**ECSE 443 - Assignment 3**

\*\*\*For the following questions, I used the data included in the assignment document. The data was treated as 2 different function to handle the discontinuity located at x = 1. Therefore, I put each equation into its own .txt files (‘Ass\_3\_data\_funct1’, ‘Ass\_3\_funct2’).

**Question 1**

\*\* For the following parts of I used the function 1 associated with the data points before x = 1.

1. Refer to Appendix A, part a, for the corresponding Matlab code.

In order to find the we will use the points [0,1], [0.2,0.916], [0.4, 0.836]. Therefore, we can denote these polynomials by:

Using the points, we interpolate the following functions:

The first derivative and second derivative of the interpolating function will be continuous at .

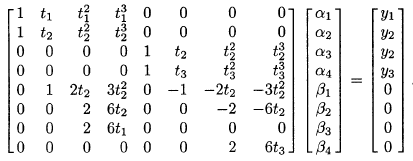




The endpoints of the seconds derivative give the following:



This will result in 4 more functions that will allow us to build the following array.



Using the following inputs, we can calculate the unknowns.

The unknowns are calculated to be:

vars = 1×8

1.000000000000000 -0.422500000000000 0 0.062499999999995 1.001000000000000 -0.437499999999999 0.074999999999996 -0.062499999999995

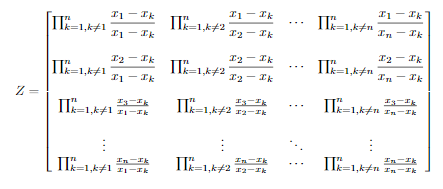
|  |  |
| --- | --- |
|  | 1 |
|  | -0.4225 |
|  | 0 |
|  | 0. 062499999999995 |
|  | 1.001 |
|  | -0. 437499999999999 |
|  | 0. 074999999999996 |
|  | -0. 062499999999995 |

With the calculate variables we find the following piecewise function:

To find we will us the case where and we calculate the following.

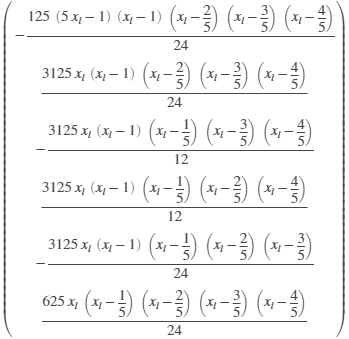
Using the built in spline method in Matlab the answer was found to be similar.

1. Refer to Appendix A, part b and Appendix C, part a, for the corresponding Matlab code since there is a function in Appendix B to the function used. To use Lagrange Polynomial interpolation the following matrix/system of equations is constructed and solved for the unknowns:



The Lagrange Polynomials and the interpolating polynomial were found to be:

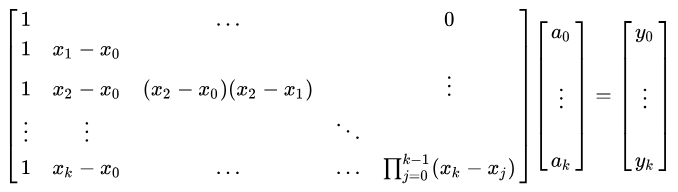
L\_coeffs =



f =  C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1313595453908254150\image15525778783790.png

Then, we plug the point 0.23 to find .

1. Refer to Appendix A, part c and Appendix C, part b, for the corresponding Matlab code since there is a function in Appendix B to the function used. To use Newton’s Polynomial interpolation the following matrix/system of equations is constructed and solved for the unknowns:



The matrices are calculated to be the following:

x\_mat = 6×6

1.0000 0 0 0 0 0

1.0000 0.2000 0 0 0 0

1.0000 0.4000 0.0800 0 0 0

1.0000 0.6000 0.2400 0.0480 0 0

1.0000 0.8000 0.4800 0.1920 0.0384 0

1.0000 1.0000 0.8000 0.4800 0.1920 0.0384

y\_mat = 6×1

1.0000

0.9160

0.8360

0.7400

0.6240

0.4000

We solve this matrix to find the unknowns.

|  |  |
| --- | --- |
|  | 1.0000 |
|  | -0.4200 |
|  | 0.5000 |
|  | -0.4167 |
|  | 0.4167 |
|  | -2.6042 |

So, the corresponding function to the unknowns is found and then we input :

f(t)=   C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1313595453908254150\image15525109254350.png

Simplified function:

f\_N(t)= C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1313595453908254150\image15525110633070.png

**Question 2**

\*\* For the following parts of I used the function 1 associated with the data points before x = 1.

1. Refer to Appendix B, part d, for the corresponding Matlab code.

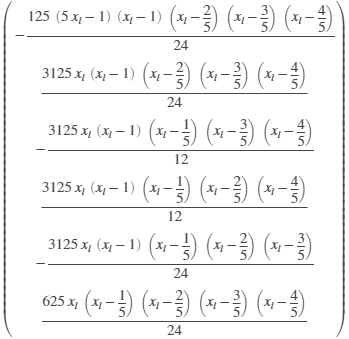
In order to find the we will use the points [0.6,0.74], [0.8,0.624], [1, 0.4]. Therefore, as we saw in question 1, a, we noticed that we can use the built-in spline method in Matlab to calculate the cubic splines.

