Akar Persamaan Nonlinear

Fadjar Fathurrahman

1 *Chapra Contoh 5.3*

Kita ingin mencari nilai dari c dari persamaan

$$v(t) = \frac{gm}{c}(1 - e^{-(c/m)t})$$

sehingga untuk $v(t = 10) = 40$, dengan $g = 9.81$ dan $m = 68.1$. Nilai c dapat dicari sebagai akar dari persamaan:

$$f(c) \equiv \frac{gm}{c}(1 - e^{-(c/m)t}) - v(t)$$

Kita akan menggunakan metode *bisection* untuk mengaproksimasi akar atau solusi dari persamaan $f(c) = 0$. Diberikan dua nilai x_l (lower) dan x_u (upper), di mana $x_u > x_l$ dan $f(c = x_l)f(c = x_u) < 0$, metode *bisection* memberikan aproksimasi akar x_r sebagai berikut:

$$x_r = \frac{x_l + x_u}{2} \tag{1}$$

Kode Python berikut ini mengilustrasikan penggunaan

```
import numpy as np

m = 68.1 # mass, kg
v = 40.0 # velocity, m/s
t = 10.0 # time, s
g = 9.81

def f(c):
    return ... # lengkapi

x_true = 14.8011 # from the text

# Initial guess
xl = 12.0
xu = 16.0

# First iteration
print("\n1st iteration: ")
print("xl = %f, xu = %f" % (xl, xu))
print("f(xl) = %f, f(xu) = %f" % (f(xl), f(xu)))
xr = # ... lengkapi
print("xr = ", xr)
ε_t = abs(xr - x_true)/x_true*100 # error in percent
print("ε_t = %.1f %%" % ε_t)
```

```
# Determine new xr should replace xu or xl (make new
↪  interval)
if f(xl)*f(xr) < 0:
    xu = xr
else:
    xl = xr

# Second iteration
print("\n2nd iteration: ")
print("xl = %f, xu = %f" % (xl, xu))
print("f(xl) = %f, f(xu) = %f" % (f(xl), f(xu)))
xr = (xl + xu)/2
print("xr = ", xr)
ε_t = abs(xr - x_true)/x_true*100 # error in percent
print("ε_t = %.1f %%" % ε_t)

if f(xl)*f(xr) < 0:
    xu = xr
else:
    xl = xr

# Third iteration
# ... teruskan jika diperlukan

# Jika xr merupakan akar dari f, maka f(xr) harus mendekati
↪  0
# Tampilkan hasil dari f(xr) di sini
```

Lengkapi kode untuk ilustrasi penggunakan bisection tersebut. Lakukan iterasi sampai suatu kriteria tertentu yang Anda tentukan. Anda boleh menggunakan loop untuk menghindari pengulangan kode.

2 *Chapra Contoh 5.6*

Kita ingin menghitung akar dari persamaan:

$$f(x) = x^{10} - 1 \tag{2}$$

dengan menggunakan metode *bisection* dan *regula falsi*.

Soal 1. Aplikasikan metode *bisection* dan *regula falsi* untuk mendapatkan aproksimasi akar dari persamaan (2)

3 *Chapra Contoh 6.1*

Pada soal ini, kita ingin mencari akar dari $f(x) = e^{-x} - x$ dengan menggunakan metode iterasi *fixed-point*. Metode ini juga dikenal dengan nama iterasi satu titik atau substitusi berurutan. Metode ini

bekerja dengan cara mengubah persamaan awal $f(x) = 0$ menjadi $x = g(x)$. Untuk contoh $f(x)$ yang diberikan kita memiliki:

$$x = e^{-x}$$

Dimulai dari tebakan awal $x_0 = 0$, kita dapat melakukan iterasi *fixed-point* dengan menggunakan kode Python berikut.

```
# Simple fixed-point iteration

import numpy as np # math module also can be used

def g(x):
    return .... # lengkapi

# Initial guess
x = 0.0
x_true = 0.56714329

print()
print("Initial point: x = ", x)
print()
for i in range(1,11): # 10 iterasi
    xnew = .... # lengkapi
    ε_a = np.abs( (xnew - x)/xnew )*100 # in percent
    ε_t = np.abs( (x_true - xnew)/x_true )*100
    print("%3d %10.6f %10.2f%% %10.2f%%" % (i, xnew, ε_a,
    ↪  ε_t))
    x = xnew
```

Soal 2. Lengkapi program Python untuk metode *fixed-point* di atas. Implementasikan program Anda sehingga dapat melakukan iterasi sampai nilai kesalahan menjadi lebih kecil dari suatu nilai tertentu yang diberikan. Anda dapat menggunakan loop `while` atau loop `for` dengan jumlah iterasi maksimum tertentu.

