Nama : Femas Arianda Rizki

NIM : 21120122130080

Kelas : Metode Numerik – B

1. Metode Matriks Balikan

Kode sumber dan kode testing:

```
# Nama  : Femas Arianda Rizki
# NIM   : 21120122130080
# Kelas : Metode Numerik - B

import numpy as np
import unittest

# Function for solving systems of linear equations using the
matrix inversion method
def inverse_matrix_method(matrix_A, matrix_B):
    try:
        A_inv = np.linalg.inv(matrix_A)
        X = np.dot(A_inv, matrix_B)
        return X
    except np.linalg.LinAlgError:
        return None

# Define the coefficient matrix (A) and constants matrix (B)
A = np.array([
                [1, -1, 2],
                [3, 0, 1],
                [1, 0, 2]])
B = np.array([
                [5],
                [10],
                [5]])

# Condition whether the inverse matrix can be applied or not
A_det = np.linalg.det(A)
if A_det == 0:
    print("Determinan matriks A adalah 0, tidak dapat dilakukan
invers atau balikan.")
else:
    X = inverse_matrix_method(A, B)
    print("Matriks solusi dari persamaan linear:")
    print(X)

    # Print the solution
    print("Jadi solusi dari persamaan linear adalah:")
```

```
    print("x = {:.1f}".format(X[0][0]))
    print("y = {:.1f}".format(X[1][0]))
    print("z = {:.1f}".format(X[2][0]))

# Test code
class TestLinearEquations(unittest.TestCase):
    def test_inverse_matrix(self):
        A = np.array([
                        [1, -1, 2],
                        [3, 0, 1],
                        [1, 0, 2]])

        B = np.array([
                        [5],
                        [10],
                        [5]])

        X = inverse_matrix_method(A, B)
        self.assertAlmostEqual(X[0][0], 3)
        self.assertAlmostEqual(X[1][0], 0)
        self.assertAlmostEqual(X[2][0], 1)

    def test_no_inverse_matrix(self):
        A = np.array([
                        [1, 2, 3],
                        [4, 8, 12],
                        [7, 8, 9]])

        B = np.array([
                        [6],
                        [15],
                        [24]])

        self.assertIsNone(inverse_matrix_method(A,B))

if __name__ == '__main__':
    unittest.main()
```

Penjelasan kode:

a. Import Library numpy dan unittest

```
import numpy as np
import unittest
```

- numpy digunakan untuk operasi matriks dan vektor pada Python.
- unittest adalah modul untuk menulis dan menjalankan unit test dalam Python.

b. Fungsi inverse_matrix_method

```
def inverse_matrix_method(matrix_A, matrix_B):
    try:
```

```
        A_inv = np.linalg.inv(matrix_A)
        X = np.dot(A_inv, matrix_B)
        return X
    except np.linalg.LinAlgError:
        return None
```

- Fungsi ini menerima dua argumen, yaitu matriks koefisien matrix_A dan matriks konstanta matrix_B.
- Fungsi mencoba melakukan invers matriks matrix_A menggunakan np.linalg.inv. Jika gagal (misalnya, karena determinan matriks adalah 0), maka fungsi akan mengembalikan None.

c. Penggunaan Fungsi inverse_matrix_method

```
A = np.array([
                [1, -1, 2],
                [3, 0, 1],
                [1, 0, 2]])
B = np.array([
                [5],
                [10],
                [5]])

A_det = np.linalg.det(A)
if A_det == 0:
    print("Determinan matriks A adalah 0, tidak dapat
dilakukan invers atau balikan.")
else:
    X = inverse_matrix_method(A, B)
    print("Matriks solusi dari persamaan linear:")
    print(X)

    print("Jadi solusi dari persamaan linear adalah:")
    print("x = {:.1f}".format(X[0][0]))
    print("y = {:.1f}".format(X[1][0]))
    print("z = {:.1f}".format(X[2][0]))
```

- Membuat matriks A dan B untuk digunakan dalam sistem persamaan linear.
- Menghitung determinan matriks A menggunakan np.linalg.det dan mengecek apakah determinan adalah 0. Jika determinan adalah 0, maka invers tidak dapat dilakukan dan pesan akan dicetak. Jika tidak, fungsi inverse_matrix_method dipanggil untuk mencari solusi sistem persamaan linear.