Using the built in spline method in Matlab the answer was found to be similar.

1. Refer to Appendix B, part b and Appendix C, part a, for the corresponding Matlab code since there is a function in Appendix B to the function used. To calculate the Lagrange Polynomial, we followed the same procedure as in question 1, b.

The Lagrange Polynomials and the interpolating polynomial were found to be:

L\_coeffs =



f =  C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1313595453908254150\image15525778783790.png

Then, we plug the point 0.78 to find .

1. Refer to Appendix B, part c and Appendix C, part b, for the corresponding Matlab code since there is a function in Appendix B to the function used. To calculate Newton’s Polynomial interpolation, we followed the same procedure as question 1, c.

The matrices are calculated to be the following:

x\_mat = 6×6

1.0000 0 0 0 0 0

1.0000 0.2000 0 0 0 0

1.0000 0.4000 0.0800 0 0 0

1.0000 0.6000 0.2400 0.0480 0 0

1.0000 0.8000 0.4800 0.1920 0.0384 0

1.0000 1.0000 0.8000 0.4800 0.1920 0.0384

y\_mat = 6×1

1.0000

0.9160

0.8360

0.7400

0.6240

0.4000

We solve this matrix to find the unknowns.

|  |  |
| --- | --- |
|  | 1.0000 |
|  | -0.4200 |
|  | 0.5000 |
|  | -0.4167 |
|  | 0.4167 |
|  | -2.6042 |

So, the corresponding function to the unknowns is found and then we input :

f\_N(t)= C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1313595453908254150\image15525110633070.png

**Question 3**

\*\* For the following question, I used the function 2 associated with the data points after x = 1.

**Appendix**

**Appendix A – Question 1 Matlab Code**

Question 1 - part a - Cubic Spline Interpolation

format long

syms f(t) f\_N(t)

% system of equations matrix

x = [1 0 0 0 0 0 0 0; 1 0.2 0.04 0.008 0 0 0 0; 0 0 0 0 1 0.2 0.04 0.008; 0 0 0 0 1 0.4 0.16 0.064; 0 1 0.4 0.12 0 -1 -0.4 -0.12; 0 0 2 1.2 0 0 -2 -1.2; 0 0 2 0 0 0 0 0; 0 0 0 0 0 0 2 2.4];

y = [1; 0.916; 0.916; 0.834; 0; 0; 0; 0];

% calculate the unknowns matrix

vars = inv(x)\*y

vars = 8×1

1.000000000000000  
 -0.422500000000000  
 0  
 0.062499999999995  
 1.001000000000000  
 -0.437499999999999  
 0.074999999999996  
 -0.062499999999995

% since our point is in the range of [0.2, 0.4] then we use the following

f(t) = vars(5) + vars(6)\*t + vars(7)\*t^2 + vars(8)\*t^3;

vpa(f(t))

ans = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1313595453908254150\image15525804623200.png

% calculate the unknown point 0.23

x\_unkn = 0.23;

double(f(x\_unkn))

ans =

0.903582062500000

% compare the above method with the builtin spline method

data = importdata(['Ass\_3\_data\_funct1.txt']);

% use the same 3 points

x = data(1:3,1);

y = data(1:3,2);

% calculate the unknown point 0.23

spline(x,y, x\_unkn)

ans =

0.903745000000000

part b

% use the same data as above

% transfer the data to its x,y coordinates

x = data(:,1);

y = data(:,2);

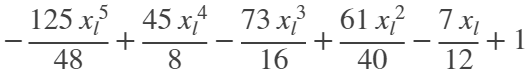
L\_coeffs = sym(ones(length(x), 1));

syms x\_l

% call the Lagrange polynomial function to get the function polynomial

f = LagrangePolynomial(x\_l, x, y)

f =



vpa(subs(f, x\_l, x\_unkn))

ans = 0.90505882109375

part c - Newton's Polynomial

% use the same data as above

% transfer the data to its x,y coordinates

% use the same data as above

% transfer the data to its x,y coordinates

x = data(:,1);

y = data(:,2);

% call the newtons polynomial funct

vars = NewtonsPolynomial(x, y)

vars = 6×1

1.000000000000000  
 -0.420000000000000  
 0.049999999999997  
 -0.416666666666658  
 0.416666666666655  
 -2.604166666666694