4 *Chapra Contoh 6.3*

Pada soal ini, kita akan mencari akar persamaan $f(x) = e^{-x} - x$ dengan menggunakan metode Newton-Raphson. Diberikan suatu nilai tebakan akar awal x_0, metode ini akan memberikan skema iterasi berikut:

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

Berikut ini adalah contoh implementasi metode Newton-Raphson untuk mencari akar dari $f(x)$:

```
import ... # lengkapi

def f(x): # fungsi f(x)
    return ...  # lengkapi

def df(x): # turunan pertama dari f(x)
    return ... # lengkapi

x = 0.0 # tebakan akar awal
for i in range(1,6):
    xnew = ... # lengkapi
    fxnew = f(xnew)
    print("%3d %18.10f %18.10e" % (i, xnew, fxnew))
    x = xnew
```

Soal 3. Lengkapi program Python untuk metode Newton-Raphson di atas. Implementasikan program Anda sehingga dapat melakukan iterasi sampai nilai kesalahan menjadi lebih kecil dari suatu nilai tertentu yang diberikan. Anda dapat menggunakan loop `while` atau loop `for` dengan jumlah iterasi maksimum tertentu.

5 *Chapra Contoh 6.6*

Salah satu kekurangan dari metode Newton-Raphson adalah perlunya menghitung turunan pertama dari fungsi yang ingin dicari akarnya. Hal ini biasanya tidak dapat Metode *secant* memiliki bentuk iterasi yang mirip dengan metode Newton-Raphson, namun turunan fungsi diaproksimasi dengan menggunakan:

$$f'(x_i) \approx \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

sehingga kita memperoleh bentuk iterasi sebagai berikut:

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

Metode *secant* memerlukan dua titik x_{-1} dan x_0 sebagai tebakan akar awal. Berbeda dengan metode *bracketing*, nilai fungsi $f(x_{-1})$ dan $f(x_0)$ tidak diharuskan memiliki tanda yang berbeda.

```
# ... definisi f dan df sama dengan contoh Newton-Raphson
x0 = 0.0
x1 = 1.0
for i in range(1,6):
    # approximation of derivative of f(x)
    dfx = (f(x0) - f(x1))/(x0 - x1)
    #
```

```
    xnew = ... # lengkapi
    fxnew = f(xnew)
    print("%3d %18.10f %18.10e" % (i, xnew, fxnew))
    x0 = x1
    x1 = xnew
```

Soal 4. Lengkapi program Python untuk metode *secant* di atas. Implementasikan program Anda sehingga dapat melakukan iterasi sampai nilai kesalahan menjadi lebih kecil dari suatu nilai tertentu yang diberikan. Anda dapat menggunakan loop `while` atau loop `for` dengan jumlah iterasi maksimum tertentu.

6 *Chapra Contoh 6.8*

Kita dapat melakukan modifikasi pada metode *secant* sehingga hanya memerlukan input satu tebakan akar awal. Aproksimasi turunan fungsi dihitung dengan cara memberikan perturbasi δ dari input

$$f'(x_i) \approx \frac{f(x_i + \delta) - f(x_i)}{\delta}$$

sehingga kita memperoleh bentuk iterasi sebagai berikut:

$$x_{i+1} = x_i - \frac{\delta f(x_i)}{f(x_i + \delta) - f(x_i)} \tag{5}$$

```
# ... definisi f dan df sama dengan contoh Newton-Raphson
x = 1.0
δ = 0.01
for i in range(1,6):
    # approximation of derivative of f(x)
    dfx = ... # lengkapi
    xnew = ... # lengkapi
    fxnew = f(xnew)
    print("%3d %18.10f %18.10e" % (i, xnew, fxnew))
    x = xnew
```

Soal 5. Lengkapi program Python untuk modifikasi metode metode *secant* di atas. Implementasikan program Anda sehingga dapat melakukan iterasi sampai nilai kesalahan menjadi lebih kecil dari suatu nilai tertentu yang diberikan. Anda dapat menggunakan loop `while` atau loop `for` dengan jumlah iterasi maksimum tertentu.

