Table of Contents

E7 Lab 10 Solutions

Spring 2016

```
format compact
format short
clear all
clc
close all
```

Question 1.1

```
type mySin
```

```
function [sinSeries] = mySin (X,N)
% Computes the Nth Taylor expansion of sine for a given 2D matrix of X
% Inputs
% X: 2D array (doubles), radians
% N: Nth degree of sine Taylor Series expansion
% Output
% sinSeries: Double same size as X

sinSeries = X*0;

for i = 1:N
   sinSeries = sinSeries + ((-1)^(i-1) * (X.^(2*i-1))./factorial(2*i-1));
```

```
end
end
```

Published Test Casess

```
a = mySin(2.4, 3)

b = mySin( [0, 1; 2, 3], 8)


a =
    0.7596
b =
         0    0.8415
    0.9093    0.1411
```

Additional Test Cases

```
a1 = mySin([-1, 4; 7, -9],2)


a1 =
   -0.8333   -6.6667
  -50.1667  112.5000
```

Question 1.2

```
type CompareTaylorConvergence


function [x, err, next_term] = CompareTaylorConvergence(N)
% Finds the error in the Nth Taylor approximation for sine, along with
% absolute value of the (N+1)th term of the Taylor expansion
% Input
% N: number of iterations of Taylor approximation
% Outputs
% x: x values that you evaluated the errors at
% error: actual error between your Taylor approximation and the Matlab
% sin() function
% next_term: absolute value of the next term in the Taylor series

x = -N*pi/2:pi/8:N*pi/2;
err = abs(mySin(x,N) - sin(x));
next_term = abs(((-1)^(N) * (x.^(2*N+1))./factorial(2*N+1)));

end
```

Published Test Case

```
[x, err, next_term] = CompareTaylorConvergence(2)


x =
```

```
  Columns 1 through 6
   -3.1416   -2.7489   -2.3562   -1.9635   -1.5708   -1.1781
  Columns 7 through 12
   -0.7854   -0.3927         0    0.3927    0.7854    1.1781
  Columns 13 through 17
    1.5708    1.9635    2.3562    2.7489    3.1416
err =
  Columns 1 through 6
    2.0261    1.0958    0.5310    0.2220    0.0752    0.0183
  Columns 7 through 12
    0.0025    0.0001         0    0.0001    0.0025    0.0183
  Columns 13 through 17
    0.0752    0.2220    0.5310    1.0958    2.0261
next_term =
  Columns 1 through 6
    2.5502    1.3080    0.6052    0.2432    0.0797    0.0189
  Columns 7 through 12
    0.0025    0.0001         0    0.0001    0.0025    0.0189
  Columns 13 through 17
    0.0797    0.2432    0.6052    1.3080    2.5502
```

Additional Test Case

```
[x2, err2, next_term2] = CompareTaylorConvergence(1)
```

```
x2 =
  Columns 1 through 6
   -1.5708   -1.1781   -0.7854   -0.3927         0    0.3927
  Columns 7 through 9
    0.7854    1.1781    1.5708
err2 =
  Columns 1 through 6
    0.5708    0.2542    0.0783    0.0100         0    0.0100
  Columns 7 through 9
    0.0783    0.2542    0.5708
next_term2 =
  Columns 1 through 6
    0.6460    0.2725    0.0807    0.0101         0    0.0101
  Columns 7 through 9
    0.0807    0.2725    0.6460
```

Question 1.3

```
type mySinPeriodic
```

```
function [approxs] = mySinPeriodic(X,N)
%converts every radian value in the 2D matrix X to the range [-pi,pi] using
%the periodicity of sine and then calculates sine at each point using the
%Nth Taylor expansion for sine

%approxs = mySin((-1).^((X - rem(X,pi))/pi).*rem(X,pi),N); %mySin(sign(X).*mod(abs

[xr,xc] = size(X);
for i = 1:(xr*xc)
   while X(i) < -pi
       X(i) = X(i)+ 2*pi;
```

```
    end
    while X(i) > pi
        X(i) = X(i) - 2*pi;
    end
end

approxs = mySin(X,N);

end
```

Published Test Case

```
c = mySinPeriodic([6; -2], 3)
```

```
c =
   -0.2794
   -0.9333
```

Additional Test Cases

```
c1 = mySinPeriodic([-1, 4; 7, -9],2)
```

```
c1 =
   -0.8333   -0.2995
    0.6554    0.6254
```

