Sistem Kontrol Optimal Modul 1 & 2

Kelompok 12 :

Alim Satria Fi’i Wijaya Kusuma
Dani Yudha Kusuma

Operasi Matematika sederhana pada matriks

Penjumlahan

```
static void add_matrix(int row, int col, int matrix_result[row][col],int matrix_input_1[row][col],int matrix_input_2[row][col]){

    for (int x = 0; x < row; x++){
        for (int y = 0; y < col; y++)
            matrix_result[x][y] = matrix_input_1[x][y] + matrix_input_2[x][y];
    }
}
```

Pengurangan

```
static void subtract_matrix(int row, int col, int matrix_result[row][col],int matrix_input_1[row][col],int matrix_input_2[row][col]){

    for (int x = 0; x < row; x++){
        for (int y = 0; y < col; y++)
            matrix_result[x][y] = matrix_input_1[x][y] - matrix_input_2[x][y];
    }
}
```

Perkalian

```
static void multiply_matrix(int row, int col, int matrix_result[row][col],int matrix_input_1[row][col],int matrix_input_2[row][col]){

    for (int x = 0; x < row; x++){
        for (int y = 0; y < col; y++){
            matrix_result[x][y] = 0;
            for (int z = 0; z < col; z++)
                matrix_result[x][y] += matrix_input_1[x][z]*matrix_input_2[z][y];
        }
    }
}
```

Transpose

```
static void transpose_matrix(int row, int col, int matrix_result[row][col],int matrix_input[row][col]){

    int matrix_tmp[row][col];

    for (int x = 0; x < row; x++){
        for (int y = 0; y < col; y++)
            matrix_tmp[y][x] = matrix_input[x][y];
    }

    for (int x = 0; x < row; x++){
        for (int y = 0; y < col; y++)
            matrix_result[x][y] = matrix_tmp[x][y];
    }
}
```

Program 1 Modul 1

```c
#define ROW 2
#define COL 2

#include "MatrixLib.h"

struct matrix_t{
    int data[2][2];
};

struct matrix_t A,B,C;

int main(void){

    // INPUT MATRIX
    input_matrix(ROW,COL,A.data);
    printf("Print Matrix A : \r\n");
    print_matrix(ROW,COL,A.data);

    input_matrix(ROW,COL,B.data);
    printf("Print Matrix B : \r\n");
    print_matrix(ROW,COL,B.data);
```

```c
    // ADD MATRIX
    add_matrix(ROW,COL,C.data,A.data,B.data);
    printf("Add Matrix : \r\n");
    print_matrix(ROW,COL,C.data);

    // SUBTRACT MATRIX
    subtract_matrix(ROW,COL,C.data,A.data,B.data);
    printf("Subtract Matrix : \r\n");
    print_matrix(ROW,COL,C.data);

    // MULTIPLY MATRIX
    multiply_matrix(ROW,COL,C.data,A.data,B.data);
    printf("Multiply Matrix : \r\n");
    print_matrix(ROW,COL,C.data);

    //TRANSPOSE MATRIX
    transpose_matrix(ROW,COL,A.data,A.data);
    printf("Transpose Matrix A : \r\n");
    print_matrix(ROW,COL,A.data);

    transpose_matrix(ROW,COL,B.data,B.data);
    printf("Transpose Matrix B : \r\n");
    print_matrix(ROW,COL,B.data);

```

Hasil Program

```
matrix[0][0] = 1
matrix[0][1] = 2
matrix[1][0] = 3
matrix[1][1] = 4

Print Matrix A :

1       2

3       4


matrix[0][0] = 5
matrix[0][1] = 6
matrix[1][0] = 7
matrix[1][1] = 8

Print Matrix B :

5       6

7       8
```

```
Add Matrix :

6       8

10      12


Subtract Matrix :

-4      -4

-4      -4


Multiply Matrix :

19      22

43      50
```

```
Transpose Matrix A :

1       3

2       4


Transpose Matrix B :

5       7

6       8
```

