MACM 316 – Computing Assignment #5

Due Date: Wednesday November 29 at 11:00pm.

Instructions: Upload to Crowdmark a single PDF file that consists of only two pages: page #1 is your report (containing all discussions, data and figures), and page #2 is a listing of your Matlab code. The assignment is due at **11:00pm sharp!** If Crowdmark indicates that your submission was late, then you will be assigned a grade of zero, with no exceptions – so don't leave submitting until the last minute. When this CA is released, you should receive an e-mail containing a link for uploading your assignment, so make sure you save this message.

- Carefully review the **"Guidelines for Computing Assignments"** posted on Canvas.
- If you have questions about this assignment then you can obtain help in the computational workshops, tutorials, or the "Computing Assignment" discussion group on Canvas.

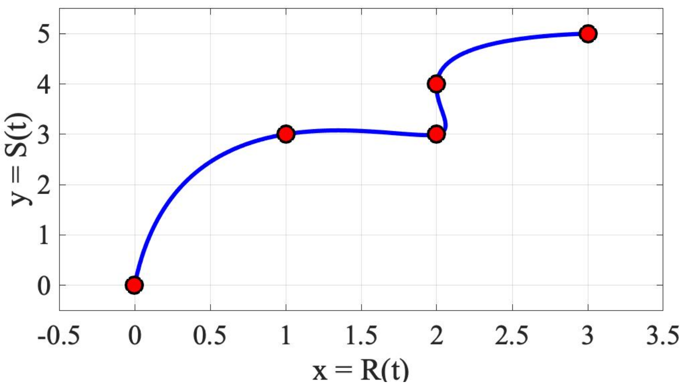
CA5 – Parametric splines for general curves

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

multi-valued!

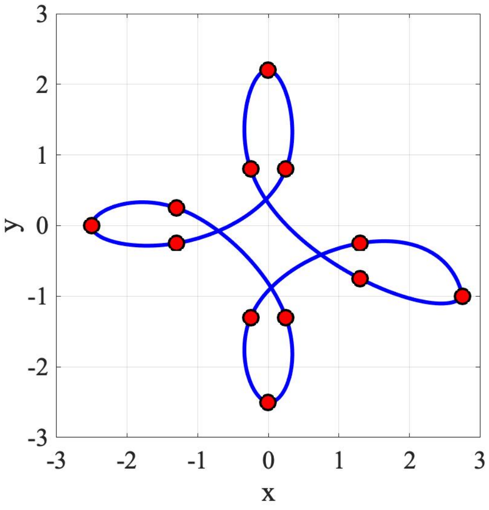


Clearly, the curve passes through multiple y-values at $x = 2$, meaning that the underlying curve is *multi-valued* and doesn't pass the *"vertical line test"* – so it's not a function!! This means that the usual cubic spline approach based on trying to interpolate a function of the form $y = f(x)$ will not work. The trick to dealing with data like this 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 cubic splines.

Based on the t-x-y data from the table, generate two cubic splines $x = R(t)$ and $y = S(t)$ 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 should reproduce the plot above).

(b) Next use a parametric spline to interpolate the more interesting "four-leaf" curve pictured below. The table lists coordinates (x_i, y_i), $i = 0, 1, \ldots, 12$, for 13 points lying 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
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 – that is, 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 two periodic cubic spline approximations for $R(t)$ and $S(t)$, plot your parametric curve, and compare to what you obtained in part (b). Verify that the cusp is eliminated.

(d) You can now get creative! Start by drawing your own 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 both ...

- *interesting:* it should cross itself at least once, such as the 4 "leaf crossings" in part (b), and
- *not too complicated:* so that it can be approximated by a cubic spline with 20–40 points.

To generate your list of 20–40 data points, you may find it helpful to save your drawing as an image file, and then use Matlab's `ginput` (graphical input from mouse). The built-in function `ginput` allows you to select points by clicking with the mouse at a sequence of along your parametric curve in the plotting window, and then outputs all x and y coordinates of the points you selected. If you type `help ginput`, you will see that it records 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'
save mydatafile.mat x y      % save x,y data points to a file
close                        % delete the huge window (if desired)
```

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.

[†]If you are looking for artistic inspiration, you can find a host of examples by entering "one line drawings" in a Google Images search. Beware that many of these images are too complex for this assignment, so choose wisely! (or simplify).