Formosa Journal of Science and Technology (FJST)
Vol.3, No.5, 2024: 971-986

Comparison of the Number of Iterations of Roots of Non-Linear Equations in the Regula Falsi Method, Secant Method, and Bisection Method using Python

Sabrina Yose Amelia[1]*, Diah Aryani[2], Ary Budi Warsito[3], Yunianto Purnomo[4]
[1,3]Faculty of Science, Technology, and Mathematics, Matana University
[2]Faculty of Computer Science, Universitas Esa Unggul
[4]Faculty of Technology and Design, Bunda Mulia University
Corresponding Author: Sabrina Yose Amelia
sabrina.amelia@student.matanauniversity.ac.id

A R T I C L E I N F O

Key Words: NonLinear Equation, Bisection Method, Regula Falsi Method, Secant Method, Python

Received : 12, March
Revised : 15, April
Accepted: 20, May

©2024 Amelia, Aryani, Warsito, Purnomo: This is an open-access article distributed under the terms of the Creative Commons Atribusi 4.0 Internasional.

A B S T R A C T

There are several methods for solving nonlinear equations, each method has a different number of iterations. This study aims to compare the number of iterations required by the Bisection, Regula Falsi, and Secant Method in finding the roots of an equation. These three methods were tested using two different equations in the Python programming language. From each equation in this study, Bisection Method requires thirteen and eighteen iterations, Regula Falsi Method requires nine and eight iterations, while Secant Method requires five iterations. From these three methods, Secant Method requires the fewest number of iterations, Regula Falsi Method requires a greater number of iterations than the Secant Method but less than the Bisection Method, and Bisection Method requires the highest number of iterations.

DOI Prefix: https://doi.org/10.55927/fjst.v3i5.9331
ISSN-E: 2964-6804
https://journal.formosapublisher.org/index.php/fjst

Perbandingan Jumlah Iterasi Akar Persamaan Non Linear pada Metode Regula Falsi, Metode Secant, dan Metode Bisection menggunakan Python

Sabrina Yose Amelia[1]*, Diah Aryani[2], Ary Budi Warsito[3], Yunianto Purnomo[4]
[1,3]Faculty of Science, Technology, and Mathematics, Matana University
[2]Faculty of Computer Science, Universitas Esa Unggul
[4]Faculty of Technology and Design, Bunda Mulia University
Corresponding Author: Sabrina Yose Amelia
sabrina.amelia@student.matanauniversity.ac.id

A R T I C L E I N F O

Kata Kunci: Persamaan NonLinear, Metode Bisection, Metode Regula Falsi, Metode Secant, Python

Received : 12, March
Revised : 15, April
Accepted: 20, May

©2024 Amelia, Aryani, Warsito, Purnomo: This is an open-access article distributed under the terms of the Creative Commons Atribusi 4.0 Internasional.

A B S T R A K

Terdapat beberapa metode untuk menyelesaikan persamaan nonlinear yang masing-masing memiliki jumlah iterasi berbeda. Penelitian ini bertujuan untuk membandingkan banyaknya iterasi yang dibutuhkan oleh Metode Bisection, Regula Falsi, dan Secant dalam menemukan akar suatu persamaan. Ketiga metode ini diuji menggunakan dua buah persamaan yang berbeda dalam bahasa pemrograman Python. Dari masing-masing persamaan dalam penelitian tersebut, Metode Bisection memerlukan tiga belas dan delapan belas iterasi, Metode Regula Falsi memerlukan sembilan dan delapan iterasi, sedangkan Metode Secant memerlukan lima iterasi. Dapat disimpulkan bahwa dari ketiga metode tersebut, Metode Secant adalah metode yang memerlukan jumlah iterasi yang paling sedikit, Metode Regula Falsi memerlukan jumlah yang lebih banyak dari Metode Secant tapi lebih sedikit dari Metode Bisection, dan Metode Bisection memerlukan jumlah iterasi paling banyak.

PENDAHULUAN

Terdapat dua buah sistem persamaan dalam metode numerik, yaitu sistem persamaan linear dan sistem persamaan nonlinear. Persamaan linear adalah persamaan yang semua variabelnya hanya berpangkat satu dan tidak melibatkan hasil perkalian, akar, trigonometri, logaritma, dan eksponensial (Anton & Rorres, 2013). Sehingga sistem persamaan linear adalah himpunan dari beberapa persamaan linear. Sedangkan sistem persamaan nonlinear adalah sebuah sistem yang menggunakan rumus $f(x) = 0$ untuk mencari akar persamaannya (Sunandar & Indrianto, 2020).

Penyelesaian sistem persamaan nonlinear dapat menggunakan beberapa metode, di antaranya adalah Metode Bisection, Metode Regula Falsi, Metode Newton-Raphson, dan Metode Secant. Masing-masing metode membutuhkan jumlah iterasi yang berbeda untuk dapat menemukan akar dari suatu persamaan nonlinear karena menggunakan rumus-rumus yang berbeda. Pada penelitian ini akan membandingkan banyaknya iterasi yang dibutuhkan oleh Metode Bisection, Metode Regula Falsi, dan Metode Secant untuk dapat menemukan akar dari suatu persamaan nonlinear. Untuk membandingkan banyaknya iterasi dalam penelitian ini, metode-metode tersebut akan diimplementasikan dalam bahasa pemrograman Python. Tujuan dari penggunaan program yang menggunakan bahasa pemrograman Python dalam penelitian adalah agar mendapatkan penyelesaian persamaan nonlinear dengan cepat sehingga membantu dalam penghematan penggunaan waktu agar tidak memakan banyak waktu dalam menghitung penyelesaian masing-masing persamaan nonlinear.

