Ni_module Package User Manual

Members:
Columba, Lorenzo Miguel L.
Datay, Danica Mae L.
Embuscado, Khayle Anthony L.
Tagalog, Florann 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 [2]:

```
import ni_module as nm
```

Step 2: Define a given Equation

In [ ]:

```
#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 [ ]:

```
#Simple Iteration Method single root
nm.simp_iter(f,-5)
```

In [ ]:

```
#Simple Iteration Method n root
nm.simp_iter_n(f,-5)
```

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 [3]:

```
# 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,g_range,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

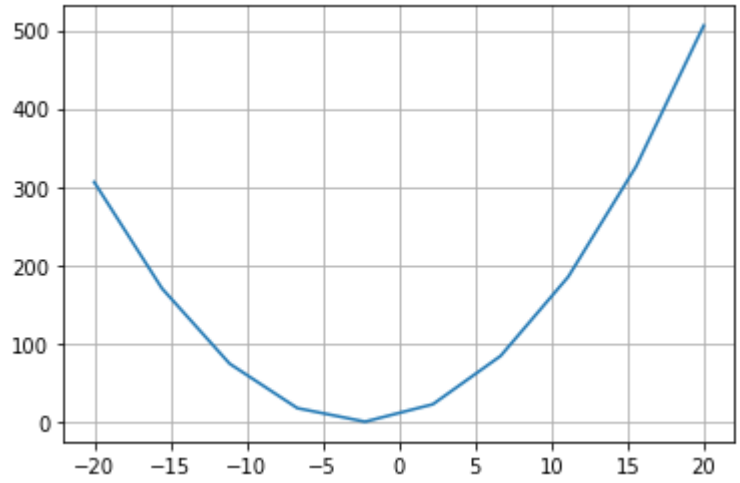
In [9]:

```
#bisection single
nm.bisec( 2 ,3,1e-10,f)
```

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

In [10]:

```
# 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 [11]:

```
# false position method single root
nm.false_pos(x,1,5)
```

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

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 [17]:

```
nm.sec_meth(x,1,3)
```

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

Activity 2.1

1. Identify **two more polynomials** preferably **orders higher than 2** and **two transcendental functions**. Write them in **LaTex**.
2. Plot their graphs you may choose your own set of pre-images.
3. Manually solve for their roots and plot them along the graph of the equation.

In [3]:

```
import numpy as np
import math
import matplotlib.pyplot as plt
```

Polynomials with orders higher than 2

$$f(x) = x^3 + 2x^2 -x -2 \\ roots: -2,-1,1$$

In [4]:

```
def f(x): return x**3+2*x**2-x-2
```

In [5]:

```
x0, x1 ,x2 = -2, -1, 1
```

In [6]:

```
X = np.arange(-10,11,1,dtype=float)
print(X)
```

```
[-10.  -9.  -8.  -7.  -6.  -5.  -4.  -3.  -2.  -1.   0.   1.   2.   3.
   4.   5.   6.   7.   8.   9.  10.]
```

In [7]:

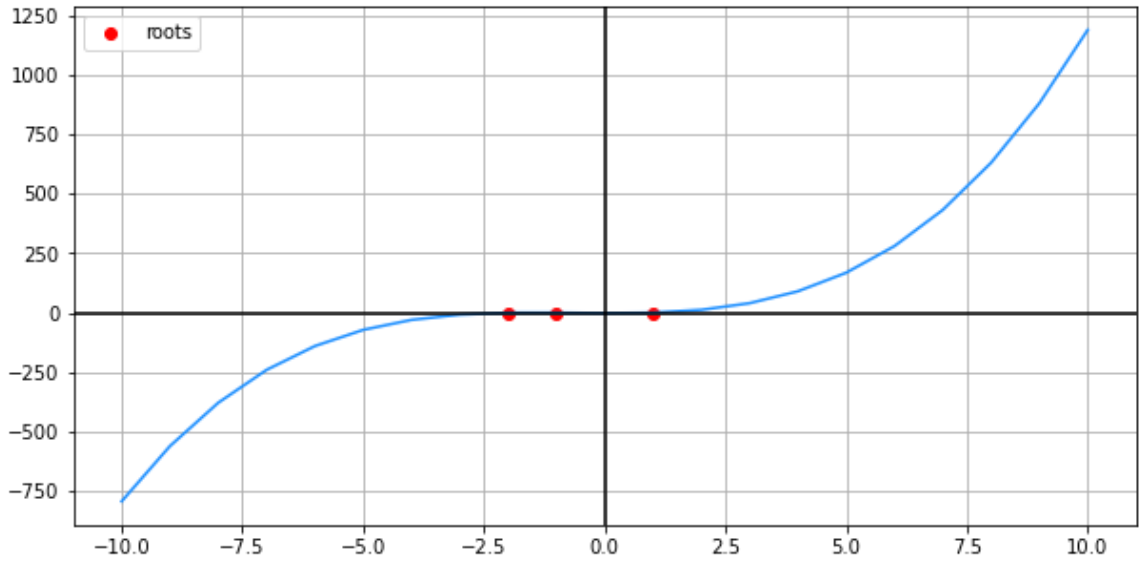
```
Y = f(X)
print(Y)
```

