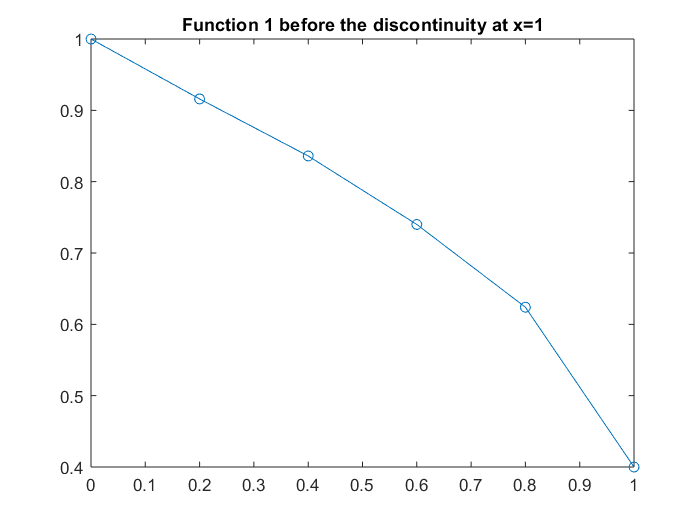
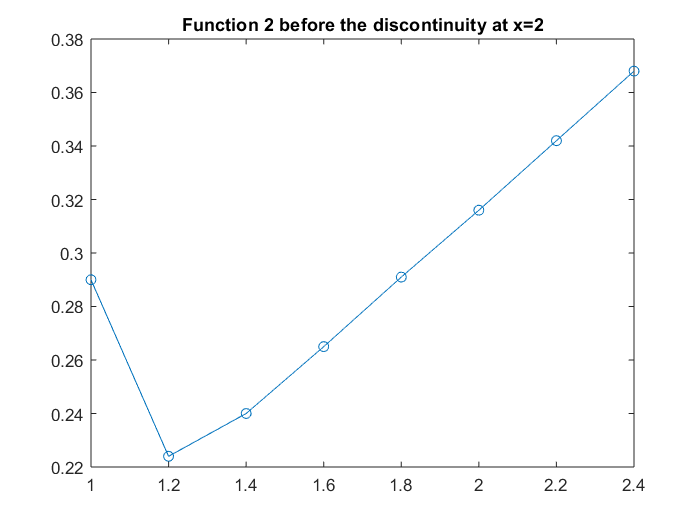
**ECSE 443 - Assignment 3**

\*\*\*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’).

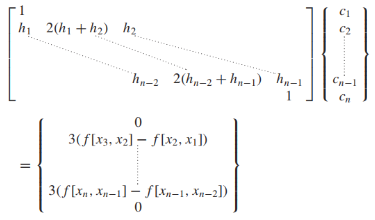
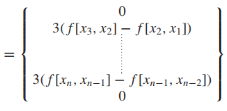
 

**Question 1**

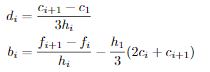
\*\* For the following parts of I used the function 1 associated with the data points before x = 1.

1. Refer to Appendix A, part a and Appendix C, part a, for the corresponding Matlab code and the cubic splines equation that was written.

In order to find the we will use all the points. Therefore, we can find the coefficients by constructing the following system of equations/matrices:

Using the following functions, the cofficients for the polynomial were found.



|  |  |
| --- | --- |
|  | 1 |
|  | -0.431770334928229 |
|  | 0 |
|  | 0.294258373205738 |
|  | 0.916 |
|  | -0.396459330143541 |
|  | 0.176555023923443 |
|  | -0.971291866028701 |

With the calculate variables we find the following piecewise function:

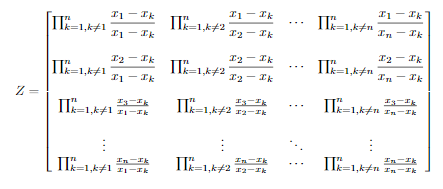
f = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1569063244162771305\image15528375064170.png

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

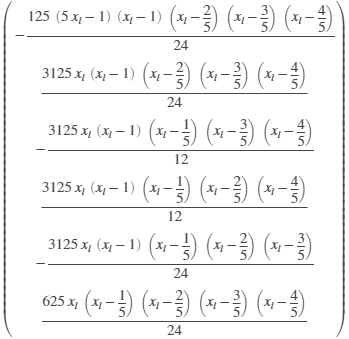
Using the built in spline method in Matlab the answer was found to be similar.

