Laporan Praktikum
Komputasi Numerik

KN - 02

SISTEM PERSAMAAN NON-LINEAR

Nama	: Rendy Putra Pratama
NPM	: 140310230037
Hari/Tanggal	: Jumat/ 22 Maret 2024
Waktu	: 08.00 WIB
Asisten	: M. Naufaldi D.

LABORATORIUM KOMPUTASI

DEPARTEMEN FISIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

2024

LEMBAR PENGESAHAN

KN - 02

SISTEM PERSAMAAN NON-LINEAR

Nama	: Rendy Putra Pratama
NPM	: 140310230037
Hari / Tanggal	: Jumat/ 22 Maret 2024
Waktu / Sesi	: 08.00 WIB
Asisten	: M. Naufaldi D.

Pretest	**Laporan Akhir**

Jatinangor, 22 Maret 2024
Asisten

(M. Naufaldi D.)

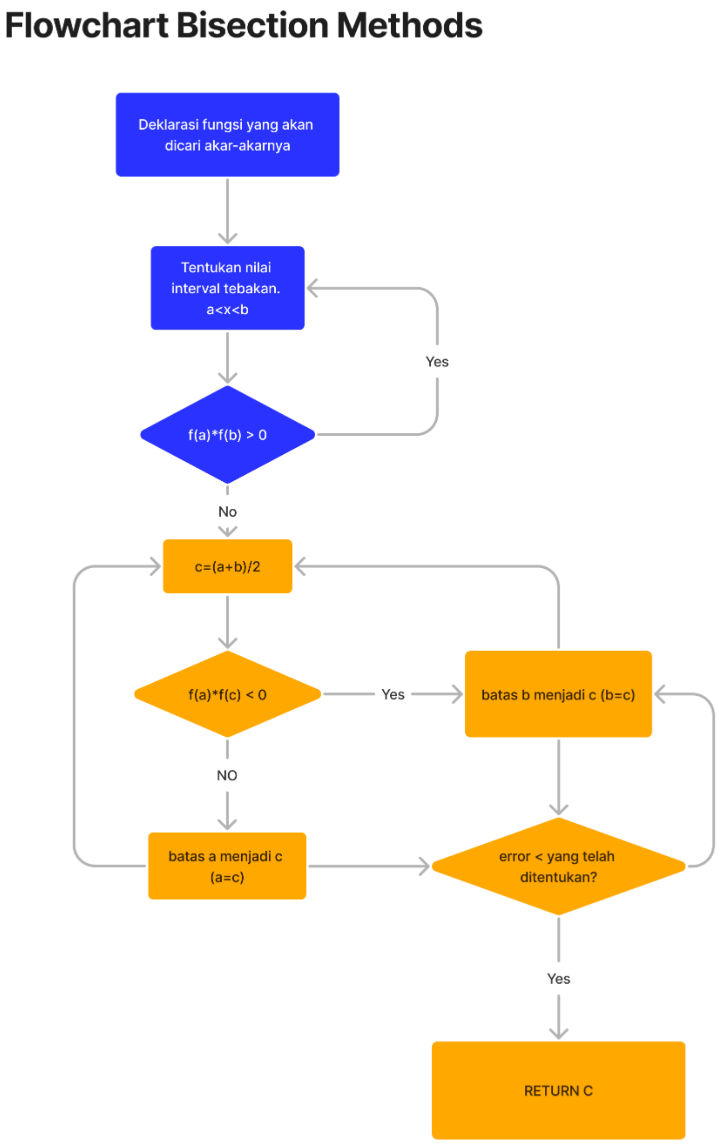
1. Tugas Praktikum

   Buatlah program dengan menggunakan bahasa Python untuk menentukan solusi dari persamaan non-linier berikut menggunakan metode *Bisection*, iterasi titik tetap, dan Newton-Raphson :

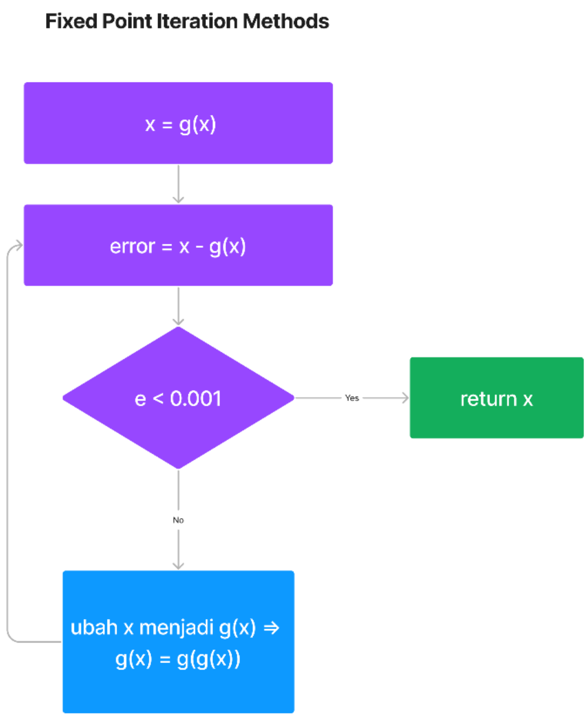
   a. $x^2 - 3x - 10 = 0$, untuk$-1 \leq x \leq 0$
   b. $\sin(x) = 0$, untuk $-\frac{\pi}{4} \leq x \leq \frac{\pi}{2}$
   c. $x^3 - 3x - 20$, untuk $1 \leq x \leq 4$
   d. $e^{-2x} - 4x$, untuk $0 \leq x \leq 1$
   e. $xe^{-x} + \cos 2x$, untuk $0 \leq x \leq 1$;$1 \leq x \leq 2$; dan $-1 \leq x \leq 1$