7 *Chapra Contoh 6.10*

Untuk kasus di mana persamaan nonlinear yang memiliki akar lebih dari satu, metode Newton-Raphson dapat mengalami kesulitan un-

tuk konvergen. Metode Newton-Raphson perlu untuk dimodifikasi sebagai berikut:

$$x_{i+1} = x_i - m\frac{f(x_i)}{f'(x_i)} \tag{6}$$

di mana m adalah multiplisitas dari akar. Alternatif lain adalah dengan mendefinisikan fungsi:

$$u(x) = \frac{f(x)}{f'(x)}$$

yang memiliki lokasi akar yang sama dengan fungsi awal $f(x)$. Dengan definisi tersebut, aplikasi metode Newton-Raphson memberikan skema iterasi sebagai berikut.

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)} \tag{7}$$

Soal 6. Buat implementasi dengan Python untuk mengimplementasikan modifikasi metode Newton-Raphson untuk akar dengan multiplisitas lebih dari satu (menggunakan persamaan (6) dan (7)) dan uji pada persamaan nonlinear $f(x) = x^3 - 5x^2 + 7x - 3$ dengan nilai tebakan akar awal $x_0 = 0$. Bandingkan hasil yang Anda dapatkan jika menggunakan metode Newton-Raphson tanpa modifikasi (persaman (3)).

8 Chapra Contoh 6.12

Tinjau suatu sistem persamaan nonlinear berikut:

$$u(x,y) = x^2 + xy - 10 = 0$$
$$v(x,y) = y + 3xy^2 - 57 = 0$$

Dengan menggunakan ekstensi dari metode Newton-Raphson untuk 2 variabel diperoleh skema iterasi berikut:

$$x_{i+1} = x_i - \frac{u_i\frac{\partial v_i}{\partial y} - v_i\frac{\partial u_i}{\partial y}}{\frac{\partial u_i}{\partial x}\frac{\partial v_i}{\partial y} - \frac{\partial u_i}{\partial y}\frac{\partial v_i}{\partial x}} \tag{8}$$

$$y_{i+1} = y_i - \frac{v_i\frac{\partial u_i}{\partial x} - u_i\frac{\partial v_i}{\partial x}}{\frac{\partial u_i}{\partial x}\frac{\partial v_i}{\partial y} - \frac{\partial u_i}{\partial y}\frac{\partial v_i}{\partial x}} \tag{9}$$

Kode berikut ini mengimplementasikan metode Newton-Raphson untuk dua variabel:

```
def u(x,y):
    return x**2 + x*y - 10

def dudx(x,y):
    return 2*x + y

def dudy(x,y):
    return x

def v(x,y):
    return ... # lengkapi

def dvdx(x,y):
    return ... # lengkapi

def dvdy(x,y):
    return ... # lengkapi

# Guess solutions
x = 1.5
y = 3.5

for i in range(1,5):
    # Jacobian matrix elements
    J11 = dudx(x,y)
    J12 = dudy(x,y)
    J21 = dvdx(x,y)
    J22 = dvdy(x,y)
    detJ = J11*J22 - J12*J21
    #
    ui = u(x,y)
    vi = v(x,y)
    # Update x
    xnew = .... # lengkapi
    ynew = .... # lengkapi
    print("x, y = %18.10f %18.10f" % (xnew, ynew))
    # TODO: Check convergence
    x = xnew
    y = ynew
```

Soal 7. Lengkapi program Python untuk metode Newton-Raphson untuk dua variabel. Implementasikan program Anda sehingga dapat melakukan iterasi sampai nilai kesalahan menjadi lebih kecil dari suatu nilai tertentu yang diberikan. Anda dapat menggunakan loop `while` atau loop `for` dengan jumlah iterasi maksimum tertentu.

Untuk variabel yang lebih dari dua, metode Newton-Raphson dapat dituliskan dengan menggunakan notasi matriks-vektor. Persamaan yang diimplementasikan adalah (lihat slide 27 oleh Pak Haris):

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{J}_i^{-1}\mathbf{F}(\mathbf{X}_i) \tag{10}$$

di mana $\mathbf{J}$ adalah matriks Jacobian:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{11}$$

Turunan parsial pada persaman (11) dievaluasi pada titik $\mathbf{X}_i$. Nilai fungsi $\mathbf{F}(\mathbf{X})$ dan $\mathbf{X}$ direpresentasikan sebagai vektor kolom:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{12}$$

dan

$$\mathbf{F}(\mathbf{X}) = \begin{bmatrix} f_1(\mathbf{X}) \\ f_2(\mathbf{X}) \\ \vdots \\ f_n(\mathbf{X}) \end{bmatrix} \tag{13}$$

