

High 57
low 20
Mean: 40
Median: 42

53/60

use more sentences to describe the answers.

Homework 2

Math 128A Numerical Analysis

10/28/2008

1

2/20 Given the points (x_i, y_i) for $i = 0 \dots n$ the Newton form of the interpolating polynomial is $p(x) = a_0 + a_1(x - x_0) + \dots + a_n(x - x_0)(x - x_1) \dots (x - x_{n-1})$

(a) Write a routine that takes arrays of data x and y and returns an array, a , containing the coefficients of the Newton form of the interpolating polynomial.

```
function [a] = newton(x,y);

    n = length(x);
    F = zeros(n,n+2);

    % if x and y are not matched exit
    if(n ~= length(y))
        sprintf('%s', '(x,y) pair not equal');
        a = 0;
        return;
    end

    %Put x and y in col 1 and 2 of F
    for i = 1:n
        F(i,1) = x(i,1);
        F(i,2) = y(i,1);
    end

    %calculate coefficients using Newtons method
    for i = 2:n
        for j = 2:i
            num = F(i,j-1+1) - F(i-1,j);
            den = x(i,1) - x(i-j+1,1); %compute xi-xi-j
            F(i,j+1) = num/den;
            F(i,n+2) = i;
        end
    end

    %Just return a vector of the coefficients
    a = diag(F(:,2:n+1));
end
```

20
16
4
10
5 3

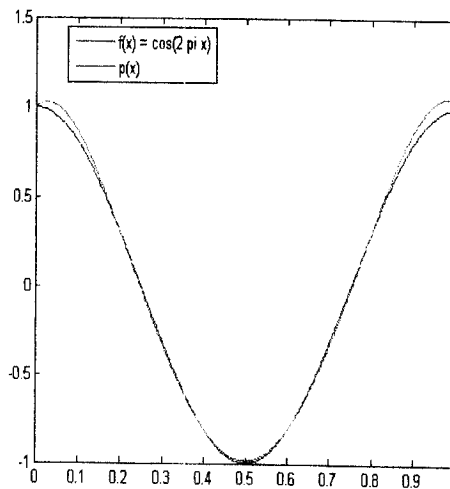
- (b) Write a routine to evaluate the interpolating polynomial at a point using Horner's method (nested multiplication). Your routine will take as input the arrays x and a and the scalar z , and it will return $p(z)$.

```
function [p] = horners2(x,a,z);  
    n = length(a);  
    p = a(n);  
  
    for i = n-1:-1:1  
        p = a(i) + p*(z-x(i));  
    end  
end
```

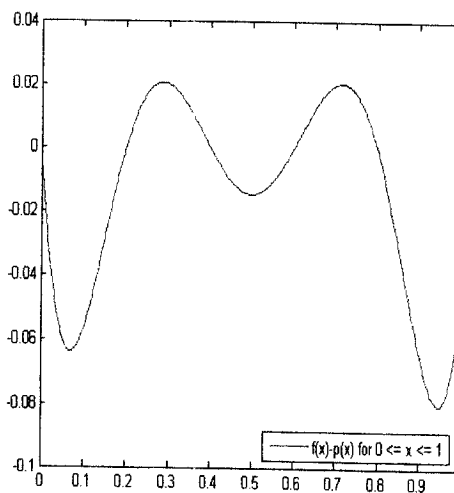
- (c) Compute the coefficients of the Newton form of the interpolating polynomial for the fifth degree polynomial that interpolates $f(x) = \cos(2\pi x)$ at the points $x_j = j/5$ for $j = 0, \dots, 5$. Display the results in a table.

j	$x_j = j/5$	$f(x) = \cos(2\pi x)$	a_n
0	0	1.0000000000000000	1.0000000000000000
1	0.2	0.309016994374947	-3.454915028125263
2	0.4	-0.809016994374947	-5.338137289060529
3	0.6	-0.809016994374947	32.189270247390361
4	0.8	0.309016994374947	-40.236587809237960
5	1.0	1.0000000000000000	0.0000000000000050

(d) Make a plot of $f(x)$ and the interpolating polynomial, $p(x)$, from part (c) on the same axes for $0 \leq x \leq 1$. Plot the difference of f and p for $0 \leq x \leq 1$.



(a) $f(x) = \cos(2\pi x)$



(b) $f(x) - p(x)$

Figure 1: (a) Comparison of $f(x)$ and $p(x)$ for $0 \leq x \leq 1$. (b) Errors between $f(x)$ and $p(x)$

- (e) Estimate the maximum of $|f(x) - p(x)|$ on the interval $[0, 1]$ by evaluating f and p for a large number of points between 0 and 1.

