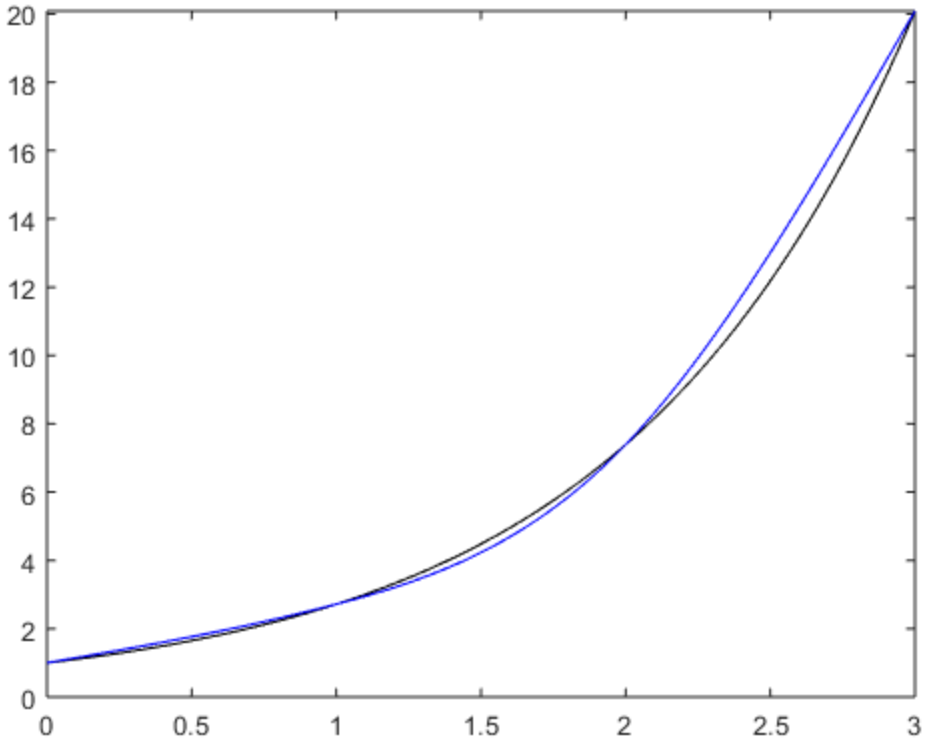
Table of Contents

Problem 1

Approximate f(x) = ex by forming a natural cubic spline S(x) using the data (0, 1), (1, e), (2, e^2); (3, e^3). Plot S(x) using a solid blue \ curve and f(x) using a solid black curve in the same figure.

```
% Put data for problem in nx2 matrix
dataMatrix = [ 0, exp(0);
               1, exp(1);
               2, exp(2);
               3, exp(3)];

% Define function for this problem
f = @(x) exp(x);

%Define figure number
figureNumber = 1;

% Graph underlying function and cubic spline on same curve
GraphFunctionAndSpline(dataMatrix, f,figureNumber);
```