Kode berikut ini adalah alternatif implementasi dari metode Newton-Raphson dengan menggunakan notasi matriks-vektor.

```
import numpy as np

def f(X):
    x = X[0]
    y = X[1]
    f1 = x**2 + x*y - 10
    f2 = ... # lengkapi
    return np.array([f1,f2])

def calc_jac(X):
    x = X[0]
    y = X[1]
    dudx = ... # lengkapi
    dudy = ... # lengkapi
    dvdx = ... # lengkapi
    dvdy = ... # lengkapi
    return np.array([
        [dudx, dudy],
        [dvdx, dvdy]
    ])
```

```
X = np.array([1.5, 3.5]) # initial guess
for i in range(1,6): # change this if needed
    fX = f(X)
    nfX = np.linalg.norm(fX) # calculate norm of f(X)
    print("X = ", X, "nfX = ", nfX)
    if nfX <= 1e-10: # stop the iteration if norm of f(X)
    ↪  become smaller than certain value
        print("Converged")
        break
    J = ... # calculate Jacobian matrix here
    invJ = np.linalg.inv(J) # calculate inverse of Jacobian
    ↪  matrix
    Xnew = X - np.matmul(invJ, fX)  # Update X
    X = np.copy(Xnew)  # replace X with Xnew
```

Soal 8. Lengkapi program Newton-Raphson di atas dan lakukan modifikasi jika diperlukan. Perhatikan bahwa pustaka Numpy digunakan untuk melakukan operasi matriks-vektor. Misalnya fungsi `np.linalg.inv` digunakan untuk menhitung invers dari matriks Jacobian dan `np.matmul` untuk melakukan operasi perkalian matriks $\mathbf{J}^{-1}$ dengan vektor kolom $\mathbf{F}(\mathbf{X})$.

9 *Menggunakan Pustaka Python*

Pada bagian ini, kita akan menggunakan beberapa pustaka SciPy yang dapat digunakan untuk mencari akar persamaan nonlinear. Beberapa fungsi tersebut dapat ditemukan pada modul `scipy.optimize`.

- `scipy.optimize.root_scalar`: untuk mencari akar persamaan nonlinear yang terdiri dari satu variabel.
- `scipy.optimize.fsolve`: untuk mencari akar dari sistem persamaan nonlinear (lebih dari satu variabel)

Modul `scipy.optimize` juga menyediakan beberapa fungsi lain seperti `optimize.bisect` yang dapat digunakan untuk mencari akar persamaan nonlinear, yang berbeda hanyalah *interface* atau *function signature*-nya.
Pada contoh berikut ini, kita akan mencari akar dari sistem persamaan nonlinear pada Chapra Contoh 6.12 dengan menggunakan `fsolve`:

```
from scipy.optimize import fsolve

def f(x_):
    # x = x_[0] and y = x_[1]
    x = x_[0]
    y = x_[1]
```

```
    u = x**2 + x*y - 10
    v = y + 3*x*y**2 - 57
    return [u, v]

root = fsolve(f, [1.0, 3.5]) # using initial guess as in the book
print(root)
```

Numpy menyediakan modul khusus untuk merepresentasikan polinomial, yaitu `numpy.polynomial`. Modul ini menyediakan banyak fungsi untuk melakukan berbagai operasi terkait polinomial. Pada contoh berikut ini, kita akan mencari akar-akar (real dan kompleks) dari polinomial:

$$f(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$$

Kode Python:

```
from numpy.polynomial import Polynomial
p = Polynomial([1.25, -3.875, 2.125, 2.75, -3.5, 1.0])
print(p.roots())
```

Silakan membaca dokumentasi berikut untuk informasi lebih lanjut.

- https://docs.scipy.org/doc/scipy/reference/optimize.html#root-finding
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html
- https://numpy.org/doc/stable/reference/routines.polynomials.package.html

Gunakan Numpy untuk menentukan akar-akar dari polinomial berikut:

- $f(x) = x^3 - x^2 + 2x - 2$
- $f(x) = 2x^4 + 6x^2 + 8$
- $f(x) = x^4 - 2x^3 + 6x^2 - 2x + 5$
- $f(x) = -2 + 6.2x - 4x^2 + 0.7x^3$
- $f(x) = 9.34 - 21.97x + 16.3x^2 - 3.704x^3$
- $f(x) = x^4 - 2x^3 + 6x^2 - 2x + 5$

