject to the two constraints.

We rewrite the second constraint as

$$\alpha_i y_i + \alpha_j y_j = - \sum_{n \neq i, j} \alpha_n y_n \tag{13.75}$$

and multiply both sides by  $y_i$  to get

$$y_i^2 \alpha_i + y_i y_j \alpha_j = \alpha_i + s \alpha_j = \left( - \sum_{n \neq i, j} \alpha_n y_n \right) y_i = \delta, \quad \text{i.e.} \quad \alpha_i = \delta - s \alpha_j \tag{13.76}$$

where we have defined  $s = y_i y_j$  and  $\delta = - \sum_{n \neq i, j} \alpha_n y_n y_i$ . As  $\delta$  is independent of  $\alpha_i$  and  $\alpha_j$ , it remains the same before and after they are updated, and we have

$$\alpha_i^{new} + s \alpha_j^{new} = \alpha_i^{old} + s \alpha_j^{old} = \delta \tag{13.77}$$

i.e.,

$$\Delta \alpha_i = \alpha_i^{new} - \alpha_i^{old} = -s (\alpha_j^{new} - \alpha_j^{old}) = -s \Delta \alpha_j \tag{13.78}$$

We now consider a closed-form solution for updating  $\alpha_i$  and  $\alpha_j$  in each iteration of this two-variable optimization problem. We first rewrite Eq. (13.25) as

$$\sum_{n \neq i, j} \alpha_n y_n \mathbf{x}_n = \mathbf{w} - \alpha_i y_i \mathbf{x}_i - \alpha_j y_j \mathbf{x}_j \tag{13.79}$$

so that the two summations in  $L(\alpha_i, \alpha_j)$ , now denoted by  $v_k$  ( $k = i, j$ ), can be

written as

$$\begin{aligned}
 v_k &= \sum_{n \neq i, j} \alpha_n y_n K_{nk} = \sum_{n=1}^N \alpha_n y_n K_{nk} - \alpha_i y_i K_{ik} - \alpha_j y_j K_{jk} \\
 &= \left( \sum_{n=1}^N \alpha_n y_n K_{nk} + b \right) - b - \alpha_i y_i K_{ik} - \alpha_j y_j K_{jk} \\
 &= u_k - b - \alpha_i y_i K_{ik} - \alpha_j y_j K_{jk}
 \end{aligned} \tag{13.80}$$

where

$$u_k = f(\mathbf{z}_k) = \sum_{n=1}^N \alpha_n y_n K_{nk} + b \tag{13.81}$$

is the output in Eq. (13.53) corresponding to input  $\mathbf{x}_k$ . Now the objective function above can be written as:

$$L(\alpha_i, \alpha_j) = \alpha_i + \alpha_j - \frac{1}{2} (\alpha_i^2 K_{ii} + \alpha_j^2 K_{jj} + 2s\alpha_i\alpha_j K_{ij}) - \alpha_i y_i v_i - \alpha_j y_j v_j \tag{13.82}$$

Substituting  $\alpha_i = \delta - s\alpha_j$  (Eq. (13.78)) into  $L(\alpha_i, \alpha_j)$ , we can rewrite it as a function of a single variable  $\alpha_j$  alone:

$$\begin{aligned}
 L(\alpha_j) &= \delta + (1-s)\alpha_j - \frac{1}{2} [(\delta - s\alpha_j)^2 K_{ii} + \alpha_j^2 K_{jj} + 2s(\delta - s\alpha_j)\alpha_j K_{ij}] \\
 &\quad - (\delta - s\alpha_j)y_i v_i - \alpha_j y_j v_j \\
 &= \frac{1}{2} (2K_{ij} - K_{ii} - K_{jj})\alpha_j^2 + [1-s + s\delta(K_{ii} - K_{ij}) + y_j(v_i - v_j)]\alpha_j + \text{Const.}
 \end{aligned} \tag{13.83}$$

where  $s^2 = (y_i y_j)^2 = 1$ , and scalar Const. represents all constant terms independent of  $\alpha_i$  and  $\alpha_j$  and can therefore be dropped. Now the objective function becomes a quadratic function of the single variable  $\alpha_j$ . Consider the coefficients of this quadratic function:

- The coefficient of  $\alpha_j^2$ :

$$\frac{1}{2} (2K_{ij} - K_{ii} - K_{jj}) = \frac{1}{2} \eta \tag{13.84}$$

where  $\eta$  is defined as

$$\begin{aligned}
 \eta &= 2K_{ij} - K_{ii} - K_{jj} = 2\mathbf{z}_i^T \mathbf{z}_j - \mathbf{z}_i^T \mathbf{z}_i - \mathbf{z}_j^T \mathbf{z}_j = -(\mathbf{z}_i - \mathbf{z}_j)^T (\mathbf{z}_i - \mathbf{z}_j) \\
 &= -\|\mathbf{z}_i - \mathbf{z}_j\|^2 \leq 0
 \end{aligned} \tag{13.85}$$

- The coefficient of  $\alpha_j$ :

$$\begin{aligned}
& 1 - s + s\delta(K_{ii} - K_{ij}) + y_j(v_i - v_j) \\
&= 1 - s + (s\alpha_i + \alpha_j)(K_{ii} - K_{ij}) \\
&\quad + y_j[(u_i - b - \alpha_i y_i K_{ii} - \alpha_j y_j K_{ij}) - (u_j - b - \alpha_i y_i K_{ij} - \alpha_j y_j K_{jj})] \\
&= 1 - s + s\alpha_1(K_{ii} - K_{ij}) + \alpha_2(K_{ii} - K_{ij}) \\
&\quad + y_j(u_i - u_j) - \alpha_i s K_{ii} - \alpha_j K_{ij} + \alpha_i s K_{ij} + \alpha_j K_{jj} \\
&= y_j^2 - s - \alpha_j(2K_{ij} - K_{ii} - 2K_{jj}) + y_j(u_i - u_j) \\
&= y_j(u_i - y_j - u_j + y_j) - \alpha_j(2K_{ij} - K_{ii} - K_{jj}) \\
&= y_j(E_i - E_j) - \alpha_j \eta
\end{aligned} \tag{13.86}$$

where we have defined

$$E_k = u_k - y_k = \sum_{n=1}^N \alpha_n^{old} y_n K_{nk} + b - y_k \quad (k = 1, 2) \tag{13.87}$$

as the difference between the desired output  $y_k$  and the actual output based on the previous values of the variables  $\alpha_n^{old}$  ( $n = 1, \dots, N$ ).

Now the quadratic objective function can be rewritten as:

$$L(\alpha_j) = \frac{1}{2} \eta \alpha_j^2 + [y_j(E_i - E_j) - \alpha_j^{old} \eta] \alpha_j \tag{13.88}$$

Given the previous values  $\alpha_n^{old}$  ( $n = 1, \dots, N$ ) in the coefficients, we will find  $\alpha_j^{new}$  that maximizes  $L(\alpha_j)$ . Consider its first and second order derivatives:

$$\begin{aligned}
\frac{d}{d\alpha_j} L(\alpha_j) &= \eta \alpha_j + y_j(E_i - E_j) - \alpha_j^{old} \eta \\
\frac{d^2}{d\alpha_j^2} L(\alpha_j) &= \eta \leq 0
\end{aligned} \tag{13.89}$$

As the second order derivative of  $L(\alpha_j)$  is  $\eta \leq 0$ , it has a maximum. Solving the equation  $dL(\alpha_j)/d\alpha_j = 0$ , we get  $\alpha_j^{new}$  that maximizes  $L(\alpha_j)$  based on the old value  $\alpha_j^{old}$ :

$$\alpha_j^{new} = \alpha_j^{old} + \frac{y_j(E_j - E_i)}{\eta} = \alpha_j^{old} + \Delta\alpha_j, \quad \Delta\alpha_j = \alpha_j^{new} - \alpha_j^{old} = \frac{y_j(E_j - E_i)}{\eta}, \tag{13.90}$$

Note that this is an unconstrained solution, which needs to be modified if the constraints in Eq. (13.74) are violated. As shown in the figure below,  $\alpha_i$  and  $\alpha_j$  are inside the square of size  $C$ , due to the first constraint  $0 \leq \alpha_i, \alpha_j \leq C$ ; also, due to the second constraint  $\alpha_i + s\alpha_j = \delta$  (Eq. (13.76)), they have to be on the diagonal line segment inside the square.

The four panels correspond to the four possible cases, depending on whether  $s = 1$  ( $y_i = y_j = \pm 1$ , panels 1 and 2), or  $s = -1$  ( $y_i = -y_j = \pm 1$ , panels 3 and 4), and whether  $\delta < C$  (panels 1 and 3) or  $\delta > C$  (panels 2 and 4), which can

be further summarized below in terms of the lower bound  $L$  and upper bound  $H$  of  $\alpha_2$ :

- If  $s = 1$ , then  $\alpha_i + s\alpha_j = \alpha_i + \alpha_j = \delta$ ,
  - if  $\delta < C$ , then  $\min(\alpha_j) = 0$ ,  $\max(\alpha_j) = \delta$  (first panel)
  - if  $\delta > C$ , then  $\min(\alpha_j) = \delta - C$ ,  $\max(\alpha_j) = C$  (second panel)
 Combining these two cases, we have

$$\begin{cases} L = \max(0, \delta - C) = \max(0, \alpha_i + \alpha_j - C) \\ H = \min(\delta, C) = \min(\alpha_i + \alpha_j, C) \end{cases} \quad (13.91)$$

- If  $s = -1$ , then  $\alpha_i + s\alpha_j = \alpha_i - \alpha_j = \delta$ ,
  - if  $\delta > 0$ , then  $\min(\alpha_j) = 0$ ,  $\max(\alpha_j) = C - \delta$  (third panel)
  - if  $\delta < 0$ , then  $\min(\alpha_j) = -\delta$ ,  $\max(\alpha_j) = C$  (fourth panel)
 Combining these two cases, we have

$$\begin{cases} L = \max(0, -\delta) = \max(0, \alpha_j - \alpha_i) \\ H = \min(C - \delta, C) = \min(C + \alpha_j - \alpha_i, C) \end{cases} \quad (13.92)$$

Now we apply the constraints in terms of  $L$  and  $H$  to the optimal solution  $\alpha_j^{new}$  obtained previously to get  $\alpha_j$  that is feasible as well as optimal:

$$\alpha_j^{new} \Leftarrow \begin{cases} H & \text{if } \alpha_j^{new} \geq H \\ \alpha_j^{new} & \text{if } L < \alpha_j^{new} < H \\ L & \text{if } \alpha_j^{new} \leq L \end{cases} \quad (13.93)$$

We can further find  $\alpha_i$  based on Eq. (13.78):

$$\alpha_i^{new} = \alpha_i^{old} - s\Delta\alpha_j = \alpha_i^{old} - s(\alpha_j^{new} - \alpha_j^{old}) \quad (13.94)$$

and update the weight vector based on Eq. (13.57):

$$\begin{aligned} \mathbf{w}^{new} &= \sum_{n \neq i, j} y_n \alpha_n^{old} \mathbf{x}_n + y_i \alpha_i^{new} \mathbf{x}_i + y_j \alpha_j^{new} \mathbf{x}_j \\ &= \mathbf{w}^{old} - y_i \alpha_i^{old} \mathbf{x}_i - y_j \alpha_j^{old} \mathbf{x}_j + y_i \alpha_i^{new} \mathbf{x}_i + y_j \alpha_j^{new} \mathbf{x}_j \\ &= \mathbf{w}^{old} + y_i \Delta\alpha_i \mathbf{x}_i + y_j \Delta\alpha_j \mathbf{x}_j = \mathbf{w}^{old} + \Delta\mathbf{w} \end{aligned} \quad (13.95)$$

where  $\Delta\mathbf{w} = \mathbf{w}^{new} - \mathbf{w}^{old} = y_i \Delta\alpha_i \mathbf{x}_i + y_j \Delta\alpha_j \mathbf{x}_j$ . To update  $b$ , consider:

$$\begin{aligned} \Delta E_k &= E_k^{new} - E_k^{old} = u_k^{new} - u_k^{old} \\ &= \sum_{n=1}^N \alpha_n^{new} y_n K_{nk} + b^{new} - \sum_{n=1}^N \alpha_n^{old} y_n K_{nk} - b^{old} \\ &= y_i \Delta\alpha_i K_{ik} + y_j \Delta\alpha_j K_{jk} + b^{new} - b^{old} \quad (k = 1, 2) \end{aligned} \quad (13.96)$$

Consider the following cases:

- If  $0 < \alpha_k < C$  for either  $k = i$  or  $k = j$ , then  $y_k E_k = 0$  according to Eq. (13.69). We let  $E_k^{new} = 0$  and solve the equation above to get:

$$b_k^{new} = b_k^{old} - (E_k^{old} + y_i \Delta\alpha_i K_{ik} + y_j \Delta\alpha_j K_{jk}) \quad (k = 1 \text{ or } 2) \quad (13.97)$$

- If  $0 < \alpha_k < C$  for both  $k = i$  and  $k = j$ , then  $b_i^{new} = b_j^{new}$ .
- If  $\alpha_k = 0$  or  $\alpha_k = C$  for both  $k = i$  and  $k = 2$ , then  $E_k \neq 0$ , and  $b_i^{new} \neq b_j^{new}$ , their average  $b^{new} = (b_i^{new} + b_j^{new})/2$  can be used.

The computation above for maximizing  $L(\alpha_i, \alpha_j)$  is carried out iteratively each time when a pair of variables  $\alpha_i$  and  $\alpha_j$  is selected to be updated. Specifically, we select a variable  $\alpha_i$  that violates any of the KKT conditions given in Eq. (13.69) in the outer loop of the SMO algorithm, and then pick a second variable  $\alpha_j$  in the inner loop of the algorithm, both to be optimized. The selection of these variables can be either random (e.g., by following their order), or guided by some heuristics, such as always choosing the variable with maximum step size  $\Delta\alpha_n = \alpha_n^{new} - \alpha_n^{old}$ . This iterative process is repeated until convergence when the KKT conditions are satisfied by all  $\alpha_1, \dots, \alpha_N$  variables, i.e., no more variables need to be updated. All data points corresponding to  $\alpha_n \neq 0$  are the support vectors, based on which we can find the offset  $b$  by Eq. (13.72) and classify any unlabeled point  $\mathbf{x}$  by Eq. (13.73).

The Matlab code for the essential part of the SMO algorithm is listed below, based mostly on a simplified SMO algorithm with some modifications.

In particular, the if-statement

```
if (alpha(i)<C & yE<-tol) | (alpha(i)>0 & yE>tol)
```

checks the three KKT conditions in Eq. (13.69), where  $yE = y_n E_n$  and  $tol$  is a small constant. The first half checks if the first two of the three conditions are violated, while the second half checks if the second two of the three conditions are violated.

```
[X y]=getData; % get training data
[m n]=size(X); % size of data
alpha=zeros(1,n); % alpha variables
bias=0; % initial bias
it=0; % iteration index
while (it<maxit) % number of iterations less than maximum
    it=it+1;
    changed_alphas=0; % number of changed alphas
    N=length(y); % number of support vectors
    for i=1:N % for each alpha_i
        Ei=sum(alpha.*y.*K(X,X(:,i)))+bias-y(i);
        yE=Ei*y(i);
        if (alpha(i)<C & yE<-tol) | (alpha(i)>0 & yE>tol) % KKT violation
            for j=[1:i-1,i+1:N] % for each alpha_j not equal alpha_i
                Ej=sum(alpha.*y.*K(X,X(:,j)))+bias-y(j);
                ai=alpha(i); % alpha_i old
                aj=alpha(j); % alpha_j old
                if y(i)==y(j) % s=y_i y_j=1
```

---

```

        L=max(0,alpha(i)+alpha(j)-C);
        H=min(C,alpha(i)+alpha(j));
    else
        % s=y_i y_j=-1
        L=max(0,alpha(j)-alpha(i));
        H=min(C,C+alpha(j)-alpha(i));
    end
    end
    if L==H
        % skip when L==H
        continue
    end
    eta=2*K(X(:,j),X(:,i))-K(X(:,i),X(:,i))-K(X(:,j),X(:,j));
    alpha(j)=alpha(j)+y(j)*(Ej-Ei)/eta; % update alpha_j
    if alpha(j) > H
        alpha(j) = H;
    elseif alpha(j) < L
        alpha(j) = L;
    end
    if norm(alpha(j)-aj) < tol
        % skip if no change
        continue
    end
    alpha(i)=alpha(i)-y(i)*y(j)*(alpha(j)-aj); % find alpha_i
    bi = bias - Ei - y(i)*(alpha(i)-ai)*K(X(:,i),X(:,i))...
        -y(j)*(alpha(j)-aj)*K(X(:,j),X(:,i));
    bj = bias - Ej - y(i)*(alpha(i)-ai)*K(X(:,i),X(:,j))...
        -y(j)*(alpha(j)-aj)*K(X(:,j),X(:,j));
    if 0<alpha(i) & alpha(i)<C
        bias=bi;
    elseif 0<alpha(j) & alpha(j)<C
        bias=bj;
    else
        bias=(bi+bj)/2;
    end
    end
    changed_alphas=changed_alphas+1; % one more alpha changed
end
end
end
if changed_alphas==0
    % no more changed alpha, quit
    break
end
I=find(alpha~=0);
alpha=alpha(I);
Xsv=X(:,I);
ysv=y(I);
end
nsv=length(ysv);

```

where  $K(\mathbf{X}, \mathbf{x})$  is a function that takes an  $m \times n$  matrix  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$  and an  $n$ -D vector  $\mathbf{x}$  and produces the kernel functions  $K(\mathbf{x}_i, \mathbf{x})$ , ( $i = 1, \dots, n$ ) as output.

```
function ker=K(X,x)
    global kfunction
    global gm % parameter for Gaussian kernel
    [m n]=size(X);
    ker=zeros(1,n);
    if kfunction=='l'
        for i=1:n
            ker(i)=X(:,i).'*x; % linear kernel
        end
    elseif kfunction=='g'
        for i=1:n
            ker(i)=exp(-gm*norm(X(:,i)-x)); % Gaussian kernel
        end
    elseif kfunction=='p'
        for i=1:n
            ker(i)=(X(:,i).'*x).^3; % polynomial kernel
        end
    end
end
```

Having found all non-zero  $\alpha_n \neq 0$  and the corresponding support vectors, we further find the offset or bias term:

```
bias=0;
for i=1:nsv
    bias=bias+(ysv(i)-sum(ysv.*alpha.*K(Xsv,Xsv(:,i))));
end
bias=bias/nsv;
```

Any unlabeled data point  $\mathbf{x}$  is classified into class  $C_+$  or  $C_-$ , depending on whether the following expression is greater or smaller than zero:

```
y=sum(alpha.*ysv.*K(Xsv,x))+bias;
```

**Example 1:** The same training set used previously to test the SVM with a hard margin is used again to test the SVM with a soft margin. Two results are shown below, corresponding to two different values  $C = 1.2$  and  $C = 0.05$  for the weight of the error term  $\sum_{n=1}^N \xi_n$  in the objective function. We see that when  $C = 1.2$  is large, the number of support vectors (the four solid dots) is small due to the small errors  $\xi_n$  allowed, and the decision boundary determined by these support vectors independent of the rest of the data set (circles) is mostly dictated by the local points close to the boundary, not necessarily a good

reflection of the global distribution of the data points. This result is similar to the previous one generated by the hard-margin SVM.

On the other hand, when  $C = 0.1$  is small, a greater number of data points become support vectors due to the larger (the 13 solid dots) error  $\xi_n$  allowed, and the resulting linear decision line determined by these support vectors reflects global distribution of the data points, and it better separates the two classes in terms of their mean positions.

**Example 2:** The classification results for the XOR data set are shown below with  $C = 1.2$  and  $C = 0.05$ . As the two classes in the XOR data set are not linearly separable, a Gaussian or radial basis function (RBF) kernel is used to map the data from 2-D space to an infinite dimensional space, which is partitioned into two regions for the two classes. When  $C = 1.2$  is large, the decision boundary is dictated by a relatively small number of support vectors (solid dots, with small error  $\xi_n$ ) and all data points are classified correctly, but it is more prone to the overfitting problem, if some of the small number of support vectors are outliers due to noise.

On the other hand, when  $C = 0.05$  is small, the decision boundary is determined by a greater number of support vectors, better reflecting the global distribution of the data points. However, as the local points close to the boundary are not relatively deemphasized, some miss-classification occurs, some nine red dots are classified into the blue region incorrectly.

**Example 3:**

**Example 4:**

## 13.5 Multi-Class Classification

The SVM method is inherently a binary classifier, but it can be adapted to classification problems of more than two classes. In the following, we consider a general  $K$ -class classifier based on a training set  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , of which each sample  $\mathbf{x}_n$  is labeled by  $y_n \in \{1, \dots, K\}$  to indicate  $\mathbf{x}_n \in C_{y_n}$ .

We first consider two straight forward and imperial methods for multiclass classification based directly on binary SVM.

- One-Vs-One (1V1)

Any unlabeled  $\mathbf{x}$  is classified to a class which receives the maximum votes out of the  $K(K-1)/2$  binary classifications between every pair of the  $K$  classes.

- One-Vs-Rest (1VR)

This method converts a  $K$ -class problem ( $K > 2$ ) into  $K$  binary problems. First, we regroup the  $K$  classes  $C_1, \dots, C_K$  into two classes  $C_+ = C_k$  and  $C_- = \{C_l | l = 1, \dots, K, l \neq k\}$  and find the corresponding decision function  $f_k(\mathbf{x})$  representing quantitatively to what extent a given  $\mathbf{x}$  belongs to class  $C_+ = C_k$  (if  $f_k(\mathbf{x}) > 0$ ), instead of  $C_-$  containing all remaining



$K - 1$  classes (if  $f_k(\mathbf{x}) < 0$ ). After this process has been carried out for all  $k = 1, \dots, K$ , the  $\mathbf{x}$  can be classified as below:

$$\text{if } f_l(\mathbf{x}) = \max\{f_1(\mathbf{x}), \dots, f_K(\mathbf{x})\}, \quad \text{then } \mathbf{x} \in C_l \quad (13.98)$$

We further consider another method for multiclass classification by directly generalizing the binary SVM so that the decision margin between any two of the  $K$  classes is maximized. First, we define a linear function for each of the  $K$  classes:

$$f_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k \quad (k = 1, \dots, K) \quad (13.99)$$

Once the parameters  $\mathbf{w}_k$  and  $b_k$  are determined during the training set, any unlabeled point  $\mathbf{x}$  can be classified as below:

$$\text{if } f_l(\mathbf{x}) = \max_k \{f_k(\mathbf{x}) \quad (k = 1, \dots, K)\}, \quad \text{then } \mathbf{x} \in C_l \quad (13.100)$$

The figure below illustrates the classification of  $K = 3$  classes in  $d = 2$  dimensional feature space. Here each straight lines  $H_k$  (plane or hyperplane if  $d \geq 3$ ) is determined by equation  $f_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k = 0$ , ( $k = 1, \dots, K$ ), with normal direction in  $\mathbf{w}_k$  and distance  $d(H_k, \mathbf{0}) = |b_k|/||\mathbf{w}_k||$  to the origin (same as in Eq. (13.4)).

According to the classification rule in Eq. (13.100), the 2-D feature space is partitioned into three regions each for one of the  $K = 3$  classes by the decision boundaries  $B_{12}$ ,  $B_{23}$ ,  $B_{31}$ , of which boundary  $B_{ij}$  between classes  $C_i$  and  $C_j$  is composed of all points  $\mathbf{x} \in B_{ij}$  satisfying  $f_i(\mathbf{x}) = f_j(\mathbf{x})$ . Same as in binary SVM, we further define for each of the  $K$  classes two additional straight lines (planes or hyperplanes if  $d \geq 3$ ) parallel to  $H_k$ , denoted by  $H_{k\pm}$  and represented by equations  $f_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k \pm c_k$ . Their distances to  $H_k$  can be found as (same as in Eq. (13.9)):

$$d(H_{k\pm}, H_k) = \frac{|c_k|}{||\mathbf{w}_k||} \quad (13.101)$$

Also as shown in the figure, the intersections of these parallel lines for classes  $C_1$  and  $C_2$  further determine two straight lines parallel to the decision boundary  $B_{12}$ , defining the decision margin between the support vectors of  $C_1$  and  $C_2$ . It can be seen that this decision margin is monotonically related to both  $|c_1|/||\mathbf{w}_1||$  and  $|c_2|/||\mathbf{w}_2||$ , which can be maximized by minimizing both  $||\mathbf{w}_1||$  and  $||\mathbf{w}_2||$ .

Now the multiclass classification problem can be formulated as to find the parameters  $\mathbf{w}_k$  and  $b_k$  for all  $k = 1, \dots, K$  so that  $\sum_{k=1}^K ||\mathbf{w}_k||^2$  is minimized and thereby the decision margins for all boundaries  $B_{ij}$  between  $C_i$  and  $C_j$  are maximized, under the constraint that every training sample  $\mathbf{x}_n$  labeled by  $y_n$  is correctly classified with a decision margin of 1 (the distance between the support

vectors on both sides of  $B_{ij}$  is 2):

$$\begin{aligned} \text{minimize: } & \frac{1}{2} \sum_{k=1}^K \|\mathbf{w}_k\|^2 \\ \text{subject to: } & \mathbf{w}_{y_n}^T \mathbf{x}_n + b_{y_n} \geq \mathbf{w}_k^T \mathbf{x}_n + b_k + 2, \quad (k = 1, \dots, K, k \neq y_n) \end{aligned} \quad (13.102)$$

Similar to Eq. (13.54) in soft margin SVM, the constraints of the optimization problem above can be relaxed by including in each constraint an extra error term  $\xi_{nk} \geq 0$ , which is to be minimized. Now the problem can be reformulated as

$$\begin{aligned} \text{minimize: } & \frac{1}{2} \sum_{k=1}^K \|\mathbf{w}_k\|^2 + C \sum_{k=1}^K \sum_{n=1}^N \xi_{nk} \\ \text{subject to: } & \mathbf{w}_{y_n}^T \mathbf{x}_n + b_{y_n} \geq \mathbf{w}_k^T \mathbf{x}_n + b_k + 2 - \xi_{nk} \\ & \xi_{nk} \geq 0 \\ & (n = 1, \dots, N, \quad k = 1, \dots, K, k \neq y_n) \end{aligned} \quad (13.103)$$

The Lagrangian function of this constrained optimization problem is

$$\begin{aligned} L(\mathbf{w}, b, \xi, \alpha, \beta) = & \frac{1}{2} \sum_{k=1}^K \|\mathbf{w}_k\|^2 + C \sum_k \sum_n \xi_{nk} \\ & - \sum_k \sum_n \alpha_{nk} [(\mathbf{w}_{y_n} - \mathbf{w}_k)^T \mathbf{x}_n + b_{y_n} - b_k - 2 + \xi_{nk}] - \sum_k \sum_n \beta_{nk} \xi_{nk} \end{aligned} \quad (13.104)$$

where

$$\begin{aligned} \alpha_{ny_n} = \beta_{ny_n} = 0, \quad \xi_{ny_n} = 2; \\ \alpha_{nk} \geq 0, \quad \beta_{nk} \geq 0, \quad \xi_{nk} \geq 0 \\ (n = 1, \dots, N, \quad k = 1, \dots, K, k \neq y_n) \end{aligned} \quad (13.105)$$

$$\frac{\partial}{\partial \mathbf{w}_k} L(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_k \mathbf{w}_k + \sum_n \alpha_{nk} \mathbf{x}_n = 0 \quad (13.106)$$

$$\frac{\partial}{\partial b_k} L(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_k \sum_n \alpha_{nk} = 0 \quad (13.107)$$

$$\frac{\partial}{\partial \xi_{nk}} L(\mathbf{w}, b, \xi, \alpha, \beta) = C - \alpha_{nk} - \beta_{nk} = 0 \quad (13.108)$$

$$\mathbf{f}(\mathbf{x}) = \mathbf{W}^T \mathbf{x} \quad (13.109)$$

where

$$\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K] \quad (13.110)$$

An unlabeled sample  $\mathbf{x}$  is classified to class  $C_i$  if  $f_i(\mathbf{x}) = \max(f_1(\mathbf{x}), \dots, f_K(\mathbf{x}))$ .

The training of the classifier is to find all weight vectors in  $\mathbf{W}$  based on the training dataset. To do so, we first define a cost function

////

## 13.6 Kernelized Bayes classifier

The naive Bayes classification is based on a quadratic decision function and is therefore unable to classify data that are not quadratically separable. However, as shown below, the Bayes method can be reformulated so that all data points appear in the form of an inner product, and the kernel method can be used to map the original space into a higher dimensional space in which all groups can be separated even though they are not quadratically separable in the original space.

Consider a binary Bayes classifier by which any sample  $\mathbf{x}$  is classified into either of the two classes  $C_+$  and  $C_-$  depending on whether  $\mathbf{x}$  is on the positive or negative side of the quadratic decision surface (Eq. (12.22)):

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{W} \mathbf{x} + \mathbf{w}^T \mathbf{x} + w \begin{cases} > 0, & \mathbf{x} \in C_+ \\ < 0, & \mathbf{x} \in C_- \end{cases} \quad (13.111)$$

As show in Eq. (12.21), here

$$\begin{aligned} \mathbf{W} &= -\frac{1}{2}(\boldsymbol{\Sigma}_+^{-1} - \boldsymbol{\Sigma}_-^{-1}) \\ \mathbf{w} &= \boldsymbol{\Sigma}_+^{-1} \mathbf{m}_+ - \boldsymbol{\Sigma}_-^{-1} \mathbf{m}_- \\ w &= -\frac{1}{2}(\mathbf{m}_+^T \boldsymbol{\Sigma}_+^{-1} \mathbf{m}_+ - \mathbf{m}_-^T \boldsymbol{\Sigma}_-^{-1} \mathbf{m}_-) - \frac{1}{2} \log \frac{|\boldsymbol{\Sigma}_+|}{|\boldsymbol{\Sigma}_-|} + \log \frac{P(C_+)}{P(C_-)} \end{aligned} \quad (13.112)$$

where  $\boldsymbol{\Sigma}_+$  and  $\boldsymbol{\Sigma}_-$  are respectively the covariance matrices of the samples in  $C_+$  and  $C_-$ ,

$$\mathbf{m}_+ = \frac{1}{N_+} \sum_{\mathbf{x} \in C_+} \mathbf{x}, \quad \mathbf{m}_- = \frac{1}{N_-} \sum_{\mathbf{x} \in C_-} \mathbf{x} \quad (13.113)$$

are their mean vectors, and  $P_+ = N_+/N$  and  $P_- = N_-/N$  are their prior probabilities. Specially if  $\boldsymbol{\Sigma}_+ = \boldsymbol{\Sigma}_- = \boldsymbol{\Sigma}$  and therefore  $\mathbf{W} = \mathbf{0}$ , the quadratic decision surface becomes a linear decision plane discribed by

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w = [\boldsymbol{\Sigma}^{-1}(\mathbf{m}_+ - \mathbf{m}_-)]^T \mathbf{x} + w = 0 \quad (13.114)$$

in the same form as the decision equation for the support vector machine. An unlabeled sample  $\mathbf{x}$  is classified into either of the two classes depending on on which side of a threshold  $-w$  its projection onto the normal direction  $\mathbf{w}$  of the decision plane lies.

As matrices  $\mathbf{W}$ ,  $\boldsymbol{\Sigma}_+^{-1}$ , and  $\boldsymbol{\Sigma}_-^{-1}$  are all symmetric, they can be written in the

following eigendecomposition form:

$$\mathbf{W} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T = \mathbf{U}\mathbf{U}^T, \quad \mathbf{\Sigma}_+^{-1} = \mathbf{V}_+\mathbf{\Lambda}_+\mathbf{V}_+^T = \mathbf{U}_+\mathbf{U}_+^T, \quad \mathbf{\Sigma}_-^{-1} = \mathbf{V}_-\mathbf{\Lambda}_-\mathbf{V}_-^T = \mathbf{U}_-\mathbf{U}_-^T \quad (13.115)$$

where  $\mathbf{U} = \mathbf{V}\mathbf{\Lambda}^{1/2}$ ,  $\mathbf{U}_+ = \mathbf{V}_+\mathbf{\Lambda}_+^{1/2}$  and  $\mathbf{U}_- = \mathbf{V}_-\mathbf{\Lambda}_-^{1/2}$ . We can write vector  $\mathbf{w}$  as:

$$\mathbf{w} = \mathbf{\Sigma}_+^{-1}\mathbf{m}_+ - \mathbf{\Sigma}_-^{-1}\mathbf{m}_- = \mathbf{U}_+\mathbf{U}_+^T\frac{1}{N_+}\sum_{\mathbf{x}_+ \in C_+}\mathbf{x}_+ - \mathbf{U}_-\mathbf{U}_-^T\frac{1}{N_-}\sum_{\mathbf{x}_- \in C_-}\mathbf{x}_- \quad (13.116)$$

Any unlabeled sample  $\mathbf{x}$  can now be classified into either of the two classes based on its decision function:

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{x}^T\mathbf{W}\mathbf{x} + \mathbf{w}^T\mathbf{x} + w \\ &= \mathbf{x}^T\mathbf{U}\mathbf{U}^T\mathbf{x} + \left(\frac{1}{N_+}\sum_{\mathbf{x}_+}\mathbf{x}_+^T\mathbf{U}_+\mathbf{U}_+^T\right)\mathbf{x} - \left(\frac{1}{N_-}\sum_{\mathbf{x}_-}\mathbf{x}_-^T\mathbf{U}_-\mathbf{U}_-^T\right)\mathbf{x} + w \\ &= \mathbf{z}^T\mathbf{z} + \frac{1}{N_+}\sum_{\mathbf{z}_{++}}(\mathbf{z}_{++}^T\mathbf{z}_+) - \frac{1}{N_-}\sum_{\mathbf{z}_{--}}(\mathbf{z}_{--}^T\mathbf{z}_-) + w \end{aligned} \quad (13.117)$$

where

$$\begin{cases} \mathbf{z}_{++} = \mathbf{U}_+^T\mathbf{x}_+ & (\mathbf{x}_+ \in C_+) \\ \mathbf{z}_{--} = \mathbf{U}_-^T\mathbf{x}_- & (\mathbf{x}_- \in C_-) \end{cases}, \quad \begin{cases} \mathbf{z} = \mathbf{U}^T\mathbf{x} \\ \mathbf{z}_+ = \mathbf{U}_+^T\mathbf{x} \\ \mathbf{z}_- = \mathbf{U}_-^T\mathbf{x} \end{cases} \quad (13.118)$$

As all data points appear in the form of an inner product, the kernel method can be used by replacing all inner products in the decision function by the corresponding kernel functions:

$$f(\mathbf{x}) = K(\mathbf{z}, \mathbf{z}) + \frac{1}{N_+}\sum_{\mathbf{z}_{++}}K(\mathbf{z}_{++}, \mathbf{z}_+) - \frac{1}{N_-}\sum_{\mathbf{z}_{--}}K(\mathbf{z}_{--}, \mathbf{z}_-) + b = p(\mathbf{x}) + b \quad (13.119)$$

This is the decision function in the new higher dimensional space, where  $p(\mathbf{x})$  is defined as a function composed of all terms in  $f(\mathbf{x})$  except the last offset term  $b$ , which is to replace  $w$  in the original space. To find this  $b$ , we first map all training samples into a 1-D space  $x_n = p(\mathbf{x}_n)$ , ( $n = 1, \dots, N$ ) and sort them together with their corresponding labelings  $y_1, \dots, y_N$ , and then search through all  $N - 1$  possible ways to partition them into two groups indexed respectively by  $1, \dots, k$  and  $k + 1, \dots, N$  to find the optimal  $k$  corresponding to the maximum labeling consistency measured by

$$\left| \sum_{n=1}^k y_n \right| + \left| \sum_{n=k+1}^N y_n \right|, \quad (k = 1, \dots, N - 1) \quad (13.120)$$

The middle point  $(x_k + x_{k+1})/2$  between  $x_k$  and  $x_{k+1}$  is used as the optimal threshold to separate the training samples into two classes in the 1-D space, i.e.,

the offset is  $b = -(x_k + x_{k+1})/2$ . Now the unlabeled point  $\mathbf{x}$  can be classified into either of the two classes  $C_+$  and  $C_-$ :

$$p(\mathbf{x}) + b \begin{cases} > 0, & \mathbf{x} \in C_+ \\ < 0, & \mathbf{x} \in C_- \end{cases} \quad (13.121)$$

In general, when all data points are kernel mapped to a higher dimensional space, the two classes can be more easily separated, even by a hyperplane based on the linear part of the decision function without the second order term. This allows the assumption that the two classes have the same covariance matrix so that  $\mathbf{W} = -(\boldsymbol{\Sigma}_+ - \boldsymbol{\Sigma}_-)/2 = \mathbf{0}$  and the second order term is dropped. This is the justification for the following two special cases:

- If we approximate both  $\boldsymbol{\Sigma}_+$  and  $\boldsymbol{\Sigma}_-$  by their average  $\boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_+ + \boldsymbol{\Sigma}_-)/2$ , then  $\mathbf{W} = \mathbf{0}$  and the decision function of any  $\mathbf{x}$  becomes

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + w = [\boldsymbol{\Sigma}^{-1}(\mathbf{m}_+ - \mathbf{m}_-)]^T \mathbf{x} + w \\ &= \left[ \mathbf{U}\mathbf{U}^T \left( \frac{1}{N_+} \sum_{\mathbf{x}_+} \mathbf{x}_+ - \frac{1}{N_-} \sum_{\mathbf{x}_-} \mathbf{x}_- \right) \right]^T \mathbf{x} + w \\ &= \frac{1}{N_+} \sum_{\mathbf{x}_+} \mathbf{x}_+^T \mathbf{U}\mathbf{U}^T \mathbf{x} - \frac{1}{N_-} \sum_{\mathbf{x}_-} \mathbf{x}_-^T \mathbf{U}\mathbf{U}^T \mathbf{x} + w \\ &= \frac{1}{N_+} \sum_{\mathbf{z}_+} \mathbf{z}_+^T \mathbf{z} - \frac{1}{N_-} \sum_{\mathbf{z}_-} \mathbf{z}_-^T \mathbf{z} + w \end{aligned} \quad (13.122)$$

where  $\mathbf{U}\mathbf{U}^T = \boldsymbol{\Sigma}^{-1}$ ,  $\mathbf{z}_+ = \mathbf{U}^T \mathbf{x}_+$ ,  $\mathbf{z}_- = \mathbf{U}^T \mathbf{x}_-$ , and  $\mathbf{z} = \mathbf{U}^T \mathbf{x}$ . This decision function can be converted to the following if the kernel method is used:

$$f(\mathbf{x}) = \frac{1}{N_+} \sum_{\mathbf{z}_+} K(\mathbf{z}_+, \mathbf{z}) - \frac{1}{N_-} \sum_{\mathbf{z}_-} K(\mathbf{z}_-, \mathbf{z}) + b \quad (13.123)$$

- More specially, if  $\boldsymbol{\Sigma}_+ = \boldsymbol{\Sigma}_- = \mathbf{I}$ , then the decision function above becomes

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + w = (\mathbf{m}_+ - \mathbf{m}_-)^T \mathbf{x} + w \\ &= \frac{1}{N_+} \sum_{\mathbf{x}_+} \mathbf{x}_+^T \mathbf{x} - \frac{1}{N_-} \sum_{\mathbf{x}_-} \mathbf{x}_-^T \mathbf{x} + w \end{aligned} \quad (13.124)$$

which can be converted to the following if the kernel method is used:

$$f(\mathbf{x}) = \frac{1}{N_+} \sum_{\mathbf{x}_+} K(\mathbf{x}_+, \mathbf{x}) - \frac{1}{N_-} \sum_{\mathbf{x}_-} K(\mathbf{x}_-, \mathbf{x}) + b \quad (13.125)$$

Note that if the kernel method is used to replace an inner product by a kernel function  $K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$ , we need to map all data points to  $\phi(\mathbf{x}_n)$ , ( $n = 1, \dots, N$ ) in the higher dimensional space, instead of only mapping their means

$\phi(\mathbf{m}_+)$  and  $\phi(\mathbf{m}_-)$ , because the mapping of a sum is not equal to the sum of the mapped points if the kernel is not linear:

$$\phi(\mathbf{m}_+ - \mathbf{m}_-) \neq \phi(\mathbf{m}_+) - \phi(\mathbf{m}_-), \quad \phi(\mathbf{m}_\pm) = \phi\left(\frac{1}{n_\pm} \sum_{\mathbf{x} \in C_\pm} \mathbf{x}\right) \neq \frac{1}{n_\pm} \sum_{\mathbf{x} \in C_\pm} \phi(\mathbf{x}) \quad (13.126)$$

The three cases above are summarized below:

- Type I, Eq. (13.125)

$$f(\mathbf{x}) = \frac{1}{N_+} \sum_{\mathbf{x}_+} K(\mathbf{x}_+, \mathbf{x}) - \frac{1}{N_-} \sum_{\mathbf{x}_-} K(\mathbf{x}_-, \mathbf{x}) + b \quad (13.127)$$

- Type II, Eq. (13.122)

$$f(\mathbf{x}) = \frac{1}{N_+} \sum_{\mathbf{z}_+} K(\mathbf{z}_+, \mathbf{z}) - \frac{1}{N_-} \sum_{\mathbf{z}_-} K(\mathbf{z}_-, \mathbf{z}) + b \quad (13.128)$$

- Type III, Eq. (13.119)

$$f(\mathbf{x}) = \mathbf{z}^T \mathbf{z} + \frac{1}{N_+} \sum_{\mathbf{z}_{++}} K(\mathbf{z}_{++}, \mathbf{z}_+) - \frac{1}{N_-} \sum_{\mathbf{z}_{--}} K(\mathbf{z}_{--}, \mathbf{z}_-) + b \quad (13.129)$$

The Matlab code for the essential part of the algorithm is listed below. Given  $\mathbf{X}$  and  $\mathbf{y}$  for the data array composed of  $N$  training vectors  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  and their corresponding labelings  $\mathbf{y} = [y_1, \dots, y_N]$ , the code carries out the training and then classifies any unlabeled data point into either of the two classes. Parameter `type` selects any one of the three different versions of the algorithm, and the function  $K(\mathbf{X}, \mathbf{x})$  returns a 1-D array containing all  $N$  kernel functions  $K(\mathbf{x}_i, \mathbf{x})$ , ( $i = 1, \dots, N$ ) of the column vectors in  $\mathbf{X}$  and vector  $\mathbf{x}$ .

```
X=getData;
[m n]=size(X);
X0=X(:,find(y>0)); n0=size(X0,2); % separate C+ and C-
X1=X(:,find(y<0)); n1=size(X1,2);
m0=mean(X0,2); C0=cov(X0'); % calculate mean and covariance
m1=mean(X1,2); C1=cov(X1');
if type==1
    for i=1:n
        x(i)=sum(K(X0,X(:,i)))/n0-sum(K(X1,X(:,i)))/n1;
    end
elseif type==2
    C=inv(C0+C1); [V D]=eig(C); U=(V*D^(1/2))';
    Z=U*X; Z0=U*X0; Z1=U*X1;
    for i=1:n
        x(i)=sum(K(Z0,Z(:,i)))/n0-sum(K(Z1,Z(:,i)))/n1;
    end
end
```

```

elseif type==3
    C0=inv(C0);    C1=inv(C1);    W=-(C0-C1)/2;
    [V D]=eig(W);    U=(V*D^(1/2)).';    Z=U*X;
    [V0 D0]=eig(C0);    U0=(V0*D0^(1/2))';    Z0=U0*X;    Z00=U0*X0;
    [V1 D1]=eig(C1);    U1=(V1*D1^(1/2))';    Z1=U1*X;    Z11=U1*X1;
    for i=1:n
        x(i)=K(Z(:,i),Z(:,i))+sum(K(Z00,Z0(:,i)))/n0-sum(K(Z11,Z1(:,i)))/n1;
    end
end
[x I]=sort(x);    y=y(I);    % sort 1-D data together with their labelings
smax=0;
for i=1:n-1
    % find optimal threshold value b
    s=abs(sum(y(1:i)))+abs(sum(y(i+1:n)));
    if s>smax
        smax=s;    b=-(x(i)+x(i+1))/2;
    end
end
end

```

Note that  $\mathbf{W} = -(\boldsymbol{\Sigma}_0^{-1} - \boldsymbol{\Sigma}_1^{-1})/2$  may be either positive or negative definite, and its eigenvalue matrix  $\mathbf{D}$  may contain negative values and  $\mathbf{D}^{1/2}$  may contain complex values. Given any unlabeled data point  $\mathbf{x}$ , the code below is carried out

```

if type==1
    y(i)=sum(K(X0,x))/n0-sum(K(X1,x))/n1+b;
elseif type==2
    Z=U*x;
    y(i)=sum(K(Z0,z))/n0-sum(K(Z1,z))/n1+b;
elseif type==3
    z=U*x;    z0=U0*x;    z1=U1*x;
    y(i)=K(z,z)+sum(K(Z00,z0))/n0-sum(K(Z11,z1))/n1+b;
end

```

to classify  $\mathbf{x}$  into either  $C_+$  if  $y > 0$ , or  $C_-$  if  $y < 0$ .

In comparison with the SVM method, which requires solving a QP problem by certain iterative algorithm (either the interior point method or the SMO method), the kernel based Bayes method is closed-form and therefore extremely efficient computationally. Moreover, as shown in the examples below, this method is also highly effective as its classification results are comparable and often more accurate than those of the SVM method.

We now show a few examples to test all three types of the kernel based Bayes method based on a set of simulated 2-D data. Both linear kernel and RBF kernel  $K(\mathbf{x}, \mathbf{y}) = \exp(-\gamma\|\mathbf{x} - \mathbf{y}\|)$ . The value of the parameter  $\gamma$  used in the examples is 5, but it can be fine tuned in a wide range (e.g.,  $1 < \gamma < 9$  to make proper trade-off between accuracy and avoiding overfitting. The performances of these

method are also compared with the SVM method implemented by the Matlab function:

```
fitcsvm(X',y,'Standardize',true,'KernelFunction','linear','KernelScale','auto'))
```

**Example 1:** Based on some 2-D training datasets, four different binary classifiers are tested. The correct rates of each of the four methods are listed below, together with the corresponding partitionings of the 2-D space shown in the figures.

	<i>Kernel</i>	Matlab SVM	Kernel Bayes I	Kernel Bayes II	Kernel Bayes III
Test 1	linear	93.0%	88.0%	94.0%	94.0%
Test 2	linear	73.0%	80.0%	79.5%	96.5%
Test 3	RBF	98.5%	100%	100%	100%

(13.130)

- Test 1: Two sets of data are generated based on the following mean vectors and covariance matrices:

$$\mathbf{m}_+ = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad \mathbf{m}_- = \begin{bmatrix} +1 \\ 0 \end{bmatrix}, \quad \boldsymbol{\Sigma}_+ = \boldsymbol{\Sigma}_- = 3 \times \begin{bmatrix} 1 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \quad (13.131)$$

The linear kernel is used for all methods. The kernel Bayes methods II and III achieve the best result (94%), slightly better than that of the standard SVM (93%). But the kernel Bayes method I performs poorly (84.7%), as it is based only on the means of the two classes without taking into consideration their covariances representing the distribution of the data points.

- Test 2: Two sets of data not linearly separable are classified by the four methods all based on the linear kernel. The kernel Bayes III achieves the best result (96.5%), due to its quadratic term in the decision function, the kernel Bayes I and II perform much more poorly, but still slightly better than the SVM method.
- Test 3: The same dataset as above is classified by the methods but now based on the RBF kernel. While the SVM performs reasonably well (98.5%), all three versions of the kernel Bayes method (with  $\gamma = 5$ ) achieve the perfect result with all points of the two classes completely separate. However, the partitioning of the space by the kernel Bayes III is highly fragmented, showing a sign of overfitting.

**Example 2:** The three datasets are generated using the Matlab code at <https://www.mathworks.com/help/stats/support-vector-machines-for-binary-classification>. A parameter value  $\gamma = 5$  is used for the three versions of the kernel Bayes method.



	<i>Kernel</i>	Matlab SVM	Kernel Bayes I	Kernel Bayes II	Kernel Bayes III
Test 1	Linear	58.75%	60.0%	60.25%	97.0%
Test 2	RBF	97.75%	98.50%	98.50%	99.50%
Test 3	RBF	98.0%	100%	100%	100%
Test 4	RBF	96.0%	98.5%	95.5%	97.0%

(13.132)

- Test 1: Exclusive OR dataset, linear kernel  
As the data are not linearly separable, all methods performed poorly except kernelized Bayes Type III with a second order term in the decision function.
- Test 2: Exclusive OR dataset, RBF kernel  
All four methods performed very well, but all three variations of the kernelized Bayes method achieved higher correct rates than the SVM.
- Test 3: Cocentric ring dataset, RBF kernel
- Test 4: Multi-cluster dataset, RBF kernel

**Example 3:** The classification results of Fisher's iris data by the SVM method (Matlab function `fitcsvm`) and the kernel Bayes methods are shown below. This is an example used to illustrate the SVM method in the documentation of `fitcsvm` (<https://www.mathworks.com/help/stats/fitcsvm.html>).

In this example only two (3rd and 4th) of the four features are used, with half of the samples used for training while the other half for testing. Note that the second class (setosa in green) and third class (versicolor in blue) not linearly separable can be better separated by the kernel Bayes method.

## 13.7 Hierarchical (Tree) Classifiers

Both supervised classification and unsupervised clustering can be carried out in a hierarchical fashion to classify the input patterns or group them into clusters, very much like the hierarchy of biological classifications with different *taxonomic ranks* (domain, kingdom, phylum, class, order, family, genus, and species).

- **Unsupervised clustering**

The hierarchical clustering can be obtained in either a top-down or bottom up manner.

- Top-down method:

All patterns in the data set are initially treated as a single cluster as the root of the tree, which is then subdivided (split) into a set of two or more smaller clusters, each represented as a node in the tree structure. This process is carried out recursively until eventually each cluster contains only one pattern, represented as a leaf node of the tree.

- Bottom-up method:

every pattern in the data set is initially treated as a cluster as a leaf node of the tree, which will then be merged to form larger clusters. Again, this process is carried out recursively until eventually all patterns are merged into a single cluster at the root of the tree.

In either the top-down or the bottom-up method, the specific method for the splitting or merging at each tree node is based on certain similarity measurement such as the distance between two clusters. The resulting tree structure obtained by either method can then be truncated at any level between the root and the leaf nodes to obtain a set of clusters, depending on the desired number and sizes of these clusters.

- **Supervised classification**

If labeled training data are available, both the top-down and the bottom-up clustering methods can also be used in the training stage of the supervised classification methods, with the only difference that now the splitting or merging is applied to labeled classes instead of individual patterns, and each leaf node represents one of the classes, rather than a single pattern. After the tree structure is obtained, the training is complete and any unlabeled pattern can be classified at the tree root and then subsequently the tree nodes at lower levels until it is classified into one of the leaf nodes of the tree, corresponding to a specific class.

This hierarchical classification method is especially useful when the number of classes and the number  $D$  of feature are both large. In this case it may be very difficult to select a subset of  $d < D$  features good for separating all classes for a single-level classifier, by which *all* classes need to be classified at the same time, requiring, most likely, all  $N$  features. However, for a tree classifier, since each node is a two-class classifier, it is possible to select a small number of  $d \ll D$  features that are most relevant and suitable to represent the two subsets of classes.

In the following we consider both the bottom-up and top-down methods for hierarchical clustering/classification.

#### **Bottom-Up method**

The bottom-up hierarchical classifier is trained based on  $K$  classes  $C_1, \dots, C_K$ , each containing  $n_k$  ( $k = 1, \dots, K$ ) labeled patterns  $\mathbf{x} \in C_k$ .

1. Compute the  $K(K - 1)/2$  pairwise Bhattacharyya distances between every two classes  $C_i$  and  $C_j$ :

$$d_B(C_i, C_j) = \frac{1}{4}(\mathbf{m}_i - \mathbf{m}_j)^T \left[ \frac{\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j}{2} \right]^{-1} (\mathbf{m}_i - \mathbf{m}_j) + \log \left[ \frac{|\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j|}{(|\boldsymbol{\Sigma}_i| |\boldsymbol{\Sigma}_j|)^{1/2}} \right] \quad (13.133)$$

2. Merge the two classes corresponding to the smallest  $d_B$  to form a new class

$C_i \cup C_j = C_k$ , compute its mean and covariance:

$$\mathbf{m}_k = \frac{1}{n_i + n_j} [n_i \mathbf{m}_i + n_j \mathbf{m}_j] \quad (13.134)$$

and

$$\mathbf{\Sigma}_k = \frac{1}{n_i + n_j} [n_i (\mathbf{\Sigma}_i + (\mathbf{m}_i - \mathbf{m}_k)(\mathbf{m}_i - \mathbf{m}_k)^T) + n_j (\mathbf{\Sigma}_j + (\mathbf{m}_j - \mathbf{m}_k)(\mathbf{m}_j - \mathbf{m}_k)^T)] \quad (13.135)$$

Delete the old classes  $C_i$  and  $C_j$ . Now there are  $K - 1$  classes left.

3. Compute the distance between the new class  $C_k$  and all  $K - 2$  remaining classes.
4. Repeat the previous steps until eventually all classes are merged into a single group containing all  $K$  classes, the binary tree structure is thus obtained.

#### Top-Down method

Generate a binary tree by recursively partitioning all classes into two subgroups with the maximum Bhattacharyya distance

1. Compute the between-class scatter matrix  $\mathbf{S}_B$  of the  $K$  classes, find its maximum eigenvalue  $\lambda_i$  and the corresponding eigenvectors  $\mathbf{v}_i$ ;
2. Project all data points onto  $\mathbf{v}_1$ :

$$y_n = \mathbf{x}_n^T \mathbf{v} \quad (n = 1, \dots, N) \quad (13.136)$$

3. Sort all data points  $\{y_1, \dots, y_N\}$  along this 1-D space and partition them into two subgroups with maximum Bhattacharyya (between-group) distance.
4. Carry out the steps above recursively to each of the two subgroups, until eventually every subgroup contains only one classes

Once the hierarchical structure is constructed by either the bottom-up or top-down method, we need to build a binary classifier at each node of structure, by which any given pattern is classified into either the left group  $G_l$  or right group  $G_r$ :

- According to the specific classification method used, find the discriminant functions  $D_l(\mathbf{x})$  and  $D_r(\mathbf{x})$  for the two subgroups based on the training data.
- Select the best  $d < D$  features most suitable for separating the two groups  $G_l$  and  $G_r$ , based on any of the feature selection methods such as those listed below:
  - Choosing  $d$  features directly from the  $D$  original ones using between-class distance (Bhattacharyya distance) as the criterion,
  - Carry out KLT based on the between-class scatter matrix  $\mathbf{S}_B$  and use the first  $d$  principal components for the binary classification.

As here only two groups of classes need to be distinguished, the number of features  $M$  can be expected to be small.

- Any unlabeled pattern  $\mathbf{x}$  enters the classifier at the root of the tree and is classified to either the left or the right sub-group of the node according to the discriminant function

$$\mathbf{x} \in \begin{cases} G_l & \text{if } D_l(\mathbf{x}) > D_r(\mathbf{x}) \\ G_r & \text{if } D_r(\mathbf{x}) < D_l(\mathbf{x}) \end{cases} \quad (13.137)$$

This process is carried out recursively at each of the subsequent nodes until eventually  $\mathbf{x}$  reaches one of the leaf nodes corresponding to a single class, to which the sample  $\mathbf{x}$  is therefore classified.

### Example

The hierarchical clustering method is applied to a dataset composed of seven normally distributed clusters each containing 25 sample vectors in an  $N = 4$  dimensional space. The PCA method is used to project the data in 4-D space into a 2-D space spanned by the first two principal components, as shown below:

The clustering result is shown below. Each column in the display represents the four components of a 4-D vector, color coded by a spectrum from red (low values) through green (middle) to blue (high values).

More examples of clustering analysis applied to gene data analysis in bioinformatics can be found here: <http://fourier.eng.hmc.edu/bioinformatics/intro/index.html>  
In particular, an example of this method is available here: <http://fourier.eng.hmc.edu/bioinformatics/intro/index.html>

# 14 Clustering Analysis

---

All classification algorithms discussed above are supervised in nature based on the assumed availability of some training data in which each of the data patterns  $\mathbf{x}_n \in \mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  is labeled by  $y_n \in \mathbf{y} = [y_1, \dots, y_n]^T$ , i.e., the class they each belong is known. However, when such training set is not available, there is still the need to explore some potential structure of the data in the form of a set of unknown number  $K$  of clusters  $\{C_1, \dots, C_K\}$ , each composed of a set of similar data points close to each other in the feature space. This can be accomplished by the unsupervised method of *clustering analysis*, based only on the given dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , without any additional prior knowledge, such as the labeling of the data samples. We will now consider a few algorithms for clustering analysis.

## 14.1 K-means clustering

As suggested by the name of K-means clustering, in this method a set of  $K$  mean vectors  $\mathbf{m}_1, \dots, \mathbf{m}_K$  are used to represent the clusters in the feature space, based on the assumption that there exist a set of  $K$  clusters in the dataset.

The K-means clustering algorithm can be formulated as an optimization problem to minimize an objective function

$$J = \sum_{n=1}^N \sum_{k=1}^K P_{nk} \|\mathbf{x}_n - \mathbf{m}_k\|^2 \quad (14.1)$$

where

$$P_{nk} = \begin{cases} 1 & \text{if } k = \arg \min_l \|\mathbf{x}_n - \mathbf{m}_l\| \\ 0 & \text{otherwise} \end{cases} \quad (14.2)$$

indicates that  $\mathbf{x}_n$  is assigned to the  $k$ th cluster  $C_k$  if its distance to the mean  $\mathbf{m}_k$  is minimum. To minimize the objective function  $J$  with respect to  $\mathbf{m}_k$ , we set its derivative with respect to each  $\mathbf{m}_k$  to zero:

$$\frac{d}{d\mathbf{m}_k} J = 2 \sum_{n=1}^N P_{nk} \|\mathbf{x}_n - \mathbf{m}_k\| = \mathbf{0} \quad k = 1, \dots, K \quad (14.3)$$



```

Mnew=zeros(d,K);           % initialize new means
Number=zeros(1,K);
for i=1:N                   % for all N samples
    x=X(:,i);
    dmin=inf;
    for k=1:K               % for all K clusters
        d=norm(x-Mold(:,k));
        if d<dmin
            dmin=d;  j=k;
        end
    end
    Number(j)=Number(j)+1;
    Mnew(:,j)=Mnew(:,j)+x;
end
for k=1:K
    if Number(k)>0
        Mnew(:,k)=Mnew(:,k)/Number(k);
    end
end
er=norm(Mnew-Mold);        % terminate if means no longer change
end

```

### Example 1

The K-means algorithm is applied to a simulated dataset in 3-D space with  $C = 4$  clusters. The results are shown in the figure below, where the three panels show the results corresponding to  $K = C - 1$  (left),  $K = C$  (middle), and  $K = C + 1$  (right). The initial positions of the  $K$  means are marked by the black squares, while their subsequent positions through out the iteration are marked by smaller dots. The iteration terminates once the means have moved to the centers of the clusters and no longer change positions.

The clustering results corresponding to  $K = 3, 4, 5$  can be evaluated by the separability  $tr(\mathbf{S}_T^{-1}\mathbf{S}_B)$ , the  $K$  intra-cluster distance  $tr(\mathbf{\Sigma}_k)$  ( $k = 1, \dots, K$ ) of the resulting clusters:

	K=C-1=3			K=C=4				K=C+1=5				
Separability	1.76			2.56				2.58				
Intra-cluster distance	9.1	44.3	11.8	10.8	12.7	11.1	9.1	10.8	9.1	11.1	8.9	9.4

(14.8)

and the  $K(K - 1)/2$  inter-cluster (Bhattacharyya) distances (between any two

of the  $K$  clusters):

K=C-1=3		K=C=4			K=C+1=5			
1	2	1	2	3	1	2	3	4
2	10.9	2	4.0		2	4.0		
3	21.9	3	5.1	1.6	3	5.1	1.6	
	184.7	4	2.4	2.3	4	7.0	4.3	0.3
					5	17.3	15.5	8.9
								11.1

(14.9)

We see that when  $K = 3 < C = 4$ , the intra-cluster distance of the 2nd cluster is significantly greater than the other two, indicating the cluster may contain two smaller clusters. Also, when  $K = 5 > C = 4$ , the inter-cluster distance between clusters 3 and 4 is significantly smaller than others, indicating the two clusters are too close and can therefore be merged.

While the K-means method is simple and effective, it has the main shortcoming that the number of clusters  $K$  needs to be specified, although in practice it is typically unknown a head of time. In this case, one could try different  $K$  values and compare the corresponding results in terms of the intra and inter cluster distances, as well as the separabilities of the resulting clusters, as shown in the example above. Moreover, if the intra-cluster distance of a cluster is too large indicating it may contain more than one cluster (e.g.,  $K = 3$  in the example), it can be split; on the other hand, if the inter-cluster distance between two clusters is too small (e.g.,  $K = 5$  in the example), the two clusters may belong to the same cluster and need to be merged. Following such merging and/or splitting, a few more iterations can be carried out to make sure the final clustering results are optimal. The figure below shows the clustering results of the same dataset above but modified by merging and splitting. The left pannel is for  $K = 3$ , but the second cluster (green) is split, while the right pannel is for  $K = 5$ , when the 4th (yellow) and 5th (cyan) clusters are merged.

The idea of modifying the clustering results by merging and splitting leads to the algorithm of Iterative Self-Organizing Data Analysis Technique (ISODATA), which allows the number of clusters  $K$  to be adjusted automatically during the iteration by merging clusters close to each other and splitting clusters with large intra-cluster distances. However, this algorithm is highly heuristic as the various parameters such the threshold values for merging and splitting need to be specified.

### Example 2

The K-means clustering method is applied to the dataset of ten handwritten digits from 0 to 9 used previously. The clustering result is visualized based on the KLT that maps the data samples in the original 256-D space into the 3-D space spanned by the three eigenvectors corresponding to the three greatest eigenvalues of the covariance matrix of the dataset. The ground truth labelings



are color coded as shown on the left, while the clustering result is shown on the right. We see that the clustering results match the original data reasonably well.

The clustering result is also shown in the confusion matrix, of which the columns and rows represent respectively the clusters based on the K-means clustering and the ground truth labeling (not used). In other words, the element in the  $i$ th row and  $j$ th column is the number of samples labeled to belong to the  $i$ th class but assigned to the  $j$ th cluster.

$$\begin{bmatrix} 1 & 0 & 25 & 165 & 27 & 2 & 3 & 0 & 1 & 0 \\ 0 & 209 & 0 & 0 & 2 & 0 & 13 & 0 & 0 & 0 \\ 2 & 3 & 2 & 0 & 8 & 2 & 14 & 151 & 0 & 42 \\ 103 & 0 & 0 & 1 & 5 & 3 & 83 & 0 & 0 & 29 \\ 0 & 28 & 1 & 0 & 15 & 2 & 62 & 0 & 115 & 1 \\ 44 & 0 & 1 & 1 & 167 & 0 & 8 & 0 & 3 & 0 \\ 0 & 6 & 160 & 14 & 44 & 0 & 0 & 0 & 0 & 0 \\ 1 & 6 & 0 & 0 & 3 & 122 & 74 & 2 & 1 & 15 \\ 82 & 1 & 1 & 1 & 33 & 3 & 47 & 2 & 0 & 54 \\ 4 & 2 & 0 & 1 & 0 & 28 & 119 & 1 & 64 & 5 \end{bmatrix} \quad (14.10)$$

All samples are converted from 256-dimensional vector back to  $16 \times 16$  image form and the averages of the samples in each of the ten cluster are shown in the figure below. We see that each of the ten clusters clearly represents one distinct digit.

## 14.2 Gaussian mixture model

In K-means clustering, each sample point  $\mathbf{x}$  is assigned to one of the  $K$  clusters if it has the minimum Euclidean distance to the mean of the cluster. But here the distribution of the data samples in the cluster represented by the covariance is not taken into consideration.

This issue can be addressed by the method of *Gaussian mixture model (GMM)* based on the assumption that each of the  $K$  clusters  $\{C_1, \dots, C_K\}$  in the given dataset can be modeled by a Gaussian distribution  $\mathcal{N}(\mathbf{x}; \mathbf{m}_k, \boldsymbol{\Sigma}_k)$ , ( $k = 1, \dots, K$ ) in terms of both the mean vector  $\mathbf{m}_k$  and covariance matrix  $\boldsymbol{\Sigma}_k$ . The overall distribution of the entire dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  is the weighted sum of the  $K$  Gaussian distributions:

$$p(\mathbf{x}) = \sum_{k=1}^K P_k \mathcal{N}(\mathbf{x}; \mathbf{m}_k, \boldsymbol{\Sigma}_k) = \sum_{k=1}^K P_k \left[ \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mathbf{m}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \mathbf{m}_k) \right) \right] \quad (14.11)$$

where  $P_k$  is the weight for the  $k$ th Gaussian  $\mathcal{N}(\mathbf{x}; \mathbf{m}_k, \boldsymbol{\Sigma}_k)$ , satisfying

$$\int_{-\infty}^{\infty} p(\mathbf{x}) d\mathbf{x} = \sum_{k=1}^K P_k \int_{-\infty}^{\infty} \mathcal{N}(\mathbf{x}; \mathbf{m}_k, \boldsymbol{\Sigma}_k) d\mathbf{x} = \sum_{k=1}^K P_k = 1 \quad (14.12)$$

The GMM clustering is carried out by the method of *expectation maximization (EM)*, by which all model parameters denoted by  $\theta = \{P_k, \mathbf{m}_k, \mathbf{\Sigma}_k, (k = 1, \dots, K)\}$  are estimated based on the given dataset, and then each sample in the dataset is assigned to one of the clusters.

We note that the GMM model in Eq. (14.11) is actually the same as Eq. (12.11) in the naive Bayes classification. These two methods are similar in the sense that each cluster or class  $C_k$  is modeled by a Gaussian  $\mathcal{N}(\mathbf{x}, \mathbf{m}_k, \mathbf{\Sigma}_k)$ , weighted by  $P_k$ , and the model parameters  $\mathbf{m}_k$  and  $\mathbf{\Sigma}_k$ , as well as  $P_k$ , are to be estimated based on the given dataset. However, the two methods are different in that the dataset  $\mathbf{X}$  in the supervised naive Bayes method is labeled by  $\mathbf{y}$ , while in GMM such a labeling vector is not available. Instead, here we will introduce a latent or hidden variable  $\mathbf{z} = [z_1, \dots, z_K]^T$  as the cluster labeling of the samples in the given dataset  $\mathbf{X}$ . This latent variable  $\mathbf{z}$  is binary vector, of which each component is a binary random variable  $z_k \in \{0, 1\}$ . Only one of these  $K$  components is 1, e.g.,  $z_k = 1$ , indicating a sample  $\mathbf{x}$  in the dataset belongs to the  $k$ th cluster  $C_k$ , while all others are 0, i.e., these  $K$  binary variables add up to 1,  $\sum_{k=1}^K z_k = 1$ .

We further introduce the following probabilities for each of the  $K$  clusters  $C_k$  ( $k = 1, \dots, K$ ):

- The *prior probability* for any unobserved data sample  $\mathbf{x} \in \mathbf{X}$  in the dataset to belong to  $C_k$ , represented by  $z_k = 1$ :

$$P(z_k = 1) = P_k \quad (k = 1, \dots, K) \quad \text{satisfying} \quad \sum_{k=1}^K P_k = 1 \quad (14.13)$$

These  $K$  prior probabilities add up to 1, i.e., the  $K$  events  $z_k = 1$  ( $k = 1, \dots, K$ ) are mutually exclusive and complementary, as any sample  $\mathbf{x}$  belongs to one and only one of the  $K$  clusters.

- The probability distribution of all samples in  $C_k$ , assumed to be a Gaussian:

$$p(\mathbf{x}|z_k = 1, \theta) = \mathcal{N}(\mathbf{x}; \mathbf{m}_k, \mathbf{\Sigma}_k) \quad (14.14)$$

- The joint distribution of any  $\mathbf{x}$  and  $z_k = 1$ :

$$p(\mathbf{x}, z_k = 1|\theta) = p(\mathbf{x}|z_k = 1, \theta) P(z_k = 1) = P_k \mathcal{N}(\mathbf{x}; \mathbf{m}_k, \mathbf{\Sigma}_k) \quad (14.15)$$

Marginalizing this joint probability over the latent variable  $z_k$ , we get the Gaussian mixture model, the distribution  $p(\mathbf{x})$  of any sample  $\mathbf{x}$  regardless to which cluster it belongs:

$$p(\mathbf{x}|\theta) = \sum_{k=1}^K p(\mathbf{x}, z_k = 1|\theta) = \sum_{k=1}^K p(\mathbf{x}|z_k = 1, \theta) P(z_k = 1) = \sum_{k=1}^K P_k \mathcal{N}(\mathbf{x}; \mathbf{m}_k, \mathbf{\Sigma}_k) \quad (14.16)$$

Note that Eqs. (14.13) through (14.16) are the same as Eqs. (12.8) through (12.11) in the naive Bayes classifier, respectively.

All such probabilities defined for  $z_k = 1$  can be generalized to  $\mathbf{z} = [z_1, \dots, z_K]^T$

for all  $K$  clusters:

$$p(\mathbf{z}|\boldsymbol{\theta}) = \prod_{k=1}^K P_k^{z_k} \quad (14.17)$$

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \prod_{k=1}^K \mathcal{N}(\mathbf{x}, \mathbf{m}_k, \boldsymbol{\Sigma}_k)^{z_k} \quad (14.18)$$

$$p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = p(\mathbf{z}) p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \prod_{k=1}^K (P_k \mathcal{N}(\mathbf{x}, \mathbf{m}_k, \boldsymbol{\Sigma}_k))^{z_k} \quad (14.19)$$

Given the dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  containing  $N$  i.i.d. samples, we introduce  $N$  corresponding latent variables in  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$ , of which the  $n$ th column  $\mathbf{z}_n = [z_{n1}, \dots, z_{nK}]^T$  is the labeling vector of  $\mathbf{x}_n$ , i.e.,  $\mathbf{x}_n$  belongs to  $C_k$  if  $z_{nk} = 1$  (while  $z_{nl} = 0$  for all  $l \neq k$ ). Note that here  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$  is defined in the same way as  $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]$  in softmax regression (Section 6.2), both as the labeling of  $\mathbf{X}$ , with the only difference that  $\mathbf{Y}$  is provided in the training data available for a supervised method, but here  $\mathbf{Z}$  is a latent variable not part of the data provided for unsupervised clustering. Now we have

$$p(\mathbf{x}_n, \mathbf{z}_n|\boldsymbol{\theta}) = \prod_{k=1}^K (P_k \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \boldsymbol{\Sigma}_k))^{z_{nk}}, \quad (n = 1, \dots, N) \quad (14.20)$$

The likelihood function of the GMM model parameters  $\boldsymbol{\theta}$  to be estimated can be expressed as:

$$\begin{aligned} L(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Z}) &= p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = p\left([\mathbf{x}_1, \dots, \mathbf{x}_N], [\mathbf{z}_1, \dots, \mathbf{z}_N] \middle| \mathbf{m}_k, \boldsymbol{\Sigma}_k, P_k (k = 1, \dots, K)\right) \\ &= \prod_{n=1}^N p(\mathbf{x}_n, \mathbf{z}_n|\boldsymbol{\theta}) = \prod_{n=1}^N \prod_{k=1}^K (P_k \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \boldsymbol{\Sigma}_k))^{z_{nk}} \end{aligned} \quad (14.21)$$

and the log likelihood is:

$$\begin{aligned} l(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Z}) &= \log L(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Z}) = \log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = \log \left[ \prod_{n=1}^N \prod_{k=1}^K (P_k \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \boldsymbol{\Sigma}_k))^{z_{nk}} \right] \\ &= \sum_{n=1}^N \sum_{k=1}^K z_{nk} [\log P_k + \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \boldsymbol{\Sigma}_k)] \end{aligned} \quad (14.22)$$

Similar to the method of maximum likelihood estimation (MLE) which finds the model parameters in  $\boldsymbol{\theta}$  as those that maximize the likelihood function  $L(\boldsymbol{\theta})$  or its log function  $\log L(\boldsymbol{\theta})$ , here we find the model parameters in  $\boldsymbol{\theta} = \{P_k, \mathbf{m}_k, \boldsymbol{\Sigma}_k, (k = 1, \dots, K)\}$  as those that maximize the expectation of the log likelihood function above with respect to the latent variables in  $\mathbf{Z}$  in the following two iterative steps of the EM method:

- **E-step:** Find the expectation of the log likelihood function.

We first find the posterior probability for an observed  $\mathbf{x}_n$  to belong to cluster  $C_k$ , indicated by  $z_{nk} = 1$  (and  $z_{nl} = 0$  for all  $l \neq k$ ):

$$P_{nk} = P(z_{nk} = 1 | \mathbf{x}_n, \theta) = \frac{p(\mathbf{x}_n, z_{nk} = 1 | \theta)}{p(\mathbf{x}_n | \theta)} = \frac{P_k \mathcal{N}(\mathbf{x}_n; \mathbf{m}_k, \Sigma_k)}{\sum_{l=1}^K P_l \mathcal{N}(\mathbf{x}_n; \mathbf{m}_l, \Sigma_l)} \quad (n = 1, \dots, N; k = 1, \dots, K) \quad (14.23)$$

As  $\mathbf{x}_n$  has to belong to one of the  $K$  clusters, these  $K$  posterior probabilities add up to 1, same as the prior probabilities  $P(z_k = 1) = P_k$ :

$$\sum_{k=1}^K P(z_k = 1) = \sum_{k=1}^K P_k = 1, \quad \sum_{k=1}^K P(z_{nk} = 1 | \mathbf{x}_n, \theta) = \sum_{k=1}^K P_{nk} = 1 \quad (14.24)$$

This posterior probability  $P_{nk} = p(z_{nk} = 1 | \mathbf{x}_n, \theta)$  can be considered as the *responsibility* cluster  $C_k$  takes for  $\mathbf{x}_n$ . Once we have available all parameters on the right-hand side of Eq. (14.23) for  $P_{nk}$ , it is used to assign each sample  $\mathbf{x}_n$  to a cluster  $C_k$  with the maximum responsibility  $P_{nk} = \max_l P_{nl}$ .

We note that the definition of  $P_{nk}$  is similar to the softmax function  $s_{nk}$  with the only difference that here the weighted Gaussian function are used instead of the softmax function used in softmax regression.

We also note that the posterior probability  $P_{nk}$  defined above represents a *soft* decision, in the sense that it is possible for any  $\mathbf{x}_n$  to belong to any  $C_k$  with probability  $0 < P_{nk} < 1$  for all  $k = 1, \dots, K$ , instead of a *hard* decision, in the sense that  $\mathbf{x}_n$  belongs to only one specific  $C_k$  with  $P_{nk} = 1$ , while  $P_{nl} = 0$  for all  $l \neq k$ , as in K-means clustering.

We can now find the expectation of the log likelihood with respect to the latent variables in  $\mathbf{Z}$ :

$$\begin{aligned} E_{\mathbf{Z}}(\log L(\theta | \mathbf{X}, \mathbf{Z})) &= E_{\mathbf{Z}} \left[ \sum_{n=1}^N \sum_{k=1}^K z_{nk} [\log P_k + \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \Sigma_k)] \right] \\ &= \sum_{n=1}^N \sum_{k=1}^K E_{\mathbf{Z}}(z_{nk}) [\log P_k + \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \Sigma_k)] \\ &= \sum_{n=1}^N \sum_{k=1}^K P_{nk} [\log P_k + \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \Sigma_k)] \quad (14.25) \end{aligned}$$

The last equality is due to the following fact:

$$E_{\mathbf{Z}}(z_{nk}) = 1 P(z_{nk} = 1 | \mathbf{x}_n) + 0 P(z_{nk} = 0 | \mathbf{x}_n) = P(z_{nk} = 1 | \mathbf{x}_n) = P_{nk} \quad (14.26)$$

- **M-step:** Find the optimal model parameters that maximize the expectation of the log likelihood.

We first set to zero the derivatives of the expectation of the log likelihood with respect to each of the parameters in  $\theta = \{P_k, \mathbf{m}_k, \Sigma_k, (k = 1, \dots, K)\}$ , and then solve the resulting equations to get the optimal parameters.

– Find  $P_k$ :

As all  $K$  of  $P_k$ 's need to satisfy the constraint  $\sum_{k=1}^K P_k = 1$ , we first construct the Lagrangian function composed of an extra Lagrange multiplier term for the constraint as well as the log likelihood as the objective function:

$$L(\theta, \lambda) = \sum_{n=1}^N \sum_{k=1}^K P_{nk} [\log P_k + \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \Sigma_k)] + \lambda \left( \sum_{k=1}^K P_k - 1 \right) \quad (14.27)$$

and set its derivative with respect to  $P_k$  to zero:

$$\begin{aligned} \frac{\partial}{\partial P_k} L(\theta, \lambda) &= \frac{\partial}{\partial P_k} \left[ \sum_{n=1}^N \sum_{k=1}^K P_{nk} [\log P_k + \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \Sigma_k)] + \lambda \left( \sum_{k=1}^K P_k - 1 \right) \right] \\ &= \sum_{n=1}^N P_{nk} \frac{1}{P_k} + \lambda = 0 \end{aligned} \quad (14.28)$$

Multiplying both sides by  $P_k$ , we get

$$\sum_{n=1}^N P_{nk} + P_k \lambda = N_k + P_k \lambda = 0 \quad (14.29)$$

where we have defined

$$N_k = \sum_{n=1}^N P_{nk} \quad (14.30)$$

that satisfies

$$\sum_{k=1}^K N_k = \sum_{k=1}^K \left( \sum_{n=1}^N P_{nk} \right) = \sum_{n=1}^N \left( \sum_{k=1}^K P_{nk} \right) = \sum_{n=1}^N 1 = N \quad (14.31)$$

Summing Eq. (14.29) over all  $k = 1, \dots, K$ , we get

$$\sum_{k=1}^K (N_k + P_k \lambda) = \sum_{k=1}^K N_k + \lambda \left( \sum_{k=1}^K P_k \right) = N + \lambda = 0 \quad (14.32)$$

Substituting  $\lambda = -N$  back into Eq. (14.29), we get the expression for the prior

$$p(z_k = 1) = P_k = \frac{N_k}{N} = \frac{1}{N} \sum_{n=1}^N P_{nk} \quad (14.33)$$

This is actually the same as the prior  $P_k$  in Eq. (12.8) in naive Bayes classification, but here  $N_k$  defined in Eq. (14.30) is the sum of the probabilities for all  $N$  data samples to belong to  $C_k$ , instead of the number of data samples in  $C_k$  (unknown in this unsupervised case).

