**ChE 310 Problem Set 4 (15 pts) Due Wed 2/13/19**

Collect all m-files in a single .zip file and upload the .zip file to the course webpage by midnight on Wednesday, February 13, 2019. Please note any collaborations in the Canvas upload comment box. Each student must upload their own individual copy of the work.

**4\_1 (4 pts)** Perform steps in a script file ps4\_1.m; upload this script and any additional files that are necessary to run your script.

We collect the following steady-state temperature data (°C) on a two-dimensional computer chip with a heating source near (8,8):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | $$x=0$$ | $$x=2$$ | $$x=4$$ | $$x=6$$ | $$x=8$$ |
| $$y=0$$ | 20.00 | 25.00 | 30.00 | 35.00 | 40.00 |
| $$y=2$$ | 30.00 | 27.07 | 30.39 | 38.78 | 55.00 |
| $$y=4$$ | 40.00 | 35.03 | 38.43 | 48.90 | 70.00 |
| $$y=6$$ | 50.00 | 48.15 | 53.50 | 64.69 | 85.00 |
| $$y=8$$ | 60.00 | 70.00 | 80.00 | 90.00 | 100.00 |

We have a temperature-sensitive component located at $x=4.2$, $y=3.7$ and need to ensure that the temperature at that component isn’t too high (failure threshold 45 °C). Use a *bivariate interpolating spline* as described below to determine the temperature at $x=4.2$, $y=3.7$.

1. First, plot the 3-D surface of the chip and label axes accordingly.
2. For each fixed value of y, calculate a cubic spline that describes the temperature as a function of x and evaluate that spline at x = 4.2. (You’ll generate 5 splines and generate 5 total temperatures at the points ($x=4.2$, $y\_{i}$) for the 5 $y\_{i}$ values.)
3. Now, use these data at the fixed value $x=4.2$ to calculate a cubic spline that describes the temperature as a function of $y$ for the fixed value $x=4.2$. Evaluate this spline at $y=3.7$ to obtain your final temperature.

Output the component temperature $T=f\_{4}(4.2,3.7)$ to the command line, along with the value predicted by *piecewise bilinear interpolation*. (Ensure these outputs are labeled to clearly identify which is which.) Will the component fail? Also ensure your plot displays when your script is run.

***Solution:***

%ps4\_1.m

clear, clf

%Input all temperature data and set x/y values.

T = [20.00 25.00 30.00 35.00 40.00;

 30.00 27.07 30.39 38.78 55.00;

 40.00 35.03 38.43 48.90 70.00;

 50.00 48.15 53.50 64.69 85.00;

 60.00 70.00 80.00 90.00 100.00];

x = 0:2:8; y = x;

%Create meshgrid for plotting

[X, Y] = meshgrid(x,y);

surf(X,Y,T)

xlabel('x')

ylabel('y')

zlabel('T, deg C')

hold on;

%We have 5 splines - need an interpolated value at x=4.2 for each of them.

x\_interp = zeros(5,1);

for ii = 1:5

 %Three optional lines: used to plot the 5 splines. (Take a look!)

 x\_big = linspace(0,8);

 x\_interp\_big = spline(x,T(ii,:),x\_big);

 plot3(x\_big,ones(1,length(x\_big))\*y(ii),x\_interp\_big,'r-','linewidth',3);

 %This is the only one needed

 x\_interp(ii) = spline(x,T(ii,:),4.2);

end

%Three optional lines: used to plot the spline at x = 4.2.

y\_big = x\_big;

spline\_big = spline(y,x\_interp,y\_big);

plot3(ones(1,length(y\_big))\*4.2,y\_big,spline\_big,'g-','linewidth',3);

%This one is needed. Use the interpolated temps at x = 4.2 to get the final

%spline temperatures.

spline\_T = spline(y,x\_interp,3.7);

%Use interp2 to get the bilinear interpolation for comparison.

lin\_T = interp2(X,Y,T,4.2,3.7);

fprintf('The predicted component temperature is %4.4g degrees C from spline interpolation and %4.4g degrees C from bilinear interpolation. The component will not fail.\n',spline\_T,lin\_T)

**Output:**

>> ps4\_1

The predicted component temperature is 37.5 degrees C from spline interpolation and 38.24 degrees C from bilinear interpolation. The component will not fail.



(Another view of the splines)



**4\_2 (4 pts)** Perform steps in a script file ps4\_2.m; upload this script and any additional files that are necessary to run your script.

Consider the function $f\left(x\right)=\frac{1}{\left(x-0.3\right)^{2}+0.01}+\frac{1}{\left(x-0.9\right)^{2}+0.04}-6$; this is the built-in “humps” function in MATLAB. Generate 8 data points (*x*,*y*) for this function using uniformly-spaced *x* values on the interval [0 1].