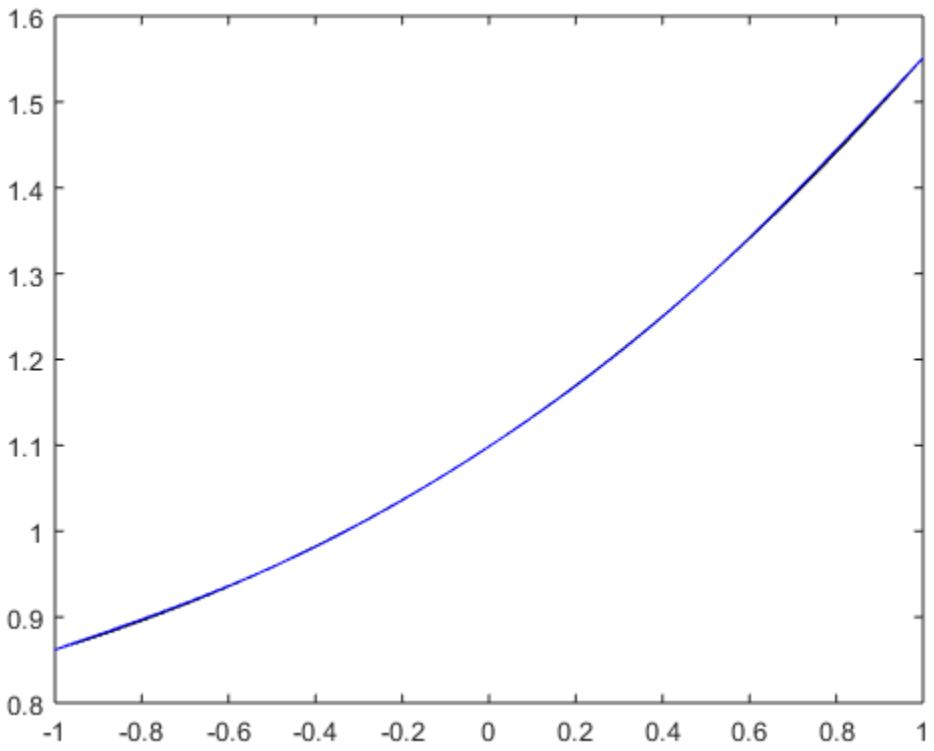


Problem 2

Repeat Problem 1 for f(x) = ln(e^x + 2) with the data

```
%      x | f(x)
%   -1.0 | 0.8619948
%   -0.5 | 0.9580201
%    0.0 | 1.0986123
%    0.5 | 1.2943767
%    1.0 | 1.5514447

% Define data matrix
dataMatrix = [-1.0, 0.8619948;
              -0.5, 0.9580201;
               0.0, 1.0986123;
               0.5, 1.2943767;
               1.0, 1.5514447];


% Define function for this problem
f = @(x)log(exp(x) +2);

%Define figure number
figureNumber = 2;

% Graph underlying function and cubic spline on same curve
```

```
GraphFunctionAndSpline(dataMatrix, f,figureNumber);
```



Problem 3

The 2014 Kentucky Derby was won by a horse named California Chrome in a time of 2:03.66 (2 minutes and 3.66 seconds) for the 1 1/4 mile race. Times at the quarter-mile, half-mile, and mile poles were 0:23.04, 0:47.37, and 1:37.45. (a) Use these values together with the starting time to construct a natural cubic spline for California Chrome's race. (b) Use the spline to predict the time at the three-quarter-mile pole and compare this to the actual time of 1:11.80. (c) Use the spline to predict California Chrome's starting speed and speed at the finish.

```
fprintf('Problem 3\n')

% Define data matrix
dataMatrix = [0.00,   0.00;
              0.25,  23.04;
              0.50,  47.37;
              1.00,  97.45;
              1.25, 123.66];

% Get coefficients for natural cubic spline
[a, b, c, d] = GetNaturalCubicSplineCoefficents(dataMatrix);

xVals = dataMatrix(:,1);
numOfDataPoints = length(xVals);
```