10 Soal Tambahan

Soal 9. Chapra Latihan 5.18 Konsentrasi jenuh dari oksigen yang terlarut dalam air dapat dihitung dengan menggunakan persamaan:

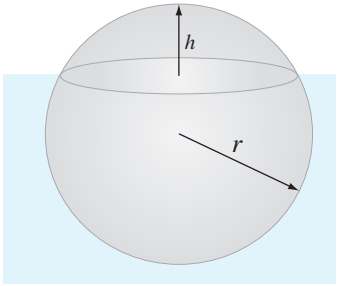
$$\ln o_{\text{sf}} = C_0 + \frac{C_1}{T_a} + \frac{C_2}{T_a^2} + \frac{C_3}{T_a^3} + \frac{C_4}{T_a^4}$$

di mana o_{sf} adalah konsentrasi oksigen dalam mg/L pada tekanan 1 atm, T_a adalah temperatur absolut, $T_a = T + 273.15$, T dalam derajat Celcius, dan parameter:

$$\begin{aligned} C_0 &= -139.34411 \\ C_1 &= 1.575701 \times 10^5 \\ C_2 &= -6.642308 \times 10^7 \\ C_3 &= 1.243800 \times 10^{10} \\ C_4 &= -8.621949 \times 10^{11} \end{aligned}$$

Gunakan metode bracketing dengan nilai tebakan awal 0 dan 40°C untuk mendapatkan suhu T jika diketahui $o_{\text{sf}} = 8, 10$ dan 12 mg/L.

Soal 10. Chapra Latihan 5.19 Menurut prinsip Archimedes, gaya apung sama dengan berat fluida yang dipindahkan oleh bagian benda yang terendam pada fluida. Perhatikan gambar di bawah ini.



Tentukan tinggi h yang mewakili bagian bola yang berada di atas air. Gunakan nilai-nilai berikut: $r = 1$ m, ρ_s = kerapatan bola = 200 kg/m^3 dan ρ_w = kerapatan air = 1000 kg/m^3. Volume bagian bola yang berada di atas permukaan air (tidak terendam) dapat dihitung dari:

$$V = \frac{\pi h^2}{3}(3r - h)$$

Soal 11. Chapra Latihan 6.25 Kesetimbangan massa dari suatu polutan

dalam suatu danau dapat dinyatakan dengan persamaan diferensial:

$$V\frac{\mathrm{d}c}{\mathrm{d}t} = W - Qc - kV\sqrt{c}$$

di mana c adalah konsentrasi polutan dalam $\mathrm{g/m^3}$, dengan parameter $V = 1 \times 10^6\,\mathrm{m^3}$, $Q = 1 \times 10^5\,\mathrm{m^3/tahun}$, $W = 1 \times 10^6\,\mathrm{g/tahun}$, dan $k = 0.25\,\mathrm{m^{0.5}/g^{0.5}/tahun}$. Hitung konsentrasi polutan dalam keadaan tunak.

Soal 12. Chapra Latihan 8.3 Dalam proses kimia, uap air ($\mathrm{H_2O}$) dipanaskan sampai suhu yang cukup tinggi sehingga sebagian besar dari air akan terdisosiasi membentuk oksigen ($\mathrm{O_2}$) dan hidrogen ($\mathrm{H_2}$) menurut persamaan reaksi

$$\mathrm{H_2O} \rightarrow \mathrm{H_2} + \mathrm{O_2}$$

Jika diasumsikan bahwa ini adalah satu-satunya reaksi yang terlibat, fraksi mol $\mathrm{H_2O}$ yang berdisosiasi, dilambangkan dengan x, dapat dihitung dari persamaan

$$K = \frac{x}{1-x}\sqrt{\frac{2p_t}{2+x}}$$

dimana K = kesetimbangan konstan reaksi dan p_t: tekanan total campuran dalam satuan atm. Jika diketahui bahwa $p_t = 3.5$ atm dan $K = 0.4$, tentukan nilai x yang memenuhi persamaan di atas.

