Nama : Edi Wicoro

NIM : 21120122130073

Mata Kuliah : Metode Numerik B

1. Metode Matriks Balikan

```
# Nama  : Edi Wicoro
# NIM  : 21120122130073
# Kelas  : Metode Numerik B / Teknik Komputer

import numpy as np
import unittest

# Fungsi untuk mencari matriks balikan menggunakan NumPy
def inverse_matrix(matrix):
    try:
        inverse = np.linalg.inv(matrix)
        return inverse
    except np.linalg.LinAlgError:
        return None

# Contoh penggunaan
A = np.array([[1, -1, 2], [3, 0, 1], [1, 0, 2]])
print("Matriks A:")
print(A)

# Langkah-langkah untuk mencari matriks balikan A
print("\nLangkah-langkah:")
det_A = np.linalg.det(A)
print("   Determinan matriks A =", det_A)
if det_A == 0:
    print("   Karena determinan A = 0, maka A tidak memiliki balikan
(singular).")
else:
    adj_A = np.linalg.inv(A) * det_A
    print("   Matriks Adjoin A:")
    print(adj_A)
    inverse_A = inverse_matrix(A)
    print("   Matriks Balikan (inverse) A:")
    print(inverse_A)

# unit test
class TestInverseMatrix(unittest.TestCase):
    def test_inverse(self):
        # Tes untuk matriks yang memiliki balikan
        matrix = np.array([[1, -1, 2], [3, 0, 1], [1, 0, 2]])
```

```
        expected_result = np.array([[0.0, 0.4, -0.2], [-1.0, 0.0,
1.0], [0.0, -0.2, 0.6]])
        print("Expected Result:")
        print(expected_result)
        print("Actual Result:")
        print(inverse_matrix(matrix))
        self.assertTrue(np.allclose(inverse_matrix(matrix),
expected_result))

    def test_singular_matrix(self):
        # Tes untuk matriks yang tidak memiliki balikan
        matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
        print("\nTest untuk matriks yang tidak memiliki balikan:")
        print("Expected Result: None")
        print("Actual Result:")
        print(inverse_matrix(matrix))
        self.assertIsNone(inverse_matrix(matrix))

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

Penjelasan Kode

1. Import Library

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

Dua library yang diimpor adalah NumPy untuk operasi matriks dan unittest untuk pengujian unit.

2. Fungsi ‘invers_matrix(matrix)’

```
def inverse_matrix(matrix):
    try:
        inverse = np.linalg.inv(matrix)
        return inverse
    except np.linalg.LinAlgError:
        return None
```

Ini adalah fungsi yang menerima input berupa matriks dan mengembalikan matriks balikan menggunakan fungsi np.linalg.inv() dari NumPy. Jika matriks tersebut singular (tidak memiliki balikan), fungsi akan menangkap LinAlgError dan mengembalikan None.

3. Langkah Mencari Matriks Balikan

```
det_A = np.linalg.det(A)
if det_A == 0:
    # Tidak memiliki balikan jika determinan A = 0
else:
    adj_A = np.linalg.inv(A) * det_A
```

```
    inverse_A = inverse_matrix(A)
```

Determinan matriks A dihitung terlebih dahulu. Jika determinan A tidak sama dengan 0 (artinya matriks A memiliki balikan), maka matriks adjoin A dihitung terlebih dahulu dengan dikalikan dengan determinan. Selanjutnya, fungsi inverse_matrix() dipanggil untuk mencari matriks balikan A.

4. Unit Test

```
class TestInverseMatrix(unittest.TestCase):
    def test_inverse(self):
        # Tes untuk matriks yang memiliki balikan
        matrix = np.array([[1, -1, 2], [3, 0, 1], [1, 0, 2]])
        expected_result = np.array([[0.0, 0.4, -0.2], [-1.0,
0.0, 1.0], [0.0, -0.2, 0.6]])
        print("Expected Result:")
        print(expected_result)
        print("Actual Result:")
        print(inverse_matrix(matrix))
        self.assertTrue(np.allclose(inverse_matrix(matrix),
expected_result))

    def test_singular_matrix(self):
        # Tes untuk matriks yang tidak memiliki balikan
        matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
        print("\nTest untuk matriks yang tidak memiliki
balikan:")
        print("Expected Result: None")
        print("Actual Result:")
        print(inverse_matrix(matrix))
        self.assertIsNone(inverse_matrix(matrix))
```

Ini adalah definisi kelas pengujian unit. Terdapat dua metode pengujian, yaitu test_inverse() untuk matriks yang memiliki balikan, dan test_singular_matrix() untuk matriks yang tidak memiliki balikan.

2. Metode Dekomposisi LU Gausss

