CONTRIBUTION

The group talked about the grades that will be given should be the same for each member of the group. Each group member participated and contribute ideas to finish this user manual before the due date. Each member would like to say thank you for their fellow group members.

USER MANUAL FOR FINDING ROOTS OF AN EQUATION

BRUTE FORCE ALGORITHM (f(x)=0)

```
def f_of_x(f,roots,tol,i, epochs=100):

    x_roots=[] # list of roots
    n_roots= roots # number of roots needed to find
    incre = i #increments
    h = tol #tolerance is the starting guess

    for epoch in range(epochs): # the list of iteration that will be using
     if np.isclose(f(h),0): # applying current h or the tolerance in the equation and the approximation on f(x) = 0
        x_roots.insert(len(x_roots), h)
        end_epochs = epoch
        if len(x_roots) == n_roots:
          break # once the root is found it will stop and print the root
     h+=incre # the change of value in h wherein if the roots did not find it will going to loop

    return x_roots, end_epochs # returning the value of the roots and the iteration or the epochs
```

Figure 1. Brute Force Algorithm (f(x)=0)

The instructions below is steps on using the package. For the user to use f_of_x function, it is needed to provide the roots, the iteration is already set into the default value of 100, the estimated guess, and a number of increase must be provided. The function created that has been created, is to find the roots of the given equation.

```
1  import numpy as np
2  from numeth_yon import first_two_method
```

Figure 1.1 Import packages

1. Import the Numpy package to the IDLE (import numpy as np), and from numeth_yon package import first_two_method for it to run the equation it is needed for the first_import_method to be next to f_of_x (Refer on Figure 1.1 and Figure 1.2).

```
1  sample1 = lambda x: x**2+x-2 ⟸ Here
2  roots, epochs = first_two_method.f_of_x(sample1,2,-10,1)
3  print("The root is: {}, found at epoch {}".format(roots,epochs+1))

The root is: [-2, 1], found at epoch 12
```

Figure 1.2 Input equation

2. Type the equation in the ‘sample1’ variable (Refer to Figure 1.2).

```
1 sample1 = lambda x: x**2+x-2
2 roots, epochs = first_two_method.f_of_x(sample1,2,-10,1)
3 print("The root is: {}, found at epoch {}".format(roots,epochs+1))
```

Here

```
The root is: [-2, 1], found at epoch 12
```

Figure 1.3 Input of Parameters

3. Input the value of parameters; the equation, how many output of the root/s, the estimated guess, the number of the increment, for the iteration it had been set to the default value 100(Refer to Figure 1.3)

```
1 sample1 = lambda x: x**2+x-2
2 roots, epochs = first_two_method.f_of_x(sample1,2,-10,1)
3 print("The root is: {}, found at epoch {}".format(roots,epochs+1))
```

```
The root is: [-2, 1], found at epoch 12
```

Here

Figure 1.4 Sample Output

4. Click Play icon or type the shortcut key Shift+Enter /Ctrl+Enter to run the cell and to show the roots (Refer to Figure 1.4).

BRUTE FORCE ALGORITHM (in terms of x)

```
### brute force algorithm (in terms of x)
def in_terms_of_x(eq,tol,epochs=100):

    funcs = eq # equation to be solved
    x_roots=[] # list of roots
    n_roots = len(funcs) # How many roots needed to find according to the length of the equation
    # epochs= begin_epochs # number of iteration
    h = tol # tolerance or the guess to adjust

    for func in funcs:
      x = 0 # initial value or initial guess
      for epoch in range(epochs): # the list of iteration that will be using
        x_prime = func(x)
        if np.allclose(x, x_prime,h):
          x_roots.insert(len(x_roots),x_prime)
          break # once the root is found it will stop and print the root
        x = x_prime
    return x_roots, epochs # returning the value of the roots and the iteration or the epochs
```

Figure 2. Brute Force Algorithm (in terms of x)

In this method of using Brute Force Algorithm in terms of x, the user must have the equation to be solved, number of iteration was already set to 100 as default value and tolerance to adjust the guess number.

```
1  import numpy as np
2  from numeth_yon import first_two_method
```