In a 2x2 subplot, plot the original function, all 8 data points, and a representation of the following (each in its own panel):

1. Piecewise linear interpolation
2. Full polynomial interpolation
3. Not-a-knot cubic spline interpolation
4. Clamped cubic spline interpolation, using the true derivative of $f$ at x = 0 and 1. (Note: the analytical derivative of this function is: $f^{'\left(x\right)}=-\frac{2\left(x-0.3\right)}{\left(\left(x-0.3\right)^{2}+0.01\right)^{2}}-\frac{2\left(x-0.9\right)}{\left(\left(x-0.9\right)^{2}+0.04\right)^{2}}$

Finally, use the four interpolating methods to interpolate the values of $f(x)$ at $x=$ 0.1, 0.5, and 0.9. Calculate the absolute value of the percent error for each interpolating method relative to the true value ($error =\left|\frac{appx-actual}{actual}\right|\*100\%$) and print the resulting percent errors to the command line. The percent errors for each of the four interpolations, along with the 2x2 subplot, should be the only output of running your script.

***Solution:***

%ps4\_2.m

clear; clf;

f = @(x) 1./((x-0.3).^2+0.01)+1./((x-0.9).^2+0.04)-6; %real function

x = linspace(0,1,8); %sampled x data

y = f(x); %sampled y data

subplot(2,2,1)

plot(x,y,'ko');

hold on

fplot(f,[x(1) x(end)],'k')

xx = linspace(x(1),x(end));

yy = interp1(x,y,xx);

plot(xx,yy,'r-')

xlabel('x');ylabel('f(x)'),title('Piecewise Linear')

subplot(2,2,2)

plot(x,y,'ko');

hold on

fplot(f,[x(1) x(end)],'k')

pfull = polyfit(x,y,length(x)-1);

fplot(@(xx) polyval(pfull,xx),[x(1),x(end)],'r')

xlabel('x');ylabel('f(x)'),title('Full Polynomial Interpolation')

subplot(2,2,3)

plot(x,y,'ko');

hold on

fplot(f,[x(1) x(end)],'k')

pp\_notaknot = spline(x,y);

fplot(@(xx) ppval(pp\_notaknot,xx),[x(1),x(end)],'r')

xlabel('x');ylabel('f(x)'),title('Not-a-Knot Cubic Spline')

fprime = @(x) -2\*(x-.3)./((x-.3).^2+0.01).^2 - 2\*(x-.9)./((x-.9).^2+.04).^2;

subplot(2,2,4)

plot(x,y,'ko');

hold on

fplot(f,[x(1) x(end)],'k')

pp\_clamped = spline(x,[fprime(0) y fprime(1)]);

fplot(@(xx) ppval(pp\_clamped,xx),[x(1),x(end)],'r')

xlabel('x');ylabel('f(x)'),title('Clamped Cubic Spline')

x\_eval = [0.1 0.5 0.9];

f\_actual = f(x\_eval);

f\_lin = [interp1(x,y,0.1) interp1(x,y,0.5) interp1(x,y,0.9)];

f\_poly = [polyval(pfull,0.1) polyval(pfull,0.5) polyval(pfull,0.9)];

f\_knot = [ppval(pp\_notaknot,0.1) ppval(pp\_notaknot,0.5) ppval(pp\_notaknot,0.9)];

f\_clamped = [ppval(pp\_clamped,0.1) ppval(pp\_notaknot,0.5) ppval(pp\_clamped,0.9)];

err\_lin = abs((f\_lin-f\_actual)./f\_actual);

err\_poly = abs((f\_poly-f\_actual)./f\_actual);

err\_knot = abs((f\_knot-f\_actual)./f\_actual);

err\_clamped = abs((f\_clamped-f\_actual)./f\_actual);

fprintf('The percent error for each method is as follows:\n')

fprintf('X Value 0.1 0.5 0.9\n')

fprintf('Piecewise linear \t %6.3g%% %6.3g%% %6.3g%%\n',err\_lin\*100)

fprintf('Full polynomial \t %6.3g%% %6.3g%% %6.3g%%\n',err\_poly\*100)

fprintf('Not-a-knot spline\t %6.3g%% %6.3g%% %6.3g%%\n',err\_knot\*100)

fprintf('Clamped spline \t %6.3g%% %6.3g%% %6.3g%%\n',err\_clamped\*100)

***Output:***

>> ps4\_2

The percent error for each method is as follows:

X Value 0.1 0.5 0.9

Piecewise linear 20.7% 26.9% 10.1%

Full polynomial 292% 10.3% 49.8%

Not-a-knot spline 97.4% 10.9% 2.03%

Clamped spline 24.7% 10.9% 1.96%



**4\_3 (4 pts)** Perform steps in a script ps4\_3.m.

Solve Problem 18.2 from the text using centered finite differences to compute the first and second derivatives. Use a subplot to plot the spline fit (T vs. depth), first derivative vs. depth, and second derivative vs. depth on the same figure. We have not covered root finding yet, so find the inflection point of interest by eye. You can zoom in on the plot to get a pretty good estimate. Record the coordinates of the inflection point on your first-as and second-derivative plots by printing an arrow showing the value of interest. Print the heat flux value to the screen.