Penelitian yang telah ada sebelumnya yaitu "Perbandingan Keefisienan Metode Newton-Raphson, Metode Secant, dan Metode Bisection dalam Mengestimasi Implied Volatilities Saham" (Rahayuni et al., 2016) yang membandingkan implied volatility, banyak iterasi, dan error relatif pada ketiga metode tersebut. Pada penelitian tersebut didapatkan kesimpulan bahwa Metode Bisection memiliki kecepatan konvergen yang paling lambat, sedangkan Metode Secant memiliki kecepatan konvergen yang lebih cepat dari Metode Bisection tapi lebih lambat dari Metode Newton-Raphson, dan Metode Newton-Raphson memiliki kecepatan konvergen yang paling cepat dari ketiga metode yang dibandingkan. Penelitian lainnya yaitu "Penyelesaian Sistem Persamaan Non-Linier dengan Metode Bisection & Metode Regula Falsi Menggunakan Bahasa Program Java" (Sunandar, 2019). Kesimpulan yang didapatkan dari penelitian ini yaitu Metode Bisection memiliki waktu eksekusi yang lebih lama dibandingkan dengan Metode Regula Falsi.

TINJAUAN PUSTAKA

Bahasa Pemrograman

Bahasa Pemrograman merupakan bahasa yang digunakan seseorang untuk berinteraksi dan memberi perintah dengan komputer sehingga komputer dapat memberikan keluaran yang diinginkan. Menurut IEEE terdapat lebih dari tiga ratus bahasa komputer yang digunakan pada tahun 2019 dan terdapat 47 bahasa pemrograman yang memiliki pengaruh signifikan menurut survey yang mereka lakukan (Peslak & Conforti, 2020). Bahasa pemrograman dapat dibagi

menjadi tiga buah kelas, yaitu *machine language, assembly language,* dan *high-level language*. Salah satu contohnya adalah bahasa pemrograman Python yang terdapat di dalam *high-level language* (Pei, 2018).

Bahasa Pemrograman Python

Bahasa pemrograman Python pertama kali dirilis pada tahun 1991. Bahasa ini merupakan bahasa yang mudah untuk digunakan yang dikembangkan oleh Guido van Rossum (Dawson, 2010). Python memiliki banyak *library* yang dapat membantu programmer dalam menulis kode-kode atau membuat program, seperti NumPy, pandas, matplotlib, dan lain sebagainya (McKinney, 2022).

Metode Bisection

Metode Bisection merupakan metode untuk mencari akar suatu persamaan nonlinear dari interval kedua titik x_1 dan x_2. Lalu hitung nilai $f(x_1)\ dan\ f(x_2)$ dengan syarat nilai $f(x_1)\ dan\ f(x_2)$ harus memiliki tanda yang berbeda (Rahayuni et al., 2016).

Caranya adalah dengan cara membagi dua hasil penjumlahan dari nilai x_1 dan x_2 yang dilakukan secara berulang-ulang sampai ditemukan hasil yang lebih kecil dari nilai toleransi yang sudah ditentukan sebelumnya (Asminah & Sahfitri, 2012) dengan rumus

$$x_t = \frac{x1 + x2}{2} \quad (1)$$

Konsep metode Bisection didasarkan pada teori kontinu Bolzano. Jika fungsi yang diberikan adalah $f(x) = 0$ dan akarnya antara $[a, b]$, akar-akar tersebut akan berada di antara $f(a)$ dan $f(b)$ jika nilai $f(x)$ adalah bilangan real kontinu dan nilai $f(a) \times f(b)$ kurang dari 0 (Burden & Faires, 2010) (Azure et al., 2019). Terdapat tiga cara untuk menghentikan iterasi pada metode Bisection, yaitu menentukan jumlah iterasi; apabila $|x_{i+1} - x_i| \leq \varepsilon$ dengan nilai ε yang diberikan memenuhi syarat ($\varepsilon > 0$); atau jika $|f(x_i)| \leq \alpha$ dengan nilai α yang diberikan memenuhi syarat ($\alpha > 0$). Hal ini disampaikan oleh Sambourou Massinanke dan Zhang ChaoZhu dalam jurnalnya yang berjudul Comparison of Genetic Algorithm and Bisection Method for Finding Roots in One-dimensional Space (Massinanke & Zhang, 2014).

Metode Regula Falsi

Metode Regula Falsi (*Regular False Method*) atau *Method of False Position* adalah metode untuk menyelesaikan persamaan nonlinear yang sudah berusia ribuan tahun dengan menggabungkan dua buah metode, yaitu Metode Bisection dan Metode Secant (Abbasbandy & Liao, 2008). Metode ini menggunakan dua buah titik (x_1 dan x_2) dengan nilai $f(x_1)\ dan\ f(x_2)$ berbeda tanda (Sulaiman et al., 2016).

Titik potong pada metode Regula Falsi yang memberikan pendekatan kepada akar yang dicari dan terus dilakukan secara berulang-ulang sampai ditemukan hasil yang lebih kecil dari nilai toleransi yang sudah ditentukan sebelumnya menggunakan rumus (Sulaiman et al., 2016)

$$x = \frac{af(b) - bf(a)}{f(b) - f(a)}$$
(2)

Metode Secant

Metode Secant merupakan sebuah metode yang dimodifikasi dari metode yang sudah ada sebelumnya, yaitu Metode *Newton-Raphson*. Pada Metode *Newton-Raphson* hanya diperlukan untuk menentukan satu buah fungsi $f(x)$ dan diharuskan untuk mencari turunan dari fungsinya. Namun, terdapat beberapa fungsi yang tidak bisa dicari fungsi turunannya dengan mudah. Sedangkan Metode Secant dilakukan dengan mengganti fungsi turunannya dengan bentuk lain yang sama. Rumus dari metode Newton-Raphson (Juhari, 2021):

$$\left(x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}\right),$$
(3)