Soal 13. Chapra Latihan 8.7 Persamaan keadan Redlich-Kwong dinyatakan sebagai:

$$p = \frac{RT}{v-b} - \frac{a}{v(v+b)\sqrt{T}}$$

di mana R = konstanta gas universal = 0.518 kJ/(kg K), T adalah temperature absolut (K), p adalah tekanan absolut (kPa), dan v = volume spesifik gas ($\mathrm{m^3/kg}$). Parameter a dan b dihitung dengan persamaan:

$$a = -0.427\frac{R^2T_c^{2.5}}{p_c}$$
$$b = 0.0866R\frac{T_c}{p_c}$$

di mana p_c dan T_c menyatakan tekanan dan temperatur kritis. Tentukan jumlah methana (yaitu v) (p_c = 4600 kPa and T_c = 191 K) yang dapat disimpan dalam tangki dengan volumne 3 $\mathrm{m^3}$ pada temperatur -40° C dengan tekanan 65000 MPa.

Soal 14. Chapra Latihan 8.18 Sebuah balok yang dikenai beban terdistribusi yang meningkat secara linier. Persamaan untuk kurva elastis

diberikan oleh persamaan:

$$y = \frac{w_0}{120EIL}\left(-x^5 + 2L^2x^3 - L^4x\right)$$

Defleksi maksimum dapat ditentukan dengan cara mencari nilai x yang memenuhi $\frac{\mathrm{d}y}{\mathrm{d}x}$. Tentukan defleksi maksimum jika diketahui bahwa: $L = 450\,\mathrm{cm}$, $E = 50000\,\mathrm{kN/cm^2}$, $I = 30000\,\mathrm{cm^4}$, dan $w0 = 2.5$ kN/cm.

Soal 15. Chapra Latihan 8.32 Suatu muatan total Q terdistribusi seragam pada suatu cincin konduktor dengan radius a. Suatu muatan uji q terletak pada jarak x dari titik tengah cincin. Gaya yand bekerja pada muatan uji yang disebabkan oleh distribusi muatan pada cincin diberikan oleh persamaan:

$$F = \frac{1}{4\pi\epsilon_0}\frac{qQx}{\left(x^2 + a^2\right)^{3/2}}$$

di mana $\epsilon_0 = 8.85 \times 10^{-12}\,\mathrm{C^2/(N\,m^2)}$. Tentukan jarak x jika $F = 1$ N, $q = Q = 2 \times 10^{-5}$ C, dan $a = 0.9$ m.

Soal 16. Chapra Latihan 8.33 Impedansi dari rangkaian paralel RLC dinyatakan oleh persamaan

$$\frac{1}{Z} = \sqrt{\frac{1}{R^2} + \left(\omega C - \frac{1}{\omega L}\right)^2}$$

Cari frekuensi angular ω untuk $Z = 75$ ohm, $R = 225$ ohm, $C = 0.6 \times 10^{-6}$ F dan $L = 0.5H$.

Soal 17. Chapra Latihan 8.39 Kecepatan kearah atas sebuah roket dapat dihitung menggunakan persamaan berikut

$$v = u\ln\frac{m_0}{m_0 - qt} - gt$$

dimana v = kecepatan kearah atas roket, u = kecepatan keluar bahan bakar relatif terhadap roket, m_0 = masa awal roket pada t = 0, q = laju konsumsi bahan bakar, dan g = percepatan gravitasi. Jika $g = 9.81\,\mathrm{m/s^2}$, $u = 2200$ m/s, $m_0 = 160000$ kg, dan $q = 2680$ kg/s, hitunglah waktu dimana kecepatan mencapai $v = 1000$ m/s.

Soal 18. Fungsi Bessel bola Fungsi Bessel bola (spherical Bessel) $j_n(x)$ dapat dituliskan sebagai berikut:

$$j_n(x) = (-x)^n\left(\frac{1}{x}\frac{\mathrm{d}}{\mathrm{d}x}\right)^n\frac{\sin(x)}{x}$$

Buatlah plot untuk $j_3(x)$ dan carilah semua akar-akarnya pada interval

(0,20). Gunakan salah satu metode untuk mencari akar persamaan nonlinear.

Soal 19. Chapra Latihan 6.23 Tentukan akar dari sistem persamaan nonlinear berikut

$$(x-4)^2 + (y-4)^2 = 5$$
$$x^2 + y^2 = 16$$

Buatlah plot dari fungsi-fungsi tersebut dan gunakan untuk mendapatkan estimasi tebakan awal untuk metode Newton-Raphson.

Soal 20. Chapra Latihan 6.24 Tentukan akar dari sistem persamaan nonlinear berikut

$$y = x^2 + 1$$
$$y = 2\cos(x)$$

Buatlah plot dari fungsi-fungsi tersebut dan gunakan untuk mendapatkan estimasi tebakan awal untuk metode Newton-Raphson.