***Solution:***

%ps4\_3.m

% Coded by Nigel F. Reuel on 9.22.2017

% This code solves problem 18.2 from the text

%

D = [0 0.5 1 1.5 2 2.5 3];

T = [70 70 55 22 13 10 10];

% Create a vector that includes the zero slope condition on the front and

% back of the temperature vector;

Y = [0 T 0];

% Detailed Input vector:

n = 10000;

xx = linspace(0,3,n);

% Fit with clamped spline

yy = spline(D,T,xx);

% Look at the curve:

% plot(xx,yy)

% Use centered finite differences to determine the first derivative:

clc

h = 3/n;

D1 = zeros(n-2,1); %<-- We lose two values because can't do centered finite difference on end values

for i = 2:n-1

 D1(i-1,1) = (yy(1,i+1)-yy(1,i-1))/(2\*h);

end

% Use centered finite differences on the D1 vector to determine the second

% derivatives

h = 3/n;

D2 = zeros(n-4,1); %<-- We lose two more values because can't do centered finite difference on end values

for i = 2:n-3

 D2(i-1,1) = (D1(i+1,1)-D1(i-1,1))/(2\*h);

 if abs(D2(i-1,1)) < 0.1

 display('Depth where second derivative is zero = ')

 (i+2)\*h

 % Note: I let these print to screen and I picked the zero value in

 % the center of the reactor (as the other inflections are caused by

 % imperfect spline fit

 % Thermocline depth =~ 1.22m

 % You can also find the depth by eye as mentioned in the problem

 % statement

 end

end

% Plot the profiles and derivatives next to each other

subplot(1,3,1)

plot(xx,yy)

xlabel('Depth (m)')

ylabel('Temperature (\circC)')

subplot(1,3,2)

plot(xx(2:n-1),D1)

xlabel('Depth (m)')

ylabel('dT/dz')

txt1 = ('\leftarrow -73\circC/m');

text(1.22,-73,txt1)

subplot(1,3,3)

plot(xx(3:n-2),D2)

xlabel('Depth (m)')

ylabel('d^2T/dz^2')

txt = ('\leftarrow 1.22m');

text(1.22,0,txt)

% Now solve for the flux (watch for units)

display('Heat flux (J) in cal/(s\*m^2):')

J = 73\*0.01\*100

***Output:***

Depth where second derivative is zero =

ans =

 2.2566

Heat flux (J) in cal/(s\*m^2):

J =

 73



**4\_4 (3 pts)** Perform steps in a script ps4\_4.m.

The velocity ($v$) of air flowing past a flat surface is measured at several distances ($x$) away from the surface. Newton’s viscosity law describes the shear stress ($τ$) at the surface ($x=0$):

$$τ=μ\frac{dv}{dx}$$

Assume a dynamic viscosity of $μ=1.8×10^{-5}\frac{N s}{m^{2}}.$

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *x* (m) | 0 | 0.002 | 0.006 | 0.012 | 0.018 | 0.024 |
| *v* (m/s) | 0 | 0.287 | 0.999 | 2.215 | 3.448 | 4.699 |

Use three methods to evaluate the derivative at $x=0$ and calculate $τ$. Output your value of $τ$ for each method, and include a recommendation as to which you think will work best.

***Solution:***

%ps4\_3

%Here are a few ways to calculate the data… there might be more!

clear ; clf

x = [0 0.002 0.006 0.012 0.018 0.024]; %m

v = [0 0.287 0.999 2.215 3.448 4.699]; %m/s

mu = 1.8e-5 ; %N\*s/m^2

plot(x,v,'ko','markersize',12,'markerface','k') %Optional

%Option #1 – First order forward finite differences

h1 = x(2)-x(1);

dydt\_first = (v(2)-v(1))./(h1);

tau\_first = mu\*dydt\_first

%Option #2 – Second order forward differences (skipping point #2, which is unevenly spaced)

h2 = x(3) - x(1) ; % = x(4) - x(3)

dydt\_second = (-v(4)+4\*v(3)-3\*v(1))/(2\*h2);

tau\_second = mu\*dydt\_second

%Option 3 – Interpolate a polynomial to some data (in this case, first 3 points) and take the derivative. Here we use polyder; you could do this derivative manually as well.

parabola = polyfit(x(1:3),v(1:3),2);

d\_parabola = polyder(parabola); %differentiate polynomial array

%d\_parabola = [2\*parabola(1) parabola(2)];

dydt\_parabola = polyval(d\_parabola,0); %evaluate derivative

tau\_parabola = mu\*dydt\_parabola

***Output:***

>> ps4\_4

tau\_first =

 0.0026

tau\_second =

 0.0027

tau\_parabola =

 0.0024