Answer: From figure 1(b) the maximum of $|f(x) - p(x)|$ is 0.0801 at $x \approx .95$

- (f) Evaluate $p(1.5)$, $p(2.0)$, and $p(2.5)$ and report the results. Why are the values of p so different from the values of f at these points compared to points in $[0, 1]$?

Answer: The interpolating polynomial is no longer constrained to the interpolation points. So, the values of $p(1.5)$, $p(2.0)$, and $p(2.5)$ are very different from the values of f . Additionally, as we move away from the interpolating points the function trends towards its limit.

The "limit" of a polynomial is ∞ !

x	$f(x) = \cos(2\pi x)$	$p(x)$
1.5	1.0000000000000000	-23.222488340937424
2.0	1.0000000000000000	-1.641847717655944e+002
2.5	-1.0000000000000000	-5.727740545586124e+002

2

16/20

On the last page of this assignment, there is a MATLAB function that takes as input vectors x and y , and returns an array, P , which contains the coefficients of the natural cubic spline through the points (x_i, y_i) . The cubic spline is defined as $S(x) = S_i(x)$ for $x_i \leq x \leq x_{i+1}$, and

$$S_i(x) = P_{i1} + P_{i2}(x - x_i) + P_{i3}(x - x_i)^2 + P_{i4}(x - x_i)^3$$

- (a) Write a routine that takes as input the arrays x and P and the scalar z and returns $S(z)$.

```
function [Sz] = twoA(x,P,z)
    i = 1;
    n = length(x);

    %check if z is in the range of x
    if(z > x(n) || z < x(i))
        return; not always ok to return null
    end

    % find i where z lives
    while (z > x(i+1))
        i = i + 1;
    end

    %compute S(z) at i
    Sz = P(i,1) + P(i,2)*(z-x(i)) + P(i,3)*(z-x(i))^2 + P(i,4)*(z-x(i))^3
end
```

- (b) In the absence of derivatives at the endpoints, one commonly uses the not-a-knot boundary condition rather than natural boundary conditions. If the points are indexed from $j = 0 \dots n$, then the not-a-knot condition requires that $S'''(x)$ be continuous at x_1 and x_{n-1} . For $S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$, our textbook derives the system of linear equations for the values of c_i :

$$h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} = 3/h_i(a_{i+1} - a_i) - 3/h_{i-1}(a_i - a_{i-1})$$

for $i = 1 \dots n - 1$, where $h_i = x_{i+1} - x_i$. We need two more equations to be able to solve for the c values. Derive the two equations that come from the not-a-knot conditions involving only c values and h values.

(2b) Answer

The not-a-knot condition $S'''(x)$ will be continuous at x_1 and x_{n-1} and $S_0'''(x_1) = S_1'''(x_1)$ and $S_i'''(x) = d_i$, $d_0 = d_1$. Therefore, by using the equation $d_i = c_{i+1} - c_i/3h_i$ we can solve for $c_0 = c_1(1 + h_0/h_1) - c_2(h_0/h_1)$

For the second boundary condition at x_{n-1} we want the condition that $S_{n-2}'''(x_{n-1}) = S_{n-1}'''(x_{n-1})$ to be true. So using the equation $d_i = c_{i+1} - c_i/3h_i$ we can solve for c_n by setting $d_{n-2} = d_{n-1}$ to get $c_n = c_{n-1}(1 + h_{n-1}/h_{n-2}) - c_{n-2}(h_{n-1}/h_{n-2})$.

- (c) Write a routine to compute the coefficients of a cubic spline with not-a-knot boundary conditions. This requires only minor changes from the natural spline code provided.

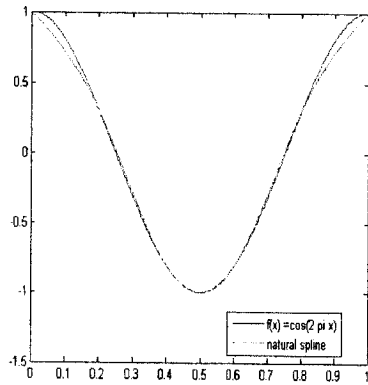
Answer: Changes to the natural spline code removed the natural spline boundary conditions and added the not-a-knot boundary conditions derived in part b. Changes to the code are below:

You changed the equations in (b) so - need to change the matrix in the code.