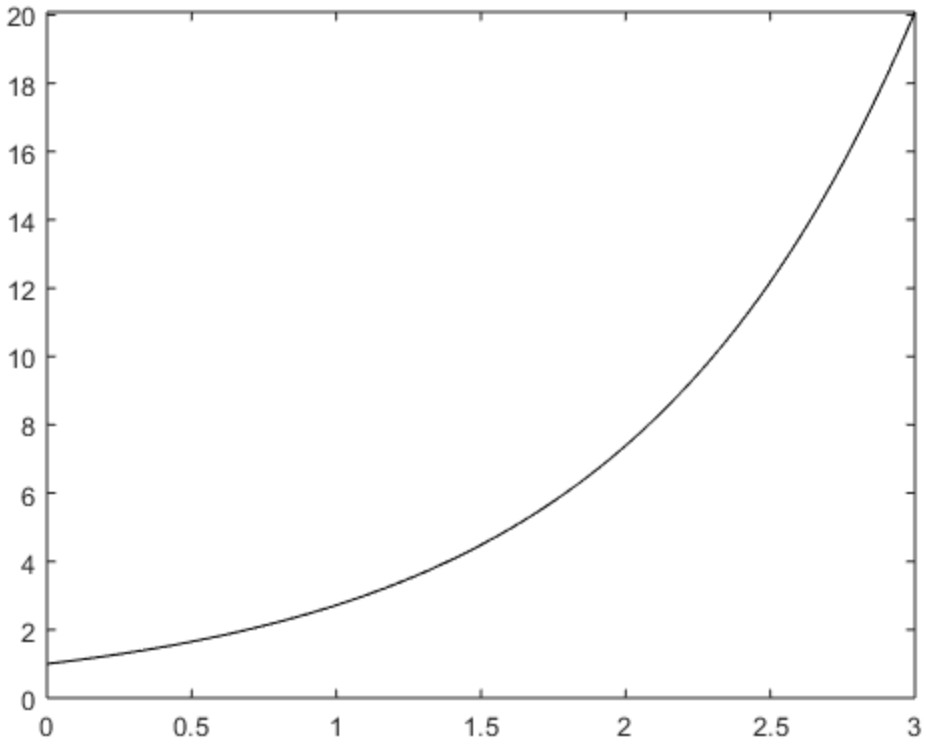
```
numOfSplines = numOfDataPoints - 1 ;

S = cell(numOfSplines,1);
% (a) Cubic spline for California Chrome's race
fprintf('(a) Cubic Spline for California Chrome''s race:\n');
for i = 1: numOfSplines
    xi = xVals(i);
    xiPlus1 = xVals(i+1);
    syms x
    S{i} = (a(i) + (b(i))*(x - xi) + (c(i))*(x - xi).^2 + (d(i))*(x -
 xi).^3);
    fprintf('S%d = %.6f + (%.6f)(x - %.2f) + (%.6f)(x - %.2f)^2 +
 (%.6f)(x - %.2f)^3\n',...
        i-1, a(i), b(i), xi, c(i), xi, d(i), xi);
    fprintf('for x in [%0.2f, %0.2f]\n', xi, xiPlus1)
end

%(b) Use spline to predicit time at three-quarter mile marker
S2 = matlabFunction(S{2}); % S2 is for x in [0.50, 1.00]

EstimatedTimeAtThreeQuarterMileMarker = S2(0.75);
ActualTimeAtThreeQuarterMileMarker = 71.80; % 1:11.80

relativeError = (abs(ActualTimeAtThreeQuarterMileMarker - ...
             EstimatedTimeAtThreeQuarterMileMarker)/...
             ActualTimeAtThreeQuarterMileMarker) *100;
fprintf('\n(b) The estimated time at the three-quarter-mile marker is
 %.2f\n',...
    EstimatedTimeAtThreeQuarterMileMarker);
fprintf('The actual time at the three-quarter-mile marker was %.2f
\n',...
    ActualTimeAtThreeQuarterMileMarker);
fprintf('The relative error is %0.2f%%\n', relativeError);

% The cubic splines are functions where time is a function of
 distance. The
% units for the derivative will be seconds per mile. If we take the
% recipricol of this, we will get miles per second, which we can then
% convert to miles per hour using 1 hour = 3600 seconds.

S0 = S{1};
dS0dt = matlabFunction(diff(S0));
initialSpeed = (1/(dS0dt(0)))*(3600);

S3 = S{4};
dS3dt = matlabFunction(diff(S3));
finalSpeed = (1/(dS3dt(1.25)))*(3600);

fprintf('(c) California Chrome''s initial speed was %0.2f mph and his
 final speed was %0.2f mph\n',...
    initialSpeed, finalSpeed);

% The estimated speeds are reasonable for a race horse.
```

