TUGAS INDIVIDU 2
SOLUSI PERSAMAAN NONLINEAR

Mata Kuliah: Metode Numerik

Disusun oleh:

Muchammad Yuda Tri Ananda
24060124110142
Kelas D

PROGRAM STUDI INFORMATIKA
FAKULTAS SAINS DAN MATEMATIKA
UNIVERSITAS DIPONEGORO
2025

1 Nomer 1 EXERCISE SET 2.1 Hal. 54 3.c

Use the Bisection method to find solutions accurate to within 10^{-2} for the equation:

$$f(x) = x^3 - 7x^2 + 14x - 6 = 0$$

on each of the following intervals:

- **a.** $[0, 1]$
- **b.** $[1, 3.2]$
- **c.** $[3.2, 4]$

1.1 Teori Dasar

Metode biseksi adalah salah satu metode numerik paling sederhana untuk mencari akar persamaan nonlinear. Metode ini menerapkan teorema nilai tengah (intermediate value theorem) yang menyatakan bahwa jika $f(a)$ dan $f(b)$ memiliki tanda yang berbeda, maka terdapat setidaknya satu nilai c dalam interval $[a, b]$ sehingga $f(c) = 0$.

Langkah-langkah metode biseksi adalah sebagai berikut:

1. Pilih interval $[a, b]$ di mana $f(a)$ dan $f(b)$ memiliki tanda berbeda.
2. Hitung titik tengah $c = \frac{a+b}{2}$.
3. Evaluasi $f(c)$. Jika $f(c) = 0$ atau $|f(c)| < \epsilon$ (toleransi), maka c adalah akar yang dicari.
4. Jika $f(c)$ dan $f(a)$ memiliki tanda yang sama, ganti a dengan c. Jika $f(c)$ dan $f(b)$ memiliki tanda yang sama, ganti b dengan c.
5. Ulangi langkah 2-4 hingga ditemukan akar atau mencapai toleransi yang diinginkan.

Kelebihan metode biseksi adalah kesederhanaan implementasi dan jaminan konvergensi jika asumsi awal terpenuhi. Namun, kekurangannya adalah konvergensi yang relatif lambat dibandingkan metode lain.

1.2 SOAL

1.2.1 $f(x) = x^3 - 7x^2 + 14x - 6 = 0$ pada interval $[3.2, 4]$

Mencari akar dari persamaan $f(x) = x^3 - 7x^2 + 14x - 6 = 0$ pada interval $[3.2, 4]$ dengan toleransi kesalahan 10^{-2}:

Iterasi	a	b	c	f(a)	f(c)	Interval Baru
1	3.2	4.0	3.6	-0.448	0.336	[3.2, 3.6]
2	3.2	3.6	3.4	-0.448	-0.016	[3.4, 3.6]
3	3.4	3.6	3.5	-0.016	0.125	[3.4, 3.5]
4	3.4	3.5	3.45	-0.016	0.046	[3.4, 3.45]
5	3.4	3.45	3.425	-0.016	0.013	[3.4, 3.425]
6	3.4	3.425	3.4125	-0.016	-0.002	Konvergen

Table 1: Iterasi metode biseksi untuk $f(x) = x^3 - 7x^2 + 14x - 6 = 0$ pada $[3.2, 4]$

Akar ditemukan pada $x \approx 3.4125$ setelah 6 iterasi, dengan kesalahan kurang dari 10^{-2}.