Figure 2.1 Import packages

1. Import the Numpy package to the IDLE (import numpy as np), and from numeth_yon package import first_two_method for it to run the equation it is needed for the first_two_method to be next to in_terms_of_x (Refer on Figure 2.1 and Figure 2.4).


```
### brute force algorithm (in terms of x)
def in_terms_of_x(eq,tol,epochs=100):

    funcs = eq # equation to be solved
    x_roots=[] # list of roots
    n_roots = len(funcs) # How many roots needed to find according to the length of the equation
    # epochs= begin_epochs # number of iteration
    h = tol # tolerance or the guess to adjust
```


Figure 2.2

2. Place a define function that you need in finding the roots, refer on Figure 2.2

```
for func in funcs:
  x = 0 # initial value or initial guess
  for epoch in range(epochs): # the list of iteration that will be using
    x_prime = func(x)
    if np.allclose(x, x_prime,h):
      x_roots.insert(len(x_roots),x_prime)
      break # once the root is found it will stop and print the root
    x = x_prime
return x_roots, epochs # returning the value of the roots and the iteration or the epochs
```

Figure 2.3

3. In the another file but same directory it can now import the package that will be using and supply the needed value for the equation to be solved, refer to Figure 2.3

```
1  sample2 = lambda x: 2-x**2
2  sample3 = lambda x: np.sqrt(2-x)
3
4  funcs = [sample2, sample3]
5  roots, epochs = first_two_method.in_terms_of_x(funcs,1e-05)
6  print("The root is {} found after {} epochs".format(roots,epochs))
```

FIgure 2.4 Input of Parameters.

4. Place the equation in sample 2 and 3 with lambda function, refer to Figure 2.4

```
The root is [-2, 1.00000172977337] found after 100 epochs
```

Figure 2.5 Sample output

5. Run the cell by clicking the run cell button or Click Shift+Enter /Ctrl+Enter to run the cell and to show the roots. The result is in Figure 2.5 with the roots and epochs.

NEWTON RAPHSON METHOD

```
### newton-raphson method
def newt_raphson(func_eq,prime_eq, inits, epochs=100):

  f = func_eq # first equation
  f_prime = prime_eq # second equation
  # epochs= max_iter # number of iteration
  x_inits = inits # guess of the roots in range
  roots = [] # list of roots

  for x_init in x_inits:
    x = x_init
    for epoch in range(epochs):
      x_prime = x - (f(x)/f_prime(x))
      if np.allclose(x, x_prime):
        roots.append(x)
        break # once the root is found it will stop and print the root
      x = x_prime
  return roots, epochs # returning the value of the roots and the iteration or the epochs
```

Figure 3. Newton Raphson Method

The instructions below is steps on using the package. To use the newt_raphson, the user must provide an equation, the derivative of the equation, the number of repetitions was set to default value of 100, and the range of searching value for the parameters. The function of the function, is to find the roots of the given equation.

```
1 import numpy as np
2 from numeth_yon import first_two_method
```

Figure 3.1 Import packages

1. Import the Numpy package to the IDLE (import numpy as np), and from numeth_yon package import first_two_method for it to run the equation it is needed for the first_two_method to be next to newt_raphson(Refer on Figure 3.1 and Figure 3.2).

```
g = lambda x: 2*x**2 - 5*x + 3 <⟸ Here
g_prime = lambda x: 4*x-5

roots, epochs = first_two_method.newt_raphson(g,g_prime, np.arange(0,5))
x_roots = np.round(roots,3)
x_roots = np.unique(x_roots)

print("The root is {} found after {} epochs".format(x_roots,epochs))
```

```
The root is [1.  1.5] found after 100 epochs
```

Figure 3.2 Input Equation

2. Type the equation in the ‘g’ variable.

```
g = lambda x: 2*x**2 - 5*x + 3
g_prime = lambda x: 4*x-5 <⟸ Here

roots, epochs = first_two_method.newt_raphson(g,g_prime, np.arange(0,5))
x_roots = np.round(roots,3)
x_roots = np.unique(x_roots)

print("The root is {} found after {} epochs".format(x_roots,epochs))
```

```
The root is [1.  1.5] found after 100 epochs
```

Figure 3.3 Input the derivative of equation on Figure 3.2

3. Type the derivative of the given equation in the variable ‘g_prime’ (Refer to Figure 3.3).

```
The root is [1.  1.5] found after 100 epochs
```

Figure 3.4 Sample Output

4. Click Play icon or type the shortcut key Shift+Enter /Ctrl+Enter to run the cell and to show the roots.