YON User Manual

D.M. Fajardo © 2021

I.J. Timbungco © 2021

M.A. Rodriguez © 2021

N.K. Vitales © 2021

Methods

For the final name of the module the group decided to be baby_roots for all methods of the roots of the equation and the final package name throughtout this sem is numeth_yon.

- Brute force algorithm (f(x)=0)
- Brute force algorithm (in terms of x)
- Bisection Method
- Regula Falsi Method (False Position)
- Secant Method

Brute force algorithm (f(x)=0)

baby_roots.f_of_x(f,roots,tol,i, epochs=100)

Definition: Returns the roots and the epochs or iteration of the given *f* which is the function or equation using the rute force algorithm (f(x)=0).

Parameters:

- **f:** is the function or equation that is need to be solve.
- **roots:** is the number of roots to be solve from the *f* or equation.
- **tol:** is for the tolerance.
- **i:** id for the incrementation to find the iteration.
- **epochs:** is where to stop

Return:

- **x_roots:** returns the value of the roots of the given function.
- **end_epochs:** returns the value of the roots where have been found.

Inside the Module

```
### brute force algorithm (f(x)=0)
def f_of_x(f,roots,tol,i, epochs=100):

    # f = eq # equation to be solved
    x_roots=[] # list of roots
    n_roots= roots # number of roots needed to find
    incre = i #increments

    # end_epochs= stop_epochs #ending point of the iteration
    # epochs= start_epochs #starting point of the iteration
    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
        x_roots.insert(len(x_roots), h)
        end_epochs = epoch
```

```
        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 t

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

Example:

```
import numpy as np
from numeth_yon import baby_roots as br
sample1 = lambda x: x**2+x-2
roots, epochs = br.f_of_x(sample1,2,-10,1,100)
print(f"The root is: {roots}, found at epoch {epochs+1}")
```

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

Brute force algorithm (in terms of x)

baby_roots.in_terms_of_x(eq,tol,epochs=100)

Definition: Returns the roots and the epochs or iteration of the given *eq* which is the function or equation using the rute force algorithm (in terms of x).

Parameters:

- ***eq:*** is the function or equation that is need to be solve.
- **tol:** is for the tolerance.
- **epochs:** is where to stop

Return:

- **x_roots:** returns the value of the roots of the given function.
- **epochs:** returns the value of the roots where have been found.

Inside the Module:

```
### 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 eq
    # 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 epo
```

Example:

```
import numpy as np
```

```
from numeth_yon import baby_roots as br
sample2 = lambda x: 2-x**2
sample3 = lambda x: np.sqrt(2-x)

funcs = [sample2, sample3]
roots, epochs = br.in_terms_of_x(funcs,1e-05)
print("The root is {} found after {} epochs".format(roots,epochs))
```

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

Newton Raphson Method

baby_roots.newt_raphson(f,f_prime, x_inits, epochs=100)

Definition: Returns the roots and the epochs or the iteration of the given function or equation using the newton raphson method.

Parameters:

- ***func:*** is the function or equation;
- **x_inits:** is where the expected roots is to find.
- **epochs:** is where to stop

Note: while the *f* is the derivation.

Return:

- **roots:** returns the value of the roots of the given function.
- **epochs:** returns the value of the roots where have been found.

Inside the Module:

```
### newton-raphson method
def derivative(f,x,dx = 1e-6):
    return (f(x + dx) - f(x - dx))/(2*dx)

def newt_raphson(func,x_inits,tol=1e-6,epochs=100):
    roots = [] # list of roots
    for x_init in x_inits:
        x = x_init
        for epoch in range(epochs):
            x_prime = derivative(func,x)
            x_comp = x - (func(x)/x_prime)
            if np.allclose(x, x_comp):
                roots.append(np.round(x,3))
                break # once the root is found it will stop and print the root
            x = x_comp
    return roots, epochs # returning the value of the roots and the iteration or the epoch
```

Example:

```
import numpy as np
from numeth_yon import baby_roots as br
func = lambda x: -5*x**5-2*x**4+x**2+1.33*x-0.4
roots,epoch = br.newt_raphson(func,x_inits=np.arange (-10,5))
np_roots = np.array(roots)
np_roots = np.unique(np_roots)
print("The roots are {}, found at after: {}".format(np_roots,epoch))
```

```
The roots are [-0.801  0.261  0.637], found at after: 100
```

Bisection Method

baby_roots.bisection(f, i1, i2, steps, h=1e-06, end_bisect=0)

Definition: Returns the roots and the end of the bisection of the given *f* which is the function or equation using the bisection method.

Parameters:

- ***f:*** is the function or equation that is need to be solve.
- **i1:** is the first interval or the minima of the expected root.
- **i2:** is the second interval or the maxima of the expected root.
- **steps:** is the increment of the intervals.
- **h:** is for the tolerance.
- **end_bisect:** is where to stop

Return:

- **roots:** returns the value of the roots of the given function.
- **end_bisect:** returns the value of the roots where have been found.

Inside the Module:

```
### Bisection Method
def bisection(f, i1, i2, steps, h=1e-06, end_bisect = 0):
  roots = [] # list of roots
  int1 = []
  int2 = []
  for i in steps: # steps for the interval of i1 and i2
    i1+=0.25
    i2+=0.25
    int1.append(i1)
    int2.append(i2)
  for (i1,i2) in zip(int1,int2):
    y1, y2 = f(i1), f(i2)
    if np.sign(y1) == np.sign(y2): # Check the signs of y are different
      pass
    else:
      for bisect in range(0,100):
        midp = np.mean([i1,i2]) # If the signs of y1 and y2 are opposite, calculate the x
        y_mid = f(midp)
        y1 = f(i1)
        if np.allclose(0,y1, h): # If y1 and y2 approach 0, halt.
          roots.append(i1)
          end_bisect = bisect
          break
        if np.sign(y1) != np.sign(y_mid): #root is in first-half interval
          i2 = midp
        else: #root is in second-half interval
          i1 = midp
  if roots is not None:
    return roots, end_bisect
```

Example:

```
import numpy as np
from numeth_yon import baby_roots as br
g = lambda x: 2*x**4 + 3*x**3 - 11*x**2 - 9*x + 15
roots, end_bisect = br.bisection(g, 0, 1.1, np.arange(-20,10))
np_roots = np.array(roots)
```

```
np_roots = np.array(roots)
np_roots = np.round(np_roots,3)
np_roots = np.unique(np_roots)
print("The root is {} found after {} bisection".format(np_roots,end_bisect))
```

```
The root is [1.    1.732] found after 31 bisection
```

Regula Falsi Method

last_three_method.falsi(f, a, b, steps, h=1e-06, pos=0):

Definition: Returns the roots and the position of the given *f* which is the function or equation using the regula falsi method.

Parameters:

- ***f:*** is the function or equation that is need to be solve.
- **a:** is the first interval or the minima of the expected root.
- **b:** is the second interval or the maxima of the expected root.
- **steps:** is the increment of the intervals.
- **h:** is for the tolerance.
- **pos:** is where to stop

Return:

- **roots:** returns the value of the roots of the given function.
- **pos:** returns the value of the roots where have been found.

Inside the Module:

```
### Regula Falsi Method
def falsi(f, a, b, steps, h=1e-06, pos=0):
  roots = [] # list of roots
  int1 = []
  int2 = []
  for i in steps: # steps for the interval of i1 and i2
    a+=0.25
    b+=0.25
    int1.append(a)
    int2.append(b)
  for (a,b) in zip(int1,int2):
    y1, y2 = f(a), f(b)
    if np.allclose(0,y1): root = a
    elif np.allclose(0,y2): root = b
    elif np.sign(y1) == np.sign(y2): # Check the signs of y are different
      pass
    else:
        for pos in range(0,100):
          c = b - (f(b)*(b-a))/(f(b)-f(a)) ##false root # Calculate the value of c and f(c
          if np.allclose(0,f(c), h): # If f(c) approaches 0, halt and obtain the root
            roots.append(c)
            break
          if np.sign(f(a)) != np.sign(f(c)): b,y2 = c,f(c) # If f(c) and f(int1) have oppo
          else: a,y1 = c,f(c) # set int1=c and y1=f(c)
  if roots is not None:
    return roots, pos
```

Example:

```
import numpy as np
from numeth_yon import baby_roots as br
```

```
g = lambda x: (x**2-2*x-2)*(3*x**2+10*x-8)
roots, pos = br.falsi(g, 0, 1.1, np.arange(-10,10,0.25))
np_roots = np.array(roots)
np_roots = np.round(np_roots,3)
np_roots = np.unique(np_roots)
print("The root is {} found after {} false position".format(np_roots,pos))
```

```
The root is [0.667 2.732] found after 32 false position
```

Secant Method

last_three_method.secant(f, a, b, steps, epochs = 100):

Definition: Returns the roots and the iteration or epochs of the given *f* which is the function or equation using the secant method.

Parameters:

- ***f:*** is the function or equation that is need to be solve.
- **a:** is the first interval or the minima of the expected root.
- **b:** is the second interval or the maxima of the expected root.
- **steps:** is the increment of the intervals.
- **epochs:** is where to stop

Return:

- **roots:** returns the value of the roots of the given function.
- **epochs:** returns the value of the roots where have been found.

Inside the Module:

```
### Secant Method
def secant(f, a, b, steps, epochs = 100):
  roots = [] # list of roots
  for i in steps: # steps for the interval of a and b
    int1 = a+i # interval 'a' will become 'int1'
    int2 = b+i # interval 'b' will become 'int2'
    for epoch in range(epochs):
      c = int2 - (f(int2)*(int2-int1))/(f(int2)-f(int1)) # Calculate for c
      if np.allclose(int2,c): # If $x_2-x_1 approx 0, halt and retrieve root
        roots.append(c)
        break
      else:
        int1,int2 = int2,c # Else int1 = int2 and int2 = c
  return roots, epochs
```

Example:

```
import numpy as np
from numeth_yon import baby_roots as br
g = lambda x: np.log(x**2+1)
roots, epochs = br.secant(g, 0, 1.1, np.arange(0,10))
np_roots = np.array(roots)
np_roots = np.round(np_roots,3)
np_roots = np.unique(np_roots)
print("The root is {} found after {} epochs".format(np_roots,epochs))
```

```
The root is [0.] found after 100 epochs
```