Modul_3_PMN

September 12, 2022

0.1 # Modul 3

0.2 Pencarian Akar Sebarang Fungsi

Pada bab ini akan mulai dibahas mengenai suatu metode numerik yang didasarkan pada matematika diskret dan algoritma dan pemrograman yang telah dibahas pada bab yang telah lalu. Lebih lanjut akan disajikan pula suatu permasalahan fisis yang dapat diselesaikan melalui metode numerik yang dimaksud.

Topik yang akan disajikan pada bab ini adalah permasalahan akar fungsi yang merupakan permasalahan yang cukup penting dalam matematika. Suatu nilai disebut sebagai akar dari sebuah fungsi jika nilai tersebut menghasilkan luaran 0 (nol) saat dimasukan ke dalam fungsi yang dimaksud. Oleh karenanya permasalahan akar ini seringkali disebut sebagai pencarian titik nol. Bentuk umum masalah pencarian akar atau pencarian titik nol bagi sebarang fungsi $f(x)$ dapat diungkapkan dalam bentuk

$$f(x) = 0 \tag{1}$$

Nilai-nilai x yang memenuhi pers (1) di atas disebut sebagai akar-akar atau titik-titik nol bagi fungsi $f(x)$ tersebut.

Untuk suatu fungsi linear maka nilai akar (titik nol) baginya dapat diperoleh dengan mudah, oleh karenanya dalam bab ini akan dibahas mengenai pencarian akar pada fungsi-fungsi yang tak linier. Banyak contoh fungsi yang tak linier dalam metematik diantaranya adalah fungsi polinom dan fungsi-fungsi trigonometrik.

Salah satu terapan dari pencarian akar fungsi adalah pada masalah optimasi. Sebagaimana kita ketahui bahwa jika kita mengetahui suatu fungsi akan bernilai ekstrimum jika turunan pertamannya bernilai nol. Memanfaatkan sifat tersebut, ditambah dengan turunan keduanya bernilai negatif, maka dapat diperoleh kondisi optimum dari fungsi tersebut. Terlihat hubungan yang cukup jelas antara kondisi optimum dan nilai akar dari suatu fungsi. Untuk mencari akar dari beberapa jenis fungsi-fungsi nonlinear terdapat beberapa metode numerik dapat digunakan untuk memperoleh nilai akarnya, berikut adalah beberapa metode yang dimaksud.

0.3 1. Metode Bisection

Metode bisection merupakan metode paling sederhana bagi penyelesaian akar fungsi tak linear pada suatu interval yang diketahui. Kelebihan dari metode ini adalah bisa digunakan bagi sembarang fungsi termasuk pada suatu fungsi-fungsi yang tidak bisa diselesaikan secara analitik. Berdasarkan namanya seseungguhnya kita bisa menebak secara intuitif maksud dari metode ini, yaitu jika pada

suatu interval tertentu terdapat suatu akar dari fungsi maka interval tersebut dibagi menjadi dua interval baru. Lalu dapat dipastikan diantara kedua interval yang terbentuk tersebut terdapat satu interval yang memuat akar (titik nol) fungsi.

Pertama asumsikan bahwa pada interval $x = a$ dan $x = c$ atau dapat dituliskan $[a, c]$, terdapat satu buah akar bagi sebuah fungsi. Kemudian metode Bisection bekerja berdasarkan fakta bahwa tanda pada dua sisi yaitu kiri dan kanan titik nol adalah berlawanan, yaitu $f(a)$ positif dan $f(c)$ negatif. Maka langkah dalam metode Bisection untuk memdekati nilai akar adalah sebagai berikut:

1. Pertama membagi interval menjadi dua (Bisect) yaitu $[a, b]$ dan $[b, c]$ dimana $b = (a + c)/2$.
2. Mencari interval yang masih mengandung akar fungsi dengan cara melakukan perkalian antara $f(a)f(b)$ dan $f(b)f(c)$. Jika $f(a)f(b) \leq 0$ maka interval $[a, b]$ mengandung akar fungsi. (lihat gambar)
3. Interval $[a, b]$ dibagi menjadi dua lagi "di-Bisect" dan prosedur pencarian interval yang mengandung akar fungsi dilakukan secara berulang.
4. Pada setiap langkah titik tengah interval akan digunakan sebagai nilai pendekatan bagi akar fungsi yang dimaksud/dicari.

