Experiment-2
Matrix Inversion

(Duration: 120 mins)

Author: Anıl Karatay
anilkaratay@iyte.edu.tr

Purpose: Any NxN square matrix $[A]$ is called invertible if there exist an NxN square matrix such that $AA^{-1} = I$ where I is identity matrix. In this lab, a $4x4$ linear equation system with the matrix inversion method will be solved in C.

Introduction

Matrix inversion is a method in linear algebra to solve a linear equation system. Consider the linear equation system given below.

$$A_{00}x_0 + A_{01}x_1 + A_{02}x_2 + ... + A_{0(n-1)}x_{n-1} = b_0$$
$$\vdots$$
$$A_{(m-1)0}x_0 + A_{(m-1)1}x_1 + A_{(m-1)2}x_2 + ... + A_{(m-1)(n-1)}x_{n-1} = b_{m-1}$$

If **m and n are equal** and the matrix [A] consisting of the coefficients "A$_{ij}$" is an invertible matrix, the vector [x] can be found by the following method.

$$[A]_{nxn}\,[x]_{nx1} - [b]_{nx1} = 0$$
$$[A]^{-1}_{nxn}\,[b]_{nx1} = [x]_{nx1}$$

In order to find the inverse of the matrix [A], the following operations should be done respectively.

1. Find the minors of the matrix for each element as shown below. Note that the arrays in C are indexed starting at 0.

$$A = \begin{bmatrix} 2 & 3 & 5 & 2 \\ -3 & 5 & 7 & 5 \\ 5 & 2 & 2 & 3 \\ 5 & 3 & 8 & 3 \end{bmatrix} \Rightarrow M_{22} = \begin{vmatrix} 2 & 3 & 2 \\ -3 & 5 & 5 \\ 5 & 3 & 3 \end{vmatrix} = 34$$

$$A = \begin{bmatrix} 2 & 3 & 5 & 2 \\ -3 & 5 & 7 & 5 \\ 5 & 2 & 2 & 3 \\ 5 & 3 & 8 & 3 \end{bmatrix} \Rightarrow M_{12} = \begin{vmatrix} 2 & 3 & 2 \\ 5 & 2 & 3 \\ 5 & 3 & 3 \end{vmatrix} = -9$$

2. Calculate the cofactor matrix, $[C]$, by using the minors.

$$C_{ij} = (-1)^{i+j}M_{ij}$$

3. Calculate the adjoint matrix, $[D]$, by transposing the cofactor matrix.

$$D_{ij} = C^T_{ij}$$

4. Calculate the determinant of the [A] matrix, det(A). Then apply the following formulation.

$$A^{-1}_{ij} = \frac{1}{det(A)}D_{ij}$$

Problem Statement

You are asked to write a C program that calculates the solution of a **4x4** linear equation system via matrix inversion method.

- Your program should ask the elements of the coefficient matrix, $[A]$, and the vector $[b]$. You should ensure that the coefficient matrix to be a square matrix and its dimension is 4x4. (10 pts.)
- You should write a function that calculates the determinant of a given 3x3 square matrix. In this part, you don't have to use any loop. You can simply multiply the corresponding elements manually. (10 pts.)
- Minor calculator function is given, but you should write a cofactor matrix generator by using the given minor calculator function. (10 pts.)
- You should write a function that calculates the determinant of a given 4x4 square matrix by using the 3x3 determinant calculator. In this part, using loop is **mandatory**. (20 pts.)
- You should write a function that carries out step 3 and step 4 in Introduction section. This function should calculate the transpose of the cofactor matrix, then the elements of the transposed matrix should be divided by the determinant of [A]. The memory of the adjoint matrix should be allocated dynamically. (15 pts.)
- You should write a multiplication function that carries out a matrix-vector multiplication to find the [x] vector, defined in Introduction section. Again, the inputs of the function should be pointer. (15 pts.)
- The functions listed above should be called appropriately within the main function. You should demonstrate both the inverse of the coefficient matrix, $[A]^{-1}$, and the [x] vector. If the determinant is 0, an error message should be written stating that the matrix is non-invertible. (20 pts.)