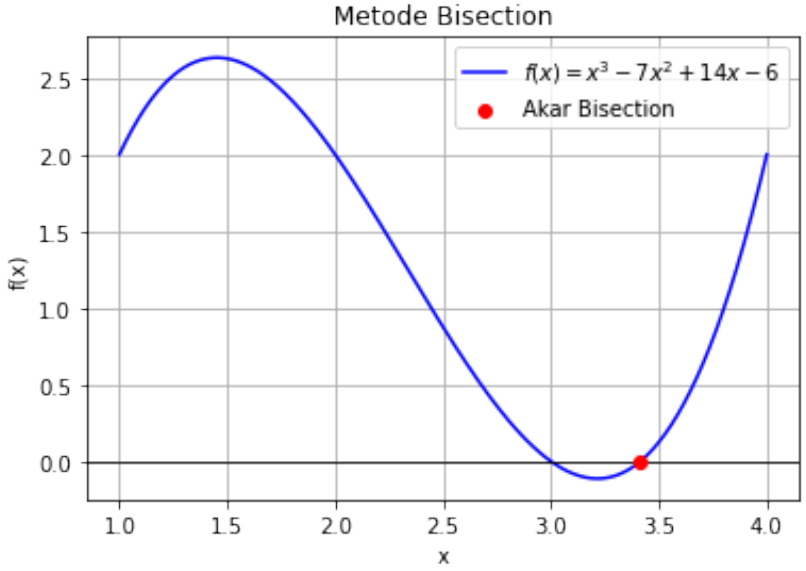

Figure 1: Visualisasi metode biseksi untuk persamaan $f(x) = x^3 - 7x^2 + 14x - 6 = 0$

2 Nomer 2 EXERCISE SET 2.3 Hal. 75 6.d

Use Newton's method to find solutions accurate to within 10^{-5} for the following problems:

$$(x-2)^2 - \ln(x) = 0$$

for $1 \leq x \leq 2$ and $e \leq x \leq 4$.

2.1 Teori Dasar

Metode Newton-Raphson adalah salah satu metode numerik yang paling efisien untuk mencari akar persamaan nonlinear. Metode ini menggunakan pendekatan garis singgung (tangent) untuk memperkirakan akar.

Rumus iterasi Newton-Raphson adalah:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Langkah-langkah metode Newton-Raphson:

1. Pilih tebakan awal x_0.
2. Hitung $f(x_0)$ dan $f'(x_0)$.
3. Hitung $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$.
4. Ulangi hingga $|x_{n+1} - x_n| < \epsilon$ atau $|f(x_{n+1})| < \epsilon$.

Kelebihan metode Newton-Raphson adalah konvergensi kuadratik (sangat cepat) jika tebakan awal cukup dekat dengan akar. Kekurangannya adalah memerlukan perhitungan turunan yang bisa sulit untuk beberapa fungsi, dan tidak selalu konvergen.

2.2 SOAL

2.2.1 $(x-2)^2 - \ln(x) = 0$

Mencari akar dari persamaan $(x-2)^2 - \ln(x) = 0$ dengan toleransi kesalahan 10^{-5}:

1. Mendefinisikan $f(x) = (x-2)^2 - \ln(x)$ dan $f'(x) = 2(x-2) - \frac{1}{x}$
2. Menggunakan tebakan awal $x_0 = 1.5$ untuk interval $[1, 2]$

Iterasi	x_n	$f(x_n)$	$f'(x_n)$	x_{n+1}
1	1.500000	-0.155465	-1.666667	1.406721
2	1.406721	0.010719	-1.897431	1.412370
3	1.412370	0.000040	-1.883290	1.412391
4	1.412391	0.000000	-1.883237	1.412391

Table 2: Iterasi metode Newton-Raphson untuk $(x-2)^2 - \ln(x) = 0$ pada $[1, 2]$

Akar ditemukan pada $x \approx 1.412391$ setelah 4 iterasi dengan kesalahan kurang dari 10^{-5}.

1. Menggunakan tebakan awal $x_0 = 3$ untuk interval $[e, 4]$

Iterasi	x_n	$f(x_n)$	$f'(x_n)$	x_{n+1}
1	3.000000	-0.098612	1.666667	3.059167
2	3.059167	0.003693	1.791448	3.057106
3	3.057106	0.000004	1.787105	3.057104

Table 3: Iterasi metode Newton-Raphson untuk $(x-2)^2 - \ln(x) = 0$ pada $[e, 4]$

Akar ditemukan pada $x \approx 3.057104$ setelah 3 iterasi dengan kesalahan kurang dari 10^{-5}.