Program 2 Modul 1

```c
#define ROW 2
#define COL 2

#include "MatrixLib.h"

struct matrix_t{
    int data[2][2];
};

struct matrix_t A,B,P, C,D,E, F;

int main (){

    // Berapakah hasil dari persamaan berikut:
    // A^T+P*A-B = C+P*A-B
    // Dengan nilai matriks A, P dan B ditentukan oleh kalian sendiri

    printf("Masukkan nilai untuk matriks A : \n");
    input_matrix(ROW,COL,A.data);
    printf("Masukkan nilai untuk matriks B : \n");
    input_matrix(ROW,COL,B.data);
    printf("Masukkan nilai untuk matriks P : \n");
    input_matrix(ROW,COL,P.data);
```

```c
    printf("Matrix A :\n");
    print_matrix(ROW,COL,A.data);
    printf("Matrix B :\n");
    print_matrix(ROW,COL,B.data);
    printf("Matrix P :\n");
    print_matrix(ROW,COL,P.data);

    multiply_matrix(ROW,COL,C.data,P.data,A.data);
    printf("P * A : \n");
    print_matrix(ROW,COL,C.data);

    transpose_matrix(ROW,COL,A.data,A.data);
    printf("Transpose matrix A : \n");
    print_matrix(ROW,COL,A.data);

    add_matrix(ROW,COL,D.data,A.data,C.data);
    printf("A^T + [P * A] : \n");
    print_matrix(ROW,COL,D.data);

    subtract_matrix(ROW,COL,E.data,D.data,B.data);
    printf("[A^T + [P * A]] - B : \n");
    print_matrix(ROW,COL,E.data);

    return 0;
}
```

Hasil Program

```
Masukkan nilai untuk matriks A :
matrix[0][0] = 1
matrix[0][1] = 2
matrix[1][0] = 3
matrix[1][1] = 4

Masukkan nilai untuk matriks B :
matrix[0][0] = 5
matrix[0][1] = 6
matrix[1][0] = 7
matrix[1][1] = 8

Masukkan nilai untuk matriks P :
matrix[0][0] = 9
matrix[0][1] = 10
matrix[1][0] = 11
matrix[1][1] = 12
```

```
Matrix A :

1       2

3       4


Matrix B :

5       6

7       8


Matrix P :

9       10

11      12
```

```
P * A :

39      58

47      70


Transpose matrix A :

1       3

2       4


A^T + [P * A] :

40      61

49      74


[A^T + [P * A]] - B :

35      55

42      66
```

determinant ,minor, cofactor, adjoint dan invers

Determinant

```
static float determinant_matrix(int row, int col, int matrix[row][col]){

    float a = 0,b = 0,c = 0;

    if (row == 3 && col == 3) {

        a = matrix[0][0]*(matrix[1][1]*matrix[2][2] - matrix[1][2]*matrix[2][1]);
        b = matrix[0][1]*(matrix[1][0]*matrix[2][2] - matrix[1][2]*matrix[2][0]);
        c = matrix[0][2]*(matrix[1][0]*matrix[2][1] - matrix[1][1]*matrix[2][0]);

        return a-b+c;
    }

    else if (row == 2 && col == 2) {

        a = matrix[0][0]*matrix[1][1];
        b = matrix[0][1]*matrix[1][0];

        return a-b;

    }
}
```

Minor

```c
static void minor_matrix(int row, int col, int matrix_result[row][col][row - 1][col- 1],int matrix_input[row][col],int dimension){

    for(int target_row = 0; target_row < row; target_row++){
        for(int target_col = 0; target_col < col; target_col++){

            int minor_row = 0,minor_col = 0;

            for (int x = 0; x < row; x++){          // row matrix
                for (int y = 0; y < col; y++){      // col matrix
                    if(x != target_row && y != target_col)
                        matrix_result[target_row][target_col][minor_row][minor_col++] = matrix_input[x][y];

                    if(minor_col == dimension - 1){ // dimension of minor matrix
                        minor_col = 0;              // reset minor col
                        minor_row++;                // add new minor row
                    }
                }
            }
        }
    }
}
```

