Nama : Naufal Yoga Pratama

NIM : 21120122130059

Mata Kuliah : Metode Numerik C

1. Metode Matriks Balikan

Kode:

```
# Nama   : Naufal Yoga Pratama
# NIM    : 21120122130059
# Kelas  : Metode Numerik C / 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
```

Kode tersebut mengimpor dua modul, yaitu numpy dan unittest. Modul numpy digunakan untuk melakukan perhitungan numerik yang efisien di Python, sedangkan modul unittest digunakan untuk menulis dan menjalankan tes di Python.

2. Fungsi ‘invers_matrix(matrix)’

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

Kode tersebut mendefinisikan fungsi bernama `inverse_matrix` yang menghitung invers dari matriks yang diberikan menggunakan library NumPy. Fungsi tersebut mencoba menghitung invers menggunakan fungsi `np.linalg.inv(matrix)` dan mengembalikan hasilnya. Jika matriks tidak dapat diinvers, maka terjadi kesalahan `LinAlgError`, dan fungsi tersebut menangkap kesalahan tersebut, mengembalikan None untuk menunjukkan bahwa matriks tidak dapat diinvers. Fungsi ini berguna untuk menghitung invers matriks, yang sering digunakan dalam berbagai aplikasi matematika dan ilmiah, seperti menyelesaikan sistem persamaan linear atau menemukan determinan matriks.

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)
```

Kode tersebut digunakan untuk menentukan apakah matriks A dapat diinvers atau tidak dengan cara menghitung determinan matriks tersebut. Jika determinan matriks A sama dengan nol, maka matriks A tidak dapat diinvers. Namun, jika determinan matriks A tidak sama dengan nol, maka kode tersebut akan menghitung adjugat dan invers matriks A menggunakan fungsi `np.linalg.inv` dan fungsi `inverse_matrix` yang tidak ditampilkan dalam kode tersebut.

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))
```

Kode tersebut mendefinisikan kelas pengujian bernama `TestInverseMatrix` yang mengwarisi dari kelas `unittest.TestCase`. Kelas ini memiliki dua metode pengujian, yaitu `test_inverse` dan `test_singular_matrix`. Metode-metode ini digunakan untuk menguji fungsi `inverse_matrix` yang menghitung invers matriks yang diberikan.

Metode `test_inverse` menguji fungsi `inverse_matrix` dengan matriks yang diketahui memiliki invers. Metode ini pertama-tama mendefinisikan matriks dan hasil yang diharapkan dari perhitungan invers. Lalu, metode ini mencetak hasil yang

diharapkan dan hasil yang sebenarnya dari fungsi `inverse_matrix`. Akhirnya, metode ini menggunakan metode `assertTrue` dari modul `unittest` untuk memastikan bahwa hasil yang sebenarnya dekat dengan hasil yang diharapkan, seperti yang dihitung oleh fungsi `np.allclose`.

Metode `test_singular_matrix` menguji fungsi `inverse_matrix` dengan matriks yang diketahui tidak memiliki invers. Metode ini pertama-tama mendefinisikan matriks dan mencetak hasil yang diharapkan (yaitu `None` karena matriks singular) dan hasil yang sebenarnya dari fungsi `inverse_matrix`. Akhirnya, metode ini menggunakan metode `assertIsNone` dari modul `unittest` untuk memastikan bahwa hasil yang sebenarnya adalah `None`.

2. Metode Dekomposisi LU Gausss

Kode:

```
# Nama   : Naufal Yoga Pratama
# NIM    : 21120122130059
# Kelas  : Metode Numerik C / Teknik Komputer


import numpy as np
import unittest

# 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("\nCara pertama 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("\nCara Kedua 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])


class TestLUDecomposition(unittest.TestCase):
    def test_decomposition(self):
        A = np.array([[-3, 2, -1], [6, -6, 7], [3, -4, 4]])
        expected_L = np.array([[1., 0., 0.], [-2., 1., 0.], [-1.,
1., 1.]])
        expected_U = np.array([[-3., 2., -1.], [0., -2., 5.], [0.,
0., -2.]])
```

```
        L, U = lu_decomposition_gauss(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 unit test
```

Kode tersebut mengimpor dua modul, yaitu numpy dan unittest. Modul numpy digunakan untuk melakukan perhitungan numerik yang efisien di Python, sedangkan modul unittest digunakan untuk menulis dan menjalankan tes di Python.

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
```

Kode tersebut mendefinisikan fungsi `lu_decomposition_gauss` yang melakukan decomposisi LU dari matriks yang diberikan menggunakan metode eliminasi Gauss. Fungsi ini menghitung elemen-elemen matriks `L` dan `U` yang mewakili bagian bawah dan atas matriks LU.

Fungsi ini menggunakan loop luar untuk menghitung elemen-elemen `L` dan loop dalam untuk menghitung elemen-elemen `U`. Elemen-elemen `U` dihitung dengan menghitung jumlah produk elemen-elemen `L` dan `U`, lalu mengurangi hasilnya dengan elemen-elemen matriks input. Elemen-elemen `L` dihitung dengan menghitung jumlah produk elemen-elemen `L` dan `U`, lalu membagi hasilnya dengan elemen-elemen `U`. Fungsi ini kemudian mengembalikan matriks `L` dan `U` yang mewakili decomposisi LU dari matriks input.

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
```

Kode tersebut mendefinisikan fungsi `solve_lu_decomposition` yang digunakan untuk menyelesaikan sistem persamaan linear menggunakan metode dekomposisi LU. Fungsi ini pertama-tama melakukan dekomposisi LU pada matriks input `A` menggunakan fungsi `lu_decomposition_gauss`.