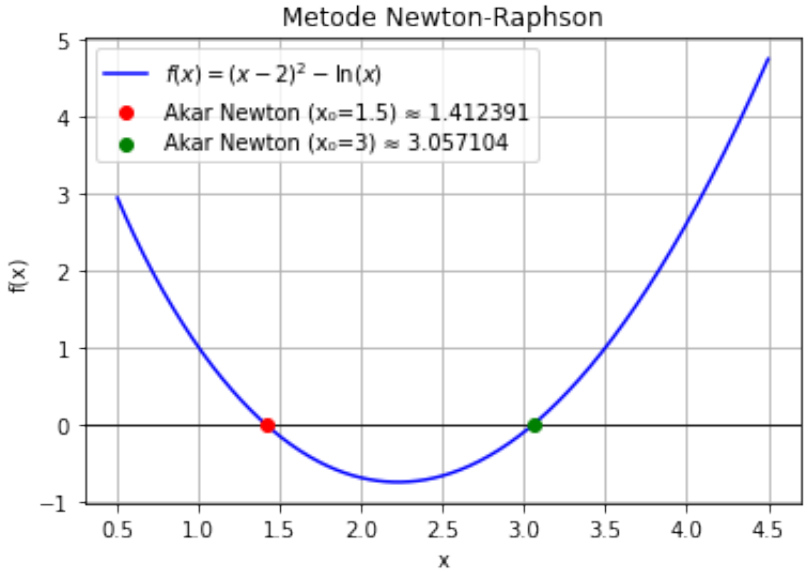

Figure 2: Visualisasi metode Newton-Raphson untuk persamaan $(x-2)^2 - \ln(x) = 0$

3 Nomer 3 EXERCISE SET 2.3 Hal. 75 7.c

Repeat Exercise 5 using the Secant method.

Use Secant's method to find solutions accurate to within 10^{-4} for the following problem:

$$x - \cos(x) = 0, \quad [0, \pi/2]$$

3.1 Teori Dasar

Metode secant adalah pengembangan dari metode Newton-Raphson yang menghindari perhitungan turunan secara eksplisit. Metode ini menggunakan dua titik untuk memperkirakan turunan melalui perbedaan terbagi.

Rumus iterasi metode secant adalah:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Langkah-langkah metode secant:

1. Pilih dua tebakan awal x_0 dan x_1.
2. Hitung x_2 menggunakan rumus secant.
3. Evaluasi $f(x_2)$. Jika cukup dekat dengan nol (sesuai toleransi kesalahan), berhenti.
4. Jika belum memenuhi toleransi, perbarui $x_0 = x_1$ dan $x_1 = x_2$, lalu ulangi proses hingga konvergen.

Kelebihan metode secant adalah tidak memerlukan perhitungan turunan sehingga lebih efisien untuk fungsi yang kompleks. Kekurangannya adalah konvergensi yang lebih lambat dibandingkan metode Newton-Raphson, tetapi lebih cepat dibandingkan metode biseksi.

3.2 SOAL

3.2.1 $x - \cos(x) = 0$ pada interval $[0, \pi/2]$

Mencari akar dari persamaan $x - \cos(x) = 0$ pada interval $[0, \pi/2]$ dengan toleransi kesalahan 10^{-4}:

Iterasi	x_0	x_1	$x_{\mathbf{new}}$	$f(x_{\mathbf{new}})$
1	0.0000000000	1.5707963268	0.6110154704	-0.2080503951
2	1.5707963268	0.6110154704	0.7232695414	-0.0263762877
3	0.6110154704	0.7232695414	0.7395671070	0.0008067229
4	0.7232695414	0.7395671070	0.7390834365	-0.0000028396
5	0.7395671070	0.7390834365	0.7390851330	-0.0000000003

Table 4: Iterasi metode secant untuk $x - \cos(x) = 0$ pada $[0, \pi/2]$

Akar ditemukan pada $x \approx 0.7390851330$ setelah 5 iterasi dengan kesalahan kurang dari 10^{-4}.