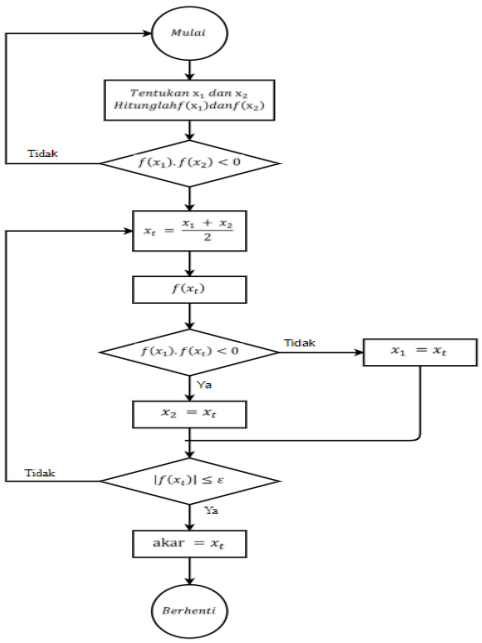
Rumus tersebut dapat disederhanakan menjadi (Rahayuni et al., 2016):

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$
(4)

METODOLOGI

Algoritma

Metode Bisection



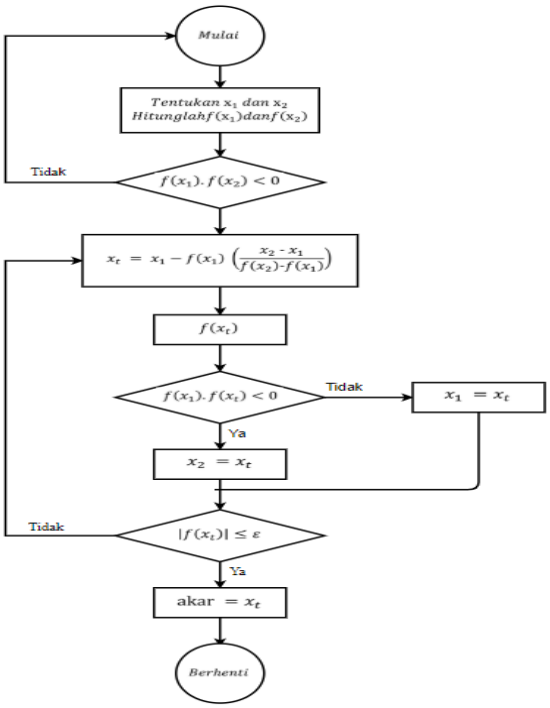
Gambar 1. Metode Bisection

Algoritma Metode *Bisection*:

1. Dimulai dengan menentukan nilai sembarang untuk x_1 dan x_2, kemudian hitung fungsi untuk $f(x_1)$ dan $f(x_2)$.

2. Cek fungsi $f(x_1)$ dikali dengan $f(x_2)$ harus lebih kecil dari nol, jadi fungsi $f(x_1)$ harus berbeda tanda dengan fungsi $f(x_2)$.
3. Tentukan nilai tengah dengan menjumlahkan nilai x_1 dan x_2 lalu hasilnya dibagi dua, dirumuskan dengan $x_t = \frac{x_1 + x_2}{2}$
4. Dari nilai x_t yang sudah ditentukan, hitunglah nilai $f(x_t)$.
5. Kalikan fungsi $f(x_1)$ dan fungsi $f(x_t)$, apabila hasilnya lebih besar dari nol, nilai x_1 diganti dengan nilai x_t. Sedangkan bila hasilnya lebih kecil dari nol, nilai x_2 diganti dengan nilai x_t.
6. Apabila nilai $|f(x_t)|$ lebih kecil atau sama dengan nilai toleransi yang sudah ditentukan sebelumnya (ε), maka nilai akarnya adalah nilai tengahnya (x_t). Sebaliknya bila nilai $|f(x_t)|$ lebih besar dari nilai ε, maka proses diulang kembali dari langkah ketiga dengan nilai x_1 dan x_2 disesuaikan menurut langkah kelima.

Metode Regula Falsi



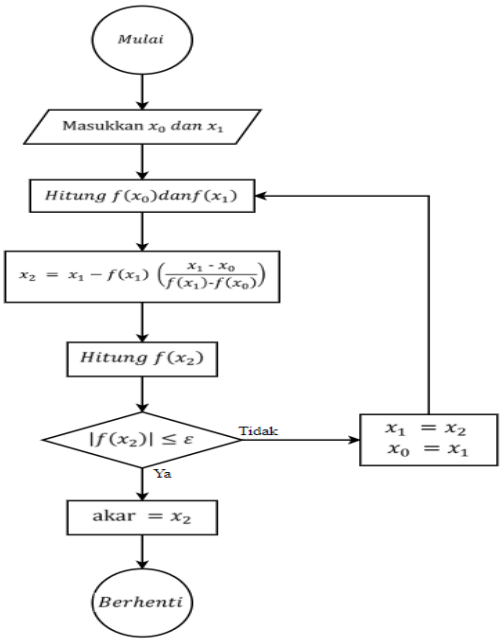
Gambar 2. Metode Regula Falsi

Algoritma Metode *Regula Falsi*:

1. Dimulai dengan menentukan nilai sembarang untuk x_1 dan x_2, kemudian hitung fungsi untuk $f(x_1)$ dan $f(x_2)$.
2. Cek fungsi $f(x_1)$ dikali dengan $f(x_2)$ harus lebih kecil dari nol, jadi fungsi $f(x_1)$ harus berbeda tanda dengan fungsi $f(x_2)$.
3. Tentukan nilai tengah dengan rumus $x_t = x_1 - f(x_1)\left(\frac{x_2 - x_1}{f(x_2) - f(x_1)}\right)$
4. Dari nilai x_t yang sudah ditentukan, hitunglah nilai $f(x_t)$.
5. Kalikan fungsi $f(x_1)$ dan fungsi $f(x_t)$, apabila hasilnya lebih besar dari nol, nilai x_1 diganti dengan nilai x_t. Sedangkan bila hasilnya lebih kecil dari nol, nilai x_2 diganti dengan nilai x_t.