Setelah n langkah perulangan maka akan diperoleh

$$\frac{c - a}{2^n} \tag{1}$$

Nilai ini dapat digunakan untuk menentukan batas toleransi bagi program untuk melakukan iterasi sampai mencapai interval terkecil. Jika diberikan toleransi ϵ maka berlaku

$$\frac{c - a}{2^n} \tag{2}$$

atau dapat dituliskan sebagai

$$n \geq \ln \frac{c - a}{\epsilon} \tag{3}$$

Berikut ini adalah kode program yang berdasarkan pada metode Bisection untuk menyelesaikan permasalahan persamaan $x^2 - 2x + 1$ dengan bahasa python.

[1]:
```
from math import pi
import numpy as np
def bisection(f,a,b,N):
    an = a
    bn = b
    for n in range(1,N+1):
        mn = (an + bn)/2
        fmn = f(mn)
        if f(an)*fmn < 0:
            an = an
            bn = mn
        elif f(bn)*fmn < 0:
            an = mn
            bn = bn
        elif fmn == 0:
            print("Diperoleh solusi eksak")
```

```
            return mn
        else:
            print("Metode gagal")
            return None
    return (an + bn)/2
```

[2]:
```
f = lambda x: x**2 - 2*x + 1
nilai_x = bisection(f,0,2,25)
print(nilai_x)
```

```
Diperoleh solusi eksak
1.0
```

0.3.1 Hal yang perlu diperhatikan pada Metode Bisection

Sudah ditunjukkan bahwa metode Bisection berhasil digunakan untuk menghitung nilai akar fungsi $x^2 - 2x + 1$. Salah satu lagkah yang dilakukan adalah memberikan rentang di mana akar fungsi dicari. Pennentuan rentang ini dapat diartika dua hal. Pertama, pada rentang tersebutlah akan dicari keberadaan nilai akar. Kedua, rentang tersebut merupakan perkiraan di mana akar yang akan dicari berada. Oleh karena itu perlu diperhatikan pemberian rentang dimana akar akan dicari.

Sebagai contoh pentingnya rentang pencarian akar, akan digunakan fungsi `sin(x)`. Dengan menggunakan fungsi tersebut dapat diuji metode Bisection dengan memberikan rentang tebakan yang berbeda sebagai berikut:

[3]:
```
from math import pi
import numpy as np
def bisection(f,a,b,N):
    an = a
    bn = b
    for n in range(1,N+1):
        mn = (an + bn)/2
        fmn = f(mn)
        if f(an)*fmn < 0:
            an = an
            bn = mn
        elif f(bn)*fmn < 0:
            an = mn
            bn = bn
        elif fmn == 0:
            print("Diperoleh solusi eksak")
            return mn
        else:
            print("Metode gagal")
            return None
    return (an + bn)/2
```

[4]:
```
f = lambda x: np.sin(x)
approx_phi = bisection(f,0,1*pi,25)
print(approx_phi)
```

```
Metode gagal
None
```

Dari uji coba ini dapat ditunjukkan jika dalam rentang yang diberikan tidak terdapat akar maka metode akan dianggap gagal, sekaligus dapat pula berarti bahwa hasilnya adalah dalam rentang tersebut dapat dibuktikan tidak terdapat akar. Sedangkan dengan meberi tebakan yang berbeda sebagai berikut

[5]:
```
f = lambda x: np.sin(x)
approx_phi = bisection(f,0,2*pi,25)
print(approx_phi)
```

```
3.14159274721655
```

Selain itu perlu diperhatikan pula jika fungsi yang dicari akarnya mengandung beberapa akar, yang berarti bahwa perlu diberikan beberapa rentang untuk menghasilkan semua akar dari fungsi yang dianalisa.