Cofactor

```
static void get_cofactor(int row, int col, int matrix_result[row][col],int matrix_input[row][col][row - 1][col- 1]){

    for (int x = 0; x < row; x++){
        for (int y = 0; y < col; y++){
            matrix_result[x][y] = pow(-1,x+y) * (matrix_input[x][y][0][0]*matrix_input[x][y][1][1]) - (matrix_input[x][y][0][1]*matrix_input[x][y][1][0]);
        }
    }
}
```

Adjoint = Transpose cofactor

```
static void transpose_matrix(int row, int col, int matrix_result[row][col],int matrix_input[row][col]){

    int matrix_tmp[row][col];

    for (int x = 0; x < row; x++){
        for (int y = 0; y < col; y++)
            matrix_tmp[y][x] = matrix_input[x][y];
    }

    for (int x = 0; x < row; x++){
        for (int y = 0; y < col; y++)
            matrix_result[x][y] = matrix_tmp[x][y];
    }
}
```

Invers

```
static void invers_matrix(int row, int col, float matrix_result[row][col],int matrix_input[row][col],float determinant){

    for (int x = 0; x < row; x++){
        for (int y = 0; y < col; y++){
            matrix_result[x][y] = matrix_input[x][y] / fabs(determinant);
        }
    }
}
```

Program 3 Modul 1

```c
#define ROW 3
#define COL 3

#define MINOR_ROW ROW-1
#define MINOR_COL COL-1

#define DIMENSION 3

#include "MatrixLib.h"

int A[ROW][COL] = {1,2,3,
                   0,1,4,
                   5,6,0};

int B[ROW][COL][MINOR_ROW][MINOR_COL];
int C[ROW][COL];
float D[ROW][COL];

```

```c

int main(){

    // input_matrix(A);
    printf("Elemen Matrix A : \n");
    print_matrix(ROW,COL,A);
    printf("determinan Matrix A : %0.2f\n\n",determinant_matrix(ROW,COL,A));
    minor_matrix(ROW,COL,B,A,DIMENSION);
    print_minor_matrix(ROW,COL,B);
    get_cofactor(ROW,COL,C,B);
    printf("cofactor matrix :\n");
    print_matrix(ROW,COL,C);
    printf("adjoint matrix :\n"); //adjoint is transpose of cofactor matrix
    transpose_matrix(ROW,COL,C,C);
    print_matrix(ROW,COL,C);
    printf("Invers matrix : \n");
    invers_matrix(ROW,COL,D,C,determinant_matrix(ROW,COL,A));
    print_float_matrix(ROW,COL,D);

    return 0;
}
```

Hasil Program

```
Elemen Matrix A :

1       2       3

0       1       4

5       6       0

determinan Matrix A : 1.00
```

```
Minor matrix [0][0] :
1 4
6 0

Minor matrix [0][1] :
0 4
5 0

Minor matrix [0][2] :
0 1
5 6

Minor matrix [1][0] :
2 3
6 0

Minor matrix [1][1] :
1 3
5 0
```

```
Minor matrix [1][2] :
1 2
5 6

Minor matrix [2][0] :
2 3
1 4

Minor matrix [2][1] :
1 3
0 4

Minor matrix [2][2] :
1 2
0 1
```

```
cofactor matrix :

-24     -20     -5

-18     -15     -4

5       4       1


adjoint matrix :

-24     -18     5

-20     -15     4

-5      -4      1


Invers matrix :

-24.00  -18.00  5.00

-20.00  -15.00  4.00

-5.00   -4.00   1.00
```

Eigen Value

Eigen value

```
static void eigen_val(int row, int col, int matrix[row][col]){

    float lamda[2];
    int matrix_temp[2][2];

    // solve using quadratic formula

    int a = 1;
    int b = -1 * (matrix[0][0] + matrix[1][1]);
    int c = (-1 * matrix[0][0] * -1 * matrix[1][1]) - (-1 * matrix[0][1] * -1 * matrix[1][0]);

    printf("%d\n",b);
    printf("%d\n",c);

    print_matrix(2,2,matrix);

    for(int x = 1; x <= 2; x++){
        lamda[x] = (-b + pow(-1,x) * sqrt(pow(b,2) - (4 * a *c)))/ (2 * a);
        printf("lamda [%d] : %0.2f\n",x,lamda[x]);
    }
}
```