6. Apabila nilai $|f(x_t)|$ lebih kecil atau sama dengan nilai toleransi yang sudah ditentukan sebelumnya (ε), maka nilai akarnya adalah nilai tengahnya (x_t). Sebaliknya bila nilai $|f(x_t)|$ lebih besar dari nilai ε, maka proses diulang kembali dari langkah ketiga dengan nilai x_1 dan x_2 disesuaikan menurut langkah kelima

Metode Secant



Gambar 3. Metode Secant

Algoritma Metode *Secant*:

1. Tentukan sembarang untuk nilai x_0 dan x_1.
2. Hitung nilai dari fungsi $f(x_0)$ dan $f(x_1)$.
3. Hitung nilai x_2 menggunakan rumus $x_2 = x_1 - f(x_1)\left(\frac{x_1 - x_0}{f(x_1)-f(x_0)}\right)$
4. Dari nilai x_2 yang sudah ditentukan, hitung nilai fungsi $f(x_2)$.
5. Apabila nilai $|f(x_t)|$ lebih besar dari nilai toleransi yang sudah ditentukan sebelumnya (ε), maka nilai x_1 diganti dengan nilai x_2 dan nilai x_0 diganti dengan nilai x_1. Sebaliknya bila nilai $|f(x_t)|$ lebih kecil atau sama dengan nilai ε maka nilai akarnya adalah x_2.

PEMBAHASAN DAN HASIL PENELITIAN

Algoritma

Equation

Pada fungsi def equations(x) diisi dengan fungsi persamaan yang akan dijalankan. Pada penelitian ini akan digunakan dua persamaan sebagai contoh, yaitu

1. $x^4 + 15x^2 - 25x + 12$

```
def equations(x) :
    return (x * x * x * x) - (15 * x * x) - (25 * x) + 12;
```

Gambar 4. Persamaan Satu

2. $x^3 + x^2 - 3x - 3$

```
def equations(x) :
    return (x * x * x) + (x * x) - (3 * x) -3;
```

Gambar 5. Persamaan Dua

Metode Bisection

Source Code dan Output

```
def bisectionMethod (a, b, equations, tolerance):
    if (equations(a) * equations(b)) > 0 :
        print(f"Tidak ada faktor untuk f(x) antara titik {a} dan {a}")
        return

    x1 = a
    x2 = b
    i = 1

    while True :
        xt = (x1 + x2) / 2

        print(f"Iterasi {i}")
        print(f"f(x1)\tf({x1}) = {equations(x1)}")
        print(f"f(x2)\tf({x2}) = {equations(x2)}")
        print(f"f(xt)\tf({xt}) = {equations(xt)}")

        if (equations(x1) * equations(xt)) < 0 :
            print("=> Lebih kecil dari nol, x2 = xt\n")
            x2 = xt
        else :
            print("=> Lebih besar dari nol, x1 = xt\n")
            x1 = xt

        i += 1

        if abs(equations(xt)) < tolerance:
            break

    print(f"\nFaktor ditemukan pada titik x = {xt}")
    print(f"\nBanyak iterasi : {i-1}")
```

Gambar 6. *Source Code* python untuk *metode Bisection*

Untuk *equations* $x^4 - 15x^2 - 25x + 12$, didapatkan *output* sebagai berikut:

```
In [4]: x1 = 0
        x2 = 1
        tolerance = 0.00001
        bisectionMethod(x1, x2, equations, tolerance)

Iterasi 1
f(x1)   f(0) = 12
f(x2)   f(1) = -27
f(xt)   f(0.5) = -4.1875
=> Lebih kecil dari nol, x2 = xt

Iterasi 2
f(x1)   f(0) = 12
f(x2)   f(0.5) = -4.1875
f(xt)   f(0.25) = 4.81640625
=> Lebih besar dari nol, x1 = xt

Iterasi 3
f(x1)   f(0.25) = 4.81640625
f(x2)   f(0.5) = -4.1875
f(xt)   f(0.375) = 0.535400390625
=> Lebih besar dari nol, x1 = xt

Iterasi 4
f(x1)   f(0.375) = 0.535400390625
f(x2)   f(0.5) = -4.1875
f(xt)   f(0.4375) = -1.7719573974609375
=> Lebih kecil dari nol, x2 = xt

Iterasi 5
f(x1)   f(0.375) = 0.535400390625
f(x2)   f(0.4375) = -1.7719573974609375
f(xt)   f(0.40625) = -0.6045980453491211
=> Lebih kecil dari nol, x2 = xt

Iterasi 6
f(x1)   f(0.375) = 0.535400390625
f(x2)   f(0.40625) = -0.6045980453491211
f(xt)   f(0.390625) = -0.031160295009613037
=> Lebih kecil dari nol, x2 = xt
```

```
Iterasi 7
f(x1)   f(0.375) = 0.535400390625
f(x2)   f(0.390625) = -0.031160295009613037
f(xt)   f(0.3828125) = 0.25298190489411354
=> Lebih besar dari nol, x1 = xt

Iterasi 8
f(x1)   f(0.3828125) = 0.25298190489411354
f(x2)   f(0.390625) = -0.031160295009613037
f(xt)   f(0.38671875) = 0.1111259947065264
=> Lebih besar dari nol, x1 = xt

Iterasi 9
f(x1)   f(0.38671875) = 0.1111259947065264
f(x2)   f(0.390625) = -0.031160295009613037
f(xt)   f(0.388671875) = 0.04003661267051939
=> Lebih besar dari nol, x1 = xt

Iterasi 10
f(x1)   f(0.388671875) = 0.04003661267051939
f(x2)   f(0.390625) = -0.031160295009613037
f(xt)   f(0.3896484375) = 0.004451595189493673
=> Lebih besar dari nol, x1 = xt

Iterasi 11
f(x1)   f(0.3896484375) = 0.004451595189493673
f(x2)   f(0.390625) = -0.031160295009613037
f(xt)   f(0.39013671875) = -0.013350991364802667
=> Lebih kecil dari nol, x2 = xt

Iterasi 12
f(x1)   f(0.3896484375) = 0.004451595189493673
f(x2)   f(0.39013671875) = -0.013350991364802667
f(xt)   f(0.389892578125) = -0.004448858383224064
=> Lebih kecil dari nol, x2 = xt

Iterasi 13
f(x1)   f(0.3896484375) = 0.004451595189493673
f(x2)   f(0.389892578125) = -0.004448858383224064
f(xt)   f(0.3897705078125) = 1.5783377520506292e-06
=> Lebih besar dari nol, x1 = xt


Faktor ditemukan pada titik x = 0.3897705078125

Banyak iterasi : 13
```

Gambar 7. Hasil Perhitungan persamaan $x^4 - 15x^2 - 25x + 12$ Metode Bisection

Pada persamaan $x^4 - 15x^2 - 25x + 12$ yang menggunakan Metode *Bisection*, iterasi berhenti pada iterasi ketiga belas dengan akar yang didapat yaitu 0,3897705078125.

