**Jaclyn Wilson**

**Homework 5!**

**All work and code must be shown for credit unless otherwise stated – did you include enough code and output to demonstrate how you solved each problem?** Problems marked **EXTRA** are required for 6310 and optional (extra credit, partial credit given) for 3310 students.

For both 3310 / 6310, extra credit will be given for good presentation, i.e. organization, readability, etc. Typing all math (using e.g. LaTeX, LyX, Word, LibreOffice) will help but is not strictly required if you have very clear handwriting.

Please: submit a single\* *organized* document such as .docx or .pdf containing all answers, code, graphs, etc. Label all parts and put them in order.

\*3310: please submit any EXTRA problems as a separate file to the corresponding EXTRA Canvas assignment.

1. The codes to fit a natural or clamped spline to given data points, natural\_cubic\_spline.m and clamped\_cubic\_spline.m, are available on Canvas due to the linear algebra required (covered later in the course). However, it is also crucial to be able to evaluate a spline function at some points x\_interp (typically in between the original data points). A nearly complete code cubic\_spline\_interp.m is available on Canvas that takes as input the spline coefficients , original data nodes , and locations to evaluate the spline at x\_interp. Fill in the one missing line of code that corresponds to the definition of each spline piece:

Alternatively you are free to program your own version of this function as long as it takes the specified inputs and returns the interpolated values on the spline. (The given code is not very efficient but gets the job done.)

y\_interp(i) = a(j) + b(j).\*(x\_interp(i)-x\_nodes(j))+c(j).\*(x\_interp(i)-x\_nodes(j)).^2 + d(j).\*(x\_interp(i)-x\_nodes(j)).^3;

1. Find the clamped cubic spline passing through the points (1, 1), (2, 2), (3, 3) with and .
2. Write the 8 equations that determine the 8 coefficients of but don’t solve them.
3. Use clamped\_cubic\_spline.m to solve for the 8 coefficients and show that these values satisfy all the equations from (a) by plugging them in.

>> x = [1 2 3]

>> y = [1 2 3]

>> y\_prime = [0 0]

>> clamped\_cubic\_spline(x, y, y\_prime)

b =

0.0000

1.5000

c =

1.5000

0.0000

d =

-0.5000

-0.5000

ans =

1

2

So, this function yielded the following coefficients:

So plugging them in:

All of the equations are correct and satisfied.

1. Write the expression for , making sure to define each piece and the -interval it is valid over.
2. What is the equation for the natural cubic spline passing through these same points? (Hint: it is very simple.)

This is the equation of a natural cubic spline since the only different is the derivative to the second degree is equal to 0 and this equation with these coefficients also satisfies this equation.

1. Graph both the clamped and natural spline on the same plot for . Make sure no artificial jaggedness is visible in the curves. You can use cubic\_spline\_interp.m.

>> x\_vec = linspace(1, 3, 100)

>> [a,b,c,d]=natural\_cubic\_spline([1 2 3], [1, 2, 3])

>> y\_interp\_natural = cubic\_spline\_interp(a,b,c,d,[1 2 3],x\_vec)

>> [a,b,c,d] = clamped\_cubic\_spline( [1 2 3], [1 2 3], [0 0] )

>> y\_interp\_clamped = cubic\_spline\_interp(a,b,c,d,[1 2 3],x\_vec)

>> figure(1);

>> plot(x\_vec,y\_interp\_natural,'r--','LineWidth',2);

>> hold on

>> grid on

>> plot(x\_vec,y\_interp\_clamped,'b--','LineWidth',2);

>> plot([1 2 3],[1 2 3],'ko','MarkerFaceColor','k');

>> hold off

>> legend('natural S(x)','clamped S(x)', 'nodes', 'Location','best','fontsize',16)

>>

Chart

Description automatically generated

1. Consider the natural cubic spline that interpolates using the nodes .
2. Use natural\_cubic\_spline.m to find and list the coefficients for each piece of .