In other cases, it is better to use the cubic spline along all points rather than just 3 points to get a better value. This is due to the fact that there would be more points allowing for a better fit of the function which would lead to a closer more precise answer.

1. Refer to Appendix A, part b and Appendix C, part b, 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:



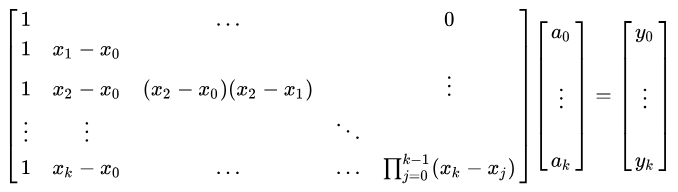
The Lagrange Polynomials and the interpolating polynomial were found to be:

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 c, 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

The results of the Newton’s Polynomial and Lagrange Polynomial are both exactly the same because they work up to the 5th order polynomial which looks much greater than order of the function 1. Therefore, the use of these interpolation methods is overkill on the type of method as they are more complex then our given function points. That is why the functions are able to produce the same results because of their higher level of complexity.

**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 and Appendix C, part a, for the corresponding Matlab code and the cubic splines equation that was written.

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 followed the same procedure as in question 1 to calculate the polynomial. The matrices were found to be:

A = 3×3

1.000000000000000 0 0

0.200000000000000 0.800000000000000 0.200000000000000

0 0 1.000000000000000

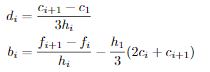
B = 3×1

0

-1.620000000000001

0

Using the following functions, the cofficients for the polynomial were found.



|  |  |
| --- | --- |
|  | 0.74 |
|  | -0.445 |
|  | 0 |
|  | -3.375 |
|  | 0.624 |
|  | -0.85 |
|  | -2.025 |
|  | 3.375 |

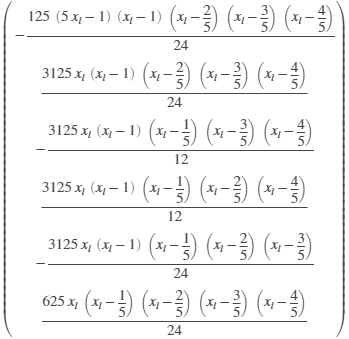
With the calculate variables we find the following piecewise function:

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

In other cases, it may have been better to use the cubic spline along all points rather than just 3 points to get a better value. This is due to the fact that there would be more points allowing for a better fit of the function which would lead to a closer more precise answer.

1. Refer to Appendix B, part b and Appendix C, part b, 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 c, 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

The results of the Newton’s Polynomial and Lagrange Polynomial are both the same because they work up to the 5th order polynomial which looks much greater than order of the function 2 (which seems linear). Therefore, the use of these interpolation methods is overkill on the type of method as they are more complex then our given function points. That is why the functions are able to produce the same results because of their higher level of complexity.

**Question 3**

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

Refer to the Appendix D and Appendix C, part A for the Matlab code and the function used to calculate the cubic spline. Similarly, to what was done in Question 1a and 2a, we used the cubic spline method on the set of three points to get a function. However, since we are extrapolating a point not on the curve, we are using a the last 3 given points of function 2.