0.3.2 Seputar Penentuan Masukan Awal atau Nilai Coba

Beberapa algoritma atau urutan penyelesaian suatu masalah dalam proses komputasi memerlukan adanya masukan awal atau nilai coba agar algoritma tersebut dapat bekerja. Sebagai contoh, dalam penentuan nilai suatu akar bagi sebarang fungsi non linear maka diperlukan dua nilai coba agar metode *Bisection* dapat memberikan nilai akar yang dicari.

Pertanyaan yang seringkali muncul terkait hal itu adalah: "Bagaimana dapat memberikan nilai coba sebagai masukan awal bagi penentuan nilai akar, mengingat nilai akar itulah yang justru ingin diketahui?". Untuk memberikan jawaban atas pertanyaan tersebut, berikut akan disampaikan tiga dari banyak mekanisme yang mungkin dapat diambil.

0.3.3 a. Mekanisme Intuisi Fisika

Beberapa proses komputasi dari suatu ungkapan matematika biasanya tidak berdiri sendiri, tetapi mewakili suatu sistem fisika tertentu. Sebagai contoh, akar yang akan dicari dari suatu fungsi tertentu boleh jadi mewakili nilai tenaga dari suatu sistem kuantum, yaitu nilai tenaga ionisasi atom Helium. Dalam konteks ini, pencarian nilai tenaga ionisasi atom Helium akan dapat diperkirakan kisaran nilai tenaganya berdasarkan informasi nilai tenaga dari sistem kuantum yang lebih sederhana namun memiliki kemiripan dengan sistem atom Helium yaitu sistem atom Hidrogen. Karena lebih sederhana maka penyelesaian sistem atonm Hidrogen relatif lebih mudah dibanding atom Helium, sehingga informasi nilai tenaga pada atom Hidrogen dapat digunakan untuk memperkirakan kisaran nilai tenaga yang diperlukan sebagai nilai coba pada proses komputasi.

0.3.4 b. Mekanisme Aproksimasi Matematika

Beberapa ungkapan matematika yang nampak rumit kadang dapat dipandang dalam bentuk ungkapan yang lebih sederhana. Hal itu dapat terjadi dalam keadaan khusus ketika memenuhi suatu

mekanisme aproksimasi tertentu antara lain:

- Satu atau beberapa suku dalam ungkapan tersebut bernilai jauh lebih kecil dibanding suku lainnya sehingga suku tersebut untuk sementara dapat diabaikan. Pengaruh pengabaian suku tersebut kadang menyebabkan penyelesaian bagi ungkapan tersebut menjadi lebih mudah dibanding ungkapan semula. Penyelesaian dari ungkapan yang belih sederhana tersebut akan dapat digunakan sebagai masukan awal atau nilai coba untuk proses komputasi bagi ungkapan semula.
- Nilai asimtotik, yaitu nilai dalam keadaan ekstrem yang sangat besar atau sangat kecil, bagi nilai akar kadang dapat digunakan untuk memperoleh sifat atau kisaran bagi nilai akar lainnya. Kisaran nilai tersebut akan dapat digunakan sebagai masukan awal atau nilai coba untuk proses komputasi bagi nilai akar yang lebih teliti.

0.3.5 c. Mekanisme Visual

Kemampuan untuk melakukan evaluasi fungsi dan menampilkan plot fungsi akan dapat digunakan sebagai cara praktis dalam menentukan masukan awal atau nilai coba proses komputasi. Sebagai gambaran, berikut akan ditunjukkan proses komputasi dengan metode *Bisection* untuk memperoleh beberapa akar dari polinomial *Hermite* pada orde ke n.

Algoritma untuk masalah tersebut adalah seperti berikut:

1. Lakukan evaluasi fungsi bagi polinomial *Hermite* pada orde tertentu n pada rentang peubah x yang dipilih.
2. Lakukan plot bagi fungsi melalui metode evaluasi fungsi langkah ke 1.
3. Cermati secara visual pada plot fungsi langkah ke 2 dan pilih dua nilai di sekitar suatu titik potong fungsi terhadap absis.
4. Gunakan dua nilai di sekitar titik potong tersebut sebagai dua nilai awal metode *Bisection* untuk memperoleh nilai akar tertentu.
5. Ulangi langkah 3 dan 4 untuk memperoleh nilai akar lainnya.