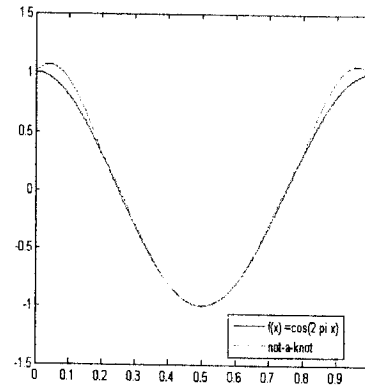
```
% append the not-a-knot boundary conditions
eq1 = z(2)*((1+(h(1)/h(2))) -z(2)*(h(1)/h(2)));
eq2 = z(n-2)*(1+(h(n-1)/h(n-2))) - z(n-3)*(h(n-1)/h(n-2));
z = [eq1; z; eq2];
```

- (d) Plot the natural and not-a-knot splines that interpolate $f(x) = \cos(2\pi x)$ at the points $x_j = j/5$ for $j = 0, \dots, 5$. Which spline looks more like the cosine function?

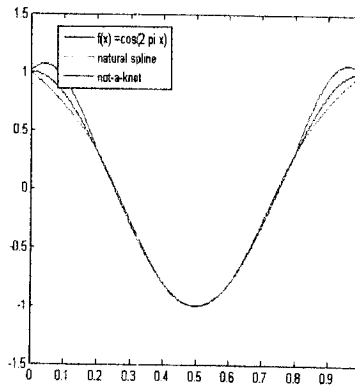
Answer The natural spline looks more like the cosine function, and the not-a-knot is smoother.



(a) $f(x)$ and natural boundary.



(b) $f(x)$ and not-a-knot.



(c) All three functions.

Figure 2: Natural spline, not-a-knot, and $f(x) = \cos(2\pi x)$ for $0 \leq x \leq 1$.

- (e) For each spline, find the maximum error in using the spline to approximate $f(x)$ between $x = 0$ and $x = 1$.

Answer

For the natural boundary condition the error, $E \approx \max|f(x) - S_n(x)| \approx 0.095168036406006$

For the not-a-knot boundary condition the error, $E \approx \max|f(x) - S_n(x)| \approx 0.1365$

3 $\frac{7}{10}$

- (a) Use $N = 11$ equally spaced points between $x = -1$ and $x = 1$, and plot the $N - 1$ degree polynomial that interpolates $g(x) = 1/1 + 25x^2$ at these points for $-1 \leq x \leq 1$. Repeat for $N = 21$.

What is the maximum error in using these interpolating polynomials to approximate $g(x)$. **Answer:** $\max_{N=11}(|f(x) - g(x)|) = 1.9156$ and $\max_{N=21}(|f(x) - g(x)|) = 58.5855$

