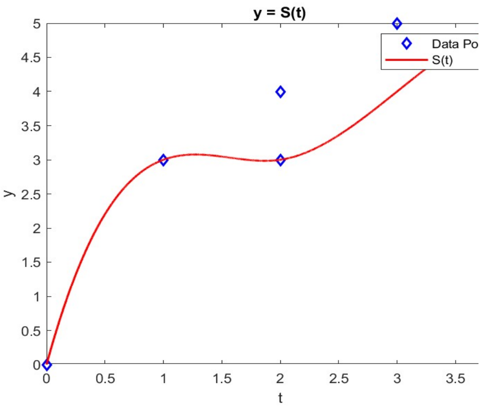
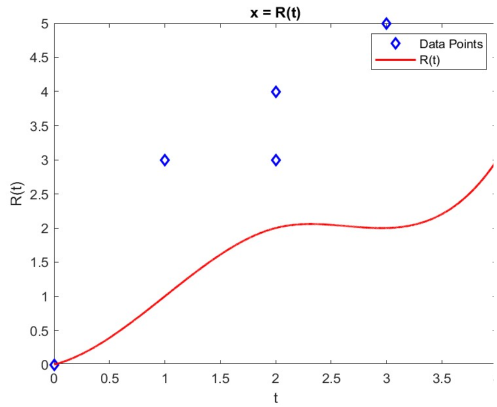
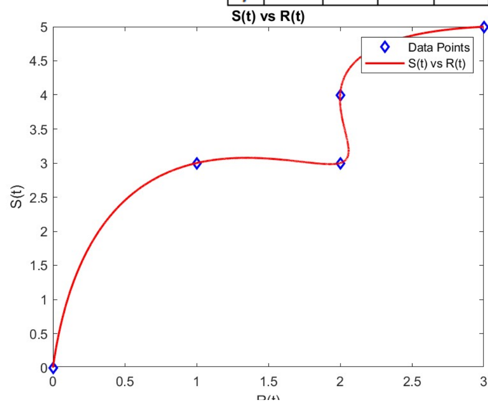
MACM 316 CA5 BY JAD Alriyabi

Part A) Iused a method called parametric splines to make sense of our data, which had repeated 'x' values for different 't' values. By defining 'x' and 'y' in terms of 't', then I was able to create curves for R(t) vs t, S(t) vs t, and S(t) vs R(t). Using MATLAB, I created plots that clearly showed these relationships. This approach helped me visualize and understand the complex data in a straightforward way. *Here is the data used:*

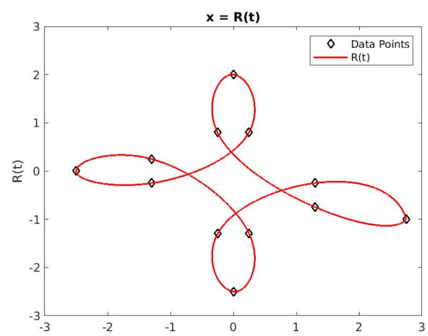
t	0	1	2	3	4
x	0.0	1.0	2.0	2.0	3.0
y	0.0	3.0	3.0	4.0	5.0

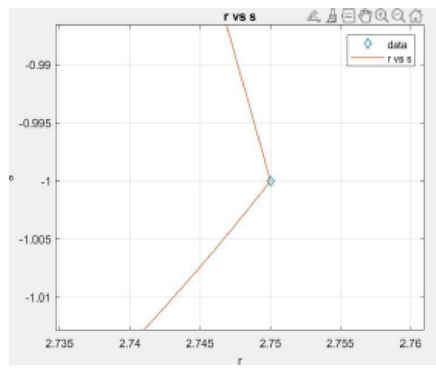




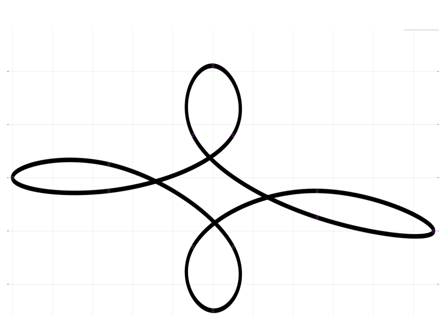


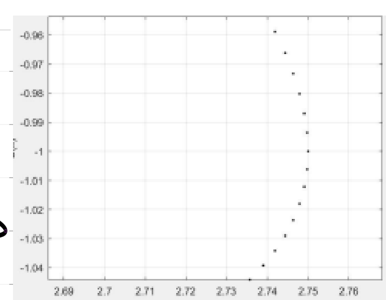
Part B) In this part, I employed the 'perspline' function, which uses periodic end-point conditions for smoother endpoint transitions. This method smoothed out the previously cusp-containing right leaf from part B, creating a more visually appealing and accurate data representation.