```
# Nama  : Edi Wicoro
# NIM   : 21120122130073
# Kelas  : Metode Numerik B / Teknik Komputer


import numpy as np

# Fungsi Dekomposisi LU menggunakan metode eliminasi Gauss
def lu_decomposition_gauss(matrix):
    n = len(matrix)
    L = np.zeros((n, n))
    U = np.zeros((n, n))
```

```
    for i in range(n):
        # Mengisi bagian diagonal L dengan 1
        L[i][i] = 1

        # Menghitung elemen-elemen U
        for k in range(i, n):
            sum = 0
            for j in range(i):
                sum += (L[i][j] * U[j][k])
            U[i][k] = matrix[i][k] - sum

        # Menghitung elemen-elemen L
        for k in range(i + 1, n):
            sum = 0
            for j in range(i):
                sum += (L[k][j] * U[j][i])
            L[k][i] = (matrix[k][i] - sum) / U[i][i]

    return L, U

# Menyelesaikan sistem persamaan linear dengan Dekomposisi LU
def solve_lu_decomposition(A, b):
    L, U = lu_decomposition_gauss(A)
    n = len(A)
    # Substitusi maju untuk mencari y
    y = np.zeros(n)
    for i in range(n):
        y[i] = (b[i] - np.dot(L[i, :i], y[:i])) / L[i, i]
    # Substitusi mundur untuk mencari x
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (y[i] - np.dot(U[i, i + 1:], x[i + 1:])) / U[i, i]
    return x

# Soal yang diberikan
A = np.array([[-3, 2, -1], [6, -6, 7], [3, -4, 4]])
b = np.array([-1, -7, -6])

# Langkah-langkah penyelesaian
print("Langkah-langkah penyelesaian:")
print("1. Menggunakan metode dekomposisi LU dengan metode eliminasi
Gauss untuk matriks koefisien.")
L, U = lu_decomposition_gauss(A)
print("   Matriks L:")
print(L)
print("   Matriks U:")
print(U)
```

```
print("\n2. Menggunakan substitusi maju dan mundur untuk mencari
solusi dari sistem persamaan linear.")

# Menyelesaikan sistem persamaan linear
solution = solve_lu_decomposition(A, b)
print("\nSolusi:")
print("x =", solution[0])
print("y =", solution[1])
print("z =", solution[2])
```

Penjelasan Kode

1. Import Library

```
import numpy as np
```

NumPy digunakan untuk operasi matriks dan vektor dalam penyelesaian sistem persamaan linear.

2. Fungsi 'Dekomposisi LU (lu_decomposition_gauss(matrix))'

```
def lu_decomposition_gauss(matrix):
    n = len(matrix)
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    for i in range(n):
        # Mengisi bagian diagonal L dengan 1
        L[i][i] = 1

        # Menghitung elemen-elemen U
        for k in range(i, n):
            sum = 0
            for j in range(i):
                sum += (L[i][j] * U[j][k])
            U[i][k] = matrix[i][k] - sum

        # Menghitung elemen-elemen L
        for k in range(i + 1, n):
            sum = 0
            for j in range(i):
                sum += (L[k][j] * U[j][i])
            L[k][i] = (matrix[k][i] - sum) / U[i][i]

    return L, U
```

Ini adalah fungsi yang mengimplementasikan dekomposisi LU dengan menggunakan metode eliminasi Gauss. Fungsi ini menerima matriks koefisien sebagai input dan mengembalikan matriks L (lower triangular) dan U (upper triangular).

3. Fungsi Penyelesaian Sistem Persamaan Linear (‘solve_lu_decomposition(A, b)’):

```
def solve_lu_decomposition(A, b):
    L, U = lu_decomposition_gauss(A)
    n = len(A)
    # Substitusi maju untuk mencari y
    y = np.zeros(n)
    for i in range(n):
        y[i] = (b[i] - np.dot(L[i, :i], y[:i])) / L[i, i]
    # Substitusi mundur untuk mencari x
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (y[i] - np.dot(U[i, i + 1:], x[i + 1:])) / U[i,
i]
    return x
```

Ini adalah fungsi untuk menyelesaikan sistem persamaan linear menggunakan dekomposisi LU. Fungsi ini menerima matriks koefisien A dan vektor hasil b sebagai input, dan mengembalikan vektor solusi.

4. Soal

```
A = np.array([[-3, 2, -1], [6, -6, 7], [3, -4, 4]])
b = np.array([-1, -7, -6])
```

Ini adalah matriks koefisien A dan vektor hasil b dari sistem persamaan linear yang diberikan.

5. Cetak Solusi

```
print("\nSolusi:")
print("x =", solution[0])
print("y =", solution[1])
print("z =", solution[2])
```

Solusi sistem persamaan linear dicetak dalam bentuk nilai x, y, dan z.

3. Metode Dekomposisi Crout