In order to find the we will use the points [2,0.316], [2.2,0.342], [2.4, 0.368]. The matrices were found to be:

A = 3×3

1.000000000000000 0 0

0.200000000000000 0.800000000000000 0.200000000000000

0 0 1.000000000000000

B = 3×1

0

0.059999999999998

0

With the calculate variables we find the following function:

f = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1569063244162771305\image15527582746220.png

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

The answer found for this case was used with 3 points. Although, it may be possible to approximate a better result with more points, the use of three points is a close approximation to the extrapolated value and one that requires less computation.

**Appendix**

**Appendix A – Question 1 Matlab Code**

Question 1 - part a - Cubic Spline Interpolation

format long

syms f\_N(t) t\_c

% import the data in

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

% set variable for the unknown x

x\_unkn = 0.23;

% data points that we will use for the splines

x\_cs = data(1:3,1);

y\_cs = data(1:3,2);

f = CubicSplineInter(t\_c, x\_cs, y\_cs, 1);

A = 3×3

1.000000000000000 0 0  
 0.200000000000000 0.800000000000000 0.200000000000000  
 0 0 1.000000000000000

B = 3×1

0  
 0.059999999999998  
 0

c = 3×1

0  
 0.074999999999998  
 0

d = 2×1

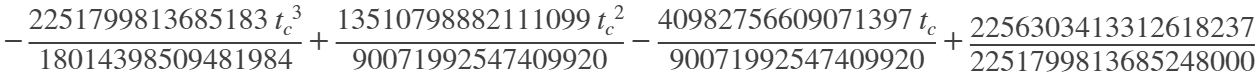
0.124999999999996  
 -0.124999999999996

b = 2×1

-0.425000000000000  
 -0.410000000000000

f = expand(f)

f =



double(subs(f, t\_c, x\_unkn))

ans =

0.903764125000000

spline(x\_cs, y\_cs, 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);

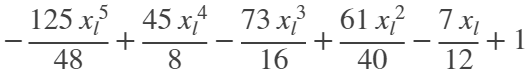
L\_coeffs = sym(ones(length(x), 1));

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

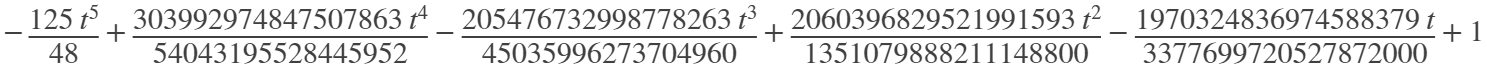
f\_N(t) = vars(1) + vars(2)\*(t-x(1)) + vars(3)\*(t-x(1))\*(t-x(2)) + vars(4)\*(t-x(1))\*(t-x(2))\*(t-x(3)) + vars(5)\*(t-x(1))\*(t-x(2))\*(t-x(3))\*(t-x(4)) + vars(6)\*(t-x(1))\*(t-x(2))\*(t-x(3))\*(t-x(4))\*(t-x(5));

vpa(f\_N(t), 5)

ans = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1569063244162771305\image15527575568983.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\_N(t) t\_c

% import the data

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

% calculate the unknown point 0.78

x\_unkn = 0.78;

% data points that we will use for the splines

x\_cs = data(4:6,1);

y\_cs = data(4:6,2);

% calculate the cubic spline and our unknown point is in between the second

% set of interval [0.8,1]

f = CubicSplineInter(t\_c, x\_cs, y\_cs, 0);

A = 3×3

1.000000000000000 0 0  
 0.200000000000000 0.800000000000000 0.200000000000000  
 0 0 1.000000000000000

B = 3×1

0  
 -1.620000000000001  
 0

c = 3×1

0  
 -2.025000000000001  
 0

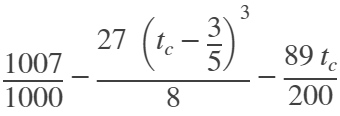
d = 2×1

-3.375000000000001  
 3.375000000000003

b = 2×1

-0.445000000000000  
 -0.850000000000000

f =



f = vpa(expand(f))

f = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1569063244162771305\image15527576183981.png

double(subs(f, t\_c, x\_unkn))

ans =

0.640217000000000

% compare with the built in spline equation

spline(x\_cs, y\_cs, 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);