Berikut akan ditunjukkan implementasi algoritma di atas untuk memperoleh akar-akar dari polinimial *Hermite* orde 5.

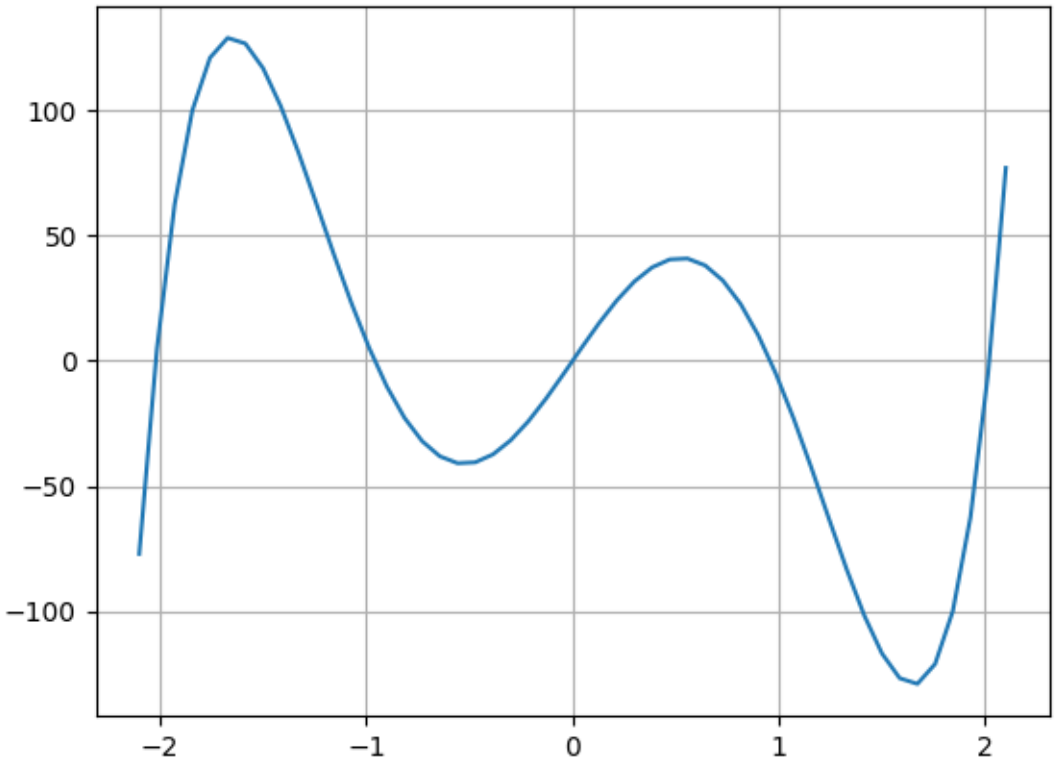
[6]:
```
import numpy as np
from matplotlib import pyplot as plt

def hermiten(n,x):
    yrekur0 = 1.0
    yrekur1 = 2.0*x
    for i in range (1,n):
        yrekur = 2.0*x*yrekur1-2.0*i*yrekur0
        yrekur0 = yrekur1
        yrekur1 = yrekur

    return yrekur
```

[7]:
```
n = 5
x = np.linspace(-2.1, 2.1, 50)
yrekur = hermiten(n,x)
```

```
plt.plot(x,yrekur)
plt.grid()
plt.show()
```



[8]:
```
from math import pi
import numpy as np
def bisection(f,a,b,N):
    an = a
    bn = b
    for n in range(1,N+1):
        mn = (an + bn)/2
        fmn = f(mn)
        if f(an)*fmn < 0:
            an = an
            bn = mn
        elif f(bn)*fmn < 0:
            an = mn
            bn = bn
        elif fmn == 0:
            print("Diperoleh solusi eksak")
            return mn
        else:
```

```
            print("Metode gagal")
            return None
    return (an + bn)/2
```

[9]:
```
f = lambda x: hermiten(n,x)
nilai_x = bisection(f,0.5,1.5,25)
print(nilai_x)
```

```
0.9585724622011185
```

Hasil di atas dapat dibandingkan hingga minimal 7 angka di belakang koma dengan hasil berdasarkan Tabel polinomial *Hermite* pada orde 5 yang memiliki akar dengan nilai 0.958572464614 dan 2.020182870456.