– Find  $\mathbf{m}_k$ :

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{m}_k} E_Z(\log L(\theta|\mathbf{X}, \mathbf{Z})) &= \frac{\partial}{\partial \mathbf{m}_k} \left[ \sum_{n=1}^N \sum_{l=1}^K P_{nl} [\log P_l + \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_l, \Sigma_l)] \right] \\
&= \sum_{n=1}^N P_{nk} \frac{\partial}{\partial \mathbf{m}_k} \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \Sigma_k) \\
&\stackrel{*}{=} \sum_{n=1}^N P_{nk} \frac{\partial}{\partial \mathbf{m}_k} \left[ -\frac{1}{2} (\mathbf{x}_n - \mathbf{m}_k)^T \Sigma_k^{-1} (\mathbf{x}_n - \mathbf{m}_k) \right] \\
&= \frac{1}{2} \sum_{n=1}^N P_{nk} \Sigma_k^{-1} (\mathbf{x}_n - \mathbf{m}_k) = \mathbf{0} \tag{14.34}
\end{aligned}$$

Note that here  $\stackrel{*}{=}$  indicates that we have neglected the scaling coefficient  $(2\pi)^{-d/2} |\Sigma_k|^{-1/2}$  of the Gaussian, which is independent of  $\mathbf{m}_k$ . Multiplying  $2\Sigma_k$  on both sides, we get

$$\sum_{n=1}^N P_{nk} (\mathbf{x}_n - \mathbf{m}_k) = \sum_{n=1}^N P_{nk} \mathbf{x}_n - \sum_{n=1}^N P_{nk} \mathbf{m}_k = \sum_{n=1}^N P_{nk} \mathbf{x}_n - N_k \mathbf{m}_k = \mathbf{0} \tag{14.35}$$

Solving for  $\mathbf{m}_k$  we get

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{n=1}^N P_{nk} \mathbf{x}_n \tag{14.36}$$

– Find  $\Sigma_k$ :

$$\begin{aligned}
\frac{\partial}{\partial \Sigma_k} E_Z(\log L(\theta|\mathbf{X}, \mathbf{Z})) &= \frac{\partial}{\partial \Sigma_k} \left[ \sum_{n=1}^N \sum_{l=1}^K P_{nl} [\log P_l + \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_l, \Sigma_l)] \right] \\
&= \sum_{n=1}^N P_{nk} \frac{\partial}{\partial \Sigma_k} \log \mathcal{N}(\mathbf{x}_n, \mathbf{m}_k, \Sigma_k) \\
&= -\frac{1}{2} \sum_{n=1}^N P_{nk} \left[ \frac{\partial}{\partial \Sigma_k} \log |\Sigma_k| + \frac{\partial}{\partial \Sigma_k} (\mathbf{x}_n - \mathbf{m}_k)^T \Sigma_k^{-1} (\mathbf{x}_n - \mathbf{m}_k) \right] \\
&= -\frac{1}{2} \sum_{n=1}^N P_{nk} [\Sigma_k^{-1} - \Sigma_k^{-1} (\mathbf{x}_n - \mathbf{m}_k) (\mathbf{x}_n - \mathbf{m}_k)^T \Sigma_k^{-1}] = \mathbf{0} \tag{14.37}
\end{aligned}$$

Here we have used the following facts (for more details see *Matrix Cookbook* <https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf>)

$$\frac{d}{d\mathbf{A}} \log |\mathbf{A}| = (\mathbf{A}^{-1})^T, \quad \frac{d}{d\mathbf{A}} (\mathbf{a}^T \mathbf{A}^{-1} \mathbf{b}) = -(\mathbf{A}^{-1})^T \mathbf{a} \mathbf{b}^T (\mathbf{A}^{-1})^T \tag{14.38}$$

Pre and post multiplying  $\Sigma_k$  on both sides of the equation above we get

$$\sum_{n=1}^N P_{nk} (\Sigma_k - (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T) = \mathbf{0} \quad (14.39)$$

Solving for  $\Sigma_k$ , we get

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N P_{nk} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T \quad (14.40)$$

We note that all model parameters in  $\theta = \{P_k, \mathbf{m}_k, \Sigma_k, (k = 1, \dots, K)\}$  found respectively in Eqs. (14.33), (14.36), and (14.40) in the M-step depend on  $P_{nk}$  in Eq. (14.23) in the E-step, which in turn is a function of these parameters, i.e., the E-step and M-step need to be carried out in an alternative and iterative fashion from some initial values of the parameters until convergence.

In summary, here is the EM clustering algorithm based on Gaussian mixture model:

1. Initialize means  $\mathbf{m}_k$ , covariance  $\Sigma_k$  and coefficient  $P_k$ .
2. The E-step:

Find the responsibility  $P_{nk}$  for all  $N$  data points and all  $K$  clusters and then  $N_k$ :

$$P_{nk} = P(r_k = 1 | \mathbf{x}_n) = \frac{P_k \mathcal{N}(\mathbf{x}_n; \mathbf{m}_k, \Sigma_k)}{\sum_{l=1}^K P_l \mathcal{N}(\mathbf{x}_n; \mathbf{m}_l, \Sigma_l)}, \quad N_k = \sum_{n=1}^N P_{nk} \quad (14.41)$$

3. The M-step:

Recalculate the parameters that maximize the likelihood function:

$$\begin{aligned} P_k &= \frac{N_k}{N} \\ \mathbf{m}_k &= \frac{1}{N_k} \sum_{n=1}^N P_{nk} \mathbf{x}_n \\ \Sigma_k &= \frac{1}{N_k} \sum_{n=1}^N P_{nk} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T \end{aligned} \quad (14.42)$$

4. If the parameters or the log likelihood function have not converged, go back to step 2. Otherwise terminate the iteration. The probability for each sample  $\mathbf{x}_n$  to belong to cluster  $C_l$  is  $p_{nl} = P(z_{nl} = 1 | \mathbf{x}_n, \theta)$ , and it is therefore assigned to  $C_k$  if  $P_{nk} = \max_l p_{nl}$ .

We can show that the K-means algorithm is actually a special case of the EM algorithm, when all covariance matrices are the same  $\Sigma_k = \varepsilon \mathbf{I}$ , where  $\varepsilon$  is a scaling factor which approaches to zero. In this case we have:

$$p(\mathbf{x} | z_k = 1, \theta) = \mathcal{N}(\mathbf{x} | \mathbf{m}_k, \varepsilon \mathbf{I}) = \frac{1}{(2\pi)^{d/2} \varepsilon^{1/2}} \exp\left(-\frac{1}{2\varepsilon} \|\mathbf{x} - \mathbf{m}_k\|^2\right) \quad (14.43)$$

and the probability for any  $\mathbf{x}_n \in \mathbf{X}$  to belong to cluster  $C_k$  is:

$$P_{nk} = P(z_k = 1 | \mathbf{x}_n, \theta) = \frac{P_k \mathcal{N}(\mathbf{x}_n; \mathbf{m}_k, \Sigma_k)}{\sum_{l=1}^K P_l \mathcal{N}(\mathbf{x}_n; \mathbf{m}_l, \Sigma_l)} = \frac{P_k \exp(-\|\mathbf{x}_n - \mathbf{m}_k\|^2 / 2\varepsilon)}{\sum_{l=1}^K P_l \exp(-\|\mathbf{x}_n - \mathbf{m}_l\|^2 / 2\varepsilon)} \quad (14.44)$$

When  $\varepsilon \rightarrow 0$ , all terms in the denominator approach to zero, but the one with minimum  $\|\mathbf{x} - \mathbf{m}_k\|$  approaches to zero most slowly, and becomes the dominant term of the denominator. If the numerator happens to be this term as well, then  $P_{nk} = 1$ , otherwise the numerator approaches zero and  $P_{nk} = 0$ . Now  $P_{nk}$  defined above becomes:

$$\lim_{\varepsilon \rightarrow 0} P_{nk} = \lim_{\varepsilon \rightarrow 0} \frac{P_k \exp(-\|\mathbf{x}_n - \mathbf{m}_k\|^2 / 2\varepsilon)}{\sum_{l=1}^K P_l \exp(-\|\mathbf{x}_n - \mathbf{m}_l\|^2 / 2\varepsilon)} = \begin{cases} 1 & \text{if } \|\mathbf{x}_n - \mathbf{m}_k\| = \min_l \|\mathbf{x}_n - \mathbf{m}_l\| \\ 0 & \text{otherwise} \end{cases} \quad (14.45)$$

Now the posterior probability  $0 < P_{nk} < 1$  defined in Eq. (14.23) for a soft decision becomes a binary value  $P_{nk} \in \{0, 1\}$  for a hard binary decision to assign  $\mathbf{x}_n$  to  $C_k$  with the smallest distance. Also  $N_k = \sum_{n=1}^N P_{nk}$  defined in Eq. (14.30) as the sum of the posterior probabilities for all  $N$  data points to belong to  $C_k$  becomes  $N_k$  as the number of data samples assigned only to  $C_k$ . In other words, now the probabilistic EM method based on both  $\mathbf{m}$  and  $\Sigma$  becomes the deterministic K-means method based on  $\mathbf{m}$  only.

We can also make a comparison between the GMM method for unsupervised clustering and the softmax regression for supervised classification. First, the latent variables  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$  in GMM play a similar role as the labeling  $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]$  in softmax regression multi-class classification. However, the difference is that  $\mathbf{Y}$  is explicitly given in the training set for a supervised classification, while  $\mathbf{Z}$  is hidden for an unsupervised clustering analysis. Second, we note that the probability  $P_{nk} = p(z_{nk} = 1 | \mathbf{x}_n, \theta)$  given in Eq. (14.23) is similar to the softmax function  $\phi_{nk} = P(y' = k | \mathbf{x}_n)$  in the softmax method in terms of their form, with the only difference that the Gaussian function is used for GMM while the exponential function is used for softmax.

### Examples

The same dataset is used to test both the K-means and EM clustering methods. The first panel shows 10 iterations of the K-means method, while the second panel shows 16 iterations of the EM method. In both cases, the iteration converges to the last plot. Comparing the two clustering results, we see that the K-means method cannot separate the red and green data points from two different clusters, both normally distributed with similar means but very different covariance matrices, while the blue data points all in the same cluster are separated into two clusters. But the EM method based on the Gaussian mixture model can correctly identified all three clusters.

The two clustering methods are also applied to the Iris dataset, which has three classes each of 50 4-dimensional sample vectors. The PCA method is used to visualize the first two principal components, as shown below. Also, as can be



seen from their confusion matrices, the error rate of the K-means method is 18/150, while that of the EM method is 5/150.

K-means			EM		
0	0	50	0	0	50
50	0	0	45	5	0
18	32	0	0	50	0

(14.46)

### 14.3 Mixture of Bernoulli

If the data are binary, i.e., each data point  $x$  is treated as a discrete random variable that takes either of two binary values such as 1 and 0 with probabilities  $\mu$  and  $1 - \mu$ , then the assumption of Gaussian distribution of the dataset is no longer valid and the Gaussian mixture model is not suitable. In this case, the *probability mass function (pmf)* of the Bernoulli distribution can be used instead:

$$\mathcal{B}(x|\mu) = \mu^x(1 - \mu)^{1-x} = \begin{cases} \mu & \text{if } x = 1 \\ 1 - \mu & \text{if } x = 0 \end{cases} \quad (14.47)$$

The mean and variance of  $x$  are

$$E(x) = 1 P(x = 1) + 0 P(x = 0) = 1 \mu + 0 (1 - \mu) = \mu \quad (14.48)$$

$$\begin{aligned} Var(x) &= E[(x - E(x))^2] = E(x^2) - E(x)^2 \\ &= 1^2 P(x = 1) + 0^2 P(x = 0) - \mu^2 = \mu - \mu^2 = \mu(1 - \mu) \end{aligned} \quad (14.49)$$

A set of  $d$  independent binary variables can be represented as a random vector  $\mathbf{x} = [x_1, \dots, x_d]^T$  with mean vector and covariance matrix as shown below:

$$E(\mathbf{x}) = \mathbf{m} = [\mu_1, \dots, \mu_N]^T \quad (14.50)$$

$$Cov(\mathbf{x}) = \mathbf{\Sigma} = diag(\mu_i(1 - \mu_i)) = \begin{bmatrix} \mu_1(1 - \mu_1) & & 0 \\ & \ddots & \\ 0 & & \mu_d(1 - \mu_d) \end{bmatrix} \quad (14.51)$$

Note that the covariance matrix  $\mathbf{\Sigma}$  is solely determined by the means  $\{\mu_1, \dots, \mu_N\}$ .

Now we can get the pmf of the a binary random vector  $\mathbf{x}$ :

$$\mathcal{B}(\mathbf{x}|\mathbf{m}) = \prod_{i=1}^d \mathcal{B}(x_i|\mu_i) = \prod_{i=1}^d \mu_i^{x_i} (1 - \mu_i)^{1-x_i} \quad (14.52)$$

and the log pmf:

$$\log \mathcal{B}(\mathbf{x}|\mathbf{m}) = \log \left( \prod_{i=1}^d \mathcal{B}(x_i|\mu_i) \right) = \sum_{i=1}^d [x_i \log \mu_i + (1 - x_i) \log(1 - \mu_i)] \quad (14.53)$$

Similar to the Gaussian mixture model, the Bernoulli mixture model of  $K$  multivariate Bernoulli distributions is defined as:

$$p(\mathbf{x}|\mathbf{m}_k, P_k, (k = 1, \dots, K)) = p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^K P_k \mathcal{B}(\mathbf{x}, \mathbf{m}_k) = \sum_{k=1}^K P_k \prod_{i=1}^d \mu_{ki}^{x_i} (1 - \mu_{ki})^{1-x_i} \quad (14.54)$$

where  $\boldsymbol{\theta} = \{\mathbf{m}_k, P_k, (k = 1, \dots, K)\}$  denotes all parameters of the mixture model to be estimated based on the given dataset, and  $\mathbf{m}_k = E_k(\mathbf{x})$  respect to  $\mathcal{B}(\mathbf{x}|\mathbf{m}_k)$ . The mean of this mixture model is

$$\mathbf{m} = E(\mathbf{x}) = \sum_{k=1}^K P_k E_k(\mathbf{x}) = \sum_{k=1}^K P_k \mathbf{m}_k \quad (14.55)$$

Also similar to the Gaussian mixture model, we introduce a set of  $K$  latent binary random variables  $\mathbf{z} = [z_1, \dots, z_K]^T$  with binary components  $z_k \in \{0, 1\}$  and  $\sum_{k=1}^K z_k = 1$ , and get the prior probability of  $\mathbf{z}$ , the conditional probability of  $\mathbf{x}$  given  $\mathbf{z}$ , and the joint probability of  $\mathbf{x}$  and  $\mathbf{z}$  as the following

$$p(\mathbf{z}|\boldsymbol{\theta}) = \prod_{k=1}^K P_k^{z_k} \quad (14.56)$$

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \prod_{k=1}^K \mathcal{B}(\mathbf{x}, \mathbf{m}_k)^{z_k} \quad (14.57)$$

$$p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = p(\mathbf{z}|\boldsymbol{\theta}) p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \prod_{k=1}^K (P_k \mathcal{B}(\mathbf{x}, \mathbf{m}_k))^{z_k} \quad (14.58)$$

Given the dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  containing  $N$  i.i.d. samples, we introduce the corresponding latent variables in  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$ , of which each  $\mathbf{z}_n = [z_{n1}, \dots, z_{nK}]^T$  is for the labeling of  $\mathbf{x}_n$ . Then we can find the likelihood function of the Bernoulli mixture model parameters  $\boldsymbol{\theta} = \{P_k, \mathbf{m}_k, (k = 1, \dots, K)\}$ :

$$\begin{aligned} L(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Z}) &= p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = p([\mathbf{x}_1, \dots, \mathbf{x}_N], [\mathbf{z}_1, \dots, \mathbf{z}_N]|\mathbf{m}_k, P_k (k = 1, \dots, K)) \\ &= \prod_{n=1}^N p(\mathbf{x}_n, \mathbf{z}_n|\boldsymbol{\theta}) = \prod_{n=1}^N \prod_{k=1}^K (P_k \mathcal{B}(\mathbf{x}_n, \mathbf{m}_k))^{z_{nk}} \end{aligned} \quad (14.59)$$

and the log likelihood function:

$$\begin{aligned} \log L(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Z}) &= \log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = \log \prod_{n=1}^N \prod_{k=1}^K (P_k \mathcal{B}(\mathbf{x}_n, \mathbf{m}_k))^{z_{nk}} \\ &= \sum_{n=1}^N \sum_{k=1}^K z_{nk} [\log P_k + \log \mathcal{B}(\mathbf{x}_n, \mathbf{m}_k)] \end{aligned} \quad (14.60)$$

Based on the same EM method used in Gaussian mixture model, we can find the optimal parameters that maximize the expectation of the log likelihood function in the following two steps:

- **E-step:** Find the expectation of the likelihood function.

We first find the posterior probability for any sample  $\mathbf{x}_n$  to belong to cluster  $C_k$ , denoted by  $P_{nk}$ :

$$P_{nk} = P(z_{nk} = 1 | \mathbf{x}_n, \boldsymbol{\theta}) = \frac{p(\mathbf{x}_n, z_{nk} = 1 | \boldsymbol{\theta})}{p(\mathbf{x}_n | \boldsymbol{\theta})} = \frac{P_k \mathcal{B}(\mathbf{x}_n; \mathbf{m}_k)}{\sum_{l=1}^K P_l \mathcal{B}(\mathbf{x}_n; \mathbf{m}_l)} \quad (n = 1, \dots, N; \quad k = 1, \dots, K) \quad (14.61)$$

which is the expectation of  $z_{nk}$ :

$$E(z_{nk}) = 1 P(z_{nk} = 1 | \mathbf{x}_n) + 0 P(z_{nk} = 0 | \mathbf{x}_n) = P(z_{nk} = 1 | \mathbf{x}_n) = P_{nk} \quad (14.62)$$

Now we can find the expectation of the log likelihood with respect to the latent variables in  $\mathbf{Z}$ :

$$\begin{aligned} E_{\mathbf{Z}}(\log L(\boldsymbol{\theta} | \mathbf{X}, \mathbf{Z})) &= E_{\mathbf{Z}} \sum_{n=1}^N \sum_{k=1}^K z_{nk} [\log P_k + \log \mathcal{B}(\mathbf{x}_n, \mathbf{m}_k)] \\ &= \sum_{n=1}^N \sum_{k=1}^K E(z_{nk}) \left[ \log P_k + \log \prod_{i=1}^d \mu_{ki}^{x_{ni}} (1 - \mu_{ki})^{1-x_{ni}} \right] \\ &= \sum_{n=1}^N \sum_{k=1}^K P_{nk} \left[ \log P_k + \sum_{i=1}^d [x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ki})] \right] \end{aligned} \quad (14.63)$$

- **M-step:** Find the optimal model parameters that maximize the expectation of the log likelihood function.

We first set to zero the derivatives of the expectation of the log likelihood with respect to each of the parameters in  $\boldsymbol{\theta} = \{P_k, \mathbf{m}_k \ (k = 1, \dots, K)\}$ , and then solve the resulting equations to get the optimal parameters.

– Find  $P_k$ : same as in the case of the GMM model:

$$P_k = \frac{N_k}{N} = \frac{1}{N} \sum_{n=1}^N P_{nk} \quad (14.64)$$

– Find  $\mathbf{m}_k$ :

$$\begin{aligned} &\frac{\partial}{\partial \mathbf{m}_k} E_{\mathbf{Z}}(\log p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\theta})) \\ &= \frac{\partial}{\partial \mathbf{m}_k} \sum_{n=1}^N \sum_{k=1}^K P_{nk} \left[ \log P_k + \sum_{i=1}^d [x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ki})] \right] \\ &= \sum_{n=1}^N P_{nk} \frac{\partial}{\partial \mathbf{m}_k} \sum_{i=1}^d [x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ki})] = \mathbf{0} \quad (14.65) \end{aligned}$$

The  $i$ th component of the equation is

$$\begin{aligned} & \sum_{n=1}^N P_{nk} \frac{d}{d\mu_{ki}} [x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ki})] \\ &= \sum_{n=1}^N P_{nk} \left( \frac{x_{ni}}{\mu_{ki}} - \frac{1 - x_{ni}}{1 - \mu_{ki}} \right) = 0 \end{aligned} \quad (14.66)$$

i.e.,

$$(1 - \mu_{ki}) \sum_{n=1}^N P_{nk} x_{ni} = \mu_{ki} \sum_{n=1}^N P_{nk} (1 - x_{ni}) = \mu_{ki} N_k - \mu_{ki} \sum_{n=1}^N P_{nk} x_{ni} \quad (14.67)$$

Solving for  $\mu_{ki}$  we get

$$\mu_{ki} = \frac{1}{N_k} \sum_{n=1}^N P_{nk} x_{ni} \quad (i = 1, \dots, d) \quad (14.68)$$

or, in vector form,

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{n=1}^N P_{nk} \mathbf{x}_n \quad (14.69)$$

**Example:**

Clustering results of hand-written digits with  $K = 10$  and  $K = 12$ . The mean vectors  $\mathbf{m}_k$  of each of the  $K$  clusters are visualized as shown:

## 14.4 General EM Algorithm

The EM method for the Gaussian mixture model can be generalized to estimate the parameters in  $\boldsymbol{\theta} = \{\theta_1, \dots, \theta_m\}$  of a generic probabilistic model  $p(\mathbf{x}, z|\boldsymbol{\theta})$ , where  $z$  is a hidden variable that takes any of the  $K$  values  $\{z_1, \dots, z_K\}$  with the corresponding probabilities  $P_1, \dots, P_K$ , satisfying  $\sum_{k=1}^K P_k = 1$ .

By marginalizing the hidden variable  $z$  we get (similar to how we get Eq. (14.16)):

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^K p(\mathbf{x}, z_k|\boldsymbol{\theta}) = \sum_{k=1}^K P_k p(\mathbf{x}|z_k, \boldsymbol{\theta}) \quad (14.70)$$

Based on the training set  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  containing  $N$  i.i.d. samples, we can further get the likelihood function of the parameter  $\boldsymbol{\theta}$  (similar to Eq. (14.21)):

$$\begin{aligned} L(\boldsymbol{\theta}|\mathbf{X}) &= \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\theta}) = \prod_{n=1}^N \sum_{k=1}^K P_k p(\mathbf{x}_n, z_k|\boldsymbol{\theta}) = \prod_{n=1}^N \sum_{k=1}^K P_k \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{P_k} \\ &= \prod_{n=1}^N E_z \left[ \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{P_k} \right] \end{aligned} \quad (14.71)$$

and the log likelihood

$$\begin{aligned} l(\boldsymbol{\theta}|\mathbf{X}) &= \log L(\boldsymbol{\theta}|\mathbf{X}) = \log \prod_{n=1}^N E_z \left[ \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{P_k} \right] \\ &= \sum_{n=1}^N \log E_z \left[ \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{P_k} \right] \geq \sum_{n=1}^N E_z \log \left[ \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{P_k} \right] \end{aligned} \quad (14.72)$$

The last step is due to Jensen's inequality (Section A.25) for the concave logarithmic function  $f(x) = \log x$ .

The final expression is a lower bound of the log likelihood function, which becomes tight if the equality holds, if the expression inside the brackets is not a random variable, but some constant  $c$ , i.e.,  $P_k = c p(\mathbf{x}_n, z_k|\boldsymbol{\theta})$  for all  $k = 1, \dots, K$ . This  $c$  can be found by

$$\sum_{k=1}^K P_k = 1 = c \sum_{k=1}^K p(\mathbf{x}_n, z_k|\boldsymbol{\theta}) \quad \text{i.e.,} \quad c = \frac{1}{\sum_{k=1}^K p(\mathbf{x}_n, z_k|\boldsymbol{\theta})} \quad (14.73)$$

Now we have

$$P_k = c p(\mathbf{x}_n, z_k|\boldsymbol{\theta}) = \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{\sum_{k=1}^K p(\mathbf{x}_n, z_k|\boldsymbol{\theta})} = \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{p(\mathbf{x}_n|\boldsymbol{\theta})} = p(z_k|\mathbf{x}_n, \boldsymbol{\theta}) \quad (14.74)$$

We see that if we set  $P_k$  to be the same as the posterior probability  $p(z_k|\mathbf{x}_n, \boldsymbol{\theta})$  of  $z_k$ , then the right hand side of Eq. (14.72) is the same as the log likelihood  $l(\boldsymbol{\theta}|\mathbf{X})$ . The optimal parameters in  $\boldsymbol{\theta}$  that maximize this log likelihood can be found by iterating the following two steps, based on some initial guess of  $\boldsymbol{\theta}$ :

- The E-step:

$$P_k = p(z_k|\mathbf{x}_n, \boldsymbol{\theta}) = \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{p(\mathbf{x}_n|\boldsymbol{\theta})} = \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{\sum_{k=1}^K p(\mathbf{x}_n, z_k|\boldsymbol{\theta})} \quad (14.75)$$

We note that this is the same as Eq. (14.23) for the specific Gaussian mixture model.

- The M-step:

$$\boldsymbol{\theta} = \arg \max_{\boldsymbol{\theta}} \sum_{n=1}^N E_z \log \left[ \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{P_k} \right] = \arg \max_{\boldsymbol{\theta}} \sum_{n=1}^N \sum_{k=1}^K P_k \log \left[ \frac{p(\mathbf{x}_n, z_k|\boldsymbol{\theta})}{P_k} \right] \quad (14.76)$$

As  $P_k$  in the E-step depends on  $\boldsymbol{\theta}$ , which can be found in the M-step depending on  $P_k$ , these two steps need to be carried out alternatively and iteratively until the final convergence of  $\boldsymbol{\theta}$ .

## 14.5 Linear Models for Binary Classification

We first consider binary classification based on the same linear model  $y = f(\mathbf{x}) + r = \mathbf{x}^T \mathbf{w} + r$  used in linear regression considered before. Any test sample  $\mathbf{x}$  is

classified into one of the two classes  $\{C_0, C_1\}$  depending on whether  $f(\mathbf{x})$  is greater or smaller than zero:

$$\text{if } f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \begin{cases} < 0 \\ > 0 \end{cases}, \quad \text{then } \mathbf{x} \in \begin{cases} C_0 \\ C_1 \end{cases} \quad (14.77)$$

Here  $\mathbf{w}$  is a model parameter to be determined based on the training set  $\mathcal{D} = \{(\mathbf{x}_n, y_n) | n = 1, \dots, N\} = \{\mathbf{X}, \mathbf{y}\}$ , where  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ ,  $\mathbf{y} = [y_1, \dots, y_N]^T$ ,  $\mathbf{x}_n \in C_n$  if  $y_n = 1$  and  $\mathbf{x}_i \in C_0$  if  $y_i = -1$ , so that  $\mathbf{w}$  fits all data points optimally in certain sense.

While in the previously considered least square classification method, we find the optimal  $\mathbf{w}$  that minimizes the squared error  $\varepsilon(\mathbf{w}) = \|\mathbf{r}\|^2$ , here we find the optimal  $\mathbf{w}$  based on a probabilistic model. Specifically, we now convert the linear function  $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$  into the probability for  $\mathbf{x}$  to belong to either class:

$$\begin{aligned} p(\mathbf{x} \in C_1 | \mathbf{w}) &= p(y = 1 | \mathbf{x}, \mathbf{w}) \\ p(\mathbf{x} \in C_0 | \mathbf{w}) &= p(y = -1 | \mathbf{x}, \mathbf{w}) = 1 - p(y = 1 | \mathbf{x}, \mathbf{w}) \end{aligned} \quad (14.78)$$

This is done by a sigmoid function defined as either a logistic or cumulative Gaussian (error function, erf):

$$\begin{aligned} \text{logistic: } \sigma(z) &= \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z} \\ \text{erf: } \phi(z) &= \int_{-\infty}^z \mathcal{N}(u|0, 1) du = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^z \exp\left(-\frac{1}{2}u^2\right) du \end{aligned} \quad (14.79)$$

In either case, we have

$$\sigma(z) = \begin{cases} 0 & z = -\infty \\ 1 & z = \infty \end{cases} \quad \text{and} \quad \sigma(-z) = 1 - \sigma(z) \quad (14.80)$$

Using this sigmoid function,  $f = \mathbf{x}^T \mathbf{w}$  in the range of  $(-\infty, \infty)$  is mapped to  $\sigma(f(\mathbf{x}))$  in the range of  $(0, 1)$ , which can be used as the probability for  $\mathbf{x}$  to belong to either class:

$$\begin{aligned} p(y = 1 | \mathbf{x}, \mathbf{w}) &= \sigma(\mathbf{x}^T \mathbf{w}) = \sigma(f) \\ p(y = -1 | \mathbf{x}, \mathbf{w}) &= 1 - p(y = 1 | \mathbf{x}, \mathbf{w}) = 1 - \sigma(f) = \sigma(-f) \end{aligned} \quad (14.81)$$

The two cases can be combined:

$$p(y | \mathbf{x}, \mathbf{w}) = \sigma(y \mathbf{x}^T \mathbf{w}) = \sigma(yf), \quad \text{or} \quad p(y | \mathbf{x}, \mathbf{w}) = \phi(y \mathbf{x}^T \mathbf{w}) = \phi(yf) \quad (14.82)$$

The binary classification problem can now be treated as a regression problem to find the model parameter  $\mathbf{w}$  that best fits the data in the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ . Such a regression problem is called *logistic regression* if  $\sigma(z)$  is used, or *probit regression* if  $\phi(z)$  is used.

Same as in the case of Bayesian regression, we assume the prior distribution of  $\mathbf{w}$  to be a zero-mean Gaussian  $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \Sigma_w)$ , and for simplicity we further

assume  $\Sigma_w = \mathbf{I}$ , and find the likelihood of  $\mathbf{w}$  based on the linear model applied to the observed data set  $\mathcal{D} = \{(\mathbf{x}_n, y_n) | n = 1, \dots, N\}$ :

$$\mathcal{L}(\mathbf{w}|\mathcal{D}) = p(\mathcal{D}|\mathbf{w}) = p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^N p(y_n|\mathbf{x}_n, \mathbf{w}) = \prod_{n=1}^N \sigma(y_n f_n) = \prod_{n=1}^N \sigma(y_n \mathbf{x}_n^T \mathbf{w}) \quad (14.83)$$

Note that this likelihood is not Gaussian (as in the case of Eq. (4.63)), because  $y = \pm 1$  is the binary class labeling of the training samples in  $\mathbf{X}$ , instead of a continuous function as in the case of regression.

The posterior of  $\mathbf{w}$  can now be expressed in terms of the prior  $p(\mathbf{w})$  and the likelihood  $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$ :

$$\begin{aligned} p(\mathbf{w}|\mathcal{D}) &= p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}, \mathbf{w}|\mathbf{X})}{p(\mathbf{y})} = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})} \propto p(\mathbf{y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w}|\mathbf{X}) \\ &= \prod_{n=1}^N p(y_n|\mathbf{x}_n, \mathbf{w}) \mathcal{N}(\mathbf{0}, \Sigma_w) = \prod_{n=1}^N \sigma(y_n f_n) \frac{1}{(2\pi)^{d/2} |\Sigma_w|^{1/2}} \exp\left(-\frac{1}{2} \mathbf{w}^T \Sigma_w^{-1} \mathbf{w}\right) \end{aligned} \quad (14.84)$$

where the denominator  $p(\mathbf{y}|\mathbf{X}) = p(\mathcal{D})$  is dropped as it is a constant independent of the variable  $\mathbf{w}$  of interest. We can further find the log posterior denoted by  $\psi(\mathbf{w})$ :

$$\psi(\mathbf{w}) = \log p(\mathbf{w}|\mathcal{D}) = \sum_{n=1}^N \log \sigma(y_n f_n) - \frac{N}{2} \log(2\pi) - \frac{1}{2} \log |\Sigma_w| - \frac{1}{2} \mathbf{w}^T \Sigma_w^{-1} \mathbf{w} \quad (14.85)$$

The optimal  $\mathbf{w}$  that best fits the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$  can now be found as the one that maximizes this posterior  $p(\mathbf{w}|\mathcal{D})$ , or, equivalently, the log posterior  $\psi(\mathbf{w})$ , by setting the derivative of  $\psi(\mathbf{w})$  to zero and solving the resulting equation below by Newton's method or conjugate gradient ascent method:

$$\begin{aligned} \mathbf{g}_\psi(\mathbf{w}) &= \frac{d}{d\mathbf{w}} \left( \sum_{n=1}^N \log \sigma(y_n f_n) \right) - \frac{d}{d\mathbf{w}} \left( \frac{1}{2} \mathbf{w}^T \Sigma_w^{-1} \mathbf{w} \right) \\ &= \sum_{n=1}^N \frac{d}{d\mathbf{w}} \left( \log \frac{1}{1 + e^{-y_n \mathbf{x}_n^T \mathbf{w}}} \right) - \Sigma_w^{-1} \mathbf{w} \\ &= \sum_{n=1}^N \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{x}_n^T \mathbf{w}}} - \Sigma_w^{-1} \mathbf{w} = \mathbf{0} \end{aligned} \quad (14.86)$$

Having found the optimal  $\mathbf{w}$ , we can classify any test pattern  $\mathbf{x}_*$  in terms of the posterior of its corresponding labeling  $y_*$ :

$$p(y_* = 1 | \mathbf{x}_*, \mathbf{w}) = \sigma(\mathbf{x}_*^T \mathbf{w}) = \sigma(f(\mathbf{x}_*)) = \frac{1}{1 + \exp(\mathbf{x}_*^T \mathbf{w})} \quad (14.87)$$

In a multi-class case with  $(C_1, \dots, C_K)$ , we can still use a vector  $\mathbf{w}_i$  ( $i = 1, \dots, K$ ) to represent each class  $C_i$ , the direction of the class with respect to the origin in the feature space, and the inner product  $\mathbf{x}^T \mathbf{w}_i$  proportional to

the projection of  $\mathbf{x}$  onto vector  $\mathbf{w}_i$  measures the extent to which  $\mathbf{x}$  belongs to  $C_i$ . Similar to the logistic function used in the two-class case, here the *softmax function* defined below is used to convert  $-\infty < \mathbf{x}^T \mathbf{w}_i < \infty$  ( $i = 1, \dots, K$ ) into the probability that  $\mathbf{x}$  belongs to  $C_i$ :

$$p(y \in C_i | \mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{x}^T \mathbf{w}_i)}{\sum_{k=1}^K \exp(\mathbf{x}^T \mathbf{w}_k)} = \begin{cases} 0 & \mathbf{x}^T \mathbf{w}_i = -\infty \\ 1 & \mathbf{x}^T \mathbf{w}_i = \infty \end{cases} \quad (14.88)$$

where  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K]$ .



# Part V

---

## Neural Networks



# 15 Biologically Inspired Networks

---

## 15.1 Biological Inspiration

In machine learning, artificial neural network (ANN) is a category of algorithms inspired by biological neural networks in the brain, and designed to carry out both supervised and unsupervised learning tasks, such as classification and clustering. To understand how such a neural network algorithm works, we first need to consider some basic concepts in biological neural system.

The human brain consists of  $10^{11}$  *neurons* interconnected through about  $10^{14}$  to  $10^{15}$  *synaptic junctions* to form millions of neural networks. Hundreds specialized cortical areas are formed based on these networks for different information processing tasks.

Functionally, a neuron consists of the following three parts:

- The *cell body or soma* containing the nucleus of the cell;
- The *dendrites* that receive electrochemical stimulations (input impulses) from other neurons and propagate them to the cell body;
- The *axon*: that conducts the impulses (output) away from the cell body to other cells;
- The *synapse* is the point at which impulses pass from one cell to another.

The function of a neuron can be modeled mathematically. At each neuron, modeled as a *node* in the neural network, receives input signals or stimuli  $x_1, \dots, x_d$  from  $d$  other neurons and becomes activated to different extents measured by its *activation* or net input, the weighted sum of the inputs:

$$a = \sum_{j=1}^d w_j x_j + b = \sum_{j=0}^d w_j x_j = \mathbf{w}^T \mathbf{x} \quad (15.1)$$

where  $b$  is the offset or bias, and  $w_j$  is a weight representing the synaptic connectivity to the  $j$ th input node:

$$w_j \begin{cases} > 0 & \text{excitatory input} \\ < 0 & \text{inhibitory input} \\ = 0 & \text{no connection} \end{cases} \quad (15.2)$$

Also, same as in the case of linear regression, here we have defined  $x_0 = 1$  and  $w_0 = b$ , so that both the weight and pattern vectors are augmented to become

$\mathbf{x} = [x_0 = 1, x_1, \dots, x_d]^T$  and  $\mathbf{w} = [w_0 = b, w_1, \dots, w_d]^T$ . The output or response of the neuron is a function of its activation:

$$y = g(a) = g\left(\sum_{j=1}^d w_j x_j + b\right) = g(\mathbf{w}^T \mathbf{x}) \quad (15.3)$$

Here  $g(x)$ , the *activation function* typically takes one of the following forms:

- Liner function:

$$g(x) = x, \quad \frac{dg(x)}{dx} = 1 \quad (15.4)$$

- Logistic sigmoid function:

$$g(x, a) = \frac{1}{1 + e^{-ax}} = \frac{e^{ax}}{1 + e^{ax}} = \begin{cases} 0 & x = -\infty \\ 1/2 & x = 0 \\ 1 & x = \infty \end{cases}, \quad \frac{dg(x)}{dx} = \frac{a e^{-ax}}{(1 + e^{-ax})^2} \quad (15.5)$$

where parameter  $a$  controls the slope of the transition from 0 to 1 around  $x = 0$ . Specially

$$g(x, 0) = \frac{1}{2}, \quad \frac{dg(x)}{dx} = 0 \quad (15.6)$$

$$g(x, \infty) = u(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}, \quad \frac{dg(x)}{dx} = \begin{cases} 0 & x \neq 0 \\ \infty & x = 0 \end{cases} \quad (15.7)$$

- Tanh (hyperbolic tangent) function:

$$g(x, a) = \frac{2}{1 + e^{-ax}} - 1 = \frac{e^{ax} - 1}{e^{ax} + 1} = \begin{cases} -1 & x = -\infty \\ 0 & x = 0 \\ 1 & x = \infty \end{cases}, \quad \frac{dg(x)}{dx} = \frac{2a e^{-ax}}{(1 + e^{-ax})^2} \quad (15.8)$$

- Rectified linear unit (ReLU):

$$g(x) = \max(0, x) = \begin{cases} 0 & x < 0 \\ x & x > 0 \end{cases}, \quad \frac{dg(x)}{dx} = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases} \quad (15.9)$$

The soft version of ReLU is the softplus function of which the derivative is the logistic function:

$$g(x) = \log(1 + e^x), \quad \frac{dg(x)}{dx} = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}} \quad (15.10)$$

The function of a neural network can be modeled mathematically as a hierarchical structure shown below containing multiple layers of neurons, called *nodes* in the context of artificial neural networks:

- The *input layer*: receives inputs from external sources;
- The *output layer*: generates output to the external world;
- The *hidden layer(s)*: between of the input and output layers, not visible from outside the network.

The purpose is to train the network according to certain mathematical rules, the *learning rules* or *learning laws*, by modifying the weights of a network iteratively based on the inputs (and the desired outputs if the learning is supervised), so that given the input of the network as the stimulus, the network will produce the desired output as the response.

In general, the method of ANN can be used for both supervised and unsupervised learning. For supervised learning, the goal is to model the relationship between certain dependent variable  $\mathbf{y}$  and independent variable  $\mathbf{x}$  by a function  $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$ , of which the parameter  $\mathbf{w}$  for all weights in the network is to be estimated based on the training set  $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ . Such a general framework is suitable for both regression and classification. Neural networks can also be used for unsupervised learning based on observed data  $\mathbf{X}$  without labeling.

Some learning paradigms of the neural networks are listed below, depending on the interpretations of the input and output of the neural network.

#### 1. *Pattern Associator*

This is the most general form of neural networks that learns and stores the associative relationship between two sets of patterns represented by vectors.

- Training: A set of  $N$  pairs of patterns  $\{(\mathbf{x}_n, \mathbf{y}_n), n = 1, \dots, N\}$  is presented to the network which then learns to establish the associative relationship between two sets of patterns:

$$f : \mathbf{x} \in \mathcal{R}^d \implies \mathbf{y} \in \mathcal{R}^m \quad (15.11)$$

- Testing: When a pattern  $\mathbf{x}_n$  in a pair is presented as the input, the network produces an output pattern  $\mathbf{y}_n$  associated to the output.

Human memory is associative in the sense that given one pattern, some associated pattern(s) may be produced. Examples include: (Evolution, Darwin), (Einstein,  $E = mc^2$ ), (food, sounding bell, salivation).

#### 2. *Auto-associator*

As a special pattern associator, auto-associator associates a prestored pattern to an incomplete or noisy version of the pattern.

- Training: A set of patterns  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  is presented to the network for it to learn and remember, i.e., the patterns are stored in the network.
- Testing: When an incomplete or noisy version of one of the patterns stored in the network is presented as the input to the network, the original pattern is retrieved by the network as the output.

#### 3. *Regression*

This is another special kind of pattern associator which takes a vector input  $\mathbf{x} \in \mathcal{R}^d$  and produces a real value  $y \in \mathcal{R}$  as a multivariable function  $y = f(\mathbf{x})$  at its only output node.

- Training: trained by a set of observed data samples, the independent vectors and their corresponding function values  $\{(\mathbf{x}_n, y_n), n = 1, \dots, N\}$ , the network is model the function.

- Testing: given any vector  $\mathbf{x}$ , the output value produced by the single output node is an estimated function value  $y = f(\mathbf{x})$ .
4. *Classification*
- This is a variation of the pattern associator of which the output patterns are a set of categorical symbols representing different classes  $\{C_1, \dots, C_K\}$ , i.e., each input pattern is classified by the network into one of the classes

$$f : \mathbf{x} \in \mathcal{R}^d \implies y \in \{C_1, \dots, C_K\} \quad (15.12)$$

5. *Regularity Detector*

This is an unsupervised learning process. The network discovers automatically the regularity in the inputs so that similar patterns are automatically detected and grouped together in the same cluster or class.

## 15.2 Hebbian Learning

Donald Hebb (1949) speculated that “When neuron A repeatedly and persistently takes part in exciting neuron B, the synaptic connection from A to B will be strengthened.” In other words, simultaneous activation of neurons leads to pronounced increases in synaptic strength between them, or “neurons that fire together wire together; neurons that fire out of sync, fail to link”.

For example, the well known *classical conditioning* (Pavlov, 1927) could be explained by Hebbian learning. Consider the following three patterns:

- Unconditioned stimulus: sight of food F
- Conditioned stimulus: sound of bell B
- Response: salivation S

The unconditioned response is:  $F \rightarrow S$ . Due to the repeated and persistent conditioning process  $F \cap B \rightarrow S$ , the synaptic connections between patterns B and S are strengthened as both are repeatedly excited simultaneously, i.e., the two patterns become associated, resulting the conditioned response  $B \rightarrow S$ .

Based on this theory of Hebbian learning, the Hebbian network can be considered as a supervised learning method that learns to establish the associative relationship between any pair of two patterns  $\mathbf{x}_n$  and  $\mathbf{y}_n$  in the given dataset  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  and  $\{\mathbf{y}_1, \dots, \mathbf{y}_N\}$ , considered as the training set. This is a 2-layer network with  $d$  nodes in the input layer to receive an input pattern  $\mathbf{x} = [x_1, \dots, x_d]^T$  and  $m$  nodes in the output layer to produce an output  $\mathbf{y} = [y_1, \dots, y_m]^T$ . Each output node is fully connected to all  $d$  input nodes through its weights:

$$y_i = \sum_{j=1}^d w_{ij} x_j = \mathbf{w}_i^T \mathbf{x} \quad (i = 1, \dots, m) \quad (15.13)$$

where  $\mathbf{w}_i = [w_{i1}, \dots, w_{id}]^T$ , or in matrix form

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_m^T \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} = \mathbf{W}\mathbf{x} \quad (15.14)$$

where  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_m]^T$  is an  $m \times d$  matrix.

The Hebbian learning rule is inspired by Hebb's theory, i.e., when both neurons  $x_j$  and  $y_i$  are activated, the synaptic connectivity, here the weight  $w_{ij}$ , between them is enhanced:

$$w_{ij}^{new} = w_{ij}^{old} + \eta x_j y_i \quad (i = 1, \dots, m, j = 1, \dots, d) \quad (15.15)$$

or in matrix form:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \eta \mathbf{y}\mathbf{x}^T \quad (15.16)$$

Here  $\eta$  is the *learning rate*, a parameter that controls how quickly the weights get modified.

As in all supervised learning, the Hebbian network is first trained and then used for association.

- **Training:**

For simplicity, we assume all weights are initialized to zero  $w_{ij} = 0$  ( $i = 1, \dots, m, j = 1, \dots, d$ ), and then train the network to find all weights based on all  $N$  pattern pairs in the dataset  $\{(\mathbf{x}_n, \mathbf{y}_n), n = 1, \dots, N\}$  based on the learning law:

$$w_{ij} = \sum_{n=1}^N x_j^{(n)} y_i^{(n)} \quad (i = 1, \dots, m, j = 1, \dots, d) \quad (15.17)$$

or in matrix form, the weight matrix is the sum of the outer-products of all  $N$  pairs of patterns:

$$\mathbf{W}_{m \times d} = \sum_{n=1}^N \mathbf{y}_n \mathbf{x}_n^T = \sum_{n=1}^N \begin{bmatrix} y_1^{(n)} \\ \vdots \\ y_m^{(n)} \end{bmatrix} [x_1^{(n)}, \dots, x_d^{(n)}] \quad (15.18)$$

- **Association:**

When one of the patterns  $\mathbf{x}_l$  is presented to the network, it produces an output:

$$\mathbf{y} = \mathbf{W}\mathbf{x}_l \quad (15.19)$$

which, as shown below, is the same as the associated pattern  $\mathbf{y}_l$ , under the following conditions:

- The  $N$  patterns  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  are treated as zero-mean random vectors,

and they are assumed to be completely uncorrelated with zero *correlation coefficient* between any  $\mathbf{x}_i$  and  $\mathbf{x}_j$ :

$$r_{ij} = \frac{\sigma_{ij}^2}{\sigma_i \sigma_j} = \frac{\sum_{k=1}^d x_{ki} x_{kj}}{\sqrt{\sum_{k=1}^d x_{ki}^2} \sqrt{\sum_{k=1}^d x_{kj}^2}} = \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|} = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad (15.20)$$

If we further assume all patterns are normalized with  $\|\mathbf{x}_n\| = 1$ , then we have  $\mathbf{x}_i^T \mathbf{x}_j = \delta_{ij}$

- The number of input nodes is greater than the number of pattern pairs:  $d \geq N$ , i.e., the capacity of the network is large enough for representing  $N$  different patterns (as there can be no more than  $d$  orthogonal vectors in a  $d$ -dimensional space).

Under these conditions, the output of the network as its response to input  $\mathbf{x}_l$  is

$$\mathbf{y} = \mathbf{W} \mathbf{x}_l = \left( \sum_{n=1}^N \mathbf{y}_n \mathbf{x}_n^T \right) \mathbf{x}_l = \mathbf{y}_l (\mathbf{x}_l^T \mathbf{x}_l) + \sum_{n \neq l} \mathbf{y}_n (\mathbf{x}_n^T \mathbf{x}_l) = \mathbf{y}_l \quad (15.21)$$

as  $\mathbf{x}_n^T \mathbf{x}_l = 0$  for all other terms with  $n \neq l$ . We see that a one-to-one correspondence relationship between  $\mathbf{x}_n$  and  $\mathbf{y}_n$  has been established for all  $n = 1, \dots, N$ . In non-ideal cases when the conditions above are not fully satisfied, the summation term is non-zero and there is an error  $\mathbf{y} - \mathbf{y}_l \neq 0$ .

### 15.3 Hopfield Network

The Hopfield network is a supervised method, based on the Hebbian learning rule. As a supervised method, the Hopfield network is trained based on a dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_K]$  containing  $K$  data samples each composed of  $d$  binary components  $\mathbf{x} = [x_1, \dots, x_d]^T$  with  $x_i \in \{-1, 1\}$ , ( $i = 1, \dots, d$ ), representing one of  $K$  different of patterns of interest. Once the network is completely trained, the weight matrix  $\mathbf{W}$  of the network is obtained in which the  $K$  patterns are stored. Given an input  $\mathbf{x}$ , an iterative computation is carried out until convergence, when one of the pre-stored complete pattern that most closely resemble the input  $\mathbf{x}$  is produced as the output. The Hopfield network can therefore be considered as an auto-associator (a content addressable memory), by which a pre-stored pattern can be retrieved based on its association to the input pattern established by the network.

Structurally, the Hopfield network is a *recurrent* network, in the sense that the outputs of its single layer of  $d$  nodes are fed back to these nodes in an iterative process.

- **Training:**

The training process is essentially the same as the Hebbian learning,



except here the two associated patterns in each pair are the same (self-association). The weight matrix of the network is obtained as the sum of the outer-products of the  $K$  patterns to be stored:

$$\mathbf{W}_{d \times d} = \frac{1}{d} \sum_{k=1}^K \mathbf{x}_k \mathbf{x}_k^T = \frac{1}{d} \sum_{k=1}^K \begin{bmatrix} x_1^{(k)} \\ \vdots \\ x_d^{(k)} \end{bmatrix} [x_1^{(k)}, \dots, x_d^{(k)}] \quad (15.22)$$

The weight connecting node  $i$  and node  $j$  is defined as

$$w_{ij} = \frac{1}{d} \sum_{k=1}^K x_i^{(k)} x_j^{(k)} = w_{ji} \quad (15.23)$$

We assume no self-connection exists  $w_{ii} = 0$  ( $i = 1, \dots, d$ )

- **Autoassociation:**

Once the weight matrix  $\mathbf{W}$  is obtained by the training process, the network can be used as a self-associator. When an input pattern  $\mathbf{x}$  is presented to the network, the outputs of the network are updated *iteratively* and *asynchronously*, one randomly selected node at a time:

$$x_i^{(n+1)} = \text{sgn} \left( \sum_{j=1}^d w_{ij} x_j^{(n)} \right) = \begin{cases} +1, & \text{if } \sum_{j=1}^d w_{ij} x_j^{(n)} \geq 0 \\ -1, & \text{if } \sum_{j=1}^d w_{ij} x_j^{(n)} < 0 \end{cases} \quad (15.24)$$

where  $x_i^{(n)}$  and  $x_i^{(n+1)}$  are the output  $x_i$  of the  $i$ th node before and after the  $n$ th iteration, respectively. As shown below, this iteration will always converge to one of the  $K$  pre-stored patterns.

We first define the *Energy function* of any two nodes  $x_i$  and  $x_j$  of  $\mathbf{x}$  as

$$e_{ij} = -w_{ij} x_i x_j \quad (15.25)$$

and the total *energy* of all  $d$  nodes in the network as the sum of all pair-wise energies:

$$\mathcal{E}(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d e_{ij} = -\frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d w_{ij} x_i x_j = -\frac{1}{2} \mathbf{x}^T \mathbf{W} \mathbf{x} \quad (15.26)$$

The interaction between these two nodes is summarized below:

	$x_j$	$x_i$	$w_{ij} > 0$	$w_{ij} < 0$
1	-1	-1	$e_{ij} < 0$	$e_{ij} > 0$
2	-1	1	$e_{ij} > 0$	$e_{ij} < 0$
3	1	-1	$e_{ij} > 0$	$e_{ij} < 0$
4	1	1	$e_{ij} < 0$	$e_{ij} > 0$

(15.27)

We make two observations.

- When the two nodes  $i$  and  $j$  reinforce each other's state,  $e_{ij} < 0$ :
  - If  $w_{ij} > 0$  (in cases 1 and 4),  $x_j = \mp 1$  tends to keep  $x_i$  to stay at the same state  $\mp 1$  in the iteration.
  - If  $w_{ij} < 0$  (in cases 2 and 3),  $x_j = \mp 1$  tends to keep  $x_i$  to stay at the same state  $\pm 1$ .
- When the two nodes  $i$  and  $j$  change each other's state,  $e_{ij} > 0$ :
  - If  $w_{ij} < 0$  (in cases 1 and 4),  $x_j = \mp 1$  tends to reverse  $x_i$  from its previous state  $\mp 1$  to  $\pm 1$ .
  - If  $w_{ij} > 0$  (in cases 2 and 3),  $x_j = \mp 1$  tends to reverse  $x_i$  from its previous state  $\pm 1$  to  $\mp 1$ .

We note that low energy  $e_{ij}$  corresponds to a stable interaction between  $x_i$  and  $x_j$ , i.e., they tend to remain unchanged, and high energy corresponds to an unstable interaction, i.e., they tend to change their states. As the result, low total  $\mathcal{E}(\mathbf{x})$  corresponds to more stable condition of the network, while high  $\mathcal{E}(\mathbf{x})$  corresponds to less stable condition.

We further show that the total energy  $\mathcal{E}(\mathbf{x})$  always decreases whenever the state of any node changes. Assume  $x_k$  has just been changed, i.e.,  $x_k^{(n+1)} \neq x_k^{(n)}$  ( $x_k = \pm 1$  but  $x_k^{(n+1)} = \mp 1$ ), while all others remain the same  $x_{l \neq k}^{(n+1)} = x_{l \neq k}^{(n)}$ . The energy before  $x_k$  changes state is

$$\begin{aligned} \mathcal{E}^{(n)}(\mathbf{x}) &= -\frac{1}{2} \left[ \sum_{i \neq k} \sum_{j \neq k} w_{ij} x_i^{(n)} x_j^{(n)} + \sum_i w_{ik} x_i^{(n)} x_k^{(n)} + \sum_j w_{kj} x_k^{(n)} x_j^{(n)} \right] \\ &= -\frac{1}{2} \sum_{i \neq k} \sum_{j \neq k} w_{ij} x_i x_j - \sum_i w_{ik} x_i^{(n)} x_k^{(n)} \end{aligned}$$

and the energy after  $x_k$  changes state is

$$\mathcal{E}^{(n+1)}(\mathbf{x}) = -\frac{1}{2} \sum_{i \neq k} \sum_{j \neq k} w_{ij} x_i x_j - \sum_i w_{ik} x_i^{(n+1)} x_k^{(n+1)} \quad (15.28)$$

The energy difference is

$$\Delta \mathcal{E} = (x_k^{(n)} - x_k^{(n+1)}) \sum_i w_{ik} x_i \quad (15.29)$$

Consider two cases:

- Case 1: if  $x_k^{(n)} = -1$ , but  $\sum_i w_{ik} x_i \geq 0$  and  $x_k^{(n+1)} = 1$ , we have  $(x_k^{(n)} - x_k^{(n+1)}) = -2 \leq 0$  and  $\Delta \mathcal{E} \leq 0$ .
- Case 2: if  $x_k^{(n)} = 1$ , but  $\sum_i w_{ik} x_i < 0$  and  $x_k^{(n+1)} = -1$ , we have  $(x_k^{(n)} - x_k^{(n+1)}) = 2 \geq 0$  and  $\Delta \mathcal{E} \leq 0$ .

As in either case,  $\Delta \mathcal{E}(\mathbf{x}) \leq 0$  is always true throughout the iteration, we conclude that  $\mathcal{E}(\mathbf{x})$  will eventually reach one of the minima of the *energy landscape*, i.e., the iteration will always converge.

We further show that each one of the  $K$  pre-stored patterns corresponds to one of the minima of the energy function. The energy function  $\mathcal{E}(\mathbf{x})$  corresponding to any  $\mathbf{x}$  can be written as

$$\mathcal{E}(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^T\mathbf{W}\mathbf{x} = -\frac{1}{2}\mathbf{x}^T\left[\frac{1}{d}\sum_{k=1}^K\mathbf{x}_k\mathbf{x}_k^T\right]\mathbf{x} = -\frac{1}{2d}\sum_{k=1}^K\mathbf{x}^T\mathbf{x}_k\mathbf{x}_k^T\mathbf{x} = -\frac{1}{2d}\sum_{k=1}^K(\mathbf{x}^T\mathbf{x}_k)^2 \quad (15.30)$$

If  $\mathbf{x}$  different (ideally orthogonal to) from any of the stored patterns, all  $K$  terms of the summation will be small (ideally zero). But if  $\mathbf{x}$  is the same as any one of the stored patterns, their inner product reaches maximum, causing the total energy to be minimized to reach one of the minima. In other words, the patterns stored in the network correspond to the local minima of the energy function. i.e., these patterns become *attractors*.

Note that it is possible to have other local minima, called *spurious states*, which do not represent any of the stored patterns, i.e., the associative memory is not perfect.

# 16 Perceptron-Based Networks

---

## 16.1 Perceptron

The perceptron network (Frank. Rosenblatt, 1958) is a two-layer learning network containing an input layer of  $d$  nodes and an output layer of  $m$  output node. The perceptron is a supervised method trained by dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , of which each sample  $\mathbf{x} = [x_1, \dots, x_d]^T$  is labeled by the corresponding component in  $\mathbf{y} = [y_1, \dots, y_N]^T$ , indicating to which one of  $K$  catagoric classes  $\mathbf{x}_n$  belongs to. The goal of the training process is to determine the weights  $\mathbf{w} = [w_1, \dots, w_d]^T$  and bias  $b$  so that the output  $\hat{y}_n = \sum_{i=1}^d w_i x_n + b$  corresponding to input  $\mathbf{x}_n$  matches its labeling  $y_n$  for all  $N$  samples in the training set in some optimal way. When the perceptron network is fully trained, it can be used to classify any unlabeled sample  $\mathbf{x}$  into one of the  $K$  classes.

We first consider the special case where the output layer has only  $m = 1$  node, and the perceptron is a binary classifier, same as the method of linear regression, and also the initial setup of the support vector machine (SVM), in the sense that all such methods are based on the linear equation  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$ , representing a hyperplane by which the  $d$ -dimensional feature space is partitioned into two regions corresponding to two classes  $C_+$  and  $C_-$ . When the parameters  $\mathbf{w}$  and  $b$  are determined in the training process, any unlabeled  $\mathbf{x}$  is classified into either of the two classes depending on whether the function value  $\hat{y} = f(\mathbf{x})$  is greater or smaller than zero:

$$\text{If } f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b \begin{cases} > 0 \\ < 0 \end{cases}, \quad \text{then } \hat{y} = \text{sign}(f(\mathbf{x})) = \begin{cases} 1 & \text{for } \mathbf{x} \in C_+ \\ -1 & \text{for } \mathbf{x} \in C_- \end{cases} \quad (16.1)$$

If we divide the equation  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$  by  $\|\mathbf{w}\|$ , we get

$$p_{\mathbf{w}}(\mathbf{x}) = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = -\frac{b}{\|\mathbf{w}\|} = b' \quad (16.2)$$

where  $p_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} / \|\mathbf{w}\|$  is the projection of  $\mathbf{x}$  onto the normal direction  $\mathbf{w}$  of the partitioning hyperplane, and  $b' = -b / \|\mathbf{w}\|$  is the vector from the origin to the hyperplane ( $|b'|$  is the distance of the hyperplane to the origin). Now the classification above can be rewritten as

$$\text{If } p_{\mathbf{w}}(\mathbf{x}) \begin{cases} > b' \\ < b' \end{cases}, \quad \text{then } \mathbf{x} \in \begin{cases} C_+ \\ C_- \end{cases} \quad (16.3)$$

We see that in all binary classification methods, an unlabeled sample  $\mathbf{x}$  is classified into either of the two classes based on the projection  $p_{\mathbf{w}}(\mathbf{x})$  of  $\mathbf{x}$  onto  $\mathbf{w}$ , which is either greater or smaller than the bias  $b'$ , depending on whether  $\mathbf{x}$  is on the positive or negative side of the plane.

In all these binary classification methods the parameters  $\mathbf{w}$  and  $b$  need to be determined based on training set  $\mathbf{X}$  labeled by  $\mathbf{y}$ , but they do so differently. While in least squares method and support vector machine these parameters are obtained by respectively the least squared method and quadratic programming, here in the perceptron network these parameters are obtained by iteratively modifying some randomly initialized values to gradually reduce the error or residual, the difference between actual output  $\hat{y}_n = f(\mathbf{x}_n)$  and the ground truth labeling  $y_n$ , denoted by  $\delta = y_n - \hat{y}_n$ , for all training samples in the dataset. This method is called the  $\delta$ -rule.

We now consider specifically the training algorithm of the perceptron network as a binary classifier. As always, we redefine the data vector as  $\mathbf{x} = [x_0 = 1, x_1, \dots, x_n]^T$  and the weight vector as  $\mathbf{w} = [w_0 = b, w_1, \dots, w_n]^T$  so that the decision function can be more conveniently written as an inner product  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  without the additional bias term.

The randomly initialized weight vector  $\mathbf{w}$  is iteratively updated based on the following mistake driven  $\delta$ -rule:

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \eta(y - \hat{y}) \mathbf{x} = \mathbf{w}^{old} + \eta \delta \mathbf{x} \tag{16.4}$$

where  $\eta > 0$  is the step size but here called the *learning rate*, which is assumed to be 1 in the following for simplicity. We can show that by the iteration above,  $\mathbf{w}$  is modified in such a way that the error  $\delta$  is always reduced.

When a training sample  $\mathbf{x}$  labeled by  $y \pm 1$  is presented to the nodes of the input layer of the perceptron, its output  $\hat{y} = \text{sign}(f(\mathbf{x}))$  may or may not match the the label  $y$ , as shown in the table:

$y$	$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}, \hat{y} = \text{sign}(f(\mathbf{x}))$	$\delta = y - \hat{y}$
1 $y = 1$	$f(\mathbf{x}) > 0, \hat{y} = 1$	$\delta = 0$
2 $y = -1$	$f(\mathbf{x}) > 0, \hat{y} = 1$	$\delta = -2$
3 $y = 1$	$f(\mathbf{x}) < 0, \hat{y} = -1$	$\delta = 2$
4 $y = -1$	$f(\mathbf{x}) < 0, \hat{y} = -1$	$\delta = 0$

(16.5)

In both the first and last cases,  $yf(\mathbf{x}) > 0$ , the error is  $\delta = y - \hat{y} = 0$ , and the weight vector  $\mathbf{w}^{new} = \mathbf{w}^{old} + \delta \mathbf{x} = \mathbf{w}^{old}$  is not modified. But in cases 2 and 3  $yf(\mathbf{x}) < 0$ , the error is  $\delta = y - \hat{y} = 1$ , the weight vector  $\mathbf{w}$  is modified in either of the following two ways:

- In case 2,  $y = -1$ , but  $\hat{y} = 1$ , then  $yf(\mathbf{x}) < 0$  and  $\delta = -2$ , we have

$$\mathbf{w}^{new} = \mathbf{w}^{old} - 2\mathbf{x} \tag{16.6}$$

When the same  $\mathbf{x}$  is presented to the network again in the future, the function is smaller than its previous value

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}^{new} = \mathbf{x}^T \mathbf{w}^{old} - 2\mathbf{x}^T \mathbf{x} < \mathbf{x}^T \mathbf{w}^{old} \quad (16.7)$$

and the output  $\hat{y}$  is more likely to be the same as the desired  $y = -1$ .

- In case 3,  $y = 1$ , but  $\hat{y} = -1$ , then  $y f(\mathbf{x}) < 0$  and  $\delta = 2$ , we have

$$\mathbf{w}^{new} = \mathbf{w}^{old} + 2\mathbf{x} \quad (16.8)$$

When the same  $\mathbf{x}$  is presented again, the function is greater than its previous value

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}^{new} = \mathbf{x}^T \mathbf{w}^{old} + 2\mathbf{x}^T \mathbf{x} > \mathbf{x}^T \mathbf{w}^{old} \quad (16.9)$$

and the output  $\hat{y}$  is more likely to be the same as the desired  $y = 1$ .

The update equations in Eq. (16.6) for  $y = -1$  and Eq. (16.8) for  $y = -1$  can be combined to become

$$\mathbf{w}^{new} = \mathbf{w}^{old} + y\mathbf{x} \quad (16.10)$$

where the scaling constant 2 is dropped as it can be absorbed into the learning rate if we let  $\eta = 1/2$ . Now the learning rule can be rewritten as:

$$\text{If } y f(\mathbf{x}) \begin{cases} > 0 \\ < 0 \end{cases} \text{ then } \begin{cases} \mathbf{w}^{new} = \mathbf{w}^{old} \\ \mathbf{w}^{new} = \mathbf{w}^{old} + y\mathbf{x} \end{cases} \quad (16.11)$$

In summary, the learning law guarantees that the weight vector  $\mathbf{w}$  is modified in such way that the performance of the network is always improved with reduced error  $\delta = y - \hat{y}$ . If  $C_+$  and  $C_-$  are linearly separable, then a perceptron will always produce  $\mathbf{w}$  in finite number of iterations to separate them.

This binary classifier with  $m = 1$  output node can now be generalized to multiclass classifier with  $m > 1$  output nodes, and each of the  $m$  weight vectors in  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_m]$  is modified by the same learning process considered above. The  $K > 2$  classes can be encoded by the  $m$  outputs  $y_i \in \{-1, 1\}$ , ( $i = 1, \dots, m$ ) in two different ways. If the *one-hot* method is used, the  $m$  binary output can encode  $K = m$  classes, i.e., the  $k$ th class is represented by an  $m$ -bit output with  $y_k = 1$  while all others  $y_l = -1$  for  $l \neq k$ . Alternatively, if binary encoding is used, the  $m$  outputs can encode as many as  $K = 2^m$  classes. For example,  $K = 4$  classes can be labeled by 4 binary vector of either 4 or 2 bits:

binary output	$y_0$	$y_1$	$y_2$	$y_3$		binary output	$y_0$	$y_1$	$y_2$	$y_3$
$y_1$	1	-1	-1	-1		$y_1$	-1	-1	1	1
$y_2$	-1	1	-1	-1	or	$y_2$	-1	1	-1	1
$y_3$	-1	-1	1	-1		$y_3$				
$y_4$	-1	-1	-1	1		$y_4$				

(16.12)

The outputs of the  $m$  output nodes form an  $m$ -dimensional binary vector  $\hat{\mathbf{y}} = [\hat{y}_1, \dots, \hat{y}_m]^T$ , which is to be compared with the labeling  $\mathbf{y}$  of the current input  $\mathbf{x}$  with error  $\delta = \|\mathbf{y} - \hat{\mathbf{y}}\|$ . When the training is complete, an unlabeled input  $\mathbf{x}$  is classified to one of the  $K$  classes with a matching label to the perceptron's output. In the case of one-hot encoding, it is possible for the binary output  $\mathbf{y}$  to not match any of the  $K$  one-hot encoded classes (e.g.,  $\hat{\mathbf{y}} = [-1 \ 1 \ -1 \ 1]^T$ ). In this case, the input  $\mathbf{x}$  can be classified to the class corresponding to the node with the greatest output value  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ .

The Matlab code for the essential part of the algorithm is listed below. Array  $\mathbf{X}$  is the dataset contains  $N$  training samples, each labeled by one of the components in array  $\mathbf{y}$  as its class identity. Array  $\mathbf{W}$  is a  $(d+1) \times m$  matrix containing  $d+1$  dimensional weight vectors each for one of the  $m$  output nodes.

```
[X Y]=DataOneHot;           % get data
K=length(unique(Y','rows')) % number of classes
X=[ones(1,N); X];          % data augmentation
[d N]=size(X);             % number of dimensions and number of samples
m=size(Y,1);               % number of output nodes
W=2*rand(d,m)-1;          % random initialization of weights
eta=1;
nt=10^4;                   % maximum number of iteration
for it=1:nt
    n=randi([1 N]);        % random index
    x=X(:,n);              % pick a training sample x
    y=Y(:,n);              % label of x
    yhat=sign(W'*x);       % binary output
    delta=y-yhat;         % error between desired and actual outputs
    for i=1:m
        W(:,i)=W(:,i)+eta*delta(i)*x; % update weights for all K output nodes
    end
    if ~mod(it,N)          % test for every epoch
        er=test(X,Y,W);
        if er<10^(-9)
            break
        end
    end
end

end
```

This is the function that test the training set based on estimated weight vectors in  $\mathbf{W}$ :

```
function er=test(X,Y,W) % test based on estimated W,
```

```

[d N]=size(X);
Ne=0; % number of misclassifications
for n=1:N
    x=X(:,n);
    yhat=sign(W'*x);
    delta=Y(:,n)-yhat;
    if any(delta) % if misclassification occurs to some output node
        Ne=Ne+1; % update number of misclassifications
    end
end
er=Ne/N; % error percentage
end

```

This is the code that generates the training set labeled by either one-hot or binary encoding method:

```

function [X,Y]=DataOneHot
d=3;
K=8;
onehot=1; % onehot=0 for binary encoding
Means=[ -1 -1 -1 -1 1 1 1 1;
        -1 -1 1 1 -1 -1 1 1;
        -1 1 -1 1 -1 1 -1 1];
Nk=50*ones(1,K);
N=sum(Nk); % total number of samples
X=[];
Y=[];
s=0.4;
s=0.2;
for k=1:K % for each of the K classes
    Xk=Means(:,k)+s*randn(d,Nk(k));
    if onehot
        Yk=-ones(K,Nk(k));
        Yk(k,:)=1;
    else % binary encoding
        dy=ceil(log2(K));
        y=2*de2bi(k-1,dy)-1;
        Yk=repmat(y',1,Nk(k));
    end
    X=[X Xk];
    Y=[Y Yk];
end
Visualize(X,Y)
end

```



**Examples**

The figure below shows the classification results of a perceptron network with  $n = 2$  input nodes and  $m = 2$  output nodes. The two output nodes encode  $2^2 = 4$  classes in a 2-D space. The training set contains 100 samples for each of the four classes labeled by  $\mathbf{y} = [y_1, y_2]$ . The two weight vectors  $\mathbf{w}_1$  and  $\mathbf{w}_2$  are initialized randomly. During the training iteration, when  $\delta = y - \hat{y} \neq 0$ , the weight vector for the output node is modified, otherwise, nothing needs to be done. After nine such modifications, the four classes are completely saperated by the two straight lines normal to the two weight vectors. The first panel shows the initial stage, while the subsequent panels show how the weight vectors are modified each time when  $\delta \neq 0$ . The darker and bigger dots represent the samples presented to the network when one of the weight vectors is modified.

The figure below shows the classification results of a perceptron of  $n = 3$  input nodes and  $m = 3$  output nodes encoding  $2^3 = 8$  classes. After 35 modifications the weight vectors, the perceptron is completely trained to classify all eight classes correctly. The first panel shows the initial stage, the following three panels show the weight vectors,  $\mathbf{w}_1$ ,  $\mathbf{w}_2$  and  $\mathbf{w}_3$ , also the normal vectors of the decision planes, from some three different viewing angles. We see that the eight classes are indeed saperated by the three planes normal to the weight vectors.

The main contrait of the perceptron algorithm as a binary classifier that the two classes are linearly separable can be removed by the kernel method, once the algorithm is modified in such a way that all data samples appear in the form of an inner product. Consider first the training process in which the  $N$  training samples are repeatedly presented to the network, and the weight vector is modified to become  $\mathbf{w}^{new} = \mathbf{w}^{old} + y_n \mathbf{x}_n$  whenever a sample  $\mathbf{x}_n$  labeled by  $y_n$  is misclassified with  $\delta = y_n - \hat{y}_n \neq 0$ . If the weight vector is initialized to zero  $\mathbf{w} = 0$ , then the weight vector by the updating rule in Eq. (16.10) can be written as a linear combination of the  $N$  training samples:

$$\mathbf{w} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n \tag{16.13}$$

where  $\alpha_n$  is the number of times sample  $\mathbf{x}_n$  labeled by  $y_n$  is misclassified. Upon receiving a new training sample  $\mathbf{x}_l$  labeled by  $y_l$ , we have

$$f(\mathbf{x}_l) = \mathbf{w}^T \mathbf{x}_l = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n^T \mathbf{x}_l, \quad \text{and} \quad \hat{y}_l = \text{sign}(f(\mathbf{x}_l)) \tag{16.14}$$

and the weight vector  $\mathbf{w}$  is updated by the delta-rule:

$$\begin{aligned} \text{If } \delta = y_l - \hat{y}_l \neq 0 \\ \text{then } \mathbf{w}^{new} = \mathbf{w}^{old} + y_l \mathbf{x}_l = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n + y_l \mathbf{x}_l \\ \text{i.e. } \alpha_l^{new} = \alpha_l^{old} + 1 \end{aligned} \tag{16.15}$$

We see that only  $\{\alpha_1, \dots, \alpha_N\}$  need to be updated during the training process,

while the weight vector  $\mathbf{w}$  in Eq. (16.13) no longer needs to be explicitly calculated. Once the training process is complete, any unlabeled  $\mathbf{x}$  can be classified into either of the two classes based on  $\{\alpha_1, \dots, \alpha_N\}$ :

$$\text{If } \mathbf{w}^T \mathbf{x} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n^T \mathbf{x} \begin{cases} > 0 \\ < 0 \end{cases}, \quad \text{then } \mathbf{x} \in \begin{cases} C_- \\ C_+ \end{cases} \quad (16.16)$$

As all data samples appear in the form of inner product in both the training and testing phase, the kernel method can be applied to replace the inner product  $\mathbf{x}_n^T \mathbf{x}_m$  by a kernel function  $k(\mathbf{x}_n, \mathbf{x}_m)$ . Also, the discussion above for  $m = 1$  output nodes can be generalized to  $m > 1$  output nodes.

Here is the Matlab code segment for the most essential parts of the kernel perceptron algorithm:

```
[X Y]=Data; % get dataset
[d N]=size(X); % d: dimension, N: number of training samples
X=[ones(1,N); X]; % augmented data
m=size(Y,1); % number of output nodes
A=zeros(m,N); % initialize alpha for all m output nodes and
K=Kernel(X,X); % get kernel matrix of all N samples
for it=1:nt
    n=randi([1 N]); % random index
    x=X(:,n); % pick a training sample
    y=Y(:,n); % and its label
    yhat=sign((A.*Y)*K(:,n)); % get yhat
    delta=Y(:,n)-yhat; % error between desired and actual output
    for i=1:m
        if delta(i)~=0 % if a misclassification
            A(i,n)=A(i,n)+1; % update the corresponding alpha
        end
    end
    if ~mod(it,N) % test for every epoch
        er=test(X,Y,A); % percentage of misclassification
        if er<10^(-9)
            break
        end
    end
end
end

function er=test(X,Y,A) % function for testing
[d N]=size(X); % d: dimension, N: number of training samples
m=size(Y,1);
Ne=0; % initialize number of misclassifications
for n=1:N % for all N training samples
    x=X(:,n); % get the nth sample
```

```

        y=Y(:,n);           % and its label
        f=(A.*Y)*Kernel(X,x); % f(x)
        yhat=sign(f);      % yhat=sign(f(x))
        if norm(y-yhat)~=0 % misclassification at some output nodes
            Ne=Ne+1;       % increase number of misclassification
        end
    end
end
er=Ne/N;                  % percentage of misclassification
end

```

**Example:**

The kernel perceptron is applied to the dataset of handwritten digits of  $K = 10$  classes for the digits from 0 to 9 with  $m = 10$  one-hot output nodes, based on the radial basis kernel. The kernel perceptron is trained on half of the data and then tested by the other half of the data. The confusion matrices of the training set by the end of the training process and the test set are both shown below. The error rate of the training set is zero, i.e., all training samples are correctly classified, while the test error rate is 0.11, i.e., 89% of the test samples are correctly classified.

$$\begin{bmatrix}
 116 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 108 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 108 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 116 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 113 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 122 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 103 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 108 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 109 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 116
 \end{bmatrix}
 \begin{bmatrix}
 104 & 7 & 0 & 0 & 0 & 1 & 2 \\
 0 & 103 & 0 & 0 & 0 & 0 & 3 \\
 1 & 0 & 103 & 0 & 0 & 0 & 1 \\
 0 & 0 & 8 & 102 & 0 & 0 & 0 \\
 1 & 1 & 0 & 0 & 103 & 2 & 0 \\
 0 & 0 & 0 & 2 & 0 & 93 & 24 \\
 0 & 1 & 1 & 0 & 0 & 0 & 95 \\
 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 1 & 0 & 1 & 1 & 3 & 2 & 1 \\
 1 & 2 & 0 & 1 & 7 & 0 & 0
 \end{bmatrix}
 \tag{16.17}$$

The constraint of the perceptron algorithm is the requirement that the classes are linearly separable due to the fact that there is only one level of learning taking place between the output and input layers. This constraint of linear separability will be removed when multi-layer networks are used, such as the back propagation algorithm to be discussed in the next section, and more generally, the deep learning networks containing a large number of learning layers between the input and output layers.

## 16.2 Back Propagation

The back propagation network (BP network or BPN) is a supervised learning algorithm that finds a wide variety of applications in practice. In the most gen-

eral sense, a BPN can be used as an associator to learn the relationship between two sets of patterns represented in vector forms. Specifically in classification, similar to the perceptron network, the BPN as a classifier is based on the training set  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , of which each pattern  $\mathbf{x}_n$ , a  $d$ -dimensional vector, is associated with the corresponding pattern, an  $m$ -dimensional vector  $\mathbf{y}_n$  in  $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]$ , as its class identity labeling, indicating to which of the  $K$  classes  $\{C_1, \dots, C_K\}$  pattern  $\mathbf{x}_i$  belongs.

Different from the perceptron in which there is only one level of learning taking place between the output and input layers in terms of the weights, a BPN is a multi-layer (three or more) hierarchical structure composed of the input, hidden, and output layers, in which learning takes place in multilevels between consecutive layers. Consequently, a BPN can be more flexible and powerful than the two-layer perceptron network. In the following, before we consider the general BPN containing multiple hidden layers, we will first derive the back propagation algorithm for the simplest BPN with only one hidden layer in between the input and output layers.

We assume the input, hidden and output layers contain respectively  $d$ ,  $l$ , and  $m$  nodes, each node in the hidden and output layers is fully connected to all nodes in the previous layer. When one of the  $N$  training patterns  $\mathbf{x}$  is presented to the input layer of the BPN, an  $m$ -D vector  $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}^h, \mathbf{W}^o)$  is produced at the output layer as the corresponding response to the input  $\mathbf{x}$ . Here  $\mathbf{W}^h = [\mathbf{w}_1^h, \dots, \mathbf{w}_l^h]$  and  $\mathbf{W}^o = [\mathbf{w}_1^o, \dots, \mathbf{w}_m^o]$  are the function parameters containing the augmented weight vectors for both the hidden and output layers, to be determined in the training process based on the training set  $\mathbf{X}$  and  $\mathbf{Y}$  so that its output  $\hat{\mathbf{y}}_n$  as a function of the current input  $\mathbf{x}_n$  matches the desired output, the labeling  $\mathbf{y}_n$ . Once the BPN is fully trained, any unlabeled pattern  $\mathbf{x}$  can then be classified into one of the  $K$  classes corresponding to the minimum  $\delta = \|\mathbf{y} - \hat{\mathbf{y}}\|$ . Note that different from the perceptron network, the output of the  $m$  output nodes are in general not binary, although they can still be binary if either one-hot or binary encoding is used for class labeling.

Specifically, the training of the BPN is an iteration of the following two-phase process:

- **The feedforward pass:**

A randomly selected sample  $\mathbf{x}_n$  labeled by  $\mathbf{y}_n$  is presented to the input layer and forwarded through the weighted connections to the hidden layer and then the output layer to produce output  $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}^h, \mathbf{W}^o)$ .

- **The backward error backpropagation:**

The squared error  $\varepsilon = \|\mathbf{y}_n - \hat{\mathbf{y}}_n\|^2/2$  measuring the difference between the desired output  $\mathbf{y}_n$  and the actual output  $\hat{\mathbf{y}}_n$  is propagated backward from the output layer through the hidden layer to the input layer, during which the weights of both the output and hidden layers are modified so that  $\varepsilon$  will be reduced when the same or similar pattern is presented in the future.

This two-phase process is iteratively carried out until eventually the error is minimized for all training samples and BPN is properly trained.

We now consider the specific computation taking place in both the forward and backward passes.

- The forward pass from input  $\mathbf{x} = [x_1, \dots, x_d]^T$  represented to the input layer to the output  $\hat{\mathbf{y}}_n = [y_1, \dots, y_m]^T$  produced by the output layer:
  - From input layer to hidden layer:

$$z_j = g(a_j^h) = g\left(\sum_{k=1}^d w_{jk}^h x_k + b_j^h\right) = g\left(\sum_{k=0}^d w_{jk}^h x_k\right) = g(\mathbf{x}^T \mathbf{w}_j^h) \quad (j = 1, \dots, l) \quad (16.18)$$

where both  $\mathbf{x} = [x_0 = 1, x_1, \dots, x_d]^T$  and  $\mathbf{w}_j^h = [w_{j0}^h = b_j^h, w_{j1}^h, \dots, w_{jd}^h]^T$  are augmented, and  $a_j^h = \mathbf{x}^T \mathbf{w}_j^h$  is the activation of the  $j$ th hidden layer node. These  $l$  equations can be expressed in vector form:

$$\mathbf{z} = \mathbf{g}(\mathbf{W}^h \mathbf{x}) \quad (16.19)$$

where  $\mathbf{z} = [z_1, \dots, z_l]^T$ , and  $\mathbf{W}^h = [\mathbf{w}_1^h, \dots, \mathbf{w}_l^h]^T$  is an  $l \times (d+1)$  matrix.