Untuk *equations* $x^3 + x^2 - 3x - 3$, didapatkan *output* sebagai berikut:

```
In [7]: x1 = 1
        x2 = 2
        tolerance = 0.00001
        bisectionMethod(x1, x2, equations, tolerance)

Iterasi 1
f(x1)   f(1) = -4
f(x2)   f(2) = 3
f(xt)   f(1.5) = -1.875
=> Lebih besar dari nol, x1 = xt

Iterasi 2
f(x1)   f(1.5) = -1.875
f(x2)   f(2) = 3
f(xt)   f(1.75) = 0.171875
=> Lebih kecil dari nol, x2 = xt

Iterasi 3
f(x1)   f(1.5) = -1.875
f(x2)   f(1.75) = 0.171875
f(xt)   f(1.625) = -0.943359375
=> Lebih besar dari nol, x1 = xt

Iterasi 4
f(x1)   f(1.625) = -0.943359375
f(x2)   f(1.75) = 0.171875
f(xt)   f(1.6875) = -0.409423828125
=> Lebih besar dari nol, x1 = xt

Iterasi 5
f(x1)   f(1.6875) = -0.409423828125
f(x2)   f(1.75) = 0.171875
f(xt)   f(1.71875) = -0.124786376953125
=> Lebih besar dari nol, x1 = xt

Iterasi 6
f(x1)   f(1.71875) = -0.124786376953125
f(x2)   f(1.75) = 0.171875
f(xt)   f(1.734375) = 0.022029876708984375
=> Lebih kecil dari nol, x2 = xt

Iterasi 7
f(x1)   f(1.71875) = -0.124786376953125
f(x2)   f(1.734375) = 0.022029876708984375
f(xt)   f(1.7265625) = -0.051755428314208984
=> Lebih besar dari nol, x1 = xt
```

```
Iterasi 8
f(x1)   f(1.7265625) = -0.051755428314208984
f(x2)   f(1.734375) = 0.022029876708984375
f(xt)   f(1.73046875) = -0.014957249164581299
=> Lebih besar dari nol, x1 = xt

Iterasi 9
f(x1)   f(1.73046875) = -0.014957249164581299
f(x2)   f(1.734375) = 0.022029876708984375
f(xt)   f(1.732421875) = 0.0035126730799674988
=> Lebih kecil dari nol, x2 = xt

Iterasi 10
f(x1)   f(1.73046875) = -0.014957249164581299
f(x2)   f(1.732421875) = 0.0035126730799674988
f(xt)   f(1.7314453125) = -0.005728195421397686
=> Lebih besar dari nol, x1 = xt

Iterasi 11
f(x1)   f(1.7314453125) = -0.005728195421397686
f(x2)   f(1.732421875) = 0.0035126730799674988
f(xt)   f(1.73193359375) = -0.0011092383647337556
=> Lebih besar dari nol, x1 = xt

Iterasi 12
f(x1)   f(1.73193359375) = -0.0011092383647337556
f(x2)   f(1.732421875) = 0.0035126730799674988
f(xt)   f(1.732177734375) = 0.0012013480154564604
=> Lebih kecil dari nol, x2 = xt

Iterasi 13
f(x1)   f(1.73193359375) = -0.0011092383647337556
f(x2)   f(1.732177734375) = 0.0012013480154564604
f(xt)   f(1.7320556640625) = 4.596249527821783e-05
=> Lebih kecil dari nol, x2 = xt

Iterasi 14
f(x1)   f(1.73193359375) = -0.0011092383647337556
f(x2)   f(1.7320556640625) = 4.596249527821783e-05
f(xt)   f(1.73199462890625) = -0.0005316610165664315
=> Lebih besar dari nol, x1 = xt

Iterasi 15
f(x1)   f(1.73199462890625) = -0.0005316610165664315
f(x2)   f(1.7320556640625) = 4.596249527821783e-05
f(xt)   f(1.732025146484375) = -0.00024285503118903762
=> Lebih besar dari nol, x1 = xt

Iterasi 16
f(x1)   f(1.732025146484375) = -0.00024285503118903762
f(x2)   f(1.7320556640625) = 4.596249527821783e-05
f(xt)   f(1.7320404052734375) = -9.844771060230073e-05
=> Lebih besar dari nol, x1 = xt

Iterasi 17
f(x1)   f(1.7320404052734375) = -9.844771060230073e-05
f(x2)   f(1.7320556640625) = 4.596249527821783e-05
f(xt)   f(1.7320480346679688) = -2.6242968324652338e-05
=> Lebih besar dari nol, x1 = xt

Iterasi 18
f(x1)   f(1.7320480346679688) = -2.6242968324652338e-05
f(x2)   f(1.7320556640625) = 4.596249527821783e-05
f(xt)   f(1.7320518493652344) = 9.859673310685935e-06
=> Lebih kecil dari nol, x2 = xt


Faktor ditemukan pada titik x = 1.7320518493652344

Banyak iterasi : 18
```

Gambar 8. Hasil perhitungan persamaan $x^3 + x^2 - 3x - 3$ dengan Metode Bisection

Untuk persamaan $x^3 + x^2 - 3x - 3$ dengan menggunakan Metode *Bisection*, akar ditemukan pada iterasi ke delapan belas dan pada titik ke 1,7320518493652344.