1.1. Flowchart

1.1.1. Flowchart : Bisection Methods



1.1.2. Flowchart : Fixed Point Iteration



1.2. Percobaan A : $x^2 - 3x - 10 = 0$

1.2.1. Bisection Methods

1.2.1.1. Listing Program

```
a = float(input("Masukkan batas awal  : "))
b = float(input("Masukkan batas akhir : "))
e = 1.0
def bisc(a,b,f,e,n):
    xA = a
    xB = b
    e = 1.0
    if f(xA)*f(xB)>0:
        while True:
            xA = float(input("Masukkan batas awal kembali  : "))
            xB = float(input("Masukkan batas akhir kembali : "))
            if f(xA)*f(xB)<0:
                break
        return None
    n = 0
    while e > 0.0001:
        xMid = (xA+xB)*0.5
        n += 1
        e = xB - xMid
        #print(f"error = {e}")
        if f(xA)*f(xB)<0:
            xB = xMid
        else:
            xA = xMid
        print(f"Hasil aproksimasi x = {xMid}\t error = {e}\t
iterasi ke-{n}")
    print(f"Hasil akar aproks = {xMid} \t Iterasi ke-{n} \t {e}")
    return xMid
f = lambda x: x**2-3*x-10
x1 = bisc(a,b,f,e,0)
print(f"Interval {a}<=x<={b}")
```

1.2.1.2. Output Program

```
Hasil aproksimasi x = -2.25000000    error = 0.25000000
iterasi ke-1
Hasil aproksimasi x = -2.12500000    error = 0.12500000
iterasi ke-2
Hasil aproksimasi x = -2.06250000    error = 0.06250000
iterasi ke-3
Hasil aproksimasi x = -2.03125000    error = 0.03125000
iterasi ke-4
Hasil aproksimasi x = -2.01562500    error = 0.01562500
iterasi ke-5
Hasil aproksimasi x = -2.00781250    error = 0.00781250
iterasi ke-6
Hasil aproksimasi x = -2.00390625    error = 0.00390625
iterasi ke-7
Hasil aproksimasi x = -2.00195312    error = 0.00195312
iterasi ke-8
Hasil aproksimasi x = -2.00097656    error = 0.00097656
iterasi ke-9
Hasil aproksimasi x = -2.00048828    error = 0.00048828
iterasi ke-10
Hasil aproksimasi x = -2.00024414    error = 0.00024414
iterasi ke-11
Hasil aproksimasi x = -2.00012207    error = 0.00012207
iterasi ke-12
Hasil aproksimasi x = -2.00006104    error = 0.00006104
iterasi ke-13
Hasil akar aproks = -2.000061        Iterasi ke-13     0.00006104
Interval -2.5<=x<=-2.0
```

1.2.2. Fixed Point Iteration

1.2.2.1. Listing Program

```
def fixPoint(a,f,e = 0.0001):
    error = 1
    n = 0
    while abs(error) > e:
        xNew = f(a)
        print(f"nilai x = {a:.6f} \t 
fx = {f(a):.6f}")
        error = xNew - a
        a = xNew
        n += 1
        if n == 10:
            return a
        print(f"Nilai fix x = {a:.6f} 
\t error = {abs(error):.6f}")

f = lambda x: (10+3*x)**(1/2)
a = float(input("Masukkan x tebakan : 
"))
xResult = fixPoint(a,f)
```

1.2.2.2. Output Program

```
nilai x = 0.500000   fx = 3.391165
Nilai fix x = 3.391165      error = 2.891165
nilai x = 3.391165   fx = 4.491491
Nilai fix x = 4.491491      error = 1.100326
nilai x = 4.491491   fx = 4.845046
Nilai fix x = 4.845046      error = 0.353555
nilai x = 4.845046   fx = 4.953296
Nilai fix x = 4.953296      error = 0.108249
nilai x = 4.953296   fx = 4.985969
Nilai fix x = 4.985969      error = 0.032673
nilai x = 4.985969   fx = 4.995789
Nilai fix x = 4.995789      error = 0.009820
nilai x = 4.995789   fx = 4.998737
Nilai fix x = 4.998737      error = 0.002948
nilai x = 4.998737   fx = 4.999621
Nilai fix x = 4.999621      error = 0.000884
nilai x = 4.999621   fx = 4.999886
Nilai fix x = 4.999886      error = 0.000265
nilai x = 4.999886   fx = 4.999966
```

1.2.3. Newton-Raphson

1.2.3.1. Listing Program

```
x = float(input("Masukkan nilai tebakkan: "))
n = 0
def NewtonRaphson(f,x,d,n):
    xInitial = x
    fDeriv = d
    e = 1
    while e > 0.001:
        xFinal = xInitial -
f(xInitial)/fDeriv(xInitial) # Newton Raphson
        e = xFinal - xInitial
        xInitial = xFinal
        n += 1
        print(f"x1 dari f(x) = {x1} \t iterasi
ke - {n} \t error = {e}")
    print(f"Solusi aproksimasi terdekat dari x2
= {xFinal} \t error = {e} \t iterasi ke-{n}")
    return xFinal

    print({e})

f = lambda x: x**2-3*x-10
d = lambda x: 2*x-3

x1 = NewtonRaphson(f,x,d,n)
```

1.2.3.2. Output Program

```
x1 dari f(x) = -2.00006103515625
iterasi ke - 1     error = -0.8888888888888893
Solusi aproksimasi terdekat dari
x2 = 5.111111111111111
error = -0.8888888888888893    iterasi ke-1
```

1.2.4. Analisis Program

Pada percobaan pertama dalam menentukan nilai $x^2 - 3x - 10 = 0$, praktikan mendapatkan satu titik akar dari persamaan tersebut pada metode bisection yaitu sebesar $x = -2.000061$. Sedangkan pada metode newton Raphson praktikan dapat menentukan akar-akar dari persamaan tersebut yaitu $x = -2.000061$ dan $x = 5.1111$.

