1 Splines

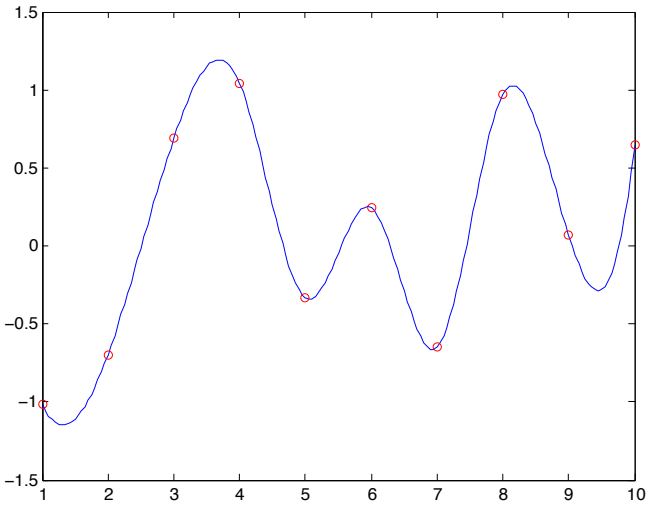

Figure 1: Plot of a spline between some specified knot points shown as red circles

In this project you will experiment with interpolating splines which are commonly used in graphics and in other applications. To begin with you are given a vector containing the value of some function $Y(x)$ for $x = 1, 2, 3, 4, \cdots, n$. Let's denote these sample values $y_1, y_2, y_3, y_4, \cdots, y_n$. Given these values we want to construct the values of the function at all the intermediate values of x like $x = 7.23$. To do this we will say that on each interval, $x = [1, 2]$, $x = [2, 3]$ etc., we will approximate the function with a cubic polynomial as follows:

$$Y(x) = Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3 \quad t = (x - i) \in [0, 1] \tag{1}$$

If we are given n values $y_i \cdots y_n$ there are clearly $(n-1)$ intervals and hence $4(n-1)$ polynomial coefficients to solve for. We can evaluate the value and the derivatives of the polynomials at the endpoints as follows:

$$Y_i(0) = a_i \quad Y_i(1) = a_i + b_i + c_i + d_i$$

$$Y_i'(t) = b_i + 2c_i t + 3d_i t^2$$

$$Y_i'(0) = b_i \quad Y_i'(1) = b_i + 2c_i + 3d_i$$

$$Y_i''(t) = 2c_i + 6d_i t$$

$$Y_i''(0) = 2c_i \quad Y_i''(1) = 2c_i + 6d_i$$

We will enforce the constraint that at each of the interior knot points, $y_2 \cdots y_{n-1}$, the function and its first and second derivatives should vary continuously. Let $D_i = Y_i'(0) = Y_{i-1}'(1)$ denote the derivative of the spline at y_i.

$$
\begin{aligned}
Y_i(0) &= y_i = a_i && (2)\\
Y_i(1) &= y_{i+1} = a_i + b_i + c_i + d_i && (3)
\end{aligned}
$$

$$
\begin{aligned}
Y_i'(0) &= D_i = b_i && (4)\\
Y_i'(1) &= D_{i+1} = b_i + 2c_i + 3d_i && (5)
\end{aligned}
$$

Rearranging the Equations 3 and 5 to eliminate d_i yields.

$$3y_{i+1} - D_{i+1} = 3y_i + 2D_i + c_i$$

$$\Rightarrow c_i = 3(y_{i+1} - y_i) - 2D_i - D_{i+1}$$

And we can then rearrange Equation 3 to express d_i as follows:

$$d_i = (y_{i+1} - y_i) - D_i - c_i$$

$$d_i = 2(y_i - y_{i+1}) + D_i + D_{i+1}$$

Summarizing:

$$
\begin{aligned}
a_i &= y_i && (6)\\
b_i &= D_i && (7)\\
c_i &= 3(y_{i+1} - y_i) - 2D_i - D_{i+1} && (8)\\
d_i &= 2(y_i - y_{i+1}) + D_i + D_{i+1} && (9)
\end{aligned}
$$

So given the y_i and the D_i we can easily recover the polynomial coefficients in each interval a_i, b_i, c_i and d_i. Since we are given the y_i we can simply focus on solving for the D_i.

