ENPH 213 Laboratory 4, 2020

The objective of this week's laboratory is to give you some experience writing code for curve fitting.

Part A – Due at the end of the lab period

The purpose of Part A is compare two different approaches to polynomial curve fitting of sampled data. The data is made of $n+1$ data pairs $(x_i, y_i), i = 1 \dots n+1$ forming n regions.

Task A1: Single polynomial fit

Your first task is to interpolate a series of x-y values:

$$x = [x_1, x_2, \dots, x_n, x_{n+1}]$$

$$y = [y_1, y_2, \dots, y_n, y_{n+1}]$$

using a single polynomial $P(x, y)$ constrained to pass through each of the $n+1$ points. The polynomial is defined by the equation:

$$P(x, y) = a_0 + a_1 x + a_2 x^2 + \cdots a_n x^n$$

which has $n+1$ unknowns, $a_0, a_1, \dots a_n$. Each data pair then represents an equation and each coefficient an unknown. This can be written as a system of equations:

$$\begin{bmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n+1} & \cdots & {x_{n+1}}^n \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_{n+1} \end{bmatrix}$$

where the leftmost matrix is called the Vandermonde matrix.

Write a function `polyInterp` with the following inputs/outputs:

INPUTS	
`x`	1D array of x values, with dimensions (1, n+1)
`y`	1D array of y values, with dimensions (1, n+1)

OUTPUT	
`a`	1D array of coefficients defining the polynomial fit, with dimensions **(n+1, 1)**

Your function will need to first create the Vandermonde matrix and then solve for the coefficients. For this week you are allowed to use the built-in matrix left division operator, a backslash (refer to the Matlab documentation for `mldivide`) to solve the system of equations. Use the test function **test_polyInterp.m** from onQ to test your function.

Task A2: Natural cubic spline fit

Your second task is to perform a cubic spline fit on a similar set of data pairs. A cubic spline fit is defined by a series of equations of the form:

$$g_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

One for **each** of the n regions, where the ith region is defined by $x_i < x < x_{i+1}$. You now have $4n$ unknown coefficients (4 coefficients for each equation, n equations). To solve for these coefficients you need $4n$ constraints. The first constraint is that each of the cubic functions must pass through the starting and ending x-y pairs:

$$g_i(x_i) = y_i$$
$$g_i(x_{i+1}) = y_{i+1}$$

This takes care of half of the required constraints. The second constraint is that the first and second derivatives of each of the cubic functions are continuous at the ending x-y pair:

$$\left.\frac{dg_i(x)}{dx}\right|_{x=x_{i+1}} = \left.\frac{dg_{i+1}(x)}{dx}\right|_{x=x_{i+1}}$$

$$\left.\frac{d^2g_i(x)}{d^2x}\right|_{x=x_{i+1}} = \left.\frac{d^2g_{i+1}(x)}{d^2x}\right|_{x=x_{i+1}}$$

Giving $n-1$ constraints each ($1 < i < n-1$), leaving us with only 2 more to define. Leaving these last two constraints open for the time being, one can derive the following recursion formula:

$$3\left[\left(\frac{y_{i+2} - y_{i+1}}{h_{i+1}}\right) - \left(\frac{y_{i+1} - y_i}{h_i}\right)\right] = c_i h_i + c_{i+1}(2h_{i+1} + 2h_i) + c_{i+2}h_{i+1}$$

The individual coefficients are given by the formulas:

$$a_i = y_i$$
$$b_i = \frac{(y_{i+1} - y_i)}{h_i} - \frac{(2c_i + c_{i+1})h_i}{3}$$
$$d_i = \frac{c_{i+1} - c_i}{3h_i}$$

where,

$$h_i = x_{i+1} - x_i$$

There are several alternatives to the last two constraints:

1. Natural Spline:

$$g_1^{(2)}(x_1) = 0, \qquad g_n^{(2)}(x_{n+1}) = 0$$

2. End Slope Spline:

$$g_1^{(1)}(x_1) = y_0^{(1)}, \qquad g_n^{(1)}(x_{n+1}) = y_n^{(1)}$$

   where $y_0^{(1)}$ and $y_n^{(1)}$ are known.