Question 1.4

```
type TaylorAccRange
```

```
function [range] = TaylorAccRange(N, tol)
%finds the range of angles for which the error in the Nth taylor expansion
%is less than the given tolerance, tol
%Output
%range: 1x2 array containing minimum and maximum angle in radians for which
%every angle in between can be found within the given tolerance

f = @(x) tol - abs(((-1)^(N)*(x^(2*N+1))/factorial(2*N+1)));
angle = fzero(f,0);
range = [-abs(angle),abs(angle)];

if abs(angle) >= pi
    range = [-inf,inf];
end
end
```

Published Test Cases

```
range1 = TaylorAccRange(1, 0.001)
```

```
range2 = TaylorAccRange(5, 0.1)
```

```
range1 =
   -0.1817    0.1817
range2 =
  -Inf   Inf
```

Additional Test Case

```
range3 = TaylorAccRange(2,0.001)
```

```
range3 =
   -0.6544    0.6544
```

Question 1.5

```
type sinLookup
```

```
function out = sinLookup(X,table)
tab_ang = table(:,1)*pi/180;
tab_val = table(:,2);
X = mod(X+pi,2*pi)-pi;
out = NaN(size(X));
for i = 1:size(X,1)
    for j = 1:size(X,2)
    [low_ang, loc] = max(tab_ang(tab_ang<=X(i,j)));
    low_val = tab_val(loc);
    high_ang = tab_ang(loc+1);
    high_val = tab_val(loc+1);

    out(i,j) = low_val + (high_val-low_val)/(high_ang-low_ang)*(X(i,j)-low_ang);
    end
end
end
```

Published Test Cases

```
table = csvread('SineLookup.csv');
d = sinLookup([-5.4, 2.3, 8.4], table)
```

```
d =
    0.7727    0.7457    0.8546
```

Additional Test Case

```
d1 = sinLookup([-1, 4; 7, -9], table)
```

```
d1 =
   -0.8414   -0.7568
    0.6570   -0.4121
```

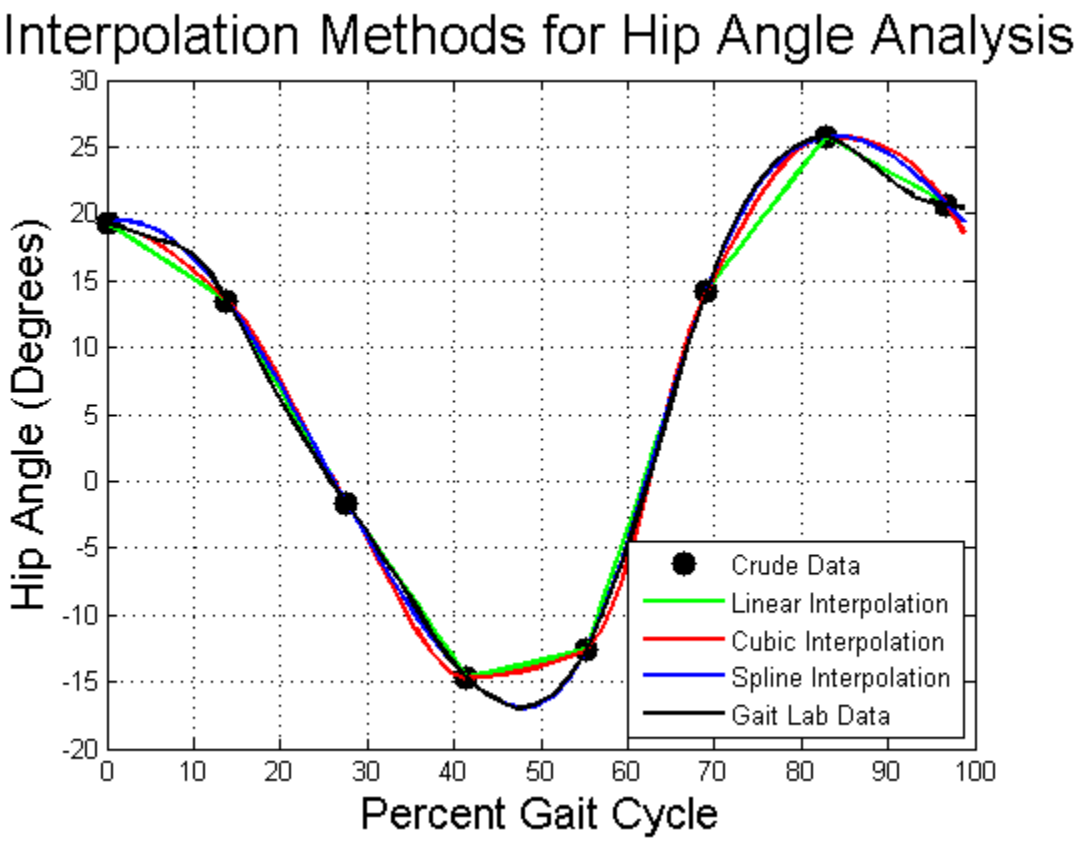
Question 2

```
type gait_data_interp
```

```
function [interpolated_angles] = gait_data_interp(crude_gait_cycle, crude_angles,

if ~any(strcmpi(interp_method, {'linear', 'cubic', 'spline'})) %If interp_method i
    disp('Please input either ''linear'', ''cubic'', or ''spline''.');
    interpolated_angles = [];
    return;
end

interpolated_angles = interp1(crude_gait_cycle, crude_angles, ideal_gait_cycle, in

end
```