```
[-792. -560. -378. -240. -140.  -72.  -30.   -8.    0.    0.   -2.    0.
   12.   40.   90.  168.  280.  432.  630.  880. 1188.]
```

In [8]:

```
plt.figure(figsize=(10,5))
plt.plot(X,Y,color='dodgerblue')
plt.axhline(color='black')
plt.axvline(color='black')
plt.grid()
plt.scatter([x0,x1,x2],[0,0,0], c='red', label='roots')

plt.legend()
plt.show()
```



$$y = -x^3+3x+2 \\ roots = 2,-1,-1 $$

In [16]:

```
def f(x): return -x**3+3*x+2
```

In [18]:

```
x0, x1 ,x2 = 2, -1, -1
```

In [19]:

```
X = np.arange(-10,11,1,dtype=float)
print(X)
```

```
[-10.  -9.  -8.  -7.  -6.  -5.  -4.  -3.  -2.  -1.   0.   1.   2.   3.
   4.   5.   6.   7.   8.   9.  10.]
```

In [20]:

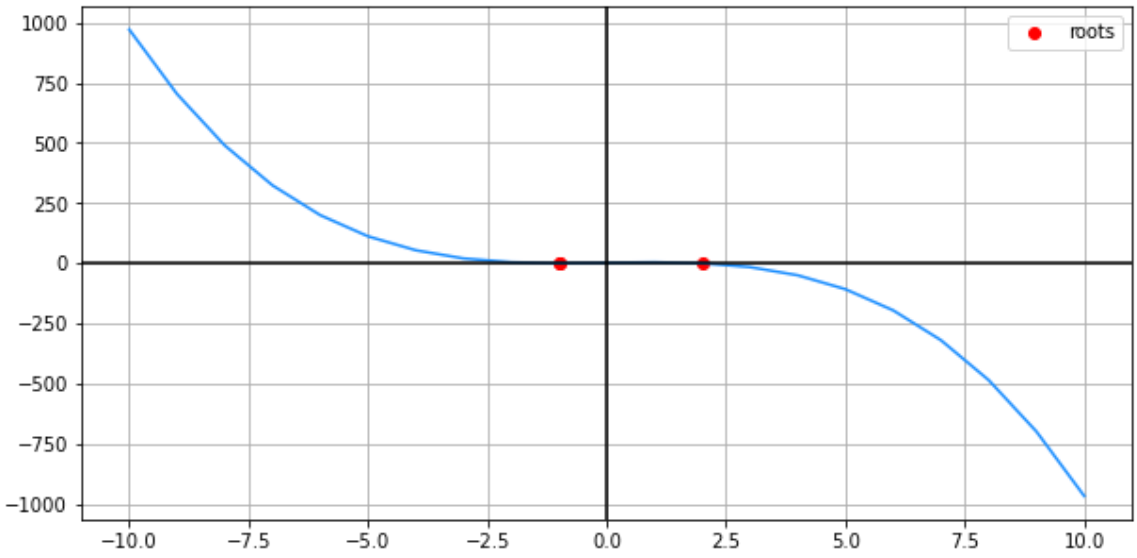
```
Y = f(X)
print(Y)
```

```
[ 972.  704.  490.  324.  200.  112.   54.   20.    4.    0.    2.    4.
    0.  -16.  -50. -108. -196. -320. -486. -700. -968.]
```

In [21]:

```
plt.figure(figsize=(10,5))
plt.plot(X,Y,color='dodgerblue')
plt.axhline(color='black')
plt.axvline(color='black')
plt.grid()
plt.scatter([x0,x1,x2],[0,0,0], c='red', label='roots')

plt.legend()
plt.show()
```



Transcendental functions

$$f(x) = \arcsin x$$

In [22]:

```
def f(x):
    return np.arcsin(x)
```

In [23]:

```
X = np.arange(-1,1,0.15,dtype=float)
print(X)
```

```
[-1.   -0.85 -0.7  -0.55 -0.4  -0.25 -0.1   0.05  0.2   0.35  0.5   0.65
  0.8   0.95]
```

In [24]:

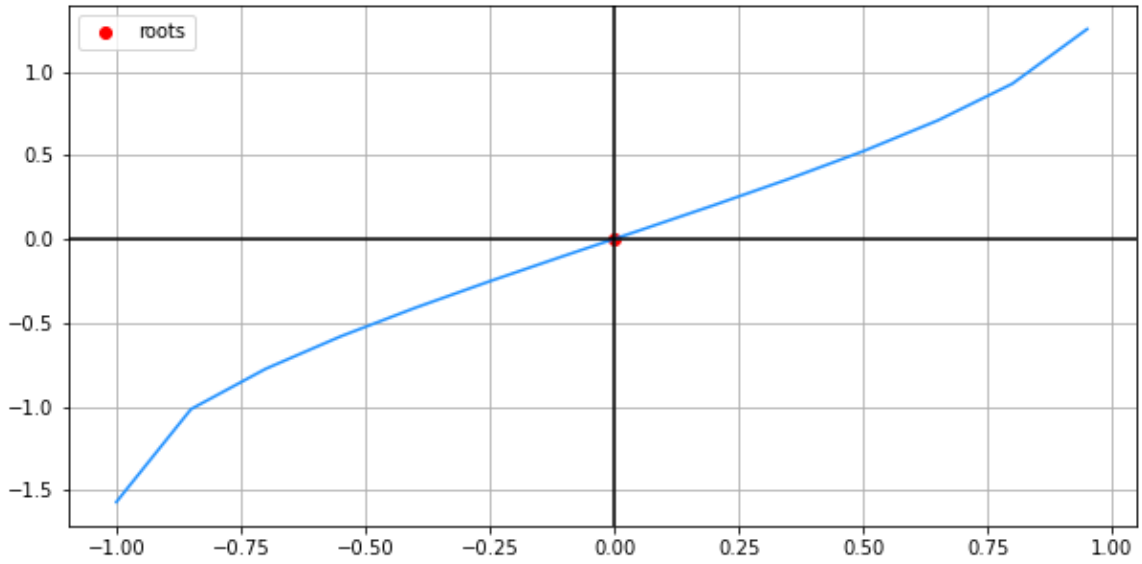
```
Q = f(X)
print(Q)
```

```
[-1.57079633 -1.01598529 -0.7753975  -0.58236424 -0.41151685 -0.25268026
 -0.10016742  0.05002086  0.20135792  0.3575711   0.52359878  0.70758444
  0.92729522  1.2532359 ]
```

In [25]:

```
plt.figure(figsize=(10,5))
plt.plot(X,Q,color='dodgerblue')
plt.axhline(color='black')
plt.axvline(color='black')
plt.grid()
plt.scatter(0,0, c='red', label='roots')

plt.legend()
plt.show()
```



$$f(x) = \arctan x$$

In [26]:

```
def f(x):
    return np.arctan(x)
```

In [27]:

```
x0 = 0
```

In [28]:

```
X = np.arange(-1,1,0.15,dtype=float)
print(X)
```

```
[-1.   -0.85 -0.7  -0.55 -0.4  -0.25 -0.1   0.05  0.2   0.35  0.5   0.65
  0.8   0.95]
```

In [29]:

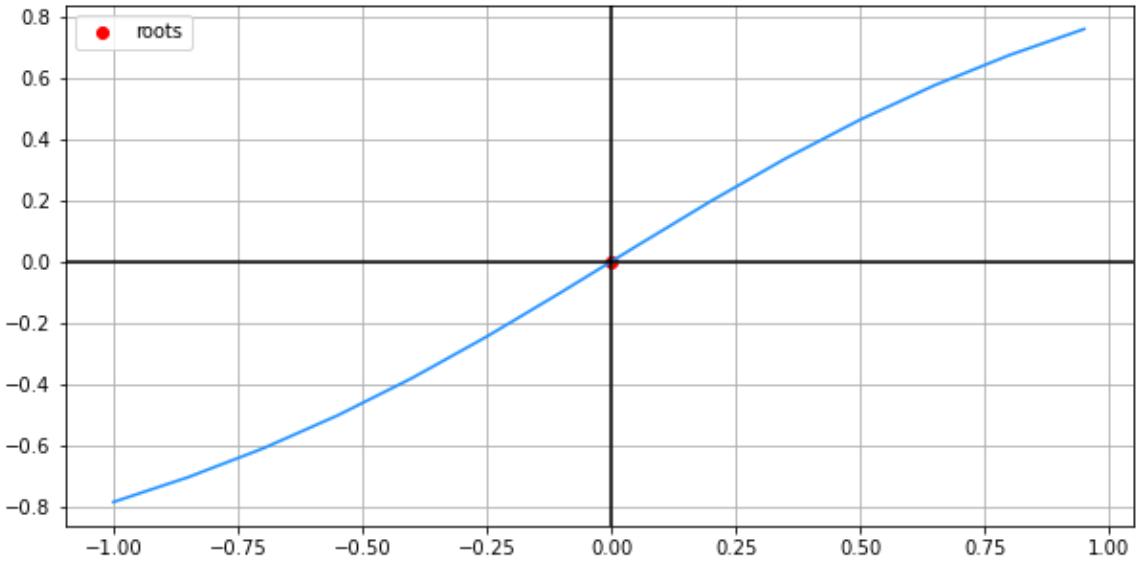
```
Q = f(X)
print(Q)
```

```
[-0.78539816 -0.70449406 -0.61072596 -0.50284321 -0.38050638 -0.24497866
 -0.09966865  0.0499584   0.19739556  0.33667482  0.46364761  0.57637522
  0.67474094  0.75976275]
```

In [30]:

```
plt.figure(figsize=(10,5))
plt.plot(X,Q,color='dodgerblue')
plt.axhline(color='black')
plt.axvline(color='black')
plt.grid()
plt.scatter(x0,0, c='red', label='roots')

plt.legend()
plt.show()
```



In [ ]:

```

```