0.4 2. Metode Newton-Raphson

Pada bagian sebelumnya telah dijelaskan mengenai suatu metode yang sangat sederhana penyelesaian numerik bagi permasalahan pencarian akar suatu fungsi dengan metode Bisection. Pada bagian ini akan dijelaskan metode yang lain yaitu metode Newton atau yang dikenal pula dengan metode Newton-Raphson. Kelebihan dari metode ini adalah dapat digunakan untuk menyelesaikan permasalahan akar kompleks dan bahkan bisa diterapkan pada persamaan-persamaan nonlinear secara simultan.

Bentuk umum masalah pencarian akar (*roots finding*) atau pencarian titik nol (*zeros finding*) dapat dinyatakan dalam bentuk ungkapan

$$f(x) = 0 \tag{1}$$

dengan x adalah akar-akar atau titik-titik nol yang akan dicari.

Metode Newton-Raphson untuk masalah pencarian akar dapat diperoleh dari deret Taylor sebagai berikut

$$f(x) = f(a) + hf'(a) + \frac{h^2}{2}f''(a) + \cdots + \frac{h^m}{m!}f^{(m)}(a) + \cdots \tag{2}$$

Dimana $h = x - a$ biasa disebut sebagai ukuran langkah (*step size*) yaitu nilai yang mewakili tingkat kehalusan diskretisasi dan $f'(a)$, $f''(a)$, $f^m(a)$, adalah beturut-turut turunan pertama, kedua dan ke m dari fungsi $f(x)$ di titik $x = a$. Merujuk pada ungkapan untuk masalah pencarian akar seperti diberikan oleh pers (1) maka untuk suatu nilai tebakan akar di $x = x_0$ dan menggunakan pendekatan hingga dua suku pertama pada pers (2) maka berlaku

$$f(x) = 0 \approx f(x_0) + (x - x_0)f'(x_0)$$

Dengan menyelesaikan persamaan di atas maka dapat diperoleh

$$0 = f(x_0) + (x - x_0)f'(x_0)$$

$$\frac{f(x_0)}{f'(x_0)} + (x - x_0) = 0$$

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Ungkapan di atas dapat dipahami sebagai berikut: ketika nilai tebakan akar diberikan oleh nilai x_0 maka persamaan tersebut dapat digunakan untuk memperoleh tebakan akar yanmg lebih baik yaitu x. Apabila hasil tebakan x ternyata belum baik maka nilai x dapat diambil sebagai nilai tebakan berikutnya yaitu x_1. Seperti langkah sebelumnya, ketika nilai tebakan akar diberikan oleh nilai x_1 maka persamaan tersebut dapat digunakan untuk memperoleh tebakan akar berikutnya yang lebih baik yaitu $x = x_2$.

Langkah tersebut dapat diulang atau rekursif hingga pada langkah atau putaran ke i diharapkan akan diperoleh nilai tebakan akar yang mendekati nilai akar yang dicari. Upaya pencarian akar secara rekursif seperti ini disebut sebagai metode Newton-Raphson dengan bentuk persamaan

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})} \tag{3}$$

Dengan menggunakan pers (3) di atas maka urutan langkah atau Algoritma dalam pencarian akar menggunakan metode Newton-Rapshon adalah sebagai berikut: 1. Tentukan bentuk fungsi yang akan dicari nilai akarnya. 2. Tentukan turunan pertama dari fungsi tersebut. 3. Berikan nilai tebakan awal dari akar. 4. Hitung nilai akar menggunakan pers (3) di atas sampai batas toleransi yang ingin dicapai dan/atau iterasi maksimal yang diinginkan.

