ni_module Package User Manual

Members:
Columba, Lorenzo Miguel L.
Datay, Danica Mae L.
Embuscado, Khayle Anthony L.
Tagalog, Flor-Ann B.

About

This user manual contains instructions on how to use and implement the package on other programs. ni_module package is consists of modules that will help you find the roots of a given equation by inserting the required parameters in each module.

Step 1 Import the ni_module package

In [1]:
```
import numpy as np
# import the package
import ni_module as nm
```

Step 2 Define a given Equation

In [20]:
```
#equations
f= lambda x: x**2+5*x+6
x_p = lambda x: 2*x + 5
x= lambda x: x**3 - np.sin(x)**3 - 4*x + 1
```

Step 3 choose a desired method to use for finding

The package contains the modules listed below

- Simple Iteration Method
- Newton Rhapson Method
- Bisection Method
- Regula Falsi Method
- secant method

Step 4 Provide the required parameter in the module choosen

simple Iteration Method

This method require the user to give a equation to solve and a initial guess
f is equal to the defined equation and *-5* is the initial guess input by the user

In [3]:
```
#Simple Iteration Method single root
nm.simp_iter(f,-5)
```

```
6
2
0
The root is: [-3], found at epoch 2
```

In [4]:
```
#Simple Iteration Method n root
nm.simp_iter_n(f,-5)
```

```
6
2
0
0
2
6
12
20
30
42
The root is: [-3, -2], found at epoch 4
```

Newton Rhapson Method

This method require the user to give a equation to solve,a initial guess, and the derivative of given equation *f* is equal to the defined equation,*-5* is the initial guess input by the user, and *x_p* is equal to the declared derivative equation of *f*

In [5]:
```
# single roots newton method
nm.newt(f,-5,x_p)
```

```
the root is  -3.0000023178253943 at epoch 5
```

In [6]:
```
# n root newton method
g_range= np.arange(0,5)
nm.newt_n(f,x_p)
```

```
roots are  [-2.] found at 7
```

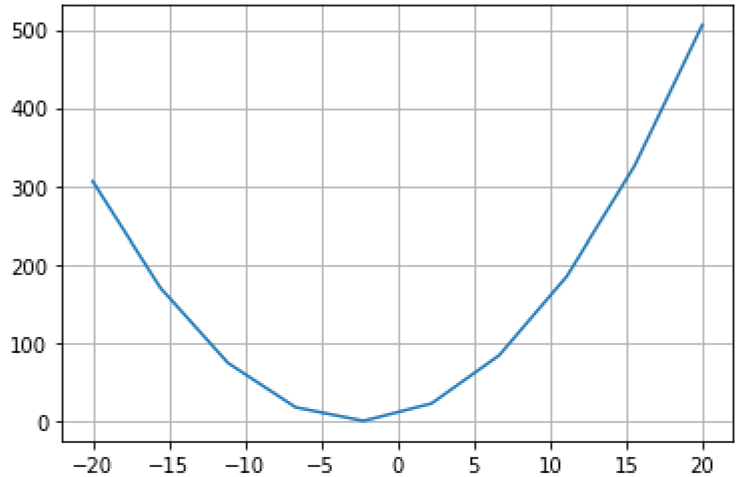
Bisection Method

This method require the user to give a equation to solve, 2 initial guesses, and a tolerance. f is equal to the defined equation,2 and 3 is the 2 initial guesses input by the user, and *1e-10* is equal to the tolerance. Tolerance is used to define the level of error

In [7]:
```
#bisection single
nm.bisec( 2 ,3,1e-10,f)
```

```
The root is 2.9999999999417923
 found at 35 bisections
```

In [8]:
```
# bisection n roots
nm.bisec_n(f,1.5,2,1e-10)
```




```
The root are [-2. -2.]
 found at bisections: 34
```

Regula Falsi (False Position Method)

This method require the user to give a equation to solve, and 2 initial guesses. f is equal to the defined equation,1 and 5 is the 2 initial guesses input by the user.

In [9]:
```
# false position method single root
nm.false_pos(x,1,5)
```

```
the root is  1.9719431995217
 at epoach: 60
```

In [23]:
```
# false position method n root
def f(x): return 3*x**2 + 7*x - 55
nm.false_pos_n(f, 0, 5, 100, 0, 2)
```

```
The roots are: [3.271175651600434, 3.2711756518495854], found at epoch 13
```

Secant Method

This method require the user to give a equation to solve, and 2 initial guesses. f is equal to the defined equation,1 and 3 is the 2 initial guesses input by the user.

In [11]:
```
# Secant method single root
nm.sec_meth(x,1,3)
```

```
the root is  1.2792256701238998
 at epoach: 99
```

In [12]:
```python
# Secant method n root
nm.secant_meth_n(f,2,0,5)
```

```
Epoch Count: 0, g_new = -0.6
Epoch Count: 1, g_new = -0.9574
Epoch Count: 2, g_new = -1.576
Epoch Count: 3, g_new = -1.8208
Epoch Count: 4, g_new = -1.9526
Epoch Count: 5, g_new = -1.9931
Epoch Count: 6, g_new = -1.9997
Epoch Count: 7, g_new = -2.0
Epoch Count: 8, g_new = -2.0

The root is: -2.0, found at 8 epochs

The roots found are:
 {-1.999999999334205}
```