Queen's University | FACULTY OF ENGINEERING AND APPLIED SCIENCE

Lab 2: Linear Regression

The objective of this week's laboratory is to give you some experience taking derivatives using Richardson's extrapolation and also some experience using inline functions. The error analysis code that you will write should be very useful for analysing the results of your ENPH253 laboratory experiments.

Before You Start

As mentioned in the instructions for last week's laboratory, MATLAB can only use code that is saved in the 'Current Folder'. Make sure to:

1. Create separate folders for each week's lab
2. Move the MATLAB files you download from onQ from your downloads folder into the folder for this week's lab
3. Set the current folder to be the one you have created for week's lab

This will help you avoid creating duplicate files at different locations, editing the wrong one and losing your work, or at least being at a loss as to why your outputs are persistently incorrect despite making numerous alterations to your code. The above steps will also help make sure that you upload the correct files to onQ.

Part A – Due at the end of the lab period

Taylor series expansions are the basis for numerical differentiation. As a simple example, the central difference formula is derived below. In this lab we will go one step further than the central difference formula and use Richardson's extrapolation formula.

For any function $f(x)$, of a single independent variable x, if we know the value of that function at x_0: $f(x_0)$, we can approximate its value at $x = x_0 + h$ using a Taylor series expansion:

$$f(x_0 + h) = f(x_0) + f'(x_0)\frac{h}{1!} + f''(x_0)\frac{h^2}{2!} + f'''(x_0)\frac{h^3}{3!} + O(h^4) \quad (1)$$

where by truncating the Taylor series we have necessarily sacrificed accuracy in our approximation of $f(x_0 + h)$. Similarly, we can approximate the value of $f(x)$ at $x = x_0 - h$ as:

$$f(x_0 - h) = f(x_0) - f'(x_0)\frac{h}{1!} + f''(x_0)\frac{h^2}{2!} - f'''(x_0)\frac{h^3}{3!} + O(h^4) \quad (2)$$

Remember that the goal of all this math is to come up with an approximation for $f'(x_0)$. By subtracting Eqn. 2 from Eqn. 1 we can cancel the terms proportional to h^2:

$$f(x_0 + h) - f(x_0 - h) = f'(x_0)2h + f'''(x_0)\frac{2h^3}{3!} + O(h^5)$$

which implies that:

$$f'(x_0) = \frac{f(x_0+h)-f(x_0-h)}{2h} - f'''(x_0)\frac{h^2}{3!} + O(h^4) \quad (3)$$

Neglecting the second term, we have that:

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0-h)}{2h} \quad (4)$$

to $O(h^2)$. This is the central difference formula. To improve our estimate of $f'(x_0)$ one can follow a similar procedure but including the term proportional to h^4 in Eqn. 3 and some more careful subtraction. This will be demonstrated in lecture and results in Richardson's extrapolation formula:

$$f'(x_0) = \frac{1}{3}\left[4\left(\frac{f(x_0+h)-f(x_0-h)}{2h}\right) - \left(\frac{f(x_0+2h)-f(x_0-2h)}{4h}\right)\right] + O(h^4) \quad (5)$$

which requires us to know the value of $f(x)$ at four points, rather than just two, but produces an estimate accurate to $O(h^4)$.

Task A1

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

INPUTS	
`f`	In-line function of one independent variable
`x`	1D array
`h`	Step size

OUTPUTS	
`df_dx`	Richardson's approximation for the derivative of `f` over the specified values of `x`

Make sure to vectorize your code, `x` and `f(x)` are 1D arrays. Once your code is complete, test it using the provided test file (**test_der.m**).

Task A2

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

INPUTS	
`f`	In-line function of one independent variable
`fder`	The known *analytical* derivative of `f`
`x`	1D array
`h`	Step size

OUTPUTS	
`figure(1)`	Plot the derivative as approximated using Richardson's extrapolation and the known analytical derivative, over `x`
`figure(2)`	Plot the absolute difference between the derivative as approximated using Richardson's extrapolation and the known analytical derivative, over `x`

Once your code is complete, test it using the provided test file (**test_plotDer.m**).

