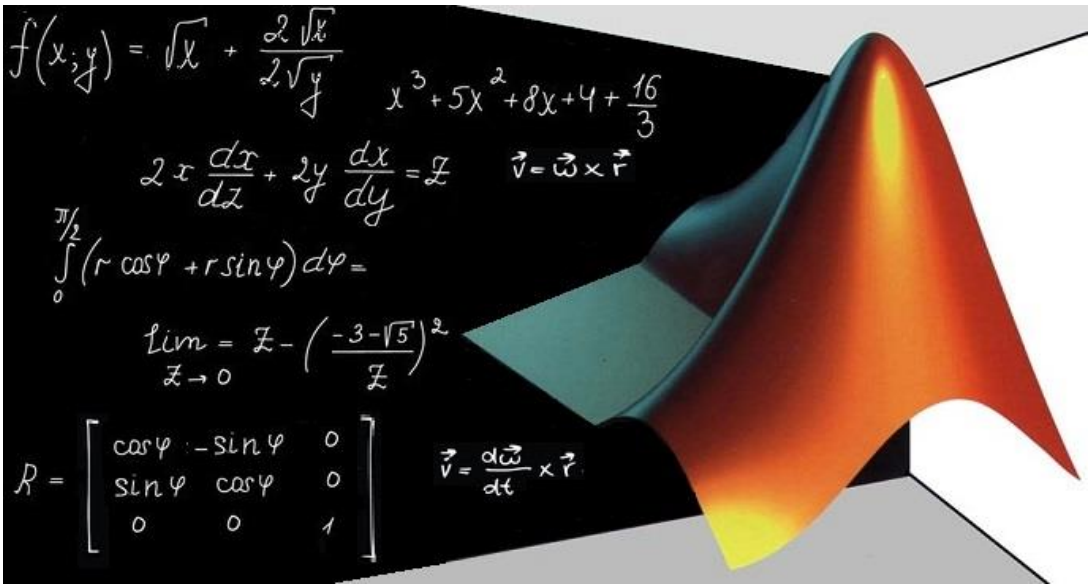
NUMERICAL METHODS

MATH2059

ASSIGNMENT REPORT 2



CAFER SAMET GÜLLÜCE – 150113081

UFUK GÜRBÜZ – 150113058

PROCEDURES

PROBLEM 1

We implemented function called ‘mygauss’ for using “Gaussian Elimination with Partial Pivoting” in MATLAB. This function gets ‘inputs an n-by-n matrix A and a vector b of size n' and returns ‘singular (0) or non-singular (1)’ and ‘outputs of the solution x’.

When you run ‘test’ script, program creates 4 matrices and call ‘mygauss’ function then you can see test matrices, vectors and results. If you prefer run ‘mygauss’ function directly, you should define an n-by-n matrix. Then, you should input a vector of size n. But, this vector must be vertical. Namely; when you define a normal vector, you should take transpose of that vector. After all, you can call my gauss function and you can see results of function.

Let’s examine these processes below:

```
>> test()
```

3-BY-3 MATRIX (1)

```
A =

    0.8173    0.3998    0.4314
    0.8687    0.2599    0.9106
    0.0844    0.8001    0.1818


b =

    0.2638
    0.1455
    0.1361


singular =

     0


x =

    0.3540
    0.1851
   -0.2307
```

4-BY-4 MATRIX (2)

```
A =

    0.8693    0.8530    0.4018    0.1839
    0.5797    0.6221    0.0760    0.2400
    0.5499    0.3510    0.2399    0.4173
    0.1450    0.5132    0.1233    0.0497


b =

    0.9027
    0.9448
    0.4909
    0.4893


singular =

     0


x =

    0.4822
    1.0872
   -1.2673
    0.3552
```