- From hidden layer to output layer:

$$\begin{aligned} \hat{y}_i &= g(a_i^o) = g\left(\sum_{j=1}^l w_{ij}^o z_j + b_i^o\right) = g\left(\sum_{j=0}^l w_{ij}^o z_j\right) \\ &= g\left(\sum_{j=0}^l w_{ij}^o g\left(\sum_{k=0}^d w_{jk}^h x_k\right)\right) \quad (i = 1, \dots, m) \end{aligned} \quad (16.20)$$

which can also be written in vector form as:

$$\hat{\mathbf{y}}_i = \mathbf{g}(\mathbf{z}^T \mathbf{w}_i^o) \quad (16.21)$$

where both  $\mathbf{z} = [z_0 = 1, z_1, \dots, z_l]^T$  and  $\mathbf{w}_i^o = [w_{i0}^o = b_i^o, w_{i1}^o, \dots, w_{il}^o]^T$  are augmented, and  $a_i^o = \mathbf{z}^T \mathbf{w}_i^o$  is the activation of the  $i$ th output layer node. These  $m$  equations can be expressed in vector form:

$$\hat{\mathbf{y}} = \mathbf{g}(\mathbf{W}^o \mathbf{z}) \quad (16.22)$$

where  $\hat{\mathbf{y}} = [\hat{y}_1, \dots, \hat{y}_m]^T$ , and  $\mathbf{W}^o = [\mathbf{w}_1^o, \dots, \mathbf{w}_m^o]^T$  is an  $m \times (l+1)$  matrix.

- The backward error propagation from error  $\varepsilon$  to the input layer:

The squared error  $\varepsilon = \|\mathbf{y} - \hat{\mathbf{y}}\|^2/2$  can be written as a function of the output weights  $w_{ij}^o$  ( $i = 1, \dots, m, j = 0, 1, \dots, l$ ) and hidden layer weights

$w_{jk}^h$  ( $j = 1, \dots, l$ ,  $k = 0, 1, \dots, d$ ):

$$\begin{aligned} \varepsilon &= \frac{1}{2} \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \frac{1}{2} \sum_{i=1}^m (\hat{y}_i - y_i)^2 = \frac{1}{2} \sum_{i=1}^m [g(a_i^o) - y_i]^2 = \frac{1}{2} \sum_{i=1}^m \left[ g \left( \sum_{j=0}^l w_{ij}^o z_j \right) - y_i \right]^2 \\ &= \frac{1}{2} \sum_{i=1}^m \left[ g \left( \sum_{j=0}^l w_{ij}^o g(a_j^h) \right) - y_i \right]^2 = \frac{1}{2} \sum_{i=1}^m \left[ g \left( \sum_{j=0}^l w_{ij}^o g \left( \sum_{k=0}^d w_{jk}^h x_k \right) \right) - y_i \right]^2 \end{aligned} \quad (6.23)$$

Similar to the objective function of the ridge regression considered in Section 4.3, an additional regularization term weighted by the *weight decay parameter*  $\lambda$  can be included in the objective function to penalize large weights and thereby prevent overfitting:

$$J(\mathbf{W}^h, \mathbf{W}^o) = \varepsilon + \frac{\lambda}{2} [\|\mathbf{W}^h\|_2^2 + \|\mathbf{W}^o\|_2^2] \quad (16.24)$$

where the two weight matrices in the regularization term should exclude the first column for the bias term  $b$  which need not to be forced to be small, and  $\|\mathbf{W}\|_2 = \sqrt{\sum_i \sum_j w_{ij}^2}$  is the *Frobenius norm* (Section A.17).

The gradient descent method is used to modify first the output layer weights and then the hidden layer weights to iteratively reduce the objective function  $J$  in the following steps:

- Find the gradient of  $J$  with respect to the output layer weights  $w_{ij}^o$  ( $i = 1, \dots, m$ ,  $j = 0, 1, \dots, l$ ) by the chain rule:

$$\frac{\partial J}{\partial w_{ij}^o} = \frac{\partial \varepsilon}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial a_i^o} \frac{\partial a_i^o}{\partial w_{ij}^o} + \lambda w_{ij}^o = (\hat{y}_i - y_i) g'(a_i^o) z_j + \lambda w_{ij}^o = -\delta_i g'(a_i^o) z_j + \lambda w_{ij}^o \quad (16.25)$$

where  $\delta_i = y_i - \hat{y}_i$ .

- Update  $w_{ij}^o$  ( $i = 1, \dots, m$ ,  $j = 0, 1, \dots, l$ ) to reduce  $J$  by gradient descent method with learning rate or step size  $\eta$ :

$$\begin{aligned} w_{ij}^o &\Leftarrow w_{ij}^o - \eta \frac{\partial J}{\partial w_{ij}^o} = w_{ij}^o - \eta (-\delta_i g'(a_i^o) z_j + \lambda w_{ij}^o) \\ &= w_{ij}^o - \eta (-d_i^o z_j + \lambda w_{ij}^o) \end{aligned} \quad (16.26)$$

where  $d_i^o = \delta_i g'(a_i^o)$ , or in matrix form:

$$\mathbf{W}^o \Leftarrow \mathbf{W}^o - \eta (-\mathbf{d}^o \mathbf{z}^T + \lambda \mathbf{W}^o) = \mathbf{W}^o + \eta \mathbf{d}^o \mathbf{z}^T - \eta \lambda \mathbf{W}^o \quad (16.27)$$

where  $\mathbf{d}^o = [d_1^o, \dots, d_m^o]^T$  is the elementwise (Hadamard) product  $\delta \odot \mathbf{g}'(\mathbf{a}^o)$ , and  $\mathbf{d}^o \mathbf{z}^T = \mathbf{d}^o \otimes \mathbf{z}$  is the outer product of  $\mathbf{d}^o$  and  $\mathbf{z} = [z_0 = 1, z_1, \dots, z_l]$ .

- Find the gradient of  $J$  with respect to the hidden layer weights  $w_{jk}^h$  ( $j = 1, \dots, l$ ,  $k = 0, 1, \dots, d$ ) by the chain rule:

$$\begin{aligned}
\frac{\partial J}{\partial w_{jk}^h} &= \frac{\partial \varepsilon}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial a_i^o} \frac{\partial a_i^o}{\partial z_j} \frac{\partial z_j}{\partial a_j^h} \frac{\partial a_j^h}{\partial w_{jk}^h} + \lambda w_{jk}^h = \sum_{i=1}^m (\hat{y}_i - y_i) \frac{\partial \hat{y}_i}{\partial a_i^o} \frac{\partial a_i^o}{\partial z_j} \frac{\partial z_j}{\partial a_j^h} \frac{\partial a_j^h}{\partial w_{jk}^h} + \lambda w_{jk}^h \\
&= - \sum_{i=1}^m \delta_i g'(a_i^o) w_{ij}^o g'(a_j^h) x_k + \lambda w_{jk}^h = - \left( \sum_{i=1}^m d_i^o w_{ij}^o \right) g'(a_j^h) x_k + \lambda w_{jk}^h \\
&= -\delta_j^h g'(a_j^h) x_k + \lambda w_{jk}^h = -d_j^h x_k + \lambda w_{jk}^h
\end{aligned} \tag{16.28}$$

where we have defined

$$\delta_j^h = \sum_{i=1}^m d_i^o w_{ij}^o, \quad d_j^h = \delta_j^h g'(a_j^h) = \left( \sum_{i=1}^m d_i^o w_{ij}^o \right) g'(a_j^h), \quad (j = 1, \dots, l) \tag{16.29}$$

or in matrix form:

$$\begin{bmatrix} \delta_1^h \\ \vdots \\ \delta_l^h \end{bmatrix} = \begin{bmatrix} w_{11}^o & \cdots & w_{1m}^o \\ \vdots & \ddots & \vdots \\ w_{l1}^o & \cdots & w_{lm}^o \end{bmatrix} \begin{bmatrix} d_1 \\ \vdots \\ d_m \end{bmatrix} = \mathbf{W}^o \mathbf{d}^o \tag{16.30}$$

Here  $\mathbf{W}^o$  is an  $m \times l$  matrix, the same as that defined above but with the first column of  $b_j$ 's removed.

- Update  $w_{jk}^h$  ( $j = 1, \dots, l$ ,  $k = 0, 1, \dots, d$ ) to reduce  $J$  by gradient descent method:

$$\begin{aligned}
w_{jk}^h &\leftarrow w_{jk}^h - \eta \frac{\partial J}{\partial w_{jk}^h} = w_{jk}^h - \eta (-\delta_j^h g'(a_j^h) x_k + \lambda w_{jk}^h) \\
&= w_{jk}^h - \eta (-d_j^h x_k + \lambda w_{jk}^h)
\end{aligned} \tag{16.31}$$

or in matrix form:

$$\mathbf{W}^h \leftarrow \mathbf{W}^h - \eta (-\mathbf{W}^o \mathbf{d}^o \odot \mathbf{g}'(\mathbf{a}^h) \mathbf{x}^T + \lambda \mathbf{W}^h) = \mathbf{W}^h + \eta \mathbf{d}^h \mathbf{x}^T - \eta \lambda \mathbf{W}^h \tag{16.32}$$

where  $\mathbf{d}^h = \mathbf{W}^o \mathbf{d}^o \odot \mathbf{g}'(\mathbf{a}^h)$  is the elementwise product of vector  $\mathbf{W}^o \mathbf{d}^o$  and  $\mathbf{g}'(\mathbf{a}^h)$ , and  $\mathbf{d}^h \mathbf{x}^T = \mathbf{d}^h \otimes \mathbf{x}$  is the outer product of vectors  $\mathbf{d}^h$  and  $\mathbf{x}$ .

In summary, here are the steps in each iteration:

1. Input a randomly selected pattern  $[x_1, \dots, x_d]^T$ , construct  $d + 1$  dimensional vector  $\mathbf{x} = [1, x_1, \dots, x_d]^T$ ;
2. Compute  $\mathbf{z} = \mathbf{g}(\mathbf{W}^h \mathbf{x})$ , and construct  $l + 1$  dimensional vector  $\mathbf{z} \leftarrow [1, \mathbf{z}]$ ;
3. Compute  $\hat{\mathbf{y}} = \mathbf{g}(\mathbf{W}^o \mathbf{z})$ ;
4. Get elementwise product  $\mathbf{d}^o = (\mathbf{y} - \hat{\mathbf{y}}) \odot \mathbf{g}'(\mathbf{a}^o) = \delta \odot \mathbf{g}'(\mathbf{a}^o)$ ;
5. Update output weights  $\mathbf{W}^o \leftarrow \mathbf{W}^o + \eta \mathbf{d}^o \mathbf{z}^T - \eta \lambda \mathbf{W}^o$ ;
6. Get elementwise product  $\mathbf{d}^h = \mathbf{W}_{m \times l}^o \mathbf{d}^o \odot \mathbf{g}'(\mathbf{a}^h) = \delta^h \odot \mathbf{g}'(\mathbf{a}^h)$ , where  $\mathbf{W}_{m \times l}^o$  is the same as  $\mathbf{W}^o$  but with its first column removed.
7. Update hidden weights  $\mathbf{W}^h \leftarrow \mathbf{W}^h + \eta \mathbf{d}^h \mathbf{x}^T - \eta \lambda \mathbf{W}^h$ ;

8. Terminate the iteration if the error  $\varepsilon$  is acceptably small for all of the training patterns. Otherwise repeat the above with another pattern in the training set.

The Matlab code for the essential part of the BPN algorithm is listed below. Array  $X$  contains  $C$  classes each with  $K$  samples, array  $Y$  are the labelings of the  $C * K$  training samples, array  $W$  contains the  $N + 1$  dimensional weight vectors for the  $M$  output nodes. Also,  $L$  is the number of hidden nodes,  $\eta$  is the learning rate between 0 and 1, and  $tol$  is the tolerance for the termination of the learning iteration (e.g., 0.01).

```
function [Wh, Wo, g]=backPropagate(X,Y)
    syms x;
    g=1/(1+exp(-x));           % Sigmoid activation function
    dg=diff(g);                % its derivative function
    g=matlabFunction(g);
    dg=matlabFunction(dg);
    [d,N]=size(X);            % number of inputs and number of samples
    X=[ones(1,N); X];        % augment X by adding a row of x0=1
    Wh=1-2*rand(L,d+1);      % Initialize hidden layer weights
    Wo=1-2*rand(M,L+1);      % Initialize output layer weights
    er=inf;
    while er > tol
        I=randperm(N);        % random permutation of samples
        er=0;
        for n=1:N            % for all N samples for an epoch
            x=X(:,I(n));     % pick a training sample
            ah=Wh*x;         % activation of hidden layer
            z=[1; g(ah)];    % augment z by adding z0=1
            ao=Wo*z;         % activation to output layer
            yhat=g(ao);      % output of output layer
            delta=Y(:,I(n))-yhat % delta error
            er=er+norm(delta)/N % test error
            do=delta.*dg(ao); % Find d of output layer
            Wo=Wo+eta*do*z'; % update weights for output layer
            dh=(Wo(:,2:L+1)')*do.*dg(ah); % delta of hidden layer
            Wh=Wh+eta*dh*x'; % update weights for hidden layer
        end
    end
end
```

The training process of BP network can also be considered as a data modeling problem as in regression to fit the given data  $\{(\mathbf{x}_i, \mathbf{y}_i), (i = 1, \dots, N)\}$  by a function with the weights of both the hidden and output layers as the parameters:

$$\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{W}^h, \mathbf{W}^o) \quad (16.33)$$



The goal is to find the optimal parameters  $\mathbf{W}^h$  and  $\mathbf{W}^o$  that minimize the difference  $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$  between the desired and the actual outputs. The Levenberg-Marquardt algorithm discussed previously can be used to obtain the parameters (e.g., the Matlab function *trainlm* <http://www.mathworks.com/help/nnet/ref/trainlm.html>).

The three-layer BPN containing a single hidden layer discussed above can be easily generalized to a multilayer BPN containing any number of hidden layers (e.g., for a deep learning network) by simply repeating steps 6 and 7 for the single hidden layer in the algorithm list above. We assume in addition to the input layer, there are in total  $L$  learning layers including all the hidden layers and the output layer, indexed by  $l = 1, \dots, L$ . Then step 6 above becomes:

$$\mathbf{d}^{(l)} = \delta^{(l)} \cdot \mathbf{g}'(\mathbf{a}^{(l)}) \quad (l = 1, \dots, L) \quad (16.34)$$

where  $\mathbf{a}^{(l)} = (\mathbf{W}^{(l)}\mathbf{z}^{(l)})$  is the activation of all nodes in the  $l$ th layer,  $\mathbf{z}^{(l)} = \mathbf{W}^{(l-1)}\mathbf{z}^{(l-1)}$  is the input to the  $l$ th layer, output from the  $(l-1)$ th layer ( $\mathbf{z}^{(1)} = \mathbf{x}$ ), and

$$\delta^{(l)} = \begin{cases} \mathbf{y} - \hat{\mathbf{y}} & (l = L) \\ \mathbf{W}^{(l)}\mathbf{d}^{(l-1)} & (1 \leq l < L) \end{cases} \quad (16.35)$$

Then the same step 7 for updating the weights can be carried out based on the gradient vector:

$$\mathbf{d}^{(l)} \otimes \mathbf{z}^{(l)} = \mathbf{d}^{(l)}(\mathbf{z}^{(l)})^T \quad (16.36)$$

The matlab code segment below is for the BP network of multiple hidden layers, in which  $L$  is the total number of learning layers and  $m(l)$  is the number of nodes in the  $l$ th layer ( $l = 1, \dots, L$ ):

```

W={1-2*rand(m(1),d+1)};           % initial weights for first layer
for l=2:L
    W{l}=1-2*rand(m(l),m(l-1)+1); % initial weights for all other layers
end
er=inf;
while er > tol
    I=randperm(N);                 % random permutation of samples
    er=0;
    for n=1:N                       % N samples for an epoch
        z={1;X(:,I(n))};           % pick a training sample
        a={W{1}*z{1}};             % activation of first layer
        for l=2:L                   % the forward pass
            z{l}=[1;g(a{l-1})];    % input to the lth layer
            a{l}=W{l}*z{l};        % activation of the lth layer
        end
        yhat=g(a{L});              % actual output of last layer
        delta=Y(:,I(n))-yhat;      % delta error
        er=er+norm(delta)/N;       % test error
        d{L}=delta.*dg(a{L});      % d for output layer
    end
end

```

```

W{L}=W{L}+eta*d{L}*z{L}'; % update weights for output layer
for l=L-1:-1:1 % the backward pass
    d{1}=(W{1+1}(:,2:end)')*d{1+1}).*dg(a{1}); % d for hidden layers
    W{1}=W{1}+eta*d{1}*z{1}'; % update weights for hidden layers
end
end
end
end

```

**Example 1:** The classification results of two previously used 2-D data sets are shown below. The error rates are respectively 13% and 11.5%, and the confusion matrices are:

$$\begin{bmatrix} 185 & 14 & 1 \\ 5 & 181 & 14 \\ 11 & 33 & 156 \end{bmatrix} \quad \begin{bmatrix} 176 & 24 \\ 22 & 178 \end{bmatrix} \quad (16.37)$$

**Example 2:**

The back propagation network applied to the classification of the dataset of handwritten digits from 0 to 9 used previously. Out of the  $N = 2240$  samples in the dataset, half is used for training while the other half for testing. Shown below are the confusion matrices of both the training (left) and testing (right) phases. Out of the 1120 samples for training 27 are misclassified (2.4%), and out of the 1120 samples for testing 74 are misclassified (6.6%).

$$\begin{bmatrix} 113 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 107 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 106 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 113 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 113 & 8 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 120 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 102 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 1 & 0 & 103 & 0 & 1 \\ 0 & 1 & 0 & 1 & 2 & 0 & 0 & 0 & 104 & 1 \\ 1 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 112 \end{bmatrix}, \quad \begin{bmatrix} 100 & 1 & 0 & 0 & 0 \\ 0 & 111 & 1 & 1 & 1 \\ 1 & 0 & 113 & 1 & 0 \\ 2 & 1 & 1 & 100 & 0 \\ 0 & 1 & 0 & 0 & 105 \\ 1 & 0 & 0 & 0 & 3 \\ 0 & 2 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 0 & 2 & 0 & 0 & 4 \end{bmatrix}, \quad (16.38)$$

Before leaving this section, we further make the following comments regarding how the back propagation network, and in general, multilayer feed forward network, is closely related to the general method of regression discussed previously.

- The computational model for a single neuron in Eq. (15.3) based on the logistic activation function in Eq. (15.5) is actually the same as Eq. (6.6) in logistic regression in Section 6.1:

$$g\left(\sum_{j=1}^d w_j x_j + b\right) = g(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} \quad (16.39)$$

treated as the conditional probability  $p(\hat{y} = 1|\mathbf{x}, \mathbf{w})$ . Both methods are to determine the model parameter  $\mathbf{w}$ , but based on different methods.

- A supervised learning neural network can be considered as a function  $\hat{\mathbf{y}} = \mathbf{f}(\mathbf{x}, \mathbf{w})$  as a model of the training dataset  $\mathbf{D} = \{\mathbf{X}, \mathbf{y}\}$ , while the parameter  $\mathbf{w}$  can be found by solving an optimization problem (Chapter 4), by either the least squares method that minimizes  $\varepsilon = \|\mathbf{y} - \hat{\mathbf{y}}\|^2$ , or the maximum likelihood method that maximizes  $L(\mathbf{w}) = P(\mathbf{y}|\mathbf{X}, \mathbf{w})$ , equivalent to minimizing  $-l(\mathbf{w}) = -\log L(\mathbf{w})$ , which can be interpreted as the cross entropy between the model distribution based on  $\mathcal{D}$  and the unknown ground true distribution, as discussed in Section 4.1.
- As a neural network can be considered as a model for the function  $\mathbf{y} = \mathbf{f}(\mathbf{x})$  relating  $\mathbf{y}$  to  $\mathbf{x}$  in the given dataset  $\mathcal{D}$ , a fundamental question to ask is whether such a network is able to model any function. This question is answered by the *universal approximation theorem*, stating that a feedforward network with arbitrary width (number of nodes in a hidden layer) or depth (number of hidden layers) can represent any function with arbitrary accuracy.

While this theorem is formally proven, it can also be understood intuitively. Recall the method of linear regression based on basis functions (Chapter 4), by which a function is approximated by a regression function as a linear combination of  $K$  basis functions (Eq. (4.59)):

$$\hat{y}_m = \sum_{k=1}^K w_{mk}^o \varphi_k(\mathbf{x}) = \sum_{k=1}^K w_{mk}^o g \left( \sum_{i=0}^d w_{ki}^h x_i \right) \quad (16.40)$$

which in the same form as in Eq. (16.20) for the three-layer feed forward network discussed above, with a linear activation function for the output layer. We see the the output of the hidden layer nodes are the basis functions, and the output of an output layer node is a linear combination of such basis functions. As shown in Example 3 there, a given function can be approximated with progressively higher accuracy as more basis functions, i.e., more hidden layer nodes, are used, if the width of the network is unbounded.

If the width of the network is bounded but its depth is unbounded, we can increase the number of layers of the network so that in the equation above we can have as many levels of summations as needed, i.e., the total number of terms in these summations can increased without a bound, then by the same argument, such a network can again approximate a given function with arbitrary accuracy.

## 16.3 Autoencoder

Autoencoder is a neural network method that learns to represent the patterns in a given dataset in such a way that both the dimensionality and the noise

can be reduced. An autoencoder is similar to a back propagation network in that it iteratively updates the weights in all hidden and output layers by back propagation so that eventually its outputs are as desired. But different from a back propagation network that learns to associate a pattern  $\mathbf{y}_i$  to another pattern  $\mathbf{x}_i$  as its labeling, the autoencoder is an unsupervised learning method based on unlabeled data, and it learns to associate each pattern  $\mathbf{x}_i$  to itself.

The purpose of the autoencoder is not to simply copy the data, instead, by imposing constraints such as sparsity to the hidden layer(s) in terms of the number of active nodes, the number of nodes in the hidden layer is much smaller than those in the input and output layers, and the hidden layer becomes the *bottleneck* of the multi-layer network, and can be considered as the *code* by which the information contained in the information presented by the input layer and reproduced by the output layer is encoded by the hidden layer nodes in a lower dimensional space.

The computation taking place in an autoencoder can therefore be considered as an encoding/decoding process. The encoder is composed of the input layer and the hidden layers, and the decoder is composed of the hidden layers and the output layer. Due to the limited number of nodes in the hidden layers, the network is forced to represent the data in reduced data size for data compression, if the components of  $\mathbf{x}$  in the dataset are correlated, and to discover certain potential structures in the dataset.

We see that an autoencoder is similar to the method of PCA based on the KLT in that both can extract essential information from the data and compress the data without losing much information. However, the PCA does so in a linear manner as the principal components are linear combinations of the input signal components; while in autoencoder, the components generated by the hidden layer nodes are nonlinear functions of the input signal components due to the activation function. For this reason, an autoencoder can be a more powerful method than PCA, but with a higher computational complexity.

Same as in back propagation, a pattern vector  $\mathbf{x}$  presented to the input layer is forward propagated first to the hidden layer (same as Eq. (16.19)):

$$\mathbf{z} = \mathbf{g}(\mathbf{W}^h \mathbf{x}), \quad \text{or} \quad z_j = g \left( \sum_{k=0}^d w_{jk}^h x_k \right) \quad (j = 1, \dots, l) \quad (16.41)$$

and then further to the output layer (same as Eq. (16.22)):

$$\hat{\mathbf{x}} = \mathbf{g}(\mathbf{W}^o \mathbf{z}) \quad \text{or} \quad \hat{x}_i = g \left( \sum_{j=0}^l w_{ij}^o z_j \right) \quad (i = 1, \dots, m) \quad (16.42)$$

Also same as in back propagation, the weights in both  $\mathbf{W}^h$  and  $\mathbf{W}^o$  are updated during training in the back propagation process so that the following objective function for the squared error between the input and desired output, the same

as the input, is minimized:

$$\varepsilon = \frac{1}{2} \|\mathbf{x} - \hat{\mathbf{x}}\|^2 \quad (16.43)$$

Here we assume there is only a single hidden layer for convenience, but the computation for both the forward and backward passes can be easily extended to multiple layers.

By limiting the number  $l$  of the hidden layer to be smaller than that of input and output layer, the dimensionality of the data can be reduced from  $d$  for the dimensionality of the input  $\mathbf{x} = [x_1, \dots, x_d]^T$  to  $l$ , the number of hidden layer nodes, while most of the information in the original data is reserved.

It is interesting to compare the autoencoder and the PCA based on KLT transform in terms of how they map a given data vector  $\mathbf{x}$  in the original  $d$ -dimensional space to a lower-dimensional space. Recall the KLT previously considered:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix} = \mathbf{V}^T \mathbf{x} = \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_d^T \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{v}_1^T \mathbf{x} \\ \vdots \\ \mathbf{v}_d^T \mathbf{x} \end{bmatrix} \quad (16.44)$$

In KLT, the  $i$ th component  $y_i = \mathbf{v}_i^T \mathbf{x}$  of  $\mathbf{y}$  is the inner product of the  $i$ th eigenvector  $\mathbf{v}_i$  and the input  $\mathbf{x}$ , the projection of  $\mathbf{x}$  onto  $\mathbf{v}_i$  if it is normalized. On the other hand, in a neural network, the inner product of the weight vector  $\mathbf{w}_i$  and the input  $\mathbf{x}$  is further mapped to  $z_i = g(\mathbf{w}_i^T \mathbf{x})$ . If, specially, the activation function  $g(u) = u$  is linear (an identical function), the weight vectors  $\mathbf{w}_i$  play the same role as the eigenvectors  $\mathbf{v}_i$  as the basis vectors that span the space, i.e., autoencoder and PCA are essentially the same. But as in general, the activation function  $g(u)$  is nonlinear, the autoencoder is a nonlinear mapping, of which the linear KLT (a rotation in the space) is a special case.

Dimension reduction exemplified by the KLT is effective if the components of the pattern vectors are highly correlated, i.e., some of them may carry similar information and therefore are redundant. However, if this is not the case, i.e., if all components carry their own independent information, dimensionality reduction will inevitably cause significant information loss. Similarly in an autoencoder, if the components of the data points are not mostly independent of each other, the autoencoder may not be able to reproduce them as the output if the number of hidden layer nodes is too small.

**Example 1:**

As dimension reduction methods, both autoencoder and PCA are applied to two datasets, the 4-D iris dataset of three classes, and a set containing eight classes each in one of the octants of the 3-D space. Then comparison of the two methods is made in terms of the separability in the resulting space of lower dimensions, measured by  $J_{B/T} = \text{tr}(\mathbf{S}_T^{-1} \mathbf{S}_B)$  defined in Eq. (8.24) in Section 8.2, and the classification accuracy (by the naive Bayes method) measured by error rate. Specifically, during the back propagation training of the autoencoder,

all samples in each dataset are used in random order as both the input and the desired out for the network gradually learn these patterns.

The figures above show the two datasets (the 3-D 8-class dataset at top, the 4-D 3-class iris dataset at bottom) when mapped to a 2-D space by the autoencoder (left) and PCA (right). We note that the data points are similarly distributed in the 2-D spaces resulted from the two methods, indicating the spaces are spanned by similar basis vectors, i.e., the weight vectors of the two hidden layer nodes are similar to the eigenvectors used in the PCA. However, by visual observation, the class separability in the 2-D space resulting from the autoencoder (left) seems to be a little better than that by the KLT (right), which is also confirmed quantitatively.

We note that in the first dataset, all three dimensions are equally important for the representation of the 8 classes, i.e., the dimensionality can hardly be reduced without losing significant amount of information. In fact, the three dimensions after the KLT contain roughly the same eigenvalues representing percentage of energy contained: 0.37%, 0.33%, 0.30%, i.e., reduction of dimensionality from 3 to 2 will impose a significant challenge for the subsequent classification in the space of reduced dimensionality, which can be measured by the separability and classification error rate, as listed in the table below for both autoencoder and KLT.

We see that for both datasets, the performance of the autoencoder (to the left of the slash symbol) is always slightly better than that of the PCA method (to the right of slash) with higher separability and lower error rate.

	Separability	Error rate	
8 classes in octants	1.52/1.51	0.304/0.320	(16.45)
Iris Dataset	1.12/1.07	0.020/0.027	

**Example 2:** In general, the autoencoder network is trained by all samples in the dataset. However, if all samples in a dataset are labeled to belong to one of a set of classes, the autoencoder can also be trained by these classes. Specifically in this example, the three classes in the iris dataset each represented by the average of all samples in the class are used to train the autoencoder. The figure below shows the 4-D iris dataset mapped to 2-D (top) and 3-D (bottom) spaces by both the autoencoder (left) and KLT (right). It is interesting to observe that in the case of autoencoder (left), the data points representing the three classes are arranged into a line structure in both 2 and 3-D spaces.

The same method is applied to the dataset of 10 classes of handwritten digits from 0 to 9, each represented by 224 training samples in image form of  $16 \times 16$  pixels, treated as a 256 dimensional vector. After the autoencoder network is fully trained based on the means of all 224 samples of each of the 10 classes, the dataset containing  $224 \times 10 = 2240$  data samples in the l-dimensional space

spanned by the hidden layer nodes that encode the 256-d samples is then visualized in 2 or 3-D space based the KLT transform. The figure below shows such visualizations of the dataset encoded by  $l = 3$  to  $l = 18$  hidden layer nodes. We note that when the dimensionality  $l$  is low, amazingly regular patterns (e.g., square and cubical structures) emerge in the spatial distribution of the data samples. The distribution will then remain mostly the same when the dimensionality  $l$  is further increased.

The desired sparsity of the hidden layer can be achieved by using a small number of hidden layer nodes, or letting only a small fraction of the hidden layer nodes to be active (but different nodes are active for different inputs). The second approach can be realized by (a) keeping only a small number of hidden layer nodes with maximum outputs while setting the rest to zero, or (b) forcing the average activation of the hidden nodes to be small.

The last method can be realized by including an extra regularization or penalty term in the objective function to encourage low average hidden node activation. To do so, we replace the second term in Eq. (16.24) by one of the following:

- L1 or L2 regularization

$$J = \varepsilon + \lambda \sum_{j=1}^l |a_j|, \quad J = \varepsilon + \frac{\lambda}{2} \sum_{j=1}^l a_j^2 \tag{16.46}$$

- KL-divergence

We first define the *sparsity parameter*  $\rho$  for the average activation of a hidden node, typically a small value close to zero (e.g.,  $\rho = 0.05$ ), and also find for the  $j$ th hidden layer node its average activation over all sample  $N$  points in the given dataset:

$$\hat{\rho}_j = \frac{1}{N} \sum_{n=1}^N \rho_j(\mathbf{x}_n) = \frac{1}{N} \sum_{n=1}^N g(a_j) = \frac{1}{N} \sum_{n=1}^N g \left( \sum_{k=0}^d w_{jk}^h x_k \right) \quad (j = 1, \dots, l) \tag{16.47}$$

where  $\rho_j(\mathbf{x}_n) = g(a_j) = g \left( \sum_k w_{jk}^h x_k \right)$  is the activation level of the  $j$ th hidden layer node while responding to input  $\mathbf{x}_n$ . The difference between the desired sparsity in terms of the desired average activation  $\rho$  and the actual one  $\hat{\rho}$  can be measured by the KL-divergence (Section 3.2.2):

$$KL(\rho || \hat{\rho}_j) = \rho \log \left( \frac{\rho}{\hat{\rho}_j} \right) + (1 - \rho) \log \left( \frac{1 - \rho}{1 - \hat{\rho}_j} \right) \tag{16.48}$$

where  $\rho$  is treated as the probability of the Bernoulli distribution for a binary random variable to be  $x = 1$  and  $1 - \rho$  is the probability for  $x = 0$ , and so is  $\hat{\rho}_j$  for another Bernoulli distribution. By minimizing  $KL(\rho || \hat{\rho}_j)$  for the difference between the two distributions, the actual activity  $\hat{\rho}_j$  is forced to be close to the desired  $\rho$ . The regularization is the sum of the KL-divergences of all  $l$  hidden layer nodes:  $\sum_{j=1}^l KL(\rho || \hat{\rho}_j)$ . We note that the

KL-divergence is a function of hidden layer weights  $w_{jk}^h$ , but independent of the output layer weights.

The plot below shows the comparison of the KL-divergence and the quadratic function  $a(\rho - \hat{\rho})^2$ . While both functions are minimized to zero when  $\hat{\rho} = \rho = 0.3$ , they behave differently as shown in the range of  $[0, 1]$ .

Similar to Eq. (16.24) for back propagation, we construct the objective function for the autoencoder as the sum of the error  $\varepsilon$  and the regularization term above to get

$$J(\mathbf{W}^h, \mathbf{W}^o) = \varepsilon + \lambda \sum_{j=1}^l KL(\rho || \hat{\rho}_j) \quad (16.49)$$

During the iterative learning the weights are updated based on gradient descent method, same as in back propagation. Given a specific input  $\mathbf{x}$  we first get the output  $\hat{\mathbf{x}}$  through the forward pass, and then modify the weights for both the output and hidden layers in the backward pass based on the gradient of the objective function with respect to the weights. While the computation for updating the weights for the output layer is the same as steps 4 and 5 in the algorithm listed before for back propagation, the computation for the weights for the hidden layer is different from steps 6 and 7 due to the regularization term, as shown below:

$$\frac{\partial}{\partial w_{jk}^h} J(\mathbf{W}^h, \mathbf{W}^o) = \frac{\partial \varepsilon}{\partial w_{jk}^h} + \lambda \frac{\partial}{\partial w_{jk}^h} \left[ \sum_{j=1}^l KL(\rho || \hat{\rho}_j) \right] \quad (16.50)$$

The first term above is the same as that in Eq. (16.28) (excluding the second regularization term), the second regularization term can be found by chain rule as:

$$\begin{aligned} & \frac{\partial}{\partial \hat{\rho}_j} \left[ \sum_{j=1}^l \rho \log \left( \frac{\rho}{\hat{\rho}_j} \right) + (1 - \rho) \log \left( \frac{1 - \rho}{1 - \hat{\rho}_j} \right) \right] \frac{\partial \hat{\rho}_j}{\partial w_{jk}^h} \\ &= \left[ \rho \frac{\partial}{\partial \hat{\rho}_j} (\log \rho - \log \hat{\rho}_j) + (1 - \rho) \frac{\partial}{\partial \hat{\rho}_j} [(\log(1 - \rho) - \log(1 - \hat{\rho}_j))] \right] \frac{\partial}{\partial w_{ij}^h} g \left( \sum_{k=0}^d w_{jk}^h z_i \right) \\ &= \left( -\frac{\rho}{\hat{\rho}_j} + \frac{1 - \rho}{1 - \hat{\rho}_j} \right) g'(a_j) x_k \end{aligned} \quad (16.51)$$

Now  $d_j^h$  in Eq. (16.28) for back propagation (defined in Eq. (16.29) ) can be modified as below for the autoencoder:

$$d_j^h = \left[ \left( \sum_{i=1}^m d_i^o w_{ij}^o \right) + \lambda \left( -\frac{\rho}{\hat{\rho}_j} + \frac{1 - \rho}{1 - \hat{\rho}_j} \right) \right] g'(a_j^h) \quad (16.52)$$

Using this modified  $d_j^h$  in step 6 of the algorithm for the propagation listed in the previous section, we get the algorithms for the autoencoder.



Autoencoder and PCA are similar for XOR and cocentric datasets. but different for data of digits from 0 to 9.

## 16.4 Deep Learning

Deep learning, a method based on multi-layer feed forward network with multiple hidden layers, has gained popularity and caused a new wave of enthusiasm for artificial neural networks in the recent years, due heavily to the progress in computational capability as well as the latest development in new algorithms.

Conceptually, deep learning network can be simply understood as a network composed of multiple learning layers, including more than one hidden layer as well as the output layer. The perceptron network with a single learning layer and the three-layer network with the hidden and output learning layers discussed previously can be considered as special cases of deep learning. If we model a learning layer by a nonlinear function that maps an input  $\mathbf{x}$  to an output

$$\mathbf{y} = \mathbf{g}(\mathbf{w}^T \mathbf{x}) = \mathbf{f}\mathbf{x}, \mathbf{w} \quad (16.53)$$

than an L-layer deep learning network can simply be described by

$$\mathbf{y} = \mathbf{f}^{(L)}(\mathbf{f}^{(L-1)}(\dots \mathbf{f}^{(1)}(\mathbf{x}, \mathbf{w}^{(1)}), \mathbf{w}^{(L-1)}), \mathbf{w}^{(L)}) \quad (16.54)$$

all for the purpose of modeling some desired relationship between input  $\mathbf{x}$  and output  $\mathbf{y}$ . When the multiple layers in a deep learning network are fully connected (FC), i.e., a node in a layer gets inputs from all nodes in the previous layer weighted by  $\mathbf{w}$ , then the method of back propagation discussed previously is used to update the weights based on gradient descent.

The basic structure of the multi-layer network has been inspired by the biology of the brain. The hierarchical structure of the network is composed of lower level layers where the neurons selectively respond to some simple local features in the data, while neurons in subsequent higher level layers selectively respond to progressively more complex and global patterns. Such a hierarchy is very similar to the two main visual pathways in the primate brain, known as the dorsal or “where” pathway and the ventral or “what” pathway, both formed by multiple levels of cortical areas starting from the primary visual cortex (V1), each composed of different types of neurons in terms of the size and specific type of the visual stimuli they are selectively responsive to.

Specifically, the hierarchy of ventral pathway known for visual object recognition is composed of neurons in the lower layer such as V1 where the simple cells with small receptive field selectively respond to edges of different orientations, and neurons with progressively larger receptive fields in the higher layers selectively respond to more complex features in larger areas of the visual field, such as neurons in the inferior temporal (IT) area sensitive to visual patterns such as image of faces.

As an example of deep learning networks, the evolutionary neural network

(CNN) is composed of multiple 2-D layers for the purpose of image object recognition. This network can be considered to a large extent a mimic of the ventral pathway of the visual cortex described above, in the sense that the nodes in the lower layers extract local features in the data (mimicing the simple cells in V1), while the nodes in higher layers respond selectively to more global patterns represented in terms of the local features detected by the lower layers (mimicing the cells in IT).

Specifically, in the lower layers of the CNN, local image features such as edges are detected by convolution, a basic technique commonly used in image processing. Mathematically, convolution is an operation defined for two functions  $f(t)$  and  $g$ :

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(t - \tau)h(\tau)d\tau = \int_{-\infty}^{\infty} h(t - \tau)x(\tau)d\tau \quad (16.55)$$

which is generalized to 2-D discrete functions in image processing:

$$y[m, n] = \sum_{i=-k}^k \sum_{j=-k}^k x[m - i, n - j] h[i, j] \quad (16.56)$$

where  $h[-i, -j] = h[i, j]$ , called *convolution kernel* or sometimes *filters*, is a central symmetric function defined over  $-k \leq i, j \leq k$ , represented by a  $(2k + 1) \times (2k + 1)$  square matrix, and  $y[m, n]$  is a pixel of the image  $y = h * x$  obtained by convolving image  $x$  by kernel  $h$ . The figure below illustrates 2-D convolution with a  $3 \times 3$  kernel.

We see that each pixel  $y[m, n]$  of the resulting image  $y$  is the sum of 9 products. As the kernel slides over the input image  $x$  pixel by pixel and then row by row, all pixels of  $y$  are calculated.

These four kernels are for detecting edges in vertical and horizontal orientations:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad (16.57)$$

while these four kernels are for detecting edges in two diagonal orientations:

$$\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad (16.58)$$

The figure below shows the effects of edge detection by convolution. The cat image is convolved with the kernels for in Eq. (16.57), while the panda image is convolved with kernels in Eq. (16.58).

We note that the convolution as a linear combination is in the same general operation  $\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{w})$  in neural networks with the elements in the kernel treated

as the weights, by which  $\mathbf{x}$  as the output from a lower arranged as a 2-D array is fed forward into the next layer that produces its output  $\mathbf{y}$ .

These  $3 \times 3$  kernels can be extended to larger kernels for detecting more complex features in a larger area of the visual field in higher layers of the network. imperical operations:

- *Convolution:*

Each node in the convolution layers is only locally connected to a set of neighboring nodes in a square patch in the previous layer. In particular, a node in the first layer take as input the pixel values inside its receptive fields, a subregion of the image. It then generates the convolution, the sum of the inputs weighted by the kernel components, and feed it to the next layer. At each position, multiple nodes with the same receptive field form a column in the depth dimension, all performing convolution but based on different kernel to extract different types of local features, such as edges in various orientations as shown above.

- *Padding:*

Padding is needed for the output of convolution to maintain the same size as the input. For example, for a  $5 \times 5$  kernel, the first pixel at the top-left corner of the output is in the position of the pixel in the 3th row and 3th column in the input, i.e., the output is 4 pixels smaller in both dimension than the input. To prevent this size reduction from one layer to the next, the input needs to be padded with additional rows and columns of pixels around its four borders (zeros or taking same value as the border pixels).

- *Stride:*

Stride  $s$  is the number of pixels for the revolution kernel to shift in both dimensions in convolution from one . In regular convolution,  $s = 1$ , the size of the resulting image is the same as the input. But when  $s > 1$ , the kernel shifts  $s$  pixels from the center of one receptive field to next. In other words, stride is the distance between the receptive field centers of neighboring neurons. The size of the output will be  $1/s$  of the original in both dimensions.

- *Pooling:*

Pooling is a down-sampling method by which a local patch (e.g.,  $2 \times 2$ ) in the previous layer is represented by their average or maximum as a single value in the next layer called pooling layer, for dimension reduction that serves two purposes: (a) local shift and rotational invariance and (b) computation reduction.

- *Dilution or Dropout:*

Certain fraction of nodes randomly selected in a layer is dropped by setting the corresponding weights to zero with a set probability, for the purpose of preventing overfitting, so that the network is less sensitive to

certain detailed variations in the data due most likely to noise and thereby more generalizable.

- In the last stage of the CNN, all nodes are fully connected and they taking progressively larger and more complex features extracted by the lower layers as inputs, these fully connected layers to carry out the final recognition based on backpropagation.

- *Weight decay:*

As mentioned before, a multilayer neural network can be considered as a function  $\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{w})$  for modeling the relationship between  $\mathbf{x}$  and  $\mathbf{y}$ , same as many supervised learning algorithms regression and classification. Similarly, including a penalty term proportional to  $\|\mathbf{w}\|$  or its square in the objective function is an effective way to regulate the algorithm and avoid overfitting.

## 16.5 Generative Adversarial Networks

<https://arxiv.org/pdf/1701.00160.pdf> <https://developers.google.com/machine-learning/gan>  
<https://www.youtube.com/watch?v=vi7lACKOUao> <http://neuralnetworksanddeeplearning.com/>

In general, generative methods try to learn the the probability density function (PDF) as a model for the data of interest. But in the case of *generative adversarial network GAN*, the network learns to generate samples from the pdf instead of the specific form of the pdf.

The GAN is essentially a two-player minimax game in which two players, the *discriminator* and the *generator*, try to achieve their own goals adversarial to each other.

- The discriminator is a binary classifier represented by a function  $D(\mathbf{x}; \theta_D)$  that maps an input  $\mathbf{x}$  to a real value between 1 and 0, indicating respectively whether  $\mathbf{x}$  is a real sample from the dataset, or it is a fake sample artificially generated. In principle the discriminator could be any kind of binary classifier with parameters which can be improved by gradient descent during the training process. But it is typically assumed to be deep neural network.
- The generator is modeled as a function  $\mathbf{x} = G(\mathbf{z}; \theta_G)$  that maps an input  $\mathbf{z}$ , a random vector (latent variable) with lower dimension then  $\mathbf{x}$ , into another vector  $\mathbf{x}$ , a fake sample that mimics the real samples as close to those real one from the dataset as possible so that the discriminator cannot distinguish between the real and the fake. The input  $\mathbf{z}$  controls various variations in the outputs of the generator. Typically the generator is also a deep neural network that maps random input  $\mathbf{z}$  ot a desired output.

The goal of the learning process is to reach the *Nash equilibrium*, a term in game theory, at which neither the discriminator nor the generator can improve its performance any further in terms of what they each try to achieve. Specifically,

the discriminator can no longer distinguish between the fake data samples from the generator and the real samples from the dataset, i.e., the model of the given dataset is estimated in terms of the new samples drawn from the learned pdf, although the pdf itself is not explicitly specified. The discriminator is only to help train the generator, once the generator is fully trained, the discriminator is no longer needed.

The discriminator tries to maximize the expectation of its output  $D(\mathbf{x})$  when the input is real but minimize the expectation of its output  $D(G\mathbf{z})$ , or maximize  $1 - D(G\mathbf{z})$ , when the input is fake from the generator, i.e., the discriminator is to maximize the following objective function:

$$J(D, G) = E_{p(x)}[\log D(\mathbf{x}, \theta_D)] + E_{p(z)}[\log(1 - D(G(\mathbf{z}, \theta_G)))] \quad (16.59)$$

while the generator is to minimize it, i.e., the goal of the overall GAN is

$$\min_G \max_D J(D, G) \quad (16.60)$$

Convergence of GANs is hard to reach as the discriminator and the generator are each optimizing a different objective function.

used mostly in image data, deep convolutional GAN, to generate large images (e.g., human faces) (e.g., a z-vector  $\mathbf{z}_{man+glasses} - \mathbf{z}_{man} + \mathbf{z}_{woman}$  may produce output of woman with glasses.)

# 17 Competition-Based Networks

---

## 17.1 Competitive Learning Network

Competitive learning is a neural network algorithm for unsupervised clustering, similar to the K-means algorithm considered previously. The competitive learning takes place in a two-layer network composed of an input layer of  $d$  nodes that receives an input vector  $\mathbf{x} = [x_1, \dots, x_d]^T$  as a point in the  $d$ -dimensional feature space, and an output layer of  $m$  nodes that produces an output pattern  $\mathbf{y} = [y_1, \dots, y_m]^T$  representing the clusters of interest. Each input variable  $x_i$  may take either a continuous real value or a binary value depending on the specific application, the outputs  $y_i \in \{0, 1\}$  are binary, of which only one is 1 while all others are 0 (one-hot), as the result of a winner-take-all completion based on their activation values.

In each iteration of the learning process, a pattern  $\mathbf{x}$  randomly chosen from the unlabeled dataset  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  is presented to the input layer of the network, while each node of the output layer gets an activation value, a linear combination of all such  $d$  inputs, i.e., the inner product of its weight vector  $\mathbf{w}_i$  and the input vector  $\mathbf{x}$ :

$$a_i = \sum_{j=1}^d w_{ij}x_j = \mathbf{w}_i^T \mathbf{x} = \|\mathbf{w}_i\| \|\mathbf{x}\| \cos \theta \quad (i = 1, \dots, m) \quad (17.1)$$

where  $\theta$  is the angle between vectors  $\mathbf{w}_i$  and  $\mathbf{x}$  in the  $d$ -dimensional feature space. If the data vectors are optionally normalized with unit length  $\|\mathbf{x}\|^2 = 1$ , i.e., they are points on the unit hypersphere in the space. Optionally if the weight vectors  $\mathbf{w}_i$  is also normalized with  $\|\mathbf{w}_i\| = 1$ , then their inner product above is only affected by the angle  $\theta$  between them.

The output of the network is determined by a winner-take-all competition. The node in the output layer that is maximally activated will output 1 while all others output 0:

$$y_i = \begin{cases} 1 & \text{if } a_i = \mathbf{w}_i^T \mathbf{x} = \max_j \mathbf{w}_j^T \mathbf{x} \\ 0 & \text{otherwise} \end{cases}, \quad (i = 1, \dots, K) \quad (17.2)$$

The inner product  $\mathbf{w}_i^T \mathbf{x}$  is closely related to the Euclidean distance  $d(\mathbf{x}, \mathbf{w}) =$

$\|\mathbf{w}_i - \mathbf{x}\|$ :

$$\begin{aligned} d^2(\mathbf{w}_i, \mathbf{x}) &= \|\mathbf{w}_i - \mathbf{x}\|^2 = (\mathbf{w}_i - \mathbf{x})^T (\mathbf{w}_i - \mathbf{x}) = \mathbf{w}_i^T \mathbf{w}_i - 2\mathbf{w}_i^T \mathbf{x} + \mathbf{x}^T \mathbf{x} \\ &= \|\mathbf{w}_i\|^2 + \|\mathbf{x}\|^2 - 2\mathbf{w}_i^T \mathbf{x} = \|\mathbf{w}_i\|^2 + \|\mathbf{x}\|^2 - 2\|\mathbf{w}_i\| \|\mathbf{x}\| \cos \theta \end{aligned} \quad (17.3)$$

If both  $\mathbf{x}$  and  $\mathbf{w}_i$  are normalized, the winning node with the maximum activation  $a_i = \mathbf{w}_i^T \mathbf{x}$  also has the minimum Euclidean distance  $\|\mathbf{w}_i - \mathbf{x}\|$  and angular distance  $\theta$ . Therefore the competition above can also be expressed as the following:

$$y_i = \begin{cases} 1 & \text{if } \|\mathbf{w}_i - \mathbf{x}\| = \min_j \|\mathbf{w}_j - \mathbf{x}\| \\ 0 & \text{otherwise} \end{cases}, \quad (i = 1, \dots, K) \quad (17.4)$$

The only winning node that receives the highest activation gets to modify its weight vector:

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + y_i \eta (\mathbf{x} - \mathbf{w}_i^{old}) = \begin{cases} (1 - \eta)\mathbf{w}_i^{old} + \eta\mathbf{x} & \text{if } y_i = 1 \\ \mathbf{w}_i^{old} & \text{if } y_i = 0 \end{cases} \quad (17.5)$$

where  $0 < \eta < 1$  is the learning rate (step size). Optionally the new weight vector can also be renormalized. For the iteration process to gradually approach a stable clustering result, the learning rate  $\eta$  needs to be reduced through the iterations from its initial value (e.g.,  $\eta_0 = 0.9$ ) toward 0 by a certain decay factor  $\alpha$  (e.g.,  $\alpha = 0.99$ ). This way the learning rate decays exponentially for the learning process to gradually slow down and becomes stabilized. The decay factor  $\alpha$  depends on the size of the dataset, and it may need to be determined heuristically and experimentally.

The competitive learning is illustrated in the figure above, where the modified weight vector of the winner  $\mathbf{w}_i^{new}$  is between  $\mathbf{x}$  and the old  $\mathbf{w}_i^{old}$ , i.e., the effect of the learning process is that the winner's weight vector which is closest to the current input pattern  $\mathbf{x}$  is pulled even closer to it.

Here are the iterative steps in the competitive learning:

- Step 0: Normalize all data vectors  $\mathbf{x}$ , initialize randomly the weight vectors  $\mathbf{w}$  for the output nodes (e.g., any  $m$  randomly chosen samples from the dataset):  $\mathbf{w}_1^{(0)}, \mathbf{w}_2^{(0)}, \dots, \mathbf{w}_m^{(0)}$ , set iteration index to zero  $l = 1$ ;
- Step 1: Choose randomly an input pattern  $\mathbf{x}$  from the dataset and calculate the activation for each of the output nodes:

$$a_i = \mathbf{w}_i^T \mathbf{x} \quad (i = 1, \dots, m) \quad (17.6)$$

- Step 2: find the index of the winning node  $j = \arg \max_i a_i$ , and update its weights:

$$\mathbf{w}_j^{(l+1)} = \mathbf{w}_j^{(l)} + \eta(\mathbf{x} - \mathbf{w}_j^{(l)}) \quad (17.7)$$

Reduce learning rate  $\eta \leftarrow \alpha\eta$ . Optionally, renormalize  $\mathbf{w}_j^{(l+1)}$ .

- Step 3: Terminate the iteration if the clustering result has gradually stabilized, when the learning rate  $\eta$  is reduced from its initial value  $\eta_0$  to some small value (e.g., 0.1) and the weight vectors no longer change significantly. Otherwise  $l \leftarrow l + 1$ , go back to Step 1.

This iterative process can be more intuitively understood as shown in the figure below. Every time a sample  $\mathbf{x}$  (a red dot) is presented to the input layer, one of the output nodes will become the winner and its weight vector  $\mathbf{w}$  (an blue  $\times$ ) closest to the current input  $\mathbf{x}$  is drawn even closer to  $\mathbf{x}$ . As this process is carried out iteratively many times, each cluster of similar sample points in the space will draw one of the weight vectors towards its central area, and the corresponding output node will always win and output 1 whenever any member of the cluster is presented to the network in the future. In other words, after this unsupervised learning, the feature space is partitioned into  $K$  regions each corresponding to one of the  $K$  clusters, represented by one of the output nodes, whose weight vector is in the central area of the region. We see that this process is very similar to the K-means clustering algorithm, in which each cluster is represented by the mean vector of all of its members.

It is possible in some cases that the data samples are not distributed in such a way that they form a set of clearly saperable clusters the feature space. In the extreme case, they may even form a continuum. In such cases, a small number of output nodes (possibly even just one) may become frequent winners, while others become “dead nodes” if they never win and consequently never get the chance to odify their weight vectors. To avoid such meaningless outcome, a mechanism is needed to ensure that all nodes have some chance to win. This can be implemented by including an extra bias term in the learning rule:

$$a_i = \mathbf{w}_i^T \mathbf{x} + b_i \quad (17.8)$$

where  $b_i$  is the bias term proportional to the difference between the “fair share” of winning  $1/m$  and the actual winning frequency:

$$b_i = c \left( \frac{1}{m} - \frac{\text{number of winnings of the } i\text{th node}}{\text{total number iterations so far}} \right) \quad (17.9)$$

The value of the bias term  $b_i$  for the  $i$ th node will change its winning frequency. If the node is winning more than its share  $1/m$ , then  $b_i < 0$  and it becomes harder for it to win in the future. On the other hand, if the node rarely wins,  $b_i > 0$  and its chance to win in the future is increased. Here hyperparameter  $c$  is some scaling coefficient. The greater  $c$ , the more balanced the competition will be. It needs to be fine tuned based on the specific nature of the data being analyzed.

This process of competitive learning can also be viewed as a *vector quantization* process, by which the continuous vector space is quantized to become a set of  $K$  discrete regions, called *Voronoi diagram (tessellation)*. Vector quantization can be used for data compression. A cluster of similar signals  $\mathbf{x}$  in the vector space can all be approximately represented by the weight vector  $\mathbf{w}$  of one of a small number of  $K$  output nodes in the neighborhood of  $\mathbf{x}$ , thereby the data size can be significantly reduced.

The Matlab code for the iteration loop of the algorithm is listed below.



```

[d N]=size(X);           % dataset containing N samples
b=zeros(1,K);           % bias terms for K clusters
freq=zeros(1,K);       % winning frequencies
eta=0.9;                % initial learning rate
decay=0.99;            % decay rate
it=0;
while eta>0.1           % main iterations
    it=it+1;
    W0=W;               % initial weight vectors
    x=X(:,randi(N,1));  % randomly select an input sample
    dmin=inf;
    for k=1:K           % find winner among all K output nodes
        d=norm(x-W(:,k))-b(k);
        if dmin>d
            dmin=d; m=k; % mth node is the winner
        end
    end
    w=W(:,m)+eta*(x-W(:,m)); % modify winner's weights
    W(:,m)=w/norm(w);      % renormalize its weights (optional)
    share(m)=share(m)+1;  % update share for winner
    b=c*(1/K-share/it);  % modify biases for all nodes
    eta=eta*decay;        % reduce learning rate
end

```

### Examples

The competitive learning method is applied to a set of simulated 2-D data of four clusters. The network has  $d = 2$  input nodes and  $K = 4$  output nodes. The first 16 iterations are shown in the figure below, where open circles represent the data points, while the solid back squares represent the weight vectors. Also, the blue and red squares represent the weight vector of the winner before and after its modification, visualized by the straight line connecting the two squares. We see that the  $K = 4$  weight vectors randomly initialized are iteratively modified one at a time, and after these 16 iterations, they each move to the center of one of the four clusters. The separability of the clustering result measured by  $\text{tr}\mathbf{S}_T^{-1}\mathbf{S}_B$  is 1.0. When the number of output nodes is reduced to 3 and 2, the separability is also reduced to 0.76 and 0.46, respectively. When  $K = 5$  output nodes are used, one of the four clusters is represented by two output nodes.

The same method is applied to a set of 3-D data of eight clusters. The results are shown in the figure below, where the top and bottom rows show the data and weight vectors before and after clustering, respectively. The left column shows the 2-D space spanned by the first two principal components, while the right column shows the original 2-D space. Again we see that the weight vectors move

from their random initial locations to the centers of the eight clusters as the result of the clustering.

The competitive learning algorithm is also applied to the dataset of handwritten digits from 0 to 9, and the clustering result is shown as the average of all samples in each cluster in  $16 \times 16$  image form shown in the figure below.

## 17.2 Self-Organizing Map (SOM)

The *Self-organizing map (SOM)* is a two-layer unsupervised neural network learning algorithm that maps any input pattern presented to its input layer, a vector  $\mathbf{x} = [x_1, \dots, x_d]^T$  in a  $d$ -dimensional feature space, to a set of output nodes that forms a low-dimensional space called *feature map*, typically a 2-D grid (lattice), although 1-D and 3-D spaces can also be used. A SOM can therefore be considered as a dimensionality reduction method, by which the dataset in the high-dimensional feature space can be visualized and the features and structures of the dataset can be discovered.

The learning rule of the SOM is similar to that of the competitive learning network considered previously in that all output nodes compete based on their activation levels upon the input pattern presented to input nodes of the network. However, different from the competitive learning network where only the winning node gets to modify its weight vector and learn to respond to a cluster of similar patterns individually and independently, here in SOM network all output nodes in the neighborhood of the winning node get to modify their weight vectors and thereby learn to respond collectively to a cluster of similar input patterns. As the result, all output nodes are locked into a continuum feature map with smooth and gradual variation in responsive selectivity.

The SOM as a learning algorithm is motivated by the mapping process in the brain, by which signals from various sensory systems (e.g., visual and auditory) are projected (mapped) to different cortical areas and responded to by the neurons wherein in such a way that all neurons in a local cortical area selectively respond to similar stimuli, as shown in the following examples.

- Spatially adjacent stimuli are represented and responded to by neurons in adjacent positions in cortex, known as *topographic organization*.
- Sound signals of different frequencies are mapped to the primary auditory cortex where neighboring neurons respond to similar frequencies.
- The visual field is topographically projected from the retina to the primary visual cortex V1 where adjacent neurons respond to adjacent areas in the visual field. This projection of the visual field is called *retinotopic map*.
- Visual edges/lines of different orientations are mapped to V1 where neighboring neurons in the *columnar architecture* respond to similar orientations.
- Visual motion of different directions are mapped to the columnar architecture

of the middle temporal area (MT or V5) where neighboring neurons respond to similar motion directions.

In a typical SOM network, the output nodes are organized in a 2-D feature map, and the competitive learning algorithm previously discussed is modified so that the learning takes place at not only the winning node in the output layer, but also a set of nodes in the neighborhood of the winner. The weight vectors of all such nodes are modified:

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + u_i \eta (\mathbf{x} - \mathbf{w}_i^{old}) \quad (17.10)$$

where  $u_i$  is a weighting function based on the Euclidean distance  $d_i$  from the  $i$ th node in the neighborhood to the winner in the 2-D feature map, such as a Gaussian function centered at the winner:

$$u_i = \exp\left(-\frac{d_i^2}{2\sigma^2}\right) \begin{cases} = 1 & \text{for winner with } d_i = 0 \\ < 1 & \text{for all other nodes} \end{cases} \quad (17.11)$$

where  $\sigma$  is a hyperparameter that controls the width of the Gaussian function. We see that all neurons in the neighborhood of the winner learn from the current input pattern in such a way that their weight vectors are modified to be closer to the input vector in the  $d$ -dimensional feature space, but they do so to different extents. Specifically, for the winner itself with  $d_i = 0$  and  $u_k = 1$ , the learning rate for the winner is maximally  $\eta$ , while all other nodes with  $d_i > 0$  also modify their weights but with some lower rates  $u_i \eta$ . Those closer to the winner with smaller  $d_i$  and thereby greater  $u_i$  will learn more than those farther away.

Here are the specific steps of the SOM learning algorithm:

1. Initialize weights of a set of output nodes arranged in a 2-D map.
2. Choose randomly an input pattern vector  $\mathbf{x}$  in the high-dimensional feature space and calculate the activation  $y_i = \mathbf{w}_i^T \mathbf{x}$  for each of the output nodes.
3. Find the winning node with maximum activation  $y_k$  and update the weights of all nodes in its neighborhood by Eq. (17.10).
4. Go back to step 2 until the feature map is no longer changing or a set maximum number of iterations is reached.

In the training process, both  $\sigma$  controlling the size of the neighborhood and  $\eta$  for the learning rate will be gradually reduced.

Shown below is the the Matlab code segment for the essential part of the SOM algorithm.

```
[d N]=size(X);           % dataset of N sample vectors
W=rand(d,M,M);          % initialization of weights
for i=1:M
    for j=1:M
        w=reshape(W(:,i,j),[d 1]);
        W(i,j,:)=W(:,i,j)/norm(w);    % normalize all weight vectors
```

```

        end
    end
    eta=0.9;           % initial learning rate
    sgm=M;           % width of Gaussian neighborhood
    decay=0.999;     % rate of decay
    for it=1:nt      % training iterations
        n=randi([1 N]);
        x=X(:,n);    % pick randomly an input
        amax=-inf;
        for i=1:M    % find the winner
            for j=1:M
                w=reshape(W(:,i,j),[d 1]);
                a=x'*w;           % activation as inner product
                if a>amax
                    amax=a; wi=i; wj=j; % record winner so far
                end
            end
        end
        for i=1:M
            for j=1:M
                dist=(wi-i)^2+(wj-j)^2; % distance to winner
                c=exp(-dist/sgm);      % Gaussian weights
                w=reshape(W(:,i,j),[d 1]); % get a weight vector
                w=w+eta*c*(x-w);        % modify weights
                w=w/norm(w);           % normalize weight vectors
                W(:,i,j)=w;           % save modified weight vector
            end
        end
        eta=eta*decay; % reduce learning rate
        sgm=sgm*decay; % reduce width of Gaussian
    end
end

```

**Example 1:**

This example of “self-organizing map” is how the SOM algorithm got its name. The output nodes are organized into a 2-D map and they learn to respond to the positions of a set of random points  $\mathbf{x} = [x_1, x_2]^T$  in a 2-D feature space. The weights of the output nodes are randomly initialized, and then modified during the learning process when the data points are repeatedly presented to the input of the network. When the SOM has been trained, due to the spatial correlation nature of the algorithm, the output nodes close to each other in the 2-D feature map respond to a set of adjacent data points in the 2-D space, thereby forming a self-organized spatial map, a roughly regular grid.

The 2-D map of the output nodes is visualized as shown in the figure below, where the spatial position of each node is determined by the two components

of its weight vector  $\mathbf{w} = [w_1, w_2]^T$  treated as its coordinates, and each node is connected to its four neighbors in the 2-D grid to indicate the spatial structure of the feature map.

In this example, the competitive learning is based on the distance between the weight vector  $\mathbf{w}_i$  and the current input  $\mathbf{x}$ , instead of their inner product  $\mathbf{w}_i^T \mathbf{x}$ . The node with the shortest distance  $\|\mathbf{w}_i - \mathbf{x}\|$  becomes the winner, and its weight vector  $\mathbf{w}$  is pulled even closer to the input, dragging along with it all its neighboring nodes.

The figure below shows the iterative modification of the weights. In each of the 16 iterations, the output node (blue dot) is closest to the input vector (red dot) and it becomes the winner.

The figure below shows the subsequent iterations in every 50 iterations. The resulting configuration of the output nodes is roughly a regular 2-D grid, a self-organized map in the 2-D feature space.

#### Example 2:

In this example the output nodes of the SOM are arranged into a 1-D loop and they respond to a set of cities in north America represented by their coordinates (longitudes and latitudes), for the purpose of finding a reasonably short path that goes through all of these cities (the traveling salesman problem). These nodes are visualized in terms of their weight vectors as points in a 2-D map. The weights are initialized to form circle in the map the cities. During the learning iteration when they learn to respond to the 2-D city coordinates as the input, their weights are modified so that they are gradually pulled toward the cities, thereby forming eventually a path through all the cities. In this case, the inner product  $\mathbf{w}_i^T \mathbf{x}$  is used to determine the winner of each iteration.

The figure below shows the results of the first 16 iterations:

The figure below shows the subsequent iterations in every 50 iterations. The resulting configuration of the output nodes approaches the desired path that passes through all cities. Although this result does not guarantee the path to be the shortest, it is reasonably close to such a solution required by the traveling salesman problem. The final result is unaffected by how the weights are initialized.

#### Example 3:

In this example, the output nodes in the 2-D feature map learn to respond to different colors treated as vectors in a 3-D space spanned by the three primary colors red, green and blue (RGB). When the SOM is trained, neighboring output nodes respond to similar colors. Here all input vectors are normalized (the hue of a color is fully determined by the ratios of the three primaries), and so are the weight vectors, which are always renormalized in each iteration.

To visualize the output nodes in the 2-D map, they are color coded based on the three components of their weight vectors treated as the three RGB colors, indicating their favored colors, the color they respond maximally, as shown in the figure below. The left panel is the nodes in the feature map with randomly initialized weights. The middle one shows these nodes after the SOM is fully trained. The right panel shows all the winners during the learning process, all encoded by

the input colors they win. The black areas are composed of the output nodes that never win but learn along with their neighboring winners. Interestingly, the winners seem to form some structure in the 2-D space. More detailed discussion can be found here: <http://fourier.eng.hmc.edu/e161/lectures/NeuralNetworks/node15.html>.

More examples of SOM can be found here: <https://en.wikipedia.org/wiki/Self-organizing>

## **Part VI**

---

# **Reinforcement Learning**





# 18 Introduction to Reinforcement Learning

---

## 18.1 Reinforcement Learning

Reinforcement learning (RL) can be considered as one of the three basic machine learning paradigms, alongside supervised learning (e.g., regression and classification) and unsupervised learning (e.g., clustering) discussed previously. The goal of RL is for the algorithm, a software *agent*, to learn to make a sequence of decisions, called *policy*, for a specific task in a given environment, for the purpose of receiving the maximum rewards from the environment.

Different from supervised learning for either regression or classification based on a given training dataset composed of a set of i.i.d. sample points  $\mathbf{x}_n$  labeled by  $y_n$ , ( $n = 1, \dots, N$ ), RL is a learning method without any examples of optimal behaviors. Instead, RL learns on its own without relying on any labeled training dataset explicitly informing the agent what the correct responses are while interacting with the environment. On the other hand, different from the unsupervised methods that learns passively from the given dataset, RL is a trial-and-error learning method, in which the agent actively interacts with and learns from the given environment.

Specifically, RL as a sequential method depends on the dynamics of the environment which is modeled by a *Markov decision process (MDP)*, a stochastic system of multiple states with probabilistic state transitions and rewards. If the MDP model of the environment in terms of its the state transition and reward probabilities is known, the agent only needs to find the optimal policy in terms of what action to take at each state of the MDP to achieve maximum accumulated rewards as the consequence of its actions. The task in this *model-based* case is called *planning*. On the other hand, if MDP of the environment is unknown, the agent needs to learn the environment by repeatedly running the dynamic process of the MDP based on some initial policy and gradually evaluate the received rewards and improve the policy to eventually reach optimality. The task in this *model-free* case is called *control*.

Depending on the action taken by the agent in the current state  $s$ , the system transits to the next state  $s'$  following certain transition probability distribution. The RL is to choose the optimal action in the current state, to maximize the long-term expected reward, as the feedback of the environment. This is typically carried out by *dynamic programming (DP)*, a class of algorithms seeking to

simplify a complex problem by breaking it up into a set of sub-problems which can be solved recursively.

The figure below illustrates the basic concepts of RL, where the agent makes a decision in terms of the action  $a_t$  to take in the current state  $s_t$  by following some policy  $\pi(a|s)$ , while the environment makes a transition from the current state  $s_t$  to the next state  $s_{t+1}$  by following the transition probability, a conditional probability  $p(s'|s, a)$  for the next state  $s_{t+1} = s'$  given the current state  $s_t = s$  and the action  $a_t$  taken by the agent, and provides a reward  $r_{t+1}$ .

## 18.2 Markov Decision Process

*Markov decision process (MDP)* is the mathematical framework for reinforcement learning. To understand MDP, we first consider the basic concept of *Markov process* or *Markov chain*, a stochastic model of a system that can be characterized by a set of states  $S = \{s_1, \dots, s_N\}$  of size  $|S| = N$  (cardinality of  $S$ ). The dynamics of the system in terms of its state transition is modeled by the transition probability  $P_{ij} = P(s_j|s_i)$  from the current state  $s_t = s_i$  to the next state  $s_{t+1} = s_j$  with  $i, j = 1, \dots, N$ . The system is assumed to be memoryless, i.e., its future is independent of the past history  $h_t = \{s_t, s_{t-1}, \dots, s_0\}$ , given the present  $s_t$ :

$$P(s_{t+1}|h_t) = P(s_{t+1}|s_t, s_{t-1}, \dots, s_0) \quad (18.1)$$

For example, if the state of a helicopter is described by its linear and angular position and velocity based on which its future position and velocity can be completely determined, then it can be modeled by a Markov process; but if the state is only described by its position, then its position in the past is needed for determining its future (e.g., to find its velocity), and it is not a Markov process.

All transition probabilities can be organized as an  $N \times N$  state transition matrix:

$$\mathbf{P} = \begin{bmatrix} P_{11} & \cdots & P_{1N} \\ \vdots & \ddots & \vdots \\ P_{N1} & \cdots & P_{NN} \end{bmatrix} \quad (18.2)$$

As the transition probabilities from any current state  $s_i$  to the  $N$  possible next states  $s_1, \dots, s_N$  have to sum up to 1, we have

$$\sum_{j=1}^N P_{ij} = \sum_{j=1}^N P(s_j|s_i) = 1, \quad (i = 1, \dots, N) \quad (18.3)$$

Any matrix satisfying this property is called a *stochastic matrix*.

In the following, our discussion is mostly concentrated on the the general transition from the current state  $s_t = s$  to the next state  $s_{t+1} = s'$  with transition probability  $P(s_{t+1} = s'|s_t = s) = P_{ss'}$ .

**Example** Weather in Los Angeles:

$$\mathbf{P} = \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.4 & 0.3 & 0.3 \\ 0.5 & 0.2 & 0.3 \end{bmatrix} \quad (18.4)$$

We next consider a *Markov reward process (MRP)*, represented by a tuple  $\langle S, P, R, \gamma \rangle$  of four elements for the states, state transition probabilities, rewards, and the *discount factor*. Here the *reward*  $R$  can be considered as a feedback signal to the agent at each time step of the sequence of state transitions, indicating how well it is doing with respect to the overall task. The reward can be negative, as a penalty, for situations to be avoided. The performance of the agent is measured by the sum of rewards accumulated over all time steps of the Markov process, weighted by the discount factor  $\gamma \in [0, 1]$  that discounts future rewards. If  $\gamma$  is close to 0, then the immediate reward is emphasized (short-sighted or greedy), but if  $\gamma$  is close to 1, then rewards in the future steps will be almost as valuable as immediate ones (far-sighted).

The sequence of state transitions of an MRP from an initial state  $s_0$  to a terminal state  $s_T$  is called an *episode*, and the number of time steps  $T$  is called *horizon*, which can be either finite or infinite.

Both the reward  $r$  and the next state  $s'$  resulting from arriving at the current state  $s$  are considered as random variables with joint probability  $p(s', r|s) = P(s_{t+1} = s', r_{t+1} = r | s_t = s)$ , based on which both the state transition probability and state reward probability can be found by marginalization:

$$p(s'|s) = P(s_{t+1} = s' | s_t = s) = \sum_{r \in R} p(s', r|s), \quad P(r|s) = \sum_{s' \in S} p(s', r|s) \quad (18.5)$$

Note that conventionally the reward  $r$  received after arriving at state  $s_t$  is denoted by  $r_{t+1}$ , instead of  $r_t$ . The index  $s' \in S$  for the summation over all states will be abbreviated to  $s'$  in the subsequent discussion. The expectation of the reward at  $s$  is

$$r(s) = E[r_{r+1} | s_t = s] = \sum_r r P(r|s) = \sum_r r \sum_{s'} P(s', r|s) \quad (18.6)$$

We define the *return*  $G_t$  at each step  $s = s_t$  as the accumulate reward, the sum of the immediate reward after arriving at the current state  $s = s_t$  and all delayed rewards in the future states upto a terminal state  $s_T$  with reward  $r_T$  at the end of the episode, discounted by  $\gamma$ :

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \quad (18.7)$$

We further define the *state value* function  $v(s)$  at state  $s = s_t$  as the expectation of the return, which can be further expressed recursively in terms of the values

$v(s')$  of all possible next states  $s' = s_{t+1}$ :

$$\begin{aligned}
 v(s) &= E[G_t | s_t = s] = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t = s] \\
 &= E[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \dots) | s_t = s] \\
 &= E[r_{t+1} | s = s_t] + \gamma E[G_{t+1} | s_{t+1} = s', s_t = s] \\
 &= r(s) + \gamma \sum_{s'} P(s' | s) v(s') \tag{18.8}
 \end{aligned}$$

where  $r(s)$  is given in Eq. (18.6). The value of a terminal state  $s_T$  at the end of an episode is zero, as there will be no next state and thereby no more future reward.

This equation is called the *Bellman equation*, by which the value  $v(s)$  at current state  $s$  is expressed recursively in terms of the immediate reward  $r(s)$  and the values  $v(s')$  of all possible next states, without explicitly invoking all future rewards. Based on this *bootstrap* idea of the Bellman equation, a multi-step MRP problem can be expressed recursively as a subproblem concerning only a single state transition from  $s$  to  $s'$ , as a backward induction to find current state value from all future ones. For this reason, the Bellman equation plays an essential role in future discussion of some important algorithms in reinforcement learning.

The Bellman equation in Eq. (18.8) holds for all states  $s_1, \dots, s_N$ , and the resulting  $N$  equations can be expressed in vector form as:

$$\begin{aligned}
 \mathbf{v} &= \begin{bmatrix} v(s_1) \\ \vdots \\ v(s_N) \end{bmatrix} = \begin{bmatrix} r(s_1) \\ \vdots \\ r(s_N) \end{bmatrix} + \gamma \begin{bmatrix} P(s_1|s_1) & \dots & P(s_N|s_1) \\ \vdots & \ddots & \vdots \\ P(s_1|s_N) & \dots & P(s_N|s_N) \end{bmatrix} \begin{bmatrix} v(s_1) \\ \vdots \\ v(s_N) \end{bmatrix} \\
 &= \mathbf{r} + \gamma \mathbf{P} \mathbf{v} \tag{18.9}
 \end{aligned}$$

where  $\mathbf{P}$  is called the *stochastic matrix*. Solving this linear equation system of size  $N$  we find

$$\mathbf{v} = (\mathbf{I} - \gamma \mathbf{P})^{-1} \mathbf{r} \tag{18.10}$$

Such a solution exists as matrix  $\mathbf{I} - \gamma \mathbf{P}$  is invertible. This can be shown by noting that all eigenvalues of the stochastic matrix  $\mathbf{P}$  are not greater than 1:  $|\lambda| \leq 1$  (Section A.9), and all eigenvalues of  $\gamma \mathbf{P}$  are smaller than 1:  $\gamma \lambda < 1$ , i.e., the eigenvalue of  $\mathbf{I} - \gamma \mathbf{P}$  is greater than zero:  $1 - \gamma \lambda > 0$ . Consequently the determinant  $\det(\mathbf{I} - \gamma \mathbf{P})$  of this coefficient matrix, as the product of all its nonzero eigenvalues, is non-zero and therefore matrix  $\det(\mathbf{I} - \gamma \mathbf{P})$  is invertible.

Alternatively, the Bellman equation in Eq. (18.8) can also be solved iteratively by the general method of *dynamic programming (DP)*, which solves a multi-stage planning problem by a backward induction and find the value function recursively. We first rewrite the Bellman equation as

$$\mathbf{v} = \mathbf{r} + \gamma \mathbf{P} \mathbf{v} = B(\mathbf{v}) \tag{18.11}$$

where  $B(\mathbf{v}) = \mathbf{r} + \gamma \mathbf{P} \mathbf{v}$  is defined as a vector-valued function (an operator)

applied to the vector argument  $\mathbf{v}$ . Now the Bellman equation can be solved iteratively from an arbitrary initial value, such as  $\mathbf{v}_0 = \mathbf{0}$ :

$$\mathbf{v}_{n+1} = B(\mathbf{v}_n) = \mathbf{r} + \gamma \mathbf{P} \mathbf{v}_n \quad (18.12)$$

This iteration will always converge to the root of the equation, the *fixed point* of function  $B(\mathbf{v})$ , as it is a *contraction mapping* first introduced in Section 1.3, that satisfies:

$$\begin{aligned} \|B(\mathbf{v}_i) - B(\mathbf{v}_j)\| &= \|\mathbf{r} + \gamma \mathbf{P} \mathbf{v}_i - (\mathbf{r} + \gamma \mathbf{P} \mathbf{v}_j)\| = \gamma \|\mathbf{P}(\mathbf{v}_i - \mathbf{v}_j)\| \\ &\leq \gamma \|\mathbf{P}\| \|\mathbf{v}_i - \mathbf{v}_j\| = \gamma \|\mathbf{v}_i - \mathbf{v}_j\| < \|\mathbf{v}_i - \mathbf{v}_j\| \end{aligned} \quad (18.13)$$

That  $B(\mathbf{v})$  is a contraction can also be proven by showing the norm of its Jacobian matrix, its derivative with respect to its vector argument  $\mathbf{v}$ , is smaller than 1. We first find the Jacobian matrix

$$\mathbf{J}_B = \frac{d}{d\mathbf{v}} B(\mathbf{v}) = \frac{d}{d\mathbf{v}} (\mathbf{r} + \gamma \mathbf{P} \mathbf{v}) = \gamma \mathbf{P} \quad (18.14)$$

and then find its p-norm with  $p = \infty$  (equivalent to  $p = 1, 2$ ), the maximum absolute row sum:

$$\|\mathbf{J}_B\|_{p=\infty} = \|\gamma \mathbf{P}\|_{\infty} = \gamma \|\mathbf{P}\|_{\infty} = \gamma \max_{1 \leq i \leq N} \sum_{j=1}^N |P_{ij}| = \gamma < 1 \quad (18.15)$$

The last equality is due to Eq. (18.3), i.e.,  $\|\mathbf{P}\|_{\infty} = 1$ .

In summary, the Bellman equation in Eq. (18.8) can be solved by either of the two methods in Eqs. (18.10) and (18.12), so long as  $\gamma < 1$ . When the size of state space  $|S| = N$  is large, the complexity  $O(N^3)$  is too high then the iterative method may be more efficient.

Based on the definition of MRP, we further define *Markov decision process (MDP)* as an MRP with certain decision-making rule called *policy*, denoted by  $\pi$ , following the policy the agent takes one of the actions  $a \in A(s)$  available at each state  $s$  to control the state transition of the process. An MDP can therefore be described by a tuple  $\langle S, P, R, A, \gamma \rangle$ , of which  $A$  is the set of all actions. The policy  $\pi$  can be either stochastic denoted by  $\pi(a|s) = P(a_t = a | s_t = s)$  as the conditional probability of taking action  $a$  in state  $s$ , or deterministic denoted by  $a = \pi(s)$ . A policy is *soft* if any of the actions available at a state is possible to be taken, i.e.,  $\pi(a|s) > 0$  for all  $a \in A(s)$  and all  $s \in S$ . In particular, a policy is  $\epsilon$ -soft if  $\pi(a|s) \geq \epsilon/|A(s)|$  for some small value of  $\epsilon$ .

As there are  $|A(s_n)|$  possible actions to take in state  $s_n \in S$ , the total number of policies is  $|A(s_1)| \cdots |A(s_N)| = \prod_{n=1}^N |A(s_n)|$ . If the number of available actions  $|A|$  is the same for all  $|S|$  states, then the total number of policies is  $|A|^{|S|}$ .

Following a given policy from an initial state  $s_0$  will result a sequence of state transitions called a *trajectory* of the episode:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \cdots s_T \quad (18.16)$$

of which the state visited at the t-th step denoted by  $s_t$  can be any of the  $N$

states in  $S = \{s_1, \dots, s_N\}$  of the MDP, not to be confused with the  $t$ -th state in  $S$ . Note that in an episode some of the states in  $S$  may be visited multiple times, while some others may never be visited. Also note that due to the random nature of the MDP, following the same policy may not result in the same trajectory.

Along the trajectory of an episode, an accumulation of discounted rewards from all states visited will be received. Our goal is to find the optimal policy as a sequence of actions  $a_0, a_1, a_2, \dots$  for a given MDP model of the environment in terms of its state transition dynamics and rewards, so that the value  $v(s_0)$  at the start state  $s_0$ , the expectation of the sum of all discounted future rewards, is maximized. This optimization problem can be solved by the method of dynamic programming, and such a process is called *planning*.

In an MDP, both the next state  $s'$  and reward  $r$  are assumed to be random variables with joint probability  $p(s', r|s, a) = P(s_{t+1} = s', r_{t+1} = r|s_t = s, a_t = a)$  conditioned on the action  $a$  and the previous state  $s$ . The state transition probability becomes

$$p(s'|s, a) = P(s_{t+1} = s'|s_t = s, a_t = a) = \sum_{r \in R} p(s', r|s, a) \quad (18.17)$$

and the expected reward  $r$  received after arriving at the current is

$$r(s, a) = E[r_{t+1}|s_t = s, a_t = a] = \sum_r r p(r|s, a) = \sum_r r \sum_{s'} p(s', r|s, a) \quad (18.18)$$

Following a specific random policy  $\pi(a|s)$ , the agent takes an action  $a \in A(s)$  to transit from the current state  $s_t = s$  to the next state  $s_{t+1} = s'$  with probability

$$p_\pi(s'|s) = P(s_{t+1} = s'|s_t = s) = \sum_{a \in A(s)} \pi(a|s) p(s'|s, a) \quad (18.19)$$

satisfying  $\sum_{s'} P_\pi(s'|s) = 1$  and the reward in state  $s$ :

$$r_\pi(s) = E_\pi[r_{t+1}|s_t = s] = \sum_{a \in A(s)} \pi(a|s) r(s, a) \quad (18.20)$$

where  $E_\pi$  denotes the expectation with respect to a certain policy  $\pi(a|s)$ . The summation over all available actions  $a \in A(s)$  in state  $s$  will be abbreviated to  $a$  in the following.

We further define two important functions with respect to a given policy  $\pi$  of an MDP:

- The *state value function*  $v_\pi(s)$  is the expected return at each state  $s$  of the MDP while following policy  $\pi$ , similar to the value function of an MRP as  $v(s)$  in Eq. (18.8), but now treated as a function of action  $a$  as well as state

$s$ :

$$\begin{aligned}
 v_\pi(s) &= E_\pi[G_t | s_t = s] \\
 &= \sum_a \pi(a|s) \left( r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_\pi(s') \right) \\
 &= \sum_a \pi(a|s) q_\pi(s, a) \tag{18.21}
 \end{aligned}$$

where  $q_\pi(s, a)$  is defined below.

- The *state-action value function*  $q_\pi(s, a)$  is the expected return of taking a specific action  $a$  (irrelevant to policy  $\pi$ ) in state  $s$ , and then following  $\pi$  in all subsequent states:

$$\begin{aligned}
 q_\pi(s, a) &= E_\pi[G_t | s_t = s, a_t = a] \\
 &= r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_\pi(s') \\
 &= r(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') q_\pi(s', a') \tag{18.22}
 \end{aligned}$$

where  $v_\pi(s')$  is recursively represented as a weighted sum of  $q_\pi(s', a')$  based on Eq. (18.21).

The state-action value  $q_\pi(s, a)$  plays a more important role than  $v_\pi(s)$  as it allows the freedom of taking any action independent of a given policy  $\pi$ , thereby allowing the opportunity to improve an existing policy, e.g., by taking a *greedy* action to maximize the value  $q_\pi(s, a)$ . The state-action value function  $q_\pi(s, a)$  is often abbreviated as the action value or *Q-value* for convenience in future discussions.

As a function of the state-action pair  $(s, a)$ , the state action value  $q_\pi(s, a)$  can be represented as a table of  $|S|$  rows each for one of the states  $s$ , and  $|A|$  columns each for one of the actions  $a$ . The Q-value for each state-action pair stored in the table can be updated iteratively by various algorithms for learning the Q-values.

In particular, if the policy is deterministic with  $\pi(a|s) = 1$  and  $a = \pi(s)$ , then we have

$$P_\pi(s'|s) = P(s'|s, a), \quad r_\pi(s) = r(s, a) \tag{18.23}$$

and the Bellman equations in Eqs. (18.21) and (18.22) become the same:

$$v_\pi(s) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_\pi(s') = q_\pi(s, \pi(s)) \tag{18.24}$$

same as the Bellman equation of an MRP in Eq. (18.8), and can be solved iteratively if  $\gamma < 1$ , same as in Eq. (18.12).

The figure above illustrates the Bellman equations in Eqs. (18.21) and (18.22), showing the bootstrapping of the state value function in the dashed box on the left, and that of the action value function in dashed box on the right. Specifically

here the word bootstrapping means the iterative method that updates the estimated value at a state  $s$  based on the estimated value of the next state  $s'$ . After taking one of the actions  $a \in A(s)$  available in state  $s$  based on policy  $\pi(a|s)$ , an immediate reward  $r(s, a)$  is received. However, which state the MDP will transit into as the next state  $s'$  is random depending on  $P(s'|s, a)$  of the environment but independent of the policy. This uncertainty is represented graphically by a dashed circle called a *chance node* associated with the action value  $q_\pi(s, a)$  as the sum of the immediate reward  $r(s, a)$  and the expected value of the next state, the weighted average of values  $v(s')$  of all possible state  $s' \in S$ .

### 18.3 Model-Based Planning

Given a complete MDP model in terms of  $p(s', r|s, a)$  of the environment, the goal of model-based planning is to find the optimal policy  $\pi^*$  that achieves the maximum value:

$$v^*(s) = \max_{\pi} v_{\pi}(s), \quad \pi^*(s) = \arg \max_{\pi} v_{\pi}(s) \geq \pi \quad \forall \pi \quad (18.25)$$

based on the partial order that compares different policies according to the corresponding values:

$$\begin{aligned} \text{If} \quad & v_{\pi_1}(s) \geq v_{\pi_2}(s) \geq \dots \geq v_{\pi_k}(s), \quad \forall s \in S \\ \text{Then} \quad & \pi_1 \geq \pi_2 \geq \dots \geq \pi_k \end{aligned} \quad (18.26)$$

The problem of policy optimization can be addressed based on two subproblems: *policy evaluation* to measure how good a policy is, and *policy improvement* to get a better policy for higher values.

One way to solve this optimization problem is to use brute-force search to enumerate all  $|A(s)|$  available policies at each of the  $N = |S|$  states  $s \in S$ . If we assume the numbers of available actions in all states are the same for simplicity, i.e.,  $A(s_1) = \dots = A(s_N) = A$ , then the computational complexity of this method is  $O(|A|^{|S|})$ , likely to be too high for such a brute-force search to be practical if  $N$  is large.

A more efficient and therefore more practical way to optimize the policy is to do it iteratively. Consider a simpler task of improving upon an existing deterministic policy  $\pi$  at a single transition step from the current state  $s$  to a next state  $s'$ . Instead of taking the action  $a = \pi(s)$  dictated by the policy, we can improve it by taking the *greedy action* that maximizes the action value function in Eq. (18.22), while subsequent steps still follow the old policy  $\pi$ . Now we get a new policy:

$$\pi'(s) = \arg \max_a q_{\pi}(s, a) = \arg \max_a \left[ r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_{\pi}(s') \right] \quad (18.27)$$



so that the action value is higher than that of the old one:

$$q_\pi(s, \pi'(s)) = \max_a q_\pi(s, a) = \max_a \left[ r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_\pi(s') \right] \geq q_\pi(s, \pi(s)) = v_\pi(s) \quad (18.28)$$

where the last equality is due to Eq. (18.24) for a deterministic policy.

While it is obvious that this greedy method will result in a higher return for the single state transition from  $s$  to  $s'$ , we can further prove the following *policy improvement theorem* stating that following this greedy method in all subsequent states, we will get a new policy  $\pi'$  that achieves higher values than the those of the old one  $\pi$  at all states:

$$v_{\pi'}(s) \geq v_\pi(s), \quad \forall s \in S \quad (18.29)$$

The proof is by recursively applying the greedy method to replace the old policy  $\pi(s)$  by the new one  $\pi'(s)$  for a higher value for each of the subsequent steps one at a time:

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \max_a q_\pi(s, a) = \max_a \left[ r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_\pi(s') \right] \\ &= r(s, \pi'(s)) + \gamma \sum_{s'} P(s'|s, \pi'(s)) v_\pi(s') \\ &\leq r(s, \pi'(s)) + \gamma \sum_{s'} P(s'|s, \pi'(s)) \max_{a'} q_\pi(s', a') \\ &= r(s, \pi'(s)) + \gamma \sum_{s'} P(s'|s, \pi'(s)) \left[ r(s', \pi'(s')) + \gamma \sum_{s''} P(s''|s', \pi'(s')) v_\pi(s'') \right] \\ &= r(s, \pi'(s)) + \gamma r(s', \pi'(s')) + \gamma \sum_{s'} P(s'|s, \pi'(s)) \left[ \gamma \sum_{s''} P(s''|s', \pi'(s')) v_\pi(s'') \right] \\ &\leq \dots \\ &= r(s, \pi'(s)) + \gamma r(s', \pi'(s')) + \gamma^2 r(s'', \pi'(s'')) + \dots \\ &= v_{\pi'}(s) \end{aligned} \quad (18.30)$$

i.e.,  $\pi'(s) \geq \pi(s)$ .

The optimal state value  $v^*(s)$  and action value  $q^*(s, a)$  can therefore be achieved by replacing the weighted average over all possible actions at each state ( $s$  and  $s'$ ) in the Bellman equations in Eqs. (18.21) and (18.22) by their maximum:

$$v^*(s) = \max_a q^*(s, a) = \max_a \left( r(s, a) + \gamma \sum_{s'} P(s'|s, a) v^*(s') \right) \quad (18.31)$$

and

$$q^*(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) v^*(s') = r(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} q^*(s', a') \quad (18.32)$$

Eqs. (18.31) and (18.32) are the *Bellman optimality equations*, which should hold for all  $N$  states in  $S = \{s_1, \dots, s_N\}$  and can be written in vector form:

$$\mathbf{v}^* = \max_a (\mathbf{r}_a + \gamma \mathbf{P}_a \mathbf{v}^*) \quad (18.33)$$

where

$$\mathbf{v}^* = \begin{bmatrix} v^*(s_1) \\ \vdots \\ v^*(s_N) \end{bmatrix}, \quad \mathbf{r}_a = \begin{bmatrix} r(s_1, a) \\ \vdots \\ r(s_N, a) \end{bmatrix}, \quad \mathbf{P}_a = \begin{bmatrix} P(s_1|s_1, a) & \cdots & P(s_N|s_1, a) \\ \vdots & \ddots & \vdots \\ P(s_1|s_N, a) & \cdots & P(s_N|s_N, a) \end{bmatrix} \quad (18.34)$$

This nonlinear equation system can be solved iteratively to for  $\mathbf{v}^*$ :

$$\mathbf{v}_{n+1}^* = \max_a (\mathbf{r}_a + \gamma \mathbf{P}_a \mathbf{v}_n^*) = B(\mathbf{v}_n) \quad (18.35)$$

This iteration is to be used for policy evaluation in the algorithm *value iteration* below for finding the optimal policy. This iteration will converge as function  $B(\mathbf{v})$  is a contraction mapping with  $\gamma < 1$ :

$$\begin{aligned} \|B(\mathbf{v}_i) - B(\mathbf{v}_j)\| &= \left\| \max_{a_i} (\mathbf{r}_{a_i} + \gamma \mathbf{P}_{a_i} \mathbf{v}_i) - \max_{a_j} (\mathbf{r}_{a_j} + \gamma \mathbf{P}_{a_j} \mathbf{v}_j) \right\| \\ &\leq \left\| \max_{a_i} (\mathbf{r}_{a_i} + \gamma \mathbf{P}_{a_i} \mathbf{v}_i - \mathbf{r}_{a_i} - \gamma \mathbf{P}_{a_i} \mathbf{v}_j) \right\| \\ &= \max_{a_i} \gamma \|\mathbf{P}_{a_i} (\mathbf{v}_i - \mathbf{v}_j)\| \leq \max_{a_i} \gamma \|\mathbf{P}_{a_i}\| \|\mathbf{v}_i - \mathbf{v}_j\| \\ &= \gamma \|\mathbf{v}_i - \mathbf{v}_j\| < \|\mathbf{v}_i - \mathbf{v}_j\| \end{aligned} \quad (18.36)$$

where  $\|\mathbf{P}\| = \|\mathbf{P}\|_\infty = 1$  is the p-norm ( $p = \infty$ ) of a stochastic matrix, which is known to be 1.

Given the complete information of the MDP model in terms of  $p(s', r|s, a)$ , we can find the optimal policy  $\pi^*$  using either *policy iteration (PI)* or *value iteration (VI)* based on the DP method.

#### • Policy Iteration

Carry out the following two processes iteratively from some arbitrary initial policy  $\pi$  until convergence when  $\pi$  no longer changes:

– Policy evaluation:

Given a deterministic policy  $\pi$ , Eq. (18.21) becomes the Bellman equation in Eq. (18.24), a linear equation system of  $N = |S|$  equations of  $N$  unknowns  $v_\pi(s_1), \dots, v_\pi(s_N)$ , which can be solved iteratively as in Eq. (18.12) to find values at all states:

$$v_{n+1}^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) v_n^\pi(s') \quad \forall s \in S$$

– Policy improvement:

Given  $v_\pi(s)$  based on policy  $\pi$ , get a better policy  $\pi'$  by the greedy method as in Eq. (18.27):

$$\pi'(s) = \arg \max_a [r(s, a) + \gamma \sum_{s'} P(s'|s, \pi(s)) v_\pi(s')] \quad \forall s \in S$$

We see that in each round of the policy iteration, two intertwined and interacting processes, the policy evaluation and policy improvement, are carried out alternatively, one depending on the other, and one completing

before the other starts. The value functions  $v_\pi(s)$  at all states are systematically evaluated before they are updated, and the policy  $\pi(a|s)$  at all states is updated before it is evaluated. While these two processes chase each other as moving target iteratively and eventually converge to the optimal policy  $\pi^*$  achieving the optimal value  $v^*$ . This approach is said to be *synchronous*, as shown here:

$$\pi_o \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} v_{\pi^*} = v^* \quad (18.37)$$

The pseudo code for the algorithm is listed below:

Initialize:

$v(s) \in \mathbb{R}$ ,  $v(s) = 0$  if  $s$  is terminal node,  $\pi(s) \in A(s)$ ,  $\forall s \in S$ ;  
Set tolerance  $tol$  (small positive value);

**repeat**

Policy Evaluation (find  $v(s) \forall s \in S$ ):

**repeat**

$\delta = 0$

**for**  $\forall s \in S$ :

$v'(s) = r(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s))v(s')$

$\delta = \max(\delta, |v(s) - v'(s)|)$

**until**  $\delta < tol$

Policy Improvement (by taking greedy action):

$Done = T$

**for** each  $s \in S$

$\pi'(s) = \arg \max_a [r(s, a) + \gamma \sum_{s'} P(s'|s, \pi(s))v(s')]$

**if**  $\pi'(s) \neq \pi(s)$ ,  $Done = F$

**end for**

**until**  $done$

• **Value Iteration:**

One drawback of the policy iteration method is the high computational complexity of the inner iteration for policy evaluation inside the outer iteration for policy evaluation. To speed up this process, we could terminate the inner iteration early before convergence. In the extreme case, we simply combine policy evaluation and improvement into a single step, resulting in the following value iteration contains two steps:

– Find the optimal values:

Based on some initial value  $v_0(s)$ , solve the Bellman optimality equation in Eq. (18.31) by iteration in Eq. (18.35):

$$v_{n+1}(s) = \max_a \left[ r(s, a) + \gamma \sum_{s'} P(s'|s, a)v_n(s') \right] \quad \forall s \in S \quad (18.38)$$

– Find the optimal policy:

Find  $\pi^*(s)$  that maximizes  $v^*(s)$ :

$$\pi^*(s) = \arg \max_a \left[ r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_n(s') \right] \quad \forall s \in S \quad (18.39)$$

Here is the pseudo code for the algorithm:

Initialize:  $v(s) = 0 \quad \forall s \in S$

**repeat**

$\delta = 0$ ;

**for** each  $s \in S$

$v'(s) = \max_a [r(s, a) + \gamma \sum_{s'} P(s'|s, a) v(s')]$

$\delta = \max(\delta, |v'(s) - v(s)|)$

$v(s) = v'(s)$

**end for**

**until**  $\delta < tol$

$\pi^*(s) = \arg \max_a [r(s, a) + \gamma \sum_{s'} P(s'|s, a) v(s')]$

In the PI method considered above, the value functions at all states are evaluated before the policy is updated for all states, as illustrated below:

This synchronous policy iteration can be modified so that the values at some or even one of the states are evaluated in-place before other state values are updated, and the policy at some or even one of the states are updated in-place before that at other states, resulting in an *asynchronous* method called *general policy iteration (GPI)*, to be discussed in detail in the next sections on model-free control.

## 18.4 Model-Free Evaluation and Control

The previously discussed dynamic programming methods find the optimal policy based on the assumption that the MDP model of the environment is completely available, i.e., the dynamics of MDP in terms of its state transition and reward mechanism are known. A given policy  $\pi$  can then be evaluated based the transition probabilities  $p(s'|s, a)$ , and improved based on greedy action selection at each state. Such a model-based optimization problem, called *planning*, involves no learning of the environment.

Now we consider the optimization problem with an unknown MDP model of the environment. In this model-free problem, called *control*, the state transition and reward probabilities of the MDP model are unknown, the value functions can no longer be calculated directly based on Eqs. (18.21) and (18.22) as in model-based planning. Instead, now we need to learn the MDP model by sampling the environment, i.e., by repeatedly running a large number of episodes of the stochastic process of the environment while following some given policy, and estimate the value functions as the average of the actual returns received during

the sampling process. This method is generally referred to as *Monte Carlo (MC) method*. At the same time, the given policy can also be improved to gradually approach optimality.

This approach is called *general policy iteration (GPI)*, as illustrated in Eq. (18.40) below, similar to the *policy iteration (PI)* algorithm for model-based planning illustrated in Eq. (18.37). In general, all model-free control algorithms are based on GPI by which the two alternating processes of policy evaluation and policy improvement are carried out iteratively. This GPI can also be similarly illustrated in Fig. ?? for the GI.

$$\pi_o \xrightarrow{E} q_{\pi_o} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} q_{\pi^*} = v^* \quad (18.40)$$

While GPI and GI illustrated in respectively in Eqs. (18.40) and (18.37) may look similar to each other, there are some essential differences between the two, as listed below.

- First, in model-free control the policy improvement is based on the action-value function  $q_\pi(s, a)$ , instead of the state value function  $v_\pi(s)$  in model-based planning. This is because without the a specific MDP model the state value function can no longer be calculated, while the action value function can still be estimated based on the actual rewards received while sampling the environment. Similar to how we improve the policy by taking a greedy action to achieve a higher state value in model-based planning, here we improve the policy by taking an action different from that dictated by the given policy  $\pi$  to achieve a higher action value based on some  $\epsilon$ -greedy method, as discussed below.
- Second, as estimating the action value by the MC method requires running a large number of episodes while the sampling of the environment, the computational complexity for policy evaluation in the model-free case is much higher than that in model-based planning. To speed up this process in model-free control, the iterative policy evaluation is no longer fully carried for it to converge to the true action value of the policy. Instead, in model-free control, the iterative update of the action value is carried out only once, followed immediately by the next phase of policy improvement. Such an estimated action value is inevitably very noisy, but due to the much reduced complexity, we can afford to run a large number of sample episodes while sampling the environment.
- Third, in the PI process of model-based planning, the action value is found based on all available actions at each state (Eq. (18.22)) during policy evaluation, and we only need to *exploit* the greedy action at each state to find the policy that achieves the maximum action value  $q_\pi(s)$  (Eq. (18.27)) during policy improvement. However, here in model-free control, we have to learn the action value  $q_\pi(s, a)$ , as a function of action  $a \in A(s)$  as well as state  $s$  by sampling the environment. Such an estimated action value based

only on some partial sample data is inevitably noisy, especially in the early stage of learning when many of the states have not yet been visited yet. We therefore need to *explore* all actions at each state to better learn the value function, as well as to exploit the greedy action to improve the policy.

This issue of *exploration* versus *exploitation* can be addressed by the  $\epsilon$ -greedy method to make a proper trade-off between the exploitation of the greedy action and the exploration of other non-greedy actions, so that the agent can be exposed to all possible state-action pairs and gradually learn the action value function while at the same time also improve the policy being followed. In this method, we define  $\epsilon \in [0, 1]$  as the probability to explore any randomly chosen action  $a \in |A(s)|$  out of all actions available in state  $s$ , each with an equal probability  $\epsilon/|A(s)|$ , and  $1 - \epsilon$  is the probability of taking the greedy action as one of the  $|A(s)|$  actions, with probability  $1 - \epsilon + \epsilon/|A(s)|$ .

$$\pi'(a|s) = \begin{cases} \arg \max_a q(s, a) & Pr = 1 - \epsilon + \epsilon/|A(s)| \\ \text{any } a \in A(s) & Pr = \epsilon/|A(s)| \end{cases} \quad (18.41)$$

For example, if  $|A(s)| = 4$  and  $\epsilon = 2/3$ , the greedy action may be chosen with probability  $1 - \epsilon + \epsilon/|A(s)| = 3/6$ , while each of the remaining three non-greedy actions may be chosen with probability  $\epsilon/|A(s)| = 1/6$ . Such a policy describes the behavior of the agent, and is therefore called *behavior policy*, different from the policy being followed.

We further note that a larger value of  $\epsilon$  (close to 1) can be used to emphasize exploration at the early stage of the iteration when many state-action pairs have not been visited yet, while progressively smaller  $\epsilon$  values (approaching to 0) can be used to emphasize exploitation for higher values in the later stage of the iteration when the action value function has been more reliably learned. For example, we can let  $\epsilon_k = 1/k$  be inversely proportional to the current number of iterations  $k$ , so that the larger  $\epsilon$  values at the early stage of the iteration allow more state-action pairs to be explored, while smaller  $\epsilon$  values in the later stage of the iteration encourage exploitation of the greedy actions. To the limit  $k \rightarrow \infty$  and  $\epsilon \rightarrow 0$ , the  $\epsilon$ -greedy method approaches absolute greedy method.

Similar to how we proved the policy improvement theorem in Eq. (18.30) stating that the greedy policy  $\pi'(s)$  achieves state value  $v_{\pi'}(s)$  no lower than  $v_{\pi}(s)$  achieved by the original policy  $\pi$ , i.e.,  $\pi'(s) \geq \pi(s)$ , here we can also prove the same policy improvement theorem stating that the  $\epsilon$ -greedy policy  $\pi'$  achieves action value  $q_{\pi'}(s, a)$  no lower than  $q_{\pi}(s, a)$  by the original policy  $\pi$ :

$$\begin{aligned} q_{\pi}(s, \pi'(s)) &= \sum_a \pi'(a|s) q_{\pi}(s, a) \\ &= \frac{\epsilon}{|A(s)|} \sum_a q_{\pi}(s, a) + (1 - \epsilon) \max_a q_{\pi}(s, a) \\ &\geq \frac{\epsilon}{|A(s)|} \sum_a q_{\pi}(s, a) + (1 - \epsilon) \sum_a \frac{\pi(a|s) - \epsilon/|A(s)|}{1 - \epsilon} q_{\pi}(s, a) \end{aligned} \quad (18.42)$$

where the inequality is due to the fact that the maximum value of  $q_\pi(s, a)$  is no less than the average of action values  $q_\pi(s, a)$  over all  $a \in A(s)$  weighted by some normalized coefficients adding up to 1:

$$\sum_a \frac{\pi(a|s) - \epsilon/|A(s)|}{1 - \epsilon} = \frac{1}{1 - \epsilon} \sum_a \left( \pi(a|s) - \frac{\epsilon}{|A(s)|} \right) = \frac{1}{1 - \epsilon} (1 - \epsilon) = 1 \quad (18.43)$$

Continuing the equation above we further get

$$\begin{aligned} q_\pi(s, \pi'(s)) &\geq \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s, a) - \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s, a) + \sum_a \pi(a|s) q_\pi(s, a) \\ &= \sum_a \pi(a|s) q_\pi(s, a) = v_\pi(s) \end{aligned} \quad (18.44)$$

The iterative process of the GPI method for model-free control is illustrated in the figure below, similar to the PI method for model-based planning based planning illustrated in Fig. ??, but with the state value  $v_\pi(s)$  replaced by the action value  $q_\pi(s, a)$ , and the greedy method replaced by  $\epsilon$ -greedy method.

Also, here the estimated action value  $\tilde{q}$  is obtained with only one iteration for policy evaluation, instead of a much more accurate estimate that could otherwise be obtained if the iterative evaluation was fully carried out to its convergence (the top line). At the same time, based on the estimated  $\tilde{q}$  policy improvement is carried out by the  $\epsilon$ -greedy method (the bottom line).

The main task in model-free control is to evaluate the state and action-value functions given in Eqs. (18.21) and (18.22) while following a given policy  $\pi$  without a specific MDP model. This is done by sampling of the dynamic process of the environment, and then estimating the value functions as the average of the actual returns received by the agent. Specifically, we run a large number of episodes (all assumed to terminate) of the MDP model of the environment by taking actions  $a_t = \pi(s_t)$  in each state  $s_t$ , ( $t = 1, \dots, T$ ):

$$s_0 \rightarrow (a_1, r_1, s_1) \rightarrow (a_2, r_2, s_2) \rightarrow \dots \rightarrow (a_T, r_T, s_T) \quad (18.45)$$

and estimate the value functions based on the averaged returns  $G_t$  actually received from these episodes.

Such a sequence of states visited is called a *trajectory*, and the trajectories of different episodes are in general different from each another due to the random nature of the environment. In the following sections we will consider specific algorithms for the implementation of the general model-free control discussed above.

To prepare for specific discussion of the GPI methods, we first consider a general problem of the estimation of the value of a random variable  $x$  as the running average of its samples being collected in real time. Specifically the average  $\hat{x}_n$  based on previous  $n$  samples  $x_1, \dots, x_n$  is updated incrementally upon receiving

a new sample  $x_{n+1}$ :

$$\begin{aligned}\hat{x}_{n+1} &= \frac{1}{n+1} \sum_{i=1}^{n+1} x_i = \frac{1}{n+1} \left( x_{n+1} + \sum_{i=1}^n x_i \right) \\ &= \frac{1}{n+1} (x_{n+1} + n\hat{x}_n) = \hat{x}_n + \frac{1}{n+1} (x_{n+1} - \hat{x}_n)\end{aligned}\quad (18.46)$$

This running average can be generalized to

$$\hat{x}_{n+1} = \hat{x}_n + \alpha(x_{n+1} - \hat{x}_n) \quad (18.47)$$

This equation can be interpreted as

$$\begin{aligned}\text{newEstimate} &= \text{oldEstimate} + \text{stepSize} \times (\text{target} - \text{oldEstimate}) \\ &= \text{oldEstimate} + \text{stepSize} \times \text{error} \\ &= \text{oldEstimate} + \text{increment}\end{aligned}\quad (18.48)$$

and considered as one iterative step in the method of *stochastic gradient descent* (*SGD*), for minimizing the squared error  $\varepsilon = (x_{n+1} - \hat{x}_n)^2/2$  between the old average  $\hat{x}_n$  and the latest sample  $x_{n+1}$ , and the second term is the gradient  $d\varepsilon/d\hat{x}_n$  weighted by the step size  $\alpha \in [0, 1]$ , which controls how samples are weighted differently. In the extreme case when  $\alpha = 1$ ,  $\hat{x}_n = x_n$  with no contribution from any previous samples; on the other hand when  $\alpha$  is close to 0,  $\hat{x}_n$ , the most recent samples has little contribution. We can therefore properly adjust  $\alpha$  to fit our specific need. For example, if we gradually reduce  $\alpha$ , then the estimated average will become stabilized while enough samples have been collected. On the other hand, if we let  $0 < \alpha < 1/(n+1)$ , the more recent samples are weighted more heavily than the earlier ones. Such a strategy is suitable when estimating parameters of a nonstationary system, such as an MDP with varying behaviors when the policy is being modified continuously.

Specifically in model-free control Eq. (18.47) can be used to iteratively update the estimated value functions while sampling the environment. Both the state and action values as the expected return are estimated as the average of all previous sample returns, and they are updated incrementally when a new sample return  $G_{n+1}$ , the target, becomes available:

$$\begin{aligned}v_{n+1}(s) &\Leftarrow v_n(s) + \alpha(G_n - v_n(s)) \\ q_{n+1}(s, a) &\Leftarrow q_n(s, a) + \alpha(G_n - q_n(s, a))\end{aligned}\quad (18.49)$$

as we will see in the following sections.

### 18.4.1 Monte Carlo (MC) Algorithms

We first consider a simple problem of evaluating an existing policy  $\pi$  in terms of its value function  $v_\pi(s) = E[G_t]$  at state  $s = s_t$ , the expectation of return  $G_t$ ,



which can be obtained in the model-based case by the Bellman equation in Eq. (18.24):

$$v_\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) v_\pi(s') \quad (18.50)$$

based on  $v_\pi(s')$  of all possible next states each weighted by the corresponding transition probability  $p(s'|s, a)$ , which is no longer available now in the model-free case. We can still find the value function based on the MC method by running multiple sample episodes in the environment, and estimate  $v_\pi(s)$  as the average of the actual returns  $G_t$ , which in turn can be calculated as the sum of discounted rewards  $r_{t+1}, \dots, r_T$  from the current state  $s = s_t$  onward to the terminal state  $s_T$ , found at the end of the episode (assumed to have finite horizon).

This method has two versions, *first-visit* and *every-visit*, depending on whether only the first or every visit to a state in the trajectory of an episode is counted in calculation of the return. The pseudo code below is for first-visit version due to the if statement, which can be removed for the every-visit version.

Input: a given policy  $\pi$  to be evaluated

Initialize:  $G_t = [ ]$  (empty lists) for all  $t = 0, \dots, T - 1$

**loop** (for each episode)

run episode to the end while following  $\pi$  to get sample data:  $(s_0, a_0, r_1), (s_1, a_1, r_2), \dots, (s_{T-1}, a_{T-1}, r_T)$

$G = 0$

**for**  $t = T - 1, T - 2, \dots, 0$  (for each step)

$G = \gamma G + r_{t+1}$

**if**  $s_t \notin \{s_0, \dots, s_{t-1}\}$  (first visit)

$G_t = [G_t, G]$  (append  $G$  to list  $G_t$ )

$V(s_t) = \text{average}(G_t), \forall t = 0, \dots, T - 1$

In the every-visit case, all sample returns  $G_t$  from multiple visits to state  $s_t$  are used, although they may be correlated and not independent of each other, i.e., they are not necessarily i.i.d. samples and their average may be biased. This problem can be avoided in the first-visit case, where only the returns of the first visit to each state are counted as independent samples drawn from the same distribution, and their average is not biased. However, as only a fraction of the sample points collected is used in the calculation of the returns, the cost of this unbiased estimation is its high variance, which can be reduced only if a large number of episodes are used to get enough samples for a more statistically reliable average. This is the typical trade-off between variance and bias errors.

In the code above, the function *average* finds the average of all elements of a list, only when all sample returns are available at the end of each episode. A more computationally efficient method (in terms of both space and complexity) is to find the average incrementally as in Eq. (18.49). Based on this incremental average, policy evaluation in terms of both the state and action values can also be carried out by the code below:

Initialize  $v(s) = 0, \forall s$

```

loop (for each episode)
  run episode following  $\pi$  to get  $G_t, \forall t$ 
  for  $t = 0, \dots, T - 1$  (for each step)
    if  $s_t$  is visited the first time
       $v(s_t) = v(s_t) + \alpha(G_t - v(s_t))$ 

```

and

```

Initialize:  $q(s, a) = 0, \forall (s, a)$ 
loop (for each episode)
  run episode following  $\pi$  to get  $G_t, \forall t$ 
  for all  $(s, a)$  pairs visited
    if  $(s, a)$  is visited the first time
       $q(s, a) = q(s, a) + \alpha(G_t - q(s, a))$ 

```

The *if* condition can be removed for every-visit version of the algorithm.

We next consider the MC algorithm for model-free control, based on the generalized policy iteration of alternating and interacting policy evaluation and policy improvement. Specifically, while sampling the environment by following an existing policy  $\pi$  the action value  $q_\pi(s)$ , instead of the state value  $v_\pi(s)$ , is gradually learned and the policy is gradually improved at the same time. This iteration will converge to the optimal policy at the limit when the number of iterations goes to infinity.

The pseudo code for this algorithm is listed below.

```

Input:  $\epsilon > 0, \alpha > 0$ , and a policy  $\pi$ 
Initialize:  $q(s, a) = 0, \forall (s, a), \pi = \epsilon - \text{soft}$  (arbitrarily)
loop (for each episode)
  run episode following  $\pi$  to get  $G_0, \dots, G_{T-1}$ 
  for  $t = 0, \dots, T - 1$ 
    if  $(s_t, a_t)$  is visited the first time
       $q(s_t, a_t) = q(s_t, a_t) + \alpha(G_t - q(s_t, a_t))$ 
       $a^* = \arg \max_a q(s_t, a_t)$ 
      for  $\forall a \in A(s_t)$ 
        
$$\pi(a|s_t) = \begin{cases} 1 - \epsilon + \epsilon/|A(s_t)| & \text{if } a = a^* \\ \epsilon/|A(s_t)| & \text{else} \end{cases}$$


```

Here the  $\epsilon$ -greedy policy  $\pi(a|s)$  is based on action value function  $q(s, a)$  (maximized by the greedy action). Through out the iteration, the policy is modified following the action value function, while at the same time the action value function is modified based on the policy, until the process converges to the desired optimality.

We note that the optimal policy  $\pi(s)$  obtained by the algorithm above is not completely deterministic due to the  $\epsilon$ -greedy approach, but it is said to be near-deterministic as the greedy action is favored over other non-greedy actions.

## 18.4.2 Temporal Difference (TD) Algorithms

The temporal difference (TD) method is a combination of the MC method considered above and the bootstrapping DP method based on the Bellman equation. The main difference between the TD and MC methods is the target, the return  $G$ , in the incremental average in Eq. (18.49) for estimating either the state or action value functions. While in the MC method the target is the actual return  $G_t$  calculated at the end of the episode when all subsequent rewards  $r_{t+1}, \dots, r_T$  are available, here in the TD method the target, called *TD target*, is the sum of the immediate reward  $r_{t+1}$  and the previously estimated value  $v_\pi(s_{t+1})$  at the next state:

$$G_t = r_{t+1} + \gamma v_\pi(s_{t+1}) \quad (18.51)$$

We see that the TD method is an in-place bootstrapping method, and it makes more efficient use of the sample data and updates more frequently the value functions being estimated and the policy being improved at every step of an episode, instead of at the end of the episode as in the MC method. Also, it can be used even if the dynamic process of the environment is non-episodic with infinite horizon.

Again, we first consider the simpler problem of policy evaluation. As in the MC method, we estimate the value function  $v_\pi(s)$  as the average of the actual returns  $G$  found by running multiple episodes while sampling the environment. Substituting this TD target into the running average in Eq. (18.49) for the value function above, we get

$$\begin{aligned} v_\pi(s) &\leftarrow v_\pi(s) + \alpha(G - v_\pi(s)) \\ &= v_\pi(s) + \alpha(r + \gamma v_\pi(s') - v_\pi(s)) = v_\pi(s) + \alpha\delta \end{aligned} \quad (18.52)$$

where  $\delta$ , called the *TD error*, is the difference between the TD target and the previous estimate:

$$\delta = G - v_\pi(s) = r + \gamma v_\pi(s') - v_\pi(s) \quad (18.53)$$

and  $v_\pi(s')$  is the current estimate of the value of the next state (bootstrapping). It can be shown that the iteration of this TD method converges to the true value  $v_\pi(s)$ , if the step size is small enough.

Here is the pseudo code for policy evaluation using the TD method, based on parameters  $\gamma$  and  $\alpha$ :

```

Input: a given policy  $\pi$  to be evaluated
Initialize:  $v(s)$  for all  $s \in S$  arbitrarily,  $v(s) = 0$  for terminal state  $s$ ,  $\alpha \in (0, 1]$ 
loop (for each episode)
     $s = s_0$ 
    while  $s$  is not terminal (for each step)
        take action  $a = \pi(s)$ , get reward  $r$  and next state  $s'$ 
         $v(s) = v(s) + \alpha[r + \gamma v(s') - v(s)]$ 
         $s = s'$ 

```

As the TD algorithm updates the value function at every step of the episode, it uses the sample data more frequently and efficiently, and therefore has lower variance error than the CM method that updates the estimated value function at the end of each episode. On the other hand, the TD method may be biased when compared to the first-visit MC method, due to the arbitrary initialization of the value functions.

Based on the TD method for model-free policy evaluation, we now further consider the TD method for model-free control to gradually learn the optimal policy by updating the Q-values of all state-action pairs  $(s, a)$  estimated iteratively as the running average of the sample Q-values at each time step of an episode while sampling the environment.

The Q-values for all state-action pairs can be stored in a state-action table which is iteratively updated by running many episodes of the unknown MDP by the TD method to gradually approach the maximum Q-values achievable by taking each action at each state, i.e., the optimal action is the one with the highest Q-value.

This method has two different flavors, the *on-policy* algorithm, which updates the Q-value by following the current policy such as an  $\epsilon$ -greedy policy, and the *off-policy* algorithm, which updates the Q-value by taking actions different from the current policy such as the greedy action. In particular, if the policy currently being followed is greedy (instead of  $\epsilon$ -greedy), the two algorithms are the same.

- **State-Action-Reward-State-Action (SARSA)**

The Q-value  $q(s, a)$  of each state-action pair  $(s, a)$  is estimated as the running average of the return  $G$ , the sum of the immediate reward  $r$  and the action value  $q(s', a')$  of the next state based on action  $a'$  dictated by the current policy (e.g.,  $\epsilon$ -greedy):

$$\begin{aligned} q(s, a) &\Leftarrow q(s, a) + \alpha(G - q(s, a)) \\ &= q(s, a) + \alpha(r + \gamma q(s', a') - q(s, a)) \end{aligned} \quad (18.54)$$

This algorithm is called *SARSA*, as it updates the Q-value based on the current state  $s$  and action  $a$ , the immediate reward  $r$ , and the next state  $s'$  and action  $a'$ . The pseudo code of SARSA algorithm is listed below.

Initialize:  $q(s, a) = 0$  for all  $(s, a)$ ,  $\alpha \in (0, 1]$ ,  $\epsilon > 0$ , denote  $\epsilon$ -greedy policy

$\pi(a|s)$  by  $\pi$

**loop** (for each episode)

$s = s_0$

get action  $a$  according to  $\pi$  based on  $q(s, a)$

**while**  $s$  is not terminal (for each step)

take action  $a$ , get reward  $r$  and next state  $s'$

get action  $a'$  according to  $\pi$  based on  $q(s', a)$

$q(s, a) = q(s, a) + \alpha[r + \gamma q(s', a') - q(s, a)]$

$s = s', a = a'$

As a variation of SARSA, the *expected SARSA* updates the Q-value at state  $s$  based on the expected Q-value of the next state  $s'$ , the weighted average of the Q-values resulting from all possible actions, instead of only one action. Consequently, the estimated Q-values by the expected SARSA have lower variance than SARSA, and a higher learning rate  $\alpha$  can be used to speed up the learning process.

$$q(s, a) = q(s, a) + \alpha \left( r + \gamma \sum_a \pi(a|s') q(s', a) - q(s, a) \right) \quad (18.55)$$

- **Q-Learning**

Same as SARSA, the Q-learning algorithm also estimates the Q-value  $q(s, a)$  of each state-action pair  $(s, a)$  as the running average of the return  $G$ , the sum of the immediate reward  $r$  and the action value  $q(s', a')$  of the next state based on the greedy action  $a'$  that maximizes the next state value  $q(s', a')$ , different from that dictated by the current policy.

$$q(s, a) \leftarrow q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - q(s, a)) \quad (18.56)$$

The pseudo code of SARSA algorithm is listed below.

Initialize:  $q(s, a) \quad \forall s \in S, \forall a \in A(s)$  arbitrarily ( $q(s, a) = 0$  for terminal  $s$ ),  
 $\alpha \in (0, 1], \epsilon > 0$

**loop** (for each episode)

    Initialize state  $s$

**while**  $s$  is not terminal (for each step)

        take action  $a$  according to  $\pi$  based on  $q(s, a)$ , get reward  
          $r$  and next state  $s'$

$q(s, a) = q(s, a) + \alpha [r + \gamma \max_{a'} q(s', a') - Q(s, a)]$

$s = s'$

Here is the comparison of the MC and TD methods in terms of their pros and cons:

- The MC method estimates the state value  $v_\pi(s_t)$ , the expected return, by the true return  $G_t$  obtained at the end of the episode, i.e., the estimated value is updated once every episode;

The TD method estimates  $v_\pi(s_t)$  as the sum of the immediate reward  $r_{t+1}$  and the estimated state value at the next state  $v_\pi(s')$ , i.e., it is a bootstrap method, and the estimated value is updated at every step of every episode.

- MC can only learn episodic (terminating) environments with complete episodes; while TD can learn continuing (non-terminating) environment of incomplete episodes.

- MC estimates  $v_\pi(s_t)$  is based on sample returns  $G_t$ , and it is unbiased, while TD uses the bootstrap approach to find the TD target based on sample data that are not necessarily i.i.d., and it is more sensitive to the initial guess of the value functions, it is more biased.
- MC is based on the sample returns  $G_t$  affected by many random events (state transitions, actions, and rewards), and in particular the first-visit version of the MC method only makes use of the sample data from the first visit of a state, it does not use the available data efficiently and it has high variance, while on the other hand the TD method is based on only one random variable, the estimated return, and it makes more frequent and efficient use of the sample data, it has lower variance.
- In the MC method, especially the first-visit version of it, the estimated value functions is unbiased; on the other hand, as the TD method is based on the bootstrap strategy and relies more strongly on the initial guess of the value functions being estimated, it tends to be more biased.

Here is a summary of the dynamic programming (DP) method for model-based planning, and the Monte-Carlo (MC) and time difference (TD) methods for model-free control:

### 18.4.3 TD( $\lambda$ ) Algorithm

The MC and TD methods considered previously can be unified by the  $n$ -step TD( $\lambda$ ) method that spans a spectrum of which the MC and TD methods are two special cases at the opposite extremes.

Recall that both MC and TD algorithms updates iteratively the value functions being estimated and the policy being improved based on Eq. (18.49), but they estimate the target  $G_t$  in the equation differently. In an MC algorithm, the target is the actual return  $G_t = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{T-t-1} r_T$ , the sum of discounted rewards of all future steps upto the terminal state  $s_T$ , available only at the end of each episode. On the other hand, in a TD algorithm, the target is  $G_t = r_{t+1} + \gamma v_\pi(s_{t+1})$  the sum of the immediate reward  $r_{t+1}$  available at each step of the episode, and the discounted value of the next state, based on bootstrapping. We therefore see that an MC algorithm updates the value functions and policy once every episode, while a TD algorithm updates once every step in the episode.

As a trade-off between the MC and TD methods, the  $n$ -step TD algorithm can be considered a generalization of the TD method, where the target in Eq. (18.49) is an  $n$ -step return, the sum of the discounted rewards in the  $n$  subsequent states and the discounted value function at the following state  $s_{t+n}$  with  $1 \leq n < \infty$ :

$$G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{n-1} r_{t+n} + \gamma^n v_\pi(s_{t+n}) \quad (18.57)$$

Specially, when

- $n = 1$ : the 1-step return is the sum of the immediate reward and the estimated value of the next state, the same as the TD target in the TD method:

$$G_{t:t+1} = r_{t+1} + \gamma v(s_{t+1}) \quad (18.58)$$

- $n = T - t$ : the  $n$ -step return is the sum of the discounted rewards from all future states upto the terminal state at the end of the episode, i.e., it is the return  $r_t$  defined in Eq. (18.7), the same as in the MC method:

$$G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T + v_\pi(s_{t+n}) = G_t \quad (18.59)$$

where the value function of the terminal state  $v_\pi(s_{t+n}) = v_\pi(s_T) = 0$  is zero.

- $T - t < n < \infty$ : All states  $s_n$  beyond the terminal state  $s_T$  with  $n > T - t$  remain the same as the terminal state with value  $v(s_T) = 0$  and return  $G_t$ , same as in the MC method.

We see that all these  $n$ -step returns form a spectrum with the MC and TD methods at the two ends.

Based on these  $n$ -step returns of different  $n$  values, the  $n$ -step TD algorithm can be further generalized to a more computationally advantageous and therefore more useful algorithm called  $\text{TD}(\lambda)$ , by which the MC and TD algorithms are again unified as two special cases at the opposite ends of a spectrum.

We first define the  $\lambda$ -return  $G_t^\lambda$  as the weighted average of all  $n$ -step returns for  $n = 1, 2, \dots, \infty$ :

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \quad (18.60)$$

where  $0 \leq \lambda < 1$ . Note that the weights decay exponentially, and they are normalized due to the coefficient  $1 - \lambda$ :

$$(1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} = (1 - \lambda) \sum_{n=0}^{\infty} \lambda^n = \frac{1 - \lambda}{1 - \lambda} = 1 \quad (18.61)$$

The  $\lambda$ -return  $G_t^\lambda$  defined above can be expressed in two summations, the sum of the first  $T - t - 1$   $n$ -step returns  $G_{t:t+1}, \dots, G_{t:t+T-1}$ , and the sum of all subsequent  $n$ -step returns  $G_{t:T} = \dots = G_{t:\infty} = G_t$ :

$$\begin{aligned} G_t^\lambda &= (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + (1 - \lambda) \sum_{n=T-t}^{\infty} \lambda^{n-1} G_t \\ &= (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t, \quad (0 \leq t \leq T) \end{aligned} \quad (18.62)$$

where the coefficient of  $G_t$  in the second term is the sum of coefficients in the second summation:

$$(1 - \lambda) \sum_{n=T-t}^{\infty} \lambda^{n-1} = (1 - \lambda) \sum_{m=0}^{\infty} \lambda^m \lambda^{T-t-1} = \lambda^{T-t-1} \quad (18.63)$$

where  $m = n - T + t$ . We see that in Eq. (18.62) all  $n$ -step returns  $G_{t:t+n}$  are weighted by exponentially decaying coefficient  $\lambda^{n-1}$ , except the true return  $G_t$  which is weighted by  $\lambda^{T-t-1}$ .

Again consider two special cases:

- $\lambda = 0$ : all terms in Eq. (18.62) are zero, except the first one with  $n = 1$ :

$$G_t^{\lambda=0} = \lambda^{n-1} G_{t:t+n} \Big|_{n=1} = 0^0 G_{t:t+1} = r_{t+1} + \gamma v(s_{t+1}) \quad (18.64)$$

This is simply the TD target, i.e., TD(0) is the TD method.

- $\lambda = 1$ : the first summation is zero and

$$G_t^{\lambda=1} = G_t \quad (18.65)$$

i.e., TD(1) is the MC method.

We therefore see that TD( $\lambda$ ) is a general algorithm of which the two special cases TD(0) and TD(1) are respectively the TD and MC methods.

In Eq. (18.62),  $G_t^\lambda$  is calculated as the weighted sum of all  $n$ -step returns  $G_{t:t+n}$  upto the last one  $G_t$  available at the end of the episode. This summation can be truncated to include fewer terms before reaching the terminal state at the end of the episode:

$$G_{t:h}^\lambda = (1 - \lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{h-t-1} G_{t:h}, \quad (0 \leq t < h \leq T) \quad (18.66)$$

Specially when  $h = T$ ,  $G_{t:h}^\lambda = G_t^\lambda$  same as before, and when  $h = t + 1$ ,  $G_{t:h}^\lambda = G_{t:t+1} = r_{t+1} + \gamma v(s_{t+1})$ , same as the TD target.

Based on  $\lambda$ -return, the iterative update of the value function  $v_\pi(s_t)$  in Eq. (18.52) can be modified to:

$$v_\pi(s_t) = v_\pi(s_t) + \alpha(G_t^\lambda - v_\pi(s_t)) = v_\pi(s_t) + \alpha\delta_t \quad (18.67)$$

where

$$\delta_t = G_t^\lambda - v_\pi(s_t) \quad (18.68)$$

This iteration is called the *forward view* of TD( $\lambda$ ), which is similar to the MC method, as they are based on either  $G_t$  or  $G_t^\lambda$ , available only at the end of each episode.

An alternative method is the *backward view* of TD( $\lambda$ ), which can be shown to be equivalent to the forward-view of TD( $\lambda$ ), but it is similar to the TD(0) method, as they are both based on the immediate reward  $r_{t+1}$  available at each time step of the episode, and therefore more computationally convenient.

Specifically, the backward view of TD( $\lambda$ ) is based the *eligibility trace*  $e_t(s)$  for each of the states, which decays exponentially upon each state transition

$$e_t(s) = \gamma\lambda e_{t-1}(s) \quad (\forall s \in S) \quad (18.69)$$



but is boosted by an increment 1 to become  $e_t(s) = e_t(s) + 1$  if  $s = s_t$  is currently visited.

Now the iterative update of the estimated value function in Eq. (18.52) is modified so that *all* states, instead of only the one currently visited, are updated, but to different extents based on  $e_t(s)$ :

$$v_\pi(s) = v_\pi(s) + \alpha \delta_t e_t(s), \quad \forall s \in S \quad (18.70)$$

where the TD error  $\delta_t$  is the same as in TD(0), given in Eq. (18.53):

$$\delta_t = r_{t+1} + \gamma v_\pi(s_{t+1}) - v_\pi(s) \quad (18.71)$$

The eligibility trace  $e_t(s)$  as defined above is motivated by the frequency and recency of the visits to each state. If a state  $s$  has been more frequently and recently visited compared to others, its  $e_t(s)$  is greater than others and its value function  $v_\pi(s)$  will be updated by a greater increment than others.

Here is the pseudo code for the backward view of the TD( $\lambda$ ) method for policy evaluation:

```

Input: policy  $\pi$  to be evaluated
Initialize:  $v(s) = 0 \quad \forall s \in S$ 
loop (for each episode)
     $s = s_0$ 
     $e(s) = 0, \forall s \in S$ 
    while  $s$  is not terminal (for each step)
        take action  $a$  based on  $\pi(a|s)$ , get reward  $r$  and next state  $s'$ 
         $\delta = r + \gamma v(s') - v(s)$ 
         $e(s) = e(s) + 1$ 
        for all  $s$ 
             $v(s) = v(s) + \alpha \delta e(s)$ 
             $e(s) = \gamma \lambda e(s)$ 
         $s = s'$ 

```

Note that values at all states are updated, but to different extents depending on  $e(s)$ , different from the TD algorithm where only the value at the state currently visited is updated.

Again consider two special cases:

- $\lambda = 0$ ,  $e(s) = 0$  for all states except the current  $s$  being visited, i.e., TD( $\lambda$ ) becomes TD(0), the same as the TD method.
- $\lambda = 1$ , then  $e(s)$  is scaled down by a factor  $\gamma < 1$  for all states except the current  $s$  being visited, i.e., TD( $\lambda$ ) becomes TD(1), the same as the MC method. However, different from the MC method that updates the value function  $v(s)$  at the end of each episode, here  $v(s)$  is still updated at every step of the episode due to the backward view of the method.

This backward view of the TD( $\lambda$ ) method can be applied to model-free control when the state value function is replaced by the action value function. Corresponding to the SARSA and Q-learning algorithms based on TD(0) in the previous section, here are the two algorithms based on eligibility traces:

- SARSA( $\lambda$ ) algorithm:

Initialize:  $q(s, a) = 0, \forall (s, a), \alpha \in (0, 1], \epsilon > 0$ , denote  $\epsilon$ -greedy policy  $pi(a|s)$  by  $\pi$

**loop** (for each episode)

$s = s_0$

$e(s, a) = 0, \forall (s, a)$

get action  $a$  according to  $\pi$  based on  $q(s, a)$

**while**  $s$  is not terminal (for each step)

$e(s, a) = e(s, a) + 1$

take action  $a$ , get reward  $r$  and next state  $s'$

get action  $a'$  according to  $\pi$  based on  $q(s', a)$

$\delta = r + \gamma q(s', a') - q(s, a)$

**for** all  $(s, a)$

$q(s, a) = q(s, a) + \alpha \delta e(s, a)$

$e(s, a) = \gamma \lambda e(s, a)$

$s = s', a = a'$

- Q( $\lambda$ ) algorithm:

Initialize:  $q(s, a) = 0, \forall (s, a), \alpha \in (0, 1], \epsilon > 0$ , denote  $\epsilon$ -greedy policy  $pi(a|s)$  by  $\pi$

**loop** (for each episode)

$s = s_0$

$e(s, a) = 0, \forall (s, a)$

get action  $a$  according to  $\pi$  based on  $q(s, a)$

**while**  $s$  is not terminal (for each step)

$e(s, a) = e(s, a) + 1$

take action  $a$ , get reward  $r$  and next state  $s'$

get action  $a'$  according to  $\pi$  based on  $q(s', a)$

$a^* = \arg \max_b q(s', b)$

$\delta = r + \gamma q(s', a^*) - q(s, a)$

**for** all  $(s, a)$

$q(s, a) = q(s, a) + \alpha \delta e(s, a)$

**if**  $a' = a^*$  **then**  $e(s, a) = \gamma \lambda e(s, a)$  **else**  $e(s, a) = 0$

$s = s', a = a'$

Note that Q( $\lambda$ ) algorithm is different from the SARSA( $\lambda$ ) algorithm in two ways. First, the action value is updated based on the greedy action  $a^*$ , instead of the  $a'$  by policy  $\pi$ ; second, if the  $\epsilon$ -greedy policy  $\pi$  happens to choose a random non-greedy action  $a' \neq a^*$  with probability  $\epsilon$  to explore rather than exploiting  $a^*$ , the eligibility traces of all states are reset to zero. There are other different versions of the algorithm which do not reset these traces.

## 18.5 Value Function Approximation

All previously considered algorithms are based on either the state values function  $v_\pi(s)$  for each state  $s$  or the action value  $q_\pi(s, a)$  for each state-action pair. As these function values can be considered as either a 1-D or 2-D table, the algorithms based on such function values are called tabular methods. However, when the number of states and the number of actions in each state are large (e.g., the game Go has  $10^{170}$  states), the tabular methods are no longer suitable due to the unrealistic table sizes. In such a case, we can consider approximating the state value function  $v(s)$  or action-value function  $q(s, a)$  by a function  $\hat{v}(s, \mathbf{w})$  or  $\hat{q}(s, a, \mathbf{w})$ , parameterized by a set of parameters as the components of vector  $\mathbf{w}$ . By doing so the state or action value functions can be represented more conveniently in terms of a much smaller number of parameters in  $\mathbf{w}$ .

This approach can be considered as a regression problem to fit a continuous function  $\hat{v}(s, \mathbf{w})$  or  $\hat{q}(s, a, \mathbf{w})$  to a set of discrete and finite data points  $v_\pi(s)$  or  $q_\pi(s, a)$  obtained by sampling the MDP of the environment, so that all points in the state space, including those for states or state-action pairs never actually observed during the sampling process can be represented.

Examples of such value function approximators include

- linear combination of all features
- multi-layer neural networks
- decision trees

Same as in regression problems, we desire to find the optimal parameter  $\mathbf{w}$  for the modeling function  $\hat{v}_\pi(s, \mathbf{w})$  so that the following objective function, the mean squared error between the approximated values and the sample values, is minimized:

$$J(\mathbf{w}) = \frac{1}{2} E_\pi [(v_\pi(s) - \hat{v}(s, \mathbf{w}))^2] \quad (18.72)$$

where the expectation  $E_\pi$  is with respect to all states visited while following some policy  $\pi$ . The gradient vector of  $J(\mathbf{w})$  is

$$\begin{aligned} \nabla J(\mathbf{w}) &= \frac{d}{d\mathbf{w}} J(\mathbf{w}) = \frac{1}{2} E_\pi \left[ \frac{d}{d\mathbf{w}} [(v_\pi(s) - \hat{v}(s, \mathbf{w}))^2] \right] \\ &= -E_\pi [(v_\pi(s) - \hat{v}(s, \mathbf{w})) \nabla \hat{v}(s, \mathbf{w})] \end{aligned} \quad (18.73)$$

and the optimal parameter vector  $\mathbf{w}^*$  can be found iteratively by the gradient descent method

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \Delta \mathbf{w} = \mathbf{w}_n - \alpha \nabla J(\mathbf{w}) = \mathbf{w}_n + \alpha E_\pi [(v_\pi(s) - \hat{v}(s, \mathbf{w})) \nabla \hat{v}(s, \mathbf{w})] \quad (18.74)$$

where  $\alpha$  is the step size or learning rate, and

$$\Delta \mathbf{w} = -\alpha \nabla J(\mathbf{w}) = \alpha E_\pi [(v_\pi(s) - \hat{v}(s, \mathbf{w})) \nabla \hat{v}(s, \mathbf{w})] \quad (18.75)$$

If the method of stochastic gradient descent (SGD) is used, i.e., the parameter

$\mathbf{w}$  is updated based on only one data point at each iteration instead of the expectation of the data points, then  $E_\pi$  for the expectation can be dropped:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \Delta \mathbf{w} = \mathbf{w}_n + \alpha(v_\pi(s) - \hat{v}(s, \mathbf{w})) \nabla \hat{v}(s, \mathbf{w}) \quad (18.76)$$

Here the true value function  $v_\pi(s)$ , the expected return, is unknown and needs to be estimated by the actual return  $G_t$  at each state  $s = s_t$  based on the MC method, or  $r_{t+1} + \gamma \hat{v}_\pi(s', \mathbf{w})$  in terms of the immediate reward  $r_{t+1}$  and the estimated value  $\hat{v}_\pi(s', \mathbf{w})$  of the next state  $s'$  based on the TD method, by sampling the environment. Such an approach can be considered as a supervised learning based on labeled training data set:  $\{(s_t, G_t), \forall t\}$  for the MC method, or  $\{(s_t, r_{t+1}), \forall t\}$  for the TD method.

In the following, we will consider the special case where the state value  $v_\pi(s)$  is approximated by a linear function

$$\hat{v}(s, \mathbf{w}) = \sum_{i=1}^d x_i w_i = \mathbf{w}^T \mathbf{x}(s) \quad (18.77)$$

based on the assumption that the value function of each state can be approximated as a linear combination of a set of features in the feature vector  $\mathbf{x}(s) = [x_1(s), \dots, x_d(s)]^T$  for state  $s$ , weighted by the corresponding weights in the weight vector  $\mathbf{w} = [w_1, \dots, w_d]^T$  for all states.

As a simple example, if we represent each of the  $N = |S|$  states  $s_n$  by a  $d = N$  dimensional binary feature vector  $\mathbf{x}(s_n) = [x_1(s_n), \dots, x_N(s_n)]^T$  of all zero components  $x_i(s_n) = 0$  except the  $n$ -th one  $x_n(s_n) = 1$ , then the weight vector can be  $\mathbf{w} = [w_1, \dots, w_N]^T = [v_\pi(s_1), \dots, v_\pi(s_N)]^T$ , and we have  $\hat{v}_\pi(s_n, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s_n) = v_\pi(s_n)$ . In this case, each state  $s$  is represented by its value function  $v_\pi(s)$ , same as in all previous discussions.

The objective function for the mean squared error of such a linear approximating function is

$$J(\mathbf{w}) = \frac{1}{2} E_\pi [(v_\pi(s) - \hat{v}(s, \mathbf{w}))^2] = \frac{1}{2} E_\pi [(v_\pi(s) - \mathbf{w}^T \mathbf{x}(s))^2] \quad (18.78)$$

and its gradient is

$$\nabla J(\mathbf{w}) = \frac{d}{d\mathbf{w}} J(\mathbf{w}) = -E_\pi [(v_\pi(s) - \mathbf{w}^T \mathbf{x}(s)) \mathbf{x}(s)] \quad (18.79)$$

If all data points in terms of state  $s$  represented by  $\mathbf{x}(s)$  and the corresponding return  $G(s)$  as a sample of the true value  $v_\pi(s)$  have already been collected, then the value function approximation can be carried out as an off-line algorithm in a batch manner, the same as in the linear least squares linear regression problem discussed in a previous chapter, and the optimal weight vector that minimizes the squared error

$$\mathbf{w}^* = \arg \max_{\mathbf{x}} \sum_s [G(s) - \hat{v}_\pi(s, \mathbf{w})]^2 \quad (18.80)$$

can be found by the same pseudo inverse method

$$\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{G} \quad (18.81)$$

where  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  is a matrix containing all  $N$  samples for the states and  $\mathbf{G} = [G_1, \dots, G_N]^T$  is a vector containing the corresponding returns.

However, as reinforcement learning is typically an online problem, the parameter  $\mathbf{w}$  for the approximator needs to be modified in real-time whenever a new piece of data becomes available during the sampling of the environment. In this case, we can find the gradient descent method, based on the gradient vector of  $J(\mathbf{w})$ :

$$\begin{aligned} \nabla J(\mathbf{w}) &= \frac{d}{d\mathbf{w}} J(\mathbf{w}) = -E_\pi(v_\pi(s) - \hat{v}(s, \mathbf{w})) \nabla \hat{v}(s, \mathbf{w}) \\ &= -E_\pi(v_\pi(s) - \mathbf{w}^T \mathbf{x}(s)) \mathbf{x}(s) \end{aligned} \quad (18.82)$$

and the optimal parameter  $\mathbf{w}^*$  can be found iteratively

$$\begin{aligned} \mathbf{w}_{n+1} &= \mathbf{w}_n + \Delta \mathbf{w} = \mathbf{w}_n + \alpha E_\pi [v_\pi(s) - \hat{v}(s, \mathbf{w})] \nabla \hat{v}(s, \mathbf{w}) \\ &= \mathbf{w}_n + \alpha E_\pi [v_\pi(s) - \mathbf{w}^T \mathbf{x}(s)] \mathbf{x}(s) \end{aligned} \quad (18.83)$$

where  $\Delta \mathbf{w}$  is the increment in each iteration:

$$\Delta \mathbf{w} = \alpha E_\pi [v_\pi(s) - \hat{v}(s, \mathbf{w})] \nabla \hat{v}(s, \mathbf{w}) = \alpha E_\pi [v_\pi(s) - \mathbf{w}_n^T \mathbf{x}(s)] \mathbf{x}(s) \quad (18.84)$$

If stochastic gradient descent is used, then  $\mathbf{w}$  is updated whenever a new sample data point  $\mathbf{x}(s_t)$  is available at a time step  $t$ , then the expectation  $E_\pi$  in the equations above can be dropped.

We note that the expectation of the estimated  $\mathbf{w}$  found by this SGD method is the same as that obtained by full GD method. Also this iteration will convergence to the global minimum of the objective function  $J(\mathbf{w})$  in Eq. (18.78) as it is a quadratic function with only one minimum which is global.

Specifically, the state value function is unknown and can be estimated in several different ways, similar to the corresponding algorithms discussed before.

- **MC method**

The value function  $v_\pi(s)$  is estimated by the actual return  $G_t$  for each state  $s_t$  visited in an episode, obtained only at the end of each episode while sampling the environment:

$$\Delta \mathbf{w} = \alpha [G_t - \hat{v}(s_t, \mathbf{w})] \nabla \hat{v}(s_t, \mathbf{w}) = \alpha [G_t - \mathbf{w}^T \mathbf{x}(s_t)] \mathbf{x}(s_t) \quad (18.85)$$

Here is the pseudo code for the algorithm based on the MC method, similar to that for value evaluation algorithms discussed previously:

Initialize  $\mathbf{w} = \mathbf{0}$

**loop** (for each episode)

run current episode to the end to get  $G_t$ ,  $t = 1, \dots, T$

**for**  $t = 1, \dots, T$

**if**  $s_t$  is visited the first time

```


$$\mathbf{w} = \mathbf{w} + \alpha[G_t - \mathbf{w}^T \mathbf{x}(s_t)]\mathbf{x}(s_t)$$

end if
end for
end loop

```

- **TD(0) method**

The value function  $v_\pi(s)$  is estimated by the TD target, the sum of the immediate reward  $r_{t+1}$  and the approximated value of the next state  $s_{t+1}$ , at each time step of each episode while sampling the environment:

$$\Delta \mathbf{w} = \alpha[r_{t+1} + \gamma \mathbf{w}^T \mathbf{x}(s_{t+1}) - \mathbf{w}^T \mathbf{x}(s_t)]\mathbf{x}(s_t) \quad (18.86)$$

Here is the pseudo code for the algorithm based on the TD method, similar to that for value evaluation algorithms discussed previously:

```

Initialize  $\mathbf{w} = \mathbf{0}$ 
loop (for each episode)
  for each time step in current episode
    take action  $a = \pi(s)$ , get reward  $r$  and next state  $s'$ 
     $\mathbf{w} = \mathbf{w} + \alpha[r + \gamma \mathbf{w}^T \mathbf{x}(s') - \mathbf{w}^T \mathbf{x}(s)]\mathbf{x}(s)$ 
  end for
end loop

```

- **TD( $\lambda$ ) method**

The value function  $v_\pi(s)$  is estimated based on  $\lambda$ -return  $G_t^\lambda$  given in Eq. (18.62). As discussed before, TD( $\lambda$ ) has two flavors:

- Forward view: same as MC method, except the true return  $G_t$  is replaced by  $\lambda$ -return  $G_t^\lambda$  available only at the end of each episode:

$$\Delta \mathbf{w} = \alpha[G_t^\lambda - \mathbf{w}^T \mathbf{x}(s_t)]\mathbf{x}(s_t) \quad (18.87)$$

- Backward view: similar to TD(0), this is based on the immediate reward  $r_{t+1}$  available at every step of each episode:

$$\delta_t = \alpha[r_{t+1} + \gamma \mathbf{w}^T \mathbf{x}(s_{t+1}) - \mathbf{w}^T \mathbf{x}(s_t)]\mathbf{x}(s_t) \quad (18.88)$$

$$e_t = \gamma \lambda e_{t-1} + \mathbf{x}(s_t), \quad \Delta \mathbf{w} = \alpha \delta_t e_t \quad (18.89)$$

Given a policy  $\pi$  for an MDP, we define a probability distribution  $d(s)$  over all states visited according to  $\pi$ , satisfying

$$\sum_s d(s) = 1 \quad (18.90)$$

and the following balance equation:

$$d(s') = \sum_s \sum_a \pi(a|s) p(s'|s, a) d(s) \quad (18.91)$$

As both  $d(s')$  and  $d(s)$  represent the same distribution, they must be identical.

Given the probability distribution  $d(s)$ , the mean squared error of the value function approximation for a policy  $\pi$  can be expressed as

$$MSE(\mathbf{w}) = \sum_s d(s) [v_\pi(s) - \hat{v}_\pi(s, \mathbf{w})]^2 \quad (18.92)$$

It can be shown that the MC method for the linear value function approximation will converge to the optimal weight vector  $\mathbf{w}_{MC}$  that minimizes the mean squared error above:

$$MSE(\mathbf{w}_{MC}) = \min_{\mathbf{w}} \sum_s d(s) [v_\pi(s) - \hat{v}_\pi(s, \mathbf{w})]^2 \quad (18.93)$$

## 18.6 Control based on Function Approximation

The control algorithms based on approximated value functions also follow the general method of the general policy iteration, as illustrated below:

We note that this is similar to the algorithms for model-free control illustrated in Fig. ??, but with the action-value function  $q_\pi(s, a)$  is replaced by the parameter  $\mathbf{w}$  of the approximation action function  $\hat{q}(s, a, \mathbf{w})$ . In particular, for a linear function, we have:

$$\hat{q}(s, a, \mathbf{w}) = \sum_n w_n x(s, a) = \mathbf{w}^T \mathbf{x}_n(s, a) \quad (18.94)$$

where  $\mathbf{x}(s, a)$  is the feature vector for the state-action pair  $(s, a)$ . We need to find the optimal parameter  $\mathbf{w}$  that minimizes the objective function, the mean square error of the approximation:

$$J(\mathbf{w}) = \frac{1}{2} E_\pi [(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w}))^2] = \frac{1}{2} E_\pi [(q_\pi(s, a) - \mathbf{w}^T \mathbf{x}(s, a))^2] \quad (18.95)$$

with gradient vector:

$$\begin{aligned} \nabla J(\mathbf{w}) &= \frac{d}{d\mathbf{w}} J(\mathbf{w}) = -E_\pi [q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})] \nabla \hat{q}(s, a, \mathbf{w}) \\ &= -E_\pi [(q_\pi(s, a) - \mathbf{w}^T \mathbf{x}(s, a)) \mathbf{x}(s, a)] \end{aligned} \quad (18.96)$$

If the stochastic gradient descent method is used based on a single sample of the action value  $q_\pi(s, a)$ , instead of its expectation, then  $E_\pi$  can be dropped, and the optimal weight vector  $\mathbf{w}^*$  that minimizes  $J(\mathbf{w})$  in Eq. (18.95) can be learned iteratively:

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t + \Delta \mathbf{w} = \mathbf{w}_t - \alpha \nabla \mathbf{J}(\mathbf{w}) \\ &= \mathbf{w}_t + \alpha [(q_\pi(s_t, a_t) - \hat{q}_\pi(s_t, a_t, \mathbf{w})) \nabla \hat{q}_\pi(s_t, a_t, \mathbf{w})] \\ &= \mathbf{w}_t + \alpha [(q_\pi(s_t, a_t) - \mathbf{w}_t^T \mathbf{x}(s_t, a_t)) \mathbf{x}(s_t, a_t)] \end{aligned} \quad (18.97)$$

where  $\Delta \mathbf{w}$  is the increment of the update:

$$\begin{aligned} \Delta \mathbf{w} &= -\alpha \nabla \mathbf{J}(\mathbf{w}) = \alpha [q_\pi(s_t, a_t) - \hat{q}_\pi(s_t, a_t, \mathbf{w})] \nabla \hat{q}_\pi(s_t, a_t, \mathbf{w}) \\ &= \alpha [q_\pi(s_t, a_t) - \mathbf{w}_t^T \mathbf{x}(s_t, a_t)] \mathbf{x}(s_t, a_t) \end{aligned} \quad (18.98)$$

As the true Q-value  $q_\pi(s, a)$  in the expression is unknown, it needs to be estimated by some target depending on the specific methods used:

- **MC method:**

The true  $q_\pi(s, a)$  is replaced by the sample return  $G_t$  as the target, obtained at the end of each episode:

$$\Delta \mathbf{w} = \alpha [G_t - \hat{q}(s_t, a_t, \mathbf{w})] \nabla \hat{q}(s_t, a_t, \mathbf{w}) = \alpha [G_t - \mathbf{w}_t^T \mathbf{x}(s_t, a_t)] \mathbf{x}(s_t, a_t) \quad (18.99)$$

- **TD(0) method:**

The action value function  $q_\pi(s, a)$  is replaced by the TD target, the sum of the immediate reward, available at each step of each episode, and the approximated action value of the next state  $s_{t+1}$ :

– SARSA (on-policy):

$$\begin{aligned} \Delta \mathbf{w} &= \alpha [r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{q}(s_t, a_t, \mathbf{w})] \nabla \hat{q}(s_t, a_t, \mathbf{w}) \\ &= \alpha [r_{t+1} + \gamma \mathbf{w}^T \mathbf{x}(s_{t+1}, a_{t+1}) - \mathbf{w}_t^T \mathbf{x}(s_t, a_t)] \mathbf{x}(s_t, a_t) \end{aligned} \quad (18.100)$$

Following Eq. (18.71), the TD error is defined as:

$$\delta_t = r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{q}(s_t, a_t, \mathbf{w}) \quad (18.101)$$

then we get

$$\Delta \mathbf{w} = \alpha \delta_t \nabla \hat{q}(s_t, a_t, \mathbf{w}) \quad (18.102)$$

– Q-learning (off-policy):

$$\begin{aligned} \Delta \mathbf{w} &= \alpha [r_{t+1} + \gamma \max_{a'} \hat{q}(s_{t+1}, a', \mathbf{w}) - \hat{q}(s_t, a_t, \mathbf{w})] \nabla \hat{q}(s_t, a_t, \mathbf{w}) \\ &= \alpha [r_{t+1} + \gamma \max_{a'} \mathbf{w}^T \mathbf{x}(s_{t+1}, a') - \mathbf{w}_t^T \mathbf{x}(s_t, a_t)] \mathbf{x}(s_t, a_t) \end{aligned} \quad (18.103)$$

- **TD( $\lambda$ ) method:**

In the forward-view version of the TD( $\lambda$ ) method the action function  $q_\pi(s, a)$  is approximated by  $\lambda$ -return  $G_t^\lambda$  as the target, available only at the end of each episode:

$$\Delta \mathbf{w} = \alpha [G_t^\lambda - \mathbf{w}_t^T \mathbf{x}(s_t, a_t)] \mathbf{x}(s_t, a_t) \quad (18.104)$$

The backward-view version of the TD( $\lambda$ ) method based on eligibility traces is more advantageous in both space and temporal complexity as well as learning efficiency.

We first define an eligibility trace vector which is set to zero at the beginning of the episode, but then decays

$$e_t = \gamma \lambda e_{t-1} + \nabla_w \hat{q}(s_t, a_t, \mathbf{w}) \quad (18.105)$$

$$\Delta \mathbf{w} = \alpha \delta_t e_t \quad (18.106)$$



and Eq. (18.70)

$$v_\pi(s) = v_\pi(s) + \alpha \delta_t e_t(s), \quad \forall s \in S \quad (18.107)$$

////

Recall the TD error for the backward view of the TD( $\lambda$ ) method first given in Eq. (18.71):

$$\delta_t = (r_{t+1} + \gamma v_\pi(s_{t+1}) - v_\pi(s)) \quad (18.108)$$

In summary, here are the conceptual (not necessarily algorithmic) steps for the general model-free control based on approximated action-value function:

- Learn parameter  $\mathbf{w}$  as in Eq. (18.97),
- Get the Q-values as in Eq. (18.94)
- Obtain the policy by  $\epsilon$ -greedy approach as in Eq. (18.41)

These steps are also illustrated below:

$$\text{Training of } \mathbf{w} \implies Q\text{-value} \implies \text{Policy } \pi \quad (18.109)$$

## 18.7 Deep Q-learning

Previously we approximated the state or action value functions by a linear function  $\hat{v}_\pi(s, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s)$ , or  $\hat{q}_\pi(s, a, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s, a)$  parameterized by the weight vector  $\mathbf{w}$  based on a set of features in  $\mathbf{x}$ . However, these features need to hand picked or designed based on the specific problem to solve.

Network!

## 18.8 Policy Gradient Methods

All RL algorithms previous considered are based on either state or action-value function and the policy is indirectly derived from them by greedy or  $\epsilon$ -greedy method. However, as the ultimate goal of an RL problem is to find the optimal policy that maximizes the return, it make sense to consider this as an optimization problem to directly find the optimal policy based on an objective function representing the total return to be maximized, by the gradient ascent method, as discussed in the following.

Previously we approximate the state value function  $v_\pi(s)$  and action value function  $q_\pi(s, a)$  by some parameterized functions  $\hat{v}_\pi(s, \mathbf{w})$  and  $\hat{q}_\pi(s, a, \mathbf{w})$  respectively, of which the parameter  $\mathbf{w}$  can be obtained by sampling the environment, as a supervised learning process. Now we construct a model of a stochastic policy as a parameterized function:

$$\pi(a|s, \theta) = P(a|s, \theta) \quad (18.110)$$

where vector  $\theta = [\theta_1, \dots, \theta_d]^T$  represents some  $d$  parameters of the model. As the dimensionality  $d$  is typically smaller than the number of states and actions, such a parameterized policy model is suitable in cases where the numbers of states and actions are large or even continuous.

As a specific example, the policy model can be based on the soft-max function:

$$\pi(a|s, \theta) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}} \quad (18.111)$$

satisfying  $\sum_a \pi(a|s, \theta) = 1$ . Here the summation is over all possible actions, and  $h(s, a, \theta)$  is the *preference* of action  $a$  in state  $s$ , which can be a parameterized function such as a simple linear function  $h(s, a, \theta) = \theta^T \mathbf{x}(s, a)$ , or a neural network with weights represented by  $\theta$ , the same as how the value functions are approximated in Section 18.5. According to this policy, an action with higher preference  $h(s, a, \theta)$  will have a higher probability to be chosen.

Different from how we find the parameter  $\mathbf{w}$  of  $\hat{q}(s, a, \mathbf{w})$  by minimizing  $J(\mathbf{w})$ , the mean squared error between the value function  $q_\pi(s, a)$  and its model  $\hat{q}_\pi(s, a, \mathbf{w})$ , based on gradient descent, here we find the parameter  $\theta$  of  $\pi(a|s, \theta)$  by maximizing  $J(\theta)$  representing the value function, the expected return, under the policy, based on gradient ascent. Such methods are therefore called *policy gradient methods*.

The value-function based methods considered previously and the policy-based methods considered here are summarized below, together with the *actor-critic method*, as the combination of the two:

- Value-based: the policy is derived by the greedy or  $\epsilon$ -greedy method based on the value function learned during sampling the environment as a supervised learning process.
- Policy-based: the policy is directly learned without explicitly value function estimation.
- Actor-Critic: parameters for both the value function model and policy model are learned simultaneously.

Advantages of policy-based RL includes better convergence properties, but may stuck at local optimum, effective in high-dimensional or continuous action space can learn stochastic policies, but have high variance.

How good a policy is may be measured by different objective functions all related to the values or rewards associated with the policy being evaluated, depending on the environment of the specific problem:

- In episodic environments with finite horizon, we can use the value, the expected return (sum of discounted future rewards), of the start state  $s_0$ :

$$J_1(\theta) = v_{\pi_\theta}(s_0) \quad (18.112)$$

- In continuing (online) environments with infinite horizon, we can use the

average value

$$J_{avV}(\theta) = \sum_s d_{\pi_\theta}(s) v_{\pi_\theta}(s) \quad (18.113)$$

where  $d_{\pi_\theta}(s)$  is the stationary distribution of all states under policy  $\pi(a|s, \theta)$ .

We can find the optimal parameter  $\theta^*$  for the policy model  $\pi(a|s, \theta)$  by solving the maximization problem:

$$\theta^* = \arg \max_{\theta} J(\theta) \quad (18.114)$$

based on the gradient of  $J(\theta)$ :

$$\nabla_{\theta} J(\theta) = \frac{d}{d\theta} J(\theta) = \left[ \frac{\partial J}{\partial \theta_1}, \dots, \frac{\partial J}{\partial \theta_n} \right]^T \quad (18.115)$$

by the iterative gradient ascent method:

$$\theta_{t+1} = \theta_t + \Delta\theta = \theta_t + \alpha \nabla_{\theta} J(\theta) \quad (18.116)$$

Here we use  $t$  for the index of the iteration based on the assumption that the a new sample point is available at every time step of an episode while sampling the environment following policy  $\pi(a|s, \theta)$ .

While it is conceptually straight forward to see how gradient ascent method can be used to find the optimal parameter  $\theta^*$  that maximizes  $J(\theta)$ , it is not easy to actually find its gradient  $\nabla J(\theta)$ , which depends on not only what actions to take at the states directly determined by the policy  $\pi(a|s, \theta)$ , but also how the states are distributed under the policy in an unknown environment. This challenge is addressed by the following *policy gradient theorem*. The proof of the theorem below leads to an expression of the gradient  $\nabla_{\theta} J(\theta)$ , which can be used to find the optimal parameter  $\theta^*$  iteratively by the stochastic gradient ascent method.

**Proof** of policy gradient theorem:

$$\begin{aligned}
\nabla_{\theta} v_{\pi_{\theta}}(s) &= \nabla_{\theta} \left( \sum_a \pi(a|s, \theta) q_{\pi_{\theta}}(s, a) \right) \\
&\stackrel{1}{=} \sum_a [\nabla_{\theta} \pi(a|s, \theta) q_{\pi_{\theta}}(s, a) + \pi(a|s, \theta) \nabla_{\theta} q_{\pi_{\theta}}(s, a)] \\
&\stackrel{2}{=} \sum_a \left[ \nabla_{\theta} \pi(a|s, \theta) q_{\pi_{\theta}}(s, a) + \pi(a|s, \theta) \nabla_{\theta} \sum_{s'} \sum_r P(s', r|s, a) (r + v_{\pi_{\theta}}(s')) \right] \\
&\stackrel{3}{=} \sum_a \left[ \nabla_{\theta} \pi(a|s, \theta) q_{\pi_{\theta}}(s, a) + \pi(a|s, \theta) \sum_{s'} \sum_r P(s', r|s, a) \nabla_{\theta} v_{\pi_{\theta}}(s') \right] \\
&\stackrel{4}{=} \sum_a \left[ \nabla_{\theta} \pi(a|s, \theta) q_{\pi_{\theta}}(s, a) + \pi(a|s, \theta) \sum_{s'} P(s'|s, a) \nabla_{\theta} v_{\pi_{\theta}}(s') \right] \\
&\stackrel{5}{=} \sum_a \nabla_{\theta} \pi(a|s, \theta) q_{\pi_{\theta}}(s, a) + \sum_a \sum_{s'} \pi(a|s, \theta) P(s'|s, a) \nabla_{\theta} v_{\pi_{\theta}}(s') \\
&\stackrel{6}{=} \phi(s) + \sum_{s'} \sum_a P(s'|s, a) \pi(a|s, \theta) \nabla_{\theta} v_{\pi_{\theta}}(s') \tag{18.117}
\end{aligned}$$

where

1. product rule of derivative:  $(uv)' = u'v + uv'$
2.  $q_{\pi_{\theta}}(s, a) = \sum_{s'} \sum_r P(s', r|s, a) (r + v_{\pi_{\theta}}(s'))$
3.  $r$  is not a function of  $\theta$
4.  $\sum_r P(s', r|s, a) = P(s'|s, a)$  in Eq. (18.17)
5.  $\pi(a|s, \theta)$  is independent of  $s'$  and moved inside summation over  $s'$
6. We have defined

$$\phi(s) = \sum_a \nabla_{\theta} \pi(a|s, \theta) q_{\pi_{\theta}}(s, a) \tag{18.118}$$

We note that Eq. (refPolicyGradientThm1) is a recursion by which  $\nabla_{\theta} v_{\pi_{\theta}}(s)$  is expressed as a function in terms  $\nabla_{\theta} v_{\pi_{\theta}}(s')$ .

We further define  $Pr_{\pi}(s \rightarrow x, k)$  as the probability of transitioning from state  $s$  to a state  $x$  after  $k$  steps following  $\pi(a|s, \theta)$ :

$$s \xrightarrow{\pi(a|s, \theta)} s' \xrightarrow{\pi(a|s', \theta)} s'' \xrightarrow{\pi(a|s'', \theta)} \dots \xrightarrow{\pi(a|s^{(k-1)}, \theta)} s^{(k)} = x \tag{18.119}$$

with the following properties:

•

$$Pr_{\pi}(s \rightarrow s, 0) = 1 \tag{18.120}$$

•

$$Pr_{\pi}(s \rightarrow s', 1) = \sum_a \pi(a|s, \theta) P(s'|s, a) \tag{18.121}$$

- 
- 

$$\sum_{s'} Pr_{\pi}(s \rightarrow s', 1) = 1 \quad (18.122)$$

$$Pr_{\pi}(s \rightarrow s'', 2) = Pr_{\pi}(s \rightarrow s', 1)Pr_{\pi}(s' \rightarrow s'', 1) \quad (18.123)$$

Continuing Eq. (18.117) we keep rolling out the recursion of  $\nabla_{\theta} v_{\pi_{\theta}}(s')$  and get:

$$\begin{aligned} \nabla_{\theta} v_{\pi_{\theta}}(s) &= \phi(s) + \sum_{s'} \sum_a \pi(a|s, \theta) P(s'|s, a) \nabla_{\theta} v_{\pi_{\theta}}(s') \\ &\stackrel{1}{=} \phi(s) + \sum_{s'} Pr_{\pi}(s \rightarrow s', 1) \nabla_{\theta} v_{\pi_{\theta}}(s') \\ &\stackrel{2}{=} \phi(s) + \sum_{s'} Pr_{\pi}(s \rightarrow s', 1) \left[ \phi(s') + \sum_{s''} Pr_{\pi}(s' \rightarrow s'', 1) \nabla_{\theta} v_{\pi_{\theta}}(s'') \right] \\ &\stackrel{3}{=} \phi(s) + \sum_{s'} Pr_{\pi}(s \rightarrow s', 1) \phi(s') + \sum_{s''} Pr_{\pi}(s \rightarrow s', 1) Pr_{\pi}(s' \rightarrow s'', 1) \nabla_{\theta} v_{\pi_{\theta}}(s'') \\ &\stackrel{4}{=} \phi(s) + \sum_{s'} Pr_{\pi}(s \rightarrow s', 1) \phi(s') + \sum_{s''} Pr_{\pi}(s \rightarrow s'', 2) \nabla_{\theta} v_{\pi_{\theta}}(s'') \\ &\stackrel{5}{=} \phi(s) + \sum_{s'} Pr_{\pi}(s \rightarrow s', 1) \phi(s') + \sum_{s''} Pr_{\pi}(s \rightarrow s'', 2) \phi(s'') + \sum_{s'''} Pr_{\pi}(s \rightarrow s''', 2) \nabla_{\theta} v_{\pi_{\theta}}(s''') \\ &= \dots \stackrel{6}{=} \sum_{x \in S} \sum_{k=0}^{\infty} Pr_{\pi}(s \rightarrow x, k) \phi(x) \end{aligned} \quad (18.124)$$

where

1. Eq. (18.121)
2. unroll  $\nabla_{\theta} v_{\pi_{\theta}}(s')$
3. Eq. (18.122)
4. Eq. (18.123)
5. unroll  $\nabla_{\theta} v_{\pi_{\theta}}(s'')$
6. keep unrolling recursively to infinity, and Eq. (18.120)

Now the gradient of the objective  $J(\theta) = v_{\pi_{\theta}}(s_0)$  can be written as

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \nabla_{\theta} v_{\pi}(s_0) = \sum_{s \in S} \sum_{k=0}^{\infty} Pr_{\pi}(s_0 \rightarrow s, k) \phi(s) \\ &= \sum_{s \in S} \eta(s) \phi(s) = \left( \sum_{s'} \eta(s') \right) \sum_{s \in S} \frac{\eta(s)}{\sum_{s'} \eta(s')} \phi(s) \\ &\propto \sum_{s \in S} \frac{\eta(s)}{\sum_{s'} \eta(s')} \phi(s) = \sum_{s \in S} \mu_{\pi}(s) \sum_a \nabla_{\theta} \pi(a|s, \theta) q_{\pi_{\theta}}(s, a) \end{aligned} \quad (18.125)$$

Here the proportionality is introduced due to the dropping of  $\sum_{s'} \eta(s')$  as a

constant independent of state  $s$ , and we also defined

$$\eta(s) = \sum_{k=0}^{\infty} Pr_{\pi}(s_0 \rightarrow s, k), \quad (18.126)$$

as the sum of all probabilities for visiting state  $s$  from the start state  $s_0$ , and

$$\mu_{\pi}(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')} \quad (18.127)$$

as a normalized version of  $\eta(s)$  representing the probability distribution of visiting state  $s$  while following policy  $\pi$ . Eq. (18.125) can be further written as

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \nabla_{\theta} v_{\pi}(s_0) \propto \sum_{s \in S} \mu_{\pi}(s) \sum_a \pi(a|s, \theta) \frac{\nabla_{\theta} \pi(a|s, \theta)}{\pi(a|s, \theta)} q_{\pi_{\theta}}(s, a) \\ &= \sum_{s \in S} \mu_{\pi}(s) \sum_a \pi(a|s, \theta) \nabla_{\theta} \ln \pi(a|s, \theta) q_{\pi_{\theta}}(s, a) \\ &= E_{\pi_{\theta}} [\nabla_{\theta} \ln \pi(a|s, \theta) q_{\pi_{\theta}}(s, a)] \end{aligned} \quad (18.128)$$

where  $E_{\pi_{\theta}}$  denotes the expectation over all actions in each state  $s$  weighted by  $\pi(a|s, \theta)$  and all states  $s \in S$  weighted by  $\mu_{\pi}(s)$ .

Q.E.D.

We see that the gradient  $\nabla_{\theta} J(\theta)$  is now expressed as the expectation of the action value function  $q_{\pi}(s, a)$ , weighted by the gradient of the logarithm of the corresponding policy  $\pi(a|s, \theta)$ . Now Eq. (18.116) can be further written as

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta) = \theta_t + \alpha E_{\pi_{\theta}} [q_{\pi_{\theta}}(s_t, a_t) \nabla_{\theta} \ln \pi(a_t|s_t, \theta)] \quad (18.129)$$

We further note that Eq. (18.125) still hold if an arbitrary bias term  $b(s)$  independent of action  $a$  is included:

$$\begin{aligned} \nabla_{\theta} J(\theta) &\propto \sum_{s \in S} \mu_{\pi}(s) \sum_a \nabla_{\theta} \pi(a|s, \theta) (q_{\pi_{\theta}}(s, a)) \\ &= \sum_{s \in S} \mu_{\pi}(s) \sum_a \nabla_{\theta} \pi(a|s, \theta) (q_{\pi_{\theta}}(s, a) + b(s)) \end{aligned} \quad (18.130)$$

as

$$\sum_a \nabla_{\theta} \pi(a|s, \theta) b(s) = b(s) \nabla_{\theta} \sum_a \pi(a|s, \theta) = b(s) \nabla_{\theta} 1 = 0 \quad (18.131)$$

Now Eq. (18.116) can also be written as

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta) = \theta_t + \alpha E_{\pi_{\theta}} [(q_{\pi_{\theta}}(s_t, a_t) + b(s_t)) \nabla_{\theta} \ln \pi(a_t|s_t, \theta)] \quad (18.132)$$

The expectation  $E_{\pi_{\theta}}$  in Eqs. (18.129) and (18.132) can be dropped if the method of stochastic gradient ascent is used, as in all algorithms below, where each iterative step is based on only one sample data point instead of its expectation.

Specifically, for a soft-max policy model as given in Eq. (18.111), we have

$$\begin{aligned}
\nabla_{\theta} \ln \pi(a|s, \theta) &= \nabla_{\theta} \ln \left( \frac{e^{\theta^T \mathbf{x}(s,a)}}{\sum_b e^{\theta^T \mathbf{x}(s,b)}} \right) = \nabla_{\theta} \left( \theta^T \mathbf{x}(s,a) - \ln \sum_b e^{\theta^T \mathbf{x}(s,b)} \right) \\
&= \mathbf{x}(s,a) - \frac{\nabla_{\theta} \sum_b e^{\theta^T \mathbf{x}(s,b)}}{\sum_b e^{\theta^T \mathbf{x}(s,b)}} = \mathbf{x}(s,a) - \frac{\sum_b e^{\theta^T \mathbf{x}(s,b)} \mathbf{x}(s,b)}{\sum_b e^{\theta^T \mathbf{x}(s,b)}} \\
&= \mathbf{x}(s,a) - \sum_b \frac{e^{\theta^T \mathbf{x}(s,b)}}{\sum_b e^{\theta^T \mathbf{x}(s,b)}} \mathbf{x}(s,b) \\
&= \mathbf{x}(s,a) - \sum_b \pi(b|s, \theta) \mathbf{x}(s,b) \tag{18.133}
\end{aligned}$$

We list set of popular policy-based algorithms below, based on either the MC or TD methods, generally used in previous algorithms.

- **REINFORCE (MC) policy gradient**

In this algorithm, the action value function  $q_{\pi_{\theta}}(s_t, a_t)$  in Eq. (18.129) is replaced by the return  $G_t$  obtained at the end of each episode while sampling the environment. Now the equation becomes:

$$\theta_{t+1} = \theta_t + \alpha \gamma^t G_t \nabla_{\theta} \ln \pi(a|s, \theta) \tag{18.134}$$

Note that the discount factor  $\gamma^t$  is included as the expression for  $\nabla_{\theta} J(\theta)$  in Eq. (18.128) assumed  $\gamma = 1$  for simplicity.

Here is the pseudo code for the algorithm:

```

Initialize  $\pi(a|s, \theta)$ 
loop (for each episode)
    At the end of episode, get  $G_t$  for each state visited
    for  $t = 1, \dots, T$ 
         $\theta = \theta + \alpha \gamma^t G_t \nabla_{\theta} \ln \pi(a_t|s_t, \theta)$ 
    end loop

```

- **Actor-Critic (TD) policy gradient**

As its name suggests, this algorithm is based on two approximation function models, the first for the policy  $\pi(a|s, \theta)$  parameterized by  $\theta$ , the *actor*, same as in the REINFORCE algorithm above, and the second for the value function  $\hat{v}_{\pi}(s, \mathbf{w})$  parameterized by  $\mathbf{w}$ , the *critic*, same as in Eq. (18.97).

Specifically, in Eq. (18.132), the action value function  $q_{\pi_{\theta}}(s_t, a_t)$  is replaced by its bootstrapping expression  $r_{t+1} + \gamma \hat{v}_{\pi}(s', \mathbf{w})$ , and the bias term  $b(s)$  is replaced by the approximated value function  $\hat{v}(s_t, \mathbf{w})$ . Now both parameters  $\mathbf{w}$  and  $\theta$  can be found iteratively at every step of an episode while sampling the environment (the TD method) by stochastic gradient method with the expectation  $E_{\pi}$  dropped:

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha_w [(r_{t+1} + \gamma \hat{v}_{\pi}(s', \mathbf{w}) - \hat{v}_{\pi}(s, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}_{\pi}(s, \mathbf{w})] \\
\theta_{t+1} &= \theta_t + \alpha_{\theta} [(r_{t+1} + \gamma \hat{v}_{\pi}(s', \mathbf{w}) - \hat{v}_{\pi}(s, \mathbf{w})) \nabla_{\theta} \ln \pi(a|s, \theta)] \tag{18.135}
\end{aligned}$$

Here is the pseudo code for the algorithm:

```

initialize model parameters  $\theta$  and  $\mathbf{w}$ 
initialize step sizes  $\alpha_\theta$  and  $\alpha_w$ 
loop (for each episode)
  initialize  $s = s_0, t = 0$ 
  while  $s$  is not terminal (for each step)
    take action  $a$  following  $\pi(a|s, \theta)$ , find reward  $r$  and next
      state  $s'$ 
    find TD error:  $\delta = r + \gamma \hat{v}(s', \mathbf{w}) - \hat{v}(s, \mathbf{w})$ 
     $\mathbf{w} = \mathbf{w} + \alpha_w \delta \nabla \hat{v}(s, \mathbf{w})$ 
     $\theta = \theta + \alpha_\theta \gamma^t \delta \nabla \ln \pi(a|s, \theta)$ 
     $s = s'$ 
     $t = t + 1$ 
  end while
end loop

```

- **Backward view of TD( $\lambda$ ) policy gradient**  
 Here is the pseudo code for the algorithm:
 

```

initialize model parameters  $\theta$  and  $\mathbf{w}$ 
initialize step sizes  $\alpha_\theta$  and  $\alpha_w$ 
initialize trace-decay rates  $\lambda_\theta$  and  $\lambda_w$ 
loop (for each episode)
  initialize  $s = s_0, t = 0$ 
  initialize  $\mathbf{z}_\theta = \mathbf{0}, \mathbf{z}_w = \mathbf{0}$ 
  while  $s$  is not terminal (for each step)
    take action  $a$  following  $\pi(a|s, \theta)$ , find reward  $r$  and next
      state  $s'$ 
    find TD error:  $\delta = r + \gamma \hat{v}(s', \mathbf{w}) - \hat{v}(s, \mathbf{w})$ 
     $\mathbf{z}_\theta = \gamma \lambda_\theta \mathbf{z}_\theta + \nabla \gamma^t \ln \pi(a|s, \theta)$ 
     $\mathbf{z}_w = \gamma \lambda_w \mathbf{z}_w + \gamma^t \nabla \hat{v}(s, \mathbf{w})$ 
     $\mathbf{w} = \mathbf{w} + \alpha_w \delta \nabla \hat{v}(s, \mathbf{w})$ 
     $\theta = \theta + \alpha_\theta \delta \nabla \ln \pi(a|s, \theta)$ 
     $s = s'$ 
     $t = t + 1$ 
  end while
end loop

```

Following the similar steps, the policy gradient theorem for environment with continuous state and action spaces can be also proven:

$$\nabla_\theta J(\theta) = \int_s \mu_\pi(s) \int_a \nabla_\theta \pi(a|s, \theta) q_{\pi_\theta}(s, a) da ds \quad (18.136)$$



# Appendix A A Review of Linear Algebra

---

## A.1 Inner Product Space

- **Vector space**

A *vector space* over field  $F$  is a set  $V$  which is closed under the following two operations of addition and scalar multiplication defined for its members (called vectors), i.e., the results of the operations are also members of  $V$ .

1. The vector addition that maps any two vectors  $\mathbf{x}, \mathbf{y} \in V$  to another vector  $\mathbf{x} + \mathbf{y} \in V$  satisfying the following properties:

- Commutativity:  $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$ .
- Associativity:  $\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$ .
- Existence of zero: there is a vector  $\mathbf{0} \in V$  such that:  $\mathbf{0} + \mathbf{x} = \mathbf{x} + \mathbf{0} = \mathbf{x}$ .
- Existence of inverse: for any vector  $\mathbf{x} \in V$ , there is another vector  $-\mathbf{x} \in V$  such that  $\mathbf{x} + (-\mathbf{x}) = \mathbf{0}$ .

2. The scalar multiplication that maps a vector  $\mathbf{x} \in V$  and scalar  $a \in F$  ( $F$  can be a real or complex field) to another vector  $a\mathbf{x} \in V$  with the following properties:

- $a(\mathbf{x} + \mathbf{y}) = a\mathbf{x} + a\mathbf{y}$ .
- $(a + b)\mathbf{x} = a\mathbf{x} + b\mathbf{x}$ .
- $ab\mathbf{x} = a(b\mathbf{x})$ .
- $1\mathbf{x} = \mathbf{x}$ .

where  $\mathbf{0} = [0, \dots, 0]^T$  and  $\mathbf{1} = [1, \dots, 1]^T$  are two constant vectors.

A subset  $W$  of  $V$  is a subspace of  $V$ , denoted by  $W \subseteq V$ , if it is also a vector space, i.e., it is closed under the same operations defined in  $V$ :

1. The zero vector  $\mathbf{0}$  of  $V$  must be in  $W$  (the zero vector is unique in  $\mathbf{V}$ , which  $\mathbf{W}$  must have).

2. For any  $\mathbf{x}, \mathbf{y} \in W$ ,  $\mathbf{x} + \mathbf{y} \in W$ .

3. For any  $\mathbf{x} \in W$ ,  $a\mathbf{x} \in W$ .

Listed below is a set of typical vector spaces:

- $n$ -D vector space  $\mathbb{R}^n$  or  $\mathbb{C}^n$

This space contains all  $n$ -D vectors expressed as an  $n$ -tuple, an or-

dered list of  $n$  elements (or components):

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = [x_1, \dots, x_n]^T, \quad (\text{A.1})$$

which can be used to represent a discrete signal containing  $n$  samples. We will always represent a vector as a vertical or column vector, or the transpose of a horizontal or row vector. The space is denoted by either  $\mathbb{C}^n$  if the elements are complex  $x_i \in \mathbb{C}$ , or  $\mathbb{R}^n$  if they are all real  $x_i \in \mathbb{R}$  ( $i = 1, \dots, n$ ).

A subspace of  $\mathbb{R}^n$  can be a  $\mathbb{R}^m$  ( $m < n$ ) that passes origin (zero). For example, any 2-D plane passing through the origin of a 3-D space is its subspace. However, if the 2-D plane does not pass through the origin, it is not a subspace. Also, as 3-D cube or sphere centered at the origin is not a subspace, as it is not closed under the operations of addition and scalar multiplication.

- A vector space can be defined to contain all  $m \times n$  matrices composed of  $n$   $m$ -D column vectors:

$$\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n] = \begin{bmatrix} a_{11} & a_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}, \quad (\text{A.2})$$

where the  $i$ th column is an  $m$ -D vector  $\mathbf{a}_i = [x_{1i}, \dots, x_{mi}]^T$ . Such a matrix can be converted to an  $mn$ -D vector by cascading all of the column (or row) vectors.

- $l^2$  space:

The concept of an  $n$ -D space  $\mathbb{R}^n$  or  $\mathbb{C}^n$  can be generalized by allowing the dimension  $n$  to become to infinity so that a vector in the space becomes a sequence  $\mathbf{x} = [\dots, x_i, \dots]^T$  for  $0 \leq i < \infty$  or  $-\infty < i < \infty$ . If all vectors are square-summable, the space is denoted by  $l^2$ . All discrete energy signals are vectors in  $l^2$ .

- $\mathcal{L}^2$  space:

A vector space can also be a set of real or complex valued continuous functions  $x(t)$  defined over either a finite range such as  $0 \leq t < T$ , or an infinite range  $-\infty < t < \infty$ . If all functions are square-integrable, the space is denoted by  $\mathcal{L}^2$ . All continuous energy signals are vectors in  $\mathcal{L}^2$ .

Note that the term “vector”, generally denoted by  $\mathbf{x}$  in the following, may be interpreted in two different ways. First, in the most general sense, it represents a member of a vector space, such as any of the vector spaces considered above; e.g., a function  $\mathbf{x} = x(t) \in \mathcal{L}^2$ . Second, in a more

narrow sense, it can also represent a tuple of  $n$  elements, an  $n$ -D vector  $\mathbf{x} = [x_1, \dots, x_n]^T \in \mathbb{C}^n$ , where  $n$  may be infinity. It should be clear what a vector  $\mathbf{x}$  represents from the context.

- **Linear independence**

A set of vectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$  are linearly independent if none of them can be represented as a linear combination of the others.

**Theorem:** For any set of linearly independent vectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ , if their linear combination is zero

$$\sum_{j=1}^n c_j \mathbf{v}_j = \mathbf{0} \tag{A.3}$$

then all coefficients must be zero  $c_1 = \dots = c_n = 0$ , or  $\mathbf{c} = \mathbf{0}$ .

**Proof:** Assume  $c_k \neq 0$ , then we would get

$$\mathbf{v}_k = -\frac{1}{c_k} \sum_{j=1, j \neq k}^n c_j \mathbf{v}_j \tag{A.4}$$

i.e.,  $\mathbf{v}_k$  is a linear combination of the remaining  $n - 1$  vectors, in contradiction with the assumption that they are independent.

In particular, in the  $n$ -D Euclidean space, the  $n$  vectors can be written as  $\mathbf{v}_j = [v_{1j}, \dots, v_{nj}]^T$  ( $j = 1, \dots, n$ ), and their linear combination

$$\sum_{j=1}^n c_j \mathbf{v}_j = [\mathbf{v}_1, \dots, \mathbf{v}_n] \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} c_{11} & \dots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \dots & c_{nn} \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \mathbf{V}\mathbf{c} \tag{A.5}$$

For this to be a zero vector, i.e., for the homogeneous equation  $\mathbf{V}\mathbf{c} = \mathbf{0}$  to hold, the coefficient vector  $\mathbf{c}$  has to be zero, as  $\mathbf{V}$  is a full rank matrix. QED

- **Convex combination**

The *convex combination* of  $n$  vectors  $\mathbf{x}_1, \dots, \mathbf{x}_n$  is their sum weighted by coefficients  $\{c_1, \dots, c_n\}$  that add up to 1:

$$\sum_{i=1}^n c_i \mathbf{x}_i, \quad \sum_{i=1}^n c_i = 1 \tag{A.6}$$

The *convex hull* of these points is the set of all their convex combinations. For example, the convex hull of three points in a plane is the triangle formed by these points as vertices, in which any point is a convex combination of the three vertices.

- **Inner product**

An *inner product* in a vector space  $V$  is a function that maps two vectors  $\mathbf{x}, \mathbf{y} \in V$  to a scalar  $\langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{C}$  or  $\mathbb{R}$  and satisfies the following conditions:

– Positive definiteness:

$$\langle \mathbf{x}, \mathbf{x} \rangle \geq 0, \quad \langle \mathbf{x}, \mathbf{x} \rangle = 0 \text{ iff } \mathbf{x} = \mathbf{0}. \tag{A.7}$$

- Conjugate symmetry:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \overline{\langle \mathbf{y}, \mathbf{x} \rangle}. \quad (\text{A.8})$$

If the vector space is real, the inner product becomes *symmetric*:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle. \quad (\text{A.9})$$

- Linearity in the first variable:

$$\langle a\mathbf{x} + b\mathbf{y}, \mathbf{z} \rangle = a\langle \mathbf{x}, \mathbf{z} \rangle + b\langle \mathbf{y}, \mathbf{z} \rangle, \quad (\text{A.10})$$

where  $a, b \in \mathbb{C}$ . The linearity does not apply to the second variable unless the coefficients are real  $a, b \in \mathbb{R}$ :

$$\begin{aligned} \langle \mathbf{x}, a\mathbf{y} + b\mathbf{z} \rangle &= \overline{\langle a\mathbf{y} + b\mathbf{z}, \mathbf{x} \rangle} = \overline{a\langle \mathbf{y}, \mathbf{x} \rangle + b\langle \mathbf{z}, \mathbf{x} \rangle} = \bar{a}\langle \mathbf{x}, \mathbf{y} \rangle + \bar{b}\langle \mathbf{x}, \mathbf{z} \rangle \\ &\neq a\langle \mathbf{x}, \mathbf{y} \rangle + b\langle \mathbf{x}, \mathbf{z} \rangle \end{aligned}$$

In the special case when  $b = 0$ , we have

$$\langle a\mathbf{x}, \mathbf{y} \rangle = a\langle \mathbf{x}, \mathbf{y} \rangle, \quad \langle \mathbf{x}, a\mathbf{y} \rangle = \bar{a}\langle \mathbf{x}, \mathbf{y} \rangle. \quad (\text{A.11})$$

More generally we have

$$\left\langle \sum_n c_n \mathbf{x}_n, \mathbf{y} \right\rangle = \sum_n c_n \langle \mathbf{x}_n, \mathbf{y} \rangle, \quad \left\langle \mathbf{x}, \sum_n c_n \mathbf{y}_n \right\rangle = \sum_n \bar{c}_n \langle \mathbf{x}, \mathbf{y}_n \rangle. \quad (\text{A.12})$$

An *inner product space* is a vector space with inner product defined. In particular, when the inner product is defined,  $\mathbb{C}^n$  is called a *unitary space* and  $\mathbb{R}^n$  is called a *Euclidean space*.

Some examples of the inner product are listed below:

- In an  $n$ -D vector space, the inner product, also called the *dot product*, of two vectors  $\mathbf{x} = [x_1, \dots, x_n]^T$  and  $\mathbf{y} = [y_1, \dots, y_n]^T$  is defined as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \bar{\mathbf{y}} = \mathbf{y}^* \mathbf{x} = [x_1, \dots, x_n] \begin{bmatrix} \bar{y}_1 \\ \vdots \\ \bar{y}_n \end{bmatrix} = \sum_{i=1}^n x_i \bar{y}_i, \quad (\text{A.13})$$

where  $\mathbf{y}^* = \bar{\mathbf{y}}^T$  is the conjugate transpose of  $\mathbf{y}$ .

- In a space of 2-D matrices containing  $m \times n$  elements, the inner product of two matrices  $\mathbf{A}$  and  $\mathbf{B}$  is defined as

$$\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{i=1}^m \sum_{j=1}^n a_{ij} \bar{b}_{ij}. \quad (\text{A.14})$$

When the column (or row) vectors of  $\mathbf{A}$  and  $\mathbf{B}$  are concatenated to form two  $mn$ -D vectors, their inner product takes the same form as that of two  $n$ -D vectors.

- In a function space, the inner product of two function vectors  $\mathbf{x} = x(t)$  and  $\mathbf{y} = y(t)$  is defined as

$$\langle x(t), y(t) \rangle = \int_a^b x(t)\overline{y(t)} dt = \int_a^b y(t)\overline{x(t)} dt = \overline{\langle y(t), x(t) \rangle}. \quad (\text{A.15})$$

- The covariance of two random variables  $x$  and  $y$  can be considered as an inner product

$$\langle x, y \rangle = \sigma_{xy}^2 = E[(x - \mu_x)\overline{(y - \mu_y)}] = E(x\overline{y}) - \mu_x\overline{\mu_y} \quad (\text{A.16})$$

Specially when  $\mu_x = \mu_y = 0$ , we have

$$\langle x, y \rangle = E[x\overline{y}]. \quad (\text{A.17})$$

• **Vector norm**

In general, the norm  $\|\mathbf{x}\|$  of a vector  $\mathbf{x} \in V$  is a non-negative real scalar that measures its size or length.  $\|\mathbf{x}\| = 0$  if and only if  $\mathbf{x} = \mathbf{0}$ . There exist different definitions for vector norm, as shown in Section ???. A vector  $\mathbf{x}$  is *normalized* (becomes a *unit vector*) if  $\|\mathbf{x}\| = 1$ . Any given vector can be normalized when divided by its own norm:  $\mathbf{x}/\|\mathbf{x}\|$ .

The most widely used *2-norm* of a vector is defined as

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \langle \mathbf{x}, \mathbf{x} \rangle^{1/2}, \quad (\text{A.18})$$

The vector norm squared  $\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle$  can be considered as the energy of the vector. In particular, in an  $n$ -D unitary space, the 2-norm of a vector  $\mathbf{x} = [x_1, \dots, x_n]^T \in \mathbb{C}^n$  is:

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\mathbf{x}^T \overline{\mathbf{x}}} = \sqrt{\mathbf{x}^* \mathbf{x}} = \left( \sum_{i=1}^n x_i \overline{x_i} \right)^{1/2} = \left( \sum_{i=1}^n |x_i|^2 \right)^{1/2}. \quad (\text{A.19})$$

The total energy contained in this vector is its norm squared:

$$\mathcal{E} = \|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle = \sum_{i=1}^n |x_i|^2. \quad (\text{A.20})$$

Similarly, in a function space, the norm of a function vector  $\mathbf{x} = x(t)$  is defined as

$$\|\mathbf{x}\| = \left( \int_a^b x(t)\overline{x(t)} dt \right)^{1/2} = \left( \int_a^b |x(t)|^2 dt \right)^{1/2}, \quad (\text{A.21})$$

where the lower and upper integral limits  $a < b$  are two real numbers, which may be extended to all real values  $\mathbb{R}$  in the entire real axis  $-\infty < t < \infty$ . This norm exists only if the integral converges to a finite value; i.e.,  $x(t)$  is an *energy signal* containing finite energy:

$$\int_{-\infty}^{\infty} |x(t)|^2 dt < \infty. \quad (\text{A.22})$$

All such functions  $x(t)$  satisfying the above are square-integrable, and they form a function space denoted by  $\mathcal{L}^2(\mathbb{R})$ .

- **Cauchy-Schwarz inequality**

The Cauchy-Schwarz inequality holds for any two vectors  $\mathbf{x}$  and  $\mathbf{y}$  in an inner product space  $V$ :

$$|\langle \mathbf{x}, \mathbf{y} \rangle|^2 \leq \langle \mathbf{x}, \mathbf{x} \rangle \langle \mathbf{y}, \mathbf{y} \rangle; \quad \text{i.e.,} \quad 0 \leq |\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \|\mathbf{y}\|. \quad (\text{A.23})$$

**Proof:** If either  $\mathbf{x}$  or  $\mathbf{y}$  is zero,  $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ , the theorem holds (an equality). Otherwise, we consider the following inner product:

$$\langle \mathbf{x} - \lambda \mathbf{y}, \mathbf{x} - \lambda \mathbf{y} \rangle = \|\mathbf{x}\|^2 - \bar{\lambda} \langle \mathbf{x}, \mathbf{y} \rangle - \lambda \langle \mathbf{y}, \mathbf{x} \rangle + |\lambda|^2 \|\mathbf{y}\|^2 \geq 0, \quad (\text{A.24})$$

where  $\lambda \in \mathbb{C}$  is an arbitrary complex number, which can be assumed to be:

$$\lambda = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{y}\|^2}, \quad \text{then} \quad \bar{\lambda} = \frac{\langle \mathbf{y}, \mathbf{x} \rangle}{\|\mathbf{y}\|^2}, \quad |\lambda|^2 = \frac{|\langle \mathbf{x}, \mathbf{y} \rangle|^2}{\|\mathbf{y}\|^4}. \quad (\text{A.25})$$

Substituting these into the previous equation we get

$$\|\mathbf{x}\|^2 - \frac{|\langle \mathbf{x}, \mathbf{y} \rangle|^2}{\|\mathbf{y}\|^2} \geq 0; \quad \text{i.e.,} \quad |\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \|\mathbf{y}\|. \quad (\text{A.26})$$

The equation holds only if  $\mathbf{x} - \lambda \mathbf{y} = 0$  or  $\mathbf{x} = \lambda \mathbf{y}$ , i.e., the two vectors are linearly dependent.

- **Distance between two vectors**

The distance  $d(\mathbf{x}, \mathbf{y})$  between two vectors  $\mathbf{x}$  and  $\mathbf{y}$  is a real constant that satisfies:

- $d(\mathbf{x}, \mathbf{y}) = 0$  iff  $\mathbf{x} = \mathbf{y}$
- $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$
- $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$

In an inner-product space in which the inner product  $\langle \mathbf{x}, \mathbf{y} \rangle$  between any two vectors  $\mathbf{x}$  and  $\mathbf{y}$  is defined, the distance between the two points can be defined as the norm of the difference between the two points:

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| = \sqrt{\langle (\mathbf{x} - \mathbf{y}), (\mathbf{x} - \mathbf{y}) \rangle} \quad (\text{A.27})$$

The norm of a vector can now be seen as its distance to the origin  $\mathbf{0}$  of the space:  $\|\mathbf{x}\| = \|\mathbf{x} - \mathbf{0}\|$ .

A vector space  $V$  is a *metric space* if a distance (or *metric*)  $d(\mathbf{x}, \mathbf{y})$  between any two vector (two points)  $\mathbf{x}$  and  $\mathbf{y}$  is defined.

- **Angle between two vectors**

The *angle* between two vectors  $\mathbf{x}$  and  $\mathbf{y}$  is defined as

$$\theta = \cos^{-1} \left( \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|} \right). \quad (\text{A.28})$$

Now the inner product of  $\mathbf{x}$  and  $\mathbf{y}$  can also be written as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta \leq \|\mathbf{x}\| \|\mathbf{y}\| \quad (\text{A.29})$$

This is the Cauchy-Schwarz inequality. In particular,

- If  $\theta = 0$ ,  $\cos \theta = 1$ , then  $\mathbf{x}$  and  $\mathbf{y}$  are collinear or linearly dependent, and the inner product is maximized:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \|\mathbf{y}\| \quad (\text{A.30})$$

i.e., the Cauchy-Schwarz inequality becomes an equality.

- If  $0 < \theta < \pi/2$ ,  $0 < \cos \theta < 1$ , we get the Cauchy-Schwarz inequality:

$$\langle \mathbf{x}, \mathbf{y} \rangle < \|\mathbf{x}\| \|\mathbf{y}\| \quad (\text{A.31})$$

- If  $\theta = \pi/2$ ,  $\cos \theta = 0$ , then  $\mathbf{x}$  and  $\mathbf{y}$  are orthogonal to each other, and the inner product is minimized:

$$\langle \mathbf{x}, \mathbf{y} \rangle = 0 \quad (\text{A.32})$$

Two vectors  $\mathbf{x}$  and  $\mathbf{y}$  are *orthogonal* or *perpendicular* to each other, denoted by  $\mathbf{x} \perp \mathbf{y}$ , if their inner product is zero  $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ , i.e., the angle between them is  $\theta = \cos^{-1} 0 = \pi/2$ .

- **Projection**

The *orthogonal projection* of a vector  $\mathbf{x} \in V$  onto another vector  $\mathbf{y} \in V$  is defined as a vector

$$\mathbf{p}_y(\mathbf{x}) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{y}\|} \frac{\mathbf{y}}{\|\mathbf{y}\|} = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\langle \mathbf{y}, \mathbf{y} \rangle} \mathbf{y} = \|\mathbf{x}\| \cos \theta \frac{\mathbf{y}}{\|\mathbf{y}\|}, \quad (\text{A.33})$$

where  $\mathbf{y}/\|\mathbf{y}\|$  is the unit vector along the direction of  $\mathbf{y}$ . In particular, if  $\mathbf{y}$  is normalized with  $\|\mathbf{y}\| = 1$ , then

$$\mathbf{p}_y(\mathbf{x}) = \langle \mathbf{x}, \mathbf{y} \rangle \mathbf{y} = \|\mathbf{x}\| \cos \theta \mathbf{y} \quad (\text{A.34})$$

Note that

$$\|\mathbf{p}_y(\mathbf{x})\| = \|\mathbf{x}\| \cos \theta \quad (\text{A.35})$$

If only the magnitude of the projection is of interest, the unit vector  $\mathbf{y}/\|\mathbf{y}\|$  can be dropped:

$$p_y(\mathbf{x}) = \langle \mathbf{x}, \mathbf{y} \rangle / \|\mathbf{y}\| \quad (\text{A.36})$$

- **Cauchy space**

A sequence of points in a metric space  $x_0, x_1, x_2, \dots$  is a *Cauchy sequence* if it converges, i.e., for any  $\epsilon > 0$ , there exists an integer  $N > 0$  so that the following is true for any  $m, n > N$ :

$$d(x_m, x_n) < \epsilon \quad (\text{A.37})$$

If the limit of any Cauchy sequence of points in the space is also in the space, the space is *complete*, referred to as a *Cauchy space*.

## A.2 Orthogonal Basis and Gram-Schmidt Process

Assume the  $n$ -D vector space  $\mathbb{R}^n$  or  $\mathbb{C}^n$  is spanned by a set of  $n$  independent basis vectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ , not necessarily orthogonal, so that any vector  $\mathbf{x} \in \mathbb{R}^n$  can be represented as a linear combination of the basis vectors:

$$\mathbf{x} = \sum_{i=1}^n c_i \mathbf{v}_i = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \mathbf{V}\mathbf{c} \quad (\text{A.38})$$

where  $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]$  is an  $n \times n$  matrix composed of the  $n$  vectors and the coefficients in  $\mathbf{c} = [c_1, \dots, c_n]^T$  can be found by solving the linear equation system to get  $\mathbf{c} = \mathbf{V}^{-1}\mathbf{x}$  with complexity is  $O(n^3)$ .

These linearly independent vectors  $\mathbf{v}_1, \dots, \mathbf{v}_n$  can be converted into a set of orthogonal vectors  $\{\mathbf{u}_1, \dots, \mathbf{u}_n\}$  satisfying  $\mathbf{u}_i^T \mathbf{u}_j = 0$  ( $i \neq j$ ) by the following *Gram-Schmidt process*:

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{v}_1 \\ \mathbf{u}_2 &= \mathbf{v}_2 - \mathbf{p}_{\mathbf{u}_1}(\mathbf{v}_2) = \mathbf{v}_2 - \left( \frac{\mathbf{v}_2^T \mathbf{u}_1}{\mathbf{u}_1^T \mathbf{u}_1} \right) \mathbf{u}_1 \\ \mathbf{u}_3 &= \mathbf{v}_3 - \mathbf{p}_{\mathbf{u}_1}(\mathbf{v}_3) - \mathbf{p}_{\mathbf{u}_2}(\mathbf{v}_3) = \mathbf{v}_3 - \left( \frac{\mathbf{v}_3^T \mathbf{u}_1}{\mathbf{u}_1^T \mathbf{u}_1} \right) \mathbf{u}_1 - \left( \frac{\mathbf{v}_3^T \mathbf{u}_2}{\mathbf{u}_2^T \mathbf{u}_2} \right) \mathbf{u}_2 \\ &\dots\dots\dots \\ \mathbf{u}_k &= \mathbf{v}_k - \sum_{i=1}^{k-1} \mathbf{p}_{\mathbf{u}_i}(\mathbf{v}_k) = \mathbf{v}_k - \sum_{i=1}^{k-1} \left( \frac{\mathbf{v}_k^T \mathbf{u}_i}{\mathbf{u}_i^T \mathbf{u}_i} \right) \mathbf{u}_i \end{aligned} \quad (\text{A.39})$$

where  $\mathbf{p}_{\mathbf{u}_i}(\mathbf{v}_k)$  is the projection of  $\mathbf{v}_k$  onto  $\mathbf{u}_i$ :

$$\mathbf{p}_{\mathbf{u}_i}(\mathbf{v}_k) = \left( \frac{\mathbf{v}_k^T \mathbf{u}_i}{\mathbf{u}_i^T \mathbf{u}_i} \right) \mathbf{u}_i \quad (\text{A.40})$$

We see that  $\mathbf{u}_2$  so obtained is indeed orthogonal to all  $\mathbf{u}_1$

$$\mathbf{u}_2^T \mathbf{u}_1 = \left[ \mathbf{v}_2 - \left( \frac{\mathbf{v}_2^T \mathbf{u}_1}{\mathbf{u}_1^T \mathbf{u}_1} \right) \mathbf{u}_1 \right]^T \mathbf{u}_1 = \mathbf{v}_2^T \mathbf{u}_1 - \frac{\mathbf{v}_2^T \mathbf{u}_1}{\mathbf{u}_1^T \mathbf{u}_1} \mathbf{u}_1^T \mathbf{u}_1 = 0 \quad (\text{A.41})$$

and, by mathematical induction, we can further prove that  $\mathbf{u}_k^T \mathbf{u}_i = 0$  for all  $i = 1, \dots, k-1$ . In other words,  $\mathbf{u}_k$  is the component of  $\mathbf{v}_k$  that is orthogonal to all previously found orthogonal vectors  $\mathbf{u}_1, \dots, \mathbf{u}_{k-1}$ .

Given a set of orthogonal basis  $\{\mathbf{u}_1, \dots, \mathbf{u}_n\}$  satisfying  $\langle \mathbf{u}_i, \mathbf{u}_j \rangle = \mathbf{u}_i^T \mathbf{u}_j = 0$  for any  $i \neq j$ , we can again represent any given vector  $\mathbf{x}$  as a linear combination of these basis as

$$\mathbf{x} = \sum_{i=1}^n d_i \mathbf{u}_i \quad (\text{A.42})$$

The coefficients  $d_i$  can now be obtained easily. Premultiplying  $\mathbf{u}_j^T$  on both sides



of the equation above, we get

$$\mathbf{u}_j^T \mathbf{x} = \mathbf{u}_j^T \left( \sum_{i=1}^n d_i \mathbf{u}_i \right) = \sum_{i=1}^n d_i \mathbf{u}_j^T \mathbf{u}_i = d_j \mathbf{u}_j^T \mathbf{u}_j \quad (\text{A.43})$$

Solving for  $c_j$  we get

$$d_j = \frac{\mathbf{u}_j^T \mathbf{x}}{\mathbf{u}_j^T \mathbf{u}_j}, \quad j = 1, \dots, n \quad (\text{A.44})$$

As the  $n$  coefficients are decoupled, each of them can be obtained separately with linear computational complexity  $O(n)$  with total complexity  $O(n^2)$  for all  $n$  of them. The complexity is reduced to  $O(n^2)$  from  $O(n^3)$  for solving the equation system  $\mathbf{c} = \mathbf{V}^{-1} \mathbf{b}$ , needed in the case where the basis is not orthogonal. This is the reason why orthogonal bases are preferred in general. Now the vector  $\mathbf{x}$  can be written as

$$\mathbf{x} = \sum_{i=1}^n d_i \mathbf{u}_i = \sum_{i=1}^n \left( \frac{\mathbf{u}_i^T \mathbf{x}}{\mathbf{u}_i^T \mathbf{u}_i} \right) \mathbf{u}_i = \sum_{i=1}^n \mathbf{p}_{\mathbf{u}_i}(\mathbf{x}) \quad (\text{A.45})$$

where the  $i$ th term of the summation above is simply the projection  $\mathbf{p}_{\mathbf{u}_i}(\mathbf{x})$  of  $\mathbf{x}$  onto the  $i$ th basis vector  $\mathbf{u}_i$ .

The concept of orthogonal vectors satisfying  $\mathbf{u}^T \mathbf{v} = 0$  can be generalized to *conjugate vectors* that satisfy  $\mathbf{u}^T \mathbf{A} \mathbf{v} = 0$  with respect to a symmetric matrix  $\mathbf{A} = \mathbf{A}^T$ . This can be expressed in the form of inner product

$$\mathbf{u}^T \mathbf{A} \mathbf{v} = (\mathbf{A} \mathbf{v})^T \mathbf{u} = \mathbf{v}^T \mathbf{A} \mathbf{u} \quad (\text{A.46})$$

Conjugate vectors with respect to  $\mathbf{A}$  can also be considered as orthogonal to each other with respect to  $\mathbf{A}$ . Two orthogonal vectors satisfying  $\mathbf{u}^T \mathbf{v} = 0$  can be considered as a special case of conjugate vectors with respect to  $\mathbf{A} = \mathbf{I}$ .

Based on this generalized orthogonality, we can also define the projection of  $\mathbf{v}$  onto  $\mathbf{u}$  with respect to  $\mathbf{A}$  as:

$$\mathbf{p}_{\mathbf{u}}(\mathbf{v}) = \left( \frac{\mathbf{u}^T \mathbf{A} \mathbf{v}}{\mathbf{u}^T \mathbf{A} \mathbf{u}} \right) \mathbf{u} \quad (\text{A.47})$$

Given a set of  $\mathbf{A}$ -conjugate basis vectors  $\{\mathbf{d}_1, \dots, \mathbf{d}_n\}$  satisfying  $\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0$  for any  $i \neq j$ , we can represent any  $\mathbf{x}$  as

$$\mathbf{x} = \sum_{i=1}^n a_i \mathbf{d}_i \quad (\text{A.48})$$

The coefficients  $a_i$  can be obtained by pre-multiplying  $\mathbf{d}_j^T \mathbf{A}$  on both sides:

$$\mathbf{d}_j^T \mathbf{A} \mathbf{x} = \sum_{i=1}^n a_i \mathbf{d}_j^T \mathbf{A} \mathbf{d}_i = a_j \mathbf{d}_j^T \mathbf{A} \mathbf{d}_j \quad (\text{A.49})$$

Solving for  $a_j$  we get

$$a_j = \frac{\mathbf{d}_j^T \mathbf{A} \mathbf{x}}{\mathbf{d}_j^T \mathbf{A} \mathbf{d}_j}, \quad j = 1, \dots, n \quad (\text{A.50})$$

and  $\mathbf{x}$  is now represented as the sum of its  $\mathbf{A}$ -projections onto the  $n$  basis vectors  $\{\mathbf{d}_1, \dots, \mathbf{d}_n\}$ :

$$\mathbf{x} = \sum_{i=1}^n a_i \mathbf{d}_i = \sum_{i=1}^n \left( \frac{\mathbf{d}_i^T \mathbf{A} \mathbf{x}}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} \right) \mathbf{d}_i = \sum_{i=1}^n \mathbf{p}_{\mathbf{d}_i}(\mathbf{x}) \quad (\text{A.51})$$

A set of independent basis vectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$  can now also be converted to an orthogonal basis with respect to a symmetric vector  $\mathbf{A}$  by Gram-Schmidt process:

$$\mathbf{d}_k = \mathbf{v}_k - \sum_{i=1}^{k-1} \mathbf{p}_{\mathbf{d}_i}(\mathbf{v}_k) = \mathbf{v}_k - \sum_{i=1}^{k-1} \left( \frac{\mathbf{v}_k^T \mathbf{A} \mathbf{d}_i}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} \right) \mathbf{d}_i, \quad (k = 1, \dots, n) \quad (\text{A.52})$$

### A.3 Rank, Trace, Determinant, Transpose, and Inverse of Matrices

Let  $\mathbf{A}$  be an  $m \times n$  square matrix:

$$\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n] = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}_{m \times n} \quad (\text{A.53})$$

where

$$\mathbf{a}_j = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{bmatrix}, \quad (j = 1, \dots, n) \quad (\text{A.54})$$

is the  $j$ th column vector and  $[a_{i1} \cdots a_{in}]$  is the  $i$ th row vector ( $i = 1, \dots, m$ ). If  $m = n$ ,  $\mathbf{A}$  is a *square matrix*. In particular, if all entries of a square matrix are zero except those along the diagonal, it is a *diagonal matrix*. Moreover, if the diagonal entries of a diagonal matrix are all one, it is the *identity matrix*:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \quad (\text{A.55})$$

- **Rank**

The  $m$  row vectors span the *row space* of  $\mathbf{A}$  and the  $n$  column vectors span the *column space* of  $\mathbf{A}$ . The *rank* of each space is its dimension, the number of independent vectors in the space. The row and column spaces have the same rank, which is also the rank of matrix  $\mathbf{A}$ , i.e.:

$$r = \text{rank}(\mathbf{A}) \leq \min(m, n) \quad (\text{A.56})$$

In other words, the rank of matrix  $\mathbf{A}$  is the number of its independent rows or columns.

- **Transpose**

The *transpose*  $\mathbf{x}^T$  a column vector  $\mathbf{x}$  is a row vector:

$$\mathbf{x}^T = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}^T = [x_1, \dots, x_n] \quad (\text{A.57})$$

The transpose  $\mathbf{A}^T$  of a matrix  $\mathbf{A}$  is obtained by switching the positions of elements  $a_{ij}$  and  $a_{ji}$  for all  $i \in \{1, \dots, m\}$  and  $j \in \{1, \dots, n\}$ . In other words, the  $i$ th column of  $\mathbf{A}$  becomes the  $i$ th row of  $\mathbf{A}^T$ , or equivalently, the  $i$ th row of  $\mathbf{A}$  becomes the  $i$ th column of  $\mathbf{A}^T$ :

$$\mathbf{A}^T = [\mathbf{a}_1 \cdots \mathbf{a}_n]^T = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix} \quad (\text{A.58})$$

where vector  $\mathbf{a}_i$  is the  $i$ th column of  $\mathbf{A}$  and its transpose  $\mathbf{a}_i^T$  is the  $i$ th row of  $\mathbf{A}^T$ .

The properties of the transpose:

- $(\mathbf{A}^T)^T = \mathbf{A}$
- $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$
- $(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$
- $\mathbf{A}^T$  and  $\mathbf{A}$  have the same eigenvalues and eigenvectors.

If  $\mathbf{A} = \mathbf{A}^T$ , it is a *symmetric matrix*.

- **Conjugate transpose**

The *conjugate transpose* of a matrix  $\mathbf{A}$ , denoted by  $\mathbf{A}^*$ , is by taking the complex conjugate of the its transpose:

$$\mathbf{A}^* = (\overline{\mathbf{A}})^T = \overline{\mathbf{A}^T} \quad (\text{A.59})$$

i.e.,  $(\mathbf{A}^*)_{ij} = \overline{\mathbf{A}_{ji}}$ , the  $ij$ -th entry of  $\mathbf{A}^*$  is the complex conjugate of the  $ji$ -th entry of  $\mathbf{A}$ .

The properties of the conjugate transpose:

- $(\mathbf{A}^*)^* = \mathbf{A}$
- $(c\mathbf{A})^* = \bar{c}\mathbf{A}^*$
- $(\mathbf{A} + \mathbf{B})^* = \mathbf{A}^* + \mathbf{B}^*$
- $(\mathbf{AB})^* = \mathbf{B}^* \mathbf{A}^*$
- $\det(\mathbf{A}^*) = (\det \mathbf{A})^* = \overline{\det \mathbf{A}}$
- $\text{tr}(\mathbf{A}^*) = (\text{tr} \mathbf{A})^* = \overline{\text{tr} \mathbf{A}}$
- $(\mathbf{A}^*)^{-1} = (\mathbf{A}^{-1})^*$ .
- The eigenvalues of  $\mathbf{A}^*$  are the complex conjugates of the eigenvalues of  $\mathbf{A}$ . (But their eigenvectors are not related.)
- $\langle \mathbf{Ax}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{A}^* \mathbf{y} \rangle$

**Proof:**

$$\langle \mathbf{Ax}, \mathbf{y} \rangle = (\mathbf{Ax})^* \mathbf{y} = \mathbf{x}^* \mathbf{A}^* \mathbf{y} = \mathbf{x}^* (\mathbf{A}^* \mathbf{y}) = \langle \mathbf{x}, \mathbf{A}^* \mathbf{y} \rangle \quad (\text{A.60})$$

If  $\mathbf{A} = \mathbf{A}^*$ , it is a *Hermitian matrix*. A real Hermitian matrix is a symmetric matrix. The inverse of an invertible Hermitian matrix is also Hermitian, i.e., if  $\mathbf{A}^* = \mathbf{A}$ , then  $(\mathbf{A}^{-1})^* = \mathbf{A}^{-1}$ .

- **Trace**

The *trace* of a square matrix  $A$  is the sum of its diagonal elements:

$$\text{tr}(\mathbf{A}) = \sum_{i=1}^n a_{ii} \quad (\text{A.61})$$

The properties of the trace:

- $\text{tr}(c\mathbf{A}) = c \text{tr}(\mathbf{A})$
- $\text{tr}(\mathbf{A}^T) = \text{tr}(\mathbf{A})$
- $\text{tr}(\mathbf{A} + \mathbf{B}) = \text{tr}(\mathbf{B} + \mathbf{A})$
- $\text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA})$
- $\text{tr}(\mathbf{A}^T\mathbf{B}) = \text{tr}(\mathbf{AB}^T)$
- $\text{tr}(\mathbf{R}^{-1}\mathbf{AR}) = \text{tr}(\mathbf{R}^{-1}(\mathbf{AR})) = \text{tr}((\mathbf{AR})\mathbf{R}^{-1}) = \text{tr}(\mathbf{A})$

- **Determinant**

The *determinant* of a square matrix  $A$  is denoted by  $\det(\mathbf{A}) = |\mathbf{A}|$ , and  $\det(\mathbf{A}) \neq 0$  if and only if it is full rank, i.e.,  $\text{rank}(\mathbf{A}) = n$ .

The properties of the determinant:

- $\det(\mathbf{I}) = 1$
- $\det(\mathbf{A}^T) = \det(\mathbf{A})$
- $\det(\mathbf{A}^{-1}) = \det(\mathbf{A})^{-1}$
- $\det(\mathbf{AB}) = \det(\mathbf{A}) \det(\mathbf{B})$
- $\det(c\mathbf{A}) = c^n \det(\mathbf{A})$

- **Inverse**

If  $\mathbf{A}$  is an  $m \times n$  full rank square matrix with  $m = n = r$ , then there exists an *inverse matrix*  $\mathbf{A}^{-1}$  that satisfies  $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$ . The inverse  $\mathbf{A}^{-1}$  does not exist if  $\mathbf{A}$  is not square ( $m \neq n$ ) or full rank ( $r < m = n$ ).

The properties of the inverse:

- $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$
- $(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$
- $\det(\mathbf{A}^{-1}) = \det(\mathbf{A})^{-1}$
- $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$

For example, the solution of a linear equation system  $\mathbf{Ax} = \mathbf{b}$  of  $m$  equations and  $n = m$  variables can be obtained as  $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ .

## A.4 Normal, Unitary, and Similar Matrices

- **Commutative matrices**

Two square matrices  $\mathbf{A}$  and  $\mathbf{B}$  *commute* if  $\mathbf{AB} = \mathbf{BA}$ .

Obviously all diagonal matrices commute.

- **Normal matrix**

A square matrix  $\mathbf{A}$  is *normal* if it commutes with its conjugate transpose:  $\mathbf{A}^* \mathbf{A} = \mathbf{A} \mathbf{A}^*$ . If  $\mathbf{A}^* = \mathbf{A}^T$  is real, then  $\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T$ .

Obviously unitary matrices ( $\mathbf{A}^* = \mathbf{A}^{-1}$ ), Hermitian matrices ( $\mathbf{A}^* = \mathbf{A}$ ), and skew-Hermitian matrices ( $\mathbf{A}^* = -\mathbf{A}$ ) are all normal. But there exist normal matrices that are not Hermitian.

- **Unitary matrix**

$\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_n]$  is a *unitary matrix* if its conjugate transpose is equal to its inverse  $\mathbf{U}^* = \mathbf{U}^{-1}$ , i.e.,  $\mathbf{U}^* \mathbf{U} = \mathbf{I}$ . When a unitary matrix  $\bar{\mathbf{U}} = \mathbf{U}$  is real, it becomes an *orthogonal matrix*,  $\mathbf{U}^T = \mathbf{U}^{-1}$ .

The column (or row) vectors of a unitary matrix  $\mathbf{A}$  are *orthonormal*, i.e. they are both orthogonal and normalized:

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle = \mathbf{u}_i^T \bar{\mathbf{u}}_j = \mathbf{u}_j^* \mathbf{u}_i = \sum_k u_{ik} \bar{u}_{jk} = \delta_{ij} \triangleq \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad (\text{A.62})$$

Let  $\lambda$  and  $\mathbf{v}$  be an eigenvalue and the corresponding eigenvector of  $\mathbf{U}$ , i.e.,  $\mathbf{U}\mathbf{v} = \lambda\mathbf{v}$ , then we have

$$\mathbf{v}^* \mathbf{v} = \mathbf{v}^* \mathbf{U}^* \mathbf{U} \mathbf{v} = (\mathbf{U} \mathbf{v})^* \mathbf{U} \mathbf{v} = \lambda^2 \mathbf{v}^* \mathbf{v} \quad (\text{A.63})$$

We see that  $\lambda^2 = 1$ , i.e.,  $|\lambda| = 1$ .

A Hermitian matrix  $\mathbf{A}$  can be converted to a diagonal matrix  $\mathbf{\Lambda}$  (or diagonalized) by a particular unitary matrix  $\mathbf{U}$ :

$$\mathbf{U}^* \mathbf{A} \mathbf{U} = \mathbf{\Lambda} = \text{diag}[\lambda_1, \dots, \lambda_n] \quad (\text{A.64})$$

where  $\mathbf{\Lambda}$  is a diagonal matrix, i.e., all its off diagonal elements are 0.

- **Similar matrices**

Two matrices  $\mathbf{A}$  and  $\mathbf{B}$  are similar if they are related by  $\mathbf{A} = \mathbf{P}^{-1} \mathbf{B} \mathbf{P}$ . Similar matrices have the same determinant, trace, and eigenvalues.

## A.5 Positive/Negative (Semi)-Definite Matrices

Given a Hermitian matrix  $\mathbf{A} = \mathbf{A}^*$  and any non-zero vector  $\mathbf{x} \neq \mathbf{0}$ , we can construct a quadratic form  $\mathbf{x}^* \mathbf{A} \mathbf{x}$ . The matrix  $\mathbf{A}$  is said to be

- *positive definite*  $\mathbf{A} > 0$ , if  $\mathbf{x}^* \mathbf{A} \mathbf{x} > 0$
- *positive semi-definite*  $\mathbf{A} \geq 0$ , if  $\mathbf{x}^* \mathbf{A} \mathbf{x} \geq 0$
- *negative definite*  $\mathbf{A} < 0$ , if  $\mathbf{x}^* \mathbf{A} \mathbf{x} < 0$
- *negative semi-definite*  $\mathbf{A} \leq 0$ , if  $\mathbf{x}^* \mathbf{A} \mathbf{x} \leq 0$
- *indefinite* if there exists  $\mathbf{x}$  and  $\mathbf{y}$  such that  $\mathbf{x}^T \mathbf{A} \mathbf{x} < 0 < \mathbf{y}^T \mathbf{A} \mathbf{y}$ .

For example, consider the covariance matrix of a random vector  $\mathbf{x}$

$$\mathbf{\Sigma}_x = E[(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^*] \quad (\text{A.65})$$

The corresponding quadratic form is

$$\begin{aligned}\mathbf{v}^* \Sigma_x \mathbf{v} &= \mathbf{v}^* E[(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^*] \mathbf{v} \\ &= E[\mathbf{v}^*(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^* \mathbf{v}] = E(s^2) \geq 0\end{aligned}$$

where  $s = \mathbf{v}^*(\mathbf{x} - \mathbf{m}_x)$  is a scalar. Therefore  $\Sigma_x$  is positive semi-definite.

As  $\mathbf{A}$  is Hermitian, its eigenvalues  $\lambda_1, \dots, \lambda_n$  are real and its eigenvector matrix is unitary  $\mathbf{V}^{-1} = \mathbf{V}^*$ , by which it can be diagonalized:

$$\mathbf{V}^{-1} \mathbf{A} \mathbf{V} = \mathbf{V}^* \mathbf{A} \mathbf{V} = \Lambda, \quad \text{i.e.} \quad \mathbf{A} = \mathbf{V} \Lambda \mathbf{V}^* \quad (\text{A.66})$$

where vector  $\mathbf{y} = \mathbf{V}^* \mathbf{x}$  is a unitary transform of vector  $\mathbf{x}$ .

For any  $\mathbf{x} = [x_1, \dots, x_n]^T$ , we have

$$\begin{aligned}\mathbf{x}^* \mathbf{A} \mathbf{x} &= \mathbf{x}^* (\mathbf{V} \Lambda \mathbf{V}^*) \mathbf{x} = (\mathbf{V}^* \mathbf{x})^* \Lambda (\mathbf{V}^* \mathbf{x}) = \mathbf{y}^* \Lambda \mathbf{y} = \sum_{i=1}^n \lambda_i |y_i|^2 \\ &= \begin{cases} > 0 \text{ is positive definite} & \text{if } \lambda_i > 0 \\ \geq 0 \text{ is positive semi-definite} & \text{if } \lambda_i \geq 0 \\ < 0 \text{ is negative definite} & \text{if } \lambda_i < 0 \\ \leq 0 \text{ is negative semi-definite} & \text{if } \lambda_i \leq 0 \end{cases} \quad (\text{for all } i)\end{aligned}$$

Also, if some eigenvalues are positive and some others are negative,  $\mathbf{x}^* \mathbf{A} \mathbf{x}$  may be either positive or negative depending on  $\mathbf{x}$ , i.e.,  $\mathbf{A}$  is indefinite.

As the eigenvalues of  $\mathbf{A}^{-1}$  are  $1/\lambda_i$ ,  $i = (1, \dots, n)$ ,  $\mathbf{A}^{-1}$  and  $\mathbf{A}$  share the same positive/negative definiteness.

## A.6 Woodbury Matrix Identity and Sherman-Morrison Formula

The Woodbury matrix identity gives the inverse of an  $n \times n$  square matrix  $\mathbf{A}$  modified by a perturbation term  $\mathbf{U} \mathbf{B} \mathbf{V}^T$

$$(\mathbf{A} + \mathbf{U} \mathbf{B} \mathbf{V}^T)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{U} (\mathbf{B}^{-1} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^T \mathbf{A}^{-1} \quad (\text{A.67})$$

**Proof:**

$$\begin{aligned}& (\mathbf{A} + \mathbf{U} \mathbf{B} \mathbf{V}^T) [\mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{U} (\mathbf{B}^{-1} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^T \mathbf{A}^{-1}] \\ &= [\mathbf{I} - \mathbf{U} (\mathbf{B}^{-1} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^T \mathbf{A}^{-1}] + [\mathbf{U} \mathbf{B} \mathbf{V}^T \mathbf{A}^{-1} - \mathbf{U} \mathbf{B} \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U} (\mathbf{B}^{-1} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^T \mathbf{A}^{-1}] \\ &= \mathbf{I} + \mathbf{U} \mathbf{B} \mathbf{V}^T \mathbf{A}^{-1} - [\mathbf{U} (\mathbf{B}^{-1} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^T \mathbf{A}^{-1} + \mathbf{U} \mathbf{B} \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U} (\mathbf{B}^{-1} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^T \mathbf{A}^{-1}] \\ &= \mathbf{I} + \mathbf{U} \mathbf{B} \mathbf{V}^T \mathbf{A}^{-1} - (\mathbf{U} + \mathbf{U} \mathbf{B} \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U}) (\mathbf{B}^{-1} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^T \mathbf{A}^{-1} \\ &= \mathbf{I} + \mathbf{U} \mathbf{B} \mathbf{V}^T \mathbf{A}^{-1} - \mathbf{U} \mathbf{B} (\mathbf{B}^{-1} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U}) (\mathbf{B}^{-1} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^T \mathbf{A}^{-1} \\ &= \mathbf{I} + \mathbf{U} \mathbf{B} \mathbf{V}^T \mathbf{A}^{-1} - \mathbf{U} \mathbf{B} \mathbf{V}^T \mathbf{A}^{-1} = \mathbf{I}\end{aligned} \quad (\text{A.})$$

Consider some special cases:

- If  $\mathbf{U} = \mathbf{V} = \mathbf{I}$ , then we get

$$(\mathbf{A} + \mathbf{B})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}(\mathbf{A}^{-1} + \mathbf{B}^{-1})^{-1}\mathbf{A}^{-1} \quad (\text{A.69})$$

- If  $\mathbf{B} = \mathbf{I}$ , and let  $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_m]$  and  $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_m]$ , then we get the inverse of a rank- $m$  modified matrix:

$$(\mathbf{A} + \mathbf{U}\mathbf{V}^T)^{-1} = \left( \mathbf{A} + \sum_{i=1}^m \mathbf{u}_i \mathbf{v}_i^T \right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{U} (\mathbf{I} + \mathbf{V}^T \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^T \mathbf{A}^{-1} \quad (\text{A.70})$$

- More specially, if  $m = 1$ ,  $\mathbf{U} = \mathbf{u}$  and  $\mathbf{V} = \mathbf{v}$  become vectors, then we get the Sherman-Morrison formula:

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1} \mathbf{u} \mathbf{v}^T \mathbf{A}^{-1}}{1 + \mathbf{v}^T \mathbf{A}^{-1} \mathbf{u}} \quad (\text{A.71})$$

## A.7 Inverse and Determinant of Partitioned Symmetric Matrices

- **Theorem 1** (inverse of a partitioned symmetric matrix)

Divide an  $n \times n$  symmetric matrix  $\mathbf{A}$  into four blocks

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \quad (\text{A.72})$$

The inverse matrix  $\mathbf{B} = \mathbf{A}^{-1}$  can also be divided into four blocks:

$$\mathbf{B} = \mathbf{A}^{-1} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} \quad (\text{A.73})$$

Here we assume the dimensionalities of these blocks are:

- $\mathbf{A}_{11}$  and  $\mathbf{B}_{11}$  are  $p \times p$ ,
- $\mathbf{A}_{22}$  and  $\mathbf{B}_{22}$  are  $q \times q$ ,
- $\mathbf{A}_{12} = \mathbf{A}_{21}^T$  and  $\mathbf{B}_{12} = \mathbf{B}_{21}^T$  are  $p \times q$

with  $p + q = n$ . Then we have

$$\begin{aligned} \mathbf{B}_{11} &= (\mathbf{A}_{11} - \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \mathbf{A}_{21})^{-1} = \mathbf{A}_{11}^{-1} + \mathbf{A}_{11}^{-1} \mathbf{A}_{12} (\mathbf{A}_{22} - \mathbf{A}_{21} \mathbf{A}_{11}^{-1} \mathbf{A}_{12})^{-1} \mathbf{A}_{21} \mathbf{A}_{11}^{-1} \\ \mathbf{B}_{22} &= (\mathbf{A}_{22} - \mathbf{A}_{21} \mathbf{A}_{11}^{-1} \mathbf{A}_{12})^{-1} = \mathbf{A}_{22}^{-1} + \mathbf{A}_{22}^{-1} \mathbf{A}_{21} (\mathbf{A}_{11} - \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \mathbf{A}_{21})^{-1} \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \\ \mathbf{B}_{21} &= -\mathbf{A}_{22}^{-1} \mathbf{A}_{21} (\mathbf{A}_{11} - \mathbf{A}_{12} \mathbf{A}_{22}^{-1} \mathbf{A}_{21})^{-1} \\ \mathbf{B}_{12} &= -\mathbf{A}_{11}^{-1} \mathbf{A}_{12} (\mathbf{A}_{22} - \mathbf{A}_{21} \mathbf{A}_{11}^{-1} \mathbf{A}_{12})^{-1} \end{aligned} \quad (\text{A.74})$$

**Proof:**

$$\begin{aligned} \mathbf{I}_n &= \mathbf{A} \mathbf{A}^{-1} = \mathbf{A} \mathbf{B} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}_{11} \mathbf{B}_{11} + \mathbf{A}_{12} \mathbf{B}_{21} & \mathbf{A}_{11} \mathbf{B}_{12} + \mathbf{A}_{12} \mathbf{B}_{22} \\ \mathbf{A}_{21} \mathbf{B}_{11} + \mathbf{A}_{22} \mathbf{B}_{21} & \mathbf{A}_{21} \mathbf{B}_{12} + \mathbf{A}_{22} \mathbf{B}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_p & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_q \end{bmatrix} \end{aligned}$$

Equate each of the four blocks to get:

$$\begin{aligned}
 \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} &= \mathbf{I}_p, & \text{or } \mathbf{B}_{11} &= \mathbf{A}_{11}^{-1} - \mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{B}_{21} \\
 \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} &= \mathbf{0}, & \text{or } \mathbf{B}_{12} &= -\mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{B}_{22} \\
 \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} &= \mathbf{0}, & \text{or } \mathbf{B}_{21} &= -\mathbf{A}_{22}^{-1}\mathbf{A}_{21}\mathbf{B}_{11} \\
 \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} &= \mathbf{I}_q & \text{or } \mathbf{B}_{22} &= \mathbf{A}_{22}^{-1} - \mathbf{A}_{22}^{-1}\mathbf{A}_{21}\mathbf{B}_{12}
 \end{aligned} \tag{A.75}$$

Plug  $\mathbf{B}_{21}$  into  $\mathbf{B}_{11}$  to get:

$$\mathbf{B}_{11} = \mathbf{A}_{11}^{-1} + \mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21}\mathbf{B}_{11} \tag{A.76}$$

Solve for  $\mathbf{B}_{11}$  to get:

$$\mathbf{B}_{11} = (\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21})^{-1} \tag{A.77}$$

Applying theorem 1 to this expression, we also get the other expression in the theorem. Similarly we can get

$$\begin{aligned}
 \mathbf{B}_{22} &= (\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12})^{-1} \\
 \mathbf{B}_{21} &= -\mathbf{A}_{22}^{-1}\mathbf{A}_{21}(\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21})^{-1} \\
 \mathbf{B}_{12} &= -\mathbf{A}_{11}^{-1}\mathbf{A}_{12}(\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12})^{-1}
 \end{aligned}$$

- **Theorem 2** (Determinant of a partitioned symmetric matrix)

$$|\mathbf{A}| = \left| \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \right| = |\mathbf{A}_{22}| |\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21}| = |\mathbf{A}_{11}| |\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}| \tag{A.78}$$

**Proof:**

$$\begin{aligned}
 \mathbf{A} &= \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} \\ \mathbf{A}_{21} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}_{11}^{-1}\mathbf{A}_{12} \\ \mathbf{0} & \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \end{bmatrix} \\
 &= \begin{bmatrix} \mathbf{I} & \mathbf{A}_{12} \\ \mathbf{0} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21} & \mathbf{0} \\ \mathbf{A}_{22}^{-1}\mathbf{A}_{21} & \mathbf{I} \end{bmatrix}
 \end{aligned}$$

The theorem is proved as we also know that

$$|\mathbf{BC}| = |\mathbf{B}| |\mathbf{C}| \tag{A.79}$$

and

$$\left| \begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{X} & \mathbf{C} \end{bmatrix} \right| = \left| \begin{bmatrix} \mathbf{B} & \mathbf{X} \\ \mathbf{0} & \mathbf{C} \end{bmatrix} \right| = |\mathbf{B}| |\mathbf{C}| \tag{A.80}$$



## A.8 Unitary Transform

Given any *unitary matrix*  $\mathbf{A}$  satisfying  $\mathbf{A}\mathbf{A}^* = \mathbf{A}^*\mathbf{A} = \mathbf{I}$ , we can define a *unitary transform* of a vector  $\mathbf{x} = [x_1, \dots, x_n]^T$ :

$$\left\{ \begin{array}{l} \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \mathbf{A}^* \mathbf{x} = \begin{bmatrix} \bar{\mathbf{a}}_1^T \\ \vdots \\ \bar{\mathbf{a}}_n^T \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad (\text{forward transform}) \\ \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \mathbf{A} \mathbf{y} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^n y_i \mathbf{a}_i \quad (\text{inverse transform}) \end{array} \right. \quad (\text{A.81})$$

When  $\mathbf{A} = \bar{\mathbf{A}}$  is real,  $\mathbf{A}^{-1} = \mathbf{A}^T$  is an orthogonal matrix and the corresponding transform is an orthogonal transform.

The first equation above is the *forward transform* and can be written in component form as:

$$y_i = \bar{\mathbf{a}}_i^T \mathbf{x} = \langle \mathbf{x}, \mathbf{a}_i \rangle = \sum_{j=1}^n x_j \bar{a}_{ij}, \quad (i = 1, \dots, n) \quad (\text{A.82})$$

The transform coefficient is an inner product  $y_i = \langle \mathbf{x}, \mathbf{a}_i \rangle$ , representing the projection of vector  $\mathbf{x}$  onto the  $i$ th column vector  $\mathbf{a}_i$  of the transform matrix  $\mathbf{A}$ . The second equation is the *inverse transform* and can also be written in component form as:

$$x_j = \sum_{i=1}^n a_{ji} y_i, \quad (j = 1, \dots, n) \quad (\text{A.83})$$

By this transform, vector  $\mathbf{x}$  is represented as a linear combination (weighted sum) of the  $n$  column vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$  of matrix  $\mathbf{A}$ . Geometrically,  $\mathbf{x}$  is a point in the  $n$ -dimensional space spanned by these  $n$  orthonormal basis vectors. Each coefficient (coordinate)  $y_i$  is the projection of  $\mathbf{x}$  onto the corresponding basis vector  $\mathbf{a}_i$ .

As the  $n$ -dimensional space can be spanned by the column vectors of *any*  $n \times n$  unitary (orthogonal) matrix, a vector  $\mathbf{x}$  in the space can be represented by any of such matrices, each defining a different transform.

### Examples:

- When  $\mathbf{A} = \mathbf{I} = [\dots, \mathbf{e}_i, \dots]$  is an identity matrix, we have

$$\mathbf{x} = \sum_{i=1}^n y_i \mathbf{a}_i = \sum_{i=1}^n x_i \mathbf{e}_i \quad (\text{A.84})$$

where  $\mathbf{e}_i = [0, \dots, 0, 1, 0, \dots, 0]^T$  is the  $i$ th column of  $\mathbf{I}$  with the  $i$ th element equal 1 and all other 0.

- When  $a_{m,n} = w[m,n] = e^{-j2\pi mn/N}$ , the corresponding transform is discrete Fourier transform. The  $n$ th column vector  $\mathbf{w}_n$  of the transform matrix

$\mathbf{W} = [\mathbf{w}_0, \dots, \mathbf{w}_{N-1}]$  represents a sinusoid of a frequency  $nf_0$ , and the corresponding complex coordinate  $y_n = (\mathbf{x}, \mathbf{w}_n)$  represents the magnitude  $|y_n|$  and phase  $\angle y_n$  of this  $n$ th frequency component. The Fourier transform  $\mathbf{y} = \mathbf{W}\mathbf{x}$  represents a rotation of the coordinate system.

A unitary (orthogonal if real) transform  $\mathbf{y} = \mathbf{A}\mathbf{x}$  can be interpreted geometrically as the rotation of the vector  $\mathbf{x}$  about the origin, or equivalently, the representation of the same vector in a rotated coordinate system. A unitary transform  $\mathbf{y} = \mathbf{A}\mathbf{x}$  does not change the inner product. Let  $\mathbf{u} = \mathbf{A}^*\mathbf{x}$  and  $\mathbf{v} = \mathbf{A}^*\mathbf{y}$  be the unitary transforms of vectors  $\mathbf{x}$  and  $\mathbf{t}$ , then the inner product of these two vectors is preserved:

$$\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{A}^*\mathbf{u}, \mathbf{A}^*\mathbf{v} \rangle = \mathbf{x}^* \mathbf{A} \mathbf{A}^* \mathbf{y} = \mathbf{x}^* \mathbf{y} = \langle \mathbf{x}, \mathbf{y} \rangle \quad (\text{A.85})$$

In particular, if  $\mathbf{x} = \mathbf{y}$ , we see that the vector norm is preserved by the unitary transform:

$$\|\mathbf{u}\|^2 = \langle \mathbf{u}, \mathbf{u} \rangle = \langle \mathbf{x}, \mathbf{x} \rangle = \|\mathbf{x}\|^2 \quad (\text{A.86})$$

This is the Parseval's identity that indicates that the norm (length) of a vector is preserved under any unitary transform. If  $\mathbf{X}$  is interpreted as a signal, then its length  $\|\mathbf{x}\|^2 = \|\mathbf{y}\|^2$  represents the total energy or information contained in the signal, which is conserved during any unitary transform. However, some other features of the signal may change, e.g., the signal may be decorrelated and its total energy redistributed among its components after the transform, which may be desirable in many applications.

If  $\mathbf{x}$  is a random vector with mean vector  $\mathbf{m}_x$  and covariance matrix  $\Sigma_x$ :

$$\mathbf{m}_x = E(\mathbf{x}), \quad \Sigma_x = E(\mathbf{x}\mathbf{x}^*) - \mathbf{m}_x \mathbf{m}_x^* \quad (\text{A.87})$$

then its transform  $\mathbf{y} = \mathbf{A}^*\mathbf{x}$  has the following mean vector and covariance matrix:

$$\mathbf{m}_y = E(\mathbf{y}) = E(\mathbf{A}^*\mathbf{x}) = \mathbf{A}^* E(\mathbf{x}) = \mathbf{A}^* \mathbf{m}_x \quad (\text{A.88})$$

$$\begin{aligned} \Sigma_y &= E(\mathbf{y}\mathbf{y}^*) - \mathbf{m}_y \mathbf{m}_y^* = E[(\mathbf{A}^*\mathbf{x})(\mathbf{A}^*\mathbf{x})^*] - (\mathbf{A}^* \mathbf{m}_x)(\mathbf{A}^* \mathbf{m}_x)^* \\ &= E[\mathbf{A}^*(\mathbf{x}\mathbf{x}^*)\mathbf{A}] - \mathbf{A}^* \mathbf{m}_x \mathbf{m}_x^* \mathbf{A} = \mathbf{A}^* [E(\mathbf{x}\mathbf{x}^*) - \mathbf{m}_x \mathbf{m}_x^*] \mathbf{A} \\ &= \mathbf{A}^* \Sigma_x \mathbf{A} \end{aligned}$$

In general the unitary transform of any square matrix  $\mathbf{A}$  by a unitary matrix  $\mathbf{R}$  is

$$\mathbf{B} = \mathbf{R}^* \mathbf{A} \mathbf{R} \quad (\text{A.89})$$

## A.9 Eigenvalue Decomposition

For any  $n \times n$  square matrix  $\mathbf{A}$ , if there exist a vector  $\mathbf{v}$  and a scalar  $\lambda$  such that the following *eigenequation* holds:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}, \quad (\text{A.90})$$

then  $\lambda$  and  $\mathbf{v}$  are called the *eigenvalue* and *eigenvector* of matrix  $\mathbf{A}$ , respectively. In other words, the linear transformation of vector  $\mathbf{v}$  by  $\mathbf{A}$  has the same effect of scaling the vector by factor  $\lambda$ . (Note that for an  $m \times n$  non-square matrix  $\mathbf{A}$  with  $m \neq n$ ,  $\mathbf{A}\mathbf{v}$  is an  $m$ -D vector but  $\lambda\mathbf{v}$  is  $n$ -D vector, i.e., no eigenvalues and eigenvectors are defined.)

Given  $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ , we also have  $\mathbf{A}c\mathbf{v} = \lambda c\mathbf{v}$  for any scalar constant  $c$ , i.e., the eigenvector  $\mathbf{v}$  is not unique but up to any scaling factor. For the uniqueness of  $\mathbf{v}$ , we typically keep it normalized so that  $\|\mathbf{v}\| = 1$ .

To obtain  $\lambda$ , we rewrite the above equation as

$$(\lambda\mathbf{I} - \mathbf{A})\mathbf{v} = \mathbf{0} \quad (\text{A.91})$$

For this homogeneous equation system to have non-zero solutions for  $\mathbf{v}$ , the determinant of its coefficient matrix has to be zero:

$$\det(\lambda\mathbf{I} - \mathbf{A}) = |\lambda\mathbf{I} - \mathbf{A}| = 0 \quad (\text{A.92})$$

This is the *characteristic polynomial equation* of matrix  $\mathbf{A}$ . Solving this  $n$ th order equation of  $\lambda$ , we get  $n$  eigenvalues  $\{\lambda_1, \dots, \lambda_n\}$ . Substituting each  $\lambda_i$  back into the homogeneous equation system, we get the corresponding eigenvector  $\mathbf{v}_i$ . We can put all  $n$  eigen-equations  $\mathbf{A}\mathbf{v}_i = \lambda_i\mathbf{v}_i$  together and obtain the more compact form:

$$\mathbf{A}\mathbf{V} = \mathbf{A}[\mathbf{v}_1, \dots, \mathbf{v}_n] = [\lambda_1\mathbf{v}_1, \dots, \lambda_n\mathbf{v}_n] = [\mathbf{v}_1, \dots, \mathbf{v}_n] \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} = \mathbf{V}\mathbf{\Lambda} \quad (\text{A.93})$$

where we have defined

$$\mathbf{\Lambda} = \text{diag}[\lambda_1, \dots, \lambda_n] \quad \text{and} \quad \mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n] \quad (\text{A.94})$$

The eigen-equations can be written in some alternative forms:

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}, \quad \text{or} \quad \mathbf{V}^{-1}\mathbf{A}\mathbf{V} = \mathbf{\Lambda} \quad (\text{A.95})$$

In the first form,  $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$  is expressed as a product of three matrices, called *eigenvalue decomposition* of the matrix; in the second form,  $\mathbf{A}$  is diagonalized by its eigenvector matrix  $\mathbf{V}$  to become a diagonal matrix, its eigenvalue matrix  $\mathbf{\Lambda}$ .

Given  $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ , we have the following:

- $\mathbf{A}^T$  has the same eigenvalues and eigenvectors as  $\mathbf{A}$ .

**Proof:** As a matrix  $\mathbf{A}$  and its transpose  $\mathbf{A}^T$  have the same determinant, they have the same characteristic polynomial:

$$|\mathbf{A} - \lambda\mathbf{I}| = |(\mathbf{A} - \lambda\mathbf{I})^T| = |\mathbf{A}^T - \lambda\mathbf{I}| \quad (\text{A.96})$$

therefore they have the same eigenvalues and eigenvectors.

- The eigenvalues and eigenvectors of  $\mathbf{A}^*$  are the complex conjugate of the eigenvalues and eigenvectors as  $\mathbf{A}$ .

- $\mathbf{A}^T \mathbf{A}$  has the same eigenvectors as  $\mathbf{A}$ , but its eigenvalues are  $\lambda^2$ .

**Proof:**

$$\mathbf{A}^T \mathbf{A} \mathbf{v} = \mathbf{A}^T \lambda \mathbf{v} = \lambda^2 \mathbf{v} \quad (\text{A.97})$$

- $\mathbf{A}^k$  has the same eigenvectors as  $\mathbf{A}$ , but its eigenvalues are  $\{\lambda_1^k, \dots, \lambda_n^k\}$ , where  $k$  is a positive integer.

**Proof:**

$$\mathbf{A}^2 \mathbf{v} = \mathbf{A} \mathbf{A} \mathbf{v} = \mathbf{A} \lambda \mathbf{v} = \lambda^2 \mathbf{v} \quad (\text{A.98})$$

This result can be generalized to

$$\mathbf{A}^k \mathbf{v} = \lambda^k \mathbf{v} \quad (\text{A.99})$$

- In particular when  $k = -1$ , i.e., the eigenvalues of  $\mathbf{A}^{-1}$  are  $\{\lambda_1^{-1}, \dots, \lambda_n^{-1}\}$ .

**Proof:** Left multiplying  $\mathbf{A}^{-1}$  on both sides of  $\mathbf{A} \mathbf{v}_i = \lambda \mathbf{v}_i$  we get

$$\mathbf{A}^{-1} \mathbf{A} \mathbf{v}_i = \mathbf{I} \mathbf{v}_i = \mathbf{v}_i = \lambda_i \mathbf{A}^{-1} \mathbf{v}_i \quad (\text{A.100})$$

Dividing both sides by  $\lambda$  we get

$$\mathbf{A}^{-1} \mathbf{v}_i = \frac{1}{\lambda_i} \mathbf{v}_i = \lambda_i^{-1} \mathbf{v}_i \quad (\text{A.101})$$

- The eigenvalues of a matrix are invariant under any unitary transform  $\mathbf{B} = \mathbf{R}^* \mathbf{A} \mathbf{R}$ , where  $\mathbf{R}$  is unitary, satisfying  $\mathbf{R}^{-1} = \mathbf{R}^*$ , or  $\mathbf{R}^* \mathbf{R} = \mathbf{R} \mathbf{R}^* = \mathbf{I}$

**Proof:**

Let  $\mathbf{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_n)$  and  $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]$  be the eigenvalue and eigenvector matrices of a square matrix  $\mathbf{A}$ :

$$\mathbf{A} \mathbf{V} = \mathbf{V} \mathbf{\Lambda} \quad (\text{A.102})$$

and  $\mathbf{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_n)$  and  $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_n]$  be the eigenvalue and eigenvector matrices of  $\mathbf{B} = \mathbf{R}^* \mathbf{A} \mathbf{R}$ , a unitary transform of  $\mathbf{A}$ :

$$\mathbf{B} \mathbf{U} = (\mathbf{R}^* \mathbf{A} \mathbf{R}) \mathbf{U} = \mathbf{U} \mathbf{\Sigma} \quad (\text{A.103})$$

Left-multiplying  $\mathbf{R}$  on both sides we get the eigenequation of  $\mathbf{A}$

$$\mathbf{R} \mathbf{R}^* \mathbf{A} \mathbf{R} \mathbf{U} = \mathbf{A} (\mathbf{R} \mathbf{U}) = (\mathbf{R} \mathbf{U}) \mathbf{\Sigma} \quad (\text{A.104})$$

We see that  $\mathbf{A}$  and  $\mathbf{B} = \mathbf{R}^* \mathbf{A} \mathbf{R}$  have the same eigenvalues  $\mathbf{\Sigma} = \mathbf{\Lambda}$  and their eigenvector matrices are related by  $\mathbf{V} = \mathbf{R} \mathbf{U}$  or  $\mathbf{U} = \mathbf{R}^* \mathbf{V}$ .

- Given all eigenvalues  $\lambda_1, \dots, \lambda_n$  of a matrix  $\mathbf{A}$ , its trace and determinant can be obtained as

$$\text{tr}(\mathbf{A}) = \sum_{k=1}^n \lambda_k, \quad \det(\mathbf{A}) = \prod_{k=1}^n \lambda_k \quad (\text{A.105})$$

- The *spectrum* of an  $n \times n$  square matrix  $\mathbf{A}$  is the set of its eigenvalues  $\{\lambda_1, \dots, \lambda_n\}$ . The *spectral radius* of  $\mathbf{A}$ , denoted by  $\rho(\mathbf{A})$ , is the maximum of the absolute values of the elements in the spectrum:

$$\rho(\mathbf{A}) = \max(|\lambda_1|, \dots, |\lambda_n|) \quad (\text{A.106})$$

where  $|z| = \sqrt{x^2 + y^2}$  is the modulus of a complex number  $z = x + jy$ . If all eigenvalues are sorted such that  $|\lambda_1| \geq \dots \geq |\lambda_n|$  then  $\rho(\mathbf{A}) = |\lambda_1| = |\lambda_{max}|$ . As the eigenvalues of  $\mathbf{A}^{-1}$  are  $\{1/\lambda_{max}, \dots, 1/\lambda_{min}\}$ ,  $\rho(\mathbf{A}^{-1}) = 1/|\lambda_{min}|$ .

- If  $\mathbf{A}$  and  $\mathbf{B}$  are similar, i.e.,

$$\mathbf{B} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P} \quad (\text{A.107})$$

then they have the same eigenvalues.

**Proof:** Let  $\mathbf{\Lambda}$  and  $\mathbf{V}$  be the eigenvalue and eigenvector matrices of  $\mathbf{A}$ :

$$\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{\Lambda}, \quad \mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} \quad (\text{A.108})$$

then we have

$$\mathbf{B} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P} = \mathbf{P}^{-1}\mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}\mathbf{P} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1} \quad (\text{A.109})$$

i.e.,  $\mathbf{\Lambda}$  and  $\mathbf{U} = \mathbf{P}^{-1}\mathbf{V}$  are the eigenvalue and eigenvector matrices of  $\mathbf{B} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}$ .

- If  $\mathbf{A}$  is Hermitian  $\mathbf{A}^* = \bar{\mathbf{A}}^T = \mathbf{A}$  (symmetric  $\mathbf{A}^T = \mathbf{A}$  if real) (e.g., the covariance matrix of a random vector), then all of its eigenvalues  $\bar{\lambda}_i = \lambda_i$  are real, and all of its eigenvectors are orthogonal,  $\mathbf{v}_i^* \mathbf{v}_j = 0$  ( $i \neq j$ ) i.e.,  $\mathbf{V}^{-1} = \mathbf{V}^*$ . And we further have

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T = \mathbf{V}\mathbf{\Lambda}^{1/2}\mathbf{\Lambda}^{1/2}\mathbf{V}^T = \mathbf{U}\mathbf{U}^T \quad (\text{A.110})$$

where  $\mathbf{U} = \mathbf{V}\mathbf{\Lambda}^{1/2}$ .

**Proof:**

Let  $\lambda$  and  $\mathbf{v}$  be an eigenvalue and the corresponding eigenvector of a Hermitian matrix  $\mathbf{A} = \mathbf{A}^*$ , i.e.,  $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ , then we have

$$\begin{aligned} (\mathbf{A}\mathbf{v})^* \mathbf{v} &= (\lambda\mathbf{v})^* \mathbf{v} = \bar{\lambda}\mathbf{v}^* \mathbf{v} = \bar{\lambda} \|\mathbf{v}\|^2 \\ &= \mathbf{v}^* \mathbf{A}^* \mathbf{v} = \mathbf{v}^* \mathbf{A} \mathbf{v} = \mathbf{v}^* \lambda \mathbf{v} = \lambda \|\mathbf{v}\|^2 \end{aligned} \quad (\text{A.111})$$

i.e.,  $\bar{\lambda} = \lambda$  is real. We also have

$$\begin{aligned} \mathbf{v}_i^* \mathbf{A} \mathbf{v}_j &= \mathbf{v}_i^* \lambda_j \mathbf{v}_j = \lambda_j \mathbf{v}_i^* \mathbf{v}_j \\ &= (\mathbf{v}_j^* \mathbf{A} \mathbf{v}_i)^* = (\mathbf{v}_j^* \lambda_i \mathbf{v}_i)^* = \bar{\lambda}_i \mathbf{v}_i^* \mathbf{v}_j = \lambda_i \mathbf{v}_i^* \mathbf{v}_j \end{aligned} \quad (\text{A.112})$$

i.e.,

$$\lambda_j \mathbf{v}_i^* \mathbf{v}_j = \lambda_i \mathbf{v}_i^* \mathbf{v}_j, \quad \text{or} \quad (\lambda_i - \lambda_j) \mathbf{v}_i^* \mathbf{v}_j = 0 \quad (\text{A.113})$$

As  $\lambda_i \neq \lambda_j$ , we get  $\mathbf{v}_i^* \mathbf{v}_j = 0$ , i.e., the eigenvectors corresponding to different eigenvalues are orthogonal.

QED

When all eigenvectors are normalized  $\mathbf{v}_i^* \mathbf{v}_i = \|\mathbf{v}_i\|^2 = 1$ , they become orthonormal

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle = \mathbf{v}_i^* \mathbf{v}_j = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad (\text{A.114})$$

i.e., the eigenvector matrix  $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]^T$  is unitary (orthogonal if  $\mathbf{A}$  is real):

$$\mathbf{V}^{-1} = \mathbf{V}^* \quad \text{i.e.} \quad \mathbf{V}^* \mathbf{V} = \mathbf{I} \quad (\text{A.115})$$

and we have

$$\mathbf{V}^{-1} \mathbf{A} \mathbf{V} = \mathbf{V}^* \mathbf{A} \mathbf{V} = \mathbf{\Lambda}, \quad (\text{A.116})$$

Left and right multiplying by  $\mathbf{V}$  and  $\mathbf{V}^* = \mathbf{V}^{-1}$  respectively on the two sides, we get

$$\mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^* = [\mathbf{v}_1, \dots, \mathbf{v}_n] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^* \\ \vdots \\ \mathbf{v}_n^* \end{bmatrix} = \sum_{i=1}^n \lambda_i \mathbf{v}_i \mathbf{v}_i^* \quad (\text{A.117})$$

This is the *spectral theorem* indicating that  $\mathbf{A}$  can be written as a linear combination of  $n$  matrices  $\mathbf{v}_i \mathbf{v}_i^*$  weighted by  $\lambda_i$  ( $n = 1, \dots, n$ ).

The significance of this property is that a linear operation  $\mathbf{y} = \mathbf{A} \mathbf{x}$  applied to vector  $\mathbf{x}$  can be mapped to a new vector space in which the operations of the components are independent of each other. Consider

$$\mathbf{y} = \mathbf{A} \mathbf{x} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^* \mathbf{x} \quad (\text{A.118})$$

Pre-multiplying  $\mathbf{V}^*$  on both sides we get

$$\mathbf{y}' = \mathbf{V}^* \mathbf{y} = \mathbf{\Lambda} \mathbf{V}^* \mathbf{x} = \mathbf{\Lambda} \mathbf{x}' \quad (\text{A.119})$$

where we have defined  $\mathbf{x}' = \mathbf{V}^* \mathbf{x}$  and  $\mathbf{y}' = \mathbf{V}^* \mathbf{y}$ , the unitary transform of  $\mathbf{x}$  and  $\mathbf{y}$ , respectively. We see that for the  $i$ th component we have the following, independent of all other components:

$$y'_i = \lambda_i x'_i \quad (\text{A.120})$$

- The entries on the diagonal of an upper (or lower) triangular matrix are its eigenvalues.

**Proof:** Let  $\mathbf{A}$  be an upper triangular matrix with  $a_{ij} = 0$  for all  $i > j$ . The eigenvalues of  $\mathbf{A}$  are the roots of the following homogeneous characteristic equations:

$$\det(\mathbf{A} - \lambda \mathbf{I}) = \prod_{i=1}^n (a_{ii} - \lambda) = 0 \quad (\text{A.121})$$

The first equal sign is due to the fact that  $\mathbf{A} - \lambda \mathbf{I}$  is also an upper-triangular matrix, and the determinant of an upper-triangular matrix is the product

of all its diagonal entries. We therefore see that each diagonal entry  $a_{ii}$ , as a root of the characteristic equation, is also an eigenvalue of  $\mathbf{A}$ .

QED

- Similar matrices have the same eigenvalues.

**Proof:** Let  $\lambda$  and  $\mathbf{v}$  be an eigenvalue and the corresponding eigenvector of  $\mathbf{A}$  satisfying  $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ , and  $\mathbf{B} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}$  be a similar matrix of  $\mathbf{A}$ . We have  $\mathbf{A} = \mathbf{P}\mathbf{B}\mathbf{P}^{-1}$  and

$$\mathbf{A}\mathbf{v} = \mathbf{P}\mathbf{B}\mathbf{P}^{-1}\mathbf{v} = \lambda\mathbf{v}, \quad \text{i.e.,} \quad \mathbf{B}\mathbf{P}^{-1}\mathbf{v} = \lambda\mathbf{P}^{-1}\mathbf{v} \quad (\text{A.122})$$

In other words,  $\lambda$  is also an eigenvalue of  $\mathbf{B}$  with the corresponding eigenvector  $\mathbf{P}^{-1}\mathbf{v}$ .

QED

- All eigenvalues of a *stochastic matrix*  $\mathbf{P}$  are no greater than 1. A stochastic matrix is a matrix of which component  $P_{ij}$  is the probability for a state transition of a Markov process (chain) from  $s_i$  to  $s_j$ , i.e.,  $0 \leq P_{ij} \leq 1$  and  $\sum_j P_{ij} = 1$ .

**Proof:** First, as  $\mathbf{P}\mathbf{1} = \mathbf{1}$ ,  $\lambda = 1$  is one of the eigenvalues of  $\mathbf{P}$ . Next let  $\lambda$  and  $\mathbf{v}$  be an eigenvalue and the corresponding eigenvector of  $\mathbf{P}$ , i.e.,  $\mathbf{P}\mathbf{v} = \lambda\mathbf{v}$ , and let  $|v_k| \geq \max\{|v_1|, \dots, |v_n|\}$ . The  $k$ th row of the eigenequation equation is

$$|\lambda v_k| \leq |\lambda| |v_k| = \left| \sum_{j=1}^n P_{kj} v_j \right| \leq \sum_{j=1}^n P_{kj} |v_j| \leq |v_k| \sum_{j=1}^n P_{kj} = |v_k| \quad (\text{A.123})$$

i.e.,  $|\lambda| \leq 1$ .

QED

## A.10 Generalized Eigenvalue Problem

The *generalized eigenvalue problem* of two symmetric matrices  $\mathbf{A} = \mathbf{A}^T$  and  $\mathbf{B} = \mathbf{B}^T$  is to find a scalar  $\lambda_i$  and the corresponding vector  $\mathbf{v}_i$  for the following generalized eigen equation to hold:

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{B}\mathbf{v}_i, \quad (i = 1, \dots, n) \quad (\text{A.124})$$

or in matrix form

$$\mathbf{A}\mathbf{V} = \mathbf{B}\mathbf{V}\mathbf{\Lambda} \quad (\text{A.125})$$

The eigenvalue and eigenvector matrices  $\mathbf{\Lambda}$  and  $\mathbf{V}$  can be found in the following steps.

- Solve the eigenvalue problem of  $\mathbf{B}$  to find its diagonal eigenvalue matrix  $\mathbf{\Lambda}_B$  and orthogonal eigenvector matrix  $\mathbf{V}_B = (\mathbf{V}_B^T)^{-1}$  so that

$$\mathbf{B}\mathbf{V}_B = \mathbf{V}_B \mathbf{\Lambda}_B, \quad \text{or} \quad \mathbf{V}_B^{-1} \mathbf{B} \mathbf{V}_B = \mathbf{V}_B^T \mathbf{B} \mathbf{V}_B = \mathbf{\Lambda}_B \quad (\text{A.126})$$

- Left and right multiplying both sides of the second equation above by  $\Lambda^{-1/2}$  (whitening) we get

$$\Lambda_B^{-1/2}(\mathbf{V}_B^T \mathbf{B} \mathbf{V}_B) \Lambda_B^{-1/2} = \Lambda_B^{-1/2} \Lambda_B \Lambda_B^{-1/2} = \mathbf{I} \quad (\text{A.127})$$

We define

$$\mathbf{V}'_B = \mathbf{V}_B \Lambda_B^{-1/2} \quad (\text{A.128})$$

and the above becomes

$$(\mathbf{V}'_B)^T \mathbf{B} \mathbf{V}'_B = \mathbf{I} \quad (\text{A.129})$$

Note that  $\mathbf{V}'_B$  is not orthogonal

$$(\mathbf{V}'_B)^{-1} = (\mathbf{V}_B \Lambda_B^{-1/2})^{-1} = \Lambda_B^{1/2} \mathbf{V}_B^{-1} = \Lambda_B^{1/2} \mathbf{V}_B^T \neq \Lambda_B^{-1/2} \mathbf{V}_B^T = \mathbf{V}_B^T \quad (\text{A.130})$$

- Apply the same transform to  $\mathbf{A}$ :

$$(\mathbf{V}'_B)^T \mathbf{A} \mathbf{V}'_B = (\Lambda_B^{-1/2} \mathbf{V}_B^T) \mathbf{A} (\mathbf{V}_B \Lambda_B^{-1/2}) = \mathbf{A}' \quad (\text{A.131})$$

Note that  $\mathbf{A}'$  is symmetric as well as  $\mathbf{A}$ :

$$\mathbf{A}'^T = (\mathbf{V}_B^T \mathbf{A} \mathbf{V}_B)^T = \mathbf{V}_B^T \mathbf{A} \mathbf{V}_B = \mathbf{A}' \quad (\text{A.132})$$

- Diagonalize  $\mathbf{A}'$

As  $\mathbf{A}'$  is symmetric,, it can be diagonalized by its orthogonal eigenvector matrix  $\mathbf{V}_A$ :

$$\mathbf{V}_A^T \mathbf{A}' \mathbf{V}_A = \Lambda \quad (\text{A.133})$$

i.e.,

$$\begin{aligned} \mathbf{V}_A^T (\Lambda_B^{-1/2} \mathbf{V}_B^T \mathbf{A} \mathbf{V}_B \Lambda_B^{-1/2}) \mathbf{V}_A &= (\mathbf{V}_A^T \Lambda_B^{-1/2} \mathbf{V}_B^T) \mathbf{A} (\mathbf{V}_B \Lambda_B^{-1/2} \mathbf{V}_A) \\ &= \mathbf{V}^T \mathbf{A} \mathbf{V} = \Lambda \end{aligned}$$

where we have defined

$$\mathbf{V} = \mathbf{V}_B \Lambda_B^{-1/2} \mathbf{V}_A \quad (\text{A.134})$$

which is not orthogonal:

$$\mathbf{V}^{-1} = (\mathbf{V}_B \Lambda_B^{-1/2} \mathbf{V}_A)^{-1} = \mathbf{V}_A^{-1} \Lambda_B^{1/2} \mathbf{V}_B^{-1} = \mathbf{V}_A^T \Lambda_B^{1/2} \mathbf{V}_B^T \neq \mathbf{V}_A^T \Lambda_B^{-1/2} \mathbf{V}_B^T = \mathbf{V}^T \quad (\text{A.135})$$

- This  $\mathbf{V}$  also diagonalizes  $\mathbf{B}$ :

$$\begin{aligned} \mathbf{V}^T \mathbf{B} \mathbf{V} &= (\mathbf{V}_B \Lambda_B^{-1/2} \mathbf{V}_A)^T \mathbf{B} (\mathbf{V}_B \Lambda_B^{-1/2} \mathbf{V}_A) = \mathbf{V}_A^T \Lambda_B^{-1/2} (\mathbf{V}_B^T \mathbf{B} \mathbf{V}_B) \Lambda_B^{-1/2} \mathbf{V}_A \\ &= \mathbf{V}_A^T \Lambda_B^{-1/2} \Lambda_B \Lambda_B^{-1/2} \mathbf{V}_A = \mathbf{V}_A^T \mathbf{V}_A = \mathbf{I} \end{aligned}$$

- Now we have

$$\begin{cases} \mathbf{V}^T \mathbf{A} \mathbf{V} = \Lambda \\ \mathbf{V}^T \mathbf{B} \mathbf{V} = \mathbf{I} \end{cases} \quad (\text{A.136})$$



Right multiplying both sides of the second equation by  $\mathbf{\Lambda}$  we get  $\mathbf{V}^T \mathbf{B} \mathbf{V} \mathbf{\Lambda} = \mathbf{\Lambda}$ , then equating the left-hand side to that of the first equation, we get

$$\mathbf{V}^T \mathbf{A} \mathbf{V} = \mathbf{V}^T \mathbf{B} \mathbf{V} \mathbf{\Lambda}, \quad \text{i.e.} \quad \mathbf{A} \mathbf{V} = \mathbf{B} \mathbf{V} \mathbf{\Lambda} \quad (\text{A.137})$$

i.e.,  $\mathbf{\Lambda}$  and  $\mathbf{V}$  are the eigenvalue and eigenvector matrices of the generalized eigenvalue problem. Note, however, as shown above,  $\mathbf{V}$  is not orthogonal.

## A.11 Rayleigh Quotient

The *Rayleigh quotient*  $R(\mathbf{A})$  of a hermitian matrix  $\mathbf{A} = \mathbf{A}^*$  (symmetric if  $\mathbf{A}$  is real) is defined as

$$R(\mathbf{A}) = \frac{\mathbf{w}^* \mathbf{A} \mathbf{w}}{\mathbf{w}^* \mathbf{w}} \quad (\text{A.138})$$

where  $\mathbf{w}$  is any non-zero vector. In particular, if  $\mathbf{w} = \mathbf{v}_i$  is the  $i$ th eigenvector of  $\mathbf{A}$ , then the Rayleigh quotient is the corresponding to eigenvalue  $\lambda_i$ :

$$R(\mathbf{A}) = \frac{\mathbf{v}_i^* \mathbf{A} \mathbf{v}_i}{\mathbf{v}_i^* \mathbf{v}_i} = \frac{\lambda_i \mathbf{v}_i^* \mathbf{v}_i}{\mathbf{v}_i^* \mathbf{v}_i} = \lambda_i \quad (\text{A.139})$$

The *Rayleigh quotient* of two symmetric matrices  $\mathbf{A}$  and  $\mathbf{B}$  is a function of a vector  $\mathbf{w}$  defined as:

$$R(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{A} \mathbf{w}}{\mathbf{w}^T \mathbf{B} \mathbf{w}} \quad (\text{A.140})$$

To find the optimal  $\mathbf{w}$  corresponding to the extremum (maximum or minimum) of  $R(\mathbf{w})$ , we find its derivative with respect to  $\mathbf{w}$ :

$$\frac{d}{d\mathbf{w}} R(\mathbf{w}) = \frac{2\mathbf{A}\mathbf{w}(\mathbf{w}^T \mathbf{B} \mathbf{w}) - 2\mathbf{B}\mathbf{w}(\mathbf{w}^T \mathbf{A} \mathbf{w})}{(\mathbf{w}^T \mathbf{B} \mathbf{w})^2} \quad (\text{A.141})$$

Setting it to zero we get

$$\mathbf{A}\mathbf{w}(\mathbf{w}^T \mathbf{B} \mathbf{w}) = \mathbf{B}\mathbf{w}(\mathbf{w}^T \mathbf{A} \mathbf{w}), \quad \text{i.e.} \quad \mathbf{A}\mathbf{w} = \frac{\mathbf{w}^T \mathbf{A} \mathbf{w}}{\mathbf{w}^T \mathbf{B} \mathbf{w}} \mathbf{B}\mathbf{w} = R(\mathbf{w}) \mathbf{B}\mathbf{w} = \lambda \mathbf{B}\mathbf{w} \quad (\text{A.142})$$

The second equation can be recognized as a generalized eigenvalue problem with  $\lambda = R(\mathbf{w})$  being the eigenvalue and  $\mathbf{w}$  the corresponding eigenvector. Solving this we get the vector  $\mathbf{w}$  corresponding to the maximum/minimum eigenvalue  $\lambda = R(\mathbf{w})$ , which maximizes/minimizes the Rayleigh quotient.

## A.12 Normal Matrices and Diagonalizability

**Theorem:** The product of two unitary matrices is unitary.

**Proof:** Let  $\mathbf{U}$  and  $\mathbf{V}$  be unitary, i.e.,  $\mathbf{U}^* = \mathbf{U}^{-1}$  and  $\mathbf{V}^* = \mathbf{V}^{-1}$ , then  $\mathbf{UV}$  is unitary:

$$(\mathbf{UV})^* = \mathbf{V}^* \mathbf{U}^* = \mathbf{V}^{-1} \mathbf{U}^{-1} = (\mathbf{UV})^{-1} \quad (\text{A.143})$$

**Theorem:** Two square matrices  $\mathbf{A}$  and  $\mathbf{B}$  are simultaneously diagonalizable if and only if they commute.

**Proof**

- Let  $\mathbf{A}$  and  $\mathbf{B}$  be simultaneously diagonalizable by  $\mathbf{R}$

$$\mathbf{R}^{-1}\mathbf{A}\mathbf{R} = \mathbf{\Lambda}_A, \quad \mathbf{R}^{-1}\mathbf{B}\mathbf{R} = \mathbf{\Lambda}_B \quad (\text{A.144})$$

then

$$\mathbf{AB} = (\mathbf{R}\mathbf{\Lambda}_A\mathbf{R}^{-1})(\mathbf{R}\mathbf{\Lambda}_B\mathbf{R}^{-1}) = (\mathbf{R}\mathbf{\Lambda}_A\mathbf{\Lambda}_B\mathbf{R}^{-1}) = (\mathbf{R}\mathbf{\Lambda}_B\mathbf{\Lambda}_A\mathbf{R}^{-1}) = (\mathbf{R}\mathbf{\Lambda}_B\mathbf{R}^{-1})(\mathbf{R}\mathbf{\Lambda}_A\mathbf{R}^{-1}) \quad (\text{A.145})$$

- Let  $\mathbf{A}$  and  $\mathbf{B}$  commute, i.e.,  $\mathbf{AB} = \mathbf{BA}$ . Assuming  $\mathbf{u}$  is an eigenvector of  $\mathbf{A}$  corresponding to eigenvalue  $\lambda$ , i.e.,  $\mathbf{A}\mathbf{u} = \lambda\mathbf{u}$ , then

$$\mathbf{A}\mathbf{B}\mathbf{u} = \mathbf{B}(\mathbf{A}\mathbf{u}) = \lambda\mathbf{B}\mathbf{u} \quad (\text{A.146})$$

we see that  $\mathbf{B}\mathbf{u}$  is also an eigenvector of  $\mathbf{A}$  corresponding to the same eigenvalue  $\lambda$ , i.e.,  $\mathbf{B}\mathbf{u}$  must be a scaled version of  $\mathbf{u}$  (in the same 1-D space):  $\mathbf{B}\mathbf{u} = \gamma\mathbf{u}$ , i.e.,  $\mathbf{u}$  is also an eigenvector of  $\mathbf{B}$ .

**Theorem:** A matrix is normal if and only if it is unitarily diagonalizable.

**Proof**

- If  $\mathbf{A}$  is unitarily diagonalizable:

$$\mathbf{A}\mathbf{U} = \mathbf{U}\mathbf{\Lambda}, \quad \mathbf{U}^{-1}\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{\Lambda}, \quad \mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^* \quad (\text{A.147})$$

where  $\mathbf{U}^* = \mathbf{U}^{-1}$  is unitary and  $\mathbf{\Lambda}$  is a diagonal matrix satisfying  $\mathbf{\Lambda}^*\mathbf{\Lambda} = \mathbf{\Lambda}\mathbf{\Lambda}^*$ , then  $\mathbf{A}$  is normal:

$$\begin{aligned} \mathbf{A}\mathbf{A}^* &= (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^*)(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^*)^* = (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^*)(\mathbf{U}\mathbf{\Lambda}^*\mathbf{U}^*) = \mathbf{U}\mathbf{\Lambda}\mathbf{\Lambda}^*\mathbf{U}^* \\ &= \mathbf{U}\mathbf{\Lambda}^*\mathbf{\Lambda}\mathbf{U}^* = (\mathbf{U}\mathbf{\Lambda}^*\mathbf{U}^*)(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^*) = \mathbf{A}^*\mathbf{A} \end{aligned}$$

- If  $\mathbf{A}$  is normal, then it is diagonalizable by a unitary matrix. First we show any matrix  $\mathbf{A}$  can be written as

$$\mathbf{A} = \mathbf{B} + i\mathbf{C} \quad (\text{A.148})$$

where

$$\mathbf{B} = \frac{1}{2}(\mathbf{A} + \mathbf{A}^*) = \mathbf{B}^*, \quad \mathbf{C} = -\frac{1}{2}(\mathbf{A} - \mathbf{A}^*) = \mathbf{C}^*, \quad (\text{A.149})$$

are both Hermitian, and diagonalizable by a unitary matrix. As  $\mathbf{A}$  is normal, we have

$$0 = \mathbf{A}\mathbf{A}^* - \mathbf{A}^*\mathbf{A} = (\mathbf{B} + i\mathbf{C})(\mathbf{B} - i\mathbf{C}) - (\mathbf{B} - i\mathbf{C})(\mathbf{B} + i\mathbf{C}) = 2i(\mathbf{C}\mathbf{B} - \mathbf{B}\mathbf{C}) \quad (\text{A.150})$$

We see that  $\mathbf{C}\mathbf{B} = \mathbf{B}\mathbf{C}$ , i.e.,  $\mathbf{B}$  and  $\mathbf{C}$  commute, and they can be simultaneously diagonalized by some unitary matrix  $\mathbf{U}$ :

$$\mathbf{U}^*\mathbf{B}\mathbf{U} = \mathbf{\Lambda}_B, \quad \mathbf{U}^*\mathbf{C}\mathbf{U} = \mathbf{\Lambda}_C, \quad (\text{A.151})$$

and so can  $\mathbf{A} = \mathbf{B} + i\mathbf{C}$ :

$$\mathbf{U}^* \mathbf{A} \mathbf{U} = \mathbf{U}^* (\mathbf{B} + i\mathbf{C}) \mathbf{U} = \boldsymbol{\Lambda}_B + i\boldsymbol{\Lambda}_C = \boldsymbol{\Lambda}_A \quad (\text{A.152})$$

Q.E.D.

If  $\mathbf{A}$  is normal, i.e., it commutes with its conjugate transpose  $\mathbf{A}^* \mathbf{A} = \mathbf{A} \mathbf{A}^*$ , then  $\mathbf{A}$  and  $\mathbf{A}^*$  can simultaneously be diagonalized by their unitary eigenvector matrix  $\mathbf{U}$ :

$$\mathbf{A} = \mathbf{U} \boldsymbol{\Lambda} \mathbf{U}^*, \quad \mathbf{A}^* = \mathbf{U} \bar{\boldsymbol{\Lambda}} \mathbf{U}^* \quad (\text{A.153})$$

## A.13 Centering Matrix

The centering matrix is defined as

$$\mathbf{C}_n = \mathbf{I}_n - \frac{1}{n} \mathbf{1} \mathbf{1}^T \quad (\text{A.154})$$

where  $\mathbf{1} = [1, \dots, 1]^T$  and  $\mathbf{1} \mathbf{1}^T$  is an  $n \times n$  matrix of all 1's. For example,

$$\mathbf{C}_1 = 0; \quad \mathbf{C}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}; \quad (\text{A.155})$$

$$\mathbf{C}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \quad (\text{A.156})$$

When  $\mathbf{C}_n$  is applied to an  $n$ -D vector  $\mathbf{v} = [v_1, \dots, v_n]^T$ , we get

$$\mathbf{C}_n \mathbf{v} = \mathbf{v} - \frac{1}{n} (\mathbf{1} \mathbf{1}^T) \mathbf{v} = \mathbf{v} - \frac{1}{n} \sum_{i=1}^n v_i \mathbf{1} = \mathbf{v} - \bar{v} \mathbf{1} \quad (\text{A.157})$$

where  $\bar{v}$  is the mean of all components of  $\mathbf{v}$ :

$$\bar{v} = \frac{1}{n} \mathbf{1}^T \mathbf{v} = \frac{1}{n} \sum_{i=1}^n v_i \quad (\text{A.158})$$

We see that the mean of  $\mathbf{v}$  is removed from the resulting vector  $\mathbf{C}_n \mathbf{v}$ .

**Example:**  $\mathbf{v} = [3, -2, 5]^T$ , with mean  $\bar{v} = (3 - 2 + 5)/3 = 2$ ,

$$\mathbf{u} = \mathbf{C}_3 \mathbf{v} = \frac{1}{3} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ -2 \\ 5 \end{bmatrix} = \begin{bmatrix} 3 \\ -2 \\ 5 \end{bmatrix} - \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \\ 3 \end{bmatrix} \quad (\text{A.159})$$

The mean of the resulting vector  $\mathbf{u}$  is  $\bar{u} = 0$ . In particular, if  $\mathbf{v} = \mathbf{1}$ , then

$$\mathbf{C}_n \mathbf{v} = \mathbf{C}_n \mathbf{1} = \left[ \mathbf{I} - \frac{1}{n} \mathbf{1} \mathbf{1}^T \right] \mathbf{1} = \mathbf{1} - \frac{1}{n} \mathbf{1} \mathbf{1}^T \mathbf{1} = \mathbf{1} - \frac{1}{n} \mathbf{1} n = \mathbf{0} \quad (\text{A.160})$$

i.e.,  $\mathbf{C}_n \mathbf{1} = \mathbf{0}$ . We therefore see that  $\lambda = 0$  and  $\mathbf{v} = \mathbf{1}$  are the eigenvalue and corresponding eigenvector of  $\mathbf{C}_n$ . Also, consider any zero-mean vector  $\mathbf{v} = [v_1, \dots, v_n]$  satisfying  $\sum_{i=1}^n v_i = \mathbf{1}^T \mathbf{v} = 0$  with  $n - 1$  degrees of freedom, we have

$$\mathbf{C}_n \mathbf{v} = \left[ \mathbf{I} - \frac{1}{n} \mathbf{1} \mathbf{1}^T \right] \mathbf{v} = \mathbf{v} - \frac{1}{n} \mathbf{1} \mathbf{1}^T \mathbf{v} = \mathbf{v} \quad (\text{A.161})$$

i.e.,  $\mathbf{C}_n \mathbf{v} = \mathbf{1} \mathbf{v}$ . We therefore see that  $\lambda = 1$  and any  $\mathbf{v}$  satisfying  $\mathbf{1}^T \mathbf{v} = 0$  are the eigenvalue of multiplicity  $n - 1$  and the corresponding eigenvector of  $\mathbf{C}_n$ .

The centering matrix  $\mathbf{C}_n$  has the following properties:

- Idempotence:  $\mathbf{C}_n^k = \mathbf{C}_n$

$$\mathbf{C}_n^2 = \mathbf{C}_n \mathbf{C}_n = \left( \mathbf{I}_n - \frac{1}{n} \mathbf{1} \mathbf{1}^T \right) \left( \mathbf{I}_n - \frac{1}{n} \mathbf{1} \mathbf{1}^T \right) = \mathbf{I}_n - \frac{2}{n} \mathbf{1} \mathbf{1}^T + \frac{n}{n^2} \mathbf{1} \mathbf{1}^T = \mathbf{C}_n \quad (\text{A.162})$$

Once the mean of vector  $\mathbf{v}$  is removed, its mean is zero and subsequent removal of mean has not effect.

- $\mathbf{C}_n$  has eigenvalue  $\lambda = 1$  of multiplicity  $n - 1$  and eigenvalue  $\lambda = 0$  of multiplicity 1.
- $\mathbf{C}_n$  is singular, its inverse does not exist. Once the mean of vector  $\mathbf{v}$  is removed, it cannot be reconstructed by an inverse process.

## A.14 Singular Value Decomposition

The eigenvalue decomposition of a square matrix  $\mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^{-1}$  is of great importance and widely used. However, for an  $m \times n$  non-square matrix  $\mathbf{A}$ , no eigenvalues and eigenvector exist. In this case, we can still find its *singular values* and the corresponding left and right *singular vectors*, and then carry out *singular value decomposition (SVD)*.

**Theorem:** An  $m \times n$  matrix  $\mathbf{A}$  of rank  $r \leq \min(m, n)$  can be diagonalized by two orthogonal matrices  $\mathbf{U}_{m \times m}$  and  $\mathbf{V}_{n \times n}$ :

$$\mathbf{U}^T \mathbf{A} \mathbf{V} = \mathbf{\Sigma} \quad \text{or} \quad \mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \quad (\text{A.163})$$

where

- $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_m]$  is an  $m \times m$  matrix composed of the orthogonal eigenvectors of the symmetric matrix  $\mathbf{A} \mathbf{A}^T$ , also called the *left singular vectors* of  $\mathbf{A}$ :

$$\mathbf{U}^T (\mathbf{A} \mathbf{A}^T) \mathbf{U} = \mathbf{\Lambda} \quad \text{or} \quad \mathbf{A} \mathbf{A}^T = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T \quad (\text{A.164})$$

where  $\mathbf{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_n)$  is the eigenvalue matrix of  $\mathbf{A} \mathbf{A}^T$ .

- $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]$  is an  $n \times n$  matrix composed of the orthogonal eigenvectors of the symmetric matrix  $\mathbf{A}^T \mathbf{A}$ , also called the *right singular vectors* of  $\mathbf{A}$ :

$$\mathbf{V}^T (\mathbf{A}^T \mathbf{A}) \mathbf{V} = \mathbf{\Lambda} \quad \text{or} \quad \mathbf{A}^T \mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T \quad (\text{A.165})$$

Note that  $\mathbf{A}^T\mathbf{A}$  and  $\mathbf{A}\mathbf{A}^T$  have the same eigenvalue matrix  $\mathbf{\Lambda}$  (see proof below).

- $\mathbf{\Sigma} = \mathbf{\Lambda}^{1/2}$  is an  $m \times n$  diagonal matrix of  $r$  non-zero values  $\sigma_i = \sqrt{\lambda_i}$  ( $i = 1, \dots, r$ ), the square roots of the eigenvalues of  $\mathbf{A}\mathbf{A}^T$  or  $\mathbf{A}^T\mathbf{A}$ , defined as the *singular values* of  $\mathbf{A}$ :

$$\mathbf{\Lambda}^{1/2} = \begin{bmatrix} \sqrt{\lambda_1} & & & \\ & \ddots & & \\ & & \sqrt{\lambda_r} & \\ & & & \end{bmatrix} = \begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_r & \\ & & & \end{bmatrix} = \mathbf{\Sigma} \quad (\text{A.166})$$

**Proof:**

Pre-multiplying  $\mathbf{A}$  on both sides of the eigenequation of  $\mathbf{A}^T\mathbf{A}$ :

$$(\mathbf{A}^T\mathbf{A})\mathbf{V} = \mathbf{V}\mathbf{\Lambda} \quad (\text{A.167})$$

we get

$$(\mathbf{A}\mathbf{A}^T)\mathbf{A}\mathbf{V} = \mathbf{A}\mathbf{V}\mathbf{\Lambda}, \quad (\text{A.168})$$

Note that  $\mathbf{A}^T\mathbf{A}$  and  $\mathbf{A}\mathbf{A}^T$  have the same eigenvalue matrix  $\mathbf{\Lambda}$ , and  $\mathbf{A}\mathbf{V}$  is the eigenvector matrix of  $\mathbf{A}\mathbf{A}^T$ . As the columns (or rows) of  $\mathbf{A}\mathbf{V}$  are not normalized,  $\mathbf{A}\mathbf{V}$  is not orthogonal:

$$(\mathbf{A}\mathbf{V})^T(\mathbf{A}\mathbf{V}) = \mathbf{V}^T\mathbf{A}^T\mathbf{A}\mathbf{V} = \mathbf{V}^T\mathbf{V}\mathbf{\Lambda} = \mathbf{\Lambda} \neq \mathbf{I} \quad (\text{A.169})$$

If we post-multiply  $\mathbf{A}\mathbf{V}$  by  $\mathbf{\Lambda}^{-1/2}$ , its  $i$ th column is scaled by  $1/\sqrt{\lambda_i}$  ( $i = 1, \dots, n$ ) and thereby normalized, the resulting matrix becomes orthogonal:

$$\begin{aligned} (\mathbf{A}\mathbf{V}\mathbf{\Lambda}^{-1/2})^T(\mathbf{A}\mathbf{V}\mathbf{\Lambda}^{-1/2}) &= \mathbf{\Lambda}^{-1/2}\mathbf{V}^T\mathbf{A}^T\mathbf{A}\mathbf{V}\mathbf{\Lambda}^{-1/2} \\ &= \mathbf{\Lambda}^{-1/2}\mathbf{V}^T\mathbf{V}\mathbf{\Lambda}\mathbf{\Lambda}^{-1/2} = \mathbf{\Lambda}^{-1/2}\mathbf{\Lambda}\mathbf{\Lambda}^{-1/2} = \mathbf{I} \end{aligned}$$

We denote this matrix composed of normalized orthogonal eigenvectors of  $\mathbf{A}\mathbf{A}^T$  by

$$\mathbf{A}\mathbf{V}\mathbf{\Lambda}^{-1/2} = \mathbf{U} \quad (\text{A.170})$$

Post-multiplying  $(\mathbf{V}\mathbf{\Lambda}^{-1/2})^{-1} = \mathbf{\Lambda}^{1/2}\mathbf{V}^{-1}$  on both sides, we get the SVD of  $\mathbf{A}$ :

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}^{1/2}\mathbf{V}^{-1} = \mathbf{U}\mathbf{\Lambda}^{1/2}\mathbf{V}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (\text{A.171})$$

Q.E.D.

The SVD is illustrated graphically below for the two cases of  $m < n$  and  $m > n$ :

The component form of  $\mathbf{A}\mathbf{V}\mathbf{\Lambda}^{-1/2} = \mathbf{U}$  above is

$$\mathbf{A}\mathbf{v}_i \frac{1}{\sqrt{\lambda_i}} = \mathbf{u}_i \quad (i = 1, \dots, n) \quad (\text{A.172})$$

relating the eigenvectors  $\mathbf{v}_i$  of  $\mathbf{A}^T\mathbf{A}$  to the eigenvectors  $\mathbf{u}_i$  of  $\mathbf{A}\mathbf{A}^T$ .

We further consider some special cases:

- If  $\mathbf{A}^T = \mathbf{A}^{-1}$  is unitary, i.e.,  $\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T = \mathbf{I}$ , its eigenvalue matrix is  $\mathbf{\Lambda} = \mathbf{I}$ , and all singular values are unity  $\sigma_i = 1$  for all  $i = 1, \dots, n$ .
- If  $\mathbf{A}^T = \mathbf{A}$  is symmetric, i.e.,  $\mathbf{A} \mathbf{A}^T = \mathbf{A}^T \mathbf{A} = \mathbf{A}^2$ , and its singular value decomposition is  $\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$ , i.e.,

$$\mathbf{A} \mathbf{A}^T = \mathbf{U} \mathbf{\Sigma}^2 \mathbf{U}^T, \quad \mathbf{A}^T \mathbf{A} = \mathbf{V} \mathbf{\Sigma}^2 \mathbf{V} \quad (\text{A.173})$$

On the other hand, the eigenvalue decomposition of  $\mathbf{A}^2 = \mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T$  can be written as:

$$\mathbf{A}^2 = \mathbf{W} \mathbf{\Lambda}^2 \mathbf{W} \quad (\text{A.174})$$

where  $\mathbf{\Lambda}$  is the eigenvalue matrix of  $\mathbf{A}$ . We see that  $\mathbf{\Lambda}^2 = \mathbf{\Sigma}^2$ , i.e., the singular values are the absolute values of its eigenvalues  $\sigma_i = \sqrt{\lambda_i^2} = |\lambda_i|$  ( $i = 1, \dots, n$ ), and the left and right eigenvectors are the same as the eigenvectors  $\mathbf{U} = \mathbf{V} = \mathbf{W}$ .

- If  $\mathbf{A}$  is normal satisfying  $\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T$  (e.g., symmetric, Hermitian, unitary, etc.), both  $\mathbf{A}$  and  $\mathbf{A}^T$  can be simultaneously diagonalized by their unitary eigenvector matrix  $\mathbf{U} = \mathbf{V}$ , i.e.,

$$\mathbf{A} \mathbf{u}_i = \lambda_i \mathbf{u}_i, \quad \mathbf{A} \mathbf{u}_i = \overline{\lambda_i} \mathbf{u}_i, \quad (i = 1, \dots, n) \quad (\text{A.175})$$

or in matrix form

$$\mathbf{A} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T, \quad \mathbf{A}^T = \mathbf{U} \overline{\mathbf{\Lambda}} \mathbf{U}^T \quad (\text{A.176})$$

we have

$$\mathbf{A} \mathbf{A}^T = (\mathbf{U} \mathbf{\Lambda} \mathbf{U}^T)(\mathbf{U} \overline{\mathbf{\Lambda}} \mathbf{U}^T) = \mathbf{U} \mathbf{\Lambda} \overline{\mathbf{\Lambda}} \mathbf{U}^T = \mathbf{U} |\mathbf{\Lambda}|^2 \mathbf{U}^T \quad (\text{A.177})$$

i.e., the singular values of any norm matrix  $\mathbf{A}$  are simply the modulus of its eigenvalues  $\sigma_i(\mathbf{A}) = |\lambda_i(\mathbf{A})|$  ( $i = 1, \dots, n$ ).

The  $m \times n$  matrix  $\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$  can be considered as a linear transformation that converts a vector  $\mathbf{x} \in \mathbb{C}^n$  to another vector  $\mathbf{y} \in \mathbb{C}^m$  in three steps:

$$\mathbf{y} = \mathbf{A} \mathbf{x} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \mathbf{x} \quad (\text{A.178})$$

1. Rotate vector  $\mathbf{x}$  by the orthogonal matrix  $\mathbf{V}^T$ :

$$\mathbf{y}_1 = \mathbf{V}^T \mathbf{x}. \quad (\text{A.179})$$

2. Scale each dimension  $Y_k$  of  $\mathbf{y}_1$  by a factor of  $\sigma_k$  ( $k = 1, \dots, r$ ):

$$\mathbf{y}_2 = \mathbf{\Sigma} \mathbf{y}_1 = \mathbf{\Sigma} \mathbf{V}^T \mathbf{x}. \quad (\text{A.180})$$

3. Rotate vector  $\mathbf{y}_2$  by the orthogonal matrix  $\mathbf{U}$ :

$$\mathbf{y} = \mathbf{U} \mathbf{y}_2 = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \mathbf{x} = \mathbf{A} \mathbf{x}. \quad (\text{A.181})$$

The figure below illustrates the transformation of the three vertices of a triangle in 2-D space by a matrix  $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ , which first rotates the vertices by 45 degrees CCW, scale horizontally and vertically by a factor of 3 and 2, respectively, and then rotate CW by 30 degrees.

The SVD has many applications, two of which are considered below.

The SVD equation

$$\mathbf{U}^T \mathbf{A} \mathbf{V} = \mathbf{\Lambda}^{1/2} = \mathbf{\Sigma} \tag{A.182}$$

can be considered as the forward SVD transform. Pre-multiplying  $\mathbf{U}$  and post-multiplying  $\mathbf{V}^T$  on both sides, we get the inverse transform:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = [\mathbf{u}_1, \dots, \mathbf{u}_m] \begin{bmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix} = \sum_{i=1}^r \sigma_i [\mathbf{u}_i \mathbf{v}_i^T] \tag{A.183}$$

by which the original matrix  $\mathbf{A}$  is represented as a linear combination of  $r$  matrices  $[\mathbf{u}_i \mathbf{v}_i^T]$  weighted by the singular values  $\sqrt{\lambda_i}$  ( $i = 1, \dots, r$ ). We can rewrite both the forward and inverse SVD transform as a pair of forward and inverse transforms:

$$\begin{cases} \mathbf{\Sigma} = \mathbf{\Lambda}^{1/2} = \mathbf{U}^T \mathbf{A} \mathbf{V} \\ \mathbf{A} = \mathbf{U} \mathbf{\Lambda}^{1/2} \mathbf{V}^T = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T. \end{cases} \tag{A.184}$$

This result indicate that the SVD can be used for data compression, if the 2-D data stored in an  $m \times n$  matrix  $\mathbf{A}$  can be approximated by keeping in the summation above only  $k \ll r \leq \min(m, n)$  terms corresponding to the  $k$  greatest singular values. The amount of data can be significantly reduced from  $m \times n$  to no more than  $k(m + n + 1)$ .

Also, the pseudo-inverse of any matrix (non-square, not full-rank) can be found given its SVD

$$\mathbf{A} = \mathbf{U} \mathbf{\Lambda}^{1/2} \mathbf{V}^T = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \tag{A.185}$$

as

$$\mathbf{A}^- = (\mathbf{V}^T)^{-1} \mathbf{\Sigma}^- \mathbf{U}^{-1} = \mathbf{V} \mathbf{\Sigma}^- \mathbf{U}^T \tag{A.186}$$

where  $\mathbf{\Sigma}^-$  is pseudo-inverse of  $\mathbf{\Sigma}$ , composed of the reciprocals  $1/\sigma_k$  of the  $R$  singular values along the diagonal. The pseudo-inverse matrix is needed to find optimal solution of an inconsistent linear equation system.

## A.15 Pseudo-Inverse

The inverse of an  $m \times n$  matrix  $\mathbf{A}$  does not exist if it is not square  $m \neq n$ . But we can still find its *pseudo-inverse*, an  $n \times m$  matrix denoted by  $\mathbf{A}^-$ , if  $\text{rank}(\mathbf{A}) = r = \min(m, n)$ , in either of the following ways:

- If  $m > n = r$ ,  $\mathbf{A}^T \mathbf{A}$  is an  $n \times n$  full-rank invertible matrix, and we define the *left inverse*:

$$\mathbf{A}^- = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \quad (\text{A.187})$$

which satisfies

$$\mathbf{A}^- \mathbf{A} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{A} = \mathbf{I}_{n \times n} \quad (\text{A.188})$$

But

$$\mathbf{A} \mathbf{A}^- = \mathbf{A} (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \neq \mathbf{I} \quad (\text{A.189})$$

- If  $n > m = r$ ,  $\mathbf{A} \mathbf{A}^T$  is an  $m \times m$  full-rank invertible matrix, and we define the *right inverse*:

$$\mathbf{A}^- = \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} \quad (\text{A.190})$$

which satisfies

$$\mathbf{A} \mathbf{A}^- = \mathbf{A} \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} = \mathbf{I}_{m \times m} \quad (\text{A.191})$$

But

$$\mathbf{A}^- \mathbf{A} = \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} \mathbf{A} \neq \mathbf{I} \quad (\text{A.192})$$

The left and right inverses defined respectively for  $m > n$  and  $m < n$  are the same. If  $m > n$ , then we have the left inverse  $\mathbf{A}^- = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$ . We then define  $\mathbf{B} = \mathbf{A}^T$ , and take the transpose on  $\mathbf{A}^-$  to get

$$(\mathbf{A}^-)^T = [(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T]^T = \mathbf{A} (\mathbf{A}^T \mathbf{A})^{-1} = (\mathbf{A}^T)^-; \quad (\text{A.193})$$

i.e.,  $\mathbf{B}^- = \mathbf{B}^T (\mathbf{B} \mathbf{B}^T)^{-1}$ , which is the second definition of the pseudo inverse of  $\mathbf{B}$ .

We can show that  $(\mathbf{A}^-)^- = \mathbf{A}$ . If  $m > n$ , then we have

$$\begin{aligned} (\mathbf{A}^-)^- &= [(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T]^- = [(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T]^T [(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T [(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T]^T]^{-1} \\ &= \mathbf{A} (\mathbf{A}^T \mathbf{A})^{-1} [(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{A} (\mathbf{A}^T \mathbf{A})^{-1}]^{-1} \\ &= \mathbf{A} (\mathbf{A}^T \mathbf{A})^{-1} (\mathbf{A}^T \mathbf{A}) = \mathbf{A}. \end{aligned}$$

Similarly we can show this is the case if  $m < n$ .

In particular, when  $m = n$ ,  $\mathbf{A}$  is a square invertible matrix, and both the left and right pseudo-inverse defined above become the regular inverse

$$\begin{aligned} \mathbf{A}^- &= (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T = \mathbf{A}^{-1} (\mathbf{A}^T)^{-1} \mathbf{A}^T = \mathbf{A}^{-1} \\ \mathbf{A}^- &= \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} = \mathbf{A}^T (\mathbf{A}^T)^{-1} \mathbf{A}^{-1} = \mathbf{A}^{-1} \end{aligned}$$



If  $r < \min(m, n)$ , then  $\mathbf{A}^{-1}$  does not exist, and  $\mathbf{A}^{-}$  is neither left nor right inverse, because  $\mathbf{A}^{-}\mathbf{A} \neq \mathbf{I}$  and  $\mathbf{A}\mathbf{A}^{-} \neq \mathbf{I}$ , as they are not full rank.

The pseudo inverse of  $\mathbf{A}$  can also be obtained based on its SVD  $\mathbf{A}_{m \times n} = \mathbf{U}_{m \times m} \boldsymbol{\Sigma}_{m \times n} \mathbf{V}_{n \times n}^T$ . Consider the following two cases:

- If  $m > n$ , the left pseudo inverse is

$$\begin{aligned} \mathbf{A}^{-} &= (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T = [(\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T)^T (\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T)]^{-1} (\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T)^T \\ &= (\mathbf{V} \boldsymbol{\Sigma}^T \mathbf{U}^T \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T)^{-1} (\mathbf{V} \boldsymbol{\Sigma}^T \mathbf{U}^T) = \mathbf{V} (\boldsymbol{\Sigma}^T \boldsymbol{\Sigma})^{-1} \mathbf{V}^T \mathbf{V} \boldsymbol{\Sigma}^T \mathbf{U}^T \\ &= \mathbf{V} [(\boldsymbol{\Sigma}^T \boldsymbol{\Sigma})^{-1} \boldsymbol{\Sigma}^T] \mathbf{U}^T = \mathbf{V} \boldsymbol{\Sigma}^{-} \mathbf{U}^T \end{aligned}$$

- If  $m < n$ , the right pseudo inverse is

$$\begin{aligned} \mathbf{A}^{-} &= \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} = (\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T)^T [(\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T) (\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T)^T]^{-1} \\ &= (\mathbf{V} \boldsymbol{\Sigma}^T \mathbf{U}^T) (\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T \mathbf{V} \boldsymbol{\Sigma}^T \mathbf{U}^T)^{-1} = \mathbf{V} \boldsymbol{\Sigma}^T \mathbf{U}^T (\mathbf{U} \boldsymbol{\Sigma} \boldsymbol{\Sigma}^T \mathbf{U}^T)^{-1} \\ &= \mathbf{V} \boldsymbol{\Sigma}^T \mathbf{U}^T \mathbf{U} (\boldsymbol{\Sigma} \boldsymbol{\Sigma}^T)^{-1} \mathbf{U}^T = \mathbf{V} [\boldsymbol{\Sigma}^T (\boldsymbol{\Sigma} \boldsymbol{\Sigma}^T)^{-1}] \mathbf{U}^T = \mathbf{V} \boldsymbol{\Sigma}^{-} \mathbf{U}^T \end{aligned}$$

In either case, we have

$$\mathbf{A}_{n \times m}^{-} = (\mathbf{U}_{m \times m} \boldsymbol{\Sigma}_{m \times m} \mathbf{V}_{n \times n}^T)^{-} = \mathbf{V}_{n \times n} \boldsymbol{\Sigma}_{n \times m}^{-} \mathbf{U}_{m \times m}^T \tag{A.194}$$

where  $\boldsymbol{\Sigma}_{n \times m}^{-}$  is the pseudo inverse of  $\boldsymbol{\Sigma}_{m \times n}$  with  $1/\sigma_1, \dots, 1/\sigma_r$  on its diagonal.

We also have

$$\mathbf{A}^{-} \mathbf{A} = (\mathbf{V} \boldsymbol{\Sigma}^{-} \mathbf{U}^T) (\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T) = \mathbf{V} (\boldsymbol{\Sigma}^{-} \boldsymbol{\Sigma}) \mathbf{V}^T \tag{A.195}$$

if  $r = m$ , then  $\boldsymbol{\Sigma}^{-} \boldsymbol{\Sigma} = \mathbf{I}$  an  $n \times n$  identity matrix and we have  $\mathbf{A}^{-} \mathbf{A} = \mathbf{V} \mathbf{V}^T = \mathbf{I}$ . Otherwise if  $r < m$ ,  $\boldsymbol{\Sigma}^{-} \boldsymbol{\Sigma}$  has zero components on the diagonal, and  $\mathbf{A}^{-} \mathbf{A} \neq \mathbf{I}$ .

## A.16 Vector Norms

The norm of a vector  $\mathbf{x}$ , denoted by  $\|\mathbf{x}\|$ , can be intuitively interpreted as its “size”. For example, the norm of a real number  $x \in \mathbb{R}$  in the 1-D real space is its absolute value  $|x|$ , or its distance to the origin, and the norm of a complex number  $z = x + jy \in \mathbb{C}$  is its modulus  $|z| = \sqrt{x^2 + y^2}$ , its Euclidean distance to the origin. Here is the most general definition of a vector norm:

**Definition:** The norm of a vector  $\mathbf{x} \in V$  in vector space  $V$  is a real non-negative value representing the length or magnitude of the vector. Specifically, the norm of  $\mathbf{x} \in V$  satisfy the following three conditions:

- Positivity:

$$\|\mathbf{x}\| \geq 0, \quad \|\mathbf{x}\| = 0 \text{ iff } \mathbf{x} = \mathbf{0} \tag{A.196}$$

- Homogeneity:

$$\|a\mathbf{x}\| = |a| \|\mathbf{x}\|, \quad a \in \mathbb{R} \text{ or } \mathbb{C} \tag{A.197}$$

- Triangle inequality:

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \quad (\text{A.198})$$

The triangle inequality can also be expressed in alternative forms. If we define  $\mathbf{z} = -\mathbf{y}$ , the triangle inequality becomes

$$\|\mathbf{x} - \mathbf{z}\| \leq \|\mathbf{x}\| + \|\mathbf{z}\| \quad (\text{A.199})$$

If we further define  $\mathbf{u} = \mathbf{x} - \mathbf{z}$ , i.e.,  $\mathbf{z} = \mathbf{x} - \mathbf{u}$ , then the above becomes

$$\|\mathbf{u}\| \leq \|\mathbf{x}\| + \|\mathbf{x} - \mathbf{u}\|, \quad \text{i.e.} \quad \|\mathbf{u}\| - \|\mathbf{x}\| \leq \|\mathbf{x} - \mathbf{u}\| = \|\mathbf{u} - \mathbf{x}\| \quad (\text{A.200})$$

Combining the two results above, we get

$$\|\mathbf{x}\| - \|\mathbf{y}\| \leq \|\mathbf{x} - \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \quad (\text{A.201})$$

The first equality holds if  $\mathbf{x}$  and  $\mathbf{y}$  are in the same direction, the second equality holds if they are in opposite direction. More specially when  $\mathbf{y} = \mathbf{0}$ , both equalities hold.

The  $p$ -norms of an  $n$ -D vector  $\mathbf{x} = [x_1, \dots, x_n]^T$  and a function  $f(x)$  are defined as:

$$\|\mathbf{x}\|_p = \left( \sum_{i=1}^n |x_i|^p \right)^{1/p}, \quad \|f(x)\|_p = \left( \int |f(x)|^p dx \right)^{1/p} \quad (\text{A.202})$$

The  $p$ -norm satisfies the three requirements in the definition of vector norm. The first two are trivially obvious, while the third one happens to be Minkowski's inequality (see appendix):

$$\|\mathbf{x} + \mathbf{y}\|_p \leq \|\mathbf{x}\|_p + \|\mathbf{y}\|_p \quad (\text{A.203})$$

The  $p$ -norms corresponding to  $p = 1$ ,  $p = 2$ , and  $p = \infty$  are most commonly used:

- $p = 1$ , the absolute sum of all  $n$  elements:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i| \quad (\text{A.204})$$

- $p = 2$ , the Euclidean norm:

$$\|\mathbf{x}\|_2 = (\mathbf{x}^T \mathbf{x})^{1/2} = \sqrt{\sum_{i=1}^n |x_i|^2} \quad (\text{A.205})$$

- $p = \infty$ , the maximum absolute value of all  $n$  elements:

$$\|\mathbf{x}\|_\infty = \left( \sum_{i=1}^n |x_i|^\infty \right)^{1/\infty} = \max\{|x_1|, \dots, |x_n|\} \quad (\text{A.206})$$

Out of the three p-norms, the Euclidean 2-norm is the only one that is *unitary invariant*, i.e., it is conserved or invariant under any unitary transform  $\mathbf{y} = \mathbf{R}\mathbf{x}$ , where  $\mathbf{R}$  is a unitary matrix satisfying  $\mathbf{R}^{-1} = \mathbf{R}^*$ :

$$\|\mathbf{y}\|_2^2 = \mathbf{y}^* \mathbf{y} = (\mathbf{R}\mathbf{x})^* (\mathbf{R}\mathbf{x}) = \mathbf{x}^* (\mathbf{R}^* \mathbf{R}) \mathbf{x} = \mathbf{x}^* \mathbf{x} = \|\mathbf{x}\|_2^2 \quad (\text{A.207})$$

i.e., the length of the vector is not changed by any unitary transform (such as rotation, when  $\mathbf{R}$  is a rotation matrix).

**Definition:** Two norms  $\|\mathbf{x}\|_a$  and  $\|\mathbf{x}\|_b$  are equivalent if there exist two positive real constants  $c$  and  $C$  so that

$$c \|\mathbf{x}\|_b \leq \|\mathbf{x}\|_a \leq C \|\mathbf{x}\|_b \quad (\text{A.208})$$

Note that this relationship can also be written as

$$\frac{1}{C} \|\mathbf{x}\|_a \leq \|\mathbf{x}\|_b \leq \frac{1}{c} \|\mathbf{x}\|_a \quad (\text{A.209})$$

**Theorem:** All different norms  $\mathbf{x}$  are equivalent.

**Proof**

- We first show that equivalence is transitive, i.e., if both  $\|\mathbf{x}\|_a$  and  $\|\mathbf{x}\|_b$  are equivalent to  $\|\mathbf{x}\|_1$ :

$$c \|\mathbf{x}\|_1 \leq \|\mathbf{x}\|_a \leq C \|\mathbf{x}\|_1, \quad \text{and} \quad d \|\mathbf{x}\|_1 \leq \|\mathbf{x}\|_b \leq D \|\mathbf{x}\|_1, \quad (\text{A.210})$$

then they are equivalent to each other.

From the first equation we get  $\|\mathbf{x}\|_1 \leq \|\mathbf{x}\|_a/c$  and  $\|\mathbf{x}\|_1 \geq \|\mathbf{x}\|_a/C$ . Substituting these into the right and left hand sides of the second equation respectively yields:

$$\frac{d}{C} \|\mathbf{x}\|_a \leq \|\mathbf{x}\|_b \leq \frac{D}{c} \|\mathbf{x}\|_a, \quad (\text{A.211})$$

i.e.,  $\|\mathbf{x}\|_a$  and  $\|\mathbf{x}\|_b$  are equivalent.

- If we can further show that an arbitrary norm  $\|\mathbf{x}\|$  is equivalent to  $\|\mathbf{x}\|_1$ , i.e.,

$$c \|\mathbf{x}\|_1 \leq \|\mathbf{x}\| \leq C \|\mathbf{x}\|_1 \quad (\text{A.212})$$

then according to the transitivity property shown above, all norms are equivalent.

If  $\mathbf{x} = \mathbf{0}$ , it is obvious that the equalities above hold  $\|\mathbf{x}\| = \|\mathbf{x}\|_1 = 0$ . If  $\mathbf{x} \neq \mathbf{0}$ ,  $\|\mathbf{x}\|_1 > 0$ , we can define a normalized vector  $\mathbf{u} = \mathbf{x}/\|\mathbf{x}\|_1$  with  $\|\mathbf{u}\|_1 = 1$ , so that the relationship for the equivalence above can be written in terms of  $\mathbf{u}$  as

$$c = c \|\mathbf{u}\|_1 \leq \|\mathbf{u}\| \leq C \|\mathbf{u}\|_1 = C \quad (\text{A.213})$$

i.e., to prove that all norms are equivalent all we need to show is that  $\|\mathbf{u}\|$  is bounded both from above and below, as we will do below.

- We now prove that an arbitrary norm  $f(\mathbf{x}) = \|\mathbf{x}\|$  is a continuous function over  $\|\mathbf{x}\|_1$ , i.e., for any  $\epsilon > 0$ , there exists a  $\delta > 0$  so that

$$\text{if } \|\mathbf{x} - \mathbf{x}'\|_1 < \delta, \quad \text{then } \|\mathbf{x}\| - \|\mathbf{x}'\| \leq \epsilon \quad (\text{A.214})$$

Both  $\mathbf{x}$  and  $\mathbf{x}'$  can be expressed in terms of a basis  $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$  that spans the space:

$$\mathbf{x} = \sum_{i=1}^n a_i \mathbf{b}_i, \quad \mathbf{x}' = \sum_{i=1}^n a'_i \mathbf{b}_i \quad (\text{A.215})$$

then we have

$$\|\mathbf{x} - \mathbf{x}'\| = \left\| \sum_{i=1}^n (a_i - a'_i) \mathbf{b}_i \right\| \leq \sum_{i=1}^n |a_i - a'_i| \|\mathbf{b}_i\| \leq \sum_{i=1}^n |a_i - a'_i| \max_i \|\mathbf{b}_i\| = \|\mathbf{x} - \mathbf{x}'\|_1 \max_i \|\mathbf{b}_i\| \quad (\text{A.216})$$

Now consider the alternative form of the triangle inequality of  $\|\mathbf{x}\|$ :

$$\|\mathbf{x}\| - \|\mathbf{x}'\| \leq \|\mathbf{x} - \mathbf{x}'\| \leq \|\mathbf{x} - \mathbf{x}'\|_1 \max_i \|\mathbf{b}_i\| \quad (\text{A.217})$$

If we let  $\max_i \|\mathbf{b}_i\| = \epsilon/\delta$ , the above becomes

$$\|\mathbf{x}\| - \|\mathbf{x}'\| \leq \|\mathbf{x} - \mathbf{x}'\|_1 \frac{\epsilon}{\delta} \quad (\text{A.218})$$

indicating that for any given  $\epsilon$ , we can choose  $\delta = \epsilon / \max_i \|\mathbf{b}_i\|$  so that if  $\|\mathbf{x} - \mathbf{x}'\|_1 \leq \delta$ , then  $\|\mathbf{x}\| - \|\mathbf{x}'\| \leq \epsilon$ , i.e., the norm  $\|\mathbf{x}\|$  is indeed continuous over  $\|\mathbf{x}\|_1$ .

- Finally, according to the *extreme value theorem*, a continuous function, such as  $f(\mathbf{x}) = \|\mathbf{x}\|$ , defined over a compact (closed and bounded) set, such as the unit sphere  $\|\mathbf{u}\|_1 = 1$  in the n-D space, must have its maximum and minimum values:

$$c = \min_{\|\mathbf{u}\|_1=1} \|\mathbf{u}\|, \quad C = \max_{\|\mathbf{u}\|_1=1} \|\mathbf{u}\| \quad (\text{A.219})$$

i.e.,  $c \leq \|\mathbf{u}\| \leq C$ , which is what we need to prove.

Q.E.D.

The three p-norms are equivalent:

$$\begin{aligned} \|\mathbf{x}\|_\infty &\leq \|\mathbf{x}\|_1 \leq n \|\mathbf{x}\|_\infty \\ \|\mathbf{x}\|_\infty &\leq \|\mathbf{x}\|_2 \leq \sqrt{n} \|\mathbf{x}\|_\infty \\ \|\mathbf{x}\|_2 &\leq \|\mathbf{x}\|_1 \leq \sqrt{n} \|\mathbf{x}\|_2 \end{aligned}$$

The *distance* between two vectors  $\mathbf{x}, \mathbf{y} \in V$  is defined as the norm of their difference  $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ . In particular, when  $\mathbf{y} = \mathbf{0}$ , the distance  $\|\mathbf{x} - \mathbf{0}\| = \|\mathbf{x}\|$  between  $\mathbf{x}$  and the origin  $\mathbf{0}$  of the space is the norm of  $\mathbf{x}$ . Specifically,  $d(\mathbf{x}, \mathbf{y})$  can be defined based on the p-norm:

$$d_p(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}, \quad (1 \leq p \leq \infty) \quad (\text{A.220})$$

Specially, when

- $p = 1$ ,  $d_1(\mathbf{x}, \mathbf{y})$  is the city block (Manhattan) distance:

$$d_1(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_1 = \sum_{i=1}^n |x_i - y_i| \quad (\text{A.221})$$

- $p = 2$ ,  $d_2(\mathbf{x}, \mathbf{y})$  is the Euclidean distance:

$$d_2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_{i=1}^n |x_i - y_i|^2} \quad (\text{A.222})$$

- $p = \infty$ ,  $d_\infty(\mathbf{x}, \mathbf{y})$  is the Chebyshev distance:

$$d_\infty(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_\infty = \max_i |x_i - y_i| \quad (\text{A.223})$$

The three unit “circles” or “spheres”, are formed by all points  $\mathbf{x}$  of unity norm  $\|\mathbf{x}\|_p = 1$  with unity distance to the origin (blue, black, and red for  $d_\infty$ ,  $d_2$ , and  $d_1$ , respectively).

## A.17 Matrix Norms

The norm of an  $m \times n$  matrix  $\mathbf{A}$  is any function that maps  $\mathbf{A}$  to a real number  $\|\mathbf{A}\|$  that satisfies the following required properties:

- Positivity:

$$\|\mathbf{A}\| \geq 0, \quad \|\mathbf{A}\| = 0 \text{ iff } \mathbf{A} = \mathbf{0} \quad (\text{A.224})$$

- Homogeneity:

$$\|a\mathbf{A}\| = |a| \|\mathbf{A}\| \quad (\text{A.225})$$

- Triangle inequality:

$$\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|, \quad \text{or} \quad \|\mathbf{A}\| - \|\mathbf{B}\| \leq \|\mathbf{A} - \mathbf{B}\| \quad (\text{A.226})$$

The second expression can be obtained by replacing  $\mathbf{A}$  by  $\mathbf{A} - \mathbf{B}$  in the first expression.

In addition to the three required properties for matrix norms, some of them also satisfy these additional properties (not necessarily satisfied by other matrix norms):

- Subordinance:

$$\|\mathbf{Ax}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{x}\| \quad (\text{A.227})$$

- submultiplicativity:

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{B}\| \quad (\text{A.228})$$

We now consider some commonly used matrix norms:

- **The element-wise norms**

If we treat the  $m \times n$  elements of  $\mathbf{A}$  as the elements of an  $mn$ -dimensional vector, then the  $p$ -norm of this vector can be used as the  $p$ -norm of  $\mathbf{A}$ :

$$\|\mathbf{A}\|_p = \left\{ \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^p \right\}^{1/p} \quad (\text{A.229})$$

Specially, we consider the following three cases for  $p = 1, 2, \infty$ .

–  $\|\mathbf{A}\|_1$  is the absolute sum of all elements of  $\mathbf{A}$ :

$$\|\mathbf{A}\|_1 = \sum_{i=1}^m \sum_{j=1}^n |a_{ij}| \quad (\text{A.230})$$

–  $\|\mathbf{A}\|_\infty$  is the *maximum norm*, the maximum absolute value among all  $mn$  elements of  $\mathbf{A}$ :

$$\|\mathbf{A}\|_\infty = \max\{|a_{ij}|, 1 \leq i \leq m, 1 \leq j \leq n\} \quad (\text{A.231})$$

–  $\|\mathbf{A}\|_2$  is the *Frobenius norm*

$$\|\mathbf{A}\|_2 = \|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2} = \sqrt{\text{tr}(\mathbf{A}^* \mathbf{A})} = \sqrt{\sum_{i=1}^R \lambda_i} = \sqrt{\sum_{i=1}^R \sigma_i^2} \quad (\text{A.232})$$

where  $r \leq \min(m, n)$  is the rank of  $\mathbf{A}$ ,  $\lambda_i = \sigma_i^2$  is the  $i$ th non-zero eigenvalues of the positive, semi-definite matrix  $\mathbf{A}^* \mathbf{A}$ , and  $\sigma_i = \sqrt{\lambda_i}$  the  $i$ th singular value of  $\mathbf{A}$  ( $i = 1, \dots, r$ ). (This Frobenius norm  $\|\mathbf{A}\|_2$  is implemented in Matlab by the function `norm(A, 'fro')`.)

The Frobenius norm of a unitary (orthogonal if real) matrix  $\mathbf{R}$  satisfying  $\mathbf{R}^* = \mathbf{R}^{-1}$  or  $\mathbf{R}^* \mathbf{R} = \mathbf{R}^{-1} \mathbf{R} = \mathbf{I}$  is:

$$\|\mathbf{R}\|_F^2 = \text{tr}(\mathbf{R}^* \mathbf{R}) = \text{tr} \mathbf{I} = n \quad (\text{A.233})$$

The Frobenius norm is the only one out of the above three matrix norms that is *unitary invariant*, i.e., it is conserved or invariant under a unitary transformation (such as a rotation)  $\mathbf{B} = \mathbf{R}^* \mathbf{A} \mathbf{R}$ :

$$\begin{aligned} \|\mathbf{B}\|_F^2 &= \text{tr}(\mathbf{B}^* \mathbf{B}) = \text{tr}[(\mathbf{R}^* \mathbf{A} \mathbf{R})^* (\mathbf{R}^* \mathbf{A} \mathbf{R})] = \text{tr}(\mathbf{R}^* \mathbf{A}^* \mathbf{R} \mathbf{R}^* \mathbf{A} \mathbf{R}) \\ &= \text{tr}(\mathbf{R}^* \mathbf{A}^* \mathbf{A} \mathbf{R}) = \text{tr}(\mathbf{A}^* \mathbf{A}) = \|\mathbf{A}\|_F^2 = \|\mathbf{A}\|_F^2 \end{aligned}$$

where we have used the property of the trace  $\text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA})$ .

- **The natural matrix norms**

The natural norm  $\|\mathbf{A}\|$  of matrix  $\mathbf{A}$  induced by (subordinate to) the vector norm  $\|\mathbf{x}\|$  is defined as

$$\|\mathbf{A}\| = \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{A}\mathbf{x}\|\} = \sup_{\mathbf{x} \neq \mathbf{0}} \{\|\mathbf{A}\mathbf{x}\|/\|\mathbf{x}\|\} \quad (\text{A.234})$$

Specially the norm of the identity matrix  $\mathbf{A} = \mathbf{I}$  is

$$\|\mathbf{I}\| = \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{I}\mathbf{x}\|\} = \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{x}\|\} = 1 \quad (\text{A.235})$$

We now prove that the induced matrix norm satisfies all five properties given above. (Recall  $|a + b| \leq |a| + |b|$ ,  $|ab| \leq |a| |b|$ .)

1. Positivity:  $\|\mathbf{A}\| > 0$  if  $\mathbf{A} \neq \mathbf{0}$ , this is trivially obvious.

2. Homogeneity:  $\|a\mathbf{A}\| = |a| \|\mathbf{A}\|$

$$\|a\mathbf{A}\| = \sup_{\|\mathbf{x}\|=1} \{\|a\mathbf{A}\mathbf{x}\|\} = \sup_{\|\mathbf{x}\|=1} \{|a| \|\mathbf{A}\mathbf{x}\|\} = |a| \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{A}\mathbf{x}\|\} = |a| \|\mathbf{A}\| \quad (\text{A.236})$$

3. Triangle inequality:  $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$

$$\begin{aligned} \|\mathbf{A} + \mathbf{B}\| &= \sup_{\|\mathbf{x}\|=1} \{\|(\mathbf{A} + \mathbf{B})\mathbf{x}\|\} = \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{x}\|\} \leq \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{A}\mathbf{x}\| + \|\mathbf{B}\mathbf{x}\|\} \\ &\leq \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{A}\mathbf{x}\|\} + \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{B}\mathbf{x}\|\} = \|\mathbf{A}\| + \|\mathbf{B}\| \end{aligned}$$

4. Subordinance:  $\|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\| \|\mathbf{x}\|$

$$\|\mathbf{A}\| \|\mathbf{x}\| = \|\mathbf{x}\| \sup_{\mathbf{y}} \{\|\mathbf{A}\mathbf{y}\|/\|\mathbf{y}\|\} \geq \|\mathbf{x}\| \|\mathbf{A}\mathbf{y}\|/\|\mathbf{y}\| \quad (\text{A.237})$$

As  $\mathbf{y}$  is arbitrary, we let  $\mathbf{y} = \mathbf{x}$  and get

$$\|\mathbf{A}\| \|\mathbf{x}\| \geq \|\mathbf{A}\mathbf{x}\| \quad (\text{A.238})$$

In particular, if vector  $\mathbf{x}$  is the eigenvector corresponding to the greatest eigenvalue  $\lambda_{max}$  of  $\mathbf{A}^* \mathbf{A}$ :

$$(\mathbf{A}^* \mathbf{A}) \mathbf{x} = \lambda_{max} \mathbf{x} = \|\mathbf{A}\|^2 \mathbf{x} \quad (\text{A.239})$$

then the equality of the subordinance property holds. To see this, consider

$$\|\mathbf{A}\mathbf{x}\|^2 = \langle (\mathbf{A}\mathbf{x})^*, (\mathbf{A}\mathbf{x}) \rangle = \mathbf{x}^* (\mathbf{A}^* \mathbf{A}) \mathbf{x} = \mathbf{x}^* \|\mathbf{A}\|^2 \mathbf{x} = \|\mathbf{A}\|^2 \|\mathbf{x}\|^2 \quad (\text{A.240})$$

taking square root on both sides we get  $\|\mathbf{A}\mathbf{x}\| = \|\mathbf{A}\| \|\mathbf{x}\|$ .

5. Submultiplicativity:  $\|\mathbf{A}\mathbf{B}\| \leq \|\mathbf{A}\| \|\mathbf{B}\|$

$$\|\mathbf{A}\mathbf{B}\| = \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{A}\mathbf{B}\mathbf{x}\|\} \leq \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{A}\| \|\mathbf{B}\mathbf{x}\|\} = \|\mathbf{A}\| \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{B}\mathbf{x}\|\} = \|\mathbf{A}\| \|\mathbf{B}\| \quad (\text{A.241})$$

In particular, if  $\mathbf{A} = c\mathbf{B}^*$  (i.e.,  $\mathbf{A}$  and  $\mathbf{B}$  are linearly dependent), then the equality of the submultiplicativity property holds. To see this, consider

$$\|\mathbf{B}\| = \sqrt{\lambda_{max}(\mathbf{B}^* \mathbf{B})}, \quad \|\mathbf{A}\| = |c| \sqrt{\lambda_{max}(\mathbf{B}\mathbf{B}^*)} = |c| \sqrt{\lambda_{max}(\mathbf{B}^* \mathbf{B})} \quad (\text{A.242})$$

and

$$\|\mathbf{A}\mathbf{B}\| = |c| \|\mathbf{B}^* \mathbf{B}\| = |c| \sqrt{\lambda_{max}((\mathbf{B}^* \mathbf{B})(\mathbf{B}^* \mathbf{B}))} = |c| \sqrt{\lambda_{max}(\mathbf{B}^* \mathbf{B})^2} = |c| \lambda_{max}(\mathbf{B}^* \mathbf{B}) \quad (\text{A.243})$$

i.e.,  $\|\mathbf{A}\mathbf{B}\| = \|\mathbf{A}\| \|\mathbf{B}\|$ .

Consider in particular the natural matrix norms  $\|\mathbf{A}\|_p$  corresponding to the three vector p-norms  $\|\mathbf{x}\|_p$  with  $p = 1, 2, \infty$ :

- When  $p = 1$ ,  $\|\mathbf{A}\|_1$  (`norm(A,1)` in Matlab) is the maximum absolute column sum:

$$\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}| \quad (\text{A.244})$$

**Proof:** The 1-norm of vector  $\mathbf{x}$  is  $\|\mathbf{x}\|_1 = \sum_{j=1}^n |x_j|$ , we have

$$\begin{aligned} \|\mathbf{Ax}\|_1 &= \sum_{i=1}^m \left| \sum_{j=1}^n a_{ij} x_j \right| \leq \sum_{i=1}^m \sum_{j=1}^n (|a_{ij}| |x_j|) = \sum_{j=1}^n \left( |x_j| \sum_{i=1}^m |a_{ij}| \right) \\ &\leq \left( \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}| \right) \left( \sum_{j=1}^n |x_j| \right) = \|\mathbf{A}\|_1 \|\mathbf{x}\|_1 \end{aligned}$$

Assuming the  $k$ th column of  $\mathbf{A}$  has the maximum absolute sum and  $\mathbf{x}$  is normalized (as required in the definition) with  $\|\mathbf{x}\|_1 = 1$ , we have

$$\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}| = \sum_{i=1}^m |a_{ik}| \quad (\text{A.245})$$

and

$$\|\mathbf{Ax}\|_1 \leq \|\mathbf{A}\|_1 \|\mathbf{x}\|_1 = \sum_{i=1}^m |a_{ik}| \quad (\text{A.246})$$

Now we show that the equality of the above can be achieved, i.e.,  $\|\mathbf{Ax}\|_1$  is maximized, if we choose  $\mathbf{x} = \mathbf{e}_k = [0, \dots, 1, \dots, 0]^T$ , the  $k$ th unit vector (normalized):

$$\|\mathbf{Ax}\|_1 = \sum_{i=1}^m \left| \sum_{j=1}^n a_{ij} x_j \right| = \sum_{i=1}^m |a_{ik}| = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}| \quad (\text{A.247})$$

i.e.,  $\mathbf{x} = \mathbf{e}_k$  is the vector among all other normalized vectors that maximizes  $\|\mathbf{Ax}\|_1$  as required in the definition, and the resulting maximum  $\|\mathbf{A}\|_1$  is indeed  $\max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|$ . We therefore have

$$\|\mathbf{A}\|_1 = \sup_{\|\mathbf{x}\|_1=1} \{\|\mathbf{Ax}\|_1\} = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}| \quad (\text{A.248})$$

- When  $p = \infty$ ,  $\|\mathbf{A}\|_\infty$  (`norm(A,inf)` in Matlab) is the maximum absolute row sum:

$$\|\mathbf{A}\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^n |a_{ij}| \quad (\text{A.249})$$

**Proof:** When  $p = \infty$ ,  $\mathbf{x}$  is normalized if  $\|\mathbf{x}\|_\infty = \max\{x_1, \dots, x_n\} =$



1. The norm of vector  $\mathbf{Ax}$  is:

$$\|\mathbf{Ax}\|_\infty = \max \left\{ \sum_{j=1}^n |a_{ij}x_j|, \dots, \sum_{j=1}^n |a_{mj}x_j| \right\} \quad (\text{A.250})$$

which can be maximized by any normalized vector with  $\|\mathbf{x}\|_\infty = \max\{x_1, \dots, x_n\} = 1$  to become

$$\|\mathbf{Ax}\|_\infty = \max \left\{ \sum_{j=1}^n |a_{ij}|, \dots, \sum_{j=1}^n |a_{mj}| \right\} \quad (\text{A.251})$$

We therefore have

$$\|\mathbf{A}\|_\infty = \sup_{\|\mathbf{x}\|_\infty=1} \{\|\mathbf{Ax}\|_\infty\} = \max_i \sum_{j=1}^n |a_{ij}| \quad (\text{A.252})$$

For example, for any stochastic matrix  $\mathbf{P}$  satisfying  $\sum_j P_{ij} = 1$ ,  $\|\mathbf{P}\|_\infty = 1$ .

- When  $p = 2$ ,  $\|\mathbf{A}\|_2$  (`norm(A,2)` or `norm(A)` in Matlab), also called the *spectral norm*, is the greatest singular value of  $\mathbf{A}$ , square root of the greatest eigenvalue of  $\mathbf{A}^* \mathbf{A}$ , i.e., its spectral radius  $\rho(\mathbf{A}^* \mathbf{A}) = \lambda_{max}(\mathbf{A}^* \mathbf{A})$ :

$$\|\mathbf{A}\|_2 = \sigma_{max}(\mathbf{A}) = \sqrt{\lambda_{max}(\mathbf{A}^* \mathbf{A})} = \sqrt{\rho(\mathbf{A}^* \mathbf{A})} \quad (\text{A.253})$$

where  $\lambda_{max}(\mathbf{A}^* \mathbf{A}) = \sigma_{max}^2(\mathbf{A})$  is the maximal eigenvalue of  $\mathbf{A}^* \mathbf{A}$ .

**Proof:**

First we consider the eigenequation of  $\mathbf{A}^* \mathbf{A}$ :

$$\mathbf{A}^* \mathbf{A} \mathbf{v} = \lambda_i \mathbf{v}_i, \quad (i = 1, \dots, n) \quad \text{or} \quad \mathbf{A}^* \mathbf{A} \mathbf{V} = \mathbf{V} \mathbf{\Lambda}, \quad \text{or} \quad \mathbf{A}^* \mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^* \quad (\text{A.254})$$

where

$$\mathbf{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_n), \quad \text{and} \quad \mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n] \quad (\text{A.255})$$

are the eigenvector and eigenvalue matrices of  $\mathbf{A}^* \mathbf{A}$ . As  $\mathbf{A}^* \mathbf{A}$  is symmetric and positive definite, all of its eigenvalues are real and positive, and assumed to be sorted

$$\lambda_1 = \lambda_{max} \geq \lambda_2 \geq \dots \geq \lambda_n = \lambda_{min} \geq 0 \quad (\text{A.256})$$

and all corresponding eigenvectors are orthogonal and assumed to be normalized, i.e.,  $\mathbf{v}_i^* \mathbf{v}_j = \delta_{ij}$ , or  $\mathbf{V}^* = \mathbf{V}^{-1}$  is a unitary (orthogonal if real) matrix.

Next, consider the  $p = 2$  norm of  $\mathbf{Ax}$ :

$$\begin{aligned} \|\mathbf{Ax}\|_2 &= [(\mathbf{Ax})^* (\mathbf{Ax})]^{1/2} = (\mathbf{x}^* \mathbf{A}^* \mathbf{A} \mathbf{x})^{1/2} = (\mathbf{x}^* \mathbf{V} \mathbf{\Lambda} \mathbf{V}^* \mathbf{x})^{1/2} \\ &= (\mathbf{y}^* \mathbf{\Lambda} \mathbf{y})^{1/2} = \sqrt{\sum_{i=1}^n y_i^2 \lambda_i} \end{aligned}$$

Here  $\mathbf{y} = \mathbf{V}^* \mathbf{x}$  is a unitary transform of  $\mathbf{x}$  with the 2-norm  $\|\mathbf{y}\|_2 = \|\mathbf{x}\|_2$  conserved.

The right-hand side of the equation above is a weighted average of the  $n$  eigenvalues  $\lambda_1, \dots, \lambda_n$ , which is maximized if they are weighted by a normalized vector  $\mathbf{y} = [1, 0, \dots, 0]^T$  with  $\|\mathbf{y}\|_2 = 1$ , by which the greatest eigenvalue  $\lambda_{max}$  is maximally weighted while all others are weighted by 0:

$$\|\mathbf{A}\mathbf{x}\|_2 \leq \sqrt{\lambda_{max}} \quad (\text{A.257})$$

As  $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = 1$ , we have

$$\|\mathbf{A}\|_2 = \sup_{\|\mathbf{x}\|_2=1} \{\|\mathbf{A}\mathbf{x}\|_2\} = \sqrt{\lambda_{max}} \quad (\text{A.258})$$

Out of the three matrix norms defined above, the spectral norm is the only one that is *unitary invariant*, i.e., it is conserved or invariant under any unitary transform  $\mathbf{B} = \mathbf{R}^* \mathbf{A} \mathbf{R}$ :

$$\begin{aligned} \|\mathbf{B}\|_2^2 &= \lambda_{max}(\mathbf{B}^* \mathbf{B}) = \lambda_{max}[(\mathbf{R}^* \mathbf{A} \mathbf{R})^* (\mathbf{R}^* \mathbf{A} \mathbf{R})] = \lambda_{max}(\mathbf{R}^* \mathbf{A}^* \mathbf{R} \mathbf{R}^* \mathbf{A} \mathbf{R}) \\ &= \lambda_{max}(\mathbf{R}^* \mathbf{A}^* \mathbf{A} \mathbf{R}) = \lambda_{max}(\mathbf{A}^* \mathbf{A}) = \|\mathbf{A}\|_2^2 \end{aligned}$$

Here we have used the fact that the eigenvalues and eigenvectors are invariant under the unitary transform.

#### Example

$$\mathbf{A} = \begin{bmatrix} 3 & -6 & 2 \\ 2 & 5 & 1 \\ -3 & 2 & 2 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 3 & 2 & 1 \\ 2 & -3 & 0 \\ 1 & 0 & -1 \end{bmatrix} \quad (\text{A.259})$$

$$\|\mathbf{A}\|_1 = \max\{|3| + |2| + |-3|, |-6| + |5| + |2|, |2| + |1| + |2|\} = \max\{8, 13, 5\} = 13$$

$$\|\mathbf{A}\|_\infty = \max\{|3| + |-6| + |2|, |2| + |5| + |1|, |-3| + |2| + |2|\} = \max\{11, 8, 7\} = 11$$

The eigenvalues of  $\mathbf{A}^* \mathbf{A}$  are

$$\lambda_1 = 69.353, \quad \lambda_2 = 17.967, \quad \lambda_3 = 8.680 \quad (\text{A.260})$$

The singular values of  $\mathbf{A}$  are

$$\sigma_1 = \sqrt{\lambda_1} = 8.328, \quad \sigma_2 = \sqrt{\lambda_2} = 4.239, \quad \sigma_3 = \sqrt{\lambda_3} = 2.946 \quad (\text{A.261})$$

The spectral norm of  $\mathbf{A}$  is

$$\|\mathbf{A}\|_2 = \sigma_{max} = \sqrt{\lambda_{max}} = 8.328 \quad (\text{A.262})$$

The eigenvector corresponding to greatest eigenvalue  $\lambda_1$  is  $\mathbf{x} = [0.285, -0.957, 0.057]^T$ , which satisfies the equality  $\|\mathbf{A}\mathbf{x}\|_2 = \|\mathbf{A}\|_2 \|\mathbf{x}\|_2$ .

- **The Schatten norms**

The Schatten norm is defined based on the singular values  $\sigma_i$  of  $\mathbf{A}$  or the eigenvalues  $\lambda_i = \sigma_i^2$  of  $\mathbf{A}^* \mathbf{A}$ :

$$\|\mathbf{A}\|_p = \left( \sum_{i=1}^R \sigma_i^p \right)^{1/p} = \left( \sum_{i=1}^R (\sqrt{\lambda_i})^p \right)^{1/p} \quad (\text{A.263})$$

In particular, consider three common  $p$  values:

–  $p = 1$  is the *nuclear or trace* norm:

$$\|\mathbf{A}\|_1 = \sum_{i=1}^R \sigma_i = \sum_{i=1}^R \sqrt{\lambda_i} = \text{tr} \left( \sqrt{\mathbf{A}^* \mathbf{A}} \right) \quad (\text{A.264})$$

–  $p = 2$  same as the Frobenius norm:

$$\|\mathbf{A}\|_2 = \sqrt{\sum_{i=1}^R \sigma_i^2} = \sqrt{\sum_{i=1}^R \lambda_i} = \|\mathbf{A}\|_F \quad (\text{A.265})$$

–  $p = \infty$  same as the spectral norm (the induced 2-norm), the spectral radius of  $\mathbf{A}^* \mathbf{A}$ .

$$\|\mathbf{A}\|_\infty = \max\{\sigma_1, \dots, \sigma_n\} = \sigma_{max} = \sqrt{\lambda_{max}} \quad (\text{A.266})$$

As the eigenvalues and eigenvectors of  $\mathbf{A}^* \mathbf{A}$  are invariant under unitary transform, the Schatten norms are unitary invariant as well.

All matrix norms defined above are equivalent according to the theorem previously discussed.

- The Frobenius norm  $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^R \lambda_i}$  and the induced 2-norm  $\|\mathbf{A}\|_2 = \sqrt{\lambda_{max}}$  are equivalent:

$$\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{R} \|\mathbf{A}\|_2 \quad (\text{A.267})$$

The equality on the left holds when all eigenvalues  $\lambda_i$  but one are zero, and the equality on the right holds when all  $\lambda_i$  are the same.

- The Frobenius norm  $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^R \lambda_i}$  and the Schatten 1-norm  $\|\mathbf{A}\|_1 = \sum_{i=1}^R \sqrt{\lambda_i}$  are equivalent:

$$\|\mathbf{A}\|_F \leq \|\mathbf{A}\|_1 \leq \sqrt{R} \|\mathbf{A}\|_F \quad (\text{A.268})$$

The equality on the left holds when all eigenvalues  $\lambda_i$  but one are zero, and the equality on the right holds when all  $\lambda_i$  are the same.

- The element-wise maximum norm  $\|\mathbf{A}\|_{max} = \max_{i,j} \{|a_{ij}|\}$  and the Frobenius norm  $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2}$  are equivalent:

$$\|\mathbf{A}\|_{max} \leq \|\mathbf{A}\|_F \leq \sqrt{mn} \|\mathbf{A}\|_{max} \quad (\text{A.269})$$

The equality on the left holds when all elements  $\lambda_i$  but one are zero, and the equality on the right holds when all elements are the same.

•

$$\frac{1}{\sqrt{n}}\|\mathbf{A}\|_{\infty} \leq \|\mathbf{A}\|_2 \leq \sqrt{m}\|\mathbf{A}\|_{\infty} \quad (\text{A.270})$$

**Proof:** Define an  $n$ -D vector  $\mathbf{e} = [1, \dots, 1]^T$ , then the greatest absolute row sum of  $\mathbf{A}$  is

$$\|\mathbf{A}\|_{\infty} = \max_i \sum_{j=1}^n |a_{ij}| = \|\mathbf{A}\mathbf{e}\|_{\infty} \leq \|\mathbf{A}\mathbf{e}\|_2 \leq \|\mathbf{A}\|_2 \|\mathbf{e}\|_2 = \sqrt{n}\|\mathbf{A}\|_2 \quad (\text{A.271})$$

i.e.,

$$\frac{1}{\sqrt{n}}\|\mathbf{A}\|_{\infty} \leq \|\mathbf{A}\|_2 \quad (\text{A.272})$$

•

$$\frac{1}{\sqrt{m}}\|\mathbf{A}\|_1 \leq \|\mathbf{A}\|_2 \leq \sqrt{n}\|\mathbf{A}\|_1 \quad (\text{A.273})$$

**Theorem:** The spectral radius of a matrix is bounded by its matrix norm:  
 $\rho(\mathbf{A}) \leq \|\mathbf{A}\|$

**Proof:** Let  $\lambda$  and  $\mathbf{u}$  be an eigenvalue and the corresponding normalized eigenvector of a square matrix  $\mathbf{A}$ , i.e.,  $\mathbf{A}\mathbf{u} = \lambda\mathbf{u}$  and  $\|\mathbf{u}\| = 1$ . As the induced norm,  $\|\mathbf{A}\|$  satisfies

$$\|\mathbf{A}\| = \sup_{\|\mathbf{x}\|=1} \{\|\mathbf{A}\mathbf{x}\|\} \geq \|\mathbf{A}\mathbf{u}\| = \|\lambda\mathbf{u}\| = |\lambda| \|\mathbf{u}\| = |\lambda| \quad (\text{A.274})$$

But  $\lambda$  is an arbitrary eigenvalue, it can be the one with the maximum absolute value  $|\lambda|$ , i.e., the spectral radius  $\rho(\mathbf{A})$ . We therefore have  $\|\mathbf{A}\| \geq \rho(\mathbf{A})$ .

**Theorem:**

$$\rho(\mathbf{A}) \leq \|\mathbf{A}^k\|^{1/k} \quad (\text{A.275})$$

**Proof:** Let  $\lambda$  and  $\mathbf{v}$  by the eigenvalue and the corresponding eigenvector of  $\mathbf{A}$  respectively, i.e.,

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}, \quad \text{and} \quad \mathbf{A}^k\mathbf{v} = \lambda^k\mathbf{v} \quad (\text{A.276})$$

Taking norm on both sides we get

$$\|\lambda^k\mathbf{v}\| = |\lambda|^k\|\mathbf{v}\| = \|\mathbf{A}^k\mathbf{v}\| \leq \|\mathbf{A}^k\|\|\mathbf{v}\| \quad (\text{A.277})$$

Dividing both sides by  $\|\mathbf{v}\| \neq 0$  we get

$$\lambda^k \leq \|\mathbf{A}^k\|, \quad \text{i.e.} \quad \lambda \leq \|\mathbf{A}^k\|^{1/k} \quad (\text{A.278})$$

**Theorem:** A square matrix  $\mathbf{A}$  is convergent, i.e.,  $\lim_{n \rightarrow \infty} \mathbf{A}^n = 0$ , if and only if  $\rho(\mathbf{A}) < 1$ .

The proof of this theorem can be found at [http://en.wikipedia.org/wiki/Spectral\\_radius](http://en.wikipedia.org/wiki/Spectral_radius).

## A.18 Vector and Matrix Differentiation

A multivariate function  $f(\mathbf{x}) = f(x_1, \dots, x_n)$  can be considered as a hypersurface in an  $n + 1$  dimensional space defined over an  $n$ -D space, i.e., corresponding to every point  $\mathbf{x} = [x_1, \dots, x_n]^T$  in the  $n$ -D space, there exists a unique value  $f(\mathbf{x})$  (real or complex).

A *vector differentiation operator* is defined as

$$\frac{d}{d\mathbf{x}} \triangleq \left[ \frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n} \right]^T \quad (\text{A.279})$$

which, when applied to any scalar function  $f(\mathbf{x})$ , produces its derivative with respect to  $\mathbf{x}$ , the *gradient* vector of  $f(\mathbf{x})$ :

$$\mathbf{g}_f(\mathbf{x}) = \nabla f(\mathbf{x}) = \frac{d}{d\mathbf{x}} f(\mathbf{x}) = \left[ \frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right]^T = [f_{x_1}, \dots, f_{x_n}]^T \quad (\text{A.280})$$

The *Directional derivative* of a function  $f(\mathbf{x})$  is defined as the rate of change of the function value along a unit direction  $\mathbf{r} = [r_1, \dots, r_n]^T$  (with  $\|\mathbf{r}\| = 1$ ):

$$D_r f(\mathbf{x}) = \nabla_r f(\mathbf{x}) = \frac{d}{dh} f(\mathbf{x} + h\mathbf{r}) = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{r}) - f(\mathbf{x})}{h} \quad (\text{A.281})$$

Introducing  $n$  intermediate functions  $x_i + hr_i$  ( $i = 1, \dots, n$ ) of  $h$ , we can rewrite the above as the following by chain rule:

$$D_r f(\mathbf{x}) = \frac{d}{dh} f(\mathbf{x} + h\mathbf{r}) \Big|_{h=0} = \sum_{i=1}^n \frac{\partial f(\mathbf{x} + h\mathbf{r})}{\partial (x_i + hr_i)} \frac{d(x_i + hr_i)}{dh} \Big|_{h=0} = \sum_{i=1}^n f_{x_i} r_i = \mathbf{g}_f^T \mathbf{r} \quad (\text{A.282})$$

We see that the directional derivative is the projection of the gradient vector onto the unit direction  $\mathbf{r}$ , which is maximized when the gradient vector  $\mathbf{g}_f$  is in the same direction as  $\mathbf{r}$ . In other words, the function value increases most rapidly along the direction of its gradient.

Vector differentiation has the following properties:

$$\frac{d}{d\mathbf{x}} (\mathbf{a}^T \mathbf{x}) = \frac{d}{d\mathbf{x}} (\mathbf{x}^T \mathbf{a}) = \mathbf{a}, \quad \frac{d}{d\mathbf{x}} (\mathbf{x}^T \mathbf{x}) = 2\mathbf{x}, \quad \frac{d}{d\mathbf{x}} (\mathbf{x}^T \mathbf{A} \mathbf{x}) = (\mathbf{A}^T + \mathbf{A}) \mathbf{x} \quad (\text{A.283})$$

To prove the third one, consider the  $k$ th element of the vector:

$$\frac{\partial}{\partial x_k} (\mathbf{x}^T \mathbf{A} \mathbf{x}) = \frac{\partial}{\partial x_k} \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j = \sum_{i=1}^n a_{ik} x_i + \sum_{j=1}^n a_{kj} x_j \quad (k = 1, \dots, n) \quad (\text{A.284})$$

Putting all  $n$  elements in vector form, we have the above. If  $\mathbf{A}^T = \mathbf{A}$  is symmetric, then we have

$$\frac{d}{d\mathbf{x}} (\mathbf{x}^T \mathbf{A} \mathbf{x}) = 2\mathbf{A} \mathbf{x} \quad (\text{A.285})$$

In particular, when  $\mathbf{A} = \mathbf{I}$ , we have

$$\frac{d}{d\mathbf{x}}(\mathbf{x}^T \mathbf{x}) = 2\mathbf{x} \quad (\text{A.286})$$

One can compare these results with the familiar derivatives in scalar case:

$$\frac{d}{dx}(ax^2) = 2ax \quad (\text{A.287})$$

A *matrix differentiation operator* is defined as

$$\frac{d}{d\mathbf{A}} \triangleq \begin{bmatrix} \frac{\partial}{\partial a_{11}} & \cdots & \frac{\partial}{\partial a_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial a_{m1}} & \cdots & \frac{\partial}{\partial a_{mn}} \end{bmatrix} \quad (\text{A.288})$$

which can be applied to any scalar function  $f(\mathbf{A})$ :

$$\frac{d}{d\mathbf{A}} f(\mathbf{A}) = \begin{bmatrix} \frac{\partial f(\mathbf{A})}{\partial a_{11}} & \cdots & \frac{\partial f(\mathbf{A})}{\partial a_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(\mathbf{A})}{\partial a_{m1}} & \cdots & \frac{\partial f(\mathbf{A})}{\partial a_{mn}} \end{bmatrix} \quad (\text{A.289})$$

Specifically, consider  $f(\mathbf{A}) = \mathbf{u}^T \mathbf{A} \mathbf{v}$ , where  $\mathbf{u}$  and  $\mathbf{v}$  are  $m \times 1$  and  $n \times 1$  constant vectors, respectively, and  $\mathbf{A}$  is an  $m \times n$  matrix. Then we have:

$$\frac{d}{d\mathbf{A}}(\mathbf{u}^T \mathbf{A} \mathbf{v}) = \mathbf{u} \mathbf{v}^T \quad (\text{A.290})$$

For more details refer to *Matrix Cookbook* at: [https://www.math.uwaterloo.ca/~hwolkowi/matrix](https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook/)

## A.19 The Fundamental Theorem of Linear Algebra

When solving a linear equation system  $\mathbf{A}\mathbf{x} = \mathbf{b}$  with an  $M \times N$  coefficient matrix  $\mathbf{A}$ , we need to answer some questions such as the following:

- Does a solution exist, i.e., can we find an  $\mathbf{x}^*$  so that  $\mathbf{A}\mathbf{x}^* = \mathbf{b}$  holds?
- If such a solution exists, is it unique? If it is not unique, how can we find all of the solutions?
- If no solution exists, can we still find the optimal approximated solution  $\hat{\mathbf{x}}$  so that the error  $\|\mathbf{e}\| = \|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|$  is minimized?
- If the system has fewer equations than unknowns ( $M < N$ ), are there infinite solutions?
- If the system has more equations than unknowns ( $M > N$ ), is there no solution?

To answer all such questions, we consider the *fundamental theorem of linear algebra* to reveal the structure of the solutions of a given linear system  $\mathbf{A}\mathbf{x} = \mathbf{b}$ .

The  $M \times N$  coefficient matrix  $\mathbf{A} \in R^{M \times N}$  can be expressed in terms of either its  $N$  M-D column vectors  $\mathbf{c}_j$  or its  $M$  N-D row vectors  $\mathbf{r}_i^T$ :

$$\mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{M1} & \cdots & a_{MN} \end{bmatrix} = [\mathbf{c}_1, \cdots, \mathbf{c}_N] = \begin{bmatrix} \mathbf{r}_1^T \\ \vdots \\ \mathbf{r}_M^T \end{bmatrix}, \quad (\text{A.291})$$

$$\mathbf{A}^T = \begin{bmatrix} a_{11} & \cdots & a_{M1} \\ \vdots & \ddots & \vdots \\ a_{1N} & \cdots & a_{MN} \end{bmatrix} = [\mathbf{r}_1, \cdots, \mathbf{r}_M] = \begin{bmatrix} \mathbf{c}_1^T \\ \vdots \\ \mathbf{c}_N^T \end{bmatrix} \quad (\text{A.292})$$

where the  $i$ th row  $\mathbf{r}_i$ , ( $i = 1, \cdots, M$ ) and the  $j$ th column  $\mathbf{c}_j$ , ( $j = 1, \cdots, N$ ) of  $\mathbf{A}$  are both represented as column vectors:

$$\mathbf{c}_j = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{Mj} \end{bmatrix}, \quad \mathbf{r}_i = \begin{bmatrix} a_{i1} \\ \vdots \\ a_{iN} \end{bmatrix} \quad \text{i.e.} \quad \mathbf{r}_i^T = [a_{i1}, \cdots, a_{iN}] \quad (\text{A.293})$$

In general a function  $y = f(x)$  can be represented by  $f : X \rightarrow Y$ , where

- $X$  is the *domain* of the function, the set of all input or argument values;
- $Y$  is the *codomain* of the function, the set into which all outputs of the function fall;
- The *image* of  $f(x)$  is the set formed by the outputs of the function corresponding to all possible  $x \in X$ , a subset of the codomain.

A linear function  $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$  maps an N-D vector  $\mathbf{x} \in \mathbb{R}^N$  in the domain of the function into an M-D vector  $\mathbf{A}\mathbf{x} \in R^M$  in the codomain of the function. The *fundamental theorem of linear algebra* concerns the following four subspaces all associated with with the  $M \times N$  coefficient matrix  $\mathbf{A}$  with  $\text{rank } R = \text{rank}(\mathbf{A}) \leq \min(M, N)$ , i.e.,  $\mathbf{A}$  has  $R$  independent columns and rows.

- The *column space* of  $\mathbf{A}$  is a space spanned by its  $N$  M-D column vectors (of which  $R \leq N$  are independent):

$$C(\mathbf{A}) = \text{span}(\mathbf{c}_1, \cdots, \mathbf{c}_N) \subseteq R^M \quad (\text{A.294})$$

This is an R-D subspace of  $R^M$  composed of all possible linear combinations of its  $N$  column vectors:

$$x_1\mathbf{c}_1 + \cdots + x_N\mathbf{c}_N = [\mathbf{c}_1, \cdots, \mathbf{c}_N] \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \mathbf{A}\mathbf{x} \quad (\text{for any } \mathbf{x} \in \mathbb{R}^N) \quad (\text{A.295})$$

The column space  $C(\mathbf{A})$  is the *image* of the linear transformation  $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$ , and the equation  $\mathbf{A}\mathbf{x} = \mathbf{b}$  is solvable if and only if  $\mathbf{b} \in C(\mathbf{A})$ . The dimension of the column space is the rank of  $\mathbf{A}$ ,  $\dim C(\mathbf{A}) = \text{rank}(\mathbf{A}) = R$ .

- The *row space* of  $\mathbf{A}$  is a space spanned by its  $M$  N-D row vectors (of which  $R \leq M$  are independent):

$$R(\mathbf{A}) = \text{span}(\mathbf{r}_1, \dots, \mathbf{r}_M) \subseteq \mathbb{R}^N \quad (\text{A.296})$$

This is an R-D subspace of  $\mathbb{R}^N$  composed of all possible linear combinations of its  $M$  row vectors:

$$y_1 \mathbf{r}_1 + \dots + y_M \mathbf{r}_M = [\mathbf{r}_1, \dots, \mathbf{r}_M] \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \mathbf{A}^T \mathbf{y} \quad (\text{for any } \mathbf{y} \in \mathbb{R}^M) \quad (\text{A.297})$$

The row space  $R(\mathbf{A})$  is the *image* of the linear transformation  $\mathbf{f}(\mathbf{y}) = \mathbf{A}^T \mathbf{y}$ , and the equation  $\mathbf{A}^T \mathbf{y} = \mathbf{c}$  is solvable if and only if  $\mathbf{c} \in R(\mathbf{A})$ . As the rows and columns in  $\mathbf{A}$  are respectively the columns and rows in  $\mathbf{A}^T$ , the row space of  $\mathbf{A}$  is the column space of  $\mathbf{A}^T$ , and the column space of  $\mathbf{A}$  is the row space of  $\mathbf{A}^T$ :

$$R(\mathbf{A}) = C(\mathbf{A}^T), \quad C(\mathbf{A}) = R(\mathbf{A}^T) \quad (\text{A.298})$$

The rank  $R = \text{rank}(\mathbf{A})$  is the number of linearly independent rows and columns of  $\mathbf{A}$ , i.e., the row space and the column space have the same dimension, both equal to the rank of  $\mathbf{A}$ :

$$\dim C(\mathbf{A}) = \dim R(\mathbf{A}) = \text{rank}(\mathbf{A}) = R \quad (\text{A.299})$$

- The *null space (kernel)* of  $\mathbf{A}$ , denoted by  $N(\mathbf{A})$ , is the set of all N-D vectors  $\mathbf{x}$  that satisfy the homogeneous equation

$$\mathbf{A} \mathbf{x} = \begin{bmatrix} \mathbf{r}_1^T \\ \vdots \\ \mathbf{r}_M^T \end{bmatrix} \mathbf{x} = \mathbf{0}, \quad \text{or} \quad \mathbf{r}_i^T \mathbf{x} = 0 \quad (i = 1, \dots, M) \quad (\text{A.300})$$

i.e.,

$$N(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^N \mid \mathbf{A} \mathbf{x} = \mathbf{0}\} \subseteq \mathbb{R}^N \quad (\text{A.301})$$

In particular, when  $\mathbf{x} = \mathbf{0}$ , we get  $\mathbf{A} \mathbf{x} = \mathbf{0}$ , i.e., the origin  $\mathbf{0} \in N(\mathbf{A})$  is in the null space.

As  $\mathbf{r}_i^T \mathbf{x} = 0$  for any  $\mathbf{x} \in N(\mathbf{A})$  and  $\mathbf{r}_i \in R(\mathbf{A})$ , we see that the null space  $N(\mathbf{A})$  and the row space  $R(\mathbf{A}) = C(\mathbf{A}^T)$  are orthogonal to each other,  $N(\mathbf{A}) \perp R(\mathbf{A})$ .

The dimension of the null space is called the *nullity* of  $\mathbf{A}$ :  $\dim N(\mathbf{A}) = \text{nullity}(\mathbf{A})$ . The rank-nullity theorem states the sum of the rank and the nullity of an  $M \times N$  matrix  $\mathbf{A}$  is equal to  $N$ :

$$\text{rank}(\mathbf{A}) + \text{nullity}(\mathbf{A}) = \dim R(\mathbf{A}) + \dim N(\mathbf{A}) = N \quad (\text{A.302})$$



We therefore see that  $R(\mathbf{A})$  and  $N(\mathbf{A})$  are two mutually exclusive and complementary subspaces of  $\mathbb{R}^N$ :

$$R(\mathbf{A}) \perp N(\mathbf{A}), \quad R(\mathbf{A}) \cap N(\mathbf{A}) = \emptyset, \quad R(\mathbf{A}) \oplus N(\mathbf{A}) = \mathbb{R}^N \quad (\text{A.303})$$

i.e., they are *orthogonal complement* of each other, denoted by

$$N(\mathbf{A}) = R(\mathbf{A})^\perp, \quad R(\mathbf{A}) = N(\mathbf{A})^\perp \quad (\text{A.304})$$

Any N-D vector  $\mathbf{x} \in \mathbb{R}^N$  is in either of the two subspaces  $R(\mathbf{A})$  and  $N(\mathbf{A})$ .

- The null space of  $\mathbf{A}^T$  (left null space of  $\mathbf{A}$ ), denoted by  $N(\mathbf{A}^T)$ , is the set of all M-D vectors  $\mathbf{y}$  that satisfy the homogeneous equation

$$\mathbf{A}^T \mathbf{y} = [\mathbf{c}_1, \dots, \mathbf{c}_N]^T \mathbf{y} = \begin{bmatrix} \mathbf{c}_1^T \\ \vdots \\ \mathbf{c}_N^T \end{bmatrix} \mathbf{y} = \mathbf{0}, \quad \text{or} \quad \mathbf{c}_j^T \mathbf{y} = 0, \quad (j = 1, \dots, N) \quad (\text{A.305})$$

i.e.,

$$N(\mathbf{A}^T) = \{\mathbf{y} \in \mathbb{R}^M \mid \mathbf{A}^T \mathbf{y} = \mathbf{0}\} \subseteq \mathbb{R}^M \quad (\text{A.306})$$

As all  $\mathbf{y} \in N(\mathbf{A}^T)$  are orthogonal to  $\mathbf{c}_i \in C(\mathbf{A})$ ,  $N(\mathbf{A}^T)$  is orthogonal to the column space  $C(\mathbf{A}) = R(\mathbf{A}^T)$ :

$$N(\mathbf{A}^T) \perp C(\mathbf{A}), \quad N(\mathbf{A}^T) \perp R(\mathbf{A}^T) \quad (\text{A.307})$$

We see that  $C(\mathbf{A})$  and  $N(\mathbf{A}^T)$  are two mutually exclusive and complementary subspaces of  $\mathbb{R}^M$ :

$$\begin{aligned} C(\mathbf{A}) \perp N(\mathbf{A}^T), \quad C(\mathbf{A}) \cap N(\mathbf{A}^T) = \emptyset, \quad C(\mathbf{A}) \oplus N(\mathbf{A}^T) = \mathbb{R}^M, \\ \text{i.e. } N(\mathbf{A}^T) = C(\mathbf{A})^\perp, \quad C(\mathbf{A}) = N(\mathbf{A}^T)^\perp \end{aligned} \quad (\text{A.308})$$

Any M-D vector  $\mathbf{y} \in \mathbb{R}^M$  is in either of the two subspaces  $C(\mathbf{A})$  and  $N(\mathbf{A}^T)$ .

The four subspaces are summarized in the figure below, showing the domain  $\mathbb{R}^N = R(\mathbf{A}) \oplus N(\mathbf{A})$  (left) and the codomain  $\mathbb{R}^M = C(\mathbf{A}) \oplus N(\mathbf{A}^T)$  (right) of the linear mapping  $\mathbf{A}\mathbf{x} = \mathbf{b}$ , where

- $\mathbf{x}_p \in R(\mathbf{A})$  is the particular solution that is mapped to  $\mathbf{A}\mathbf{x}_p = \mathbf{b} \in C(\mathbf{A})$ , the image of  $f(\mathbf{x}) = \mathbf{A}\mathbf{x}$ ;
- $\mathbf{x}_h \in N(\mathbf{A})$  is a homogeneous solution that is mapped to  $\mathbf{A}\mathbf{x}_h = \mathbf{0} \in \mathbb{R}^M$ ;
- $\mathbf{x}_c = \mathbf{x}_p + \mathbf{x}_h$  is the complete solution that is mapped to  $\mathbf{A}\mathbf{x}_c = \mathbf{b} \in C(\mathbf{A})$ .

On the other hand,  $\mathbf{y}_p \in C(\mathbf{A}) = R(\mathbf{A}^T)$ ,  $\mathbf{y}_h \in N(\mathbf{A}^T)$ , and  $\mathbf{y}_c = \mathbf{y}_p + \mathbf{y}_h$  are respectively the particular, homogeneous and complete solutions of  $\mathbf{A}^T \mathbf{y} = \mathbf{c}$ . Here we have assumed  $\mathbf{b} \in C(\mathbf{A})$  and  $\mathbf{c} \in R(\mathbf{A})$ , i.e., both  $\mathbf{A}\mathbf{x} = \mathbf{b}$  and  $\mathbf{A}^T \mathbf{y} = \mathbf{c}$  are solvable. We will also consider the case where  $\mathbf{b} \notin C(\mathbf{A})$  later.

We now consider specifically how to find the solutions of the system  $\mathbf{A}\mathbf{x} = \mathbf{b}$  in light of the four subspaces of  $\mathbf{A}$  defined above, through the examples below.

**Example 1:**

Solve the homogeneous equation system:

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 1 & 2 & 5 \\ 2 & 3 & 8 \\ 3 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \mathbf{0} \quad (\text{A.309})$$

We first convert  $\mathbf{A}$  into the rref:

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 3 & 8 \\ 3 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 5 \\ 0 & -1 & -2 \\ 0 & -5 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 5 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (\text{A.310})$$

The  $R = \text{rank}(\mathbf{A}) = 2$  columns in the rref containing a single 1, called a *pivot*, are called the *pivot columns*, and the rows containing a pivot are called the *pivot rows*. Here,  $R = 2 < M = N = 3$ , i.e.,  $\mathbf{A}$  is a singular matrix. The two pivot rows  $\mathbf{r}_1^T = [1, 0, 1]$  and  $\mathbf{r}_2^T = [0, 1, 2]$  can be used as the basis vectors that span the row space  $R(\mathbf{A})$ :

$$R(\mathbf{A}) = c_1\mathbf{r}_1 + c_2\mathbf{r}_2 = c_1 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + c_2 \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \quad (\text{A.311})$$

Note that the pivot columns of the rref do not span the column space  $C(\mathbf{A})$ , as the row reduction operations do not reserve the columns of  $\mathbf{A}$ . But they indicate the corresponding columns  $\mathbf{c}_1 = [1 \ 2 \ 3]^T$  and  $\mathbf{c}_2 = [2 \ 3 \ 1]^T$  in the original matrix  $\mathbf{A}$  can be used as the basis that spans  $C(\mathbf{A})$ . In general the bases of the row and column spaces so obtained are not orthogonal.

The  $R$  pivot rows are the independent equations in the system of  $M$  equations, and the variables corresponding to the pivot columns (here  $x_1$  and  $x_2$ ) are the *pivot variables*. The remaining  $M - R$  non-pivot rows containing all zeros are not independent, and the variables corresponding to the non-pivot rows are *free variables* (here  $x_3$ ), which can take any values.

From the rref form of the equation, we get

$$x_1 + x_3 = 0, \quad x_2 + 2x_3 = 0 \quad (\text{A.312})$$

If we let the free variable  $x_3$  take the value 1, then we can get the two pivot variables  $x_1 = -1$  and  $x_2 = -2$ , and a special homogeneous solution  $\mathbf{x}_h = [-1 \ -2 \ 1]^T$  as a basis vector that spans the 1-D null space  $N(\mathbf{A})$ . However, as the free variable  $x_3$  can take any value  $c$ , the complete solution is the entire 1-D null space:

$$N(\mathbf{A}) = c\mathbf{x}_h = c \begin{bmatrix} -1 \\ -2 \\ 1 \end{bmatrix} \quad (\text{A.313})$$

**Example 2:**

Solve the non-homogeneous equation with the same coefficient matrix  $\mathbf{A}$  used in the previous example:

$$\mathbf{Ax} = \begin{bmatrix} 1 & 2 & 5 \\ 2 & 3 & 8 \\ 3 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 19 \\ 31 \\ 22 \end{bmatrix} = \mathbf{b} \quad (\text{A.314})$$

We use Gauss-Jordan elimination to solve this system:

$$[\mathbf{A} \quad \mathbf{b}] = \left[ \begin{array}{ccc|c} 1 & 2 & 5 & 19 \\ 2 & 3 & 8 & 31 \\ 3 & 1 & 5 & 22 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 1 & 2 & 5 & 19 \\ 0 & -1 & -2 & -7 \\ 0 & -5 & -10 & -35 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 1 & 2 & 5 & 19 \\ 0 & 1 & 2 & 7 \\ 0 & 0 & 0 & 0 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 1 & 0 & 1 & 5 \\ 0 & 1 & 2 & 7 \\ 0 & 0 & 0 & 0 \end{array} \right] \quad (\text{A.315})$$

The  $R = 2$  pivot rows correspond to the independent equations in the system, i.e.,  $\text{rand}(\mathbf{A}) = 2$ , while the remaining  $M - R = 1$  non-pivot row does not play any role as they map any  $\mathbf{x}$  to  $\mathbf{0}$ . As  $\mathbf{A}$  is singular,  $\mathbf{A}^{-1}$  does not exist. However, we can find the solution based on the rref of the system, which can also be expressed in block matrix form:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \mathbf{I} & \mathbf{F} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{pivot} \\ \mathbf{x}_{free} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{0} \end{bmatrix} \quad (\text{A.316})$$

where

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \mathbf{x}_{pivot} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{x}_{free} = x_3, \quad \mathbf{b}_1 = \begin{bmatrix} 5 \\ 7 \end{bmatrix} \quad (\text{A.317})$$

Solving the matrix equation above for  $\mathbf{x}_{pivot}$ , we get

$$\mathbf{Ix}_{pivot} + \mathbf{Fx}_{free} = \mathbf{b}_1, \quad \mathbf{x}_{pivot} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{b}_1 - \mathbf{Fx}_{free} = \begin{bmatrix} 5 \\ 7 \end{bmatrix} + x_3 \begin{bmatrix} -1 \\ -2 \end{bmatrix} \quad (\text{A.318})$$

If we let  $\mathbf{x}_{free} = x_3 = 0$ , we get a particular solution  $\mathbf{x}_p = [5 \ 7 \ 0]^T$ , which can be expressed as a linear combination of  $\mathbf{r}_1$  and  $\mathbf{r}_2$  that span  $R(\mathbf{A})$ , and  $\mathbf{x}_h$  that span  $N(\mathbf{A})$ :

$$\mathbf{x}_p = \begin{bmatrix} 5 \\ 7 \\ 0 \end{bmatrix} = \frac{11}{6}\mathbf{r}_1 + \frac{2}{3}\mathbf{r}_2 - \frac{19}{6}\mathbf{x}_h = \frac{11}{6} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \frac{2}{3} \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} - \frac{19}{6} \begin{bmatrix} -1 \\ -2 \\ 1 \end{bmatrix} \quad (\text{A.319})$$

We see that this solution is not entirely in the row space  $R(\mathbf{A})$ . In general, this is the case for all particular solutions so obtained.

Having found both the particular solution  $\mathbf{x}_p$  and the homogeneous solution  $\mathbf{x}_h$ , we can further find the complete solution  $\mathbf{x}_c$  as the sum of  $\mathbf{x}_p$  and the entire null space spanned by  $\mathbf{x}_h$ :

$$\mathbf{x}_c = \begin{bmatrix} 5 \\ 7 \\ 0 \end{bmatrix} + c \begin{bmatrix} -1 \\ -2 \\ 1 \end{bmatrix} = \mathbf{x}_p + c\mathbf{x}_h = \mathbf{x}_p + N(\mathbf{A}) \quad (\text{A.320})$$

Based on different constant  $c$ , we get a set of equally valid solutions. For example, if  $c = 0, 1, 2$ , then we get

$$\mathbf{x}_0 = \begin{bmatrix} 5 \\ 7 \\ 0 \end{bmatrix}, \quad \mathbf{x}_1 = \begin{bmatrix} 4 \\ 5 \\ 1 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 3 \\ 3 \\ 2 \end{bmatrix} \quad (\text{A.321})$$

These solutions have the same projection onto the row space  $R(\mathbf{A})$ , i.e., they have the same projections onto the two basis vectors  $\mathbf{r}_1$  and  $\mathbf{r}_2$  that span  $R(\mathbf{A})$ :

$$\mathbf{x}_0^T \mathbf{r}_1 = \mathbf{x}_1^T \mathbf{r}_1 = \mathbf{x}_2^T \mathbf{r}_1 = 5, \quad \mathbf{x}_0^T \mathbf{r}_2 = \mathbf{x}_1^T \mathbf{r}_2 = \mathbf{x}_2^T \mathbf{r}_2 = 7 \quad (\text{A.322})$$

The figure below shows how the complete solution  $\mathbf{x}_c$  can be obtained as the sum of a particular solution  $\mathbf{x}_p$  in  $R(\mathbf{A})$  and the entire null space  $N(\mathbf{A})$ . Here  $N = 3$  and space  $\mathbb{R}^3$  is composed of  $R(\mathbf{A})$  and  $N(\mathbf{A})$ , respectively 2-D and 1-D on the left, but 1-D and 2-D on the right. In either case, the complete solution is any particular solution plus the entire null space, the vertical dashed line on the left, the top dashed plane on the right. All points on the vertical line or top satisfy the equation system, as they have the same projection onto the row space  $R(\mathbf{A})$ .

If the right hand side is  $\mathbf{b}' = [19, 31, 20]^T$ , then the rref of the equation becomes:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 2 \end{bmatrix} \quad (\text{A.323})$$

The non-pivot row is an impossible equation  $0 = 2$ , indicating that no solution exists, as  $\mathbf{b}' \notin C(\mathbf{A})$  is not in the column space spanned by  $\mathbf{r}_1 = [1 \ 0 \ 1]^T$  and  $\mathbf{r}_2 = [0 \ 1 \ 2]^T$ .

**Example 3:** Find the complete solution of the following linear equation system:

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ -2 & 1 & 4 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} = \mathbf{b} \quad (\text{A.324})$$

This equation can be solved in the following steps:

- Construct the augmented matrix and then convert it to the rref form:

$$\begin{aligned} \left[ \begin{array}{cccc|c} 1 & 2 & 3 & 4 & 3 \\ 4 & 3 & 2 & 1 & 2 \\ -2 & 1 & 4 & 7 & 4 \end{array} \right] &\rightarrow \left[ \begin{array}{cccc|c} 1 & 2 & 3 & 4 & 3 \\ 0 & -5 & -10 & -15 & -10 \\ 0 & 5 & 10 & 15 & 10 \end{array} \right] \rightarrow \left[ \begin{array}{cccc|c} 1 & 2 & 3 & 4 & 3 \\ 0 & 1 & 2 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right] \\ &\rightarrow \left[ \begin{array}{cccc|c} 1 & 0 & -1 & -2 & -1 \\ 0 & 1 & 2 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right] = \left[ \begin{array}{cc|c} \mathbf{I} & \mathbf{F} & \mathbf{b}_1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right] \end{aligned}$$

The two pivot rows  $\mathbf{r}_1^T = [1 \ 0 \ -1 \ -2]$  and  $\mathbf{r}_2^T = [0 \ 1 \ 2 \ 3]$  in the rref span

$R(\mathbf{A})$ , and the two columns in the *original* matrix  $\mathbf{A}$  corresponding to the pivot columns in the rref,  $[1\ 4\ -2]^T$  and  $[2\ 3\ 1]^T$  could be used as the basis vectors that span  $C(\mathbf{A})$ .

The equation system can be represented in block matrix form:

$$\begin{bmatrix} \mathbf{I} & \mathbf{F} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_p \\ \mathbf{x}_f \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{0} \end{bmatrix} \tag{A.325}$$

where

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} -1 & -2 \\ 2 & 3 \end{bmatrix}, \quad \mathbf{x}_p = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{x}_f = \begin{bmatrix} x_3 \\ x_4 \end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix} -1 \\ 2 \end{bmatrix} \tag{A.326}$$

Multiplying out we get

$$\mathbf{x}_p + \mathbf{F}\mathbf{x}_f = \mathbf{b}_1 \tag{A.327}$$

- Find the homogeneous solution for equation  $\mathbf{A}\mathbf{x} = \mathbf{0}$  by setting  $\mathbf{b}_1 = \mathbf{0}$ :

$$\mathbf{I}\mathbf{x}_p + \mathbf{F}\mathbf{x}_f = \mathbf{0}, \quad \text{i.e.} \quad \mathbf{x}_p = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -\mathbf{F}\mathbf{x}_f = \begin{bmatrix} 1 & 2 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} \tag{A.328}$$

We let  $\mathbf{x}_f$  be either of the two standard basis vectors  $\mathbf{x}_f = [x_3, x_4]^T = [1\ 0]^T$  and  $\mathbf{x}_f = [x_3, x_4]^T = [0\ 1]^T$  of the null space  $N(\mathbf{A})$ , and get

$$\mathbf{x}_p = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 2 \\ -3 \end{bmatrix} \tag{A.329}$$

and the two corresponding homogeneous solutions:

$$\mathbf{x}_{h1} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{x}_{h2} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 0 \\ 1 \end{bmatrix} \tag{A.330}$$

- Find the particular solution of the non-homogeneous equation  $\mathbf{A}\mathbf{x} = \mathbf{b}$  by setting  $\mathbf{x}_f = \mathbf{0}$

$$\mathbf{I}\mathbf{x}_p + \mathbf{F}\mathbf{x}_f = \mathbf{x}_p = \mathbf{b}_1, \quad \text{i.e.} \quad \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix} \quad \text{i.e.} \quad \mathbf{x}_p = \begin{bmatrix} -1 \\ 2 \\ 0 \\ 0 \end{bmatrix} \tag{A.331}$$

- Find the complete solution:

$$\mathbf{x}_c = \mathbf{x}_p + N(\mathbf{A}) = \mathbf{x}_p + c_1\mathbf{x}_{h1} + c_2\mathbf{x}_{h2} = \begin{bmatrix} -1 \\ 2 \\ 0 \\ 0 \end{bmatrix} + c_1 \begin{bmatrix} 1 \\ -2 \\ 1 \\ 0 \end{bmatrix} + c_2 \begin{bmatrix} 2 \\ -3 \\ 0 \\ 1 \end{bmatrix} \tag{A.332}$$

If the right-hand side is  $\mathbf{b}' = [1, 3, 5]^T$ , then the row reduction of the augmented matrix yields:

$$\left[ \begin{array}{cccc|c} 1 & 2 & 3 & 4 & 1 \\ 4 & 3 & 2 & 1 & 3 \\ -2 & 1 & 4 & 7 & 5 \end{array} \right] \rightarrow \left[ \begin{array}{cccc|c} 1 & 2 & 3 & 4 & 3 \\ 0 & -5 & -10 & -15 & -1 \\ 0 & 5 & 10 & 15 & 7 \end{array} \right] \rightarrow \left[ \begin{array}{cccc|c} 1 & 2 & 3 & 4 & 1 \\ 0 & 1 & 2 & 3 & 0.2 \\ 0 & 0 & 0 & 0 & 1.2 \end{array} \right] \quad (\text{A.333})$$

The equation corresponding to the last non-pivot row is  $0 = 1.2$ , indicating the system is not solvable (even though the coefficient matrix does not have full rank), because  $\mathbf{b}' \notin C(\mathbf{A})$  is not in the column space.

**Example 4:** Consider the linear equation system with a coefficient matrix  $\mathbf{A}^T$ , the transpose of  $\mathbf{A}$  used in the previous example:

$$\mathbf{A}^T \mathbf{y} = \begin{bmatrix} 1 & 4 & -2 \\ 2 & 3 & 1 \\ 3 & 2 & 4 \\ 4 & 1 & 7 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 11 \\ 19 \\ 27 \end{bmatrix} = \mathbf{c} \quad (\text{A.334})$$

- Convert the augmented matrix into the rref form:

$$\left[ \begin{array}{ccc|c} 1 & 4 & -2 & 3 \\ 2 & 3 & 1 & 11 \\ 3 & 2 & 4 & 19 \\ 4 & 1 & 7 & 27 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 1 & 4 & -2 & 3 \\ 0 & -5 & 5 & 5 \\ 0 & -10 & 10 & 10 \\ 0 & -15 & 15 & 15 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 1 & 4 & -2 & 3 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 1 & 0 & 2 & 7 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right] \quad (\text{A.335})$$

The two pivot rows  $\mathbf{r}_1^T = [1 \ 0 \ 2]$  and  $\mathbf{r}_2^T = [0 \ 1 \ -1]$  are the basis vectors that span  $R(\mathbf{A}^T) = C(\mathbf{A})$ . The two vectors that span  $C(\mathbf{A})$  found in Example 3 can be expressed as linear combinations of  $\mathbf{r}_1$  and  $\mathbf{r}_2$ ,  $[1 \ 4 \ -2]^T = \mathbf{r}_1 + 4\mathbf{r}_2$  and  $[2 \ 3 \ 1]^T = 2\mathbf{r}_1 + 3\mathbf{r}_2$ , i.e., either of the two pairs can be used as the basis that span  $C(\mathbf{A})$ .

Based on the rref above, the equation system can now be written as

$$\begin{bmatrix} \mathbf{I} & \mathbf{F} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{y}_p \\ \mathbf{y}_f \end{bmatrix} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{0} \end{bmatrix} \quad (\text{A.336})$$

where

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \quad \mathbf{y}_p = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad \mathbf{y}_f = y_3, \quad \mathbf{c}_1 = \begin{bmatrix} 7 \\ -1 \end{bmatrix} \quad (\text{A.337})$$

Multiplying out we get

$$\mathbf{I}\mathbf{y}_p + \mathbf{F}\mathbf{y}_f = \mathbf{c}_1 \quad (\text{A.338})$$

- Find the homogeneous solution for  $\mathbf{A}^T \mathbf{y} = \mathbf{c} = \mathbf{0}$  by setting  $\mathbf{c} = \mathbf{0}$  and thereby

$\mathbf{c}_1 = \mathbf{0}$ . We let the free variable  $\mathbf{y}_f = y_3 = 1$  and get

$$\mathbf{I}\mathbf{y}_p + \mathbf{F}\mathbf{y}_f = \mathbf{0}, \quad \text{i.e.} \quad \mathbf{y}_p = -\mathbf{F}\mathbf{y}_f = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = - \begin{bmatrix} 2 \\ -1 \end{bmatrix} y_3 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \quad \text{i.e.} \quad \mathbf{y}_h = \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix} \quad (\text{A.339})$$

- Find the particular solution of the non-homogeneous equation  $\mathbf{A}^T \mathbf{y} = \mathbf{c} \neq \mathbf{0}$  by setting  $\mathbf{y}_f = y_3 = 0$ :

$$\mathbf{I}\mathbf{y}_p + \mathbf{F}\mathbf{y}_f = \mathbf{y}_p = \mathbf{c}_1 = \begin{bmatrix} 7 \\ -1 \end{bmatrix}, \quad \text{i.e.} \quad \mathbf{y}_p = \begin{bmatrix} 7 \\ -1 \\ 0 \end{bmatrix} \quad (\text{A.340})$$

- Find the complete solution:

$$\mathbf{y}_c = \mathbf{y}_p + N(\mathbf{A}^T) = \mathbf{y}_p + c\mathbf{y}_3 = \begin{bmatrix} 7 \\ -1 \\ 0 \end{bmatrix} + c \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix} \quad (\text{A.341})$$

If the right-hand side is  $\mathbf{c}' = [1, 1, 2, 3]^T$ ,

$$\left[ \begin{array}{ccc|c} 1 & 4 & -2 & 1 \\ 2 & 3 & 1 & 1 \\ 3 & 2 & 4 & 2 \\ 4 & 1 & 7 & 3 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 1 & 4 & -2 & 1 \\ 0 & -5 & 5 & -1 \\ 0 & -10 & 10 & -1 \\ 0 & -15 & 15 & -1 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 1 & 4 & -2 & 1 \\ 0 & 1 & -1 & 1/5 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 \end{array} \right] \quad (\text{A.342})$$

indicating the system is not solvable, as this  $\mathbf{c}' \notin C(\mathbf{A}^T) = R(\mathbf{A})$ , i.e., it is not in the column space of  $\mathbf{A}^T$  or row space of  $\mathbf{A}$ .

In the two examples above, we have obtained all four subspaces associated with this matrix  $\mathbf{A}$  with  $M = 3$ ,  $N = 4$ , and  $R = 2$ , in terms of the bases that span the subspaces:

1. The row space  $R(\mathbf{A}) = C(\mathbf{A}^T)$  is an R-D subspace of  $\mathbb{R}^4$ , spanned by the  $R = 2$  pivot rows of the rref of  $\mathbf{A}$ :

$$R(\mathbf{A}) = C(\mathbf{A}^T) = \text{span} \left( \left( \begin{bmatrix} 1 \\ 0 \\ -1 \\ -2 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} \right) \right) \quad (\text{A.343})$$

2. The null space  $N(\mathbf{A})$  is an (N-R)-D subspace of  $\mathbb{R}^4$  spanned by the  $N - R$  independent homogeneous solutions:

$$N(\mathbf{A}) = \text{span} \left( \left( \begin{bmatrix} 1 \\ -2 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ -3 \\ 0 \\ 1 \end{bmatrix} \right) \right) \quad (\text{A.344})$$

Note that the basis vectors of  $N(\mathbf{A})$  are indeed orthogonal to those of  $R(\mathbf{A})$ .

3. The column space of  $\mathbf{A}$  is the same as the row space of  $\mathbf{A}^T$ , which is the R-D subspace of  $\mathbb{R}^3$  spanned by the two pivot rows of the rref of  $\mathbf{A}^T$ .

$$C(\mathbf{A}) = R(\mathbf{A}^T) = \text{span} \left( \left[ \begin{array}{c} 1 \\ 0 \\ 2 \end{array} \right], \left[ \begin{array}{c} 0 \\ 1 \\ -1 \end{array} \right] \right) \quad (\text{A.345})$$

4. The left null space  $N(\mathbf{A}^T)$  is a (M-R)-D subspace of  $\mathbb{R}^3$  spanned by the homogeneous solutions (here one  $3 - 2 = 1$  solution):

$$N(\mathbf{A}^T) = \text{span} \left( \left[ \begin{array}{c} -2 \\ 1 \\ 1 \end{array} \right] \right) = c \left[ \begin{array}{c} -2 \\ 1 \\ 1 \end{array} \right] \quad (\text{A.346})$$

Again note that the basis vectors of  $N(\mathbf{A}^T)$  are orthogonal to those of  $C(\mathbf{A})$ .

In general, here are the ways to find the bases of the four subspaces:

- The basis vectors of  $R(\mathbf{A})$  are the pivot rows of the rref of  $\mathbf{A}$ .
- The basis vectors of  $C(\mathbf{A})$  are the pivot rows of the rref of  $\mathbf{A}^T$ .
- The basis vectors of  $N(\mathbf{A})$  are the independent homogeneous solutions of  $\mathbf{A}\mathbf{x} = \mathbf{0}$ . To find them, reduce  $\mathbf{A}$  to the rref, identify all free variables  $\mathbf{x}_f$  corresponding to non-pivot columns, set one of them to 1 and the rest to 0, solve homogeneous system  $\mathbf{A}\mathbf{x} = \mathbf{0}$  to find the pivot variables  $\mathbf{x}_p$  to get one basis vector. Repeat the process for each of the free variables to get all basis vectors.
- The basis vectors of  $R(\mathbf{A}^T)$  can be obtained by doing the same as above for  $\mathbf{A}^T$ .

Note that while the basis of  $R(\mathbf{A})$  are the pivot rows of the rref of  $\mathbf{A}$ , as its rows are equivalent to those of  $\mathbf{A}$ , the pivot columns of the rref basis of  $C(\mathbf{A})$  are not the basis of  $C(\mathbf{A})$ , as the columns of  $\mathbf{A}$  have been changed by the row deduction operations and are therefore not equivalent to the columns of the resulting rref. The columns in  $\mathbf{A}$  corresponding to the pivot columns in the rref could be used as the basis of  $C(\mathbf{A})$ . Alternatively, the basis of  $C(\mathbf{A})$  can be obtained from the rref of  $\mathbf{A}^T$ , as its rows are equivalent to those of  $\mathbf{A}^T$ , which are the columns of  $\mathbf{A}$ .

We further make the following observations:

- The basis vectors of each of the four subspaces are independent, the basis vectors of  $R(\mathbf{A})$  and  $N(\mathbf{A})$  are orthogonal, and  $\dim R(\mathbf{A}) + \dim N(\mathbf{A}) = M$ . Similarly, the basis vectors of  $C(\mathbf{A})$  and  $N(\mathbf{A}^T)$  are orthogonal, and  $\dim C(\mathbf{A}) + \dim N(\mathbf{A}^T) = N$ . In other words, the four subspaces indeed satisfy the following orthogonal and complementary properties:

$$C(\mathbf{A}) \perp N(\mathbf{A}^T), \quad R(\mathbf{A}) \perp N(\mathbf{A}), \quad C(\mathbf{A}) \oplus N(\mathbf{A}^T) = \mathbb{R}^3, \quad R(\mathbf{A}) \oplus N(\mathbf{A}) = \mathbb{R}^4 \quad (\text{A.347})$$

i.e., they are orthogonal complements:  $C(\mathbf{A}) = N(\mathbf{A}^T)^\perp$ ,  $R(\mathbf{A}) = N(\mathbf{A})^\perp$ .



- For  $\mathbf{Ax} = \mathbf{b}$  to be solvable, the constant vector  $\mathbf{b}$  on the right-hand side must be in the column space,  $\mathbf{b} \in C(\mathbf{A})$ . Otherwise the equation is not solvable, even if  $R < M < N$ . Similarly, for  $\mathbf{A}^T \mathbf{x} = \mathbf{c}$  to be solvable,  $\mathbf{c}$  must be in the row space  $\mathbf{c} \in R(\mathbf{A})$ . In the examples above, both  $\mathbf{b}$  and  $\mathbf{c}$  are indeed in their corresponding column spaces:

$$\mathbf{b} = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} + 2 \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} \in C(\mathbf{A}), \quad \mathbf{c} = \begin{bmatrix} 3 \\ 11 \\ 19 \\ 27 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 0 \\ -1 \\ -2 \end{bmatrix} + 11 \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} \in R(\mathbf{A}) \tag{A.348}$$

But as  $\mathbf{b} = [1, 3, 5]^T \notin C(\mathbf{A})$  and  $\mathbf{c} = [1, 1, 2, 3]^T \notin R(\mathbf{A})$ , the corresponding systems have no solutions.

- All homogeneous solutions of  $\mathbf{Ax} = \mathbf{0}$  are in the null space  $\mathbf{x}_h \in N(\mathbf{A})$ , but in general the particular solutions  $\mathbf{x}_p \in R^N = R(\mathbf{A}) \oplus N(\mathbf{A})$  are not necessarily in the row space  $R(\mathbf{A})$ . In the example above,  $\mathbf{x}_p \in R^N$  is a linear combination of the  $R = 2$  basis vectors of  $R(\mathbf{A})$  and  $N - R = 2$  basis vectors of  $N(\mathbf{A})$ :

$$\mathbf{x}_p = \begin{bmatrix} -1 \\ 2 \\ 0 \\ 0 \end{bmatrix} = \left( 0.1 \begin{bmatrix} 1 \\ 0 \\ -1 \\ -2 \end{bmatrix} + 0.2 \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} \right) + \left( -0.3 \begin{bmatrix} 1 \\ -2 \\ 1 \\ 0 \end{bmatrix} - 0.4 \begin{bmatrix} 2 \\ -3 \\ 0 \\ 1 \end{bmatrix} \right) = \mathbf{x}'_p + \mathbf{x}''_p \tag{A.349}$$

where

$$\mathbf{x}'_p = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} \in R(\mathbf{A}), \quad \mathbf{x}''_p = \begin{bmatrix} -1.1 \\ 1.8 \\ -0.3 \\ -0.4 \end{bmatrix} \in N(\mathbf{A}) \tag{A.350}$$

are the projections of  $\mathbf{x}_p$  onto  $R(\mathbf{A})$  and  $N(\mathbf{A})$ , respectively, and  $\mathbf{x}'_p$  is another particular solution without any homogeneous component that satisfies  $\mathbf{Ax} = \mathbf{b}$ , while  $\mathbf{x}''_p$  is a homogeneous solution satisfying  $\mathbf{Ax} = \mathbf{0}$ .

- All homogeneous solutions of  $\mathbf{A}^T \mathbf{y} = \mathbf{c}$  are in the left null space  $\mathbf{y}_h \in N(\mathbf{A}^T)$ , but in general the particular solutions  $\mathbf{y}_p \in C(\mathbf{A}) \oplus N(\mathbf{A}^T) = R^M$  are not necessarily in the column space. In the example above,  $\mathbf{y}_p \in R^M$  is a linear combination of the  $R = 2$  basis vectors of  $C(\mathbf{A})$  and  $M - R = 1$  basis vector of  $N(\mathbf{A}^T)$ :

$$\mathbf{y}_p = \begin{bmatrix} 7 \\ -1 \\ 0 \end{bmatrix} = \left( 2 \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} + 1.5 \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} \right) - 2.5 \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1.5 \\ 2.5 \end{bmatrix} - 2.5 \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix} = \mathbf{y}'_p + \mathbf{y}''_p \tag{A.351}$$

where  $\mathbf{y}'_p = [2, 1.5, 2.5]^T \in C(\mathbf{A})$  is a particular solution (without any homogeneous component) that satisfies  $\mathbf{A}^T \mathbf{y} = \mathbf{c}$ .

Here is a summary of the four subspaces associated with an  $M$  by  $N$  matrix  $\mathbf{A}$  of rank  $R$ .

	$\dim R(\mathbf{A})$	$\dim N(\mathbf{A})$	$\dim C(\mathbf{A})$	$\dim R(\mathbf{A}^T)$	solvability of $\mathbf{Ax} = \mathbf{b}$
$M = N = R$	$R$	$0$	$R$	$0$	unique solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$
$M > N = R$	$R$	$0$	$R$	$M - R$	over-constrained, solvable if
$N > M = R$	$R$	$N - R$	$R$	$0$	under-constrained, infinite s
$R < \min(M, N)$	$R$	$N - R$	$R$	$M - R$	solvable if $\mathbf{b} \in C(\mathbf{A})$ , infinite

(A.352)

The figure below illustrates a specific case with  $M = N = 3$  and  $R = 2$ . As  $\mathbf{b} \notin C(\mathbf{A})$ , the system can only be approximately solved to find  $\mathbf{Ax} = \mathbf{p} \in C(\mathbf{A})$ , which is the projection of  $\mathbf{b}$  onto the column space  $C(\mathbf{A})$ . The error  $\|\mathbf{e}\| = \|\mathbf{p} - \mathbf{b}\| = \|\mathbf{Ax}_p - \mathbf{b}\|$  is minimized,  $\mathbf{x}_p$  is the optimal approximation. We will consider ways to obtain this optimal approximation in the following sections.

## A.20 Pseudo-Inverse Solutions Based on SVD

In the previous section we obtained the solution of the equation  $\mathbf{Ax} = \mathbf{b}$  together with the bases of the four subspaces of  $\mathbf{A}$  based its rref. Here we will consider an alternative and better way to solve the same equation and find a set of orthogonal bases that also span the four subspaces, based on the pseudo-inverse and the singular value decomposition (SVD) of  $\mathbf{A}$ . The solution obtained this way is optimal in some certain sense as shown below.

Consider the SVD of an  $M \times N$  matrix  $\mathbf{A}$  of rank  $R \leq \min(M, N)$ :

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (\text{A.353})$$

where

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & & & & & & 0 \\ & \ddots & & & & & \\ & & \sigma_R & & & & \\ & & & 0 & & & \\ & & & & \ddots & & \\ 0 & & & & & & 0 \end{bmatrix}_{M \times N} \quad (\text{A.354})$$

is an  $M \times N$  matrix with  $R$  non-zero *singular values*  $\sigma_i = \sqrt{\lambda_i}$  ( $i = 1, \dots, R$ ) of  $\mathbf{A}$  along the diagonal (starting from the top-left corner), while all other components are zero, and  $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_M]$  and  $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_N]$  are two orthogonal matrices of dimensions  $M \times M$  and  $N \times N$  respectively. The column vectors  $\mathbf{u}_i$  ( $i = 1, \dots, M$ ) and  $\mathbf{v}_j$  ( $j = 1, \dots, N$ ), are called the left and right *singular vectors* of  $\mathbf{A}$ , respectively, and they can be used as the orthonormal bases to span respectively  $\mathbb{R}^M = C(\mathbf{A}) \oplus N(\mathbf{A}^T)$  and its subspaces  $C(\mathbf{A})$  and  $N(\mathbf{A}^T)$ , and  $\mathbb{R}^N = C(\mathbf{A}) \oplus N(\mathbf{A})$  and its subspaces  $C(\mathbf{A})$  and  $N(\mathbf{A})$ .

To see this, we rewrite the SVD of  $\mathbf{A}$  as:

$$\begin{aligned} \mathbf{A} &= [\mathbf{c}_1, \dots, \mathbf{c}_N] = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \\ &= [\mathbf{u}_1, \dots, \mathbf{u}_M] \begin{bmatrix} \sigma_1 & & & & 0 \\ & \ddots & & & \\ & & \sigma_R & & \\ & & & 0 & \\ 0 & & & & \ddots \\ & & & & & 0 \end{bmatrix}_{M \times N} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{v}_N^T \end{bmatrix} \\ &= \sum_{k=1}^R \sigma_k (\mathbf{u}_k \mathbf{v}_k^T) = \sum_{k=1}^R \sigma_k \mathbf{u}_k [v_{1k}, \dots, v_{Nk}] = \left[ \sum_{k=1}^R (\sigma_k v_{1k}) \mathbf{u}_k, \dots, \sum_{k=1}^R (\sigma_k v_{Nk}) \mathbf{u}_k \right] \end{aligned} \quad (\text{A.355})$$

Each column vector of  $\mathbf{A}$  can be expressed as a linear combination of the first  $R$  columns of  $\mathbf{U}$  corresponding to the non-zero singular values:

$$\mathbf{c}_j = \sum_{k=1}^R (\sigma_k v_{jk}) \mathbf{u}_k \quad (j = 1, \dots, N), \quad (\text{A.356})$$

Taking transpose on both sides of the SVD we also get:

$$\begin{aligned} (\mathbf{A}^T) &= [\mathbf{r}_1, \dots, \mathbf{r}_M] = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\ &= \sum_{k=1}^R \sigma_k (\mathbf{v}_k \mathbf{u}_k^T) = \sum_{k=1}^R \sigma_k \mathbf{v}_k [u_{1k}, \dots, u_{Mk}] = \left[ \sum_{k=1}^R (\sigma_k u_{1k}) \mathbf{v}_k, \dots, \sum_{k=1}^R (\sigma_k u_{Mk}) \mathbf{v}_k \right] \end{aligned} \quad (\text{A.357})$$

i.e., each row vector of  $\mathbf{A}$  can be expressed as a linear combination of the first  $R$  columns of  $\mathbf{V}$  corresponding to the non-zero singular values:

$$\mathbf{r}_i = \sum_{k=1}^R (\sigma_k u_{ik}) \mathbf{v}_k \quad (i = 1, \dots, M) \quad (\text{A.358})$$

We therefore see that the  $R$  columns of  $\mathbf{U}$  and  $\mathbf{V}$  corresponding to the non-zero singular values span respectively the column space  $C(\mathbf{A})$  and row space  $R(\mathbf{A})$ :

$$C(\mathbf{A}) = \text{span}(\mathbf{u}_1, \dots, \mathbf{u}_R), \quad R(\mathbf{A}) = \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_R) \quad (\text{A.359})$$

and the remaining  $M - R$  columns of  $\mathbf{U}$  and  $N - R$  columns of  $\mathbf{V}$  corresponding to the zero singular values span respectively  $N(\mathbf{A}^T)$  orthogonal to  $C(\mathbf{A})$ , and  $N(\mathbf{A})$  orthogonal to  $R(\mathbf{A})$ :

$$N(\mathbf{A}^T) = \text{span}(\mathbf{u}_{R+1}, \dots, \mathbf{u}_M), \quad N(\mathbf{A}) = \text{span}(\mathbf{v}_{R+1}, \dots, \mathbf{v}_N) \quad (\text{A.360})$$

Subspace	Definition	Dimension	Bas
$C(\mathbf{A}) \subseteq \mathbb{R}^M$	column space (image) of $\mathbf{A}$	$R$	$R$ columns of $\mathbf{U}$ corresponding
$N(\mathbf{A}^T) \subseteq \mathbb{R}^M$	left null space of $\mathbf{A}^T$	$M - R$	$M - R$ columns of $\mathbf{U}$ correspon
$R(\mathbf{A}) \subseteq \mathbb{R}^N$	row space (image) of $\mathbf{A}^T$	$R$	$R$ columns of $\mathbf{V}$ corresponding
$N(\mathbf{A}) \subseteq \mathbb{R}^N$	null space of $\mathbf{A}$	$N - R$	$N - R$ columns of $\mathbf{V}$ correspo
$\mathbb{R}^N = N(\mathbf{A}) \oplus R(\mathbf{A})$	domain of $\mathbf{A}$	$N$	all $N$ colu
$\mathbb{R}^M = N(\mathbf{A}^T) \oplus C(\mathbf{A})$	codomain of $\mathbf{A}$	$M$	all $M$ colu

(A.361)

The SVD method can be used to find the pseudo-inverse of an  $M \times N$  matrix  $\mathbf{A}$  of rank  $R \leq \min(M, N)$ :

$$\mathbf{A}^- = \mathbf{V}\Sigma^- \mathbf{U}^T \tag{A.362}$$

where both  $\mathbf{A}^-$  and

$$\Sigma^- = \begin{bmatrix} 1/\sigma_1 & & & & 0 \\ & \ddots & & & \\ & & 1/\sigma_R & & \\ & & & 0 & \\ 0 & & & & \ddots \\ & & & & & 0 \end{bmatrix}_{N \times M} \tag{A.363}$$

are  $N \times M$  matrices.

We further note that matrices  $\mathbf{U}$  and  $\mathbf{V}$  are related to each other by:

$$\begin{cases} \mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T \\ \mathbf{A}^- = \mathbf{V}\Sigma^- \mathbf{U}^T \end{cases} \quad \text{or} \quad \begin{cases} \mathbf{A}\mathbf{V} = \mathbf{U}\Sigma \\ \mathbf{A}^- \mathbf{U} = \mathbf{V}\Sigma^- \end{cases} \tag{A.364}$$

or in vector form:

$$\begin{cases} \mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i & (i = 1, \dots, R) \\ \mathbf{A}^- \mathbf{u}_i = \mathbf{v}_i / \sigma_i & (i = 1, \dots, R) \end{cases} \quad \begin{cases} \mathbf{A}\mathbf{v}_i = 0 & (i = R + 1, \dots, N) \\ \mathbf{A}^- \mathbf{u}_i = 0 & (i = R + 1, \dots, M) \end{cases} \tag{A.365}$$

indicating how the individual columns are related. We see that the last  $N - R$  columns  $\mathbf{v}_{R+1}, \dots, \mathbf{v}_N$  of  $\mathbf{V}$  form an orthogonal basis of  $N(\mathbf{A})$ ; and the last  $M - R$  columns  $\mathbf{u}_{R+1}, \dots, \mathbf{u}_M$  of  $\mathbf{U}$  form an orthogonal basis of  $N(\mathbf{A}^T)$ .

We now show that the optimal solution of the linear system  $\mathbf{A}\mathbf{x} = \mathbf{b}$  can be obtained based on the pseudo-inverse of  $\mathbf{A}$ :

$$\mathbf{x}_{svd} = \mathbf{A}^- \mathbf{b} = \mathbf{V}\Sigma^- \mathbf{U}^T \mathbf{b} \tag{A.366}$$

Specially, if  $M = N = R$ , then  $\mathbf{A}^- = \mathbf{A}^{-1}$ , and  $\mathbf{x}_{svd} = \mathbf{A}^{-1} \mathbf{b}$  is the unique and exact solution. In general when  $M \neq N$  or  $R < \min(M, N)$ , the solution may not exist, or it may not be unique, but  $\mathbf{x}_{svd}$  is still an optimal solution in two ways, from the perspective of both the domain and codomain of the linear mapping  $\mathbf{A}\mathbf{x}$ , as shown below.

- In domain  $\mathbb{R}^N$ :

Pre-multiplying  $\mathbf{V}^T = \mathbf{V}^{-1}$  on both sides of the pseudo-inverse solution  $\mathbf{x}_{svd} = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}$  given above, we get:

$$\mathbf{V}^T\mathbf{x}_{svd} = [\mathbf{v}_1, \dots, \mathbf{v}_N]^T\mathbf{x}_{svd} = \boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} = \boldsymbol{\Sigma}^{-1}[\mathbf{u}_1, \dots, \mathbf{u}_M]^T\mathbf{b}, \quad (\text{A.367})$$

or in component form:

$$\begin{bmatrix} \mathbf{v}_1^T\mathbf{x}_{svd} \\ \vdots \\ \mathbf{v}_R^T\mathbf{x}_{svd} \\ \mathbf{v}_{R+1}^T\mathbf{x}_{svd} \\ \vdots \\ \mathbf{v}_N^T\mathbf{x}_{svd} \end{bmatrix} = \begin{bmatrix} 1/\sigma_1 & & & & 0 \\ & \ddots & & & \\ & & 1/\sigma_R & & \\ & & & 0 & \\ 0 & & & & \ddots \\ & & & & & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^T\mathbf{b} \\ \vdots \\ \mathbf{u}_R^T\mathbf{b} \\ \mathbf{u}_{R+1}^T\mathbf{b} \\ \vdots \\ \mathbf{u}_M^T\mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1^T\mathbf{b}/\sigma_1 \\ \vdots \\ \mathbf{u}_R^T\mathbf{b}/\sigma_R \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (\text{A.368})$$

The first  $R$  components  $\mathbf{v}_i^T\mathbf{x}_{svd} = \mathbf{u}_i^T\mathbf{b}/\sigma_i$  ( $i = 1, \dots, R$ ) are the projection of  $\mathbf{x}_{svd}$  onto  $R(\mathbf{A})$  spanned by  $\{\mathbf{v}_1, \dots, \mathbf{v}_R\}$  corresponding to the non-zero singular values, while the last  $N - R$  components  $\mathbf{v}_i^T\mathbf{x}_{svd} = 0$  ( $i = R + 1, \dots, N$ ) are the projection of  $\mathbf{x}_{svd}$  onto  $N(\mathbf{A})$  spanned by  $\{\mathbf{v}_{R+1}, \dots, \mathbf{v}_N\}$  corresponding to the zero singular values. We see that the pseudo-inverse solution  $\mathbf{x}_{svd}$  is entirely in  $R(\mathbf{A})$ , containing no homogeneous component in  $N(\mathbf{A})$ . In other words,  $\mathbf{x}_{svd}$  has the minimum norm (closest to the origin) compared to any other possible solution  $\mathbf{x}_p$ , such as those found previously based on the rref of  $\mathbf{A}$ , containing a non-zero homogeneous component  $\mathbf{x}_h \in N(\mathbf{A})$ :

$$\|\mathbf{x}_{svd}\| \leq \|\mathbf{x}_p\| = \|\mathbf{x}_{svd} + \mathbf{x}_h\| \quad (\text{A.369})$$

As  $N(\mathbf{A}) \perp R(\mathbf{A})$ ,  $\mathbf{x}_{svd} \in R(\mathbf{A})$  is the projection of any such solution  $\mathbf{x}_p$  onto  $R(\mathbf{A})$ . The complete solution can be found as

$$\mathbf{x}_c = \mathbf{x}_{svd} + N(\mathbf{A}) = \mathbf{x}_{svd} + \sum_{j=R+1}^N c_j\mathbf{v}_j \quad (\text{A.370})$$

for any set of coefficients  $c_j$ .

- In codomain  $\mathbb{R}^M$ :

We now consider the result  $\mathbf{b}_{svd} = \mathbf{A}\mathbf{x}_{svd}$  produced by the pseudo-inverse solution  $\mathbf{x}_{svd}$ , which, as a linear combination of the columns of  $\mathbf{A}$ , is in its column space  $C(\mathbf{A})$ :

$$\mathbf{b}_{svd} = \mathbf{A}\mathbf{x}_{svd} = \mathbf{A}\mathbf{A}^{-}\mathbf{b} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T\mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} = \mathbf{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} \quad (\text{A.371})$$

Pre-multiplying  $\mathbf{U}^T = \mathbf{U}^{-1}$  on both sides we get:

$$\mathbf{U}^T\mathbf{b}_{svd} = \boldsymbol{\Sigma}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} \quad (\text{A.372})$$

or in component form:

$$\begin{bmatrix} \mathbf{u}_1^T \mathbf{b}_{svd} \\ \vdots \\ \mathbf{u}_R^T \mathbf{b}_{svd} \\ \mathbf{u}_{R+1}^T \mathbf{b}_{svd} \\ \vdots \\ \mathbf{u}_M^T \mathbf{b}_{svd} \end{bmatrix} = \begin{bmatrix} 1 & & & & 0 \\ & \ddots & & & \\ & & 1 & & \\ & & & 0 & \\ & & & & \ddots \\ 0 & & & & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^T \mathbf{b} \\ \vdots \\ \mathbf{u}_R^T \mathbf{b} \\ \mathbf{u}_{R+1}^T \mathbf{b} \\ \vdots \\ \mathbf{u}_M^T \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1^T \mathbf{b} \\ \vdots \\ \mathbf{u}_R^T \mathbf{b} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (\text{A.373})$$

The first  $R$  components  $\mathbf{u}_i^T \mathbf{b}_{svd} = \mathbf{u}_i^T \mathbf{b}$  ( $i = 1, \dots, R$ ) are the projection of  $\mathbf{b}_{svd}$  onto  $C(\mathbf{A})$  spanned by  $\{\mathbf{u}_1, \dots, \mathbf{u}_R\}$  corresponding to the non-zero singular values, while the last  $M - R$  components  $\mathbf{u}_i^T \mathbf{b}_{svd} = 0$  ( $i = R + 1, \dots, M$ ) are the projection of  $\mathbf{b}_{svd}$  onto  $N(\mathbf{A}^T)$  spanned by  $\{\mathbf{u}_{R+1}, \dots, \mathbf{u}_M\}$  corresponding to the zero singular values. We see that  $\mathbf{b}_{svd}$  and  $\mathbf{b}$  have the same projection onto  $C(\mathbf{A})$ . If  $\mathbf{b} \in C(\mathbf{A})$ , i.e.,  $\mathbf{u}_i^T \mathbf{b} = 0$  for all  $i = R + 1, \dots, M$ , then  $\mathbf{b}_{svd} = \mathbf{A}\mathbf{x}_{svd} = \mathbf{b}$  and  $\mathbf{x}_{svd}$  is an exact solution. But if  $\mathbf{b} \notin C(\mathbf{A})$ , as it contains non-zero components in  $N(\mathbf{A}^T)$ , then no solution exists. However, as  $N(\mathbf{A}^T) \perp C(\mathbf{A})$ ,  $\mathbf{b}_{svd} \in C(\mathbf{A})$  is the projection of  $\mathbf{b}$  onto  $C(\mathbf{A})$ , the error  $\|\mathbf{b} - \mathbf{b}_{svd}\|$  is minimized when compared with that of any other approximate solution  $\mathbf{x}_p \neq \mathbf{x}_{svd}$ :

$$\|\mathbf{b} - \mathbf{b}_{svd}\| = \|\mathbf{b} - \mathbf{A}\mathbf{x}_{svd}\| \leq \|\mathbf{b} - \mathbf{A}\mathbf{x}_p\| \quad (\text{A.374})$$

In this sense,  $\mathbf{x}_{svd}$  is the optimal.

Summarizing the two aspects above, we see that the pseudo-inverse solution  $\mathbf{x}_{svd}$  is optimal in the sense that both its norm  $\|\mathbf{x}_{svd}\|$  and its error  $\|\mathbf{A}\mathbf{x}_{svd} - \mathbf{b}\|$  are minimized.

- If the solution  $\mathbf{x}_{svd}$  is not unique because  $N(\mathbf{A}) \neq \emptyset$ , then the complete solution can be found by adding the entire null space to it:  $\mathbf{x}_c = \mathbf{x}_{svd} + N(\mathbf{A})$ .
- If no solution exists because  $\mathbf{b} \notin C(\mathbf{A})$ , then  $\mathbf{x}_{svd}$  is the optimal approximate solution with minimum error.

**Example:** Given the same system considered in previous examples

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ -2 & 1 & 4 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} \quad (\text{A.375})$$

we will now solve it using the pseudo-inverse method. We first find SVD of  $\mathbf{A}$  in terms of the following matrices:

$$\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3] = \begin{bmatrix} -0.535 & -0.218 & 0.817 \\ -0.267 & -0.873 & -0.408 \\ -0.802 & 0.436 & -0.408 \end{bmatrix} \quad (\text{A.376})$$

$$\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4] = \begin{bmatrix} 0.000 & -0.837 & 0.374 & -0.400 \\ -0.267 & -0.478 & -0.797 & 0.255 \\ -0.535 & -0.120 & 0.472 & 0.691 \\ -0.802 & 0.239 & -0.049 & -0.546 \end{bmatrix} \quad (\text{A.377})$$

$$\mathbf{\Sigma} = \begin{bmatrix} 10.0 & 0 & 0 & 0 \\ 0 & 5.477 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{\Sigma}^{-} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.183 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (\text{A.378})$$

$$\begin{aligned} \mathbf{A}^{-} = \mathbf{V}\mathbf{\Sigma}^{-}\mathbf{U}^T &= \begin{bmatrix} 0.000 & -0.837 & 0.374 & -0.400 \\ -0.267 & -0.478 & -0.797 & 0.255 \\ -0.535 & -0.120 & 0.472 & 0.691 \\ -0.802 & 0.239 & -0.049 & -0.546 \end{bmatrix} \begin{bmatrix} 0.100 & 0 & 0 \\ 0 & 0.183 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.535 & -0.267 & 0.802 \\ -0.218 & -0.873 & 0.436 \\ 0.817 & -0.408 & -0.408 \end{bmatrix} \\ &= \begin{bmatrix} 0.033 & 0.133 & -0.067 \\ 0.033 & 0.083 & -0.017 \\ 0.033 & 0.033 & 0.033 \\ 0.033 & -0.017 & 0.083 \end{bmatrix} \end{aligned}$$

The particular solution of the system  $\mathbf{Ax} = \mathbf{b}$  with  $\mathbf{b} = [3, 2, 4]^T \in C(\mathbf{A})$  is

$$\begin{aligned} \mathbf{x}_{svd} = \mathbf{A}^{-}\mathbf{b} &= \begin{bmatrix} 0.033 & 0.133 & -0.067 \\ 0.033 & 0.083 & -0.017 \\ 0.033 & 0.033 & 0.033 \\ 0.033 & -0.017 & 0.083 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} \\ &= -0.535 \begin{bmatrix} 0.000 \\ -0.267 \\ -0.535 \\ -0.802 \end{bmatrix} - 0.120 \begin{bmatrix} -0.837 \\ -0.478 \\ -0.120 \\ 0.239 \end{bmatrix} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 \end{aligned}$$

which is in  $R(\mathbf{A})$  spanned by the first  $R = 2$  columns  $\mathbf{v}_1$  and  $\mathbf{v}_2$ , perpendicular to  $N(\mathbf{A})$  spanned by the last  $N - R = 2$  columns  $\mathbf{v}_3$  and  $\mathbf{v}_4$ . Note that this solution  $\mathbf{x}_{svd} = [0.1, 0.2, 0.3, 0.4]^T$  is actually the first component  $\mathbf{x}'_p \in R(\mathbf{A})$  of the particular solution  $\mathbf{x}_p = [-1, 2, 0, 0]$  found in the previous section by Gauss-Jordan elimination, which is not in  $R(\mathbf{A})$ . Adding the null space to the particular solution, we get the complete solution:

$$\mathbf{x}_c = \mathbf{x}_{svd} + N(\mathbf{A}) = \mathbf{x}_p + c_1\mathbf{v}_3 + c_2\mathbf{v}_4 = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} + c_1 \begin{bmatrix} 0.374 \\ -0.797 \\ 0.472 \\ -0.049 \end{bmatrix} + c_2 \begin{bmatrix} -0.400 \\ 0.255 \\ 0.691 \\ -0.546 \end{bmatrix} \quad (\text{A.379})$$

If the right-hand side is replaced by  $\mathbf{b} = [1, 3, 5]^T \notin C(\mathbf{A})$ , no solution exists. However, we can still find the pseudo-inverse solution as the optimal approximate

solution:

$$\mathbf{x}_{svd} = \mathbf{A}^{-} \mathbf{b} = \begin{bmatrix} 0.033 & 0.133 & -0.067 \\ 0.033 & 0.083 & -0.017 \\ 0.033 & 0.033 & 0.033 \\ 0.033 & -0.017 & 0.083 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} \quad (\text{A.380})$$

which is the same as the solution for  $\mathbf{b} = [3, 2, 4]^T$ , indicating  $[3, 2, 4]^T$  happens to be the projection of  $[1, 3, 5]^T$  onto  $C(\mathbf{A})$  spanned by  $\mathbf{u}_1$  and  $\mathbf{u}_2$ . The result produced by  $\mathbf{x}_{svd}$  is  $\mathbf{b}' = \mathbf{A}\mathbf{x}_{svd} \in C(\mathbf{A})$  is the projection of  $\mathbf{b}$  onto  $C(\mathbf{A})$ , with a minimum error distance  $\|\mathbf{e}\| = \|\mathbf{b} - \mathbf{b}'\|$ , indicating  $\mathbf{x}_{svd}$  is the optimal approximate solution.

### Homework 3:

1. Use Matlab function `svd(A)` to carry out the SVD of the coefficient matrix  $\mathbf{A}$  of the linear system in Homework 2 problem 3,

$$\mathbf{A} = \begin{bmatrix} -1 & 3 & 4 & 1 \\ 2 & -4 & 3 & 2 \\ 1 & -1 & 7 & 3 \end{bmatrix} \quad (\text{A.381})$$

Find  $\mathbf{U}$ ,  $\mathbf{V}$ ,  $\mathbf{\Sigma}$ . Verify that  $\mathbf{U}$  and  $\mathbf{V}$  are orthogonal, i.e.,  $\mathbf{U}^T\mathbf{U} = \mathbf{I}$  and  $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ .

2. Obtain  $\mathbf{\Sigma}^{-}$ , i.e, find the transpose of  $\mathbf{\Sigma}$  and then replace each singular value by its reciprocal, verify that  $\mathbf{\Sigma}\mathbf{\Sigma}^{-} = \mathbf{I}$  and  $\mathbf{\Sigma}^{-}\mathbf{\Sigma} = \mathbf{I}$ . Then find the pseudo-inverse  $\mathbf{A}^{-} = \mathbf{U}\mathbf{\Sigma}^{-}\mathbf{V}^T$ .
3. Use Matlab function `pinv(A)` to find the pseudo-inverse  $\mathbf{\Sigma}^{-}$  and  $\mathbf{A}^{-}$ . Compare them to what you obtained in the previous part.
4. Identify the bases of the four subspaces  $R(\mathbf{A})$ ,  $N(\mathbf{A})$ ,  $C(\mathbf{A}) = R(\mathbf{A}^T)$ , and  $N(\mathbf{A}^T)$  based on  $\mathbf{U}$  and  $\mathbf{V}$ . Verify that these bases and those you found previously (problem 3 of Homework 2) span the same spaces, i.e., the basis vectors of one basis can be written as a linear combination of those of the other, for each of the four subspaces.
5. Use the pseudo-inverse  $\mathbf{A}^{-}$  found above to solve the system  $\mathbf{A}\mathbf{x} = \mathbf{b}$ . Find the two particular solutions  $\mathbf{x}_{p1}$  and  $\mathbf{x}_{p2}$  corresponding to two different right-hand side  $\mathbf{b}_1 = [1, 2, 3]^T$  and  $\mathbf{b}_2 = [2, 3, 2]^T$ .
6. How are the two results  $\mathbf{x}_{p1}$  and  $\mathbf{x}_{p2}$  related to each other? Give your explanation. Why can't you find a solution when  $\mathbf{b} = [2, 3, 2]^T$  but SVD method can?
7. Verify that  $\mathbf{x}_p \perp N(\mathbf{A})$ . Also find the error (or residual)  $\mathbf{e} = \mathbf{b} - \mathbf{A}\mathbf{x}_p$  for each of the two results above. Verify that  $\mathbf{e} \perp C(\mathbf{A})$ , if  $\mathbf{e} \neq 0$ .
8. In Homework 2 you used row reduction method to solve the system  $\mathbf{A}\mathbf{x} = \mathbf{b}_1$  and you should have found a particular solution  $\mathbf{x}_1 = [5, 2, 0, 0]^T$ . Also it is obvious to see that another solution is  $\mathbf{x}_2 = [0, 0, 0, 1]^T$ . Show that the projection of these solutions onto  $R(\mathbf{A})$  spanned by the first two columns of  $\mathbf{V}$  is the same as the particular solution  $\mathbf{x}_p$  found by the SVD method.



9. Show that  $\mathbf{b}_1 = [1, 2, 3]^T$  is the projection of  $\mathbf{b}_2 = [2, 3, 2]^T$  onto  $C(\mathbf{A})$ , the 2-D subspace spanned by the first two columns of  $\mathbf{U}$ . Can you also show that it is the projection of  $\mathbf{b}$  onto  $C(\mathbf{A})$  spanned by the basis you obtained in Homework 2 (not necessarily orthogonal)?
10. Give an expression of the null space  $N(\mathbf{A})$ , and then write the complete solution in form of  $\mathbf{x}_c = \mathbf{x}_p + N(\mathbf{A})$ . Verify any complete solution so generated satisfies  $\mathbf{A}\mathbf{x}_c = \mathbf{b}_1$ .

## A.21 Rank-Nullity Theorem

**Definition:** The number of independent rows in a matrix is called the row rank, the number of independent columns in a matrix is called the column rank.

**Theorem:** The row rank and column rank of any matrix are the same.

**Proof:** Let the column rank of the following  $m \times n$  matrix  $\mathbf{A}$  be  $r$

$$\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_r, \mathbf{a}_{r+1}, \dots, \mathbf{a}_n] = [\mathbf{A}_1, \mathbf{A}_0] \quad (\text{A.382})$$

where  $\mathbf{A}_1 = [\mathbf{a}_1, \dots, \mathbf{a}_r]$  is an  $m \times r$  sub-matrix formed by the  $r$  linearly independent columns in  $\mathbf{A}$ , assumed to be the first  $r$  columns (without loss of generality). Now consider the linear combination of another set of  $r$  n-D vectors  $\{\mathbf{A}_1^T \mathbf{a}_1, \dots, \mathbf{A}_1^T \mathbf{a}_r\}$ :

$$\sum_{i=1}^r c_i \mathbf{A}_1^T \mathbf{a}_i = \mathbf{A}_1^T \sum_{i=1}^r c_i \mathbf{a}_i = \mathbf{A}_1^T \mathbf{A}_1 \mathbf{c} = \mathbf{0} \quad (\text{A.383})$$

which is assumed to be zero. As the  $r \times r$  coefficient matrix  $\mathbf{A}_1^T \mathbf{A}_1$  has full rank  $r$  ( $\mathbf{A}_1$  is formed by  $r$  independent vectors), we must have  $\mathbf{c} = [c_1, \dots, c_r]^T = \mathbf{0}$ , indicating  $\{\mathbf{A}_1^T \mathbf{a}_1, \dots, \mathbf{A}_1^T \mathbf{a}_r\}$  are independent. However, as the linear combinations of  $r$  of the rows of  $\mathbf{A}$ , these  $r$  independent vector  $\mathbf{A}_1^T \mathbf{a}_i$  ( $i = 1, \dots, r$ ) are in the row space of  $\mathbf{A}$ . We therefore know that the row rank of  $\mathbf{A}$  is no smaller than its column rank.

As the same argument can also be made for  $\mathbf{A}^T$ , i.e., the column rank of  $\mathbf{A}$  is no smaller than its row rank, we conclude that the column and row ranks of  $\mathbf{A}$  must be the same. QED

**Rank-nullity theorem:** The sum of the rank of an  $m \times n$  matrix  $\mathbf{A}$  and the dimension of its nullspace  $N(\mathbf{A})$ , the nullity of  $\mathbf{A}$ , is  $n$ :

$$\text{rank}(\mathbf{A}) + \dim(N(\mathbf{A})) = \text{rank}(\mathbf{A}) + \text{Nullity}(\mathbf{A}) = n \quad (\text{A.384})$$

**Proof:**

Let  $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_r, \mathbf{a}_{r+1}, \dots, \mathbf{a}_n]$  be an  $m \times n$  matrix of rank  $r$ . We assume, without loss of generality, the first  $r$  columns are independent, and the last  $n - r$  columns are dependent, and write it as  $\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_0]$ . Here

$$\mathbf{A}_1 = [\mathbf{a}_1, \dots, \mathbf{a}_r]_{m \times r} \quad (\text{A.385})$$

is formed by the  $r$  independent columns of  $\mathbf{A}$ , i.e., if  $\mathbf{A}_1\mathbf{x} = \mathbf{0}$ , then  $\mathbf{x} = \mathbf{0}$ ; and

$$\mathbf{A}_0 = [\mathbf{a}_{r+1}, \dots, \mathbf{a}_n]_{m \times (n-r)} \quad (\text{A.386})$$

is formed by the  $n-r$  dependent columns of  $\mathbf{A}$ , i.e., any of them can be written as a linear combination of those independent columns  $\mathbf{a}_1, \dots, \mathbf{a}_r$  in  $\mathbf{A}_1$ :

$$\mathbf{a}_{r+k} = \sum_{i=1}^r c_{ik} \mathbf{a}_i = [\mathbf{a}_1, \dots, \mathbf{a}_r] \begin{bmatrix} c_{1k} \\ \vdots \\ c_{rk} \end{bmatrix} = \mathbf{A}_1 \mathbf{c}_k, \quad (k = 1, \dots, n-r) \quad (\text{A.387})$$

where  $\mathbf{c}_k = [c_{1k}, \dots, c_{rk}]^T$  is a coefficient vector. We can put all these  $n-r$  equations together to write them in matrix form as

$$\mathbf{A}_0 = [\mathbf{a}_{r+1}, \dots, \mathbf{a}_n] = \mathbf{A}_1 [\mathbf{c}_1, \dots, \mathbf{c}_{n-r}] = \mathbf{A}_1 \mathbf{C} \quad (\text{A.388})$$

where  $\mathbf{C} = [\mathbf{c}_1, \dots, \mathbf{c}_{n-r}]$  is an  $r \times (n-r)$  matrix composed of the  $n-r$  coefficient vectors. The matrix  $\mathbf{A}$  can now be written as

$$\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_0] = [\mathbf{A}_1, \mathbf{A}_1 \mathbf{C}] \quad (\text{A.389})$$

Define an  $n \times (n-r)$  matrix

$$\mathbf{X} = \begin{bmatrix} -\mathbf{C} \\ \mathbf{I} \end{bmatrix} = \begin{bmatrix} -\mathbf{c}_1 & \cdots & -\mathbf{c}_{n-r} \\ \mathbf{e}_1 & \cdots & \mathbf{e}_n \end{bmatrix} = \begin{bmatrix} -\mathbf{c}_1 & \cdots & -\mathbf{c}_{n-r} \\ 1 & \cdots & 0 \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ 0 & \cdots & 1 \end{bmatrix} \quad (\text{A.390})$$

where  $\mathbf{I}$  is an  $(n-r) \times (n-r)$  identity matrix composed of  $n$  standard basis vectors  $\mathbf{e}_1, \dots, \mathbf{e}_n$ . Now we have

$$\mathbf{A}\mathbf{X} = [\mathbf{A}_1, \mathbf{A}_1 \mathbf{C}] \begin{bmatrix} -\mathbf{C} \\ \mathbf{I} \end{bmatrix} = -\mathbf{A}_1 \mathbf{C} + \mathbf{A}_1 \mathbf{C} = \mathbf{0} \quad (\text{A.391})$$

indicating any column  $\mathbf{x}$  in  $\mathbf{X}$  is a solution of the homogeneous equation  $\mathbf{A}\mathbf{x} = \mathbf{0}$ , i.e.,  $\mathbf{x}$  is in the nullspace  $N(\mathbf{A})$ . Also, if we let  $\mathbf{X}\mathbf{u} = \mathbf{0}$ , we get

$$\mathbf{X}\mathbf{u} = \begin{bmatrix} -\mathbf{C} \\ \mathbf{I} \end{bmatrix} \mathbf{u} = \begin{bmatrix} -\mathbf{C}\mathbf{u} \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} = \mathbf{0} \quad (\text{A.392})$$

i.e.,  $\mathbf{u} = \mathbf{0}$ , indicating all columns in  $\mathbf{X}$  are independent.

On the other hand, we can also show that any solution of the homogeneous equation  $\mathbf{A}\mathbf{x} = \mathbf{0}$  must be a linear combination of the columns of  $\mathbf{X}$ . Assuming  $\mathbf{x}$  is such a solution, i.e.,

$$\mathbf{A}\mathbf{x} = \mathbf{0} = [\mathbf{A}_1, \mathbf{A}_1 \mathbf{C}] \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \mathbf{A}_1 (\mathbf{x}_1 + \mathbf{C}\mathbf{x}_2) \quad (\text{A.393})$$

As all columns in  $\mathbf{A}$  are independent, we must have

$$\mathbf{x}_1 + \mathbf{C}\mathbf{x}_2 = \mathbf{0}, \quad \text{or} \quad \mathbf{x}_1 = -\mathbf{C}\mathbf{x}_2 \quad (\text{A.394})$$

and we get

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} -\mathbf{C}\mathbf{x}_2 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} -\mathbf{C} \\ \mathbf{I} \end{bmatrix} \mathbf{x}_2 = \mathbf{X}\mathbf{x}_2 \quad (\text{A.395})$$

indicating  $\mathbf{x}$  is a linear combination of the  $n - r$  columns in  $\mathbf{X}$ .

Based on the two results obtained above, we can now conclude that the nullspace  $N(\mathbf{A})$  is spanned by the  $n - r$  independent column vectors in  $\mathbf{X}$ , i.e., the dimension of  $N(\mathbf{A})$  is  $n - r$ , and  $\text{rank}(\mathbf{A}) + \text{dim}(\mathbf{A}) = n$ .

## A.22 Taylor Series Expansion

- **Single-variable functions:**

A single-variable function  $f(x)$  can be expanded around a given point  $x$  by the Taylor series:

$$f(x + \delta x) = f(x) + f'(x)\delta x + \frac{1}{2!}f''(x)\delta x^2 + \frac{1}{3!}f'''(x)\delta x^3 + \dots \quad (\text{A.396})$$

When  $\delta x$  is small, the higher order terms can be neglected so that the function can be approximated as a quadratic function

$$f(x + \delta x) = f(x) + f'(x)\delta x + \frac{1}{2!}f''(x)\delta x^2 + \varepsilon(\delta x^3) \approx f(x) + f'(x)\delta x + \frac{1}{2!}f''(x)\delta x^2 \quad (\text{A.397})$$

or even a linear function

$$f(x + \delta x) = f(x) + f'(x)\delta x + \varepsilon(\delta x^2) \approx f(x) + f'(x)\delta x \quad (\text{A.398})$$

- **Multi-variable scalar-valued functions:**

A multi-variable function  $f(x_1, \dots, x_n) = f(\mathbf{x})$  can also be expanded by the Taylor series:

$$f(\mathbf{x} + \delta \mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^n \frac{\partial f(\mathbf{x})}{\partial x_j} \delta x_j + \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \delta x_i \delta x_j + \frac{1}{3!} \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \frac{\partial^3 f(\mathbf{x})}{\partial x_i \partial x_j \partial x_k} \delta x_i \delta x_j \delta x_k + \dots \quad (\text{A.399})$$

which can be expressed in vector form as:

$$f(\mathbf{x} + \delta \mathbf{x}) = f(\mathbf{x}) + \mathbf{g}_f^T \delta \mathbf{x} + \frac{1}{2} \delta \mathbf{x}^T \mathbf{H}_f \delta \mathbf{x} + \dots \quad (\text{A.400})$$

where  $\delta \mathbf{x} = [\delta x_1, \dots, \delta x_n]^T$ , while  $\mathbf{g}_f$  and  $\mathbf{H}_f$  are the gradient vector and the *Hessian* matrix for the first and second order derivatives of the function,

respectively:

$$\mathbf{g}_f(\mathbf{x}) = \nabla f(\mathbf{x}) = \frac{d}{d\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}_{n \times 1}, \quad (\text{A.401})$$

$$\mathbf{H}_f(\mathbf{x}) = \frac{d}{d\mathbf{x}} \mathbf{g}_f(\mathbf{x}) = \frac{d^2}{d\mathbf{x}^2} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix}_{n \times n} = \mathbf{H}_f^T \quad (\text{A.402})$$

Note that  $\mathbf{H}_f^T$  is symmetric as  $\partial^2 f(\mathbf{x})/\partial x_i \partial x_j = \partial^2 f(\mathbf{x})/\partial x_j \partial x_i$ .

When  $\delta\mathbf{x}$  is small (i.e.,  $\|\delta\mathbf{x}\|$  is small), then  $f(\mathbf{x}+\delta\mathbf{x})$  can be approximated by a quadratic function with a third order error term

$$f(\mathbf{x} + \delta\mathbf{x}) = f(\mathbf{x}) + \mathbf{g}_f^T \delta\mathbf{x} + \frac{1}{2} \delta\mathbf{x}^T \mathbf{H}_f \delta\mathbf{x} + \varepsilon(\|\delta\mathbf{x}\|^3) \quad (\text{A.403})$$

or even a linear function with a second order error term

$$f(\mathbf{x} + \delta\mathbf{x}) = f(\mathbf{x}) + \mathbf{g}_f^T \delta\mathbf{x} + \varepsilon(\|\delta\mathbf{x}\|^2) \quad (\text{A.404})$$

- **Multi-variable vector-valued functions:**

A set of  $m$  multi-variable functions  $f_1(\mathbf{x}), \dots, f_m(\mathbf{x})$  can be expressed as a vector function

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), \dots, f_m(\mathbf{x})]^T \quad (\text{A.405})$$

The Taylor expansion of the  $i$ th component is:

$$f_i(\mathbf{x} + \delta\mathbf{x}) = f_i(\mathbf{x}) + \mathbf{g}_i^T \delta\mathbf{x} + \frac{1}{2} \delta\mathbf{x}^T \mathbf{H}_i \delta\mathbf{x} + \varepsilon(\|\delta\mathbf{x}\|^3) \quad (i = 1, \dots, m) \quad (\text{A.406})$$

The first two terms of these  $m$  components can be written in vector form:

$$\mathbf{f}(\mathbf{x} + \delta\mathbf{x}) \approx \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \vdots \\ \delta x_n \end{bmatrix} = \mathbf{f}(\mathbf{x}) + \mathbf{J}_f(\mathbf{x}) \delta\mathbf{x} \quad (\text{A.407})$$

where  $\mathbf{J}_f(\mathbf{x})$  is the *Jacobian matrix* defined over the vector function  $\mathbf{f}(\mathbf{x})$ :

$$\mathbf{J}_f(\mathbf{x}) = \frac{d}{d\mathbf{x}} \mathbf{f}(\mathbf{x}) = \left[ \frac{\partial \mathbf{f}}{\partial x_1}, \dots, \frac{\partial \mathbf{f}}{\partial x_n} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}_{m \times n} \quad (\text{A.408})$$

Note that the Jacobian matrix is not necessarily square as  $m$  and  $n$  may not be the same. The 2nd and higher order terms can no longer be expressed in matrix.

Note that the Hessian matrix of a function  $f(\mathbf{x})$  can be obtained as the Jacobian matrix of the gradient vector of  $f(\mathbf{x})$ :

$$\mathbf{g}_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}, \quad \mathbf{J}_{\mathbf{g}}(\mathbf{x}) = \frac{d}{d\mathbf{x}} \mathbf{g}_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} = \mathbf{H}_f(\mathbf{x}) \quad (\text{A.409})$$

## A.23 Normal Direction of a Plane

In a  $d$ -dimensional space,

- A point is represented as a column vector  $\mathbf{x} = [x_1, \dots, x_d]^T$  containing its  $d$  coordinates;
- A surface (or supersurface if  $d > 3$ ) is represented by an equation  $f(x_1, \dots, x_d) = f(\mathbf{x}) = 0$ . Alternatively, by solving the equation above for one of the variables, we get  $x_d = g(x_1, \dots, x_{d-1})$ , representing the height along the direction of  $x_d$  of a surface defined over a  $d - 1$  dimensional space.
- A plane (or superplane if  $d > 3$ ) is a special surface described by a linear equation:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^d w_i x_i + b = 0 \quad (\text{A.410})$$

Without loss of generality, we can assume  $\mathbf{w}$  is normalized with length  $\|\mathbf{w}\| = \sqrt{\mathbf{w}^T \mathbf{w}} = 1$  (as otherwise we can divide the equation above by  $\|\mathbf{w}\|$ ).

- The  $d$ -dimensional space is divided by the plane into two regions:

$$R_+ = \{\mathbf{x} \mid \mathbf{w}^T \mathbf{x} + b > 0\}, \quad \text{and} \quad R_- = \{\mathbf{x} \mid \mathbf{w}^T \mathbf{x} + b < 0\} \quad (\text{A.411})$$

- The *normal direction* of a plane  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$  is  $\mathbf{w}$ , i.e., the projection of any point  $\mathbf{x}$  on the plane onto normal direction  $\mathbf{w}$  is constant:

$$\mathbf{p}_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} = -b \quad (\text{A.412})$$

- Solving the linear equation  $\mathbf{w}^T \mathbf{x} + b = 0$  for any of the  $d$  variables, e.g.,  $x_d$ , we get an alternative representation for the plane, as a function defined over the  $d-1$  dimensional space spanned by the remaining  $d - 1$  variables  $\{x_1, \dots, x_{d-1}\}$ :

$$x_d = g(x_1, \dots, x_{d-1}) = -\frac{1}{w_d} \left( \sum_{i=1}^{d-1} w_i x_i + b \right) \quad (\text{A.413})$$

In particular, when  $x_1 = \dots = x_{d-1} = 0$ , we get  $x_d = -b/w_d$ , the intercept of the plane on the axis of  $x_d$ .

- When the plane is represented in the form of  $x_d = g(x_1, \dots, x_{d-1}) = -(\sum_{i=1}^{d-1} w_i x_i + b)/w_d$ , the normal direction is represented the following, a scaled version of  $\mathbf{w}$ :

$$\left[-\frac{w_1}{w_d}, \dots, -\frac{w_{d-1}}{w_d}, -1\right]^T = -\frac{1}{w_d}[w_1, \dots, w_d]^T = -\frac{1}{w_d}\mathbf{w} \quad (\text{A.414})$$

**Example:** The following equation defines a plane in 3-D space spanned by  $x$ ,  $y$  and  $z$ :

$$f(x, y, z) = ax + by + cz + d = x + 2y + 3z + 4 = 0$$

where  $a = 1$ ,  $b = 2$ ,  $c = 3$  and  $d = 4$ . Solving for  $z$  we get

$$z = g(x, y) = -\frac{1}{c}(ax + by + d) = -\frac{1}{3}(x + 2y + 4)$$

When  $x = y = 0$ , we get  $z = -c/d = -4/3$ , the intercept of the plane on  $z$ -axis. The normal direction of the plane is composed of the coefficients of the three variable  $x$ ,  $y$  and  $z$  in  $f(x, y, z)$ :

$$\mathbf{n} = [a, b, c]^T = [1, 2, 3]^T \quad (\text{A.415})$$

or the coefficients of the two variables  $x$  and  $y$  of  $g(x, y)$ , with the last component  $-1$ :

$$\mathbf{n} = \left[-\frac{a}{c}, -\frac{b}{c}, -1\right]^T = -\frac{1}{c}[a, b, c]^T = -\frac{1}{3}[1, 2, 3] \quad (\text{A.416})$$

## A.24 Convexity

We first consider some concepts regarding convexity:

- *Convex set* or *convex region*: a region in an N-D space in which every point on the straight line between any pair of two points in the region is also inside the region.
- *Convex hull* or *convex envelope*, *convex closure* of a shape: the smallest convex set containing the shape.
- *Convex function*: a real-valued function  $f(\mathbf{x})$  with the property that its *epi-graph* or *supergraph*, the region above its graph, is a convex set.
- *Concave function*: if  $f(\mathbf{x})$  is convex, then  $-f(\mathbf{x})$  is concave.

A multivariate function  $f(\mathbf{x})$  is convex if and only if

$$f(\lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1 - \lambda)f(\mathbf{x}_2) \quad (\text{A.417})$$

Equivalently,  $f(\mathbf{x})$  is concave if and only if

$$f(\lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2) \geq \lambda f(\mathbf{x}_1) + (1 - \lambda)f(\mathbf{x}_2) \quad (\text{A.418})$$

The convex function is *strict* if equality in the relationship never holds. Geometrically,

the line segment connecting two points  $(\mathbf{x}_1, f(\mathbf{x}_1))$  and  $(\mathbf{x}_2, f(\mathbf{x}_2))$  is above the graph of the convex function  $f(\mathbf{x})$ .

Convexity plays an important role in optimization problems, due to its nice property that if a function  $f(\mathbf{x})$  is differentiable and strictly convex, then any point  $\mathbf{x}^*$  with a zero gradient  $\mathbf{g}_f(\mathbf{x}^*) = \mathbf{0}$  is the unique global minimum of the function. Minimizing a convex objective function, or equivalently maximizing a concave objective function, over a convex domain is called *convex optimization*.

Here are some examples and properties of convex functions:

- Single-variable functions:  $x^2, |x|, e^{ax}, -\log(x)$
- Any norm of a function or vector is convex. The intersection of the epigraphs is convex.
- The intersection of two convex sets is convex. Also,
- Affine function:  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$  is both convex and concave but not strict, as both Eq. (A.417) and Eq. (A.418) become equality when applied to the function.
- If function  $f(\mathbf{x})$  is convex, then  $a f(\mathbf{x}) + b$  is also convex, if  $a \geq 0$  so that the convexity is preserved.
- If two functions  $f(\mathbf{x})$  and  $g(\mathbf{x})$  are both convex, then their linear combination is also convex if the coefficients are nonnegative:

$$h(\mathbf{x}) = af(\mathbf{x}) + bg(\mathbf{x}), \quad a, b \geq 0 \tag{A.419}$$

This can be proven by applying the definition directly to  $h(\mathbf{x})$ .

- A linear combination of  $n$  convex function is also convex if all coefficients are nonnegative:

$$f(\mathbf{x}) = \sum_{i=1}^n w_i f_i(\mathbf{x}), \quad w_1, \dots, w_n \geq 0 \tag{A.420}$$

- Quadratic functions:  $f(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{w}^T \mathbf{x} + b$  is
  - convex if  $\mathbf{Q} \geq 0$  is positive semidefinite;
  - strictly convex if  $\mathbf{Q} > 0$  is positive definite;
  - concave if  $\mathbf{Q} \leq 0$  is negative semidefinite;
  - strictly concave if  $\mathbf{Q} < 0$  is negative definite;

If the Hessian matrix  $\mathbf{H}_f(\mathbf{x})$ , the second order derivative, of  $f(\mathbf{x})$  exists and is positive semidefinite, i.e.,  $\mathbf{H}_f(\mathbf{x}) \geq 0$  for its entire domain, then  $f(\mathbf{x})$  is convex. As simple examples in 1-D space,  $f_1(x) = x^2$  and  $f_2(x) = e^x$  are both convex, and their Hessian, the second order derivatives,  $f_1''(x) = 2$  and  $f_2''(x) = e^x$  are both greater than zero.

Specifically, let  $\mathbf{x}_1$  and  $\mathbf{x}_2$  be any two points inside the domain of  $f(\mathbf{x})$ . Then  $f(\mathbf{x})$  is convex if for any  $c \in [0, 1]$  the following holds:

$$f(c\mathbf{x}_1 + (1 - c)\mathbf{x}_2) \leq cf(\mathbf{x}_1) + (1 - c)f(\mathbf{x}_2) \tag{A.421}$$

where  $c\mathbf{x}_1 + (1 - c)\mathbf{x}_2$  is any point along the line segment between  $\mathbf{x}_1$  and  $\mathbf{x}_2$ , at which the function value is less than the linear interpolation of the two function

values  $f(\mathbf{x}_1)$  and  $f(\mathbf{x}_2)$ . If the equality does not hold, then  $f(\mathbf{x})$  is *strictly convex*. We can also define a *concave function* as the negative of a convex function. If  $f(\mathbf{x})$  is convex, then  $g(\mathbf{x}) = -f(\mathbf{x})$  is *concave*, for which the following inequality holds:

$$g(c\mathbf{x}_1 + (1-c)\mathbf{x}_2) \geq cg(\mathbf{x}_1) + (1-c)g(\mathbf{x}_2) \quad (\text{A.422})$$

A convex function  $f(\mathbf{x})$  has the property that any local minimum is also a global minimum. Moreover, if  $f(\mathbf{x})$  is strictly convex, then it has a single global minimum.

## A.25 Jensen Inequality

For a convex function  $g(\mathbf{x})$ , the following holds:

$$g\left(\sum_{i=1}^n \lambda_i \mathbf{x}_i\right) \leq \sum_{i=1}^n \lambda_i f(\mathbf{x}_i) \quad (\text{A.423})$$

For a concave function  $g(\mathbf{x})$ , the following holds:

$$g\left(\sum_{i=1}^n \lambda_i \mathbf{x}_i\right) \geq \sum_{i=1}^n \lambda_i g(\mathbf{x}_i) \quad (\text{A.424})$$

### Proof:

Let  $\lambda_1$  and  $\lambda_2$  be two arbitrary nonnegative real numbers satisfying  $\lambda_1 + \lambda_2 = 1$ . Then by the definition of convexity, we have

$$f(\lambda_1 \mathbf{x}_1 + \lambda_2 \mathbf{x}_2) \leq \lambda_1 f(\mathbf{x}_1) + \lambda_2 f(\mathbf{x}_2) \quad (\text{A.425})$$

This results of two points can be generalized to  $n$  points by induction. We assume the inequality holds for  $k$  points:

$$f\left(\sum_{i=1}^k \lambda_i \mathbf{x}_i\right) \leq \sum_{i=1}^k \lambda_i f(\mathbf{x}_i) \quad (\text{A.426})$$

Now consider the case of  $k+1$  terms with coefficients satisfying  $\lambda_1 + \dots + \lambda_{k+1} = 1$ :

$$\begin{aligned} f\left(\sum_{i=1}^{k+1} \lambda_i \mathbf{x}_i\right) &= f\left(\lambda_1 \mathbf{x}_1 + (1-\lambda_1) \sum_{i=2}^{k+1} \frac{\lambda_i}{1-\lambda_1} \mathbf{x}_i\right) \\ &\leq \lambda_1 f(\mathbf{x}_1) + (1-\lambda_1) f\left(\sum_{i=2}^{k+1} \frac{\lambda_i}{1-\lambda_1} \mathbf{x}_i\right) \end{aligned} \quad (\text{A.427})$$

But as

$$\sum_{i=2}^{k+1} \frac{\lambda_i}{1-\lambda_1} = 1 \quad (\text{A.428})$$



we have the following due to the assumption above:

$$f\left(\sum_{i=2}^{k+1} \frac{\lambda_i}{1-\lambda_1} \mathbf{x}_i\right) \leq \sum_{i=2}^{k+1} \frac{\lambda_i}{1-\lambda_1} f(\mathbf{x}_i) \quad (\text{A.429})$$

Substituting this back we get the desired inequality:

$$f\left(\sum_{i=1}^{k+1} \lambda_i \mathbf{x}_i\right) \leq \lambda_1 f(\mathbf{x}_1) + (1-\lambda_1) \left(\sum_{i=2}^{k+1} \frac{\lambda_i}{1-\lambda_1} f(\mathbf{x}_i)\right) = \sum_{i=1}^{k+1} \lambda_i f(\mathbf{x}_i) \quad (\text{A.430})$$

If  $\mathbf{x}$  is a random vector that takes any of the  $n$  values  $\mathbf{x}_i$  with probability  $P_i$  satisfying  $P_1 + \dots + P_n = 1$ , then we have

$$f(E(\mathbf{x})) = f\left(\sum_{i=1}^k \mathbf{x}_i P_i\right) \leq \sum_{i=1}^k f(\mathbf{x}_i) P_i = E(f(\mathbf{x})) \quad (\text{A.431})$$

This result can be generalized to the case where  $\mathbf{x}$  is a random vector taking continuous values with  $E[\mathbf{x}] = \int \mathbf{x} p(\mathbf{x}) d\mathbf{x}$ .

## A.26 Sensitivity and Conditioning

An important concern in numerical analysis is how sensitive the result produced by an algorithm is with respect to the noise in the input data (e.g., observational error) and computational errors (e.g., truncation error). To address this issue, we model a generic algorithm treated as a function with both input and output most generally defined (e.g., real or complex, scalar, vector, matrix, function, etc.), and describe its sensitivity to noise as below:

- A function is *well conditioned* (*well behaved*) if a small change in the input causes a small change in the output.
- A function is *ill conditioned* (*ill behaved*) if a small change in the input causes a large change in the output.

How well or ill a function is conditioned can be measured quantitatively by the absolute or relative *condition numbers* defined below:

- The absolute condition number is the upper bound of the ratio between the change in output and change in input:

$$\hat{\kappa} \geq \frac{\|\text{change in output}\|}{\|\text{change in input}\|} \quad (\text{A.432})$$

- The *relative condition number* is the upper bound of the ratio between the relative change in output and relative change in input:

$$\kappa \geq \frac{\|\text{change in output}\|/\|\text{output}\|}{\|\text{change in input}\|/\|\text{input}\|} \quad (\text{A.433})$$

where  $\|x\|$  denotes the *norm* of  $x$ . As the relative condition number compares the normalized changes in both input and output, its value is invariant to the specific units used to measure them. Obviously the smaller the condition number, the better the function is conditioned.

Consider specifically the condition numbers of some different systems.

**System with Scalar input and output:**

We first consider evaluating a scalar function  $y = f(x)$  given  $x$ . The condition number is:

$$\hat{\kappa} = \lim_{\epsilon \rightarrow 0} \sup_{|\delta x| \leq \epsilon} \frac{|\delta f(x)|}{|\delta x|}, \quad \kappa = \lim_{\epsilon \rightarrow 0} \sup_{|\delta x| \leq \epsilon} \frac{|\delta f(x)|/|f(x)|}{|\delta x|/|x|}, \quad (\text{A.434})$$

Here sup is the symbol for *supremum* (least upper bound) and  $|x|$  is either the absolute value of  $x$  if it is real or the modulus of  $x$  if it is complex. Note that in general the condition number is a function of  $x$ .

When  $\delta x$  is small,  $f(x + \delta x)$  can be approximated by the first two terms of its Taylor series:

$$f(x + \delta x) = f(x) + f'(x)\delta x + f''(x)\frac{\delta x^2}{2} + \dots \approx f(x) + f'(x)\delta x \quad (\text{A.435})$$

then

$$\delta y = f(x + \delta x) - f(x) \approx f'(x)\delta x \quad (\text{A.436})$$

Taking the absolute value or modulus on both sides, we get

$$|\delta y| \approx |f'(x)\delta x| \leq |f'(x)| |\delta x| \quad (\text{A.437})$$

and the absolute condition number is:

$$\hat{\kappa} \approx |\delta y|/|\delta x| \approx |f'(x)| \quad (\text{A.438})$$

Dividing both sides by  $|y| = |f(x)|$ , we further get

$$\frac{|\delta y|}{|y|} \approx \frac{|f'(x)| |\delta x|}{|f(x)|} = \frac{|x| |f'(x)|}{|f(x)|} \frac{|\delta x|}{|x|} \quad (\text{A.439})$$

and the relative condition number:

$$\kappa = \frac{|\delta y|/|y|}{|\delta x|/|x|} \approx \frac{|x| |f'(x)|}{|f(x)|} = \frac{|f'(x)|}{|f(x)|/|x|} \quad (\text{A.440})$$

We next consider solving the equation  $f(x) = 0$  to find  $x$  treated as the output that satisfies  $y = f(x) = 0$  treated as the input. The problem can be converted into the evaluation of the function  $x = g(y) = f^{-1}(y)$  at  $y = 0$ . The absolute condition number of this function  $g(y) = f^{-1}(y)$  is:

$$\hat{\kappa} = |g'(y)| = \left| \frac{d}{dy} [f^{-1}(y)] \right| = \frac{1}{|f'(x)|} \quad (\text{A.441})$$

which is the reciprocal of the absolute condition number  $|f'(x)|$  of the previous evaluating problem, evaluated at the root  $x^*$  satisfying  $f(x^*) = 0$ . We see that

the larger  $|f'(x)|$ , or the smaller  $g'(y) = 1/|f'(x)|$ , the better conditioned the problem is.

In the figure below shows two functions  $|f'_1(x)| > |f'_2(x)|$  for all  $x$ . For evaluating function  $y = f(x)$ ,  $f_1(x)$  is more ill-conditioned than  $f_2(x)$ , but for solving  $f(x) = 0$ ,  $f_1(x)$  is more well-conditioned than  $f_2(x)$ .

Consider as an example the following two problems both associated with a linear function  $y = f(x) = ax + b$ :

- Evaluation of  $y$  given  $x$ :

The condition number is  $\kappa = \delta y / \delta x$ . If  $x$  is changed to  $x + \delta x$  due to noise, then  $y$  becomes  $ax + b + a\delta x = y + \delta y$ , where  $\delta y = a\delta x$ , i.e.,  $\kappa = a$ .

The larger  $a$ , the more ill-conditioned the system is.

- Solving  $y = f(x) = 0$  for  $x$  given  $y = 0$ :

The condition number is  $\kappa = \delta x / \delta y$ . The given equation can be solved for  $x$ :

$$x = g(y) = \frac{1}{a}(y - b) \quad (\text{A.442})$$

If  $y$  is changed to  $y + \delta y$  due to noise, then  $x$  becomes  $(y - b)/a + \delta y/a = x + \delta x$ , where  $\delta x = \delta y/a$ , i.e.,  $\kappa = 1/a$ . The smaller  $a$ , the more ill-conditioned the system is.

### Systems with vector input and output

The discussions above can be generalized to a function  $\mathbf{y} = \mathbf{f}(\mathbf{x})$  with vector input  $\mathbf{x} = [x_1, \dots, x_N]^T$  and vector output  $\mathbf{y} = [y_1, \dots, y_M]^T$ . Again consider approximating the function by the first two terms in the Taylor series:

$$\mathbf{f}(\mathbf{x} + \delta\mathbf{x}) \approx \mathbf{f}(\mathbf{x}) + \mathbf{J}_f(\mathbf{x})\delta\mathbf{x} \quad (\text{A.443})$$

where  $\mathbf{J}_f(\mathbf{x})$  is the *Jacobian matrix* of the function. We get

$$\|\delta\mathbf{y}\| = \|\mathbf{f}(\mathbf{x} + \delta\mathbf{x}) - \mathbf{f}(\mathbf{x})\| \approx \|\mathbf{J}_f(\mathbf{x})\delta\mathbf{x}\| \leq \|\mathbf{J}_f(\mathbf{x})\| \|\delta\mathbf{x}\| \quad (\text{A.444})$$

and the absolute condition number:

$$\hat{\kappa} = \frac{\|\delta\mathbf{y}\|}{\|\delta\mathbf{x}\|} \approx \|\mathbf{J}_f(\mathbf{x})\| \quad (\text{A.445})$$

Dividing both sides of the equation above by  $\|\mathbf{y}\| = \|\mathbf{f}(\mathbf{x})\|$  we further get

$$\frac{\|\delta\mathbf{y}\|}{\|\mathbf{y}\|} \approx \frac{\|\mathbf{J}_f(\mathbf{x})\delta\mathbf{x}\|}{\|\mathbf{f}(\mathbf{x})\|} \leq \frac{\|\mathbf{x}\| \|\mathbf{J}_f(\mathbf{x})\|}{\|\mathbf{f}(\mathbf{x})\|} \frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \quad (\text{A.446})$$

and we can find the relative condition number as

$$\kappa = \frac{\|\delta\mathbf{y}\|/\|\mathbf{y}\|}{\|\delta\mathbf{x}\|/\|\mathbf{x}\|} \approx \frac{\|\mathbf{x}\| \|\mathbf{J}_f(\mathbf{x})\|}{\|\mathbf{f}(\mathbf{x})\|} \quad (\text{A.447})$$

Consider as an example the following two problems both associated with a linear function:

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} \quad (\text{A.448})$$

- Evaluating  $\mathbf{y} = \mathbf{A}\mathbf{x}$  given  $\mathbf{x}$ :

The Taylor series of this linear function is

$$\mathbf{f}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{J}_f \delta\mathbf{x} = \mathbf{A}\mathbf{x} + \delta\mathbf{A}\mathbf{x} \quad (\text{A.449})$$

i.e.,  $\delta\mathbf{y} = \mathbf{A}\delta\mathbf{x}$  and  $\|\delta\mathbf{y}\| \leq \|\mathbf{A}\| \|\delta\mathbf{x}\|$ . We get the upper bound of the absolute condition number:

$$\hat{\kappa}(\mathbf{A}) = \frac{\|\delta\mathbf{y}\|}{\|\delta\mathbf{x}\|} \leq \|\mathbf{A}\| \quad (\text{A.450})$$

The relative condition number of this linear system is

$$\kappa(\mathbf{A}) = \frac{\|\mathbf{x}\| \|\mathbf{J}_f(\mathbf{x})\|}{\|\mathbf{f}(\mathbf{x})\|} = \|\mathbf{A}\| \frac{\|\mathbf{x}\|}{\|\mathbf{A}\mathbf{x}\|} \quad (\text{A.451})$$

We also have

$$\|\mathbf{x}\| = \|\mathbf{A}^{-1}\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}^{-1}\| \|\mathbf{A}\mathbf{x}\|, \quad \text{i.e.,} \quad \|\mathbf{A}\mathbf{x}\| \geq \frac{\|\mathbf{x}\|}{\|\mathbf{A}^{-1}\|} \quad (\text{A.452})$$

Substituting this into the expression of  $\kappa(\mathbf{A})$  above we get its upper bound:

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \frac{\|\mathbf{x}\|}{\|\mathbf{A}\mathbf{x}\|} \leq \|\mathbf{A}\| \|\mathbf{A}^{-1}\| \frac{\|\mathbf{x}\|}{\|\mathbf{x}\|} = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| \quad (\text{A.453})$$

Note that  $\kappa$  is no less than 1:

$$\kappa(\mathbf{A}) = \|\mathbf{A}^{-1}\| \|\mathbf{A}\| \geq \|\mathbf{A}^{-1}\mathbf{A}\| = \|\mathbf{I}\| = 1 \quad (\text{A.454})$$

- Solving  $\mathbf{A}\mathbf{x} = \mathbf{y}$  for  $\mathbf{x}$  as output given  $\mathbf{y}$  as input:

We assume  $\mathbf{A}$  is a full-rank square matrix and therefore invertible, so that there exists a unique solution:  $\mathbf{x} = \mathbf{g}(\mathbf{y}) = \mathbf{A}^{-1}\mathbf{y}$ . The absolute condition number is  $\hat{\kappa}(\mathbf{A}) = \|\mathbf{J}_g(\mathbf{x})\| = \|\mathbf{A}^{-1}\|$ .

To find the relative condition number  $\kappa(\mathbf{A})$ , consider  $\delta\mathbf{x}$  in the output  $\mathbf{x}$  caused by  $\delta\mathbf{A}$  as well as  $\delta\mathbf{y}$ :

$$(\mathbf{A} + \delta\mathbf{A})(\mathbf{x} + \delta\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{A}\delta\mathbf{x} + \delta\mathbf{A}\mathbf{x} + \delta\mathbf{A}\delta\mathbf{x} = \mathbf{y} + \delta\mathbf{y} \quad (\text{A.455})$$

Subtracting  $\mathbf{A}\mathbf{x} = \mathbf{y}$  from both sides we get

$$\mathbf{A}\delta\mathbf{x} + \delta\mathbf{A}\mathbf{x} + \delta\mathbf{A}\delta\mathbf{x} = \delta\mathbf{y} \quad (\text{A.456})$$

i.e.,

$$\delta\mathbf{x} = \mathbf{A}^{-1}(\delta\mathbf{y} - \delta\mathbf{A}\mathbf{x} - \delta\mathbf{A}\delta\mathbf{x}) \quad (\text{A.457})$$

Taking norm on both sides, we get

$$\begin{aligned} \|\delta\mathbf{x}\| &\leq \|\mathbf{A}^{-1}\| \|\delta\mathbf{y} - (\delta\mathbf{A}\mathbf{x} + \delta\mathbf{A}\delta\mathbf{x})\| \\ &\leq \|\mathbf{A}^{-1}\| (\|\delta\mathbf{y}\| + \|(\delta\mathbf{A}\mathbf{x} + \delta\mathbf{A}\delta\mathbf{x})\|) \\ &\leq \|\mathbf{A}^{-1}\| (\|\delta\mathbf{y}\| + \|\delta\mathbf{A}\| \|\mathbf{x}\| + \|\delta\mathbf{A}\| \|\delta\mathbf{x}\|) \end{aligned} \quad (\text{A.458})$$

where we have used  $\|\mathbf{a} - \mathbf{b}\| \leq \|\mathbf{a}\| + \|\mathbf{b}\|$ ,  $\|\mathbf{a} + \mathbf{b}\| \leq \|\mathbf{a}\| + \|\mathbf{b}\|$ , and  $\|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\| \|\mathbf{x}\|$ .

Subtracting  $\|\mathbf{A}^{-1}\| \|\delta\mathbf{A}\| \|\delta\mathbf{x}\|$  from both sides

$$(1 - \|\mathbf{A}^{-1}\| \|\delta\mathbf{A}\|) \|\delta\mathbf{x}\| \leq \|\mathbf{A}^{-1}\| (\|\delta\mathbf{y}\| + \|\delta\mathbf{A}\| \|\mathbf{x}\|) \quad (\text{A.459})$$

and dividing both sides by  $(1 - \|\mathbf{A}^{-1}\| \|\delta\mathbf{A}\|) \|\mathbf{y}\|$  (assuming not zero), we get

$$\begin{aligned} \frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} &\leq \frac{\|\mathbf{A}^{-1}\|}{1 - \|\mathbf{A}^{-1}\| \|\delta\mathbf{A}\|} \left( \frac{\|\delta\mathbf{y}\|}{\|\mathbf{x}\|} + \|\delta\mathbf{A}\| \right) = \frac{\|\mathbf{A}^{-1}\| \|\mathbf{A}\|}{1 - \|\mathbf{A}^{-1}\| \|\delta\mathbf{A}\|} \left( \frac{\|\delta\mathbf{y}\|}{\|\mathbf{x}\| \|\mathbf{A}\|} + \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} \right) \\ &\leq \frac{\|\mathbf{A}^{-1}\| \|\mathbf{A}\|}{1 - \|\mathbf{A}^{-1}\| \|\delta\mathbf{A}\|} \left( \frac{\|\delta\mathbf{y}\|}{\|\mathbf{y}\|} + \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} \right) = \frac{\kappa(\mathbf{A})}{1 - \kappa(\mathbf{A}) \|\delta\mathbf{A}\|/\|\mathbf{A}\|} \left( \frac{\|\delta\mathbf{y}\|}{\|\mathbf{y}\|} + \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} \right) \end{aligned} \quad (\text{A.460})$$

where we have used  $\|\mathbf{y}\| = \|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\| \|\mathbf{x}\|$ , and  $\kappa(\mathbf{A}) = \|\mathbf{A}^{-1}\| \|\mathbf{A}\|$ .

In particular,

– if  $\delta\mathbf{A} = 0$ ,

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(\mathbf{A}) \frac{\|\delta\mathbf{y}\|}{\|\mathbf{y}\|} \quad (\text{A.461})$$

– if  $\delta\mathbf{y} = 0$ ,

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(\mathbf{A})}{1 - \kappa(\mathbf{A}) \|\delta\mathbf{A}\|/\|\mathbf{A}\|} \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} \quad (\text{A.462})$$

We see that the relative condition number is  $\kappa(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$ , same for both problems of evaluation  $\mathbf{y} = \mathbf{A}\mathbf{x}$  and solving equation  $\mathbf{A}\mathbf{x} = \mathbf{y}$ .

We further consider the spectral norm defined as the greatest singular value of the matrix  $\|\mathbf{A}\| = \sigma_{max}$ . Then  $\|\mathbf{A}^{-1}\| = 1/\sigma_{min}$  is the reciprocal of the smallest singular value of  $\mathbf{A}$ . Now the relative condition number is

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| = \frac{\sigma_{max}}{\sigma_{min}} \quad (\text{A.463})$$

We see that  $\kappa\mathbf{A}$  is large if its  $\sigma_{max}$  and  $\sigma_{min}$  are far apart in values, but it is small otherwise. It is therefore a measurement of how close  $\mathbf{A}$  is to singularity. If  $\mathbf{A}$  is singular, one or more of its singular values are zero, i.e.,  $\sigma_{min} = 0$ , then  $\kappa(\mathbf{A}) = \infty$ , and the system  $\mathbf{A}\mathbf{x} = \mathbf{y}$  is worst conditioned. On the other hand, if  $\mathbf{A} = \mathbf{I}$ , i.e., all singular values equal to 1, then  $\kappa(\mathbf{I}) = 1$ , the smallest condition number possible, and the system  $\mathbf{I}\mathbf{x} = \mathbf{y}$  is best conditioned.

In Matlab, the function `cond(A,p)` generates the condition number of matrix  $\mathbf{A}$  based on its  $p$ -norm. When  $p = 2$ , this is the ratio between the greatest and smallest singular values. The following Matlab commands give the same result:

- `cond(A,2)`
- `norm(A,2)*norm(inv(A),2)`
- `s=svd(A), s(1)/s(end)`

**Consider whether the following examples are needed...**

We now further consider the worst case scenarios for both evaluating the matrix-vector product  $\mathbf{y} = \mathbf{A}\mathbf{x}$  and solving the linear equation system  $\mathbf{A}\mathbf{x} = \mathbf{y}$ .

Consider the SVD decomposition  $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$  and assume the singular values

in  $\Sigma$  are sorted in descending order:  $\sigma_1 \geq \dots \geq \sigma_n$ . The SVD of the inverse is  $\mathbf{A}^{-1} = \mathbf{V}\Sigma^{-1}\mathbf{U}^*$ .

- The worst case for evaluating  $\mathbf{Ax} = \mathbf{y}$ :

If  $\mathbf{x} = \mathbf{v}_n$  is the minimum right singular vector corresponding to the smallest singular value  $\sigma_n = \sigma_{min}$ . The equation becomes:

$$\begin{aligned} \mathbf{y} = \mathbf{Ax} &= (\mathbf{U}\Sigma\mathbf{V}^*) \mathbf{v}_n = \mathbf{U}\Sigma \begin{bmatrix} \mathbf{v}_1^* \\ \vdots \\ \mathbf{v}_n^* \end{bmatrix} \mathbf{v}_n = \mathbf{U} \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_n \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \\ &= [\mathbf{u}_1, \dots, \mathbf{u}_n] \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \sigma_n \end{bmatrix} = \sigma_n \mathbf{u}_n \end{aligned} \quad (\text{A.464})$$

and

$$\mathbf{y} + \delta\mathbf{y} = \mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \sigma_n \mathbf{u}_n + \mathbf{A}\delta\mathbf{x} \quad (\text{A.465})$$

The absolute error  $\|\delta\mathbf{y}\| = \|\mathbf{A}\delta\mathbf{x}\|$  may be small if  $\delta\mathbf{x}$  is small, but the relative error  $\|\delta\mathbf{y}\|/\|\mathbf{y}\|$  may be large as  $\|\mathbf{y}\| = |\sigma_n| \|\mathbf{u}\|$  is small when  $\sigma_n = \sigma_{min}$  is small (e.g., close to zero).

- The worst case for solving  $\mathbf{y} = \mathbf{Ay}$ :

If  $\mathbf{y} = \mathbf{u}_1$  is the maximum left singular vector corresponding to the largest singular value  $\sigma_1 = \sigma_{max}$ . The equation becomes:

$$\begin{aligned} \mathbf{x} = \mathbf{A}^{-1}\mathbf{y} &= \mathbf{V}\Sigma^{-1}\mathbf{U}^*\mathbf{u}_1 = \mathbf{V}\Sigma^{-1} \begin{bmatrix} \mathbf{u}_1^* \\ \vdots \\ \mathbf{u}_n^* \end{bmatrix} \mathbf{u}_1 = \mathbf{V} \begin{bmatrix} 1/\sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/\sigma_n \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ &= [\mathbf{v}_1, \dots, \mathbf{v}_n] \begin{bmatrix} 1/\sigma_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \frac{1}{\sigma_1} \mathbf{v}_1 \end{aligned} \quad (\text{A.466})$$

and

$$\mathbf{x} + \delta\mathbf{x} = \mathbf{A}^{-1}(\mathbf{y} + \delta\mathbf{y}) = \frac{1}{\sigma_1} \mathbf{v}_1 + \mathbf{A}^{-1}\delta\mathbf{y} \quad (\text{A.467})$$

The absolute error  $\|\delta\mathbf{x}\| = \|\mathbf{A}^{-1}\delta\mathbf{y}\|$  may be small if  $\delta\mathbf{y}$  is small, but the relative error  $\|\delta\mathbf{x}\|/\|\mathbf{x}\|$  may be large as  $\|\mathbf{x}\| = \|\mathbf{u}\|/\sigma_1$  is small when  $\sigma_1 = \sigma_{max}$  is large.

#### Example 0:

$$\mathbf{A} = \begin{bmatrix} 5.4321 & 1.2345 \\ 10.8643 & 2.4689 \end{bmatrix}, \quad (\text{A.468})$$

The singular values are  $\sigma_1 = 12.45633$ ,  $\sigma_2 = 0.00005$

$$\|\mathbf{A}\| = \sigma_1 = 12.45633, \quad \|\mathbf{A}^{-1}\| = 1/\sigma_2 = 18684.68, \quad \kappa(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| = 12.45633 \times 18684.68 = 2327 \quad (\text{A.469})$$

As  $\mathbf{A}$  is near singular, its condition number is large, indicating  $\mathbf{Ax} = \mathbf{y}$  is an ill-conditioned problem. It is easy to see that the problem of solving  $\mathbf{Ay} = \mathbf{x}$  is ill-conditioned. Assuming  $\delta\mathbf{A} = 0$ , we have

$$\mathbf{y} + \delta\mathbf{y} = \mathbf{A}^{-1}(\mathbf{x} + \delta\mathbf{x}), \quad \delta\mathbf{y} = \mathbf{A}^{-1}\delta\mathbf{x} \quad (\text{A.470})$$

The change in output  $\mathbf{y}$  is

$$\|\delta\mathbf{y}\| \leq \|\mathbf{A}^{-1}\| \|\delta\mathbf{x}\| = 18684.68 \|\delta\mathbf{x}\| \quad (\text{A.471})$$

which is 18684.68 times greater than any change  $\delta\mathbf{x}$  in the input. However, for an evaluation problem to find  $\mathbf{y} = \mathbf{Ax}$  given  $\mathbf{x}$ ,

$$\mathbf{y} + \delta\mathbf{y} = \mathbf{A}(\mathbf{x} + \delta\mathbf{x}), \quad \delta\mathbf{y} = \mathbf{A}\delta\mathbf{x} \quad (\text{A.472})$$

The change in output  $\mathbf{y}$  is

$$\|\delta\mathbf{y}\| \leq \|\mathbf{A}\| \|\delta\mathbf{x}\| = 12.45633 \|\delta\mathbf{x}\| \quad (\text{A.473})$$

which is not too much greater than the change  $\delta\mathbf{x}$  in the input.

**Example 1:** Consider solving the linear system  $\mathbf{Ax} = \mathbf{b}$  with

$$\mathbf{A} = \frac{1}{2} \begin{bmatrix} 3.0 & 2.0 \\ 1.0 & 4.0 \end{bmatrix}, \quad \mathbf{A}^{-1} = \begin{bmatrix} 0.8 & -0.4 \\ -0.2 & 0.6 \end{bmatrix} \quad (\text{A.474})$$

The singular values of  $\mathbf{A}$  are  $\sigma_1 = 2.5583$  and  $\sigma_2 = 0.9772$ , the condition number is

$$\kappa = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| = \frac{\sigma_1}{\sigma_2} = 2.618 \quad (\text{A.475})$$

which is small, indicating this is a well-behaved system. Given two similar inputs  $\mathbf{b}_1 = [1, 1]^T$  and  $\mathbf{b}_2 = [0.99, 1.01]^T$  with  $\delta\mathbf{b} = [0.01, -0.01]^T$ , we find the corresponding solutions:

$$\mathbf{x}_1 = \mathbf{A}^{-1}\mathbf{b}_1 = [0.4/0.4], \quad \mathbf{x}_2 = \mathbf{A}^{-1}\mathbf{b}_2 = \begin{bmatrix} 0.388 \\ 0.408 \end{bmatrix}, \quad \delta\mathbf{x} = \begin{bmatrix} 0.012 \\ -0.008 \end{bmatrix} \quad (\text{A.476})$$

We have

$$\|\delta\mathbf{b}\| = 0.0141, \quad \|\mathbf{b}_1\| = 1.4142, \quad \|\delta\mathbf{x}\| = 0.0144, \quad \|\mathbf{x}_1\| = 0.5657 \quad (\text{A.477})$$

and

$$\frac{\|\delta\mathbf{x}\|/\|\mathbf{x}_1\|}{\|\delta\mathbf{b}\|/\|\mathbf{b}_1\|} = \frac{0.0255}{0.01} = 2.5495 \quad (\text{A.478})$$

**Example 2:**

$$\mathbf{A} = \frac{1}{2} \begin{bmatrix} 1.000 & 1.000 \\ 1.001 & 0.999 \end{bmatrix}, \quad \mathbf{A}^{-1} = \begin{bmatrix} -999 & 1000 \\ 1001 & -1000 \end{bmatrix} \quad (\text{A.479})$$

The singular values of  $\mathbf{A}$  are  $\sigma_1 = 1.0$  and  $\sigma_2 = 0.0005$ , the condition number is

$$\kappa = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| = \frac{\sigma_1}{\sigma_2} = 2000 \quad (\text{A.480})$$

indicating matrix  $\mathbf{A}$  is close to singularity, and the system is ill-conditioned. The solutions corresponding to the same two inputs  $\mathbf{b}_1 = [1, 1]^T$  and  $\mathbf{b}_2 = [0.99, 1.01]^T$  are

$$\mathbf{x}_1 = \mathbf{A}^{-1}\mathbf{b}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \mathbf{x}_2 = \mathbf{A}^{-1}\mathbf{b}_2 = \begin{bmatrix} 20.99 \\ -19.01 \end{bmatrix}, \quad \delta\mathbf{x} = \begin{bmatrix} -19.99 \\ 20.01 \end{bmatrix} \quad (\text{A.481})$$

We can further find

$$\|\delta\mathbf{b}\| = 0.0141, \quad \|\mathbf{b}_1\| = 1.4142, \quad \|\delta\mathbf{x}\| = 28.2843, \quad \|\mathbf{x}_1\| = 1.4142 \quad (\text{A.482})$$

and the condition number is:

$$\frac{\|\delta\mathbf{x}\|/\|\mathbf{x}_1\|}{\|\delta\mathbf{b}\|/\|\mathbf{b}_1\|} = \frac{200}{0.01} = 2000 \quad (\text{A.483})$$

We see that a small relative change  $\|\delta\mathbf{b}\|/\|\mathbf{b}_1\| = 0.01$  in the input caused a huge change  $\|\delta\mathbf{x}\|/\|\mathbf{x}_1\| = 20$  in the output (2000 times greater).

**Example 3:** Consider

$$\mathbf{A} = \begin{bmatrix} 3.1 & 2.0 \\ 6.0 & 3.9 \end{bmatrix} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \begin{bmatrix} -0.458 & -0.889 \\ -0.889 & 0.458 \end{bmatrix} \begin{bmatrix} 8.051 & 0.0 \\ 0.0 & 0.011 \end{bmatrix} \begin{bmatrix} -0.839 & -0.544 \\ -0.544 & 0.839 \end{bmatrix} \quad (\text{A.484})$$

with

$$\|\mathbf{A}\| = \sigma_1 = 8.05, \quad \|\mathbf{A}^{-1}\| = 1/\sigma_2 = 89.46, \quad \kappa = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| = 720.22 \quad (\text{A.485})$$

As  $\mathbf{A}$  is near singular, its condition number  $\kappa$  is large, indicating that the problem associated with this  $\mathbf{A}$  is ill-conditioned. We will now specifically consider both problems of evaluating and solving the linear system.

- Solve  $\mathbf{A}\mathbf{y} = \mathbf{x}$  for  $\mathbf{y}$  given  $\mathbf{x}$ :

$$\mathbf{y} + \delta\mathbf{y} = \mathbf{A}^{-1}(\mathbf{x} + \delta\mathbf{x}), \quad \delta\mathbf{y} = \mathbf{A}^{-1}\delta\mathbf{x} \quad (\text{A.486})$$

The absolute condition number is  $\hat{\kappa} = \|\mathbf{A}^{-1}\| = 89.46$ . The change in output  $\mathbf{y}$  is

$$\|\delta\mathbf{y}\| \leq \|\mathbf{A}^{-1}\| \|\delta\mathbf{x}\| = \hat{\kappa} \|\delta\mathbf{x}\| = 89.46 \|\delta\mathbf{x}\| \quad (\text{A.487})$$

which is  $\hat{\kappa} = 89.46$  times greater than any change  $\delta\mathbf{x}$  in the input. We further consider the worst case when  $\mathbf{x} = \mathbf{u}_{max}$  and the relative error:

$$\mathbf{x} = \mathbf{u}_1 = \begin{bmatrix} -0.458 \\ -0.889 \end{bmatrix}, \quad \mathbf{y} = \mathbf{A}^{-1}\mathbf{x} = \begin{bmatrix} -0.104 \\ -0.068 \end{bmatrix} \quad (\text{A.488})$$



Even for a small change  $\delta\mathbf{x}$  in the input, causing a small change  $\delta\mathbf{y}$  in the output:

$$\delta\mathbf{x} = \begin{bmatrix} 0.01 \\ -0.01 \end{bmatrix}, \quad \delta\mathbf{y} = \mathbf{A}^{-1}\delta\mathbf{x} = \begin{bmatrix} 0.656 \\ -1.011 \end{bmatrix} \quad (\text{A.489})$$

the ratios of the absolute and relative errors are:

$$\frac{\|\delta\mathbf{y}\|}{\|\delta\mathbf{x}\|} = \frac{1.205}{0.014} \approx 85.21, \quad \frac{\|\delta\mathbf{y}\|/\|\mathbf{y}\|}{\|\delta\mathbf{x}\|/\|\mathbf{x}\|} = \frac{9.702}{0.014} \approx 686 \quad (\text{A.490})$$

- Evaluate  $\mathbf{y} = \mathbf{A}\mathbf{x}$  given  $\mathbf{x}$ :

$$\mathbf{y} + \delta\mathbf{y} = \mathbf{A}(\mathbf{x} + \delta\mathbf{x}), \quad \delta\mathbf{y} = \mathbf{A}\delta\mathbf{x} \quad (\text{A.491})$$

The absolute condition number is  $\hat{\kappa} = \|\mathbf{A}\| = 8.05$ . The absolute change in output  $\mathbf{y}$  is

$$\|\delta\mathbf{y}\| \leq \|\mathbf{A}\| \|\delta\mathbf{x}\| = 8.051 \|\delta\mathbf{x}\| \quad (\text{A.492})$$

which is not too much greater than the absolute change  $\delta\mathbf{x}$  in the input. However, a much greater relative error will result if we consider the worst case when  $\mathbf{x} = \mathbf{v}_{min}$ :

$$\mathbf{x} = \mathbf{v}_2 = \begin{bmatrix} -0.5444 \\ 0.8388 \end{bmatrix}, \quad \mathbf{y} = \mathbf{A}\mathbf{x} = \begin{bmatrix} -0.0099 \\ 0.0051 \end{bmatrix} \quad (\text{A.493})$$

Even for a small change  $\delta\mathbf{x}$  in the input, causing a small change  $\delta\mathbf{y}$  in the output:

$$\delta\mathbf{x} = \begin{bmatrix} 0.01 \\ -0.01 \end{bmatrix}, \quad \delta\mathbf{y} = \mathbf{A}\delta\mathbf{x} = \begin{bmatrix} 0.011 \\ 0.021 \end{bmatrix} \quad (\text{A.494})$$

The ratios of the absolute and relative errors are:

$$\frac{\|\delta\mathbf{y}\|}{\|\delta\mathbf{x}\|} \approx 1.68, \quad \frac{\|\delta\mathbf{y}\|/\|\mathbf{y}\|}{\|\delta\mathbf{x}\|/\|\mathbf{x}\|} = \frac{2.1207}{0.0141} \approx 150 \quad (\text{A.495})$$

Although the ratio of the absolute errors is small, the ratio of the relative errors is large.

## A.27 Some Useful Inequalities

- **The interpolation inequality for  $e^x$**

$$e^{ta+(1-t)b} \leq te^a + (1-t)e^b, \quad t \in [0, 1], \quad a, b \geq 0 \quad (\text{A.496})$$

**Proof:** Evaluating the exponential function  $y = f(x) = e^x$  at two points  $x = a$  and  $x = b$  we get two points in the x-y plane  $(a, e^a)$  and  $(b, e^b)$ . The parametric expression of the secant line through these two points is

$$(ta + (1-t)b, te^a + (1-t)e^b), \quad (0 \leq t \leq 1) \quad (\text{A.497})$$

The parametric expression of the function  $y = e^x$  between these two points is:

$$(ta + (1-t)b, e^{ta+(1-t)b}), \quad (0 \leq t \leq 1) \quad (\text{A.498})$$

As the  $y = e^x$  is convex downward (or concave upward), it is never higher than the secant line function in the interval  $a \leq x \leq b$ , and we have

$$e^{ta+(1-t)b} \leq te^a + (1-t)e^b \quad (\text{A.499})$$

The equality holds only when  $t = 0$  or  $t = 1$ .

- **Young's inequality**

For any two real positive values  $p, q > 0$  satisfying

$$\frac{1}{p} + \frac{1}{q} = 1, \quad q = \frac{p}{p-1}, \quad p = \frac{q}{q-1} \quad (\text{A.500})$$

and any two real numbers  $\alpha$  and  $\beta$ , we have

$$\alpha\beta \leq \frac{\alpha^p}{p} + \frac{\beta^q}{q} \quad (\text{A.501})$$

**Proof:** Substituting  $a = p \log \alpha$  and  $b = q \log \beta$  into the interpolation inequality for  $f(x) = e^x$  above, we get:

$$\begin{aligned} e^{tp \log \alpha + (1-t)q \log \beta} &= e^{tp \log \alpha} e^{(1-t)q \log \beta} = \alpha^{tp} \beta^{(1-t)q} \\ &\leq t e^{p \log \alpha} + (1-t) e^{q \log \beta} = t \alpha^p + (1-t) \beta^q \end{aligned}$$

We further let  $t = 1/p$ , and get  $1-t = 1 - 1/p = 1/q$ , the above becomes Young's inequality:

$$\alpha\beta \leq \frac{\alpha^p}{p} + \frac{\beta^q}{q} \quad (\text{A.502})$$

- **Holder's inequality**

$$\left| \sum_i x_i y_i \right| \leq \sum_i |x_i y_i| \leq \|\mathbf{x}\|_p \|\mathbf{y}\|_q \quad (\text{A.503})$$

**Proof:** Applying Young's inequality to the following

$$\alpha = \frac{|x_i|}{\|\mathbf{x}\|_p} = \frac{|x_i|}{(\sum_i |x_i|^p)^{1/p}}, \quad \beta = \frac{|y_i|}{\|\mathbf{y}\|_q} = \frac{|y_i|}{(\sum_i |y_i|^q)^{1/q}} \quad (\text{A.504})$$

we get

$$\alpha\beta = \frac{|x_i| |y_i|}{\|\mathbf{x}\|_p \|\mathbf{y}\|_q} \leq \frac{\alpha^p}{p} + \frac{\beta^q}{q} = \frac{1}{p} \frac{|x_i|^p}{\sum_i |x_i|^p} + \frac{1}{q} \frac{|y_i|^q}{\sum_i |y_i|^q} \quad (\text{A.505})$$

Taking summation on both sides we get

$$\frac{\sum_i |x_i| |y_i|}{\|\mathbf{x}\|_p \|\mathbf{y}\|_q} \leq \frac{1}{p} + \frac{1}{q} = 1 \quad (\text{A.506})$$

i.e.,

$$\sum_i |x_i y_i| \leq \sum_i |x_i| |y_i| \leq \|\mathbf{x}\|_p \|\mathbf{y}\|_q \quad (\text{A.507})$$

In particular when  $p = 2$ ,  $q = p/(p - 1) = 2$ , we have

$$|\langle \mathbf{x}, \mathbf{y} \rangle| = \left| \sum_i x_i \bar{y}_i \right| = \left| \sum_i x_i y_i \right| \leq \sum_i |x_i y_i| \leq \|\mathbf{x}\|_2 \|\mathbf{y}\|_2 \quad (\text{A.508})$$

This is the Cauchy-Schwarz inequality.

• **Minkowski's inequality**

$$\|\mathbf{x} + \mathbf{y}\|_p \leq \|\mathbf{x}\|_p + \|\mathbf{y}\|_p, \quad p \geq 1 \quad (\text{A.509})$$

**Proof:**

$$\begin{aligned} \|\mathbf{x} + \mathbf{y}\|_p^p &= \sum_i |x_i + y_i|^p = \sum_i |x_i + y_i| |x_i + y_i|^{p-1} \leq \sum_i (|x_i| + |y_i|) |x_i + y_i|^{p-1} \\ &= \sum_i |x_i| |x_i + y_i|^{p-1} + \sum_i |y_i| |x_i + y_i|^{p-1} \end{aligned}$$

Applying Holder's inequality to each of the two terms on the right-hand side, we get

$$\|\mathbf{x} + \mathbf{y}\|_p^p \leq \|\mathbf{x}\|_p \left( \sum_i |x_i + y_i|^{(p-1)q} \right)^{1/q} + \|\mathbf{y}\|_p \left( \sum_i |x_i + y_i|^{(p-1)q} \right)^{1/q} \quad (\text{A.510})$$

But as  $1/p + 1/q = 1$ , and

$$q = p/(p - 1), \quad 1/q = (p - 1)/p, \quad (p - 1)q = p \quad (\text{A.511})$$

the above becomes:

$$\|\mathbf{x} + \mathbf{y}\|_p^p \leq \|\mathbf{x}\|_p \left( \sum_i |x_i + y_i|^p \right)^{(p-1)/p} + \|\mathbf{y}\|_p \left( \sum_i |x_i + y_i|^p \right)^{(p-1)/p} = (\|\mathbf{x}\|_p + \|\mathbf{y}\|_p) \|\mathbf{x} + \mathbf{y}\|_p^{p-1} \quad (\text{A.512})$$

Multiplying both sides by  $\|\mathbf{x} + \mathbf{y}\|_p^{1-p}$ , we get Minkowski's inequality.

# Notes

## Chapter 2

1 Lewis Fry Richardson (1881–1953).

## Author index

Peterson, K., 42  
Tranah, D.A., 42  
Young, P.D.F., 42

## Subject index

diffraction, 42

force

hydrodynamic, 42

interactive, 42