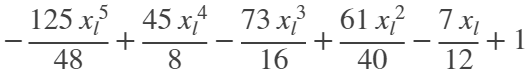
L\_coeffs = sym(ones(length(x), 1));

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

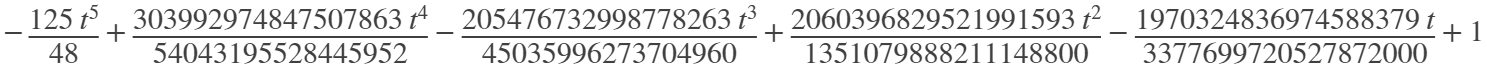
f\_N(t) = vars(1) + vars(2)\*(t-x(1)) + vars(3)\*(t-x(1))\*(t-x(2)) + vars(4)\*(t-x(1))\*(t-x(2))\*(t-x(3)) + vars(5)\*(t-x(1))\*(t-x(2))\*(t-x(3))\*(t-x(4)) + vars(6)\*(t-x(1))\*(t-x(2))\*(t-x(3))\*(t-x(4))\*(t-x(5));

vpa(f\_N(t), 5)

ans = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1569063244162771305\image15527576188764.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 – Cubic Splines Interpolation Function**

Function to Calculate the Cubic Spline Interpolation.

inputs: t\_c: symbolic variable that will be used

x: x points (3-pts)

y: y points (3-pts)

toggle: int ( 0 or 1 ) that will trigger whether to return the function bounded by first 2 points or the last 2 points

return: polynomial function

function f = CubicSplineInter(t\_c, x, y, toggle)

% find all the h step values

h = zeros(length(y)-1, 1);

for i=1:length(h)

h(i) = x(i+1)-x(i);

end

% create the A and B matrix for the system of equations

A = [1 0 0; h(1) 2\*(h(1)+h(2)) h(2); 0 0 1]

B = [0; (3\*((y(3)-y(2))/h(2))-3\*((y(2)-y(1))/h(1))); 0]

% calculate the unknown c variable

c = inv(A)\*B

% initialize the b,d variables and calculate them with the given equations

b = zeros(2, 1);

d = zeros(2,1);

for i=1:2

b(i) = (y(i+1)-y(i))/h(i) - h(i)\*(2\*c(i) + c(i+1))/3;

d(i) = (c(i+1)-c(i))/(3\*h(i));

end

d

b

% determine whether to return the function corresponding to the first

% interval or the second interval

if toggle == 0

f = y(1) + b(1)\*(t\_c-x(1)) + c(1)\*(t\_c-x(1))^2 + d(1)\*(t\_c-x(1))^3

elseif toggle == 1

f = y(2) + b(2)\*(t\_c-x(2)) + c(2)\*(t\_c-x(2))^2 + d(2)\*(t\_c-x(2))^3;

end

end

**PART B – 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 C – Newton 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 D – Question 3 Matlab Code**

Question 3 - Cubic Spline extrapolation

format long

syms t\_c

% import the data

data = importdata('Ass\_3\_data\_funct2.txt');

% split the data to get the desired points

x = data(6:8,1);

y = data(6:8, 2);

% call the cubic spline function and get the function correspoing to the

% second interval

f = CubicSplineInter(t\_c,x,y,1);

A = 3×3

1.000000000000000 0 0  
 0.200000000000000 0.800000000000000 0.200000000000000  
 0 0 1.000000000000000

B = 3×1

0  
 0  
 0

c = 3×1

0  
 0  
 0

d = 2×1

0  
 0

b = 2×1

0.130000000000000  
 0.130000000000000

f = vpa(f)

f = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard1569063244162771305\image15527625106870.png

subs(f, t\_c, 3)

ans = 0.446

**Appendix E – Alternative Spline method for 3 points, Results and Matlab code**

**PART A Q1 a – Using 3 pts for spline**

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.