YON User Manual

D.M. Fajardo © 2021

I.J. Timbungco © 2021

M.A. Rodriguez © 2021

N.K. Vitales © 2021

Methods

For this activity the module name is last_three_method, but eventually the name of the module will change after the compilation of all method, but the package name is still numeth_yon that stands for Numerical Method and the group number which is YON.

- Bisection Method
- Regula Falsi Method (False Position)
- Secant Method

Bisection Method

last_three_method.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
```

```
### Bisection Method
def bisection(f, i1, i2, steps, h=1e-06, end_bisect = 0):
  y1, y2 = f(i1), f(i2) # Calculated values of y1 and y2 given i1 and i2
  roots = [] # list of roots
  if np.sign(y1) == np.sign(y2): # Check the signs of y are different
    print("Root cannot be found in the given interval") # If the signs of y1 and y2 are the s
  else:
    for i in steps: # steps for the interval of i1 and i2
      int1 = i1+i # interval 'i1' will become 'int1'
      int2 = i2+i # interval 'i2' will become 'int2'
      intval = int1, int2 # making it a tuple
      for bisect in range(0,100):
        midp = np.mean(intval) # If the signs of y1 and y2 are opposite, calculate the x in t
        y_mid = f(midp)
        y1 = f(int1)
        if np.allclose(0,y1, h): # If y1 and y2 approach 0, halt.
          roots.append(int1)
          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 last_three_method as lt
g = lambda x: 2*x**2 - 5*x + 3
roots, end_bisect = lt.bisection(g, 0, 1, np.arange(0,10,0.25))
print("The root is {} found after {} bisection".format(roots,end_bisect))
# Output: The root is [1.0, 1.5] found after 0 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):
  y1, y2 = f(a), f(b) # Calculate values of y1 and y2 given a and b.
  roots = [] # list of roots
  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
    print("No root here") # If the signs of y1 and y2 are the same halt
  else:
    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 pos in range(0,100):
        c = int2 - (f(int2)*(int2-int1))/(f(int2)-f(int1)) ##false root # Calculate the value
        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(int1)) != np.sign(f(c)): int2,y2 = c,f(c) # If f(c) and f(int1) have opp
        else: int1,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 last_three_method as lt
g = lambda x: 2*x**2 - 5*x + 3
roots, pos = lt.falsi(g, 0, 1.1, np.arange(0,10,0.25))
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))
# Output: The root is [1.  1.5] found after 99 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
    intval1 = a+i # interval 'a' will become 'intval1'
    intval2 = b+i # interval 'b' will become 'intval2'
    for epoch in range(epochs):
      c = intval2 - (f(intval2)*(intval2-intval1))/(f(intval2)-f(intval1)) # Calculate for c
      if np.allclose(intval2,c): # If $x_2-x_1 approx 0, halt and retrieve root
        roots.append(c)
        break
      else:
        intval1,intval2 = intval2,c # Else intval1 = intval2 and intval2 = c
  return roots, epochs
```

Example:

```
import numpy as np
from numeth_yon import last_three_method as lt
g = lambda x: 2*x**2 - 5*x + 3
roots, epochs = lt.secant(g, 0, 1.1, np.arange(0,10,0.25))
np_roots = np.array(roots)
np_roots = np.round(np_roots,3)
np_roots = np.unique(np_roots)
print("The root is {} found after {} epochs".format(np_roots,epochs))
```

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