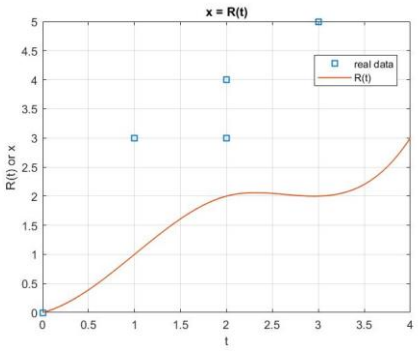
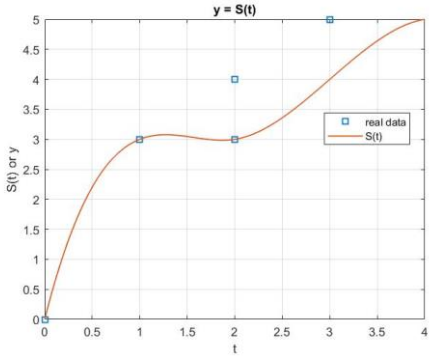
MACM 316 CA6 (RITIKA GOYAL) (301401516)

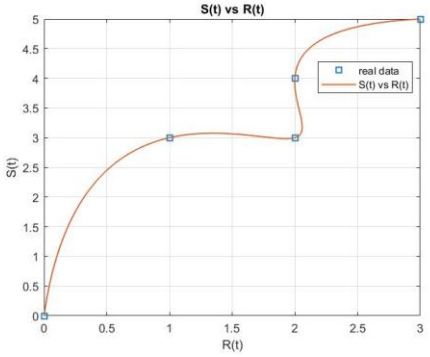
The goal of this assignment is to interpolate data that can not be described by a single-valued function by extending cubic spline and using parametric spline.

t	0	1	2	3	4
x	0.0	1.0	2.0	2.0	3.0
y	0.0	3.0	3.0	4.0	5.0

Part a) Since the given data has two same values of x for different t, the curve displayed by these data points is multi-valued and thus the parametric spline is used to interpolate the curve for these data points. Both x and y are function of t such that x = R(t) and y = S(t). The different spline curve for R(t) vs t, S(t) vs t and S(t) vs R(t) is plotted by using spline function of MATLAB (which has default not-a-knot end-point conditions) and then evaluated by ppval function to display on graph which are mentioned below.

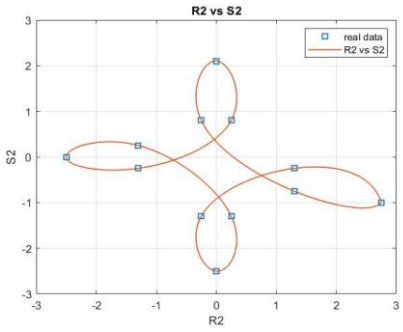


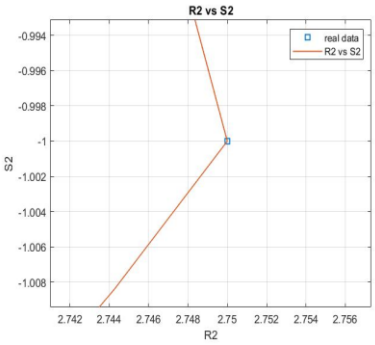




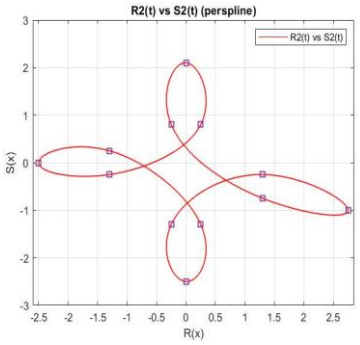
Part b) Just like part a, in this part, the data is multi-valued and thus interpolated by parametric spline using spline function, with 13 points whose curve is plotted bellow. The right leaf at point x = 2.75 and y = -1 is not smooth curve when zoomed in because the spline end points meet at cusp due to use of natural spline function of MATLAB which uses not-a-knot end point conditions (not-a-knot matches third derivative at first interior points). The zoomed in plot is displayed.

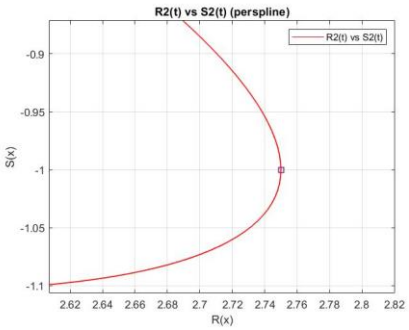
t	0	1	2	3	4
x	2.75	1.3	-0.25	0.0	0.25
y	-1.0	-0.75	0.8	2.1	0.8
t	5	6	7	8	
x	-1.3	-2.5	-1.3	0.25	
y	-0.25	0.0	0.25	-1.3	
t	9	10	11	12	
x	0.0	-0.25	1.3	2.75	
y	-2.5	-1.3	-0.25	-1.0	



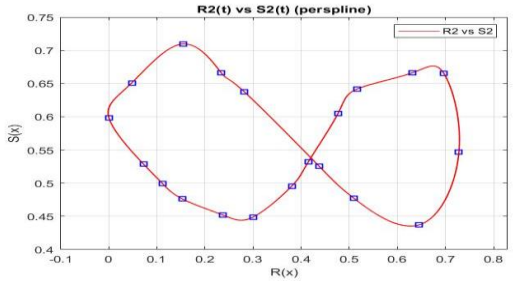


Part c) The plot is plotted using perspline function (extracted from perspline.m) which uses periodic end-point conditions instead of not-a-knot conditions to plot the data in part b. The periodic end-point condition matches first and second derivatives along with third derivative at end point which generates a smoother curve and eliminates cusp at the end points (right most leaf). The zoomed in plot of the end points is displayed on right.