5-BY-5 MATRIX (3)

```
A =

    0.3377    0.3897    0.9421    0.3532    0.6491
    0.9001    0.2417    0.9561    0.8212    0.7317
    0.3692    0.4039    0.5752    0.0154    0.6477
    0.1112    0.0965    0.0598    0.0430    0.4509
    0.7803    0.1320    0.2348    0.1690    0.5470


b =

    0.2963
    0.7447
    0.1890
    0.6868
    0.1835


singular =

     0


x =

   -0.9971
    2.3923
   -2.2394
    2.7522
    1.2915
```

NO-SOLUTION MATRIX (4)

```
A =

     1     2     0
     4     5     0
     7     8     0


b =

     1
     2
     3


singular =

     1


x =

No solution !
```

INCORRECT INPUT MATRIX (5)

```
>> matrix=[22 32 18 15; 13 22 19 7; 24 3 33 4];  % Defining a 3-by-4 matrix
>> vector=[82 68 74 16]';  % Defining a vector of size 4 and taking transpose
>> [singular,x] = mygauss(matrix,vector)  % Calling mygauss function
Matrix is not n-by-n!

singular =

     1


x =

There is not solution !
```

INCORRECT INPUT VECTOR (6)

```
>> matrix=[22 32 18 15; 13 22 19 7; 24 3 33 4; 16 2 24 26];  % Defining a 4-by-4 matrix
>> vector=[82 68 74]';  % Defining a vector of size 3 and taking transpose
>> [singular,x] = mygauss(matrix,vector)  % Calling mygauss function
b vector is not size of n!

singular =

     1


x =

There is not solution !
```

PROBLEM 2

We wrote a script file called ‘myspline’ which generates 8 data points with x values 1 to 8 and random y values between 0 and 1. Then, it plots the interpolating polynomial of degree 7. We used ‘mygauss’ function for solving system.

We implemented a algorithm also as a separate function called ‘mycubicspline’. This algorithm finds and returns the coefficients of the cubic spline. Then, we drew its graphic with curve of interpolation polynomial and showed together them in the graphic.

For running ‘myspline’ method, you should call ‘myspline’ method. It starts running and it runs ‘mycubicspline’ method itself.

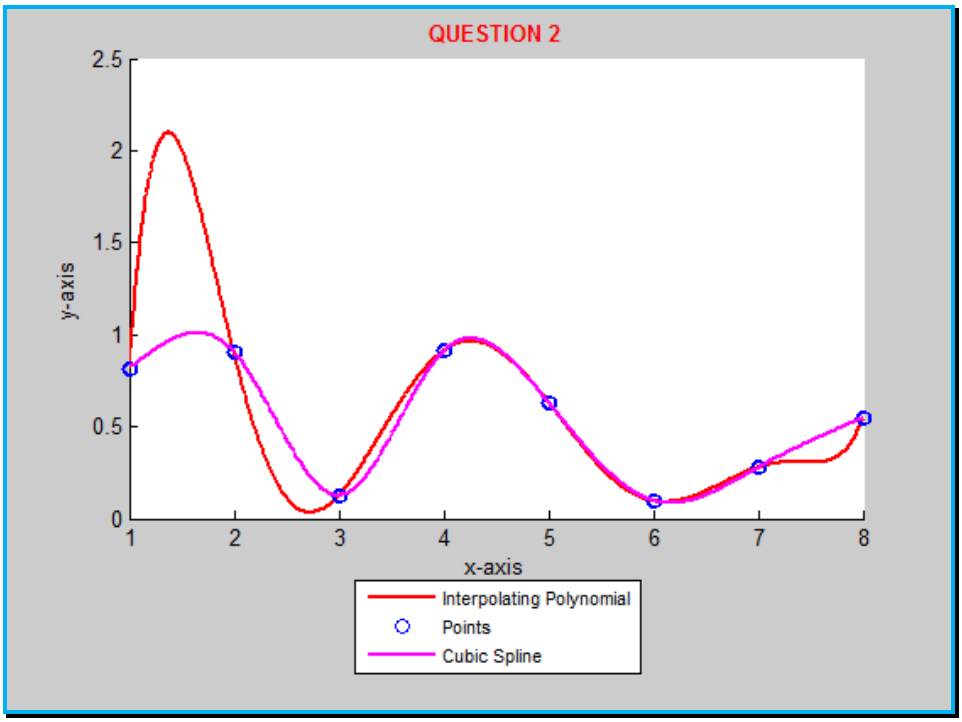
Let’s examine these processes below:

EXAMPLE (1)

```
>> myspline()

yValues =

    0.8147    0.9058    0.1270    0.9134    0.6324    0.0975    0.2785    0.5469
```

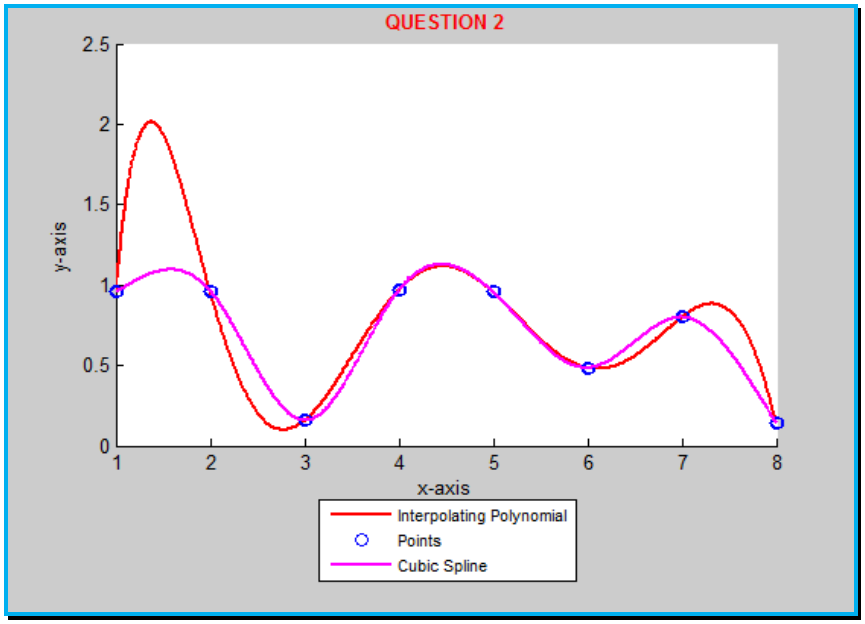


EXAMPLE (2)

```
>> myspline()

yValues =

    0.9575    0.9649    0.1576    0.9706    0.9572    0.4854    0.8003    0.1419
```