Task A3

To investigate how the accuracy of the derivative is influenced by the step size, write a function that `derOptStep` with the following inputs/outputs in the following order:

INPUTS	
`f`	Function of one independent variable
`fder`	The known *analytical* derivative of `f`
`x`	1D array

OUTPUTS	
`h_opt`	Optimal step size

Theoretically, the optimal step size would be the limit $h \to 0$ but due to round off errors there is a point where reducing the step size actually worsens the approximation. Your function needs to determine this point to within an order of magnitude (*i.e.* a factor of 10). **Consider carefully how to determine if your value for h_opt is within a factor of 10 of the true optimal step size.** You should be minimizing the average of the absolute error between the approximated and analytical derivative. This error value is calculated by taking the difference between the derivative as approximated using Richardson's extrapolation and the known analytical derivative (as you did in Task A2 above), and averaging this over the specified range of x. Once your code is complete, test it using the provided test file (**test_derOptStep.m**).

Task A4

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

INPUTS	
`g`	In-line function of up to three independent variables
`x1`	1D array
`h1`	Step size along `x1`
`x2`	1D array
`h2`	Step size along `x2`, optional
`x3`	1D array, optional
`h3`	Step size along `x3`, optional

OUTPUTS	
`dg_dx1`	Partial derivative of `g` with respect to `x1`
`dg_dx2`	Partial derivative of `g` with respect to `x2`, optional
`dg_dx3`	Partial derivative of `g` with respect to `x3`, optional

Use Richardson's extrapolation to calculate the derivatives. Your function should check to see how many input arguments are used and return the appropriate number of partial derivatives (either one, two or three). Make sure to vectorize your code so that it outputs the value of the partial derivatives at each of the inputted values for the independent variables. Once your code is complete, test it using the provided test file (**test_pDer.m**).

Task A5

The uncertainty dy in the value of a function of several independent variables $y = f(a, b, \dots)$ due to random measurement errors is given by the expression:

$$dy = \left[\left(\frac{\partial f}{\partial a} da\right)^2 + \left(\frac{\partial f}{\partial b} db\right)^2 + \dots\right]^{1/2}$$

where da is the uncertainty in a and db is the uncertainty in b.

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

INPUTS	
`f`	In-line function of up to three independent variables
`a`	1D array
`da`	Uncertainty in `a`, 1D array
`b`	1D array, optional
`db`	Uncertainty in `b`, 1D array
`c`	1D array, optional
`dc`	Uncertainty in `c`, 1D array

OUTPUTS	
`dy`	Propagated uncertainty in the function `f`

Your function should again work for functions one, two and three independent variables and be fully vectorized. Provided the uncertainties are not too large, a reasonable approach to selecting the step size for the derivatives would be to set $h_a = da$. Once your code is complete, test it using the provided test file (**test_errProp.m**).

Part A: Submission List

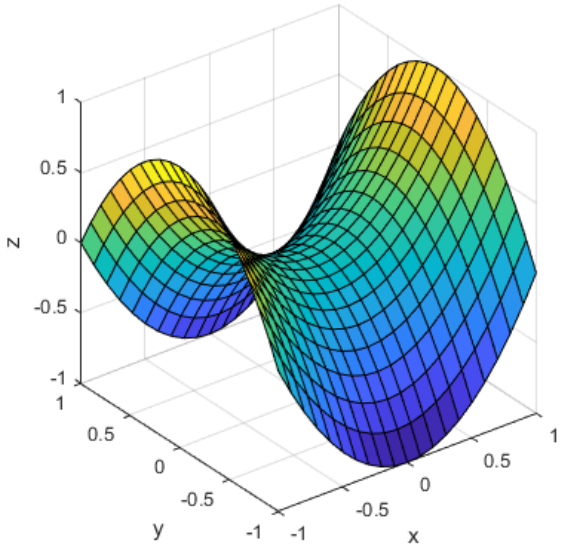
When complete, please submit the following files to the **Lab 2: Part A** dropbox on onQ for marking:

- `der.m`
- `plotDer.m`
- `derOptStep.m`
- `pDer.m`
- `errProp.m`

Any other functions you wrote that the above files use

Part B: Due Sunday @ 9PM

MATLAB provides functions for plotting surfaces and fields. As an example, let's say you wanted to plot the surface defined by an inline function f over a grid of points defined by x-values that range from -1 to 1 in steps of 0.1 and y-values that range from -1 to 1 in steps of 0.1. You would start by generating two 2-D arrays, one called XX that gives the x coordinate at each grid point and a second called YY that gives the y coordinate at each grid point. A third 2-D array ZZ would be used to store the height of the surface. A function that does just this is provided on onQ, called **surfPlot.m**. Download this script, run it and make sure that you understand each step before moving on. If you input the function: $g(x, y) = x^2 - y^2$ and plot from -1 to 1 in increments of 0.1 over both x and y the function should produce a figure that looks something like this:



Task B1:

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

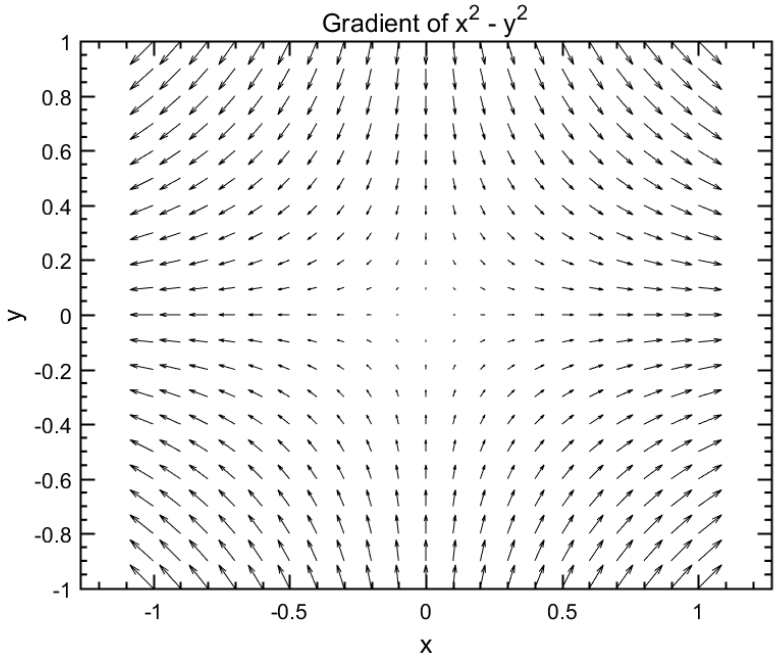
INPUTS	
`f`	In-line function of two independent variable
`x`	1D array
`y`	1D array

OUTPUTS	
`figure(1)`	Quiver plot of the gradient of `f`

The gradient of the surface is calculated according to:

$$\vec{\nabla} f = \frac{\partial f}{\partial x}\hat{x} + \frac{\partial f}{\partial y}\hat{y}$$

where $\hat{x}$ and $\hat{y}$ are the unit vectors in the x and y directions. Instead of using nested for loops to create the matrices XX any YY as was done in the provided function `surfPlot` you may also try using the `meshgrid` command. Use a step size of $h = 0.01$ to calculate the partial derivatives. Refer to the help function to figure out how to use the `quiver` command. Once your code is complete, test it using the provided test file (**test_gradient.m**). You should also have some intuition as to whether you have correctly calculated the gradient based on the topology of the surface. Your figure should look something like this:



Task B2

The electric potential V at location $\vec{r} = x\hat{x} + y\hat{y} + z\hat{z}$ due to a dipole made up of two point charges at $y = \pm 0.1$ of charge $\pm q$ is given by the expression:

$$V(r) = \frac{1}{4\pi\epsilon_o}\left(\frac{q}{r_+} - \frac{q}{r_-}\right)$$

where $r_+ = |\vec{r} - 0.1\hat{y}|$ and $r_- = |\vec{r} + 0.1\hat{y}|$.

To keep the numbers simple, let's set $q = 4\pi\epsilon_o$. The electric field can be calculated by evaluating the gradient of the electric potential:

$$\vec{E} = -\frac{\partial V}{\partial x}\hat{x} - \frac{\partial V}{\partial y}\hat{y} - \frac{\partial V}{\partial z}\hat{z}$$

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

INPUTS	
`V`	In-line function of up to three independent variables
`x`	1D array
`y`	1D array
`z`	1D array

OUTPUTS	
`Ex`	x-component of electric field, 2D array
`Ey`	y-component of electric field, 2D array
`Ez`	z-component of electric field, 2D array