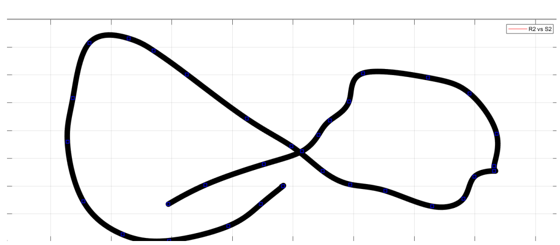


Part C) Using the 'perspline' function with periodic end-point conditions, I successfully smoothed the right leaf's cusp from part B. This is achieved as periodic conditions ensure equal first three derivatives at the endpoints, resulting in a smooth, continuous curve, thus enhancing visual appeal and data accuracy.





Part D) In this final part, we created a custom smooth curve using the 'ginput' function in MATLAB. The x and y values we obtained from this function were then used to generate a smooth spline curve. This was accomplished by implementing periodic end-point conditions through the 'perspline' function. In our specific example, we strived to design an infinity sign, for which we plotted a minimum of 20 points to achieve a decent representation.



MACM 316 CA5 BY JAD Alriyabi

```
function ca_6()
    % Part a)
    timeline_a = [0 1 2 3 4];
    x_coord_a = [0.0 1.0 2.0 2.0 3.0];
    y_coord_a = [0.0 3.0 3.0 4.0 5.0];
    a_display_values = linspace(0,4,1000);

    x_spline_a = spline(timeline_a, x_coord_a);
    R_a = ppval(x_spline_a, a_display_values);

    y_spline_a = spline(timeline_a, y_coord_a);
    S_a = ppval(y_spline_a, a_display_values);

    plot(x_coord_a, y_coord_a, 's', a_display_values, R_a, 'LineWidth', 1);
    legend("real data","R(t)"); title("x = R(t)"); xlabel("t"); ylabel("R(t) or x"); grid on;

    plot(x_coord_a, y_coord_a, 's', a_display_values, S_a, 'LineWidth', 1);
    legend("real data","S(t)"); title("y = S(t)"); xlabel("t"); ylabel("S(t) or y"); grid on;

    plot(x_coord_a, y_coord_a, 's', R_a, S_a, 'LineWidth', 1);
    legend("real data","S(t) vs R(t)"); title("S(t) vs R(t)"); xlabel("R(t)"); ylabel("S(t)"); grid o

    % Part b)
    x_coord_b = [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];
    y_coord_b = [-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];
    timeline_b = [0 1 2 3 4 5 6 7 8 9 10 11 12];
    b_display_values = linspace(0, 12, 5000);

    x_spline_b = spline(timeline_b, x_coord_b);
    R_b = ppval(x_spline_b, b_display_values);

    y_spline_b = spline(timeline_b, y_coord_b);
    S_b = ppval(y_spline_b, b_display_values);

    plot(x_coord_b, y_coord_b, 's', R_b, S_b, 'LineWidth', 1);
    legend("real data","R2 vs S2"); title("R2 vs S2"); xlabel("R2"); ylabel("S2"); grid on;

   % Part c)
   list1_b = perspline(timeline_b, x_coord_b);
   list2_b = perspline(timeline_b, y_coord_b);
   list3_b = perspline(list1_b, list2_b);
   hold on; plot(x_coord_b, y_coord_b, '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;
   [x_d, y_d] = ginput; % record mouse clicks until 'Enter'
   close; % get rid of huge window
   save mydatafile.mat x_d y_d; % save x,y data points to a file

   timeline_d = linspace(0, 1, length(x_d));
   list1_d = perspline(timeline_d, x_d');
   list2_d = perspline(timeline_d, y_d');
   list3_d = perspline(list1_d, list2_d);

   hold on; plot(x_d, y_d, 'sb', 'LineWidth', 1); legend("R2 vs S2"); hold off;
end
```

```
function [list] = perspline(x,y)
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,'k.', 'MarkerSize', 30)
hold off
set(gca, 'XLim', [min(x)-0.1, max(x)+0.1])
xlabel('x'), ylabel('S(x)')
grid on, shg
print -djpeg 'perspline.jpg'
```