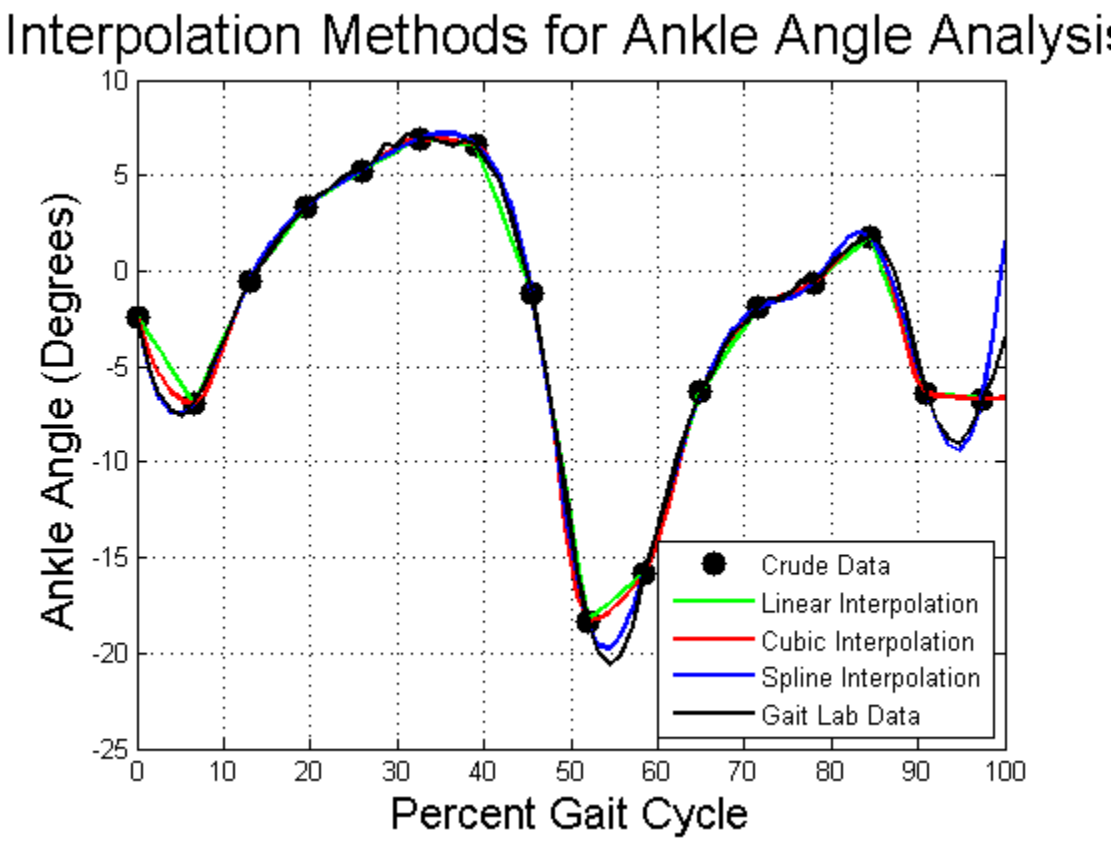
Published Test Case

```
load('GaitLabData.mat')
angle_linear = gait_data_interp(crude_gait_cycle, crude_hip_angles,...
    ideal_gait_cycle, 'linear');
angle_cubic = gait_data_interp(crude_gait_cycle, crude_hip_angles,...
    ideal_gait_cycle, 'cubic');
angle_spline = gait_data_interp(crude_gait_cycle, crude_hip_angles,...
    ideal_gait_cycle, 'spline');

figure
%Plot crude data points
plot(crude_gait_cycle, crude_hip_angles, 'k.', 'MarkerSize', 30);
hold on;
grid on;
%Plot linear interpolation
plot(ideal_gait_cycle, angle_linear, 'g', 'LineWidth', 2);
%Plot cubic interpolation
plot(ideal_gait_cycle, angle_cubic, 'r', 'LineWidth', 2);
%Plot spline interpolation
plot(ideal_gait_cycle, angle_spline, 'b', 'LineWidth', 2);
%Plot the actual gait lab data
plot(ideal_gait_cycle, gait_lab_hip_angles, 'k', 'LineWidth', 2);

xlabel('Percent Gait Cycle', 'FontSize', 16);
ylabel('Hip Angle (Degrees)', 'FontSize', 16);
title('Interpolation Methods for Hip Angle Analysis', 'FontSize', 20);
legend('Crude Data', 'Linear Interpolation', 'Cubic Interpolation',...
    'Spline Interpolation', 'Gait Lab Data', 'Location', 'SouthEast');
```