d. Unit Test Menggunakan unittest

```
class TestLinearEquations(unittest.TestCase):
    def test_inverse_matrix(self):
        # Test untuk matriks yang dapat diinvers
```

```
        A = np.array([
                        [1, -1, 2],
                        [3, 0, 1],
                        [1, 0, 2]])

        B = np.array([
                        [5],
                        [10],
                        [5]])

        X = inverse_matrix_method(A, B)
        self.assertAlmostEqual(X[0][0], 3)
        self.assertAlmostEqual(X[1][0], 0)
        self.assertAlmostEqual(X[2][0], 1)

    def test_no_inverse_matrix(self):
        # Test untuk matriks yang tidak dapat diinvers
        A = np.array([
                        [1, 2, 3],
                        [4, 8, 12],
                        [7, 8, 9]])

        B = np.array([
                        [6],
                        [15],
                        [24]])

        self.assertIsNone(inverse_matrix_method(A,B))

if __name__ == '__main__':
    unittest.main()
```

- Membuat kelas TestLinearEquations yang merupakan subclass dari unittest.TestCase.
- Mendefinisikan dua metode pengujian: test_inverse_matrix untuk matriks yang dapat diinvers dan test_no_inverse_matrix untuk matriks yang tidak dapat diinvers.
- Menggunakan self.assertAlmostEqual untuk membandingkan nilai float dan self.assertIsNone untuk memeriksa apakah fungsi mengembalikan None dengan benar.
- Jalankan unit test menggunakan unittest.main() jika file dieksekusi langsung.

2. Metode Dekomposisi LU Gauss

Kode sumber dan kode testing:

```
# Nama  : Femas Arianda Rizki
# NIM   : 21120122130080
# Kelas : Metode Numerik - B
```

```
import numpy as np
import unittest

# Function for LU gauss decomposition
def lu_gauss_decomposition(matrix):
    n = len(matrix)
    lower = np.zeros((n, n))
    upper = np.zeros((n, n))

    for i in range(n):
        lower[i][i] = 1

        for j in range(i, n):
            sum = 0
            for k in range(i):
                sum += (lower[i][k] * upper[k][j])

            upper[i][j] = matrix[i][j] - sum

        for j in range(i + 1, n):
            sum = 0
            for k in range(i):
                sum += (lower[j][k] * upper[k][i])

            lower[j][i] = (matrix[j][i] - sum) / upper[i][i]

    return lower, upper

# Function performs forward substitution  substitution on the
lower matrix and the constants matrix (B)
def forward_substitution(lower, b):
    n = len(b)
    y = np.zeros(n)

    for i in range(n):
        y[i] = b[i][0]
        for j in range(i):
            y[i] -= lower[i][j] * y[j]

    return y

# Function performs backward substitution on the upper matrix
and the result of backward substitution
def back_substitution(upper, y):
    n = len(y)
    x = np.zeros((n, 1))

    for i in range(n - 1, -1, -1):
```

```
        x[i] = y[i]
        for j in range(i + 1, n):
            x[i] -= upper[i][j] * x[j]
        x[i] /= upper[i][i]

    return x

# Function solves a system of linear equations using LU gauss
decomposition and forward-backward substitution
def solve_linear_equations(matrix, b):
    lower, upper = lu_gauss_decomposition(matrix)
    y = forward_substitution(lower, b)
    x = back_substitution(upper, y)
    return x

# Define the coefficient matrix (A) and constants matrix (B)
A = np.array([
                [1, -1, 2],
                [3, 0, 1],
                [1, 0, 2]])

B = np.array([
                [5],
                [10],
                [5]])

# Solve the linear equations
X = solve_linear_equations(A, B)
print("Matriks solusi dari persamaan linear:")
print(X)

# Print the solution
print("Jadi solusi dari persamaan linear adalah:")
print("x = {:.1f}".format(X[0][0]))
print("y = {:.1f}".format(X[1][0]))
print("z = {:.1f}".format(X[2][0]))

# Test code
class TestLinearEquations(unittest.TestCase):
    def test_lu_gauss_decomposition_(self):
        A = np.array([
                        [1, -1, 2],
                        [3, 0, 1],
                        [1, 0, 2]])

        B = np.array([
                        [5],
                        [10],
```

```
                        [5]])

        X = solve_linear_equations(A, B)
        self.assertAlmostEqual(X[0][0], 3)
        self.assertAlmostEqual(X[1][0], 0)
        self.assertAlmostEqual(X[2][0], 1)

if __name__ == '__main__':
    unittest.main()
```

Penjelasan kode:

a. Import Library numpy dan unittest

```
import numpy as np
import unittest
```

- numpy digunakan untuk operasi matriks dan vektor pada Python.
- unittest adalah modul untuk menulis dan menjalankan unit test dalam Python.

b. Fungsi lu_gauss_decomposition

```
def lu_gauss_decomposition(matrix):
    n = len(matrix)
    lower = np.zeros((n, n))
    upper = np.zeros((n, n))
```