Metode Regula Falsi

Source Code dan Output

```
def regulaFalsiMethod (a, b, equations, tolerance):
    if (equations(a) * equations(b)) > 0 :
        print(f"Tidak ada faktor untuk f(x) antara titik {a} dan {b}")
        return

    x1 = a
    x2 = b
    i = 1

    while True :
        xt = x2 - (equations(x2) *
                   ((x2-x1) / (equations(x2) - equations(x1))))

        print(f"Iterasi {i}")
        print(f"f(x1)\tf({x1}) = {equations(x1)}")
        print(f"f(x2)\tf({x2}) = {equations(x2)}")
        print(f"f(xt)\tf({xt}) = {equations(xt)}")

        if (equations(x1) * equations(xt)) < 0 :
            print("=> Lebih kecil dari nol, x2 = xt\n")
            x2 = xt
        else :
            print("=> Lebih besar dari nol, x1 = xt\n")
            x1 = xt

        i += 1

        if abs(equations(xt)) < tolerance:
            break

    print(f"\nFaktor ditemukan pada titik x = {xt}")
    print(f"\nBanyak iterasi : {i-1}")
```

Gambar 9. *Source Code* python untuk *metode Regula Falsi*

Untuk *equations* $x^4 - 15x^2 - 25x + 12$, didapatkan *output* sebagai berikut:

```
In [12]: x1 = 0
         x2 = 1
         tolerance = 0.00001
         regulaFalsiMethod(x1, x2, equations, tolerance)

Iterasi 1
f(x1)   f(0) = 12
f(x2)   f(1) = -27
f(xt)   f(0.3076923076923077) = 2.8965372360911736
=> Lebih besar dari nol, x1 = xt

Iterasi 2
f(x1)   f(0.3076923076923077) = 2.8965372360911736
f(x2)   f(1) = -27
f(xt)   f(0.3747667984189723) = 0.543803987100711
=> Lebih besar dari nol, x1 = xt

Iterasi 3
f(x1)   f(0.3747667984189723) = 0.543803987100711
f(x2)   f(1) = -27
f(xt)   f(0.3871109288102115) = 0.09686019410380453
=> Lebih besar dari nol, x1 = xt

Iterasi 4
f(x1)   f(0.3871109288102115) = 0.09686019410380453
f(x2)   f(1) = -27
f(xt)   f(0.3893017566025938) = 0.01708739752850086
=> Lebih besar dari nol, x1 = xt

Iterasi 5
f(x1)   f(0.3893017566025938) = 0.01708739752850086
f(x2)   f(1) = -27
f(xt)   f(0.38968800266610704) = 0.0030093113266769222
=> Lebih besar dari nol, x1 = xt

Iterasi 6
f(x1)   f(0.38968800266610704) = 0.0030093113266769222
f(x2)   f(1) = -27
f(xt)   f(0.38975601800414617) = 0.0005298195694560803
=> Lebih besar dari nol, x1 = xt

Iterasi 7
f(x1)   f(0.38975601800414617) = 0.0005298195694560803
f(x2)   f(1) = -27
f(xt)   f(0.38976799255449635) = 9.3275142980076e-05
=> Lebih besar dari nol, x1 = xt

Iterasi 8
f(x1)   f(0.38976799255449635) = 9.3275142980076e-05
f(x2)   f(1) = -27
f(xt)   f(0.3897701006760189) = 1.6421007956068934e-05
=> Lebih besar dari nol, x1 = xt

Iterasi 9
f(x1)   f(0.3897701006760189) = 1.6421007956068934e-05
f(x2)   f(1) = -27
f(xt)   f(0.38977047180875735) = 2.890899459373486e-06
=> Lebih besar dari nol, x1 = xt


Faktor ditemukan pada titik x = 0.38977047180875735

Banyak iterasi : 9
```

Gambar 10. Hasil Perhitungan persamaan $x^4 - 15x^2 - 25x + 12$ Metode Regula Falsi

Pada persamaan $x^4 - 15x^2 - 25x + 12$ dengan Metode Regula Falsi, diperlukan sembilan iterasi untuk mendapatkan akar 0,38977047180875735.

Untuk *equations* $x^3 + x^2 - 3x - 3$, didapatkan *output* sebagai berikut:

```
In [17]: x1 = 1
         x2 = 2
         tolerance = 0.00001
         regulaFalsiMethod(x1, x2, equations, tolerance)

Iterasi 1
f(x1)   f(1) = -4
f(x2)   f(2) = 3
f(xt)   f(1.5714285714285714) = -1.3644314868804672
=> Lebih besar dari nol, x1 = xt

Iterasi 2
f(x1)   f(1.5714285714285714) = -1.3644314868804672
f(x2)   f(2) = 3
f(xt)   f(1.7054108216432866) = -0.24774509963859614
=> Lebih besar dari nol, x1 = xt

Iterasi 3
f(x1)   f(1.7054108216432866) = -0.24774509963859614
f(x2)   f(2) = 3
f(xt)   f(1.7278827284910738) = -0.03933955131148981
=> Lebih besar dari nol, x1 = xt

Iterasi 4
f(x1)   f(1.7278827284910738) = -0.03933955131148981
f(x2)   f(2) = 3
f(xt)   f(1.731404865845108) = -0.0061106730936844045
=> Lebih besar dari nol, x1 = xt

Iterasi 5
f(x1)   f(1.731404865845108) = -0.0061106730936844045
f(x2)   f(2) = 3
f(xt)   f(1.7319508527490717) = -0.000945920667013489
=> Lebih besar dari nol, x1 = xt

Iterasi 6
f(x1)   f(1.7319508527490717) = -0.000945920667013489
f(x2)   f(2) = 3
f(xt)   f(1.732035343851165) = -0.00014634871411534078
=> Lebih besar dari nol, x1 = xt

Iterasi 7
f(x1)   f(1.732035343851165) = -0.00014634871411534078
f(x2)   f(2) = 3
f(xt)   f(1.7320484153077866) = -2.26405665912921e-05
=> Lebih besar dari nol, x1 = xt

Iterasi 8
f(x1)   f(1.7320484153077866) = -2.26405665912921e-05
f(x2)   f(2) = 3
f(xt)   f(1.7320504374844243) = -3.5025160194379623e-06
=> Lebih besar dari nol, x1 = xt


Faktor ditemukan pada titik x = 1.7320504374844243

Banyak iterasi : 8
```

Gambar 11. Hasil perhitungan persamaan $x^3 + x^2 - 3x - 3$ dengan Metode Regula Falsi

Untuk mendapatkan akar dengan nilai 1,7320504374844243 pada persamaan $x^3 + x^2 - 3x - 3$ yang menggunakan Metode Regula Falsi ini, diperlukan sebanyak delapan iterasi.

Metode Secant

Source Code dan Output

```
def secantMethod (a, b, equations, tolerance):
    if (equations(a) * equations(b)) > 0 :
        print(f"Tidak ada faktor untuk f(x) antara titik {a} dan {b}")
        return

    x0 = a
    x1 = b
    i = 1

    while True :
        x2 = x1 - (equations(x1) *
                   ((x1-x0) / (equations(x1) - equations(x0))))

        print(f"Iterasi {i}")
        print(f"f(x0)\tf({x0}) = {equations(x0)}")
        print(f"f(x1)\tf({x1}) = {equations(x1)}")
        print(f"f(x2)\tf({x2}) = {equations(x2)}\n")

        x0 = x1
        x1 = x2

        i += 1

        if abs(equations(x2)) < tolerance:
            break

    print(f"Faktor ditemukan pada titik x = {x2}")
    print(f"\nBanyak iterasi : {i-1}")
```

Gambar 12. *Source Code* phyton untuk *metode secant*

Untuk *equations* $x^4 - 15x^2 - 25x + 12$, didapatkan *output* sebagai berikut:

```
In [14]: x1 = 0
         x2 = 1
         tolerance = 0.00001
         secantMethod(x1, x2, equations, tolerance)

Iterasi 1
f(x0)   f(0) = 12
f(x1)   f(1) = -27
f(x2)   f(0.3076923076923077) = 2.8965372360911736

Iterasi 2
f(x0)   f(1) = -27
f(x1)   f(0.3076923076923077) = 2.8965372360911736
f(x2)   f(0.37476679841897237) = 0.5438039871007092

Iterasi 3
f(x0)   f(0.3076923076923077) = 2.8965372360911736
f(x1)   f(0.37476679841897237) = 0.5438039871007092
f(x2)   f(0.3902702030253216) = -0.0182189568219826

Iterasi 4
f(x0)   f(0.37476679841897237) = 0.5438039871007092
f(x1)   f(0.3902702030253216) = -0.0182189568219826
f(x2)   f(0.38976763304633577) = 0.00010638143788277432

Iterasi 5
f(x0)   f(0.3902702030253216) = -0.0182189568219826
f(x1)   f(0.38976763304633577) = 0.00010638143788277432
f(x2)   f(0.38977055054320503) = 2.053612568886365e-08

Faktor ditemukan pada titik x = 0.38977055054320503

Banyak iterasi : 5
```

Gambar 13. Hasil Perhitungan persamaan $x^4 - 15x^2 - 25x + 12$ Metode Secant

Persamaan $x^4 - 15x^2 - 25x + 12$ yang menggunakan Metode *Secant* memerlukan lima iterasi untuk mendapatkan akar 0,38977055054320503.

Untuk *equations* $x^3 + x^2 - 3x - 3$, didapatkan *output* sebagai berikut:

```
In [32]: x1 = 1
         x2 = 2
         tolerance = 0.00001
         secantMethod(x1, x2, equations, tolerance)

Iterasi 1
f(x0)   f(1) = -4
f(x1)   f(2) = 3
f(x2)   f(1.5714285714285714) = -1.3644314868804672

Iterasi 2
f(x0)   f(2) = 3
f(x1)   f(1.5714285714285714) = -1.3644314868804672
f(x2)   f(1.7054108216432866) = -0.24774509963859614

Iterasi 3
f(x0)   f(1.5714285714285714) = -1.3644314868804672
f(x1)   f(1.7054108216432866) = -0.24774509963859614
f(x2)   f(1.735135770660739) = 0.02925540230565815

Iterasi 4
f(x0)   f(1.7054108216432866) = -0.24774509963859614
f(x1)   f(1.735135770660739) = 0.02925540230565815
f(x2)   f(1.7319963707826993) = -0.0005151769146980456

Iterasi 5
f(x0)   f(1.735135770660739) = 0.02925540230565815
f(x1)   f(1.7319963707826993) = -0.0005151769146980456
f(x2)   f(1.7320506977855836) = -1.0390001730087306e-06

Faktor ditemukan pada titik x = 1.7320506977855836

Banyak iterasi : 5
```

Gambar 14. Hasil perhitungan persamaan $x^3 + x^2 - 3x - 3$ dengan Metode Regula Falsi

Pada persamaan $x^3 + x^2 - 3x - 3$, akar 1,7320506977855836 ditemukan pada iterasi kelima dengan menggunakan Metode *Secant*.

Perbandingan banyaknya iterasi dari ketiga buah metode, yaitu Metode *Bisection*, Metode *Regula Falsi*, dan Metode *Secant* yang digunakan untuk mencari akar dari kedua buah persamaan tersebut dapat dituliskan dalam bentuk tabel di bawah ini.

Table 1. ***Hasil Perhitungan akar persamaan $x^4 - 15x^2 - 25x + 12$ dan jumlah iterasi dengan Pemrograman Python***

Nama Metode	Jumlah Iterasi	Hasil Akar Persamaan
Bisection	13	0,3897705078125
Regula Falsi	9	0,38977047180875735
Secant	5	0,38977055054320503

Table 2. ***Hasil Perhitungan akar persamaan $x^3 + x^2 - 3x - 3$dan jumlah iterasi dengan Pemrograman Python***

Nama Metode	Jumlah Iterasi	Hasil Akar Persamaan
Bisection	18	1,7320518493652344
Regula Falsi	8	1,7320504374844243
Secant	5	1,7320506977855836

KESIMPULAN DAN SARAN

Berdasarkan penelitian yang menggunakan tiga buah metode, yaitu Metode Bisection, Metode Regula Falsi, dan Metode Secant yang diimplementasikan menggunakan bahasa pemrograman Python pada dua buah persamaan, dapat disimpulkan bahwa Metode Bisection memerlukan iterasi yang paling banyak untuk mencari akar sebuah persamaan

dibandingkan dengan Metode Regula Falsi dan Metode Secant, yaitu memerlukan tiga belas iterasi pada persamaan $x^4 - 15x^2 - 25x + 12$ dan delapan belas iterasi pada persamaan $x^3 + x^2 - 3x - 3$, sedangkan metode yang memerlukan iterasi paling sedikit di antara ketiga buah metode tersebut adalah Metode Secant yang memerlukan lima buah iterasi pada masing-masing persamaan yang diuji. Selanjutnya Metode Regula Falsi memerlukan iterasi yang lebih sedikit dibandingkan dengan Metode Bisection, akan tetapi memerlukan iterasi yang lebih banyak daripada Metode Secant, yaitu memerlukan sembilan metode pada persamaan pertama dan delapan iterasi pada persamaan kedua. Sehingga dapat disimpulkan bahwa dari ketiga metode tersebut, Metode Secant adalah metode yang dapat menemukan akar persamaan dengan jumlah iterasi yang paling sedikit, Metode Regula Falsi menemukan akar persamaan dengan jumlah yang lebih banyak dari Metode Secant tapi lebih sedikit dari Metode Bisection, dan Metode Bisection memerlukan jumlah iterasi yang paling banyak dalam menemukan akar suatu persamaan.

PENELITIAN LANJUTAN

Untuk penelitian lebih lanjut, dapat menggunakan kombinasi metode-metode penyelesaian persamaan nonlinear lain yang dibandingkan dengan menggunakan bahasa pemrograman lainnya.

DAFTAR PUSTAKA

Abbasbandy, S., & Liao, S. J. (2008). A New Modification of False Position Method Based on Homotopy Analysis Method. *Applied Mathematics and Mechanics (English Edition)*, *29*(2), 223–228. https://doi.org/10.1007/s10483-008-0209-z

Anton, H., & Rorres, C. (2013). *Elementary Linear Algebra Applications Version* (11th ed.). Wiley.

Asminah, & Sahfitri, V. (2012). Implementasi dan Analisis Tingkat Akurasi Software Penyelesaian Persamaan Non Linier dengan Metode Fixed Point Iteration dan Metode Bisection. *Seminar Nasional Informatika*.

Azure, I., Aloliga, G., & Doabil, L. (2019). Comparative Study of Numerical Methods for Solving Non-linear Equations Using Manual Computation. *Mathematics Letters*, *5*(4), 41. https://doi.org/10.11648/j.ml.20190504.11

Burden, R. L., & Faires, J. D. (2010). *Numerical Analysis* (9th ed.). Brooks/Cole, Cengage Learning.

Dawson, M. (2010). *Python programming for the absolute beginner*. Course Technology Cengage Learning.

Guanabara, E., Ltda, K., Guanabara, E., & Ltda, K. (2011). *Numerical Analysis* (Ninth Edit). Richard Stratton.

Massinanke, S., & Zhang, C. Z. (2014). Comparison of genetic algorithm and bisection method for finding roots in one-dimension space. *Advanced Materials Research*, *1048*, 531–536. https://doi.org/10.4028/www.scientific.net/AMR.1048.531

McKinney, W. (2022). *Python for data analysis : data wrangling with pandas, NumPy, and Jupyter* (3rd ed.). O'Reilly Media, Inc.

Pei, X. (2018). Analysis of Programming Language and Software Development by Computer. *Advances in Computer Science Research*.

Peslak, A., & Conforti, M. (2020). COMPUTER PROGRAMMING LANGUAGES IN 2020: WHAT WE USE, WHO USES THEM, AND HOW DO THEY IMPACT JOB SATISFACTION. *Issues in Information Systems*, *21*(2), 259–269. https://doi.org/10.48009/2_iis_2020_259-269

Rahayuni, I. A. E., Dharmawan, K., & Harini, L. P. I. (2016). PERBANDINGAN KEEFISIENAN METODE NEWTON-RAPHSON, METODE SECANT, DAN METODE BISECTION DALAM MENGESTIMASI IMPLIED VOLATILITIES SAHAM. *E-Jurnal Matematika*, *5*(1), 1–6. http://finance.yahoo.com,

Sulaiman, I. M., Mamat, M., Waziri, M. Y., Fadhilah, A., & Kamfa, K. U. (2016). Regula Falsi Method for Solving Fuzzy Nonlinear Equation. *Far East Journal of Mathematical Sciences*, *100*(6), 873–884. https://doi.org/10.17654/MS100060873

Sunandar, E. (2019). Penyelesaian Sistem Persamaan Non-Linier Dengan Metode Bisection & Metode Regula Falsi Menggunakan Bahasa Program Java. *PETIR: Jurnal Pengkajian Dan Penerapan Teknik Informatika*, *12*(2).

Sunandar, E., & Indrianto, I. (2020). Perbandingan Metode Newton-Raphson & Metode Secant Untuk Mencari Akar Persamaan Dalam Sistem Persamaan Non-Linier. *PETIR*, *13*(1), 72–79. https://doi.org/10.33322/petir.v13i1.893