Part d) The smooth custom curve is plotted by using ginput function of MATLAB and x and y values are extracted which is further used to generate smooth spline curve using end-point conditions through perspline function. The minimum points plotted are 20 (Tried to plot the infinity sign).



MATLAB CODE

```
function parametricSplines()
%Part a)
t = [0 1 2 3 4];
x = [0.0 1.0 2.0 2.0 3.0];
y = [0.0 3.0 3.0 4.0 5.0];
value_to_display = linspace(0,4,1000);
xt_spline = spline(t,x);
R = ppval(xt_spline,value_to_display);
yt_spline = spline(t,y);
S = ppval(yt_spline,value_to_display);
plot(x,y,'s',value_to_display,R,'LineWidth',1
);
legend("real data","R(t)");
title("x = R(t)"); xlabel("t");ylabel("R(t)
or x");grid on;
plot(x,y,'s',value_to_display,S,'LineWidth',1
);
legend("real data","S(t)");
title("y = S(t)"); xlabel("t");ylabel("S(t)
or y");grid on;
plot(x,y,'s',R,S,'LineWidth',1);
legend("real data","S(t) vs R(t)");
title("S(t) vs R(t)");
xlabel("R(t)");ylabel("S(t)");grid on;
%part b)
x2 = [2.75 1.3 -0.25 0.0 0.25 -1.3 -2.5 -1.3
0.25 0.0 -0.25 1.3 2.75];
y2 = [-1.0 -0.75 0.8 2.1 0.8 -0.25 0.0 0.25 -
1.3 -2.5 -1.3 -0.25 -1.0];
t2 = [0 1 2 3 4 5 6 7 8 9 10 11 12];
value_to_display2 = linspace(0,12,5000);
xt2_spline = spline(t2,x2);
R2 = ppval(xt2_spline,value_to_display2);
yt2_spline = spline(t2,y2);
S2 = ppval(yt2_spline,value_to_display2);
plot(x2,y2,'s',R2,S2,'LineWidth',1);
legend("real data","R2 vs S2");
title("R2 vs S2");
xlabel("R2");ylabel("S2");grid on;
%part c)
[list1] = perspline(t2,x2);
[list2] = perspline(t2,y2);
[list3] = perspline(list1,list2);
hold on;
plot(x2,y2,'s','LineWidth',1);legend('R2(t)
vs S2(t)');hold off;
% %part d)
figure('position', get(0,'screensize')) %
largest window possible
axes('position', [0 0 1 1])
axis square % make x,y-axes equal
%imshow('myimage.png') % display your drawing
on-screen
[x,y] = ginput; % record mouse clicks until
'Enter'
close % get rid of huge window
save mydatafile.mat x y % save x,y data
points to a file
t3 = linspace(0,1,length(x));
[list1] = perspline(t3,x');
[list2] = perspline(t3,y');
[list3] = perspline(list1,list2);
%load(mydatafile , 'x','y');
hold on; plot(x,y,'sb','LineWidth',1);
legend("R2 vs S2"); hold off;

end
```

PERSPLINE CODE

```
% PERSPLINE: Perform cubic spline
interpolation on a given set
%            of data points, using periodic
end-point conditions.

% NOTE: Must have y(1)=y(end)!!  So this is a
modified version
% of the data used for the other spline
examples.
function [list] = perspline(x,y)
%x = [4.00, 4.35, 4.57, 4.76, 5.26, 5.88]';
%y = [4.77, 5.77, 6.57, 6.23, 4.90, 4.77]';
x = x';
y = y';
n = length(x) - 1;
list =[];

% Set up the matrix
h = diff(x);
diag0 = [1; 2*(h(1:end-1)+h(2:end));
2*h(end)];
A = spdiags([[h;0], diag0, [0;h]], [-1, 0,
1], n+1, n+1);
% Then do a little surgery on the first/last
rows ...
A(1,2)   = 0;
A(1,end) = -1;
A(end,1) = 2*h(1);
A(end,2) = h(1);
dy = diff(y);
% ... and the RHS vector
rhs = 6*[0; diff(dy./h); dy(1)/h(1)-
dy(end)/h(end)];
m = A \ rhs;     % Solve for slopes,
m_i=S''(x_i)

% Compute the cubic polynomial coefficients
a = y;
b = dy./h - h.*m(1:end-1)/2 - h.*diff(m)/6;
c = m(1:end-1)/2;
d = diff(m)./h/6;

% Plot each spline along with the data
for i = 1 : n
  xx = linspace(x(i), x(i+1), 100);
  yy = a(i) + b(i)*(xx-x(i)) + c(i)*(xx-
x(i)).^2 ...
       + d(i)*(xx-x(i)).^3;
  list = [list,yy];
  plot(xx, yy, 'r-')
  hold on
end
plot(x,y,'r','LineWidth',1)
hold off
set(gca, 'XLim', [min(x)-0.1, max( x)+0.1])
xlabel('R(x)'), ylabel('S(x)')
title('R2(t) vs S2(t) (perspline)');
grid on; shg
print -djpeg 'perspline.jpg'
```