Queen's University | FACULTY OF ENGINEERING AND APPLIED SCIENCE

Lab 7: Solving Non-Linear Equations

The objective of this laboratory is to give you some experience solving non-linear equations using the bisection method and Newton’s method.

Before You Start

Remember that MATLAB can only use files, functions, and data within the current directory. **Make sure** you make a new folder for each lab and that it is sest as your current folder before you start.

Part A – Due at end of the lab period

In this week's Part A, you will fit a curve to a set of experimental samples using a least mean square technique. The basic idea is to select the parameters for the curve so that the sum of the squares of the differences between the experimental data points and the values given by the curve fit is minimized. The best way to see how this works is by doing an example. Assume you are given the experimental values $x = [x_1, x_2, x_3]$ and $y = [y_1, y_2, y_3]$ and you want to find the equation of the straight line $Y = Ax + B$ that best fits the curve. The sum of the squares of the differences (*Z*) would be given by the following expression:

$$Z = (Y_1 - y_1)^2 + (Y_2 - y_2)^2 + (Y_3 - y_3)^2$$

Where:

$$Y_1 = Ax_1 + B,\ Y_2 = Ax_2 + B,\ Y_3 = Ax_3 + B$$

Since *Z* is a function of two independent variables (*A* and *B*) you can think of the value of *Z* as being the height of a surface above the *A*-*B* plane. In order to find the best fit to the curve, you need to find the values of *A* and *B* that will minimize the height of the surface. You know that the slope of the surface at the location of the minimum height will be zero. Thus if you differentiate the expression for *Z* with respect to *A* and with respect to *B* and set both derivatives to zero the solution to the resulting system of equations will give the values of *A* and *B* at the minimum. Here it is easy to show that:

$$\frac{\partial Z}{\partial A} = 2\big((Ax_1 + B) - y_1\big)(x_1) + 2\big((Ax_2 + B) - y_2\big)(x_2) + 2\big((Ax_3 + B) - y_3\big)(x_3)$$

$$\frac{\partial Z}{\partial B} = 2\big((Ax_1 + B) - y_1\big) + 2\big((Ax_2 + B) - y_2\big) + 2\big((Ax_3 + B) - y_3\big)$$

Therefore, the system of equations that you need to solve for *A* and *B* is:

$$F(A, B) = \frac{\partial Z(A, B)}{\partial A} = 0$$

$$G(A, B) = \frac{\partial Z(A, B)}{\partial B} = 0$$

In this example, you need to solve a system of two linear equations. You will find that the final step in the solution will involve solving a system of non-linear equations.

Task A1

The general expression for a damped sinusoidal pulse is $V(t) = e^{-at}\cos(2\pi\, f_0 t)$ where *a* is the attenuation coefficient and f_0 is the fundamental frequency of the pulse. Your task is to determine the attenuation coefficient a and fundamental frequency f_0 that will best fit a set of measured V values, V_{fit}, measured at times t_{fit}, each with *N* values.

Using a least mean squares approach, it is easy to show that the function, Z, you need to minimize is:

$$Z(a, f_0) = \sum_{n=1}^{N} \left(V(t_{fit}(n)) - V_{fit}(n)\right)^2 = \sum_{n=1}^{N} \left(e^{-at_{fit}(n)} \cos\left(2\pi\, f_0 t_{fit}(n)\right) - V_{fit}(n)\right)^2$$

where $t_{fit}(n)$ is the n^{th} value in t_{fit}, and $V_{fit}(n)$ is the n^{th} value in t_{fit}.

As described in the previous example, you can minimize this function by taking the partial derivatives of Z with respect to a and f_0 and setting both expressions to zero. This gives you a system of two equations in two unknowns that you can solve to find a and f_0. Start by writing functions that will evaluate the partial derivatives:

Use analytical expressions for the two partial derivatives (yes, you have to do it by hand) and you can use MATLAB's built in `sum` function to perform the summation. Write a function `F` with the following inputs/output in the following order:

INPUTS	
`a`	Value of attenuation coefficient a
`f_0`	Value of fundamental frequency f_0
`V_fit`	Measured values of V
`t_fit`	Measured values of t

OUTPUTS	
`value`	Value of the partial derivative: $\frac{\partial Z}{\partial a}$ at the inputted values of a and f_0

Write a function `G` with the following inputs/output in the following order:

INPUTS	
`a`	Value of attenuation coefficient a
`f_0`	Value of fundamental frequency f_0
`V_fit`	Measured values of V
`t_fit`	Measured values of t

OUTPUTS	
`value`	Value of the partial derivative: $\frac{\partial Z}{\partial f_0}$ at the inputted values of a and f_0

The testing functions **test_F.m** and **test_G.m** are available on onQ to test your functions.