```
# Nama  : Edi Wicoro
# NIM  : 21120122130073
# Kelas  : Metode Numerik B / Teknik Komputer


import numpy as np
import unittest


def crout_decomposition(A):
    n = len(A)
    L = np.zeros((n, n))
    U = np.zeros((n, n))
```

```
    for j in range(n):
        U[j, j] = 1

        for i in range(j, n):
            sum_val = sum(L[i, k] * U[k, j] for k in range(i))
            L[i, j] = A[i, j] - sum_val

        for i in range(j, n):
            sum_val = sum(L[j, k] * U[k, i] for k in range(j))
            if L[j, j] == 0:
                return None, None     # Matriks tidak bisa
didekomposisi
            U[j, i] = (A[j, i] - sum_val) / L[j, j]

    return L, U


# Contoh penggunaan
A = np.array([[2, 4, 3],
              [3, 5, 2],
              [4, 6, 3]])


L, U = crout_decomposition(A)
print("Matrix L:")
print(L)
print("Matrix U:")
print(U)


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

# seharusnya U[[1, 1, -1], [0, 2, 0], [0, 0, 3]]
# seharusnya L[[1, 0, 0], [-1, 1, 0], [2, 0, 1]]

class TestCroutDecomposition(unittest.TestCase):
    def test_decomposition(self):
        A = np.array([[2, 4, 3],
                      [3, 5, 2],
                      [4, 6, 3]])
        expected_L = np.array([[2, 0, 0],
                               [3, -1, 0],
                               [4, -2, 2]])
        expected_U = np.array([[1, 2, 1.5],
                               [0, 1, 2.5],
                               [0, 0, 1]])
        L, U = crout_decomposition(A)
```

```
        np.testing.assert_array_almost_equal(L, expected_L)
        np.testing.assert_array_almost_equal(U, expected_U)


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

Penjelasan Kode

1. Import Library

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

NumPy digunakan untuk operasi matriks, sedangkan unittest digunakan untuk melakukan pengujian unit.

2. Fungsi Crout Decomposition (‘crout_decomposition(A)’):

```
def crout_decomposition(A):
    n = len(A)
    L = np.zeros((n, n))
    U = np.zeros((n, n))

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

        for i in range(j, n):
            sum_val = sum(L[i, k] * U[k, j] for k in range(i))
            L[i, j] = A[i, j] - sum_val

        for i in range(j, n):
            sum_val = sum(L[j, k] * U[k, i] for k in range(j))
            if L[j, j] == 0:
                return None, None    # Matriks tidak bisa
didekomposisi
            U[j, i] = (A[j, i] - sum_val) / L[j, j]

    return L, U
```

Fungsi ini mengimplementasikan algoritma dekomposisi Crout. Fungsi ini menerima matriks A sebagai input dan mengembalikan matriks segitiga bawah (L) dan matriks segitiga atas (U).

3. Contoh Penggunaan

```
A = np.array([[2, 4, 3],
              [3, 5, 2],
              [4, 6, 3]])
```

Ini adalah matriks A yang akan dipecah menggunakan dekomposisi Crout

4. Langkah Penyelesaian
    - Dua matriks segitiga, L dan U, diinisialisasi dengan nol.
    - Loop pertama (j) digunakan untuk mengisi elemen-elemen diagonal utama U dan menghitung elemen-elemen L.
    - Loop kedua (i) digunakan untuk menghitung elemen-elemen diagonal utama L dan elemen-elemen U yang berada di atas diagonal utama.

5. Cetak Hasil

```
print("Matrix L:")
print(L)
print("Matrix U:")
print(U)
```

Matriks segitiga bawah (L) dan matriks segitiga atas (U) dicetak untuk melihat hasil dekomposisi.

6. Unit Test

```
class TestCroutDecomposition(unittest.TestCase):
    def test_decomposition(self):
        A = np.array([[2, 4, 3],
                      [3, 5, 2],
                      [4, 6, 3]])
        expected_L = np.array([[2, 0, 0],
                               [3, -1, 0],
                               [4, -2, 2]])
        expected_U = np.array([[1, 2, 1.5],
                               [0, 1, 2.5],
                               [0, 0, 1]])
        L, U = crout_decomposition(A)
        np.testing.assert_array_almost_equal(L, expected_L)
        np.testing.assert_array_almost_equal(U, expected_U)
```

Kelas ini berisi metode pengujian unit untuk memastikan bahwa fungsi crout_decomposition() memberikan hasil yang benar. Dalam metode test_decomposition(), matriks L dan U yang dihasilkan dibandingkan dengan matriks yang diharapkan.