Additional Test Case

```
load('GaitLabData2.mat')
angle_linear2 = gait_data_interp(crude_gait_cycle2, crude_ankle,...
    ideal_gait_cycle2, 'linear');
angle_cubic2 = gait_data_interp(crude_gait_cycle2, crude_ankle,...
    ideal_gait_cycle2, 'cubic');
angle_spline2 = gait_data_interp(crude_gait_cycle2, crude_ankle,...
    ideal_gait_cycle2, 'spline');

figure
%Plot crude data points
plot(crude_gait_cycle2, crude_ankle, 'k.', 'MarkerSize', 30);
hold on;
grid on;
%Plot linear interpolation
plot(ideal_gait_cycle2, angle_linear2, 'g', 'LineWidth', 2);
%Plot cubic interpolation
plot(ideal_gait_cycle2, angle_cubic2, 'r', 'LineWidth', 2);
%Plot spline interpolation
plot(ideal_gait_cycle2, angle_spline2, 'b', 'LineWidth', 2);
%Plot the actual gait lab data
plot(ideal_gait_cycle2, ideal_ankle, 'k', 'LineWidth', 2);

xlabel('Percent Gait Cycle', 'FontSize', 16);
ylabel('Ankle Angle (Degrees)', 'FontSize', 16);
title('Interpolation Methods for Ankle Angle Analysis', 'FontSize', 20);
legend('Crude Data', 'Linear Interpolation', 'Cubic Interpolation',...
    'Spline Interpolation', 'Gait Lab Data', 'Location', 'SouthEast');
```



Question 3.1

```
type mySoilSpline
```

```
function [site_data] = mySoilSpline(bd, site_locations)
%bd=csvread('Borings.csv',1,2);
% convert from depths to elevations
bd(3:end,:)=repmat(bd(2,:),size(bd,1)-2,1)-bd(3:end,:);
% do the spline, and tack on the x locations to the top
sd = [site_locations; NaN(size(bd,1)-1,length(site_locations))];
for n = 2:size(sd,1)
    sd(n,:) = spline(bd(1,:),bd(n,:),site_locations);
end
% set any points past the intersection of the two layers to NaN
sd(5,sd(5,:)<sd(6,:)) = NaN;
% convert back to depths and return
sd(3:end,:)=repmat(sd(2,:),size(sd,1)-2,1)-sd(3:end,:);
site_data = sd;
end
```

Published Test Case

```
borehole_data = csvread('Borings.csv', 1, 2);
interpolated_layers = mySoilSpline(borehole_data, [1350, 1450, 1500, 1700])
```

```
Warning: All data points with NaN in their value will be
ignored.
interpolated_layers =
  1.0e+003 *
    1.3500    1.4500    1.5000    1.7000
    0.3546    0.3556    0.3545    0.3587
    0.0126    0.0120    0.0100    0.0143
    0.0048    0.0044    0.0033    0.0058
       NaN       NaN    0.0186    0.0170
    0.0237    0.0220    0.0210    0.0393
    0.0492    0.0449    0.0422    0.0625
```

Additional Test Cases

```
borehole_data2 = csvread('Borings2.csv', 1, 2);
interpolated_layers2 = mySoilSpline(borehole_data2, [1250, 1350, 1450, 1550, 1650]
```

```
Warning: All data points with NaN in their value will be
ignored.
Warning: All data points with NaN in their value will be
ignored.
interpolated_layers2 =
  1.0e+003 *
    1.2500    1.3500    1.4500    1.5500    1.6500
    0.2606    0.2622    0.2638    0.2654    0.2665
    0.0044    0.0055    0.0065    0.0076    0.0084
    0.0039    0.0049    0.0050    0.0044    0.0035
    0.0121    0.0167    0.0213       NaN       NaN
    0.0186    0.0224    0.0231    0.0224    0.0219
    0.0365    0.0310    0.0274    0.0255    0.0249
    0.0314    0.0322    0.0314    0.0298    0.0283
```

Published with MATLAB® 7.10