Selanjutnya, fungsi ini menggunakan loop untuk melakukan substitusi maju dan mundur. Substitusi maju digunakan untuk mencari vektor `y` dengan cara menghitung nilai-nilai `y` dengan menghitung jumlah produk elemen-elemen `L` dan `y` (hingga elemen `i`-th), lalu membagi hasilnya dengan elemen `i`-th dari `L`.

Substitusi mundur digunakan untuk mencari vektor `x` dengan cara menghitung nilai-nilai `x` dengan menghitung jumlah produk elemen-elemen `U` dan `x` (dari elemen (`i` + 1)-th hingga akhir), lalu membagi hasilnya dengan elemen `i` -th dari `U`. Fungsi ini kemudian mengembalikan vektor `x` sebagai solusi sistem persamaan linear.

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`.

6. Unit Test

```
class TestLUDecomposition(unittest.TestCase):
    def test_decomposition(self):
        A = np.array([[-3, 2, -1], [6, -6, 7], [3, -4, 4]])
        expected_L = np.array([[1., 0., 0.], [-2., 1., 0.], [-
1., 1., 1.]])
        expected_U = np.array([[-3., 2., -1.], [0., -2., 5.],
[0., 0., -2.]])
        L, U = lu_decomposition_gauss(A)
        np.testing.assert_array_almost_equal(L, expected_L)
        np.testing.assert_array_almost_equal(U, expected_U)


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

Kode tersebut mendefinisikan kelas pengujian bernama `TestLUDecomposition` yang mengwarisi dari kelas `unittest.TestCase`. Kelas ini memiliki satu metode pengujian bernama `test_decomposition` yang digunakan untuk menguji fungsi `lu_decomposition_gauss` dengan cara membandingkan hasilnya dengan hasil yang diharapkan.

Dalam metode pengujian, matriks `A` digunakan sebagai input untuk fungsi `lu_decomposition_gauss`, dan hasilnya disimpan dalam variabel `L` dan `U`. Kemudian, fungsi `assert_array_almost_equal` digunakan untuk membandingkan hasil `L` dan `U` dengan hasil yang diharapkan, yaitu `expected_L` dan `expected_U`. Jika hasilnya cocok, maka pengujian berjalan dengan baik; jika tidak, maka pengujian gagal.

3. Metode Dekomposisi Crout

Kode:

```
# Nama   : Naufal Yoga Pratama
# NIM    : 21120122130059
# Kelas  : Metode Numerik C / 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
```

Kode tersebut mengimpor dua modul, yaitu numpy dan unittest. Modul numpy digunakan untuk melakukan perhitungan numerik yang efisien di Python, sedangkan modul unittest digunakan untuk menulis dan menjalankan tes di Python.

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
```

Kode tersebut mendefinisikan fungsi `crout_decomposition` yang digunakan untuk melakukan dekomposisi Crout pada matriks yang diberikan. Fungsi ini menggunakan loop luar untuk menghitung elemen-elemen matriks `L` dan `U`.

Dalam loop luar, fungsi ini menghitung elemen-elemen `U` dengan cara menghitung jumlah produk elemen-elemen `L` dan `U`, lalu membagi hasilnya dengan elemen-elemen L. Fungsi ini juga menggunakan loop dalam untuk menghitung elemen-elemen L dengan cara menghitung jumlah produk elemen-elemen `L` dan `U`, lalu mengurangi hasilnya dengan elemen-elemen matriks input.

Fungsi ini kemudian mengembalikan matriks `L` dan `U` yang mewakili dekomposisi Crout dari matriks input. Dekomposisi Crout ini digunakan dalam

berbagai operasi linear algebra, seperti menyelesaikan sistem persamaan linear dan menemukan determinan matriks.

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)
```

Kode tersebut digunakan untuk mencetak matriks `L` dan `U` yang merupakan hasil dekomposisi Crout dari matriks yang diberikan. Kode ini menggunakan perintah `print` untuk mencetak string "Matrix L:" dan "Matrix U:" untuk menunjukkan bahwa output berikutnya adalah matriks `L` dan `U`.

Matriks `L` adalah bagian bawah triangular dari dekomposisi Crout, sedangkan matriks `U` adalah bagian atas triangular. Kode ini digunakan untuk memvisualisasikan hasil dekomposisi Crout, yang adalah langkah fundamental dalam berbagai operasi algebra linear, seperti menyelesaikan sistem persamaan linear dan menemukan determinan matriks.

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)
```

Kode tersebut mendefinisikan kelas pengujian bernama `TestCroutDecomposition` yang mengwarisi dari kelas `unittest.TestCase`. Kelas ini memiliki satu metode pengujian bernama `test_decomposition` yang digunakan untuk menguji fungsi `crout_decomposition` dengan cara membandingkan hasilnya dengan hasil yang diharapkan.

Dalam metode pengujian, matriks A digunakan sebagai input untuk fungsi `crout_decomposition`, dan hasilnya disimpan dalam variabel `L` dan `U`. Kemudian, fungsi `assert_array_almost_equal` digunakan untuk membandingkan hasil `L` dan `U` dengan hasil yang diharapkan, yaitu `expected_L` dan `expected_U`. Jika hasilnya cocok, maka pengujian berjalan dengan baik; jika tidak, maka pengujian gagal.