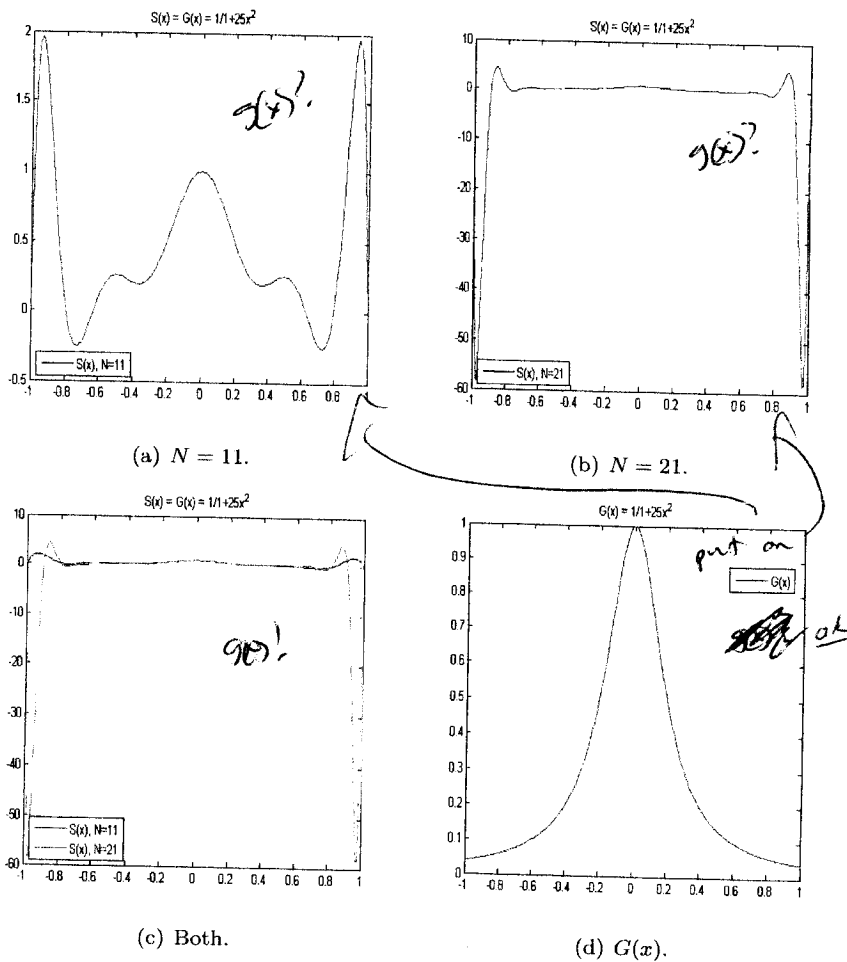
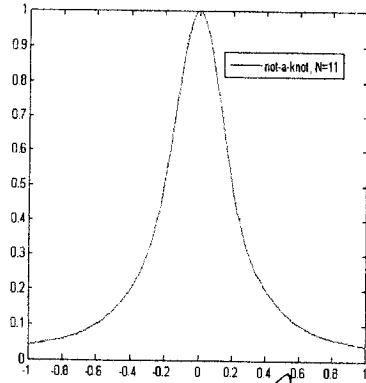
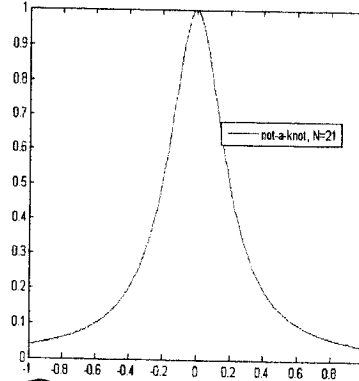


Figure 3: Newton interpolating polynomial of $g(x) = 1/1 + 25x^2$ evaluated using Horner's method using polynomials of degree $N = 10$ and $N = 20$ for $-1 \leq x \leq 1$.

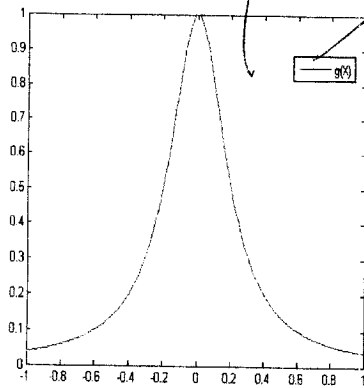
(b) Repeat part (a), but use a not-a-knot spline to interpolate the data.



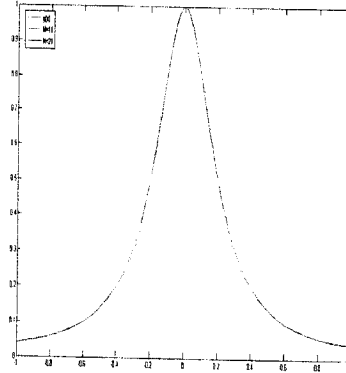
(a) $N = 11$.



(b) $N = 21$.



(c) $g(x)$.

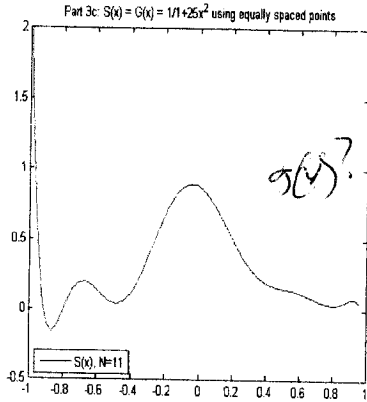


(d) All three.

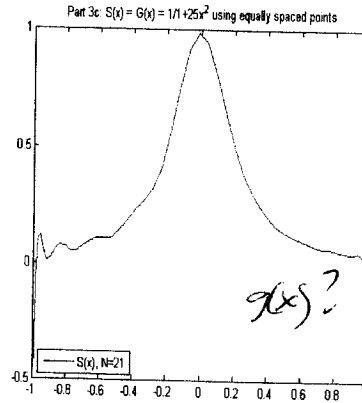
errors?