>> F = @(x) log(exp(x) + 2);

>> x1 = [-1 -0.5 0.5]

>> [a, b, c, d]=natural\_cubic\_spline(x1, F(x1))

a =

0.8620

0.9580

b =

0.1680

0.2402

c =

0

0.1443

d =

0.0962

-0.0481

>>

1. Write the expression for , making sure to define each piece and the -interval it is valid over.
2. Approximate using your spline and report the relative error of this approximation.

Actual f’(0.25)

>> g = @(x) (exp(x)./(exp(x)+2));

>> g(0.25)

ans =

0.3910

Below is derivative of the S(x) function and calculated value approximation of f’(0.25)

>> f = @(x) 0.2402+0.1443.\*(2.\*x+x)-0.0481.\*(3.\*x.^2+2.\*x+0.5.^2+x+0.5)

>> f(0.25)

ans =

0.2673

>>

Relative Error:

1. Consider the seemingly innocuous function from the lectures, .
2. Plot for .

>> F = @(x) 1./(1+x.^2);

>> x = linspace(-5,5,100);

>> plot(x, F(x))

Graphical user interface, chart

Description automatically generated

1. Compute the natural cubic splines that interpolate using equally spaced nodes ( e.g. linspace(-5, 5, n) ), trying one at a time. Use natural\_cubic\_spline.m. Plot the three splines on the graph from (a) and include a legend or otherwise label and which spline curve goes with which . Make sure no artificial jaggedness is visible in the curves. You can use cubic\_spline\_interp.m.

>> x1 = linspace(-5, 5, 5)

>> [a, b, c, d]=natural\_cubic\_spline(x1, F(x1));

>> interp\_1 = cubic\_spline\_interp(a, b, c, d, x1, F(x1));

>> x2 = linspace(-5,5,7);

>> [a, b, c, d]=natural\_cubic\_spline(x2, F(x2));

>> interp\_2 = cubic\_spline\_interp(a, b, c, d, x2, F(x2));

>> x3 = linspace(-5,5,13);

>> [a, b, c, d]=natural\_cubic\_spline(x3, F(x3));

>> interp\_3 = cubic\_spline\_interp(a, b, c, d, x3, F(x3));

>> figure(1)

>> hold on

>> plot(x1,interp\_1,'r--','LineWidth',2);

>> plot(x2,interp\_2,'b--','LineWidth',2);

>> plot(x3,interp\_3,'g--','LineWidth',2);

>> grid on

>> hold off

>> legend('n=5','n=7', 'n=13', 'Location','best','fontsize',16)

Chart, line chart

Description automatically generated

1. Briefly compare the behavior with increasing with the behavior of the Lagrange interpolating polynomials applied to this same that were discussed in lecture.

Here, we can visualize as n increases, the graph shows that the minimum value decreases.

Graphical user interface, chart, application

Description automatically generated

As seen in the graph above it is the interpolating polynomial with various values of n. In this example, as done in lecture, we see that as n increases the graph becomes more specific. It is interesting though how in this example, we see that on the boundaries, -5 and 5, we have a negative decreasing. It is also interesting how the function graphed appears to be upside down compared to this problem.

1. **EXTRA** The 3D coordinates, stored as vectors X, Y, Z, of a swimming ciliate (a unicellular organism) are stored in ciliate\_path.mat on Canvas. The partially complete script HW5\_extra.m is also provided. The goal is to create a parametric spline that interpolates the discrete 3D coordinates, forming a smooth, continuous representation of the swimming trajectory. To do this, an independent parameter is created (which can but does not have to represent time) such that one can generate and evaluate three separate parameterized interpolants for X(t), Y(t), Z(t) using the codes you already have. Complete the few missing lines of code and also submit a copy of the plot that is generated. You can use natural\_cubic\_spline.m and cubic\_spline\_interp.m.