Pada metode bisection, praktikan memberikan sebuah input user untuk menentukan batas awal dan batas akhir dari nilai yang akan dicari (yaitu $-2.5 \leq x \leq 2$). Selanjutnya, melakukan inisiasi def function, dengan variabel yang telah ditentukan pada def function (xA, xB). Sesuai dengan flowchart, program akan membandingkan nilai dari perkalian f(xA) dengan f(xB). Jika perkalian kedua nilai fungsi batas tersebut menghasilkan nilai kurang dari nol, maka program akan masuk kedalam pencarian nilai tengah $xMid = \frac{xA+xB}{2}$. Akan tetapi, jika nilai perkalian tersebut masih lebih besar dari nol, program akan meminta user untuk menginput kembali batas awal dan akhir. Selanjutnya jika syarat nilai batas awal dan batas akhir terpenuhi, program akan menentukan apakah nilai f(xA)*f(xMid) < 0, jika iya, maka program akan memperbarui batas akhirnya yang awalnya xB, menjadi xMid. Dan kemudian program akan mencari xMid baru dari batas yang terbaru. Akan tetapi, apabila syarat diatas tidak terpenuhi, maka, program akan memperbarui batas awal menjadi xMid. Dan selanjutnya program akan mencari nilai tengah kembali dengan batas awal xMid dan batas akhir adalah xB. Jika syarat tersebut telah dipenuhi, program akan mengevaluasi error dari selisih batas akhir dengan nilai tengah. Ketika error telah bernilai dibawah toleransi ($e = 0.0001$), maka syarat True pada while menjadi false dan program akan menghentikan proses looping.

Pada metode Newton Raphson, praktikan melakukan inisiasi seperti pada metode bisection yaitu untuk input user nilai tebakan, dan error. Selanjutnya praktikan melakukan def function dari Newton-Raphson yaitu $xFinal = xInitial - f(x)/f'(x)$

Setelah melakukan nilai tebakan program akan melakukan pengulangan dan selanjutnya menentukan nilai error dari xFinal – xInitial. Setelah langkah tersebut berjalan, praktikan meng-*update* nilai xInitial dengan xFinal, hal ini dilakukan agar xInitial menjadi nilai dari xFinal. Hal ini dilakukan agar iterative tetap berlanjut dengan nilai xFinal sebagai xInitial.

Praktikan mendapat 2 akar sekaligus karena pada program praktikan menginput nilai x1 sebagai pemanggilan fungsi NewtonRaphson (lihat baris kode terakhir), dan selanjutnya praktikan menginput nilai xFinal pada baris kode sebelum return xFinal. Hal ini menyebabkan iterative berlangsung secara terpisah untuk x1 dan x2, proses menentukan x1 dilakukan disekitar x = 2, dan x2 melakukan iterative melanjutkan dari iterative pada x = 2.

Pada metode fixed point, sama seperti sebelumnya praktikan melakukan inisiasi variabel-variabel awal terlebih dahulu. Selanjutnya praktikan memulai program dengan kondisi while lebih dari error, sehingga ketika error hasil iterative telah mencapai kondisi false, maka while akan terhenti. Kemudian, metode fixed point ini mengubah persamaan f(x) = 0, menjadi x = g(x). Pada tahap ini praktikan menggunakan xNew sebagai x dan $f(x) = (10 + 3x)^{0.5}$ sebagai g(x). Setelah mendapat nilai xNew, program akan menentukan error dari selisih xNew dengan a atau nilai tebakan awal dan kemudian akan dilakukan update nilai a menjadi xNew, agar program akan terus mengiterasi sampai kondisi error menyalahi kondisi true pada while.

Kesimpulan dari percobaan ini ialah persamaan $x^2 - 3x - 10$ dapat diselesaikan dengan ketiga metode tersebut, Dengan nilai error dari ketiga metode berkisar pada 10^{-5}, maknanya, ketiga metode tersebut dapat mengakprosimasikan nilai nilai dari akar persamaan kuadrat tersebut.

1.3. Percobaan B : $\sin(x) = 0$

1.3.1. Bisection Methods

1.3.1.1. Listing Program

```
from math import *
a = -pi/4
b = pi
e = 1.0
def bisc(a,b,f,e,n):
    xA = a
    xB = b
    e = 1.0
    if f(xA)*f(xB)>0:
        while True:
            xA = float(input("Masukkan batas awal
kembali  : "))
            xB = float(input("Masukkan batas akhir
kembali : "))
            if f(xA)*f(xB)<0:
                break
        return None
    n = 0
    while e > 0.0001:
        xMid = (xA+xB)*0.5
        n += 1
        e = xB - xMid
        if f(xA)*f(xB)<0:
            xB = xMid
        else:
            xA = xMid
        print(f"Hasil aproksimasi x = {xMid}\t
error = {e}\t iterasi ke-{n}")
    print(f"Hasil akar aproks = {xMid} \t Iterasi
ke-{n} \t {e}")
    return xMid
f = lambda x: sin(radians(x))
x1 = bisc(a,b,f,e,0)
```

1.3.1.2. Output Program

```
Hasil aproksimasi x = 1.178097       error = 1.963495
iterasi ke-1
Hasil aproksimasi x = 0.196350       error = 0.981748
iterasi ke-2
Hasil aproksimasi x = -0.294524      error = 0.490874
iterasi ke-3
Hasil aproksimasi x = -0.539961      error = 0.245437
iterasi ke-4
Hasil aproksimasi x = -0.417243      error = 0.122718
iterasi ke-5
Hasil aproksimasi x = -0.355884      error = 0.061359
iterasi ke-6
Hasil aproksimasi x = -0.325204      error = 0.030680
iterasi ke-7
Hasil aproksimasi x = -0.309864      error = 0.015340
iterasi ke-8
Hasil aproksimasi x = -0.302194      error = 0.007670
iterasi ke-9
Hasil aproksimasi x = -0.298359      error = 0.003835
iterasi ke-10
Hasil aproksimasi x = -0.296442      error = 0.001917
iterasi ke-11
Hasil aproksimasi x = -0.295483      error = 0.000959
iterasi ke-12
Hasil aproksimasi x = -0.295004      error = 0.000479
iterasi ke-13
Hasil aproksimasi x = -0.294764      error = 0.000240
iterasi ke-14
Hasil aproksimasi x = -0.294644      error = 0.000120
iterasi ke-15
Hasil aproksimasi x = -0.294584      error = 0.000060
iterasi ke-16
Hasil akar aproks = -0.294584        Iterasi ke-16
5.99211245267961e-05
```