Figure 4: Not-a-knot boundary cubic spline interpolating polynomial of $g(x) = 1/1 + 25x^2$ for (a) $N=11$, (b) $N=21$, (c) $g(x)$, and (d) all three.

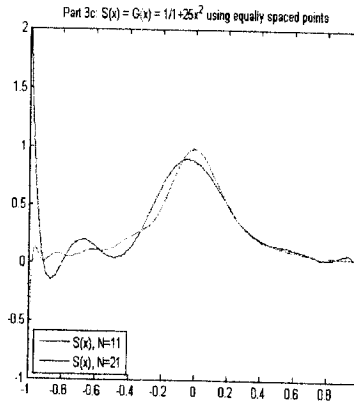
- (c) Repeat part (a) using the points $x_j = \cos(2j + 1/2N + 1)\pi, j = 0 \dots N$, instead of equally spaced points.



(a) $N = 11$.



(b) $N = 21$.



(c) Both.

Figure 5: Newton interpolating polynomial of $g(x) = 1/1+25x^2$ evaluated using Horner's method using polynomials of degree $N = 10$ and $N = 20$ for ^{non}equally spaced points $j = 0 \dots N$.

- (d) Comment on your results.

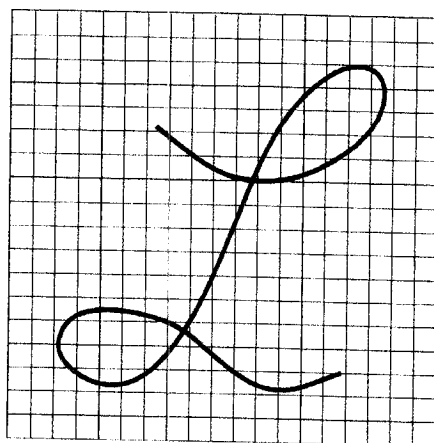
Answer: The error is large using uniformly spaced points because the Lagrange form of the interpolation error multiplies errors together. Using non-equally spaced intervals produces a smaller error due to variations in the product terms of the Lagrange form of the interpolation error.



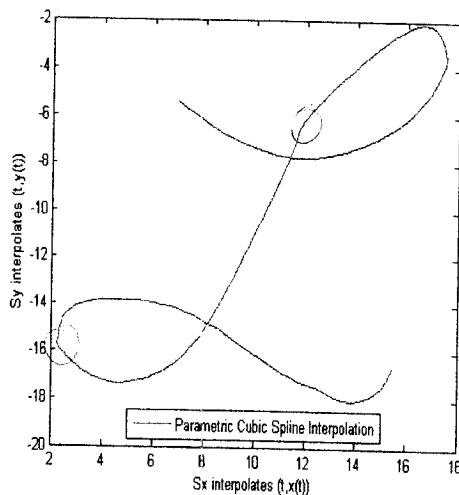
4

10/10

Use cubic splines to represent a curve $(x(s), y(s))$ that is in the shape of the script letter L below. List the points you used to make the splines, and make a plot of your curve. You may use the provided grid to estimate the nodes by hand, or use software to help identify points on the curve. For example, in MATLAB the commands `imread`, `image`, and `ginput` could be used to aid in selecting your points.



(a) Original script L font.



(b) L as cubic spline.

Figure 6: Parameterized cubic spline representation of script L using 20 points and not-a-knot boundary conditions.

Table of data selected from letter 'L'.

Selected data points for plotting L		
t	x	y
1	6.9215	5.4569
2	10.2261	7.4397
3	13.2872	7.5789
4	16.2092	5.9092
5	17.3919	3.9612
6	17.4267	3.1263
7	16.1049	2.2915
8	12.4176	5.5961
9	11.548	7.2658
10	10.087	10.7791
11	8.4868	14.2577
12	6.5388	16.7623
13	4.0343	17.284
14	2.4341	16.1013
15	2.295	15.4404
16	2.9907	14.1533
17	6.0171	13.9794
18	9.0782	15.4752
19	12.4524	17.4928
20	15.4787	16.6579

```
four          % load t,x,y
t1 = morePoints % create more t's how many more?
Ax = notaknotv3(t,x); % calc coefficients of not-a-knot for x
Ay = notaknotv3(t,y); % calc coefficients of not-a-knot for y
Px = twoD(t,Ax,t1); % evaluate cubic spline for (t,x)
Py = twoD(t,Ay,t1); % evaluate cubic spline for (t,y)
plot(Px,-Py)    % plot cubic spline
```