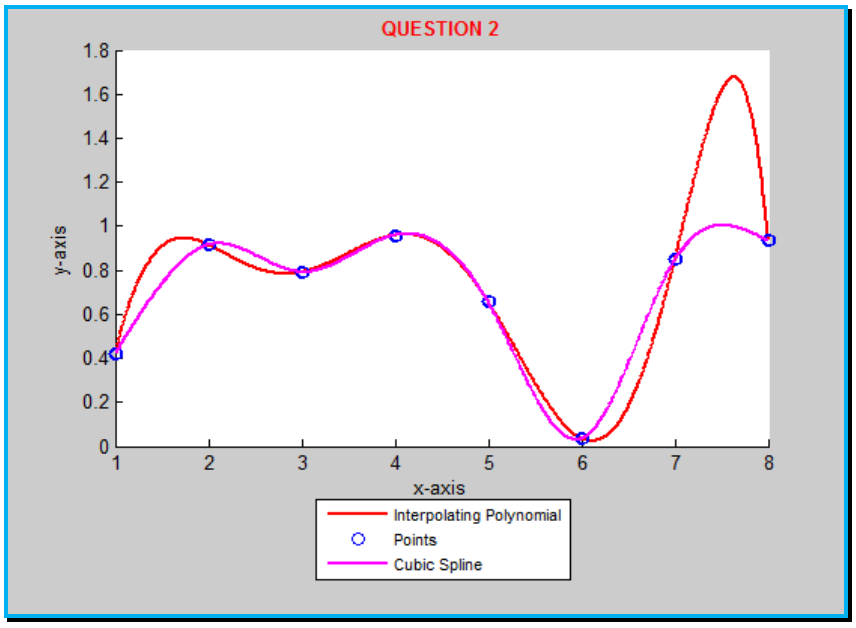


EXAMPLE (3)

```
>> myspline()

yValues =

    0.4218    0.9157    0.7922    0.9595    0.6557    0.0357    0.8491    0.9340
```

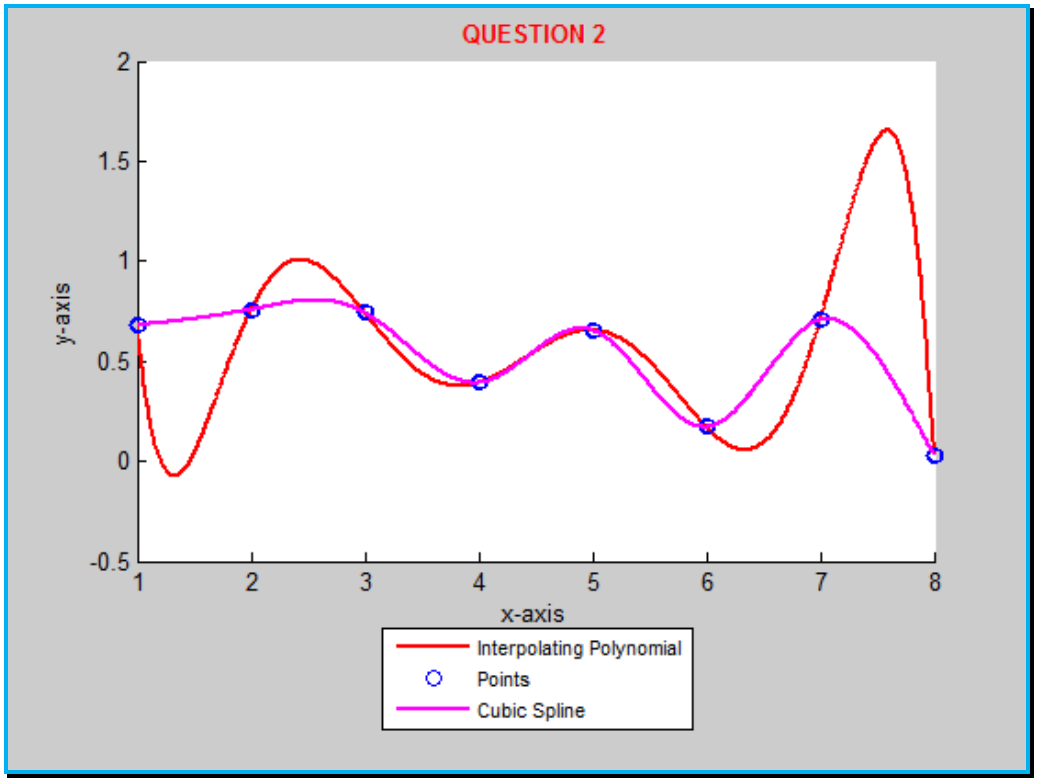


EXAMPLE (4)

```
>> myspline()

yValues =

    0.6787    0.7577    0.7431    0.3922    0.6555    0.1712    0.7060    0.0318
```



PROBLEM 3

We implemented a natural boundary cubic spline script called ‘mycurve’. When you call that script, it shows a empty screen and you can click many point. Then, it combines these points and creates curve of cubic spline.

