Nama : Imam Baihaqqy

NIM : 21120122130078

Mata Kuliah : Metode Numerik

Kelas : D

Penjelasan Kode

1. Matrix Balikan
   Kode:

```
# Nama          : Imam Baihaqqy
# NIM           : 21120122130078
# Mata Kuliah   : Metode Numerik
# Kelas         : D

import numpy as np
import unittest

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

class TestInverseMatrix(unittest.TestCase):

    def test_inverse(self):
        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]])
        self.assertTrue(np.allclose(inverse_matrix(matrix)
, expected_result))
        print ("Matrix: ")
        print(matrix)
        print ("Inverse matrix yang diharapkan:")
        print (expected_result)
        print ("Hasil Perhitungan:")
        print (inverse_matrix(matrix))
```

```
        def test_singular_matrix(self):
            matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8,
    9]])
            self.assertIsNone(inverse_matrix(matrix))

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

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

Pada kode diatas kita perlu mengimport library yang berfungsi untuk operasi/komputasi bawaan dari python dan juga library unittest untuk pengujian yang bawaan dari python

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

Fungsi `inverse_matrix(matrix)` merupakan fungsi Python yang menggunakan modul `numpy` untuk menghitung invers dari suatu matriks yang diberikan sebagai argumen. Fungsi ini mencoba menghitung invers matriks tersebut dengan menggunakan fungsi `np.linalg.inv(matrix)`, dan jika perhitungan berhasil, mengembalikan hasil inversnya. Namun, jika terjadi `LinAlgError`, yang menandakan bahwa matriks tersebut tidak memiliki invers (seperti matriks singular), fungsi akan menangkap pengecualian tersebut dan mengembalikan `None`. Dengan demikian, fungsi ini berguna untuk menghitung invers matriks dengan penanganan yang tepat terhadap kasus matriks yang tidak memiliki invers.

```
    class TestInverseMatrix(unittest.TestCase):

        def test_inverse(self):
            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]])
            self.assertTrue(np.allclose(inverse_matrix(matrix)
    , expected_result))
```

Kode di atas merupakan metode pengujian dalam kelas `TestInverseMatrix` yang menguji fungsi `inverse_matrix(matrix)`. Metode ini membandingkan hasil yang dikembalikan oleh fungsi `inverse_matrix` ketika diberikan matriks spesifik dengan hasil yang diharapkan menggunakan fungsi `np.allclose`. Jika keduanya mendekati satu sama lain dengan toleransi tertentu, pengujian dianggap berhasil. Ini memberikan verifikasi bahwa fungsi `inverse_matrix` berfungsi sesuai yang diharapkan.

```
    def test_singular_matrix(self):
            matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8,
    9]])
            self.assertIsNone(inverse_matrix(matrix))
```

Kode di atas adalah metode pengujian dalam kelas `TestInverseMatrix` yang menguji kasus ketika fungsi `inverse_matrix(matrix)` diberikan sebuah matriks singular (matriks yang tidak memiliki invers). Metode ini memanggil fungsi `inverse_matrix` dengan matriks singular yang telah ditentukan dan menggunakan `self.assertIsNone` untuk memastikan bahwa hasil yang dikembalikan oleh fungsi adalah `None`, menunjukkan bahwa matriks tersebut tidak memiliki invers. Dengan demikian, pengujian ini memverifikasi bahwa fungsi `inverse_matrix` secara benar menangani kasus matriks singular.

2. Dekomposisi Lu Gauss

   Kode:

```
# Nama                  : Imam Baihaqqy
# NIM           : 21120122130078
# Mata Kuliah     : Metode Numerik
# Kelas                 : D

import numpy as np
import unittest

def lu_decomposition(A):
    # Get the size of the matrix
```

```
    n = A.shape[0]

    # Initialize L and U matrices
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    # Perform Gaussian elimination
    for i in range(n):
        # Set diagonal elements of L to 1
        L[i][i] = 1

        # Compute upper triangular matrix U
        for k in range(i, n):
            total = sum(L[i][p] * U[p][k] for p in range(i))
            U[i][k] = A[i][k] - total