Applying the constraint that the second derivative needs to be consistent we get.

$$
\begin{aligned}
Y_{i-1}''(1) &= Y_i''(0) && (10)\\
\Rightarrow 2c_{i-1} + 6d_{i-1} &= 2c_i && (11)\\
\Rightarrow c_{i-1} + 3d_{i-1} &= c_i && (12)
\end{aligned}
$$

Substituting for c_{i-1}, d_{i-1} and c_i yields:

$$
\begin{aligned}
(3(y_i - y_{i-1}) - 2D_{i-1} - D_i) + 3(2(y_{i-1} - y_i) + D_{i-1} + D_i) &= 3(y_{i+1} - y_i) - 2D_i - D_{i+1}\\
3(y_{i-1} - y_i) + D_{i-1} + 2D_i &= 3(y_{i+1} - y_i) - 2D_i - D_{i+1}
\end{aligned}
$$

Which ultimately simplifies to:

$$D_{i-1} + 4D_i + D_{i+1} = 3(y_{i+1} - y_{i-1})$$

We have one such equation for each of the interior knot points $[y_2 \cdots y_{n-2}]$ so we need two more equations so we can solve for all n values of D_i.

One way to get the remaining two equations is to set the second derivative to zero at each of the end points. That is $Y_1''(0) = Y_n''(1) = 0$. This leads to what is referred to as the natural spline equations.

$$
\begin{aligned}
Y_1''(0) &= 0 && (13)\\
\Rightarrow 2c_i &= 0 && (14)\\
\Rightarrow 2D_1 + D_2 &= 3(y_2 - y_1) && (15)
\end{aligned}
$$

$$
\begin{aligned}
Y_n''(1) &= 0 && (16)\\
\Rightarrow 2c_{n-1} + 6d_{n-1} &= 0 && (17)\\
\Rightarrow c_{n-1} + 3d_{n-1} &= 0 && (18)\\
\Rightarrow (3(y_n - y_{n-1}) - 2D_{n-1} - D_n) + 3(2(y_{n-1} - y_n) + D_{n-1} + D_n) &= 0 && (19)\\
\Rightarrow 2D_n + D_{n-1} &= 3(y_n - y_{n-1}) && (20)
\end{aligned}
$$

When we add in these two equations we end up with the following set of linear equations in the D_i

$$
\begin{pmatrix}
2 & 1 & & & & \\
1 & 4 & 1 & & & \\
 & 1 & 4 & 1 & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & 1 & 4 & 1 \\
 & & & & 1 & 2
\end{pmatrix}
\begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_{n-1} \\ D_n \end{pmatrix}
= 3
\begin{pmatrix} y_2 - y_1 \\ y_3 - y_1 \\ y_4 - y_2 \\ \vdots \\ y_n - y_{n-2} \\ y_n - y_{n-1} \end{pmatrix}
\tag{21}
$$

Note that the system is *tridiagonal* which means that it can be solved very efficiently even for a large number of knot points. This is one of the reasons for the popularity of splines.

1.1 Assignment Part 1

Your assignment is to write a Matlab function with the following signature:

```
coeffs = my_spline (y)
```

Which takes as input a vector, y holding the knot values $y_1 \cdots y_n$ and produces as output an array called coeffs with $(n-1)$ rows and 4 columns containing the spline coefficients in each of the $(n-1)$ intervals from first to last. The first column of the matrix corresponds to the a_i coefficient, the second to the b_i coefficient, the third to the c_i coefficient and the fourth to the d_i coefficient.

The main job of your function will be to setup and solve the sparse tridiagonal system shown in Equation 21. You can allocate the matrix using Matlab's `spalloc` function and then fill in the coefficients as shown in the example code handed out with this project and used in the class lecture. Matlab also has other functions for creating sparse matrices which you could explore like `sparse` and `spdiags` but these are a bit more complex to use.

Once you have the matrix set up you can solve the linear system in the usual manner using Matlab's backslash operator which will recognize and exploit the tridiagonal structure. Given the D_i values and the y_i values you can easily solve for the coefficients as noted earlier.

