User Manual: How To Use Package and Module

Bisection, Falsi, and Secant Method

submitted by:

Aquiro, Freddielyn E.

Canoza, Cherrylyn

Joaquin, Christopher Marc

Step 1: Define the equation that you will use.

In this program, the equation that the programmers used is:

$$f(x) = 2x^2 - 5x + 3$$

This is the equation will use in Bisection and Regula Falsi Method.

Step 2: Choose a method that you want to use for finding roots.

- Bisection Method
- Regula Falsi Method
- Secant Method

Step 3: Provide your parameter.

Bisection Method

The bisection method is used to find the roots of a polynomial equation. It separates the interval and subdivides the interval in which the root of the equation lies. The principle behind this method is the intermediate theorem for continuous functions. It works by narrowing the gap between the positive and negative intervals until it closes in on the correct answer. [1]

Parameters:

func = equation to be tested

iv1 = first interval of x which gives value to y1.

iv2 = second interval of x which gives vale to y2.

nm_roots = number of roots.

epochs = how many iterations are present in the loop.

tol= tolerance

```
def func(x): return 2*x**2 - 5*x + 3 ###Equation given.
```

```
###Bisection Method function.
import numpy as np
def bisection_method(func, iv1, iv2, nm_roots, epochs, tol):
    roots = []
    y1, y2 = func(iv1), func(iv2)
    end_bisect = 0
    if np.sign(y1) == np.sign(y2):
          print("Root cannot be found in the given interval")
    else:
          for bisect in range(epochs):
              midp = np.mean([iv1,iv2])
              y_mid = func(midp)
              y1 = func(iv1)
              if np.allclose(0, y1,tol):
                  roots.append(iv1)
                  end_bisect = bisect
                  if len(roots) == nm_roots: # getting the number of roots.
                    break
              if np.sign(y1) != np.sign(y_mid): #root for first-half interval
                  iv2 = midp
              else: #root for second-half interval
                  iv1 = midp

    return print(" the roots are" ,roots, " found at" ,end_bisect)
```

```
bisection_method(func, iv1 = -1.5, iv2 = 1, nm_roots = 2, epochs = 50, tol = 1e-6) ### Given
```

```
      the roots are [0.9999999906867743, 0.9999999953433871]  found at 29
```

After declaring its parameters it will display a number of roots which satisfy the needed requirements. It will return to its declared function.

Regula Falsi Method

It is also known as *method of false position*, it is a numerical method for solving an equation in one unknown. It is quite similar to bisection method algorithm and is one of the oldest approaches. It

was developed because the bisection method converges at a fairly slow speed. In simple terms, the method is the trial and error technique of using test ("false") values for the variable and then adjusting the test value according to the outcome. [2]

```
def f(x): return 2*x**2 - 5*x + 3 ###Equation
```

```
###Function for Regula Falsi
def rfalsi(a,b):
  m = (f(b)-f(a))/(b-a) ###Formula for regula falsi
  c = a - f(a)/m
  while abs(f(c))>0.01:
    if f(c)*f(a)>0:
      a=c
    else:
      b=c
    m = (f(b)-f(a))/(b-a)
    c = a - f(a)/m

    print(c,'', f(c))
  print('The root is',c)
```

```
###Given by the user.
r = 1.1
s = 3
```

```
rfalsi(r,s)
```

```
1.1538461538461537  -0.10650887573964463
1.1860465116279069  -0.11681990265008135
1.2206896551724138  -0.12328180737217664
1.2565130260521045  -0.12491516098328903
1.2920707358813464  -0.12146010636480353
1.325958936813624  -0.11346047983628793
1.3570276748747003  -0.10209015362183127
1.3845151902744695  -0.08881132717084661
1.408078631850632  -0.07502229230446478
1.4277377515148204  -0.06181858337291235
1.4437717227654054  -0.04990503891305398
1.4566088883462625  -0.03962553451264306
1.4667349627877762  -0.03105191181076261
1.474629242494572  -0.024083406832629883
1.480727452303874  -0.018529685506726956
1.485404947303293  -0.014169021570267226
1.4889732424235185  -0.010783578811180305
1.491684083017342  -0.008177608032132966
The root is 1.491684083017342
```

Secant Method

The secant method is very similar to the bisection method except instead of dividing each interval by choosing the midpoint the secant method divides each interval by the secant line connecting the endpoints. 3

```
###Secant Method Function
import math
def secant(f,a,b,tol =1e-10,maxit=100):
  fa = f(a)
  if abs(fa) < tol:
    return a
  fb = f(b)
  if abs(fb) < tol:
    return b

  if fa*fb > 0:
    print("f(a) and f(b) must have different signs.")
    return none

  for _ in range(maxit):
    c = (a*fb - b*fa)/(fb - fa)
    fc = f(c)

    if abs(b - a) < tol:
      break
    if abs(fc) < tol:
      break
    if fa*fc > 0:
      a, fa = c, fc

    if fb*fc > 0:
      b, fb = c, fc
  return c

func = lambda x: x**3 - x - 1
a = 1
b = 5
x = secant(func, a, b, tol = 1e-20, maxit=100)
print("Solution of 1st equation: x = {}, f(x) = {}".format(x, func(x)))

func2 = lambda x: x**2 - math.exp(-x) - 1
x2 = secant(func2, -1,5, tol = 1e-6)
print("Solution of 2nd equation: x = {}, f(x) = {}".format(x2, func2(x2)))
```

```
Solution of 1st equation: x = 1.3247176381152717, f(x) = -1.360969682506763e-06
Solution of 2nd equation: x = 1.1477572664034958, f(x) = -9.556316048486835e-07
```

Reference

[1] Admin, “Bisection Method - Definition, Procedure, and Example,” BYJUS, 03-Feb-2021. [Online]. Available: https://byjus.com/maths/bisection-method/. [Accessed: 07-Mar-2021].
[2] E. Chopra, “Regula Falsi Method for finding root of a polynomial,” OpenGenus IQ: Learn Computer Science, 24-Sep-2019. [Online]. Available: https://iq.opengenus.org/regula-falsi-method/. [Accessed: 07-Mar-2021].
[3] E. Chopra, “Regula Falsi Method for finding root of a polynomial,” OpenGenus IQ: Learn Computer Science, 24-Sep-2019. [Online]. Available: https://iq.opengenus.org/regula-falsi-method/. [Accessed: 07-Mar-2021].