Task A2

Now that we have functions that can evaluate F and G for a given a and f_0, you need to find the values of a and f_0 that find the roots of F and $G.$ Since both F and G are non-linear (e.g. a and f_0 are in exponentials and trigonometric functions), you cannot use your previous methods to solve them. Rather, you will use the version of Newton's method from lecture to solve the system of non-linear equations. To find the minimum value of $Z(a, f_0)$ (your ultimate goal!), the partial derivatives of Z are set to zero (e.g. the roots of F and G must be found):

$$F(a, f_0) = \frac{\partial Z}{\partial a} = 0;\ G(a, f_0) = \frac{\partial Z}{\partial f_0} = 0$$

This gives you a system of two non-linear equations.

Then as shown in lecture, a first-order Taylor expansion can be used to show that for a starting guess at a^1 and f_0^1, and shifts h_a and h_{f_0}:

$$F\left(a^1 + h_a, f_0^1 + h_{f_0}\right) \cong F(a^1, f_0^1) + \frac{\partial F}{\partial a}(a^1, f_0^1)h_a + \frac{\partial F}{\partial f_0}(a^1, f_0^1)h_{f_0}$$
$$G\left(a^1 + h_a, f_0^1 + h_{f_0}\right) \cong G(a^1, f_0^1) + \frac{\partial G}{\partial a}(a^1, f_0^1)h_a + \frac{\partial G}{\partial f_0}(a^1, f_0^1)h_{f_0}$$

Since we are aiming to find the values of a and f_0 for which F and G are zero, we set $F\left(a^1 + h_a, f_0^1 + h_{f_0}\right) = 0$ and $G\left(a^1 + h_a, f_0^1 + h_{f_0}\right) = 0$. We can then set up a system of linear equations to solve for the shifts h_a and h_{f_0}:

$$0 \cong F(a^1, f_0^1) + \frac{\partial F}{\partial a}(a^1, f_0^1)h_a + \frac{\partial F}{\partial f_0}(a^1, f_0^1)h_{f_0}$$
$$0 \cong G(a^1, f_0^1) + \frac{\partial G}{\partial a}(a^1, f_0^1)h_a + \frac{\partial G}{\partial f_0}(a^1, f_0^1)h_{f_0}$$

Rearranging:

$$-F(a^1, f_0^1) \cong \frac{\partial F}{\partial a}(a^1, f_0^1)h_a + \frac{\partial F}{\partial f_0}(a^1, f_0^1)h_{f_0}$$
$$-G(a^1, f_0^1) \cong \frac{\partial G}{\partial a}(a^1, f_0^1)h_a + \frac{\partial G}{\partial f_0}(a^1, f_0^1)h_{f_0}$$

Converting to matrix form:

$$\rightarrow \quad \begin{bmatrix} \frac{\partial F}{\partial a}(a^1, f_0^1) & \frac{\partial F}{\partial f_0}(a^1, f_0^1) \\ \frac{\partial G}{\partial a}(a^1, f_0^1) & \frac{\partial G}{\partial f_0}(a^1, f_0^1) \end{bmatrix} \times \begin{bmatrix} h_a \\ h_{f_0} \end{bmatrix} = \begin{bmatrix} -F(a^1, f_0^1) \\ -G(a^1, f_0^1) \end{bmatrix}$$

Solving this system, gives values for h_a and h_{f_0}, which are used to shift a and f_0, respectively, giving $a^2 = a^1 + h_a$ and $f_0^2 = f_0^1 + h_{f_0}$ **(Note that a^2 and f_0^2 are not a and f_0 squared, the 2s are just indices to denote the updated values)**. Then an updated system of linear equations is made and solved giving new h_a and h_{f_0} values, and this repeats until the roots of F and G are found to within an acceptable tolerance.