1.2 Assignment Part 2

You are asked to write a Matlab function with the following signature.

```
y = eval_spline (coeffs, s)
```

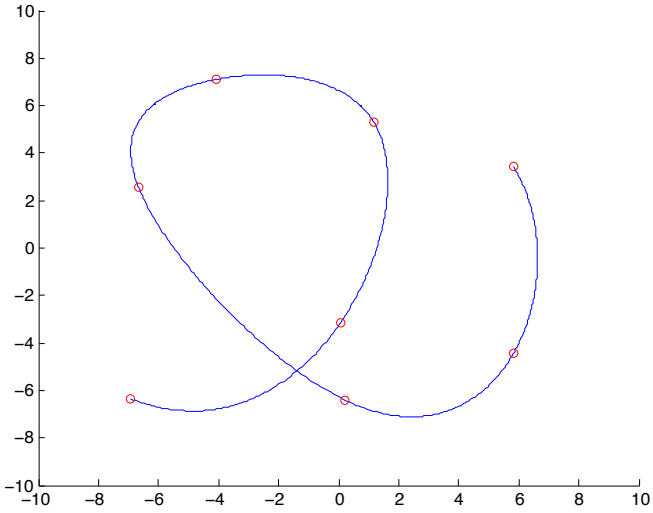

Figure 2: Plot of a 2D spline between some specified knot points shown as red circles

which takes as input the array, coeffs, produced by your `my_spline` function and a scalar value, s, which indicates the value at which you would like to evaluate the spline. For example s=1 should produce the first knot point, s = n should produce the last knot value, and an intermediate value of s, say s = 7.3 should evaluate the cubic polynomial in Equation 1 using the coefficients on the seventh row of coeffs with a t value calculated as follows $t = 7.3 - 7 = 0.3$. For values of s less than 1 or greater than n the function should return `NaN`. (Hint you may find Matlab's `floor` function useful here)

1.3 Assignment Part 3

Once you have written your functions you need to provide a script to test that the functions actually work. You need to write a script entitled `Script1` which should begin by creating a vector y with n entries where n is at least 20. Then you should call your `my_spline` function to create an interpolating spline through those y values. Finally you should write code to produce a plot like the one shown in Figure 1 by making use of your `eval_spline` function. Please use at least 20 points in each interval so you get a smooth plot. Make sure to plot the knot points using red circles so that they can be seen easily. You should write your script so that it is easy to change the number of knot points, n.

1.4 Assignment Part 4

You are going to use the functions you have already written to create 2D spline curves like the ones shown in Figure. 2. Specifically you need to write a Matlab script entitled `Script2`. Which first creates a figure using the following command: `axis([-10 10 -10 10])`. You should then allows the user to select a few points in the figure with the `getpts` command. This `getpts` commands returns the x and y coordinates of the points that the user selects. You can then use the spline function to build two splines, one through the x coordinates and another through the y coordinates. You can then create a smooth curve going through the selected points by using the `eval_spline` function. Your script should use the plot function to create a figure like the one shown in Figure 2. Please use at least 50 points in each interval so you get a smooth curve.

Please draw red circles around the user selected knot points.

1.5 Extra Credit : 20 pts

The end point conditions that were suggested earlier lead to the natural spline equations as we have described. There are other ways of coming up with the final two constraints that lead to slightly different spline solutions. One very popular variant is to apply what are known as the not-a-knot conditions which stipulate that the third derivative should be continuous at knot points 2 and $(n-1)$.

For this part of the assignment you are asked to explore the implications of this choice. Specifically work out the linear equations that result from this choice and show how the resulting equations can be incorporated into the linear system shown in Equation 21 replacing the previous endpoint equations. Note that you must ensure that the resulting system of linear equations remains tridiagonal.

You should write another variant of the spline function called `my_spline2` which has the same signature as `my_spline` but enforces these not-a-knot conditions and another version of Script1 called `Script1_2` which tests your function `my_spline_2` and compares it to the curve you get from the original function `my_spline`.

2 Submission Instructions

You will turn in all of the code for this project on Canvas.
All of the files for the assignment should be in a single folder on your computer. You will create a zip archive of that folder by right clicking on the folder and selecting the appropriate action. Once the zip archive has been created you can upload it to Canvas. Note that the Canvas site will only accept files with a .zip extension. You should submit all of the functions and scripts we asked you to write or edit along with all of the functions we provided you with.