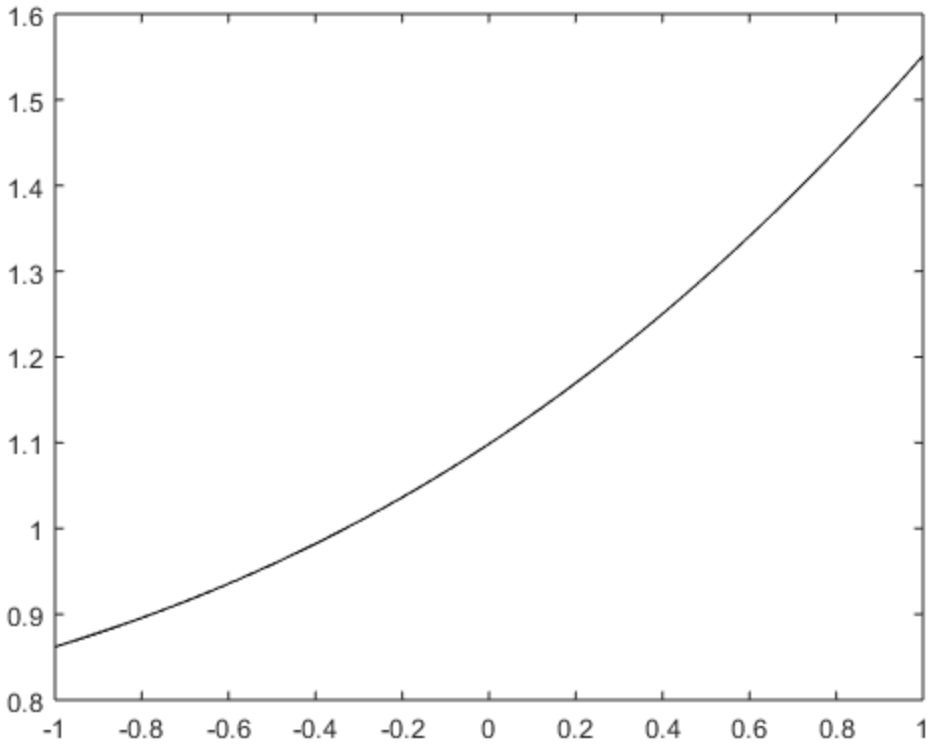
```
Problem 3
(a) Cubic Spline for California Chrome's race:
S0 = 0.000000 + (90.869508)(x - 0.00) + (0.000000)(x - 0.00)^2 +
 (20.647869)(x - 0.00)^3
for x in [0.00, 0.25]
S1 = 23.040000 + (94.740984)(x - 0.25) + (15.485902)(x - 0.25)^2 +
 (-20.679344)(x - 0.25)^3
for x in [0.25, 0.50]
S2 = 47.370000 + (98.606557)(x - 0.50) + (-0.023607)(x - 0.50)^2 +
 (6.260984)(x - 0.50)^3
for x in [0.50, 1.00]
S3 = 97.450000 + (103.278689)(x - 1.00) + (9.367869)(x - 1.00)^2 +
 (-12.490492)(x - 1.00)^3
for x in [1.00, 1.25]

(b) The estimated time at the three-quarter-mile marker is 71.70
The actual time at the three-quarter-mile marker was 71.80
The relative error is 0.14%
(c) California Chrome's initial speed was 39.62 mph and his final
 speed was 34.08 mph
```

