Ali Fele Paranj, Student number = 23760317

***Important Notes:

- You can run main.m scripts to calculate the roots. This script defines the initial values as well as the configuration values and calls the appropriate functions to calculate the root.
- Since I am comfortable in writing code with python I first implemented the solvers in python and then converted them to the matlab code. I have included the python code in the *.m files and as comment. Also you can find the python codes here: https://github.com/alifele/Computational-Physics/tree/main/UBC2021PHYS410/Assignments/First

# 1 Question1

To solve this question I have defined the following functions:

- F.m: This function implements the function that we want to find its root.
- DF.m: This function is calculating the first derivative of F function. To do this, I have used to following formula (which has error $O(h^2)$) :

$$f'(x) = \frac{-f_{j+2}+4f_{j+1}-3f_j}{2\Delta x}$$

- Fatxplusdx.m: This function calculates the value of:

$$F_{j+n} = F(x + n\Delta x)$$

This function is written to increase the readability of the code.

- randomIntervals.m and giveIntervalswithRoot.m: I have been using a kind of stochastic way to determine the number of crossing of function in [-1,1] interval. In my method, I choose n random numbers in interval [-1,1]. These random numbers are chosen in a way that the distance of two successive random numbers are not bigger than 2/n. This function returns the random intervals. In each of these random intervals (that are spanning [-1,1], I check (with the giveIntervalswithRoot.m function) if the function crosses the x axis (i.e. has any roots in that interval). Now if I make some ensemble of these intervals, then statistically speaking, it is very rare to miss any interval that has root in it.
- Bisection.m: Now the intervals calculated at the previous section (which certainly has roots inside) are ready to pass to Bisection.m function. It implements the bisection algorithms to find suitable initial values for newton's method.
- newtonFindRoot.m: this function implements the newton's method

- hybrid.m: This function calls Bisection.m function first to find suitable initial values for newton's method and then calls the newtonFindRoot.m function to calculate the roots.
- main.m: The final script that defines the initial values and configuration values, and then calls the hybrid.m function. Run this script if you want to run the codes.

## 1.1 Results and Discussion

Using the method described above, I could find 8 roots for the function in the interval [-1,1]. The roots with 12 digit precision are:

- $-0.980785280403$
- $-0.831469612302$
- $-0.555570233019$
- $-0.195090322016$
- $+0.195090322016$
- $+0.555570233019$
- $+0.831469612302$
- $+0.980785280403$

A note on convergence: The roots converged to the mentioned values quite quickly. It took about 6 iterations to converge to each of the roots mentioned above. The algorithm behaved in a very stable way and divergence is never observed throughout the calculation.

# 2 Question2

To solve this question I have used the following functions:

- F.m: This is a vector function. It gets a 1D array of 3 elements as input and returns a column vector with length 3
- FatDeltaX.m: This function is implemented to increase the readability of the code. This function calculates:

$$f(x + \delta h, y, z), f(x, y + \delta h, z), f(x, y, z + \delta h)$$

- Jacobian.m: this function claculate the jacoiab matrix. I have used F.D.A to approximate the value of $f'(x, y, z)$:

$$\frac{\partial F}{\partial x} = \frac{F(x+\delta h,y,z)-F(x,y,z)}{\delta h}$$

- newtond.m: This function implements the nD newton method. the crucial part of this funcion is calculating the value of $\Delta X$, which is done using:

$$\Delta X = J^{-1}(X)F(X)$$

- main.m: To set the initial values as well as the configuration values, I have written this script. you can simply run this script to find calculate the roots.

## 2.1 Results and Discussion

Using the method and functions described above, I have calculated the roots as following:

- x = +2.315518418880
- y = -1.881173775268
- z = -1.230890228921

A note on convergence: The roots converged to the mentioned values quite quickly. It took about 8 iterations to converge to these values.