Regula-Falsi Method

Dennis Mwendwa

March 2023

Regula-Falsi Method

Regula-Falsi method or the method of false position is a numerical method for estimating roots of a polynomial. It is a combination of the secant method and bisection methods.

The idea is that if you have a smooth function that doesn't change much, you can approximate the function with a line using two endpoints $[a, b]$. The endpoints are joined with a chord; The point where the chord crosses the x-axis is the new "guess" for the root. The appropriate endpoint is updated with the new guess, then the algorithm continues, honing in on the actual root.

The Regula-Falsi tends to be faster than the bisection method and requires fewer iterations, this comes with a higher computational cost. Compared to the secant method, it requires more iterations

Algorithm for the Regula–Falsi Method: Given a continuous function f(x)

1. Choose two initial guesses, a and b, such that $f(a)$ and $f(b)$ have opposite signs.

2. Compute the point c where the chord connecting $(a, f(a))$ and $(b, f(b))$ intersects the x-axis. Now, the equation of the chord joining A[a, f(a)] and B[b, f(b)] is given by:
$$c = \frac{a.f(b) - b.f(a)}{f(b) - f(a)}$$

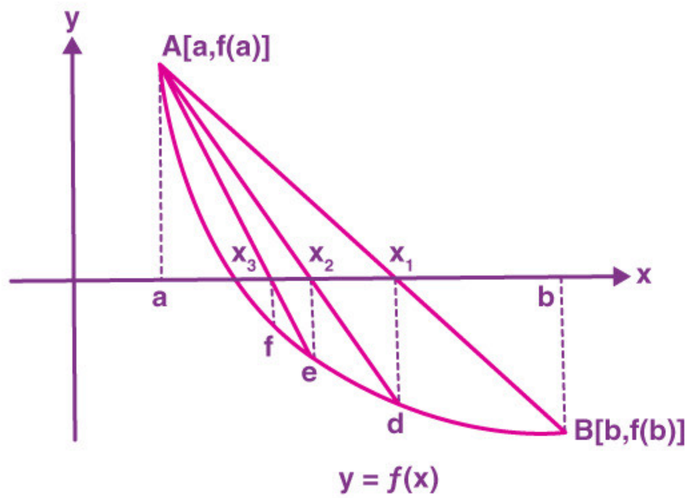
3. Compute the value of the function at c, $f(c)$.

4. Is $f(c) = 0$? If so, stop here. You've found the root. If not, continue to the next step.

5. Replace a or b with c, depending on the sign of f(c) in that if:
   - $f(a).f(b) < 0$ then $b = c$
   - $f(a).f(b) > 0$ then $a = c$

6. Continue the process repeatedly until you reach 0 or the desired accuracy.

Geometrical representation of the roots of the equation f(x) = 0 can be shown as:



Sample Problem

Show that $f(x) = x^3 + 3x - 5$ has a root in $[1, 2]$, and use the Regula-Falsi Method to determine an approximation to the root that is accurate to at least within 10^{-6}

Now, the information required to perform the Regula Falsi Method is as follows:

- $f(x) = x^3 + 3x - 5$,
- Lower Guess a = 1,
- Upper Guess b = 2,
- And tolerance $e = 10^{-6}$

Solution:

- *Iteration 1*

  $a = 1$, $b = 2$

  $f(1) = -1$, $f(2) = 9$

  $f(a).f(b) = -1 * 9 = -9 < 0$

  $c = \frac{a.f(b)-b.f(a)}{f(b)-f(a)} = 1.1$

  $f(a).f(c) = f(1).f(1.1) = 0.369 > 0$

  Since the above condition is not satisfied, we make c as our new lower guess i.e. $a = c$, $a = 1.1$ So, we have reduced the interval to : $[1, 2]- > [1.1, 2]$

Now we check the loop condition i.e.

fabs($f(c)$) $> e$ $f(c) = f(1.1) = -0.369$ fabs($f(c)$) $= 0.369 > e = 10^{-6}$ The loop condition is true so we will perform the next iteration.