        # Compute lower triangular matrix L
        for k in range(i+1, n):
            total = sum(L[k][p] * U[p][i] for p in range(i))
            L[k][i] = (A[k][i] - total) / U[i][i]

    return L, U

# Define a sample matrix
A = np.array([[4, 3, -1], [-2, -4, 5], [1, 2, 6]])

# Perform LU decomposition
L, U = lu_decomposition(A)

print("Lower triangular matrix L:")
print(L)
print("\nUpper triangular matrix U:")
print(U)

class TestLUDecomposition(unittest.TestCase):

    def test_decomposition(self):
        A = np.array([[4, 3, -1], [-2, -4, 5], [1, 2, 6]])
        expected_L = np.array([[1., 0., 0.], [-0.5, 1., 0.],
[0.25, -0.5, 1.]])
```

```
        expected_U = np.array([[4., 3., -1.], [0., -2.5, 4.5],
[0., 0., 8.5]])
        L, U = lu_decomposition(A)
        self.assertTrue(np.allclose(L, expected_L))
        self.assertTrue(np.allclose(U, expected_U))

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

Penjelasan:

```
def lu_decomposition(A):
    # Get the size of the matrix
    n = A.shape[0]

    # Initialize L and U matrices
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    # Perform Gaussian elimination
    for i in range(n):
        # Set diagonal elements of L to 1
        L[i][i] = 1

        # Compute upper triangular matrix U
        for k in range(i, n):
            total = sum(L[i][p] * U[p][k] for p in range(i))
            U[i][k] = A[i][k] - total

        # Compute lower triangular matrix L
        for k in range(i+1, n):
            total = sum(L[k][p] * U[p][i] for p in range(i))
            L[k][i] = (A[k][i] - total) / U[i][i]

    return L, U
```

Fungsi `lu_decomposition(A)` mengimplementasikan algoritma dekomposisi LU menggunakan metode eliminasi Gauss. Dengan memanfaatkan matriks nol `L` dan `U` dengan ukuran yang sama seperti matriks masukan `A`, fungsi ini secara iteratif menghitung elemen-elemen matriks `L` dan `U`. Proses iterasi ini melibatkan pengaturan elemen diagonal `L` menjadi 1, perhitungan elemen-elemen di atas

diagonal untuk matriks `U`, dan perhitungan elemen-elemen di bawah diagonal untuk matriks `L`. Setelah iterasi selesai, matriks `L` dan `U` yang dihasilkan merupakan dekomposisi LU dari matriks masukan `A`, yang kemudian dikembalikan sebagai keluaran fungsi. Dengan demikian, fungsi ini berguna untuk mendapatkan matriks lower triangular `L` dan upper triangular `U` dari matriks masukan `A` menggunakan algoritma dekomposisi LU.

```
class TestLUDecomposition(unittest.TestCase):


    def test_decomposition(self):
        A = np.array([[4, 3, -1], [-2, -4, 5], [1, 2, 6]])
        expected_L = np.array([[1., 0., 0.], [-0.5, 1., 0.],
[0.25, -0.5, 1.]])
        expected_U = np.array([[4., 3., -1.], [0., -2.5, 4.5],
[0., 0., 8.5]])
        L, U = lu_decomposition(A)
        self.assertTrue(np.allclose(L, expected_L))
        self.assertTrue(np.allclose(U, expected_U))
```

Kelas `TestLUDecomposition` adalah kelas pengujian unit yang diturunkan dari `unittest.TestCase`, yang bertujuan untuk menguji fungsi `lu_decomposition(A)` dengan matriks spesifik. Metode `test_decomposition` dalam kelas ini menguji hasil dari dekomposisi LU yang dihasilkan oleh fungsi `lu_decomposition` dengan matriks `A`. Pengujian dilakukan dengan membandingkan matriks lower triangular (`L`) dan upper triangular (`U`) yang dihasilkan oleh fungsi tersebut dengan matriks lower triangular dan upper triangular yang diharapkan (`expected_L` dan `expected_U`). Jika keduanya mendekati satu sama lain dengan toleransi tertentu, pengujian dianggap berhasil. Dengan demikian, metode ini memberikan verifikasi bahwa fungsi `lu_decomposition` menghasilkan dekomposisi LU yang benar dari matriks yang diberikan.