% plug points into the corresponding function

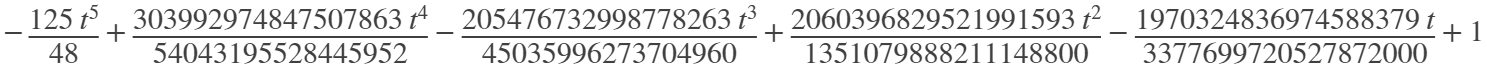
f\_N(t) = vars(1) + vars(2)\*(t-x(1)) + vars(3)\*(t-x(1))\*(t-x(2)) + vars(4)\*(t-x(1))\*(t-x(2))\*(t-x(3)) + vars(5)\*(t-x(1))\*(t-x(2))\*(t-x(3))\*(t-x(4)) + vars(6)\*(t-x(1))\*(t-x(2))\*(t-x(3))\*(t-x(4))\*(t-x(5));

vpa(f\_N(t), 5)

ans = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1313595453908254150\image15525804627383.png

f\_N(t) = simplify(f\_N(t))

f\_N(t) =



double(f\_N(x\_unkn))

ans =

0.905058821093750

**Appendix B – Question 2 Matlab Code**

Question 2 - part a - Cubic Spline Interpolation

format long

syms f(t) f\_N(t)

% calculate the unknown point 0.78

x\_unkn = 0.78;

% knowing that the spline methods returns the same results we will use it

% with points surrounding the point

data = importdata(['Ass\_3\_data\_funct1.txt']);

% use the same 3 points

x = data(4:6,1)

x = 3×1

0.600000000000000  
 0.800000000000000  
 1.000000000000000

y = data(4:6,2)

y = 3×1

0.740000000000000  
 0.624000000000000  
 0.400000000000000

% calculate the unknown point 0.78

spline(x,y, x\_unkn)

ans =

0.640460000000000

part b

% use the same data as above

% transfer the data to its x,y coordinates

x = data(:,1);

y = data(:,2);

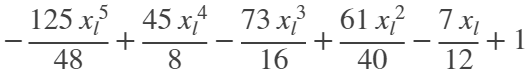
L\_coeffs = sym(ones(length(x), 1));

syms x\_l

% call the Lagrange polynomial function to get the function polynomial

f = LagrangePolynomial(x\_l, x, y)

f =



vpa(subs(f, x\_l, x\_unkn))

ans = 0.637895075

part c - Newton's Polynomial

% use the same data as above

% transfer the data to its x,y coordinates

% use the same data as above

% transfer the data to its x,y coordinates

x = data(:,1);

y = data(:,2);

% call the newtons polynomial funct

vars = NewtonsPolynomial(x, y)

vars = 6×1

1.000000000000000  
 -0.420000000000000  
 0.049999999999997  
 -0.416666666666658  
 0.416666666666655  
 -2.604166666666694

% plug points into the corresponding function

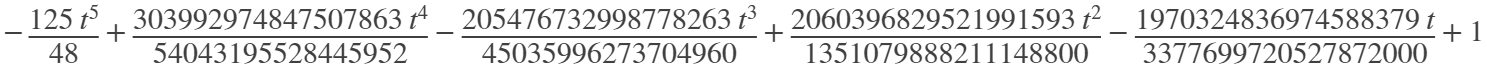
f\_N(t) = vars(1) + vars(2)\*(t-x(1)) + vars(3)\*(t-x(1))\*(t-x(2)) + vars(4)\*(t-x(1))\*(t-x(2))\*(t-x(3)) + vars(5)\*(t-x(1))\*(t-x(2))\*(t-x(3))\*(t-x(4)) + vars(6)\*(t-x(1))\*(t-x(2))\*(t-x(3))\*(t-x(4))\*(t-x(5));

vpa(f\_N(t), 5)

ans = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1313595453908254150\image15525805129642.png

f\_N(t) = simplify(f\_N(t))

f\_N(t) =



double(f\_N(x\_unkn))

ans =

0.637895074999998

**Appendix C – Question 1 & 2 Matlab Functions**

**PART A – Lagrange Polynomial Function**

Function to Calculate the Lagrange Polynomial Interpolation.

inputs: x\_l: symbolic variable that will be used

x: x points

y: y points

return: polynomial function

function f = LagrangePolynomial(x\_l, x, y)

L\_coeffs = sym(ones(length(x),1));

syms x\_l

% find all the lagrange polynomials associated with the data

for i=1:length(x)

temp = 1;

for j=1:length(x)

if j ~= i

temp = temp \* (x\_l - x(j))/(x(i)-x(j));

end

end

L\_coeffs(i) = temp;

end

L\_coeffs

f = 0;

for i=1:length(x)

temp = y(i)\*L\_coeffs(i);

f = f + temp;

end

f = simplify(f);

end

**PART B – Lagrange Polynomial Function**

Function to Calculate the Newton's Polynomial Interpolation.

inputs: x: x points

y: y points

return: coefficients of the newtons polynomial

function f = NewtonsPolynomial(x, y)

% initialize matrix of the data size

x\_mat = zeros(length(x));

y\_mat = zeros(length(x),1);

sz = length(x\_mat);

for i=1:sz

x\_mat(i, 1) = 1;

y\_mat(i) = y(i);

for j=2:sz

if i >= j

temp = 1;

% find the array points val

for k=1:j-1

temp = temp \* (x(i)-x(k));

end

x\_mat(i,j) = temp;

end

end

end

% calculate the unknowns

f = inv(x\_mat)\*y\_mat;

end

**Appendix C – Question 3 Matlab Code**