

# Linear Algebra (0031)

## Project 0

Yulwon Rhee (202211342)

Department of Computer Science and Engineering, Konkuk University

1. (a) `transposeMatrix(A, m, n)`: transpose the  $m \times n$  matrix  $A$  and return the result

Let  $A^T = B$ . Since  $B_{ij} = A_{ji}$ , the code below returns transpose of matrix  $A$ .

```
1 double** transposeMatrix(double **A, int m, int n) {
2     double** B = allocateMemory(n, m);
3
4     for (int i = 0; i < m; i++)
5         for (int j = 0; j < n; j++)
6             B[j][i] = A[i][j];
7
8     return B;
9 }
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$
$$\therefore A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

(a) Equation

```
(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
-----
(x) Solve Problem 2(b)
Select Menu : a

#### Transpose Matrix ####
Enter the number of row : 2
Enter the number of column : 3
Enter the element of matrix :
1 2 3
4 5 6

A =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000

A^T =
1.000000 4.000000
2.000000 5.000000
3.000000 6.000000
```

(b) Result Image

Fig. 1: `transposeMatrix()`

(b) `normalizeVector(v, n)`: normalise the  $n$ -dimensional vector  $\mathbf{v}$  and return the result  
 Since normalised vector is calculated by dividing all entries by its length, the code below returns the normalised vector  $\mathbf{v}$ .

```

1  double** normalizeVector(double** v, int n) {
2      double** w;
3      double len = calculateLength(v, n);
4
5      w = allocateMemory(n, 1);
6      for (int i = 0; i < n; i++)
7          w[i][0] = v[i][0] / len;
8
9      return w;
10 }
```

$$\begin{aligned}
 \mathbf{v} &= \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\
 \|\mathbf{v}\| &= \sqrt{(1)^2 + (-1)^2} \\
 \therefore \hat{\mathbf{v}} &= \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} \\
 \hat{\mathbf{v}} &\approx \begin{bmatrix} 0.707107 \\ -0.707107 \end{bmatrix}
 \end{aligned}$$

(a) Equation

```

(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
-----
(x) Solve Problem 2(b)
Select Menu : b

#### Normalise Vector ####
Enter the number of row : 2
Enter the element of matrix :
1
-1

v =
1.000000
-1.000000

Normalised v =
0.707107
-0.707107
```

(b) Result Image

Fig. 2: `normalizeVector()`

(c) `calculateLength(v, n)`: calculate the length of the  $n$ -dimensional vector  $\mathbf{v}$  and return the result

Since length of vector is calculated by square root of the sum of the square of all entries, the code below returns the length of vector  $\mathbf{v}$ .

```

1  double calculateLength(double** v, int n) {
2      double len = 0.0;
3
4      for (int i = 0; i < n; i++) {
5          len += v[i][0] * v[i][0];
6      }
7      len = sqrt(len);
8
9      return len;
10 }
```

$$\mathbf{v} = \begin{bmatrix} 1 \\ 7 \\ 4 \end{bmatrix}$$

$$\|\mathbf{v}\| = \sqrt{1^2 + 7^2 + 4^2}$$

$$\|\mathbf{v}\| = \sqrt{66}$$

$$\|\mathbf{v}\| \approx 8.124038$$

(a) Equation

```

(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
-----
(x) Solve Problem 2(b)
Select Menu : c

#### Calculate Length ####
Enter the number of row : 3
Enter the element of matrix :
1
7
4

v =
1.000000
7.000000
4.000000
Length of v = 8.124038
```

(b) Result Image

Fig.3: `calculateLength()`

(d) `scaleMatrix(A, m, n, c)`: scale the  $m \times n$  matrix  $A$  with scalar  $c$

The code below returns the matrix  $A$  scaled by  $c$  by multiplying all entries in  $A$  by  $c$ .

```

1  double** scaleMatrix(double** A, int m, int n, double c) {
2      double** cA = allocateMemory(m, n);
3      for (int i = 0; i < m; i++) {
4          for (int j = 0; j < n; j++) {
5              cA[i][j] = c * A[i][j];
6          }
7      }
8
9      return cA;
10 }
```

$$3.14 \begin{bmatrix} 1 & 3 & 2 & 4 \\ 3 & 5 & 4 & 5 \end{bmatrix} = \begin{bmatrix} 3.14 & 9.42 & 6.28 & 12.56 \\ 9.42 & 15.7 & 12.56 & 18.84 \end{bmatrix}$$

(a) Equation

```

(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
-----
(x) Solve Problem 2(b)
Select Menu : d

#### Scale Matrix ####
Enter the number of row : 2
Enter the number of column : 4
Enter the element of matrix :
1 3 2 4
3 5 4 6
Enter the value of scalar c : 3.14

A =
1.000000 3.000000 2.000000 4.000000
3.000000 5.000000 4.000000 6.000000

cA =
3.140000 9.420000 6.280000 12.560000
9.420000 15.700000 12.560000 18.840000
```

(b) Result Image

Fig. 4: `scaleMatrix()`

(e) multiplyTwoMatrices(A, m, n, B, l, k): for  $m \times n$  matrix A and  $l \times k$  matrix B, calculate and return AB. Return null if multiplication is impossible.

The code below returns the multiplication between matrix A and B or NULL if multiplication is impossible.

```

1  double** multiplyTwoMatrices(double** A, int m, int n, double** B, int p, int q) {
2      if (n != p) return NULL;
3
4      double** AB = allocateMemory(m, n);
5
6      for (int i = 0; i < m; i++) {
7          for (int j = 0; j < q; j++) {
8              AB[i][j] = 0;
9              for (int k = 0; k < p; k++) {
10                 AB[i][j] += A[i][k] * B[k][j];
11             }
12         }
13     }
14
15     return AB;
16 }

```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 & 20 & 21 \end{bmatrix} = \begin{bmatrix} 62 & 68 & 74 & 80 & 86 & 92 & 98 \\ 134 & 149 & 164 & 179 & 194 & 209 & 224 \end{bmatrix}$$

(a) Equation

```

(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
-----
(x) Solve Problem 2(b)
Select Menu : e

#### Multiply 2 Matrices ####
Enter the number of row of matrix A : 2
Enter the number of column of matrix A : 3
Enter the element of matrix A :
1 2 3
4 5 6
Enter the number of row of matrix B : 3
Enter the number of column of matrix B : 7
Enter the element of matrix B :
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21

A =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000

B =
1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000
8.000000 9.000000 10.000000 11.000000 12.000000 13.000000 14.000000
15.000000 16.000000 17.000000 18.000000 19.000000 20.000000 21.000000

AB =
62.000000 68.000000 74.000000 80.000000 86.000000 92.000000 98.000000
134.000000 149.000000 164.000000 179.000000 194.000000 209.000000 224.000000

```

(b) Result Image

```

(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
-----
(x) Solve Problem 2(b)
Select Menu : e

#### Multiply 2 Matrices ####
Enter the number of row of matrix A : 1
Enter the number of column of matrix A : 2
Enter the element of matrix A :
1 2
Enter the number of row of matrix B : 3
Enter the number of column of matrix B : 4
Enter the element of matrix B :
1 2 3 4
5 6 7 8
9 10 11 12
Multiplication is impossible.

```

(c) Result Image When Multiplication is Impossible

Fig.5: multiplyTwoMatrices()

(f) `addTwoMatrices(A, m, n, B, l, k)`: for  $m \times n$  matrix  $A$  and  $l \times k$  matrix  $B$ , calculate and return  $A + B$ . Return `null` if addition is impossible.

The code below returns the addition between matrix  $A$  and  $B$  or `NULL` if addition is impossible.

```

1  double** addTwoMatrices(double** A, int m, int n, double** B, int l, int k) {
2      if (m != l || n != k) return NULL;
3
4      double** C = allocateMemory(m, n);
5      for (int i = 0; i < m; i++) {
6          for (int j = 0; j < n; j++) {
7              C[i][j] = A[i][j] + B[i][j];
8          }
9      }
10
11     return C;
12 }

```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

(a) Equation

```

(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
-----
(x) Solve Problem 2(b)
Select Menu : f

#### Add 2 Matrices ####
Enter the number of row of matrix A : 2
Enter the number of column of matrix A : 3
Enter the element of matrix A :
1 2 3
4 5 6
Enter the number of row of matrix B : 2
Enter the number of column of matrix B : 3
Enter the element of matrix B :
7 8 9
10 11 12

A =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000

B =
7.000000 8.000000 9.000000
10.000000 11.000000 12.000000

A+B =
8.000000 10.000000 12.000000
14.000000 16.000000 18.000000

```

(b) Result Image

```

(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
-----
(x) Solve Problem 2(b)
Select Menu : f

#### Add 2 Matrices ####
Enter the number of row of matrix A : 2
Enter the number of column of matrix A : 3
Enter the element of matrix A :
1 2 3
4 5 6
Enter the number of row of matrix B : 3
Enter the number of column of matrix B : 2
Enter the element of matrix B :
1 2
3 4
5 6
Addition is impossible.

```

(c) Result Image When Addition is Impossible

Fig. 6: `addTwoMatrices()`

2. (a) Test the correctness of each of the function you wrote in 1.  
Already done in above.

(b) For given  $n \times n$  matrices  $A$  and  $\tilde{H}$ , normalize each column of  $\tilde{H}$  (let  $H$  be this normalized matrix).  
Then, calculate  $B = H^T A^H$ , and then,  $C = HBH^T$ .

```

1 void problem2b() {
2     double a[2][2] = {
3         {1, 2},
4         {3, 4}
5     };
6
7     double tildeH[2][2] = {
8         {1, 1},
9         {1, -1}
10    };
11
12    double** A = allocateMemory(2, 2);
13    for (int i = 0; i < 2; i++)
14        for (int j = 0; j < 2; j++)
15            A[i][j] = (double) a[i][j];
16    printMatrix(A, 2, 2, "A");
17
18    double** TildeH = allocateMemory(2, 2);
19    for (int i = 0; i < 2; i++)
20        for (int j = 0; j < 2; j++)
21            TildeH[i][j] = (double) tildeH[i][j];
22    printMatrix(TildeH, 2, 2, "Tilde H");
23
24    double** H = normalizeMatrix(TildeH, 2, 2);
25    printMatrix(H, 2, 2, "H");
26
27    double** HT = transposeMatrix(H, 2, 2);
28
29    double** B = multiplyTwoMatrices(HT, 2, 2, A, 2, 2);
30    B = multiplyTwoMatrices(B, 2, 2, H, 2, 2);
31    printMatrix(B, 2, 2, "B");
32
33    double** C = multiplyTwoMatrices(H, 2, 2, B, 2, 2);
34    C = multiplyTwoMatrices(C, 2, 2, HT, 2, 2);
35    printMatrix(C, 2, 2, "C");
36
37    releaseMemory(A, 2);
38    releaseMemory(TildeH, 2);
39    releaseMemory(H, 2);

```

```

40     releaseMemory(HT, 2);
41     releaseMemory(B, 2);
42     releaseMemory(C, 2);
43 }

```

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

$$\tilde{H} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$B = H^T A H$$

$$\begin{aligned}
 &= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \\
 &= \begin{bmatrix} 5 & -1 \\ -2 & 0 \end{bmatrix}
 \end{aligned}$$

$$C = H B H^T$$

$$\begin{aligned}
 &= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 5 & -1 \\ -2 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \\
 &= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}
 \end{aligned}$$

(a) Equation

```

(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices

```

```

-----
(x) Solve Problem 2(b)
Select Menu : x

```

```

A =
1.000000 2.000000
3.000000 4.000000

```

```

Tilde H =
1.000000 1.000000
1.000000 -1.000000

```

```

H =
0.707107 0.707107
0.707107 -0.707107

```

```

B =
5.000000 -1.000000
-2.000000 0.000000

```

```

C =
1.000000 2.000000
3.000000 4.000000

```

(b) Result Image

Fig. 7: problem2b()