3. Dekomposisi Crout

   Kode:

```
# Nama                  : Imam Baihaqqy
# NIM           : 21120122130078
# Mata Kuliah     : Metode Numerik
# Kelas                 : D

import numpy as np
```

```
import unittest
def crout(A):
    if A.shape[0] != A.shape[1]:
        raise ValueError('A must be a square matrix')

    n = A.shape[0]
    L = np.zeros((n, n))
    U = np.eye(n)

    for i in range(n):
        L[i, 0] = A[i, 0]

    for j in range(1, n):
        U[0, j] = A[0, j] / L[0, 0]

    for i in range(1, n):
        for j in range(1, i + 1):
            L[i, j] = A[i, j] - np.dot(L[i, 0:j], U[0:j, j])

        for j in range(i + 1, n):
            U[i, j] = (A[i, j] - np.dot(L[i, 0:i], U[0:i, j])) /
L[i, i]

    return L, U

# Test the crout function
A = np.array([[2, 4, 3],
              [3, 5, 2],
              [4, 6, 3]])
L, U = crout(A)
print("Lower triangular matrix L:")
print(L)
print("\nUpper triangular matrix U:")
print(U)

class TestCrout(unittest.TestCase):

    def test_decomposition(self):
        L, U = crout(A)
        reconstructed_A = np.dot(L, U)
```

```
        np.testing.assert_allclose(reconstructed_A, A, atol=1e-
10)

    def test_non_square_matrix(self):
        A = np.array([[1, 2, 3], [4, 5, 6]])  # Non-square
matrix
        with self.assertRaises(ValueError):
            crout(A)

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

Penjelasan:

```
def crout(A):
    if A.shape[0] != A.shape[1]:
        raise ValueError('A must be a square matrix')

    n = A.shape[0]
    L = np.zeros((n, n))
    U = np.eye(n)

    for i in range(n):
        L[i, 0] = A[i, 0]

    for j in range(1, n):
        U[0, j] = A[0, j] / L[0, 0]

    for i in range(1, n):
        for j in range(1, i + 1):
            L[i, j] = A[i, j] - np.dot(L[i, 0:j], U[0:j, j])

        for j in range(i + 1, n):
            U[i, j] = (A[i, j] - np.dot(L[i, 0:i], U[0:i, j])) /
L[i, i]

    return L, U
```

Fungsi `crout(A)` adalah sebuah implementasi dari metode dekomposisi LU menggunakan pendekatan Crout. Pada awalnya, fungsi memeriksa apakah matriks `A` adalah matriks persegi dengan memeriksa apakah jumlah baris sama dengan jumlah kolom. Jika tidak, fungsi akan memunculkan ValueError. Kemudian, matriks `L` dan `U` yang akan menyimpan matriks lower triangular dan upper

triangular dari dekomposisi LU diinisialisasi. Proses iterasi dilakukan untuk menghitung elemen-elemen dari matriks `L` dan `U` berdasarkan rumus iteratif Crout. Selama iterasi, nilai-nilai dari matriks `L` dan `U` dihitung dengan memanfaatkan nilai-nilai dari matriks `A`. Setelah proses iterasi selesai, matriks `L` dan `U` yang dihasilkan menggambarkan dekomposisi LU dari matriks `A`, dan keduanya dikembalikan sebagai output fungsi. Dengan demikian, fungsi ini digunakan untuk mendapatkan dekomposisi LU dari matriks persegi `A` menggunakan pendekatan Crout.

```
class TestCrout(unittest.TestCase):

    def test_decomposition(self):
        L, U = crout(A)
        reconstructed_A = np.dot(L, U)
        np.testing.assert_allclose(reconstructed_A, A, atol=1e-
10)

    def test_non_square_matrix(self):
        A = np.array([[1, 2, 3], [4, 5, 6]])  # Non-square matrix
        with self.assertRaises(ValueError):
            crout(A)
```

Kelas `TestCrout` adalah kelas pengujian unit yang diturunkan dari `unittest.TestCase`. Metode `test_decomposition` di dalamnya bertujuan untuk menguji apakah fungsi `crout(A)` dapat menghasilkan dekomposisi LU dengan benar dari suatu matriks persegi `A`. Pengujian dilakukan dengan membangun kembali matriks `A` dari matriks lower triangular (`L`) dan upper triangular (`U`) yang dihasilkan oleh fungsi `crout`, dan membandingkannya dengan matriks `A` asli. Jika keduanya mendekati satu sama lain dengan toleransi tertentu, pengujian dianggap berhasil.

Metode `test_non_square_matrix` menguji kasus ketika fungsi `crout(A)` diberikan matriks yang bukan matriks persegi (non-square). Dalam kasus ini, diharapkan fungsi untuk memunculkan ValueError karena dekomposisi LU hanya dapat dilakukan pada matriks persegi. Jadi, pengujian ini memastikan bahwa fungsi berperilaku dengan benar dengan menangani kasus matriks non-square.