- Fungsi ini memecahkan sistem persamaan linear menggunakan dekomposisi LU dan substitusi maju-mundur.

c. Fungsi forward_substitution

```
def forward_substitution(lower, b):
    n = len(b)
    y = np.zeros(n)
```

- Fungsi ini memecahkan sistem persamaan linear menggunakan dekomposisi LU dan substitusi maju-mundur.

d. Fungsi back_substitution

```
def back_substitution(upper, y):
    n = len(y)
    x = np.zeros((n, 1))
```

- Fungsi ini memecahkan sistem persamaan linear menggunakan dekomposisi LU dan substitusi maju-mundur.

e. Fungsi solve_linear_equations

```
def solve_linear_equations(matrix, b):
    lower, upper = lu_gauss_decomposition(matrix)
    y = forward_substitution(lower, b)
    x = back_substitution(upper, y)
    return x
```

- Fungsi ini memecahkan sistem persamaan linear menggunakan dekomposisi LU dan substitusi maju-mundur.

f. Definisi Matriks Koefisien (A) dan Konstanta (B)

```
A = np.array([
                [1, -1, 2],
                [3, 0, 1],
                [1, 0, 2]])

B = np.array([
                [5],
                [10],
                [5]])
```

- Mendefinisikan matriks koefisien A dan vektor konstanta B untuk sistem persamaan linear.

g. Penyelesaian Sistem Persamaan Linear

```
X = solve_linear_equations(A, B)
print("Matriks solusi dari persamaan linear:")
print(X)
```

- Memanggil fungsi solve_linear_equations untuk mencari solusi sistem persamaan linear dan mencetak solusinya.

h. Pengujian Unit

```
class TestLinearEquations(unittest.TestCase):
    def test_lu_gauss_decomposition_(self):
        A = np.array([
                        [1, -1, 2],
                        [3, 0, 1],
                        [1, 0, 2]])

        B = np.array([
                        [5],
                        [10],
                        [5]])

        X = solve_linear_equations(A, B)
        self.assertAlmostEqual(X[0][0], 3)
        self.assertAlmostEqual(X[1][0], 0)
        self.assertAlmostEqual(X[2][0], 1)
```

- Menggunakan unittest untuk menguji fungsi solve_linear_equations apakah mengembalikan solusi yang benar atau tidak.

i. Menjalankan Unit Test

```
if __name__ == '__main__':
    unittest.main()
```

- Menjalankan unit test jika file dijalankan sebagai skrip utama.

3. Metode Dekomposisi Crout

Kode sumber dan kode testing:

```
# Nama  : Femas Arianda Rizki
# NIM   : 21120122130080
# Kelas : Metode Numerik - B

import numpy as np
import unittest

def crout_reduction(matrix):
    n = len(matrix)
    lower = np.zeros((n, n))
    upper = np.zeros((n, n))

    for j in range(n):
        upper[j][j] = 1

        for i in range(j, n):
            sum = 0
            for k in range(j):
                sum += lower[i][k] * upper[k][j]
            lower[i][j] = matrix[i][j] - sum

        for i in range(j+1, n):
            sum = 0
            for k in range(j):
                sum += lower[j][k] * upper[k][i]
            if lower[j][j] == 0:
                return None  # Division by zero, no unique
solution
            upper[j][i] = (matrix[j][i] - sum) / lower[j][j]

    return lower, upper

def forward_substitution(lower, b):
    n = len(b)
    y = np.zeros(n)

    for i in range(n):
        y[i] = b[i][0]
        for j in range(i):
            y[i] -= lower[i][j] * y[j]
        y[i] /= lower[i][i]

    return y

def back_substitution(upper, y):
```

```
    n = len(y)
    x = np.zeros((n, 1))

    for i in range(n - 1, -1, -1):
        x[i] = y[i]
        for j in range(i + 1, n):
            x[i] -= upper[i][j] * x[j]
        x[i] /= upper[i][i]

    return x

def solve_linear_equations(matrix, b):
    lower, upper = crout_reduction(matrix)
    if lower is None or upper is None:
        return None  # No unique solution
    y = forward_substitution(lower, b)
    x = back_substitution(upper, y)
    return x

# Define the coefficient matrix (A) and constants matrix (B)
A = np.array([
                [1, -1, 2],
                [3, 0, 1],
                [1, 0, 2]])

B = np.array([
                [5],
                [10],
                [5]])

# Solve the linear equations
X = solve_linear_equations(A, B)
print("Matriks solusi dari persamaan linear:")
print(X)

# Print the solution
if X is not None:
    print("Solusi dari persamaan linear adalah:")
    print("x =", X[0])
    print("y =", X[1])
    print("z =", X[2])
else:
    print("Tidak ada solusi unik untuk sistem persamaan
linear.")

# Test code
class TestLinearEquations(unittest.TestCase):
    def test_lu_gauss_decomposition_(self):
```

```
        A = np.array([
                        [1, -1, 2],
                        [3, 0, 1],
                        [1, 0, 2]])

        B = np.array([
                        [5],
                        [10],
                        [5]])

        X = solve_linear_equations(A, B)
        self.assertAlmostEqual(X[0][0], 3)
        self.assertAlmostEqual(X[1][0], 0)
        self.assertAlmostEqual(X[2][0], 1)

if __name__ == '__main__':
    unittest.main()
```