The first step to solve this will be to write a function `findJacobian` which will return the 2x2 matrix in the equation above, which is called the Jacobian matrix. The input and output of your function should be as follows:

INPUTS	
`a`	Value of attenuation coefficient a (e.g. $a^1, a^2, \ldots$)
`f_0`	Value of fundamental frequency f_0 (e.g. $f_0^1, f_0^2, \ldots$)
`V_fit`	Measured values of V
`t_fit`	Measured values of t

OUTPUTS	
`jacobian`	The 2x2 Jacobian matrix: $\begin{bmatrix} \frac{\partial F}{\partial a}(a^1, f_0^1) & \frac{\partial F}{\partial f_0}(a^1, f_0^1) \\ \frac{\partial G}{\partial a}(a^1, f_0^1) & \frac{\partial G}{\partial f_0}(a^1, f_0^1) \end{bmatrix}$

To evaluate the partial derivatives, use your `pDer` function from Lab 2 if it worked correctly (or download **pDer.m** from onQ). **The step-size ("h value") for `pDer` should always be 1/10000 of the current variable value (e.g. $h_a = a \times \frac{1}{10000}$).** You will also need to use inline functions for `F` and `G` that only need two variables as inputs to pass into `pDer`. Please refer to Lab 3's **B.m** file for a similar example of how to do this.

Once complete, ensure your code works with the provided **test_findJacobian.m** file.

Task A3

Now that you can find the Jacobian matrix, you can now iteratively update a and f_0 to find the values that minimize Z (by finding the roots of F and G with Newton's Method). Write a function `leastMeanSquareFit` with the following inputs/output in the following order:

INPUTS	
`V_fit`	Measured values of V
`t_fit`	Measured values of t
`a_init`	Initial value/guess for attenuation coefficient a
`f0_init`	Initial value/guess for fundamental frequency f_0

OUTPUTS	
`a_opt`	Optimal value for a
`f0_opt`	Optimal value for f_0
`converged`	Logical (true or false), true if the fit converged to within a tolerance of 1×10^{-10}

Use MATLAB to solve the system of the linear equations. Also be sure to set a maximum number of iterations (10000) in case your solution doesn't converge, and also a threshold of how small F and G need to be to count as being "equal to 0" or at the root (below 1e-10 should do). A testing function **test_leastMeanSquareFit.m** is given on onQ.

Task A4

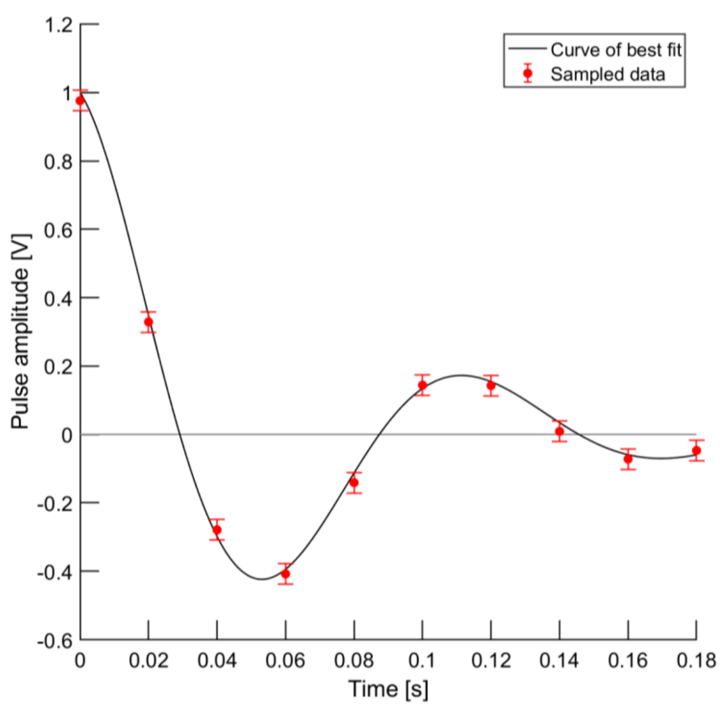
Now that you can find the optimal values for a and f_0, write a function `plotLeastMeanSquareFit` that takes in a set of measured V and t values, the error in V, and fitted values for a and f_0, and plots the original points with error bars, along with the fitted function $V(t) = e^{-at}\cos(2\pi f_0 t)$:

INPUTS	
`V_fit`	Measured values of V
`t_fit`	Measured values of t
`V_fit_error`	Error in *V* (**single value; error is the same for all points**)
`a`	Optimal value for *a*
`f_0`	Optimal value for f_0

OUTPUTS	

`[]`	No outputs; just figure should be produced. Make sure to include the commands:
	`close all;`
	`figure(1);`
	at the start of your function and please output only `figure(1)`.

A testing function **test_plotLeastMeanSquareFit.m** is given on onQ and will test your function by calling your `leastMeanSquareFit` function to find a and f_0 (if it wasn't clear, you should **not** be calling your `leastMeanSquareFit` function in `plotLeastMeanSquareFit`). If everything goes correctly, your function should produce a result similar to this:



Task A5

Your final task is to evaluate the error on your a and f_0 values by writing a function `leastMeanSquareFitWithError` that finds values for a and f_0, and the error on them. The inputs and outputs for this function are:

INPUTS	
`V_fit`	Measured values of V
`t_fit`	Measured values of t
`V_fit_error`	Error in *V* (**single value; error is the same for all points**)
`a_init`	Initial value/guess for attenuation coefficient a
`f0_init`	Initial value/guess for fundamental frequency f_0

OUTPUTS	
`a_opt`	Optimal value a
`f0_opt`	Optimal value f_0
`d_a`	Error in a
`d_f0`	Error in f_0

To find the optimal values for a and f_0, simply call your `leastMeanSquareFit` function.

To quantify the error, perform another N fits (where N is the number of points, e.g. the length of V_{fit} and t_{fit}). For each of these fits, add the error in V to a different point from V_{fit} but only that point, and perform a fit by calling your `leastMeanSquareFit` function. Be sure to always use the same initial guesses for a and f_0! For example, in the 1st fit, only the first value of V_{fit} would have the error added to it, in the 2nd fit, **only** the second value of V_{fit} would be altered, and so on.

For each of these fits, record the difference (da and df_0) between the new values a and f_0 that came from changing one of the V_{fit} points, and the true a and f_0 you found first. By the end of this process, you should have two vectors of length *N* filled with these da and df_0 values. To get the final errors (δa and δf_0), add the differences in quadrature:

$$\delta a = \left(\sum_{i=i}^{N} {da_i}^2 \right)^{\frac{1}{2}} \qquad \delta f_0 = \left(\sum_{i=i}^{N} {df_0}_i^{\,2} \right)^{\frac{1}{2}}$$

When complete, test your function with the given **test_leastMeanSquareFitWithError.m** file.

Part A: Submission List

When complete, please submit the following files to onQ for marking:

- F.m
- G.m
- findJacobian.m
- leastMeanSquareFit.m
- plotLeastMeanSquareFit.m
- leastMeanSquareFitWithError.m
- Any other functions you wrote that the above files use

Part B: Due Sunday @ 9PM

Task B1

In this final task we return to our image processing challenge. We want to fit a circle; defined by its center x-coordinate, its center y-coordinate, and its radius, to the edge coordinates we extracted using a Sobel edge detection convolution. We will use a least squares approach similar to that used above except with the cost function (the one we want to minimize), Z, given by:

$$Z(x_c, y_c, r) = \sum_{n=1}^{N} (d_n)^2 = \sum_{n=1}^{N} \left(\sqrt{(x_c - x_n)^2 + (y_c - y_n)^2} - r\right)^2$$

Where d_n is the distance from the edge point to the center of the circle and radius of the circle. We will start, just as above, by writing out the functions that will evaluate the partial derivatives:

$$Fcirc(x_c, y_c, r) = \frac{\partial Z}{\partial x_c}$$

$$Gcirc(x_c, y_c, r) = \frac{\partial Z}{\partial y_c}$$

$$Hcirc(x_c, y_c, r) = \frac{\partial Z}{\partial r}$$

Once again, **use analytical expressions for the three partial derivatives** (yes, you have to do it by hand) and you can use MATLAB's built in `sum` function to perform the summation. I have done the first partial derivative for you:

$$Fcirc(x_c, y_c, r) = \frac{\partial Z}{\partial x_c} = \sum_{n=1}^{N} \frac{2(x_c - x_n)\left(\sqrt{(x_c - x_n)^2 + (y_c - y_n)^2} - r\right)}{\sqrt{(x_c - x_n)^2 + (y_c - y_n)^2}}$$

Write three functions: `Fcirc`, `Gcirc`, and `Hcirc` all with the following inputs/outputs in the following order:

INPUTS	
`x_c`	Value of the x-coordinate of the center of the circle
`y_c`	Value of the y-coordinate of the center of the circle
`r`	Value of the circle radius
`x_edge`	X-coordinates for the edge points
`y_edge`	Y-coordinates for the edge points

OUTPUTS	
`value`	The value of evaluating the derivative of *Z* with respect to x_c (for `Fcirc`), y_c (for `Gcirc`), and r (for `Hcirc`)

The testing functions **test_Fcirc.m**, **test_Gcirc.m,** and **test_Hcirc.m** are available on onQ to test your functions.

Task B2

Write a function `findJacobianCirc` with the following inputs/outputs in the following order:

INPUTS	
`x_c`	Value of the x-coordinate of the center of the circle
`y_c`	Value of the y-coordinate of the center of the circle
`r`	Value of the circle radius
`x_edge`	X-coordinates for the edge points
`y_edge`	Y-coordinates for the edge points

OUTPUTS	
`jacobianCirc`	The 3x3 Jacobian matrix

To evaluate the partial derivatives, use your `pDer` function from Lab 2 if it worked correctly (or download **pDer.m** from onQ). Use a step-size ("h value") of `1e-4` for `pDer`. You will also need to use inline functions for `Fcirc`, `Gcirc`, and `Hcirc` that only need three variables as inputs to pass into `pDer`. Please refer to Lab 3's **B.m** file for a similar example of how to do this. Once complete, ensure your code works with the provided **test_findJacobianCirc.m** file.

Task B3

Now that you can find the new Jacobian matrix, you can now iteratively update x_c, y_c, and r to find the values that minimize Z (by finding the roots of `Fcirc`, `Gcirc`, and `Hcirc` with Newton's Method). Write a function `leastMeanSquareFitCirc` with the following inputs/outputs in the following order:

INPUTS	
`x_edge`	X-coordinates for the edge points
`y_edge`	Y-coordinates for the edge points
`xc_init`	Array of starting values for the x-coordinate of the center of the circle x_c
`yc_init`	Array of starting values for the y-coordinate of the center of the circle y_c
`r_init`	Array of starting values for the circle radius r

OUTPUTS	
`xc_opt`	Optimal value for x_c
`yc_opt`	Optimal value for y_c
`r`	Optimal value for r
`sumd`	Sum of the absolute value of the difference in distance d_n from above

Use MATLAB to solve the system of the linear equations. Your function should include:

- A maximum number of iterations in case your solution doesn't converge, I found that 15 is sufficient
- A cutoff that breaks out of the iterations for loop if `Fcirc`, `Gcirc`, and `Hcirc` are all optimized, I used `endCutoff = 1e-4;`
- A cutoff that breaks out of the iterations for loop if the values of `x_c`, `y_c`, and `r` have been shifted outside of their bounds, where the center coordinates are limited by zero and the image dimensions and the radius is limited by zero and half the smaller image dimension

A starter file **leastMeanSquareFitCirc.m** has been given to you on onQ as a starting place.

You will notice that the initial values for `x_c`, `y_c`, and `r` are arrays. The Gauss Newton method, and non-linear least squares optimization in general, is guaranteed to find a local minimum, not a global minimum. As a result, it is quite sensitive to the initial values. Your function will need to minimize $Fcirc$, $Gcirc$, and $Hcirc$ for each of the combinations of initial values. The starter function should help you understand what I mean by this. You will then need to take the mode of the optimal values for `x_c`, `y_c`, and `r`. This produces the most likely optimal value. A testing function **test_leastMeanSquareFitCirc.m** is given on onQ.

Task B4

Your final task is to plot the circle fit overlaid on the original image. Write a function `plotLeastMeanSquareFitCirc` with the following inputs/outputs in the following order:

INPUTS	
`tabletImgFilename`	Filename of the sample tablet image, string, *e.g. 'tablet2.jpg'*

OUTPUTS	

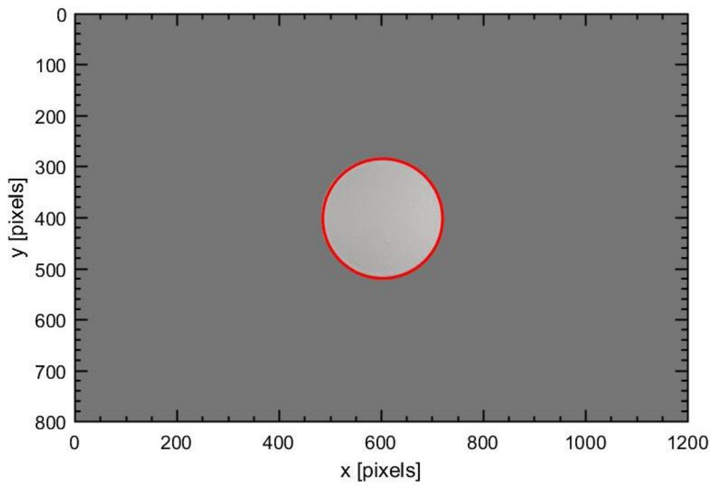
`[]`	No outputs; just figure of the fitted circle overlaid on the original image. Make sure to include the commands:
	`close all;`
	`figure(1);`
	at the start of your function and please output only `figure(1)`.

You will have to use your functions from Part B of last week's lab. Your function should:

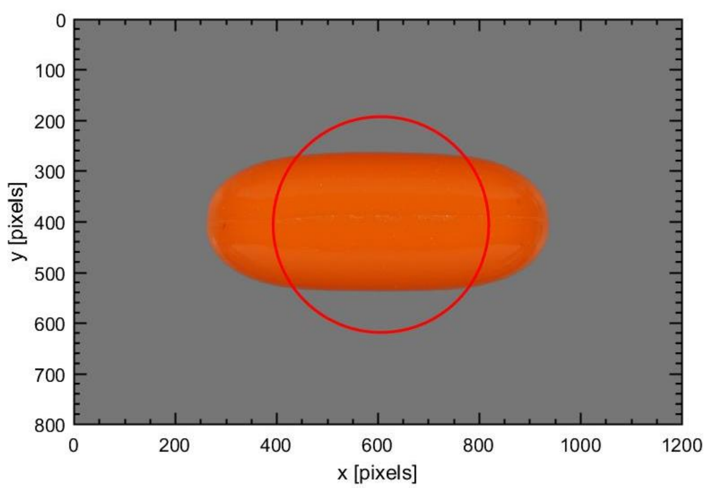
- Run your function `tabletEdgeDetection` to generate a binary, filtered image matrix. This will call your function `imfilter` so **make sure to submit both of these to the Dropbox**.
- Use the built-in Matlab function `find` to extract the edge coordinates, refer to the test file **test_leastMeanSquareFitCirc.m** as an example of how to do this.
- Run your function `leastMeanSquareFit` to find the circle center coordinates and circle radius that best fit the edge points
- Plot the image using the built-in Matlab function `imshow`
- Plot the circle fit on the same figure, I found it easiest to parameterize the circle in terms of an angle `theta`

My plots looked like this:

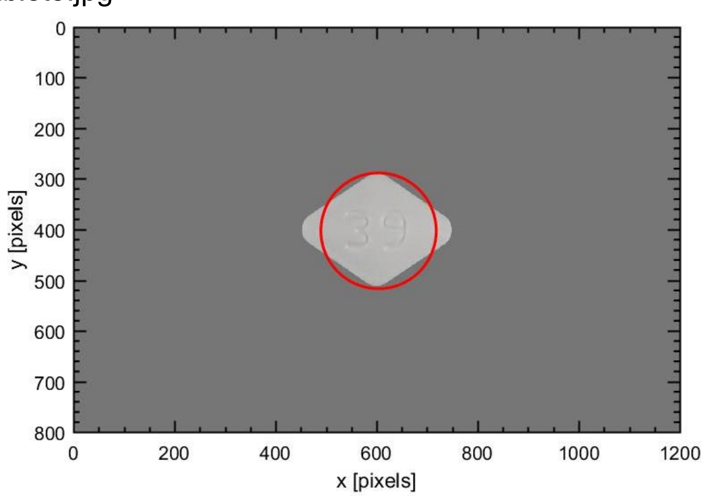
'tablet1.jpg'



'tablet2.jpg'



'tablet3.jpg'



Part B: Submission List

When complete, please submit the following files to onQ for marking:

- Fcirc.m
- Gcirc.m
- Hcirc.m
- findJacobianCirc.m
- leastMeanSquareFitCirc.m
- plotLeastMeanSquareFitCirc.m
- Any other functions you wrote that the above files use