Lab Procedure

1. The input parameters i.e. [A] matrix and [b] vector should be asked to the user in main.

$$\begin{bmatrix} \quad & A & \quad \end{bmatrix}_{4x4} \begin{bmatrix} x \end{bmatrix}_{4x1} = \begin{bmatrix} b \end{bmatrix}_{4x1}$$

2. Write a function that calculates the determinant of a 3x3 matrix. You will need this function when calculating the 4x4 matrix determinant. You don't have to use any loop for 3x3 determinant calculator.

```c
float determinant3(float Mat3[N][N])
```

3. Minor calculator function is given below. You can use this function directly for minor calculation. Mat[N][N] is the input matrix and Min[N][N] is the minor matrix of the corresponding matrix.

```c
void minorCalc(float Mat[N][N], float Min[N][N]){
  float b[N][N];
  int m, n, i, j, c, q, p;
  for (q = 0; q < 4; q++){
    for (p = 0; p < 4; p++){
      m = 0;
      n = 0;
      for (i = 0; i < 4; i++){
```

```
            for (j = 0; j < 4; j++){
               if (i != q && j != p){
                  b[m][n] = Mat[i][j];
               if (n < 2)
                  n++;
               else{
                  n = 0;
                  m++;
               }
               }
            }
         }
   Min[q][p] = determinant3(b);
      }
   }
}
```

4. Write a function that generates a cofactor matrix by using the minors of the input matrix (see the 2^{nd} step in Introduction). The structure of the cofactor matrix is given below.

$$C = \begin{bmatrix} M_{00} & -M_{01} & \cdots & \vdots \\ -M_{10} & & & \\ \vdots & & & \\ & & \cdots & M_{33} \end{bmatrix}$$

```
void cofactorCalc(float minorMatrix[N][N])
```

5. Write a function named determinant4() using the determinant3() function you wrote above. This function should return the determinant of a 4x4 matrix. In this part, you **have to** use a loop.

```
float determinant4(float Mat4[N][N])
```

6. Write a function that calculates the adjoint of the cofactor matrix and divides the elements by the determinant of matrix [A]. It was free to use an array or pointer in the previous functions, but it is **mandatory** to use pointer in this function. You cannot get points from functions written without using a pointer. You can see the function prototype below.

```
void Adjoint(float *cofactorMatrix, float *adjointMatrix, float detA)
```

7. The product of the inverted matrix $[A]^{-1}$ and the vector [b] returns the vector [x]. You should write a function that multiplies a matrix and a vector. As with the previous function, the inputs of this function **have to** be a pointer.

```
void matrix_MULT(float *invertedMatrix, float *bVector)
```

8. The functions written above should be properly called in the main or in another functions if it is required. The output of the code should give the [x] vector that is the solution of a linear equation system and the inverted matrix $[A]^{-1}$. If the determinant of [A] is zero, you should return an error message.

Briefly, whether you use a pointer or array between the steps 1-5 is up to you, but you have to use pointer in steps 6 and 7.

You can see some sample outputs of the program in the following.

```
The elements of 4x4 [A] Matrix :
1 2 -1 1
-1 1 2 -1
2 -1 2 2
1 1 -1 2

The elements of 4x1 [b] Vector :
6 3 14 8

Determinant of [A] matrix=17.000000

Inverse of [A] matrix :
        0.882353                -0.411765               0.352941                -1.000000
        0.352941                0.235294                -0.058824               0.000000
        -0.058824               0.294118                0.176471                -0.000000
        -0.647059               0.235294                -0.058824               1.000000

[x] vector :

x[0]: 1.000000
x[1]: 2.000000
x[2]: 3.000000
x[3]: 4.000000
```

```
The elements of 4x4 [A] Matrix :
1 1 1 1
1 1 1 -1
1 1 -1 -1
1 -1 -1 -1

The elements of 4x1 [b] Vector :
1 2 3 4

Determinant of [A] matrix=8.000000

Inverse of [A] matrix :
        0.500000                -0.000000               0.000000                0.500000
        -0.000000               0.000000                0.500000                -0.500000
        0.000000                0.500000                -0.500000               -0.000000
        0.500000                -0.500000               -0.000000               0.000000

[x] vector :

x[0]: 2.500000
x[1]: -0.500000
x[2]: -0.500000
x[3]: -0.500000
```