MACM 316 – Computing Assignment #6

Due Date: Friday December 4 at 11:00pm.

Submission Instructions: You must upload one .pdf file to **Crowdmark** that consists of ONLY 2 pages: page 1 is your report which should fit all of your results, discussion, data and figures into a single page; and page 2 is a listing of your code. The deadline is **11:00pm** on the due date. The actual due time is set to 11:05pm – if Crowdmark indicates that you submitted late then you will be assigned a grade of 0 on this assignment. Your TA has emailed you a Crowdmark link that you will use for the entire semester to upload your assignment solutions.

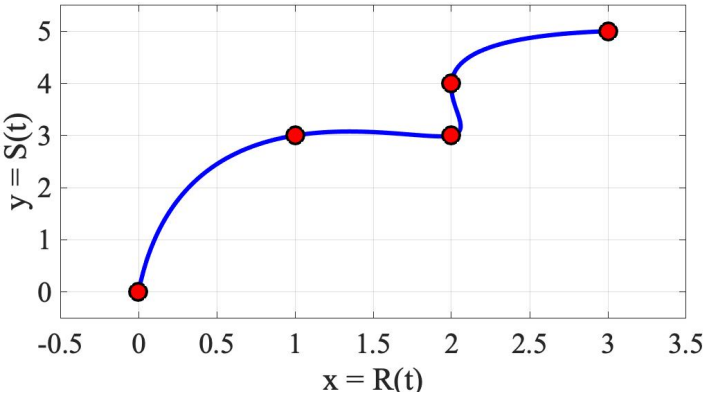
- Review the **Guidelines for Computing Assignments** carefully.
- If you have questions about this assignment or MATLAB, you can obtain help by posting your questions to the "Computing Assignment" discussion board in Canvas. This discussion board will be checked regularly and monitored continuously during computational workshop hours.

Parametric Splines For Multi-Valued Functions

In this assignment, your aim is to extend the definition of a cubic spline to interpolate data that cannot be described by a single-valued function.

(a) As a first example, consider the data points listed in the table (below, left) which are sampled from an underlying smooth curve pictured in the plot (below, right):

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



Clearly, the curve passes through multiple y-values at $x = 2$, meaning that the underlying curve is *multiply-defined* or *multi-valued* – it's not a function!! This means that the usual cubic spline approach based on trying to interpolate a function of the form $y = f(x)$ will not work. The trick to dealing with such data is to use a *parametric spline* that is based on a parametric description of the curve: $x = R(t)$ and $y = S(t)$ for some parameter t. For this example, it's easiest to assign parameter values $t = 0, 1, 2, 3, 4$ corresponding to the index of the points in the table. The particular choice of t values is actually not important as long as t is monotonically increasing[‡]. Because $R(t)$ and $S(t)$ are now both functions of t, we can interpolate them with two separate splines.[§]

Based on the t-x-y data from the table, generate two cubic splines $R(t)$ and $S(t)$ using the MATLAB `spline` function with its default not-a-knot end-point conditions. Provide three separate plots: of R versus t, S versus t, and S versus R (the last of which will reproduce the plot above).

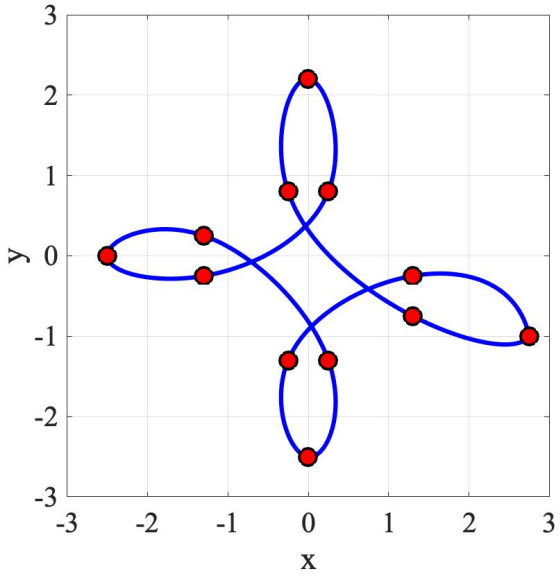
[‡]The parameter t can also be monotonically *decreasing* – try that out if you're curious!

[§]Another multiply-defined function you will be familiar with is the unit circle, $x^2 + y^2 = 1$, which can be written in parametric form as $x = R(t) = \cos t$ and $y = S(t) = \sin t$, with parameter $0 \leqslant t \leqslant 2\pi$.

(b) You will now use a parametric spline to interpolate the more complicated "four-leaf" curve pictured below. The table lists coordinates of 13 points (x_i, y_i), $i = 0, 1, \ldots, 12$ that lie along this curve.

t	x	y
0	2.75	-1.0
1	1.3	-0.75
2	-0.25	0.8
3	0.0	2.1
4	0.25	0.8
5	-1.3	-0.25
6	-2.5	0.0

t	x	y
7	-1.3	0.25
8	0.25	-1.3
9	0.0	-2.5
10	-0.25	-1.3
11	1.3	-0.25
12	2.75	-1.0



Determine the parametric spline $x = R(t)$, $y = S(t)$ that interpolates this set of points (x, y), again using MATLAB's `spline` function. Plot your parametric spline curve and verify (by zooming in on your plot) that the right-most "leaf" is different from the other three leaves in that it is not smooth – the spline endpoints meet at a *cusp*.

(c) For a periodic curve like the one in part (b), it is more appropriate to use periodic end-point conditions instead of not-a-knot conditions. That is, we should take $R_0'(t_0) = R_{n-1}'(t_n)$ and $R_0''(t_0) = R_{n-1}''(t_n)$, and similarly for $S(t)$. Use the MATLAB code `perspline.m` posted on Canvas to generate the two periodic cubic splines approximating $R(t)$ and $S(t)$, plot your parametric curve, and compare it to what you obtained in part (b). Verify that the cusp is indeed eliminated.

(d) You can now get creative! Start by drawing a periodic parametric curve, which can be *any smooth curve* of your own design as long as the endpoints meet at the same location. Your curve should be defined by at least 20 spline points (but not more than 40) and there should be *at least one* instance where your curve crosses itself (such as the four "leaf crossings" in part (b)).

To generate your list of points, you may find it helpful to use MATLAB's built-in function `ginput` (graphical input from mouse) which allows you to draw your parametric curve on the screen by clicking with the mouse at a sequence of locations in a plotting window, and then outputs the x and y coordinates of the "clicked" points. If you type `help ginput`, you can see that it reads mouse clicks within the plotting window until the "enter" key is pressed, after which it returns two vectors of coordinates. You may find the following sequence of MATLAB commands helpful:

```
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
```

The `save` command saves your data points to a file that can later on be read back in using the `load` command. You may then use your (x, y) data as input to `perspline.m` in the same way you did in part (c) to construct your parametric spline and plot your results.