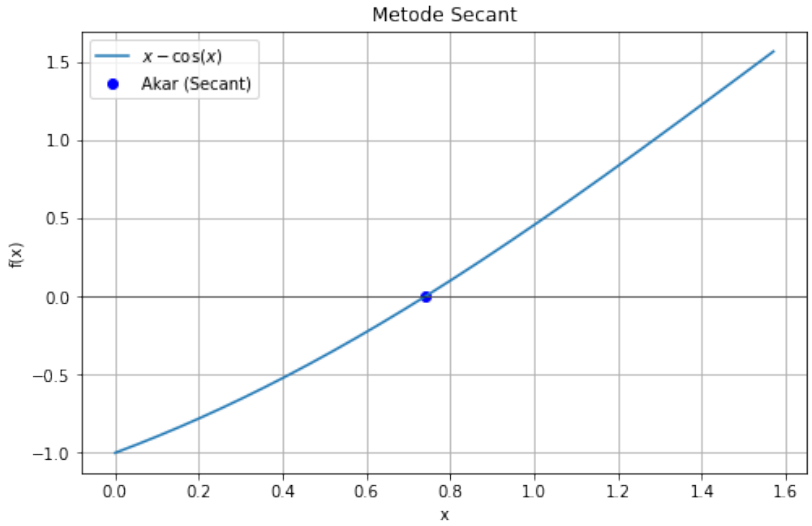

Figure 3: Visualisasi metode secant untuk persamaan $x - \cos(x) = 0$

4 Nomer 4 EXERCISE SET 2.3 Hal. 75 10.c

Repeat Exercise 6 using the method of False Position.

Use the False Position method to find solutions accurate to within 10^{-5} for the following problem:

$$2x\cos(2x) - (x-2)^2 = 0$$

for $2 \leq x \leq 3$ and $3 \leq x \leq 4$.

4.1 Teori Dasar

Metode regula falsi (false position) adalah variasi dari metode biseksi yang menggunakan interpolasi linear untuk memperkirakan akar. Mirip dengan metode biseksi,

regula falsi membagi interval menjadi dua bagian, tetapi titik bagi dipilih berdasarkan interpolasi linear antara kedua titik ujung.

Rumus untuk menghitung titik bagi pada metode regula falsi adalah:

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

Langkah-langkah metode regula falsi:

1. Pilih interval $[a, b]$ di mana $f(a)$ dan $f(b)$ memiliki tanda berbeda.
2. Hitung titik c menggunakan rumus di atas.
3. Evaluasi $f(c)$. Jika $f(c) = 0$ atau $|f(c)| < \epsilon$ (toleransi), maka c adalah akar yang dicari.
4. Jika $f(c)$ dan $f(a)$ memiliki tanda yang sama, ganti a dengan c. Jika $f(c)$ dan $f(b)$ memiliki tanda yang sama, ganti b dengan c.
5. Ulangi langkah 2-4 hingga ditemukan akar atau mencapai toleransi yang diinginkan.

Kelebihan metode regula falsi adalah konvergensi yang lebih cepat dibandingkan metode biseksi dan memiliki jaminan konvergensi jika asumsi awal terpenuhi. Kekurangannya adalah konvergensi yang masih lebih lambat dibandingkan metode Newton-Raphson dan secant untuk beberapa kasus.

4.2 SOAL

4.2.1 $2x\cos(2x) - (x-2)^2 = 0$

Mencari akar dari persamaan $2x\cos(2x) - (x-2)^2 = 0$ dengan toleransi kesalahan 10^{-5}:

1. Untuk interval $[2, 3]$:

Iterasi	a	b	c	f(c)
1	2.0000000000	3.0000000000	2.3544899167	-0.1417166747
2	2.3544899167	3.0000000000	2.3731487834	0.0216693873
3	2.3544899167	2.3731487834	2.3706741157	-0.0001125971
4	2.3706741157	2.3731487834	2.3706869080	-0.0000000853

Table 5: Iterasi metode regula falsi untuk $2x\cos(2x) - (x-2)^2 = 0$ pada $[2, 3]$

Akar ditemukan pada $x \approx 2.3706869080$ setelah 4 iterasi dengan kesalahan kurang dari 10^{-5}.

1. Untuk interval $[3, 4]$:

Iterasi	a	b	c	f(c)
1	3.0000000000	4.0000000000	3.4796988586	3.2384648091
2	3.4796988586	4.0000000000	3.6802325020	0.6636609839
3	3.6802325020	4.0000000000	3.7166480044	0.0887765806
4	3.7166480044	4.0000000000	3.7214369041	0.0110141251
5	3.7214369041	4.0000000000	3.7220297776	0.0013530372
6	3.7220297776	4.0000000000	3.7221025904	0.0001660115
7	3.7221025904	4.0000000000	3.7221115239	0.0000203658
8	3.7221115239	4.0000000000	3.7221126199	0.0000024984

Table 6: Iterasi metode regula falsi untuk $2x\cos(2x) - (x-2)^2 = 0$ pada $[3, 4]$

Akar ditemukan pada $x \approx 3.7221126199$ setelah 8 iterasi dengan kesalahan kurang dari 10^{-5}.

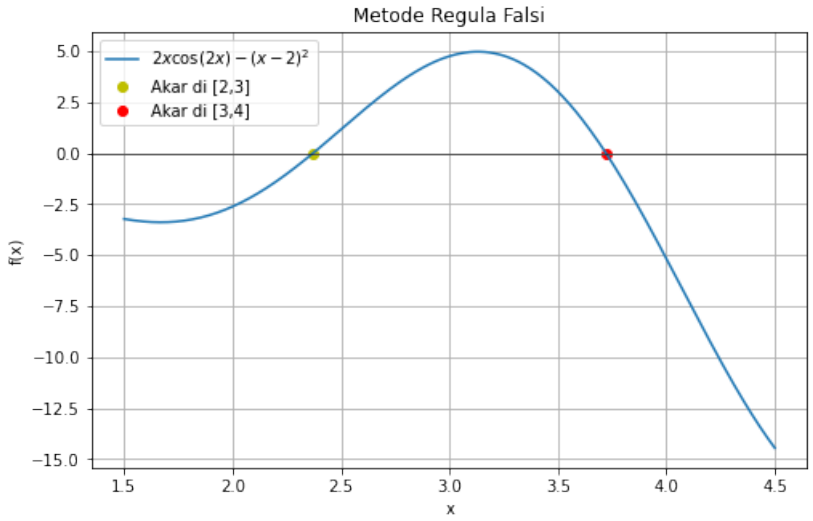

Figure 4: Visualisasi metode regula falsi untuk persamaan $2x\cos(2x) - (x-2)^2 = 0$

5 Analisis Perbandingan Metode

5.1 Kecepatan Konvergensi

Berdasarkan hasil implementasi, dapat disimpulkan bahwa metode Newton-Raphson memiliki konvergensi paling cepat, diikuti oleh metode secant, metode regula falsi, dan terakhir metode biseksi. Hal ini sesuai dengan teori matematika yang menyatakan:

- Metode Newton-Raphson memiliki konvergensi kuadratik (orde 2)
- Metode secant memiliki konvergensi super-linear (orde 1.618)
- Metode regula falsi memiliki konvergensi linear (orde 1)
- Metode biseksi memiliki konvergensi linear (orde 1)

5.2 Keandalan

Ditinjau dari segi keandalan:

- Metode biseksi dan regula falsi selalu konvergen jika fungsi kontinu dan interval awal memiliki tanda berbeda pada kedua ujungnya.
- Metode Newton-Raphson dan secant tidak selalu konvergen dan sangat bergantung pada tebakan awal.

5.3 Kompleksitas Implementasi

Ditinjau dari segi kompleksitas implementasi:

- Metode biseksi dan regula falsi relatif sederhana untuk diimplementasikan.
- Metode Newton-Raphson memerlukan perhitungan turunan yang bisa kompleks untuk beberapa fungsi.
- Metode secant tidak memerlukan turunan tetapi memerlukan dua tebakan awal.

6 Kesimpulan

Berdasarkan hasil implementasi dan analisis perbandingan metode-metode numerik untuk mencari akar persamaan nonlinear, dapat disimpulkan bahwa:

1. Metode biseksi merupakan metode paling sederhana dan selalu konvergen, namun memiliki konvergensi paling lambat.
2. Metode Newton-Raphson memiliki konvergensi paling cepat, tetapi memerlukan perhitungan turunan dan sensitif terhadap tebakan awal.
3. Metode secant menawarkan keseimbangan antara kecepatan konvergensi dan kompleksitas implementasi, karena tidak memerlukan perhitungan turunan.
4. Metode regula falsi merupakan perbaikan dari metode biseksi, dengan konvergensi lebih cepat dan tetap mempertahankan jaminan konvergensi.

Pemilihan metode tergantung pada karakteristik masalah yang dihadapi. Untuk mendapatkan solusi cepat dengan tebakan awal yang baik, metode Newton-Raphson adalah pilihan terbaik. Untuk jaminan konvergensi tanpa tebakan awal yang tepat, metode biseksi atau regula falsi lebih disarankan. Untuk fungsi dengan turunan yang kompleks atau sulit dihitung, metode secant lebih cocok digunakan.

7 Daftar Pustaka

References

[1] Burden, R. L., & Faires, J. D. (2011). *Numerical Analysis* (9th ed.). Brooks/Cole.

[2] Chapra, S. C., & Canale, R. P. (2015). *Numerical Methods for Engineers* (7th ed.). McGraw-Hill Education.

[3] Quarteroni, A., Sacco, R., & Saleri, F. (2010). *Numerical Mathematics* (2nd ed.). Springer.

[4] DeVries, P. L., & Hasbun, J. E. (2011). *A First Course in Computational Physics* (2nd ed.). Jones & Bartlett Learning.

[5] Atkinson, K. E. (2008). *An Introduction to Numerical Analysis* (2nd ed.). John Wiley & Sons.

Lampiran

Kode Program Metode Biseksi (Python)

```
# Definisi variabel simbolik
X = Symbol("x")

# FUNGSI
f = X**3 - 7 * X**2 + 14 * X - 6

# INTERVAL
a, b = 3.2, 4

# TOLERANSI
tol = 1e-2

def bisection_method(f, a, b, tol=1e-2):
    print("=== Metode Bisection ===")
    for i in range(100):
        c = (a + b) / 2  # Titik tengah
        f_c = f.subs(X, c)
        f_a = f.subs(X, a)
        f_b = f.subs(X, b)

        print(f"Iterasi {i+1}:")
        print(f"  Titik Tengah: c_{i+1} = ({a} + {b}) / 2 = {c}")
        print(f"  |a - b| = |{a} - {b}| = {abs(a - b)}")
        print(f"  f(c_{i+1}) = f({c}) = {f_c}")

        # Jika akar ditemukan atau sudah mencapai toleransi,
            berhenti
        if abs(f_c) < tol or abs(a - b) < tol:
            print(f"  Konvergen: Akar ditemukan pada x     {c}\n")
            return c

        # Perbarui interval berdasarkan tanda f(c)
        if f_c * f_a < 0:
            print(f"  Karena f(c) * f(a) < 0, interval baru: [{a},
                {c}]")
            b = c
        else:
            print(f"  Karena f(c) * f(b) < 0, interval baru: [{c},
                {b}]")
```

```
            a = c

        print("-" * 50)

    return c  # Estimasi terbaik setelah iterasi maksimum

# Menjalankan metode bisection
akar_bisection = bisection_method(f, a, b, tol)
print("SOLUSI Akar (Metode Bisection):", akar_bisection)
```

Listing 1: Implementasi Metode Biseksi dalam Python

Kode Program Metode Newton-Raphson (Python)

```
# Definisi fungsi f(x) dan turunannya f'(x)
def f(x):
    return (x - 2) ** 2 - np.log(x)

def df(x):
    return 2 * (x - 2) - 1 / x

def metode_newton_raphson(f, df, x0, tol=1e-5, max_iter=100):
    x = x0
    print(f"\n=== Metode Newton-Raphson (x0 = {x0}) ===")

    for i in range(1, max_iter + 1):
        f_x = f(x)
        df_x = df(x)

        # Cek jika turunan nol (hindari pembagian oleh nol)
        if df_x == 0:
            print(f"Iterasi {i}: Turunan nol! Newton-Raphson gagal.")
            return None

        x_new = x - f_x / df_x

        print(
            f"Iterasi {i}: x = {x:.6f}, f(x) = {f_x:.6f}, f'(x) = {df_x:.6f}, x_baru = {x_new:.6f}"
        )

        if abs(x_new - x) < tol:
            print(f"Konvergen: Akar ditemukan pada x     {x_new:.6f}\n")
            return x_new

        x = x_new

    print("Maksimum iterasi tercapai, hasil terakhir:", x)
    return x

# Interval pertama: 1     x     2 (ambil x0 = 1.5)
akar1 = metode_newton_raphson(f, df, x0=1.5)

# Interval kedua: e     x     4 (ambil x0 = 3)
akar2 = metode_newton_raphson(f, df, x0=3)
```

Listing 2: Implementasi Metode Newton-Raphson dalam Python

Kode Program Metode Secant (Python)

```
# Fungsi yang ingin kita cari akarnya
def f(x):
    return x - np.cos(x)

def secant_method(f, x0, x1, tol=1e-4, max_iter=100):
    print(f"{'Iterasi':<8} {'x0':<15} {'x1':<15} {'x_baru':<15} {'f(x_baru)':<15}")

    for i in range(1, max_iter + 1):
        f_x0, f_x1 = f(x0), f(x1)
        if f_x1 - f_x0 == 0:
            print("Dibagi dengan nol! Metode gagal.")
            return None

        x_new = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)
        f_xnew = f(x_new)

        print(f"{i:<8} {x0:<15.10f} {x1:<15.10f} {x_new:<15.10f} {f_xnew:<15.10f}")

        if abs(x_new - x1) < tol:
            print(f"\nAkar ditemukan pada x     {x_new:.10f} setelah {i} iterasi.\n")
            return x_new

        x0, x1 = x1, x_new

    print("\nMetode tidak konvergen dalam jumlah iterasi maksimum.")
    return None


# Mencari akar di interval [0, pi/2]
x0, x1 = 0, np.pi / 2

print("Mencari akar dengan metode Secant di interval [0,   /2]:")
root_secant = secant_method(f, x0, x1)
```

Listing 3: Implementasi Metode Secant dalam Python

Kode Program Metode Regula Falsi (Python)

```
# Fungsi yang ingin kita cari akarnya
def f(x):
    return 2 * x * np.cos(2 * x) - (x - 2) ** 2

def regula_falsi_method(f, a, b, tol=1e-5, max_iter=100):
    if f(a) * f(b) > 0:
        print("Interval yang diberikan tidak memiliki akar atau memiliki akar genap.")
```

```
        return None

    print(f"{'Iterasi':<8} {'a':<15} {'b':<15} {'c':<15} {'f(c)':<15}")

    for i in range(1, max_iter + 1):
        c = b - f(b) * (b - a) / (f(b) - f(a))
        f_c = f(c)

        print(f"{i:<8} {a:<15.10f} {b:<15.10f} {c:<15.10f} {f_c:<15.10f}")

        if abs(f_c) < tol:
            print(f"\nAkar ditemukan pada x     {c:.10f} setelah {i} iterasi.\n")
            return c

        if f(a) * f_c < 0:
            b = c  # Akar ada di antara a dan c
        else:
            a = c  # Akar ada di antara c dan b

    print("\nMetode tidak konvergen dalam jumlah iterasi maksimum.")
    return None


# Mencari akar di kedua interval
a1, b1 = 2.0, 3.0
a2, b2 = 3.0, 4.0

print("Mencari akar di interval [2, 3]:")
root_falsi_1 = regula_falsi_method(f, a1, b1)

print("\nMencari akar di interval [3, 4]:")
root_falsi_2 = regula_falsi_method(f, a2, b2)
```

Listing 4: Implementasi Metode Regula Falsi dalam Python