3. Periodic Spline:

$$g_1^{(1)}(x_1) = g_n^{(1)}(x_{n+1}), \qquad g_1^{(2)}(x_1) = g_n^{(2)}(x_{n+1})$$

4. Not-a-Knot Spline (MATLAB defaults to this alternative):

$$g_1^{(3)}(x_2) = g_2^{(3)}(x_2), \qquad g_{n-1}^{(3)}(x_n) = g_n^{(3)}(x_n)$$

If we assume natural spline boundary conditions, we can express the problem in matrix form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & \cdots & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\ 0 & \cdots & 0 & 0 & h_{n-1} & 2(h_{n-1}+h_n) & h_n \\ 0 & \cdots & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-1} \\ c_n \\ c_{n+1} \end{bmatrix} = \begin{bmatrix} 0 \\ 3\left[\frac{y_3-y_2}{h_2} - \frac{y_2-y_1}{h_1}\right] \\ 3\left[\frac{y_4-y_3}{h_3} - \frac{y_3-y_2}{h_2}\right] \\ \vdots \\ 3\left[\frac{y_n-y_{n-1}}{h_{n-1}} - \frac{y_{n-1}-y_{n-2}}{h_{n-2}}\right] \\ 3\left[\frac{y_{n+1}-y_n}{h_n} - \frac{y_n-y_{n-1}}{h_{n-1}}\right] \\ 0 \end{bmatrix}$$

of the form $H * c = Y$.

Write a function `natCubSpline` with the following inputs/outputs:

INPUTS	
`x`	1D array of y values, with dimensions (1, n+1)
`y`	1D array of y values, with dimensions (1, n+1)

OUTPUT	
`a`	1D array of coefficients, with dimensions **(n, 1)**
`b`	1D array of coefficients, with dimensions **(n, 1)**
`c`	1D array of coefficients, with dimensions **(n, 1)**
`d`	1D array of coefficients, with dimensions **(n, 1)**

Your function should first create the H and Y matrices and then solve for the c coefficients. Finally, using the c coefficients, your function should solve for the $a, b,$ and d coefficients. Use the test function **test_natCubSpline.m** from onQ to test your function.

Task A3: Periodic cubic spline fit

Now assume periodic boundary conditions:

$$g_1^{(1)}(x_1) = g_n^{(1)}(x_{n+1}), \qquad g_1^{(2)}(x_1) = g_n^{(2)}(x_{n+1})$$

You will only need to change the first row and last rows of both the H and Y matrices. As a start, the periodic boundary conditions can be written in terms of the coefficients a, b, c and d as:

$$b_1 = b_n + 2c_n h_n + 3d_n h_n^2$$

$$c_1 = c_n + 3d_n h_n$$

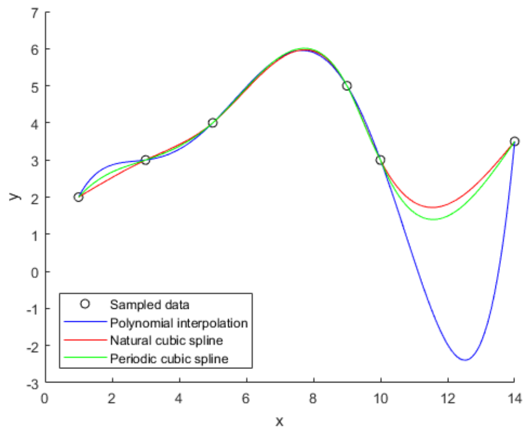
Once you've worked out the math, write a function `perCubSpline(x, y)` with the following inputs/outputs:

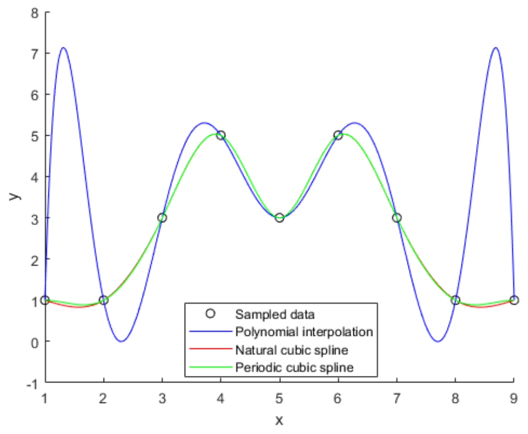
INPUTS	
`x`	1D array of y values, with dimensions (1, n+1)
`y`	1D array of y values, with dimensions (1, n+1)

OUTPUT	
`a`	1D array of coefficients, with dimensions **(n, 1)**
`b`	1D array of coefficients, with dimensions **(n, 1)**
`c`	1D array of coefficients, with dimensions **(n, 1)**
`d`	1D array of coefficients, with dimensions **(n, 1)**

Use the test function **test_perCubSpline.m** from onQ to test your function.

Finally, write a function `plotCurveFits(x, y)` that takes as its input $n + 1$ x-y data pairs performs a polynomial interpolation, natural cubic spline fit and periodic cubic spline fit, and plots the fit curves using 100 data points per region, overlaid on the sampled data points. Your plots should look something like this:





Part A: Submission List

When complete, please submit the following files to the **Lab 4: Part A** dropbox on onQ for marking:

- `polyInterp.m`
- `natCubSpline.m`
- `perCubSpline.m`
- `plotCurveFits.m`
- Any other functions you wrote that the above files use

Part B: Due Sunday @ 9PM

Task B1: Least squares polynomial

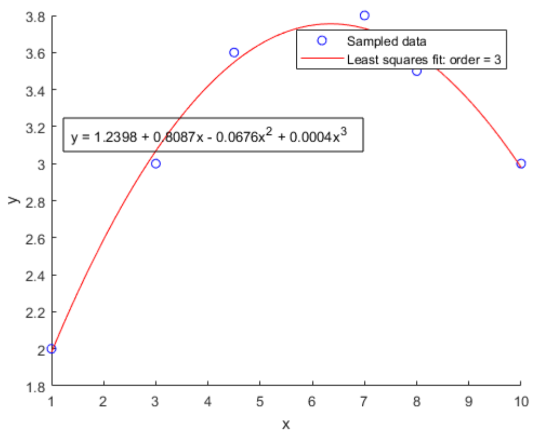
Using the least squares method discussed in class, write a function `leastSquaresPoly` with the following inputs/outputs:

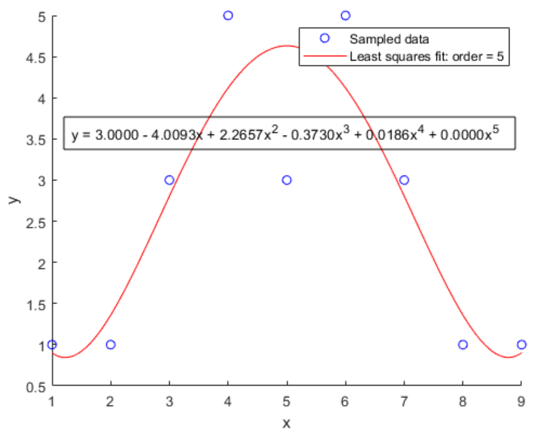
INPUTS	
`x`	1D array of x values, with dimensions **(1, n+1)**
`y`	1D array of y values, with dimensions **(1, n+1)**
`M`	Order of polynomial fit

OUTPUT	
`a`	1D array of coefficients defining the polynomial fit, with dimensions **(M+1, 1)**

Note that $N + 1 \geq \ M + 1$, *i.e.* the number of data points is greater than or equal to the number of unknown coefficients. You should again have to create a matrix to solve for the $M + 1$ required coefficients. Use the test function **test_leastSquaresPoly.m** from onQ to test your function.