Graph Function and Spline

```
function [] = GraphFunctionAndSpline(dataMatrix, f,figureNumber)
% This function finds the cubic spline for a given data set and graphs
 it
% on a plot with the underlying function
%    INPUTS - dataMatrix: nx2 rectangluar matrix with the inputs in
 column 1
%             and the outputs in column 2.
%             f: the function which generatated the data points in
%             dataMatrix.
%             figureNumber = the number figure number for the plot.
%   OUTPUTS - Plots data. Does not return anything to calling block.

% Define set of input data based on the first and last entries in the
 first
% column of dataMatrix
fOfxXVals = [dataMatrix(1,1):0.01:dataMatrix(end,1)];
% Evaluated f at the above data points.
fOfxYVals = f(fOfxXVals);

% Define figure number and plot function values
figure(figureNumber)
plot(fOfxXVals, fOfxYVals, 'k-') % black solid line
hold on

% Get the coefficients for the cubic spline
[a, b, c, d] = GetNaturalCubicSplineCoefficents(dataMatrix);
numOfSplines = size(a);

% For every subdomain of the cubic spline, get set of inputs and
 evaluate
```

```
% on the appropriate piece of the spline, then plot piece.
for i = 1: numOfSplines
    xi = dataMatrix(i,1);
    xiPlus1 = dataMatrix((i + 1),1 );
    splineXVals = [xi:0.01:xiPlus1];
    cubicSpline = @(x) a(i) + (b(i))*(x - xi) + (c(i))*(x - xi).^2 +
 (d(i))*(x - xi).^3;
    splineYVals = cubicSpline(splineXVals);
    plot(splineXVals, splineYVals, 'b-')
    hold on
end
hold off
end
```





Get coefficient Matrix for Natural Cubic Spline

```
function  [a, b, c, d] = GetNaturalCubicSplineCoefficents(dataMatrix)
% This function takes a set of data points and returns the
 coefficients for
% the cubic splines for that given data set.
%    INPUTS - dataMatrix: nx2 matrix containing inputs in column 1 and
%             outputs in column 2.
%   OUTPUTS - a: vector containing constants terms. Same as column 2
 of
%             dataMatrix
%             b: vector containing coefficients for (x-xi) term
%             c: vector containing coefficients for (x-xi)^2 term
%             d: vector containing coefficients for (x-xi)^3 term

% Extract xVals and yVals from dataMatrix
xVals = dataMatrix(:, 1);
yVals = dataMatrix(:, 2);

% find number of data points
n = length(xVals);

% set yVals vector to aCoeff
aCoeff = yVals;
```

```
% Define vector containing the step size between each data point.
h = xVals(2:end,1) - xVals(1:end-1,1);

% Initialize A matrix
A = zeros(n,n);

% Set main diagonal ends to 1
A(1,1) = 1;
A(n,n) = 1;

% Get values for main diagonal
mainDiagOfA = 2*(h(1:end-1,1) + h(2:end,1));

% Set elements on main diagonal
A(2:end-1, 2:end-1) = diag(mainDiagOfA);
% Set elements on lower diagonal
A(2:end-1, 1:end-2) = A(2:end-1, 1:end-2) + diag(h(1:end-1,1));
% Set elements on upper diagonal
A(2:end-1, 3:end) = A(2:end-1, 3:end) + diag(h(2:end));

% Initialize bSol for equation A(cCoeff) = bSol. Not to be confused
 with
% the vector containing the coefficients for (x-xi)
bSol = zeros(n,1);

% Fill bSol vector based on algorithm given in class

bSol(2:end-1) = (3./h(2:end,1)).*(aCoeff(3:end,1) - aCoeff(2:end-1,1))
 - ...
             (3./h(1:end-1,1)).*(aCoeff(2:end-1,1) -
 aCoeff(1:end-2,1));

% Find coefficients for (x-xi)^2 terms
cCoeff = inv(A)*bSol;

% Find coefficients for (x-xi)^3 terms
dCoeff = zeros(n,1);
dCoeff(1:end-1,1) = (cCoeff(2:end,1) - cCoeff(1:end-1,1))./
(3*h(1:end,1));

% Find coefficients for (x-xi) terms
bCoeff = zeros(n,1);
bCoeff(1:end-1) = (aCoeff(2:end,1) - aCoeff(1:end-1,1))./(h)...
                 -(h).*(2*cCoeff(1:end-1,1) + cCoeff(2:end,1))./3;

% Finalize coefficient vectors.
a = aCoeff(1:end-1,1);
b = bCoeff(1:end-1,1);
c = cCoeff(1:end-1,1);
d = dCoeff(1:end-1,1);

end
```

Published with MATLAB® R2018b