1.3.2. Newton-Raphson

1.3.2.1. Listing Program

```
n = 0
from math import *
from sympy import *
x = float(input("Masukkan sudut tebak sekitar -pi/4
<=x<=pi/2"))
def NewtonRaphson(f,x,d,n):
    xInitial = x
    fDeriv = d
    e = 1
    while e > 0.001:
        xFinal = xInitial -
f(xInitial)/fDeriv(xInitial) # Newton Raphson
        e = xFinal - xInitial
        xInitial = xFinal
        n += 1
        print(f"x1 dari f(x) = {xInitial:.5f} \t
iterasi ke - {n} \t error = {e}")
    print(f"Solusi aproksimasi terdekat dari x =
{xFinal} \t error = {e} \t iterasi ke-{n}")
    return xFinal

    print({e})

f = lambda x: sin(radians(x))
d = lambda x: cos(radians(x))
x1 = NewtonRaphson(f,x,d,n)
```

1.3.2.2. Output Program

```
x1 dari f(x) = 0.00000   iterasi ke - 1    error = 0
Solusi aproksimasi terdekat dari x = 0     error = 0
iterasi ke-1
```

1.3.3 Analisis Program

Pada percobaan B, praktikan melakukan pendekatan nilai sin(x) dengan metode bisection dan newton Raphson. Metode fixed point iteration tidak dapat dilakukan karena fungsi sin(x) berada dalam bentuk paling sederhana, sehingga tidak dapat ditentukan nilai x = g(x).

Dengan melakukan langkah yang sama dalam melakukan proses inisiasi dan alur koding, Praktikan mengiterasi nilai sin(x) dengan metode bisection pada batas $-\frac{\pi}{4} \leq x \leq \frac{\pi}{2}$, dan mendapatkan output senilai x = -0.294. Selanjutnya pada metode Newton-Raphson praktikan membuat turunan pertama dari sin(x) yaitu cos(x) sebagai f'(x), kemudian praktikan mencoba menghitung aproksimasi dari sin(x) pada x = 0. Dan didapatkan hasilnya sesuai dengan hasil eksak.

1.4. Percobaan C : $x^3 - 3x - 20$

1.4.1. Bisection Methods

1.4.1.1. Listing Program

```
a = float(input("Masukkan batas awal tebakan : "))
b = float(input("Masukkan batas akhir tebakan: "))
e = 1.0
def bisc(a,b,f,e,n):
    xA = a
    xB = b
    e = 1.0
    if f(xA)*f(xB)>0:
        while True:
            xA = float(input("Masukkan batas awal
kembali  : "))
            xB = float(input("Masukkan batas akhir
kembali : "))
            if f(xA)*f(xB)<0:
                break
        return None
    n = 0

    while e > 0.0001:
        xMid = (xA+xB)*0.5
        n += 1
        e = xB - xMid
        print(f"error = {e}")
        if f(xA)*f(xB)<0:
            xB = xMid
        else:
            xA = xMid
        print(f"Hasil aproksimasi x = {xMid:.6f}\t
error = {e:.6f}\t iterasi ke-{n}")
    print(f"Hasil akar aproks = {xMid:.6f} \t Iterasi
ke-{n} \t {e:.6f}")
    return xMid


f = lambda x: x**3-3*x-20
x1 = bisc(a,b,f,e,0)
```

1.4.1.2. Output Program

```
Hasil aproksimasi x = 3.250000       error = 0.250000
iterasi ke-1
Hasil aproksimasi x = 3.125000       error = 0.125000
iterasi ke-2
Hasil aproksimasi x = 3.062500       error = 0.062500
iterasi ke-3
Hasil aproksimasi x = 3.031250       error = 0.031250
iterasi ke-4
Hasil aproksimasi x = 3.046875       error = 0.015625
iterasi ke-5
Hasil aproksimasi x = 3.054688       error = 0.007812
iterasi ke-6
Hasil aproksimasi x = 3.058594       error = 0.003906
iterasi ke-7
Hasil aproksimasi x = 3.060547       error = 0.001953
iterasi ke-8
Hasil aproksimasi x = 3.061523       error = 0.000977
iterasi ke-9
Hasil aproksimasi x = 3.062012       error = 0.000488
iterasi ke-10
Hasil aproksimasi x = 3.062256       error = 0.000244
iterasi ke-11
Hasil aproksimasi x = 3.062378       error = 0.000122
iterasi ke-12
Hasil aproksimasi x = 3.062439       error = 0.000061
iterasi ke-13
Hasil akar aproks = 3.062439   Iterasi ke-13
0.000061
```

1.4.2. Fixed Point Iteration

1.4.2.1. Listing Program

```
def fixPoint(a,f,e = 0.0001):
    error = 1
    n = 0
    while abs(error) > e:
        xNew = f(a)
        print(f"nilai x = {a:.6f} \t fx = {f(a):.6f}")
        error = xNew - a
        a = xNew
        n += 1
        if n == 10:
            break
        print(f"Nilai fix x = {a:.6f} \t error =
{abs(error):.6f}")

f = lambda x:  (3*x+20)**(1/3)
a = float(input("Masukkan nilai tebakkan awal: "))
xResult = fixPoint(a,f)
```

1.4.2.2. Output Program

```
nilai x = 0.500000       fx = 2.780649
Nilai fix x = 2.780649   error = 2.280649
nilai x = 2.780649       fx = 3.048900
Nilai fix x = 3.048900   error = 0.268251
nilai x = 3.048900       fx = 3.077489
Nilai fix x = 3.077489   error = 0.028588
nilai x = 3.077489       fx = 3.080504
Nilai fix x = 3.080504   error = 0.003016
nilai x = 3.080504       fx = 3.080822
Nilai fix x = 3.080822   error = 0.000318
nilai x = 3.080822       fx = 3.080856
Nilai fix x = 3.080856   error = 0.000033
```

1.4.3. Newton-Raphson

1.4.3.1. Listing Program

```
x = float(input("Masukkan nilai tebakkan: "))
n = 0
def NewtonRaphson(f,x,d,n):
    xInitial = x
    fDeriv = d
    e = 1
    while abs(e) > 0.001:
        xFinal = xInitial - 
f(xInitial)/fDeriv(xInitial) # Newton Raphson
        e = xFinal - xInitial
        xInitial = xFinal
        n += 1
        print(f"x1 dari f(x) = {xInitial:.6f} \t 
iterasi ke - {n} \t error = {abs(e):.6f}")
    print(f"Solusi aproksimasi terdekat dari x2 = 
{xInitial:.6f} \t error = {abs(e):.6f} \t iterasi ke-
{n}")
    return xFinal

    print({e})

f = lambda x: x**3-3*x-20
d = lambda x: 3*x**2-3
```

1.4.3.2. Output Program

```
x1 dari f(x) = 4.000000        iterasi ke - 1    error 
= 2.000000
x1 dari f(x) = 3.288889        iterasi ke - 2    error 
= 0.711111
x1 dari f(x) = 3.095052        iterasi ke - 3    error 
= 0.193836
x1 dari f(x) = 3.080932        iterasi ke - 4    error 
= 0.014120
x1 dari f(x) = 3.080859        iterasi ke - 5    error 
= 0.000073
Solusi aproksimasi terdekat dari x2 = 3.080859   error 
= 0.000073   iterasi ke-5
```

1.4.4 Analisis Program

Pada percobaan ini, praktikan menentukan nilai dari $f(x) = x^3 - 3x - 20$. Pertama praktikan mencoba dengan metode bisection, dengan menginput nilai batas 3 sampai 3.5, dan didapat hasil sebesar 3.0624 dengan error 0.000061. Hasilnya mendekati juga didapat pada metode fixed point dengan nilai 3.080 dengan error 0.000033, dan juga didapat pada metode newton Raphson dengan nilai 3.080 akan tetapi dengan error sebesar 0.000073.

Dari percobaan diatas, metode fixed point menghasilkan tingkat error yang lebih rendah. Kesimpulan pada percobaan ini ialah persamaan $f(x) = x\hat{}3 - 3x - 20$ dapat diaproksimasikan akarnya dengan optimal menggunakan metode fixed position

1.5. Percobaan D : $e^{-2x} - 4x$

1.5.1. Bisection Methods

1.5.1.1. Listing Program

```
from math import *
a = float(input("Masukkan batas awal tebakan : "))
b = float(input("Masukkan batas akhir tebakan: "))
e = 1.0
def bisc(a,b,f,e,n):
    xA = a
    xB = b
    e = 1.0
    if f(xA)*f(xB)>0:
        while True:
            xA = float(input("Masukkan batas awal
kembali  : "))
            xB = float(input("Masukkan batas akhir
kembali : "))
            if f(xA)*f(xB)<0:
                break
        return None
    n = 0

    while e > 0.0001:
        xMid = (xA+xB)*0.5
        n += 1
        e = xB - xMid
        if f(xA)*f(xB)<0:
            xB = xMid
        else:
            xA = xMid
        print(f"Hasil aproksimasi x = {xMid:.6f}\t
error = {e:.6f}\t iterasi ke-{n}")
    print(f"Hasil akar aproks = {xMid:.6f} \t Iterasi
ke-{n} \t {e:.6f}")
    return xMid


f = lambda x: exp(-2*x)-4*x
x1 = bisc(a,b,f,e,0)
```

1.5.1.2. Output Program

```
Hasil aproksimasi x = 0.250000       error = 0.250000
iterasi ke-1
Hasil aproksimasi x = 0.125000       error = 0.125000
iterasi ke-2
Hasil aproksimasi x = 0.062500       error = 0.062500
iterasi ke-3
Hasil aproksimasi x = 0.093750       error = 0.031250
iterasi ke-4
Hasil aproksimasi x = 0.109375       error = 0.015625
iterasi ke-5
Hasil aproksimasi x = 0.117188       error = 0.007812
iterasi ke-6
Hasil aproksimasi x = 0.121094       error = 0.003906
iterasi ke-7
Hasil aproksimasi x = 0.123047       error = 0.001953
iterasi ke-8
Hasil aproksimasi x = 0.124023       error = 0.000977
iterasi ke-9
Hasil aproksimasi x = 0.124512       error = 0.000488
iterasi ke-10
Hasil aproksimasi x = 0.124756       error = 0.000244
iterasi ke-11
Hasil aproksimasi x = 0.124878       error = 0.000122
iterasi ke-12
Hasil aproksimasi x = 0.124939       error = 0.000061
iterasi ke-13
Hasil akar aproks = 0.124939   Iterasi ke-13
0.000061
```

1.5.2. Fixed Point Iteration

1.5.2.1. Listing Program

```
from math import *
def fixPoint(a,f,e = 0.0001):
    error = 1
    n = 0
    while abs(error) > e:
        xNew = f(a)
        print(f"nilai x = {a:.6f} \t fx = {f(a):.6f}")
        error = xNew - a
        a = xNew
        n += 1
        if n == 10:
            break
        print(f"Nilai fix x = {a:.6f} \t error =
{abs(error):.6f}")

f = lambda x: exp(-2*x)/4
a = float(input("Masukkan nilai tebakkan awal: "))
xResult = fixPoint(a,f)
```

1.5.2.2. Output Program

```
nilai x = 0.500000        fx = 0.091970
Nilai fix x = 0.091970    error = 0.408030
nilai x = 0.091970        fx = 0.207996
Nilai fix x = 0.207996    error = 0.116027
nilai x = 0.207996        fx = 0.164921
Nilai fix x = 0.164921    error = 0.043075
nilai x = 0.164921        fx = 0.179759
Nilai fix x = 0.179759    error = 0.014838
nilai x = 0.179759        fx = 0.174503
Nilai fix x = 0.174503    error = 0.005256
nilai x = 0.174503        fx = 0.176347
Nilai fix x = 0.176347    error = 0.001844
nilai x = 0.176347        fx = 0.175698
Nilai fix x = 0.175698    error = 0.000649
nilai x = 0.175698        fx = 0.175926
Nilai fix x = 0.175926    error = 0.000228
nilai x = 0.175926        fx = 0.175846
Nilai fix x = 0.175846    error = 0.000080
```

1.5.3. Newton-Raphson

1.5.3.1. Listing Program

```
from math import *
x = float(input("Masukkan nilai tebakkan: "))
n = 0
def NewtonRaphson(f,x,d,n):
    xInitial = x
    fDeriv = d
    e = 1
    while abs(e) > 0.001:
        xFinal = xInitial -
f(xInitial)/fDeriv(xInitial) # Newton Raphson
        e = xFinal - xInitial
        xInitial = xFinal
        n += 1
        print(f"x1 dari f(x) = {xInitial:.6f} \t
iterasi ke - {n} \t error = {abs(e):.6f}")
    print(f"Solusi aproksimasi terdekat dari x2 =
{xInitial:.6f} \t error = {abs(e):.6f} \t iterasi ke-
{n}")
    return xFinal

    print({e})

f = lambda x: exp(-2*x)-4*x
d = lambda x: -2*exp(-2*x)-4
```

1.5.3.2. Output Program

```
x1 dari f(x) = 0.155362        iterasi ke - 1
error = 0.344638
x1 dari f(x) = 0.175756        iterasi ke - 2
error = 0.020393
x1 dari f(x) = 0.175867        iterasi ke - 3
error = 0.000111
Solusi aproksimasi terdekat dari x2 = 0.175867
error = 0.000111   iterasi ke-3
```

1.5.4. Analisis Program

Pada percobaan ini praktikan mencari aproksimasi dari fungsi $e^{-2x} - 4x$, Dengan langkah yang sama dalam mendeklarasikan variabel dari setiap metode. Selanjutnya praktikan melakukan aproksimasi menggunakan metode bisection dengan rentang dari 0 sampai 0.5 didapatkan dari hasil metode tersebut ialah nilai x = 0.1249 dengan error = 0.000061, sedangkan pada metode fixed point praktikan mendapatkan x = 0.175 dengan error = 0.00008, dan pada newton Raphson x = 0.175867 dengan error = 0.000111.

Jika melihat nilai eksak dari fungsi tersebut, nilai akar dari $e^{-2x} - 4x$ ialah 0.17586, yang mana nilai tersebut lebih teraproksimasi pada metode newton Raphson. Maka kesimpulannya dalam percobaan ini, metode newton Raphson lebih akurat dalam menentukan nilai akar dari fungsi $e^{-2x} - 4x$. Hal ini dikarenakan metode newton Raphson membandingkan nilai dari gradien disuatu titik dengan fungsi aslinya, dan grafik $e^{-2x} - 4x$ pada interval 0 sampai 0.5 memiliki kemiringan garis yang ekstrim, sehingga pendekatan dari kemiringan garis semakin akurat.

1.6. Percobaan E : $xe^{-x} + \cos 2x$

1.6.1. Bisection Methods

1.6.1.1. Listing Program

```
from math import *
import numpy as np
a = float(input("Masukkan batas awal kembali  : "))
b = float(input("Masukkan batas akhir kembali : "))
e = 1.0
def bisc(a,b,f,e,n):
    xA = a
    xB = b
    e = 1.0
    if f(xA)*f(xB)>0:
        while True:
            xA = float(input("Masukkan batas awal 
kembali  : "))
            xB = float(input("Masukkan batas akhir 
kembali : "))
            if f(xA)*f(xB)<0:
                break
        return None
    n = 0

    while e > 0.0001:
        xMid = (xA+xB)*0.5
        n += 1
        e = xB - xMid
        if f(xA)*f(xB)<0:
            xB = xMid
        else:
            xA = xMid
        print(f"Hasil aproksimasi x = {xMid:.6f}\t 
error = {e:.6f}\t iterasi ke-{n}")
    print(f"Hasil akar aproks = {xMid:.6f} \t Iterasi 
ke-{n} \t {e:.6f}")
    return xMid
```

1.6.1.2. Output Program

```
Hasil aproksimasi x = -0.750000      error = 0.250000
iterasi ke-1
Hasil aproksimasi x = -0.875000      error = 0.125000
iterasi ke-2
Hasil aproksimasi x = -0.812500      error = 0.062500
iterasi ke-3
Hasil aproksimasi x = -0.781250      error = 0.031250
iterasi ke-4
Hasil aproksimasi x = -0.765625      error = 0.015625
iterasi ke-5
Hasil aproksimasi x = -0.757812      error = 0.007812
iterasi ke-6
Hasil aproksimasi x = -0.753906      error = 0.003906
iterasi ke-7
Hasil aproksimasi x = -0.751953      error = 0.001953
iterasi ke-8
Hasil aproksimasi x = -0.750977      error = 0.000977
iterasi ke-9
Hasil aproksimasi x = -0.750488      error = 0.000488
iterasi ke-10
Hasil aproksimasi x = -0.750244      error = 0.000244
iterasi ke-11
Hasil aproksimasi x = -0.750122      error = 0.000122
iterasi ke-12
Hasil aproksimasi x = -0.750061      error = 0.000061
iterasi ke-13
Hasil akar aproks = -0.750061        Iterasi ke-13
0.000061
```

1.6.2. Fixed Point Iteration

1.6.2.1. Listing Program

```
from math import *
def fixPoint(a,f,e = 0.0001):
    error = 1
    n = 0
    while abs(error) > e:
        xNew = f(a)
        print(f"nilai x = {a:.6f} \t fx = {f(a):.6f}")
        error = xNew - a
        a = xNew
        n += 1
        #if n == 10:
         #   break
        print(f"Nilai fix x = {a:.6f} \t error =
{abs(error):.6f}")

f = lambda x: cos(radians(2*x))/-exp(-x)
a = float(input("Masukkan nilai tebakkan awal: "))
xResult = fixPoint(a,f)
```