Write a second function `plotLeastSquaresPoly(x, y, M)` that plots the polynomial fit, using 100 times as many data points, overlaid on the sampled data points. Use Matlab's `annotation` function to label the polynomial curve fit with its equation. Use the test function **test_plotLeastSquaresPoly.m** from onQ to test your function. Your plots should look something like this:





Task B2: Generating a symmetric airfoil

Write a **new** function `a=leastSquaresPoly2(x,y,M)` that takes the same inputs as `leastSquaresPoly` but finds the coefficients for the modified polynomial:

$$y = a_1 x^{1/2} + a_2 x^1 + \cdots a_M x^{M-1} + a_{M+1} x^M$$

You should find that the coefficient matrices involved in solving the modified least squares problem are mostly the same as those in Task B1, but with some slight modifications. Use the test function **test_leastSquaresPoly2.m** from onQ to test your function

Write a final function `plotAirfoil(x,y,M,s)` that takes as its input a set of x-y data pairs, the desired order of the modified polynomial fit `M` and a scaling factor `s`, and outputs two figures.

Figure 1 should include:

- The original data pairs
- The Mth order modified polynomial fit as found using your `leastSquaresPoly2` function and the original data pairs, using 100 times the number of given data points
- A reflection of the Mth order modified polynomial fit about the x-axis (y = 0)

Figure 2 should include:

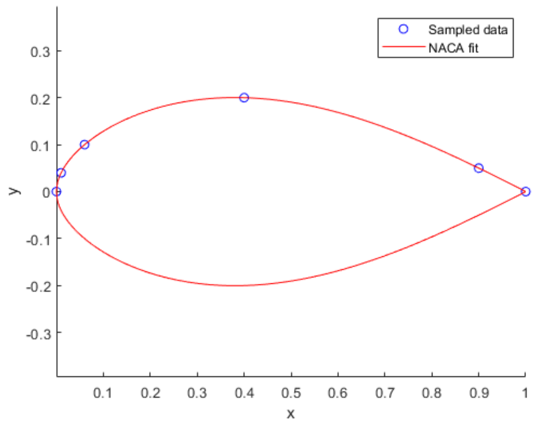
- The original data pairs
- The Mth order modified polynomial fit as found using your `leastSquaresPoly2` function and the original data pairs, using 100 times the number of given data points
- Data pairs of the original x values but with scaled versions of the original y values according to:

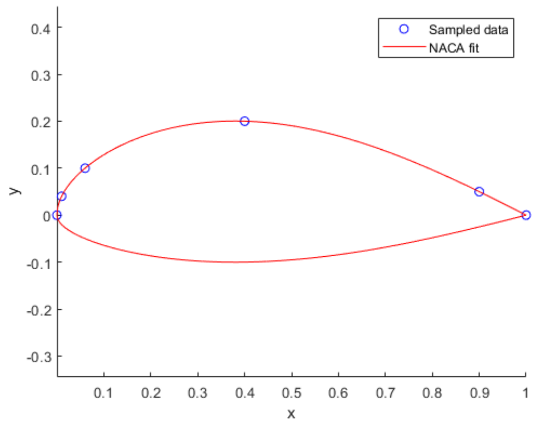
```
yScaled = s.*y;
```

- The Mth order modified polynomial fit as found using your `leastSquaresPoly2` function and the scaled data pairs

You do **not** need to include any annotation on this plot but still make sure to include a legend and axis labels. Also use the command `axis('equal')` so that the data units are the same length along each axis.

Use the test function **test_plotAirfoil.m** from onQ to test your function. The outputted graphs should have a typical airfoil shape, the modified polynomial function above for order $M = 4$ is actually based on a National Advisory Committee for Aeronautics (NACA) series of airfoils, where y is the height of the airfoil. Most airfoils are asymmetric with a flatter profile on the bottom surface. The first airfoil outputted by the test function should be symmetric and the maximum thickness of the section should be about a third of the way back from the leading edge. The second airfoil should be antisymmetric, noting that `s` is negative, so the scaled data pairs and corresponding modified polynomial fit are plotted along the bottom of the airfoil. Example plots are shown below.





Part B: Submission List

When complete, please submit the following files to the **Lab 4: Part A** dropbox on onQ for marking:

- `leastSquaresPoly.m`
- `plotLeastSquaresPoly.m`
- `leastSquaresPoly2.m`
- `plotAirfoil.m`
- Any other functions you wrote that the above files use