- *Iteration 2*

  $a = 1.1,\ b = 2$

  $f(a) = f(1) = -5,\ f(b) = f(2) = 9$

  $c = \frac{a.f(b)-b.f(a)}{f(b)-f(a)} = 1.135446686$

  $f(a).f(c) = f(1).f(1.135446686) = -0.1297975921 < 0$

  Since the above condition is not satisfied, we make c as our new lower guess i.e. $a = c$, $a = 1.135446686$

  Again we have reduced the interval to : $[1, 2]- > [1.135446686, 2]$

  Now we check the loop condition i.e.

  fabs($f(c)$) $> e$ $f(c) = -0.1297975921$

  fabs($f(c)$) $= 0.1297975921 > e = 10^{-6}$ The loop condition is true so we will perform the next iteration.

  As you can see, it converges to a solution which depends on the tolerance and number of iteration the algorithm performs.

Python Implementation

This program implements false position (Regula Falsi) method for finding real root of nonlinear equation in python programming language.

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

x = np.array(range(-6,6))
y = x**3 + 3*x - 5
plt.plot(x,y)
plt.grid()
plt.show()

def f(x):
    return x**3+3*x-5


def regula(a,b):
    x = 0
    i=1
    condition = True
    while condition:
        n = str(x)
        x = a - ((b-a)/(f(b)-f(a)))*f(a)
        if f(x)<0:
            a=x
        else:
            b=x
        print("Iteration number: ", i, "     x = ",x, "     f(x) = ",f(x))

        m = str(x)
        if m[0:t+3]==n[0:t+3]:
            condition = False
        else:
            condition = True
            i = i+1

    print("Root found at x = ", x)
    print("Time taken: ",timeit.timeit())


a = input("First approximation root: ")
b = input("Second approximation root: ")
```

```
t = input("Enter precision of decimal places: ")
a = float(a)
b = float(b)
t = int(t)

if f(a)*f(b)>0:
    print("Given appproximation roots do not give a solution")
    print("Try again with different values")
else:
    regula(a,b)
```

For our equation, our initial guesses were 1 and 2 and the output:

```
Iteration number:  1      x =  1.1      f(x) =  -0.3689999999999998
Iteration number:  2      x =  1.1354466858789627      f(x) =  -0.1297975921309309
Iteration number:  3      x =  1.147737970248558      f(x) =  -0.04486805098132063
Iteration number:  4      x =  1.1519657086726893      f(x) =  -0.015415586390996161
Iteration number:  5      x =  1.1534157744799958      f(x) =  -0.005285298529249971
Iteration number:  6      x =  1.1539126438421201      f(x) =  -0.0018107788348853404
Iteration number:  7      x =  1.1540828403853087      f(x) =  -0.0006202314857430835
Iteration number:  8      x =  1.154141132418883      f(x) =  -0.00021242488214312516
Iteration number:  9      x =  1.1541610965554805      f(x) =  -7.275190177225e-05
Iteration number:  10     x =  1.154167933876746      f(x) =  -2.491603862431191e-05
Iteration number:  11     x =  1.1541702755129777      f(x) =  -8.533204644223247e-06
Iteration number:  12     x =  1.15417107747201      f(x) =  -2.922434727103962e-06
Iteration number:  13     x =  1.1541713521252337      f(x) =  -1.000869155554085e-06
Iteration number:  14     x =  1.1541714461878683      f(x) =  -3.427754666773808e-07
Iteration number:  15     x =  1.1541714784022312      f(x) =  -1.1739298244606289e-07

Root found at x =  1.1541714784022312
Time taken:  0.034652400005143136
```

Limitations

While Regula Falsi Method, like Bisection Method, is always convergent; meaning that it is always leading towards a specific limit and relatively simple to understand, there are also some drawbacks when this algorithm is used. As both Regula-Falsi and Bisection method are similar, there are some common limitaions both the algorithms have i.e. :

- **Rate of convergence:** The convergence of the regula falsi method can be very slow in some cases(May converge slowly for functions with big curvatures) as explained above.
- **Relies on sign changes:** If a function $f(x)$ is such that it just touches the x -axis for example say $f(x) = 2$ then it will not be able to find lower guess (a) such that $f(a) * f(b) < 0$
- **Cannot detect Multiple Roots:**Like Bisection method, Regula Falsi Method fails to identify multiple different roots, which makes it less desirable to use compared to other methods that can identify multiple roots.

References

1. Stephanie Glen. *"Regula-Falsi method: Definition, Example"* From StatisticsHowTo.com:
2. Regula Falsi Method for finding root of a polynomial-*Eklavya Chopra*
3. Byju's Learning-*False Position Method*
4. Youtube