Suatu hal yang perlu diingat pada metode ini adalah diperlukan turunan bagi suatu fungsi yang ingin diketahui nilai akarnya menggunakan metode ini. Oleh karenanya tidak sebarang fungsi dapat diselesaikan, melainkan hanya fungsi yang mempunyai turunan. Selain itu diperlukan nilai tebakan awal yang sesuai agar proses pencarian akar dapat dilakukan. Terkait tebakan awal tersebut teknik yang sama dengan pemberian rentang pada metode Bisection yaitu menggunakan intuisi fisis, menggunakan aproksimasi matematis dan visual melalui plotting fungsi, dapat diterapkan pada metode Newton-Rapshon.

Berikut adalah kode sumber yang merupakan contoh implementasi dari metode Newton-Raphson untuk mencari akar dari persamaan $x^2 - 2x + 1$ yang mempunyai turunan pertama $2x - 2$.

[10]:
```
# fungsi yang ingin dicari akarnya
def fung(x):
    return x**2 - 4*x + 3

#turunan pertama fungsi
def dfung(x):
    return 2*x - 4
```

[11]:
```
def NewtonRaphson(x0,tol,imak):
    i = 0
    while i < imak:
        if dfung(x0) == 0.0:
```

```
            print('Ada pembagian dengan nol')
            break

        i = i + 1
        x1 = x0 - fung(x0)/dfung(x0)
        delta = x1 - x0

        #selisih antara dua variiabel tetap positif
        if (delta < 0):
            delta = - delta

        print('Hasil iterasi ke-%d, x1 = %0.6f and f(x1) = %0.6f' % (i, x1,␣
↪fung(x1)))
        x0 = x1

        if  i >= imak:
            print('\nNilai akar adalah: %0.8f' % x1)

        #penghentian perhitungan jika selisih sudah lebih kecil dari toleransi
        if  delta <= tol:
            print('\nNilai akar adalah: %0.8f' % x1)
            break
```

[13]:
```
# bagian input
x0 = input('Berikan nilai awal X0: ')
tol = input('Berikan nilai toleransi: ')
imak = input('Iterasi maksimal: ')

# konfersi x0 dan tol kedalam float
x0 = float(x0)
tol = float(tol)

# konversi imak menjadi integer
imak = int(imak)

# Pemanggilan Newton-Rapshon
NewtonRaphson(x0,tol,imak)
```

```
Berikan nilai awal X0: 0.1
Berikan nilai toleransi: 0.001
Iterasi maksimal: 10
Hasil iterasi ke-1, x1 = 0.786842 and f(x1) = 0.471752
Hasil iterasi ke-2, x1 = 0.981274 and f(x1) = 0.037804
Hasil iterasi ke-3, x1 = 0.999828 and f(x1) = 0.000344
Hasil iterasi ke-4, x1 = 1.000000 and f(x1) = 0.000000

Nilai akar adalah: 0.99999999
```

0.4.1 Laju Konvergensi

Laju konvergensi merupakan istilah bagi kecepatan suatu metode hingga mendapatkan hasil yang dapat diterima. Metode Newton-Raphson memiliki laju konvergensi orde dua, sedangkan metode Bisection memiliki laju konvergensi orde satu. Secara mudah, laju konvergensi orde dua memiliki sifat bahwa saat suatu iterasi memiliki ralat sebesar 0.01 maka ralat pada iterasi berikutnya menjadi $(0.01)^2 = 0.0001$. Oleh karena itu, pada umumnya metode Newton-Raphson akan lebih cepat memperoleh nilai akar dibanding metode Bisection.

Berikut gambaran laju konvergensi metode Bisection ketika dibandingkan dengan metode Newton-Raphson.