1.6.2.2. Output Program

```
nilai x = -0.500000      fx = -0.606438
Nilai fix x = -0.606438        error = 0.106438
nilai x = -0.606438      fx = -0.545167
Nilai fix x = -0.545167        error = 0.061271
nilai x = -0.545167      fx = -0.579640
Nilai fix x = -0.579640        error = 0.034472
nilai x = -0.579640      fx = -0.559985
Nilai fix x = -0.559985        error = 0.019654
nilai x = -0.559985      fx = -0.571108
Nilai fix x = -0.571108        error = 0.011123
nilai x = -0.571108      fx = -0.564787
Nilai fix x = -0.564787        error = 0.006321
nilai x = -0.564787      fx = -0.568371
Nilai fix x = -0.568371        error = 0.003584
nilai x = -0.568371      fx = -0.566336
Nilai fix x = -0.566336        error = 0.002035
nilai x = -0.566336      fx = -0.567490
Nilai fix x = -0.567490        error = 0.001154
nilai x = -0.567490      fx = -0.566835
Nilai fix x = -0.566835        error = 0.000655
nilai x = -0.566835      fx = -0.567207
Nilai fix x = -0.567207        error = 0.000372
nilai x = -0.567207      fx = -0.566996
Nilai fix x = -0.566996        error = 0.000211
nilai x = -0.566996      fx = -0.567116
Nilai fix x = -0.567116        error = 0.000120
nilai x = -0.567116      fx = -0.567048
Nilai fix x = -0.567048        error = 0.000068
```

1.6.3. Newton-Raphson

1.6.3.1. Listing Program

```
from math import *
x = float(input("Masukkan nilai tebakkan: "))
n = 0
def NewtonRaphson(f,x,d,n):
    xInitial = x
    fDeriv = d
    e = 1
    while abs(e) > 0.0001:
        xFinal = xInitial -
f(xInitial)/fDeriv(xInitial) # Newton Raphson
        e = xFinal - xInitial
        xInitial = xFinal
        n += 1
        print(f"x1 dari f(x) = {xInitial:.6f} \t
iterasi ke - {n} \t error = {abs(e):.6f}")
    print(f"Solusi aproksimasi terdekat dari x2 =
{xInitial:.6f} \t error = {abs(e):.6f} \t iterasi ke-
{n}")
    return xInitial

    print({e})

f = lambda x: x*exp(-x)+cos(radians(2*x))
d = lambda x: exp(-x)*(1-x)-sin(radians(2*x))*2

x1 = NewtonRaphson(f,x,d,n)
```

1.6.3.2. Output Program

```
x1 dari f(x) = -0.577582       iterasi ke - 1    error
= 0.127582
x1 dari f(x) = -0.567307       iterasi ke - 2    error
= 0.010275
x1 dari f(x) = -0.567076       iterasi ke - 3    error
= 0.000231
x1 dari f(x) = -0.567072       iterasi ke - 4    error
= 0.000003
Solusi aproksimasi terdekat dari x2 = -0.567072
error = 0.000003   iterasi ke-4
```

1.6.4 Analisis Program

Pada percobaan terakhir praktikan menentukan nilai dari $f(x) = xe^{-x} + \cos 2x$. PPada 3 metode ketika praktikan menginput nilai batas selain -1 sampai 1, program tidak memberikan output. Hal ini mengindikasikan bahwa pada selang interval 0 sampai 1, 1 sampai 2, tidak ada titik potong dengan sumbu x. Sehingga program tidak memberikan output.

Selanjutnya nilai dari akar persamaan pada metode bisection berbeda dengan metode fixed position dan newton Raphson yang berada pada

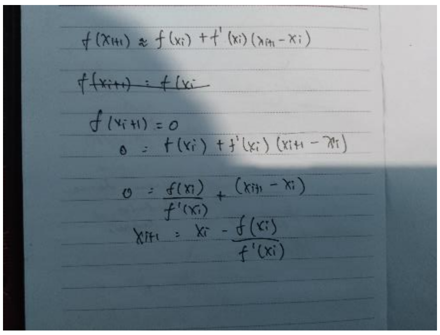
0.567072, yaitu sebesar 0.075. Akan tetapi nilai eksak dari akar persamaan tersebut juga berada pada nilai x = 0.427, maknanya ketiga metode belum dapat mencapai akprosimasi terdekat dari fungsi $xe^{-x} + \cos 2x$.

2. Tugas Akhir

1. Turunkan Persamaan 2.5 mengunakan pendekatan deret Taylor.
2. Hitunglah berapa iterasi yang diperlukan untuk mencapai konvergensi dalam mencari solusi persamaan $x^2 - 3x - 10 = 0$menggunakan metode *Bisection* dengan tingkat kesalahan yang diperbolehkan bernilai 10^{-3} serta nilai $a = 1$dan $b = 3{,}5$. Bandingkan dengan hasil yang didapat jika dikerjakan secara analitik menggunakan Persamaan 2.1
3. Buatlah program dengan menggunakan bahasa Python untuk menentukan solusi dari persamaan non-linier pada bagian Percobaan menggunakan metode Secant dan Regula Falsi.
4. Buatlah tabel perbandingan untuk metode *Bisection,* iterasi titik tetap, Newton-Raphson, Secant, dan Regula Falsidalam menentukan solusi persamaan $e^{-2x} - 4x = 0$ dengan parameter yang sama (*error*, toleransi, batas/tebakan awal, dan hasilnya apakah konvergen/divergen)untuk setiap metode. Sertakan pula jumlah langkah/jumlah iterasiyang terjadi saat melakukan proses perhitungan. Kemudian buatlah analisa berdasarkan tabel tersebut.

Jawab :

1.



2. Terdapat nol perulangan pada interval 1 sampai 3.5, karena pada bisection methods interval tersebut memiliki hasil kali f(a)*f(b) > 0, sehingga nilai interval harus diubah. Jika interval diubah sedemikian hingga, proses pengulangan hingga 10^{-3} akan dicapai pada iterasi ke-11. Yaitu nilai akar x = 4.9980. dengan error = 0.0019. Berikut perhitungan analitik

$$f(x) = x^2 - 3x - 10$$

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$$

$$D = b^2 - 4ac$$
$$= 9 - 4 \cdot 1 \cdot (-10)$$
$$= 49$$

$$x_{1,2} = \frac{3 \pm \sqrt{49}}{2 \cdot 1} = \frac{3 \pm 7}{2}$$

$$x_1 = \frac{3-7}{2} = -2$$

$$x_2 = \frac{10}{2} = 5 \therefore$$

3. Program Secant dan Regula Falsi

3.1 Program Secant

```
from math import *
import numpy as np
a = float(input("Masukkan batas awal kembali  : "))
b = float(input("Masukkan batas akhir kembali : "))
e = 1.0
def secn(a,b,f,e,n):
    xOri = a
    xInitial = b
    e = 1.0
    if f(xOri)*f(xInitial)>0:
        while True:
            xA = float(input("Masukkan batas awal 
kembali  : "))
            xB = float(input("Masukkan batas akhir 
kembali : "))
            if f(xOri)*f(xInitial)<0:
                break
        return None
    n = 0

    while e > 0.0001:
        xMid =  xInitial - f(xInitial)*(xInitial-
xOri)/(f(xInitial)-f(xOri))
        n += 1
        e = xInitial - xMid
        if f(xOri)*f(xInitial)<0:
            xInitial = xMid
        else:
            xOri = xMid
        print(f"Hasil aproksimasi x = {xMid:.6f}\t 
error = {abs(e):.6f}\t iterasi ke-{n}")
    print(f"Hasil akar aproks = {xMid:.6f} \t Iterasi 
ke-{n} \t error = {abs(e):.6f}")
    return xMid


f = lambda x: x**2-3*x-10 # ubah dengan fungsi yang 
diinginkan
x1 = secn(a,b,f,e,0)
```

Output :

Hasil aproksimasi x = 4.857143   error = 1.142857   iterasi ke-1

Hasil aproksimasi x = 5.024390   error = 0.167247   iterasi ke-2

Hasil akar aproks = 5.024390   Iterasi ke-2   error = 0.167247

3.2 Program Regula Falsi

```
from math import *
import numpy as np
a = float(input("Masukkan batas awal kembali  : "))
b = float(input("Masukkan batas akhir kembali : "))
e = 1.0
def regul(a,b,f,e,n):
    xOri = a
    xInitial = b
    e = 1.0
    if f(xOri)*f(xInitial)>0:
        while True:
            xA = float(input("Masukkan batas awal 
kembali  : "))
            xB = float(input("Masukkan batas akhir 
kembali : "))
            if f(xOri)*f(xInitial)<0:
                break
        return None
    n = 0

    while e > 0.0001:
        xMid =  (f(xInitial)*(xOri)-
f(xOri)*xInitial)/(f(xInitial)-f(xOri))
        n += 1
        e = xInitial - xMid
        if f(xOri)*f(xInitial)<0:
            xInitial = xMid
        else:
            xOri = xMid
        print(f"Hasil aproksimasi x = {xMid:.6f}\t 
error = {abs(e):.6f}\t iterasi ke-{n}")
    print(f"Hasil akar aproks = {xMid:.6f} \t Iterasi 
ke-{n} \t error = {abs(e):.6f}")
    return xMid


f = lambda x: x**2-3*x-10 #Ubah dengan fungsi yg 
diinginkan
x1 = regul(a,b,f,e,0)
```

Output:

Hasil aproksimasi x = 4.857143   error = 1.142857   iterasi ke-1

Hasil aproksimasi x = 5.024390   error = 0.167247   iterasi ke-2

Hasil akar aproks = 5.024390   Iterasi ke-2   error = 0.167247

4.

Metode	Error	batas	tebakan	hasil	iterasi	div/conv
Bisection	0.000061	0<x<0.5	none	0.124939	13	convergen
Fixed Position	0.00008	none	0.5	0.175846	10	convergen
Newton-Raphson	0.000111	none	0.5	0.175867	3	convergen
Secant	0.000032	0<x<0.5	none	0.175869	4	convergen
Regula-Falsi	0.000032	0<x<0.5	none	0.175869	4	convergen

Analisis : Dari tabel diatas metode secant dan regula falsi memiliki tingkat keakuratan paling besar. Hal ini diindikasikan oleh nilai error yang paling kecil yaitu sebesar 0.000032. Kedua metode tersebut cukup efektif karena hanya memerlukan 4 kali pengulangan dalam mencari nilai yang diinginkan.

Sedangkan metode bisection meski memiliki error yang realtif kecil, akan tetapi hasil dari perhitungannya masih jauh dari nilai eksak yaitu 0.175866. Sehingga metode ini kurang optimal dalam melakukan penentuan nilai dari akar suatu fungsi $e^{-2x} - 4x$.

Selanjutnya metode Newton Rapson merupakan metode yang memiliki hasil perhitungan paling mendekati nilai eksaknya yaitu 0.175867, eksaknya 0.175867. Metode ini juga hanya perlu membutuhkan pengulangan sebanyak 3 kali. Sehingga metode ini merupakan metode yang paling efektif dan efisien.

Sedangkan metode Fixed Position, memiliki nilai hasil pendekatan jauh lebih baik dibandingkan dengan bisection. Hal ini juga didukung dengan proses iterasi yang semakin sedikit. Metode ini cukup efektif dalam mencari nilai akar persamaan $e^{-2x} - 4x$ akan tetapi kurang efisien.

DAFTAR PUSTAKA

C. Chapra, S., & P. Canale, R. (2015). *Numerical Methods in Engineering* (7th ed.). McGraw-Hill Education.

Jack. (n.d.). *Looking for Taylor series expansion of*. math.stackexchange.com. Retrieved March 17, 2024, from https://math.stackexchange.com/q/1443789

Kiusalaas, J. (2009). *NUMERICAL METHODS IN ENGINEERING* (2nd ed.). New York: Cambridge University Press.

Munir, Rinaldi. (2017). *Metode Numerik*. Bandung: Penerbit INFORMATIKA.