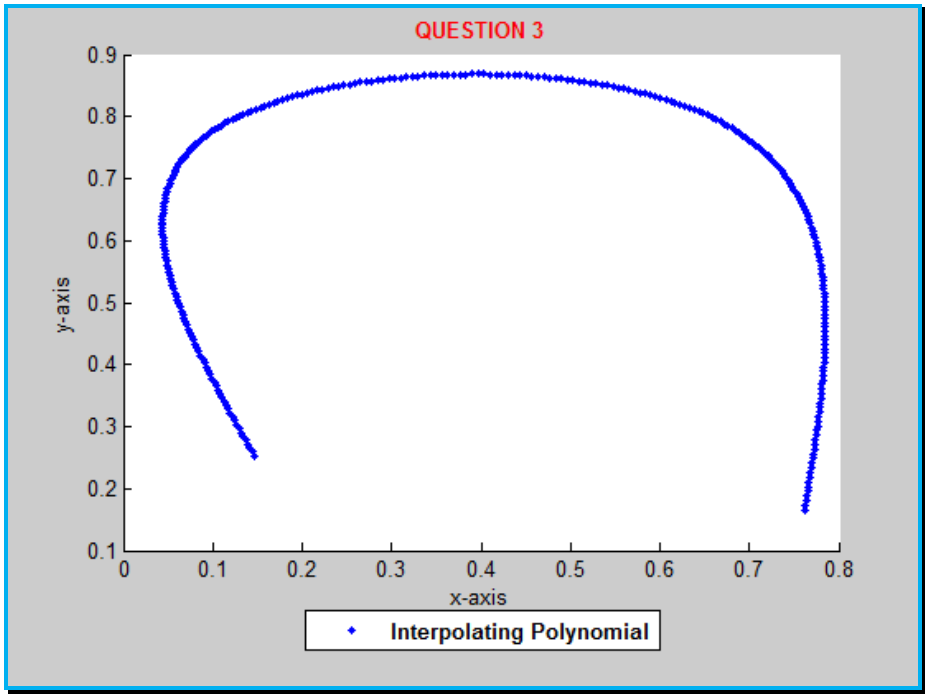
This script uses also ‘mycubicspline’ algorithm for finding the coefficients of the cubic spline. Then, it draws a cubic spline and shows it.

For running, you should input ‘n’ number. It is number of points in the cubic spline. Then, you should call ‘mycurve’ script and should click the screen according to value of ‘n’.

Let’s examine these processes below:

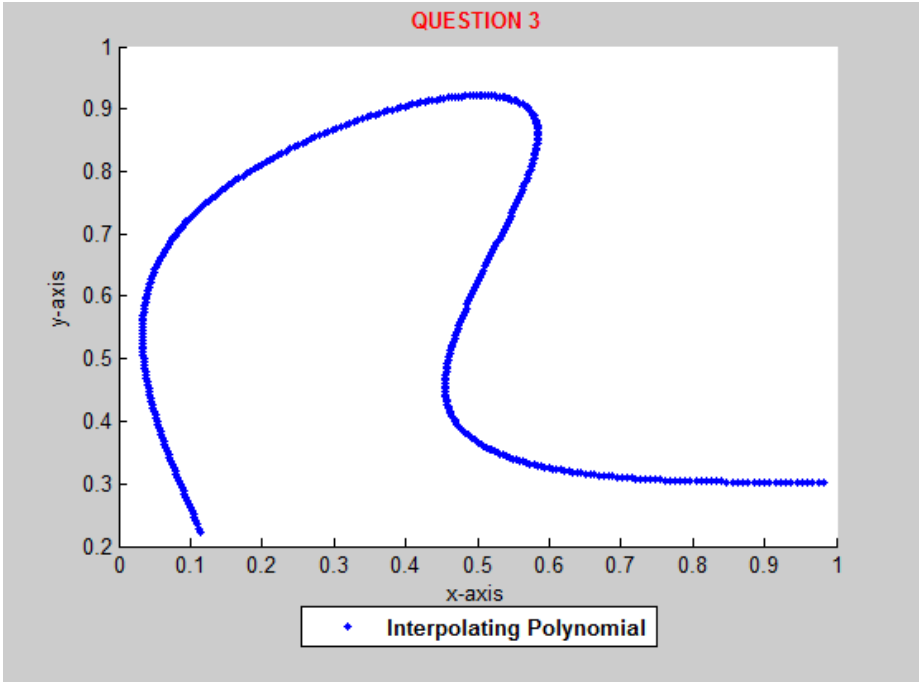
4 POINTS - EXAMPLE (1)

```
>> n = 4; % Numbers of points on the screen
>> mycurve() % Calling method
The X Coordinates are: [0.14631      0.10253      0.67627      0.76152]
The Y Coordinates are: [0.25292      0.77924      0.78509       0.1652]
```