Penjelasan kode:

a. Import Library numpy dan unittest

```
import numpy as np
import unittest
```

- numpy digunakan untuk operasi matriks dan vektor pada Python.
- unittest adalah modul untuk menulis dan menjalankan unit test dalam Python.

b. Fungsi crout_reduction

```
def crout_reduction(matrix):
    n = len(matrix)
    lower = np.zeros((n, n))
    upper = np.zeros((n, n))

    for j in range(n):
        upper[j][j] = 1

        for i in range(j, n):
            sum = 0
            for k in range(j):
                sum += lower[i][k] * upper[k][j]
            lower[i][j] = matrix[i][j] - sum

        for i in range(j+1, n):
            sum = 0
            for k in range(j):
                sum += lower[j][k] * upper[k][i]
            if lower[j][j] == 0:
                return None  # Division by zero, no unique
solution
            upper[j][i] = (matrix[j][i] - sum) / lower[j][j]
```

```
    return lower, upper
```

- Fungsi ini melakukan reduksi Crout pada matriks input matrix dan mengembalikan matriks lower dan upper hasil reduksi.

c. Fungsi forward_substitution

```
def forward_substitution(lower, b):
    n = len(b)
    y = np.zeros(n)

    for i in range(n):
        y[i] = b[i][0]
        for j in range(i):
            y[i] -= lower[i][j] * y[j]
        y[i] /= lower[i][i]

    return y
```

- Fungsi ini melakukan substitusi maju pada matriks lower dan vektor konstanta b

d. Fungsi back_substitution

```
def back_substitution(upper, y):
    n = len(y)
    x = np.zeros((n, 1))

    for i in range(n - 1, -1, -1):
        x[i] = y[i]
        for j in range(i + 1, n):
            x[i] -= upper[i][j] * x[j]
        x[i] /= upper[i][i]

    return x
```

- Fungsi ini melakukan substitusi mundur pada matriks upper dan hasil substitusi maju.

e. Fungsi solve_linear_equations

```
def solve_linear_equations(matrix, b):
    lower, upper = crout_reduction(matrix)
    if lower is None or upper is None:
        return None  # No unique solution
    y = forward_substitution(lower, b)
    x = back_substitution(upper, y)
    return x
```

- Fungsi ini memecahkan sistem persamaan linear menggunakan metode reduksi Crout dan substitusi maju-mundur.

f. Definisi Matriks Koefisien (A) dan Konstanta (B)

```
A = np.array([
                [1, -1, 2],
```

```
                [3, 0, 1],
                [1, 0, 2]])

B = np.array([
                [5],
                [10],
                [5]])
```

- Mendefinisikan matriks koefisien A dan vektor konstanta B untuk sistem persamaan linear.

g. Penyelesaian Sistem Persamaan Linear

```
X = solve_linear_equations(A, B)
print("Matriks solusi dari persamaan linear:")
print(X)

if X is not None:
    print("Solusi dari persamaan linear adalah:")
    print("x =", X[0])
    print("y =", X[1])
    print("z =", X[2])
else:
    print("Tidak ada solusi unik untuk sistem persamaan
linear.")
```

- Memanggil fungsi solve_linear_equations untuk mencari solusi sistem persamaan linear dan mencetak solusinya. Jika tidak ada solusi unik, maka pesan yang sesuai akan dicetak.

h. Pengujian Unit

```
class TestLinearEquations(unittest.TestCase):
    def test_lu_gauss_decomposition_(self):
        A = np.array([
                        [1, -1, 2],
                        [3, 0, 1],
                        [1, 0, 2]])

        B = np.array([
                        [5],
                        [10],
                        [5]])

        X = solve_linear_equations(A, B)
        self.assertAlmostEqual(X[0][0], 3)
        self.assertAlmostEqual(X[1][0], 0)
        self.assertAlmostEqual(X[2][0], 1

if __name__ == '__main__':
    unittest.main()
```

- Menggunakan unittest untuk menguji fungsi solve_linear_equations apakah menghasilkan solusi yang benar untuk kasus yang diberikan atau tidak.