Program 4 Modul 1

```c
#include "MatrixLib.h"

static int A[2][2] = {3,0,
                      8,-1};

int main(){

    eigen_val(2,2,A);

    return 0;
}
```

Hasil Program

```
3       0

8       -1


lamda [1] : -1.00
lamda [2] : 3.00
```

Mencari matriks hessian

Representasi Fungsi ke matriks

$$f(x, y) = x^3y^3 + 3x^2 + y^2$$

	x^0	x^1	x^2	x^3
y^0	0	0	3	0
y^1	0	0	0	0
y^2	1	0	0	0
y^3	0	0	0	1

Fungsi turunan terhadap x y

```
void get_differential_multivar(int row,int col, int diff_x, int diff_y, int function_input[row][col], int function_output[row][col]){

    int function_temp[row][col];

    if(diff_x != 0){
        for (int i = 0; i < row; i++){
            for (int j = 0; j < col; j++){
                int x = (diff_x != diff_y) ? (get_factorial(j)) : (j * diff_x);
                function_temp[i][j - diff_x] = function_input[i][j] * x;
            }
        }
    }
    else
        transfer_matrix(row,col,function_temp,function_input);

    if(diff_y != 0){
        for (int i = 0; i < row; i++){
            for (int j = 0; j < col; j++){
                int x = (diff_x != diff_y) ? (get_factorial(i)) : (i * diff_y);
                function_output[i - diff_y][j] = function_temp[i][j] * x;
            }
        }
    }
    else
        transfer_matrix(row,col,function_output,function_temp);

    for(int i = 0; i < row; i++){ // clean error value
        for(int j = 0; j < col; j++)
            if(i >= row - diff_y || j >= col - diff_x)
                function_output[i][j] = 0;
    }
}
```

Menghitung matriks hessian

```c
void hessian_matrix(int row, int col,int function_input[row][col]){

    int x = 0;

    print("\n");

    for (int i = 0; i < 2; i++){
        for (int j = 0; j < 2; j++){
            printf("Matrix hessian [%d][%d] : \n",i,j);
            get_differential_multivar(row,col,step[x][0],step[x][1],function_input,hessian_output[i][j].data);
            print_function_matrix(row,col,hessian_output[i][j].data);
            x++;
        }
    }
}
```

Program 1 Modul 2

```c
#define poly_x 3
#define poly_y 3

#define x_index poly_x + 1 //COL
#define y_index poly_y + 1 //ROW

#include "CalculusLib.h"

static int function_input[4][4] = {0,   0,  3,  0, // f(x,y) = x^3*y^3 + 3x^2 + y^2
                                   0,   0,  0,  0,
                                   1,   0,  0,  0,
                                   0,   0,  0,  1};

int function_output[4][4];

int main(){

    printf("Representasi fungsi polynomial dengan matrix :\n");
    print_function_matrix(y_index,x_index,function_input);
    printf("Hessian matrix : \n");
    hessian_matrix(y_index,x_index,function_input,2,-1);

    return 0;

}
```

Hasil program

```
Matrix hessian [0][0] :

6       0       0       0

0       0       0       0

0       0       0       0

0       6       0       0


Matrix hessian [0][1] :

0       0       0       0

0       0       0       0

0       0       9       0

0       0       0       0
```

$$\frac{\partial^2}{\partial x^2} \Rightarrow 6xy^3 + 6$$

$$\frac{\partial^2}{\partial x \partial y} \Rightarrow 9x^2y^2$$

Hasil program

```
Matrix hessian [1][0] :

0       0       0       0

0       0       0       0

0       0       9       0

0       0       0       0

Matrix hessian [1][1] :

2       0       0       0

0       0       0       6

0       0       0       0

0       0       0       0
```

$$\frac{\partial^2}{\partial y \partial x} \Rightarrow 9x^2y^2$$

$$\frac{\partial^2}{\partial y^2} \Rightarrow 6x^3y + 2$$

Terimakasih