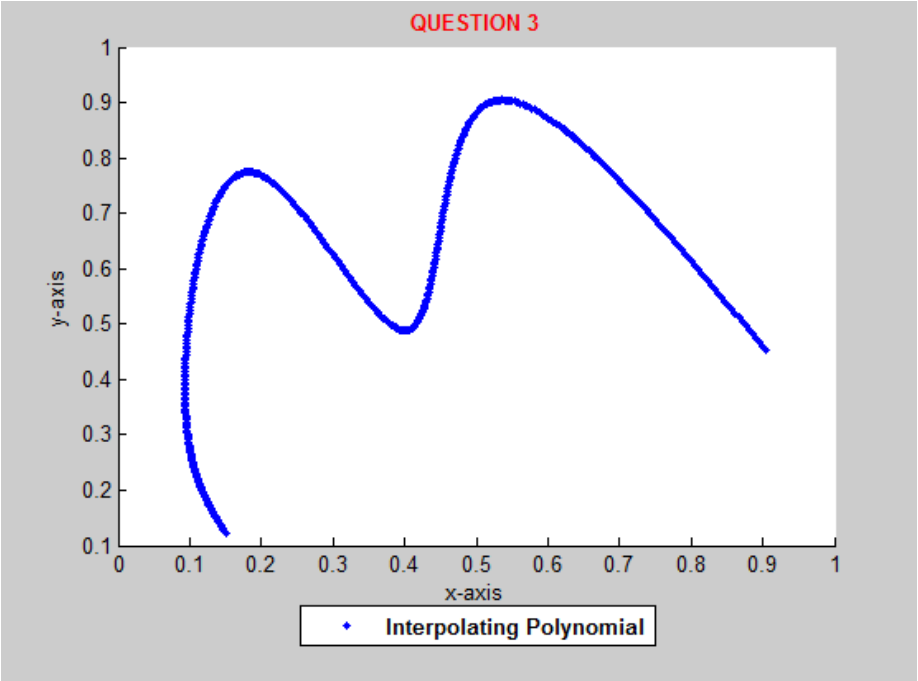
5 POINTS - EXAMPLE (2)

```
>> n = 5; % Numbers of points on the screen
>> mycurve() % Calling method
The X Coordinates are: [0.11406     0.097926      0.57488      0.45737      0.98272]
The Y Coordinates are: [0.22368      0.72368       0.8962      0.43129      0.30263]
```



6 POINTS - EXAMPLE (3)

```
>> n = 6; % Numbers of points on the screen
>> mycurve() % Calling method
The X Coordinates are: [0.15092    0.093318     0.19009     0.40668     0.51728     0.90438]
The Y Coordinates are: [0.12135     0.39035     0.77339     0.48977     0.89912     0.45175]
```



7 POINTS - EXAMPLE (4)

```
>> n = 7; % Numbers of points on the screen
>> mycurve() % Calling method
The X Coordinates are: [0.27995     0.12327     0.34217      0.1394      0.4712     0.48733     0.89747]
The Y Coordinates are: [0.033626     0.17982     0.34064     0.64181     0.86111     0.51316     0.58333]
```