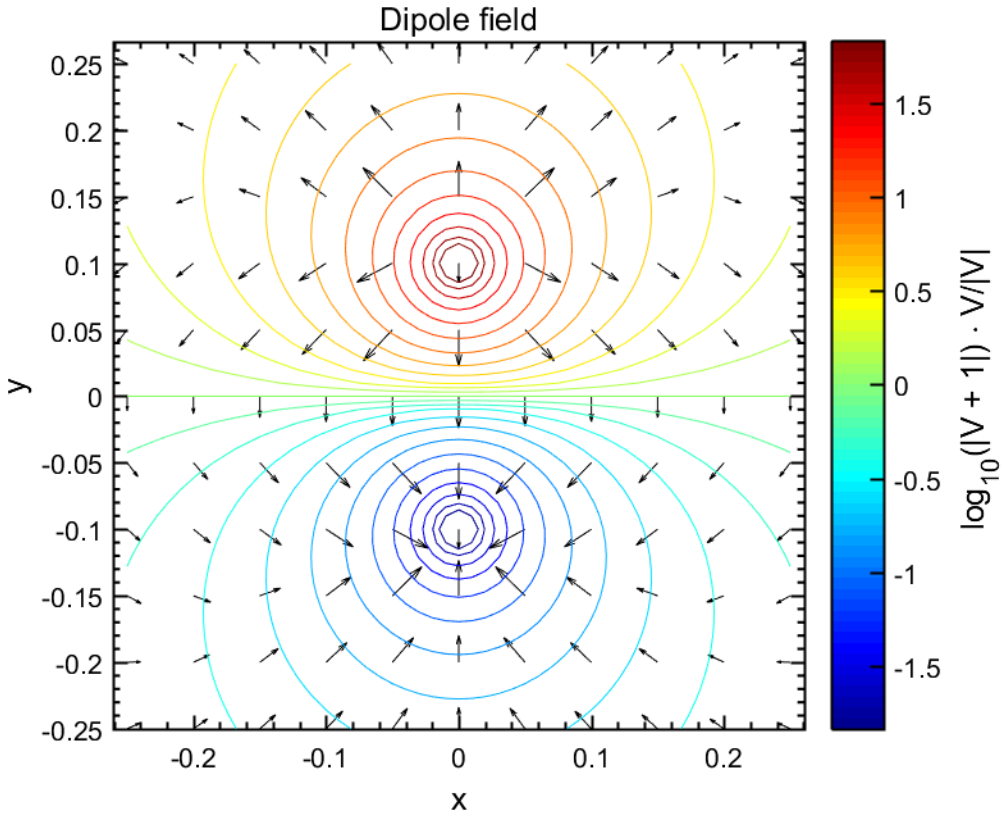
Use a step size of $h = 0.01$ to calculate the partial derivatives. Once your code is complete, test it using the provided test file (**test_eGrad.m**).

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

INPUTS	
`V`	In-line function of up three independent variables
`x`	1D array
`y`	1D array

OUTPUTS	
`figure(1)`	Quiver plot of the x and y electric field coordinates overlaid on a contour plot of the electric potential

Only consider the x-y plane (z=0), appealing to a symmetry argument. Once your code is complete, test it using the provided test file (test_plotEgrad.m), which will use your function to plot the electric field and electric potential for the electric dipole as specified above. Refer to the documentation for the `quiver` command for an example of this type of plot. Your figure should look something like this (Note: you do not have to use logarithmic scaling to receive full marks):



For those interested in replicating the above plot more closely:

To plot the electric field vectors using the `quiver` command, I added two additional steps to my plotter function:

- Increase the spacing in the meshgrid given to you in the test script by a factor of five
- Set the `scale` property in the `quiver` command to be 0.5, halving the size of the arrows

These steps make sure the plotted arrows don't overlap each other.

To plot the electric potential (V) using the `contour` command, I added the following steps to my plotter function:

- Save the sign of the potential (V/|V|) at each point in the meshgrid to a new variable I called `VVplot_sign`
- Take the logarithm of the absolute value of the potential **incremented by 1**. By incrementing the potential by 1, I am setting its logarithm to zero along the x-axis, as seen in the above figure
- Multiply the result by `VVplot_sign`

Reminder, this assignment is not group work. Every line of code that you write must be your own. If you have MATLAB problems, please don't hesitate to contact one of the course instructors.

Part B: Submission List

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

- `gradient.m`
- `eGrad.m`
- `plotEgrad.m`

Any other functions you wrote that the above files use