[14]:
```
def Bisection(a,b,tol,imak):
    i = 0
    an = a
    bn = b
    while i < imak:
        i = i + 1
        mn = (an + bn)/2.0
        fmn = fung(mn)
        fan = fung(an)
        fbn = fung(bn)
        if fan*fmn < 0:
            an = an
            bn = mn
        elif fbn*fmn < 0:
            an = mn
            bn = bn
        elif fmn == 0:
            print("Diperoleh solusi eksak")
            return mn
        else:
            print("Metode gagal")
            return None

        delta = (bn - an)/bn

        #selisih antara dua variiabel tetap positif
        if (delta < 0):
            delta = - delta

        print('Hasil iterasi ke-%d, mn = %0.6f and f(mn) = %0.6f' % (i, mn,␣
↪fmn))

        if  i >= imak:
            print('\nNilai akar adalah: %0.8f' % mn)

        #penghentian perhitungan jika selisih sudah lebih kecil dari toleransi
```

```
        if  delta <= tol:
            print('\nNilai akar adalah: %0.8f' % mn)
            break
```

[15]:
```
# bagian input
a = input('Berikan nilai awal ke 1: ')
b = input('Berikan nilai awal ke 2: ')
tol = input('Berikan nilai toleransi: ')
imak = input('Iterasi maksimal: ')

# konfersi a, b dan tol kedalam float
a = float(a)
b = float(b)
tol = float(tol)

# konversi imak menjadi integer
imak = int(imak)

# Pemanggilan Newton-Rapshon
Bisection(a,b,tol,imak)
```

```
Berikan nilai awal ke 1: 0.1
Berikan nilai awal ke 2: 1.1
Berikan nilai toleransi: 0.001
Iterasi maksimal: 10
Hasil iterasi ke-1, mn = 0.600000 and f(mn) = 0.960000
Hasil iterasi ke-2, mn = 0.850000 and f(mn) = 0.322500
Hasil iterasi ke-3, mn = 0.975000 and f(mn) = 0.050625
Hasil iterasi ke-4, mn = 1.037500 and f(mn) = -0.073594
Hasil iterasi ke-5, mn = 1.006250 and f(mn) = -0.012461
Hasil iterasi ke-6, mn = 0.990625 and f(mn) = 0.018838
Hasil iterasi ke-7, mn = 0.998438 and f(mn) = 0.003127
Hasil iterasi ke-8, mn = 1.002344 and f(mn) = -0.004682
Hasil iterasi ke-9, mn = 1.000391 and f(mn) = -0.000781
Hasil iterasi ke-10, mn = 0.999414 and f(mn) = 0.001172

Nilai akar adalah: 0.99941406

Nilai akar adalah: 0.99941406
```

0.5 Tugas

Pencarian akar-akar atau titik-titik nol dari beberapa fungsi khas (*special functions*) banyak dilakukan di beberapa prosedur penyelesaian permasalahan fisika dan matematika. Sebagai contoh, pencarian titik-titik nol bagi salah satu fungsi khas, yaitu Polinomial Hermite $H_n(x)$, akan muncul pada beberapa permasalahan sistem kuantum serta pencarian peubah bebas dalam masalah

kuadratur numerik. Dalam hal ini bentuk masalah pencarian akar-akar bagi fungsi akan berbentuk:

$$H_n(x) = 0$$

dengan n adalah orde dari polinomial Hermite.

Metode evaluasi polinomial Hermite $H_n(x)$ menggunakan ungkapan deret maupun kaitan rekurensi dapat merujuk pada materi yang disampaikan di Modul 2 Praktikum Metode Numerik. Ungkapan bagi turunan pertama polinomial Hermite dapat dinyatakan dalam bentuk

$$\frac{dH_n(x)}{dx} = H_n^{'}(x) = 2nH_{n-1}(x)$$

1. Metode Bisection > Tentukan keseluruhan akar-akar dari polinomial $H_n(x)$ pada orde 10 yaitu $n = 10$ dengan menggunakan metode Bisection. 2. Metode Newton-Raphson > Tentukan keseluruhan akar-akar dari polinomial $H_n(x)$ pada orde 10 yaitu $n = 10$ dengan menggunakan metode Newton-Raphsonn.

[ ]: