

## Lab 7: Linear Algebra in C on a Microcontroller SOLUTIONS

*Digital Design and Computer Architecture: RISC-V Edition (Harris & Harris, Elsevier © 2021)*

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.

### 2. [5] Code

```
// Lab_6_code_es.c
// erik_spjut@hmc.edu 24 October 2017

#include <stdlib.h>

double* newMatrix(int m, int n) {
    return (double*)malloc(m*n*sizeof(double));
}

double* newIdentityMatrix(int n) {
    double *mat = newMatrix(n, n);
    int i, j;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            mat[j+i*n] = (i==j);
    return mat;
}

double dotproduct(int n, double a[], double b[]) {
    volatile int i;
    double sum;
    for (i=0; i<n; i++) {
        if (i==0) sum = 0;
        sum += a[i]*b[i];
    }
    return sum;
}

void add(int m, int n, double *A, double *B, double *Y) { // Y = A + B
    int i, j;
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            Y[i*n+j] = A[i*n+j] + B[i*n+j];
}

void linearcomb(int m, int n, double sa, double sb, double *A, double *B, double *Y) {
    // Y = sa*A + sb*B
    int i, j;
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            Y[i*n+j] = sa*A[i*n+j] + sb*B[i*n+j];
}

void transpose(int m, int n, double *A, double *At) { // At = transpose(A)
    int i, j;
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            At[j*m+i] = A[i*n+j];
}

int equal(int m, int n, double *A, double *B) { // returns 1 if equal, 0 if not
    int i, j;
    for (i=0; i<m; i++)
        for (j=0; j<n; j++) {
            if (A[i*n+j] != B[i*n+j]) return 0;
        }
}
```

```

    return 1;
}

void mult(int m1, int n1m2, int n2, double *A, double *B, double *Y) { // Y = A * B
    int i, j, k;
    double sum;
    for (i=0; i<m1; i++)
        for (j=0; j<n2; j++) {
            sum =0;
            for (k=0; k<n1m2; k++)
                sum += A[i*n1m2+k] * B[k*n2+j];
            Y[i*n2+j] = sum;
        }
}

int main(void) {

    double v1[3] = {4, 2, 1}; // 1 x 3 vector
    double v2[3] = {1, -2, 3}; // 1 x 3 vector
    double dp = dotproduct(3, v1, v2);
    double m1[9] = {0, 0, 2, 0, 0, 0, 2, 0, 0}; // 3 x 3 matrix
    double *m2 = newIdentityMatrix(3); // 3 x 3 identity matrix
    double *m3 = newMatrix(3, 3); // 3x3matrix
    double m4[6] = {2, 3, 4, 5, 6, 7}; // 3x2matrix
    double *m5 = newMatrix(3, 2); // 3x2matrix
    double m6[6] = {6, 2, 5, 8, 2, 7}; // 2x3matrix
    double *m7 = newMatrix(3, 2); // 3x2matrix
    double *m8 = newMatrix(3, 2); // 3x2matrix
    double expected[6] = {2, 1, 0, 1, 0, -1};
    int eq;

    add(3, 3, m1, m2, m3);

    mult(3, 3, 2, m3, m4, m5);
    // 3x2result
    transpose(2, 3, m6, m7);

    linearcomb(3, 2, 1, 1-dp, m5, m7, m8);

    eq = equal(3, 2, m8, expected);

    return eq;
}

```

3. [1] m8 matches expectations of  $\begin{bmatrix} 2 & 1 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}$

4. [2XC] Extra credit, if applicable

$$x = \begin{bmatrix} -1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

```

double determinant(int n, double *A) { // determinant of A
    int i, j, k, col;
    double det = 0;

```

```

double sgn;

if (n < 1) det = 0; // undefined behavior
else if (n == 1) det = A[0]; // trivial case for 1x1 matrix
else if (n == 2) {
    det = A[0]*A[3] - A[1]*A[2]; // base case for 2x2 matrix
} else {
    // otherwise recursively compute determinant
    // for each column, form a submatrix excluding that column and the first row
    double *Asub = newMatrix(n-1, n-1);
    for (i=0; i<n; i++) {
        // alternate adding and subtracting terms
        if (i%2) sgn = -1.0;
        else sgn = 1.0;
        for (j=0; j<n-1; j++) {
            for (k=0; k<n-1; k++) {
                col = k + (k>=i); // skip over column i
                Asub[j*(n-1)+k] = A[(j+1)*n+col];
            }
            det += sgn * A[i] * determinant(n-1, Asub);
        }
        free(Asub);
    }
    return det;
}

int invert(int n, double *A, double *Y) { // Y = A^-1; return 0 if not invertable, 1 otherwise
    // special cases for small matrices
    if (n < 1) return 0;
    else if (n == 1) {
        if (A[0] == 0) return 0;
        Y[0] = 1/A[0];
        return 1;
    }
    // first compute matrix of minors. Temporarily store in Y
    // for each element, compute the determinant of a submatrix ignoring that row and col
    int i, j, k, l, row, col;
    double *cofactors = newMatrix(n, n);
    double *Asub = newMatrix(n-1, n-1);
    double det;

    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            // Create a submatrix Asub excluding row i and col j
            for (k=0; k<n-1; k++) {
                for (l=0; l<n-1; l++) {
                    row = k + (k>=i); // skip over i
                    col = l + (l>=j); // and j
                    Asub[k*(n-1)+l] = A[row*n+col];
                }
            }
            // compute the determinant of the submatrix
            det = determinant(n-1, Asub);
            // flip the sign of every other entry to make matrix of cofactors
            if ((i+j)%2) det *= -1;
            cofactors[i*n+j] = det;
        }
    }
    free(Asub); // done with submatrix storage
    // Transpose the matrix of cofactors
    transpose(n, n, cofactors, Y);
    free(cofactors); // done with cofactors storage

    // scale transposed matrix by 1/determinant
    det = determinant(n, A);
    if (det == 0) return 0; // not invertable
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            Y[i*n+j] /= det;
    return 1;
}

```

```

int solve(int n, double *A, double *b, double *x) { // solve  $Ax = b$  for  $x = A^{-1} * b$ 
    double *Ainv = newMatrix(n,n); // space for inverse
    if (invert(n, A, Ainv)) {
        mult(n, n, 1, Ainv, b, x);
        free(Ainv);
        return 1;
    }
    else return 0;
}

// extra credit: solving system of equations with matrix inversion
free(m2); free(m3); free(m5); free(m7); free(m8); // free up memory
double m9[16] = {3, 1, 1, 1, 2, 3, 4, -1, -4, 0, 0, 1, 0, 3, -2, 0}; // 4x4 matrix
double b[4] = {6, 12, 8, 0}; // 4 x 1 vector
double *x = newMatrix(4, 1); // space for result
double xexpected[4] = {-1, 2, 3, 4};
solve(4, m9, b, x);
eq = equal(4, 1, x, xexpected);

```

If you have suggestions for further improvements of this lab, you're welcome to include them at the end of your lab.