Annastasha P.A.

21120122140096

Metode Numerik Kelas C

1. Metode Matriks Balikan

```
#Annastasha Prunnel Anjelina
#NIM 21120122140096
#Metode Numerik Kelas C

import numpy as np
import unittest

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

A = np.array([[1, -1, 2], [3, 0, 1], [1, 0, 2]])
inverse_A = inverse_matrix(A)
if inverse_A is not None:
    print("Matriks Balikan (inverse Matrix) A:")
    print(inverse_A)
else:
    print("Matriks A tidak memiliki balikan (inverse).")

# 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]])
        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]])
        self.assertIsNone(inverse_matrix(matrix))

if __name__ == '__main__':
```

```
41.     unittest.main()
```

Pada code di atas digunakan numpy untuk menghitung invers matriks menggunakan fungsi *'inverse_matrix()'*, serta implementasi menggunakan unit test menggunakan modul *'unittest'*. Hasil eksekusi dari kode tersebut akan menunjukkan apakah fungsi invers matriks berperilaku sesuai yang diharapkan.
Kode ini adalah contoh implementasi fungsi untuk menghitung invers dari sebuah matriks menggunakan NumPy. Ini juga mencakup unit test menggunakan modul unittest untuk memastikan fungsi berperilaku dengan benar dalam kasus yang diinginkan.

bagian-bagian utama dari kode:

1. import numpy as np: Ini mengimpor pustaka NumPy dan memberi alias sebagai np, yang umumnya digunakan sebagai konvensi.

2. import unittest: Ini mengimpor modul unittest yang digunakan untuk menulis dan menjalankan unit test di Python.

3. def inverse_matrix(matrix): Ini adalah fungsi yang mengambil matriks sebagai argumen dan mencoba menghitung inversnya menggunakan np.linalg.inv(). Jika invers berhasil dihitung, fungsi mengembalikan inversnya. Jika matriks singular atau tidak memiliki invers, fungsi mengembalikan None.

4. A = np.array([[1, -1, 2], [3, 0, 1], [1, 0, 2]]): Ini adalah contoh matriks yang akan kita coba hitung inversnya.

5. inverse_A = inverse_matrix(A): Ini memanggil fungsi inverse_matrix() untuk mencoba menghitung invers dari matriks A.

6. if inverse_A is not None:: Ini memeriksa apakah hasil invers tidak None, artinya invers berhasil dihitung.

7. print("Matriks Balikan (inverse Matrix) A:"): Ini hanya mencetak pesan untuk memberi tahu bahwa invers matriks A akan ditampilkan.

8. print(inverse_A): Ini mencetak hasil invers dari matriks A.

9. else:: Bagian ini menangani kasus ketika invers matriks tidak dapat dihitung.

10. class TestInverseMatrix(unittest.TestCase):: Ini adalah kelas untuk unit test. Ini mengambil unittest.TestCase sebagai kelas dasar.

11. def test_inverse(self):: Ini adalah metode untuk menguji fungsi inverse_matrix() ketika matriks memiliki invers yang valid.

12. def test_singular_matrix(self):: Ini adalah metode untuk menguji fungsi inverse_matrix() ketika matriks tidak memiliki invers (singular).

13. if __name__ == '__main__':: Ini adalah blok yang akan dieksekusi jika skrip ini dijalankan sebagai skrip utama. Itu akan menjalankan semua unit test yang didefinisikan dalam kelas TestInverseMatrix.

2. Metode Dekomposisi LU Gauss

```
#Annastasha Prunnel Anjelina
#NIM 21120122140096
#Metode Numerik Kelas C

import unittest
import numpy as np


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


# Testing
A = np.array([[2, -1, -1], [0, -4, 2], [6, -3, 1]])
L, U = lu_decomposition_gauss(A)
print("Matriks L:")
print(L)
print("Matriks U:")
print(U)


class TestLUDecomposition(unittest.TestCase):
```

```
    def test_decomposition(self):
        A = np.array([[2, -1, -1], [0, -4, 2], [6, -3, 1]])
        expected_L = np.array([[1., 0., 0.], [0., 1., 0.], [3., 0.,
1.]])
        expected_U = np.array([[2., -1., -1.], [0., -4., 2.], [0.,
0., 4.]])
        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 ini adalah implementasi dari metode dekomposisi LU (Lower-Upper) menggunakan metode Gauss untuk sebuah matriks persegi. Metode ini memecah matriks menjadi dua matriks, yaitu matriks segitiga bawah (L) dan matriks segitiga atas (U).
Fungsi lu_decomposition_gauss() dapat secara akurat menghitung faktorisasi LU dari matriks yang diberikan.

Mari kita jelaskan bagian-bagian utama dari kode:

1. `import unittest`: Ini mengimpor modul `unittest` yang digunakan untuk menulis dan menjalankan unit test di Python.

2. `import numpy as np`: Ini mengimpor pustaka NumPy dan memberi alias sebagai `np`, yang umumnya digunakan sebagai konvensi.

3. `def lu_decomposition_gauss(matrix)`: Ini adalah fungsi untuk melakukan dekomposisi LU menggunakan metode Gauss. Fungsi ini mengambil matriks sebagai argumen dan mengembalikan dua matriks, yaitu matriks segitiga bawah (L) dan matriks segitiga atas (U).

4. `# Testing`: Ini adalah bagian kode yang akan digunakan untuk menguji fungsi `lu_decomposition_gauss()`.

5. `A = np.array([[2, -1, -1], [0, -4, 2], [6, -3, 1]])`: Ini adalah contoh matriks yang akan kita dekomposisi.

6. `L, U = lu_decomposition_gauss(A)`: Ini memanggil fungsi `lu_decomposition_gauss()` untuk melakukan dekomposisi pada matriks A dan menyimpan hasilnya dalam matriks L dan U.

7. `print("Matriks L:")` dan `print("Matriks U:")`: Ini mencetak matriks L dan U ke layar.

8. `class TestLUDecomposition(unittest.TestCase):`: Ini adalah kelas untuk unit test. Ini mengambil `unittest.TestCase` sebagai kelas dasar.

9. `def test_decomposition(self):`: Ini adalah metode untuk menguji fungsi `lu_decomposition_gauss()`.

10. `expected_L` dan `expected_U`: Ini adalah matriks yang diharapkan sebagai hasil dekomposisi dari matriks A.

11. `np.testing.assert_array_almost_equal(L, expected_L)` dan `np.testing.assert_array_almost_equal(U, expected_U)`: Ini adalah pernyataan pengujian yang membandingkan hasil dekomposisi aktual dengan hasil yang diharapkan. Jika keduanya hampir sama (dalam toleransi yang ditentukan), maka pengujian akan lolos.

12. `if __name__ == '__main__':`: Ini adalah blok yang akan dieksekusi jika skrip ini dijalankan sebagai skrip utama. Itu akan menjalankan semua unit test yang didefinisikan dalam kelas `TestLUDecomposition`.

Dengan menjalankan skrip ini, kita dapat memastikan bahwa fungsi dekomposisi LU yang diimplementasikan memberikan hasil yang diharapkan.

3. Metode Dekomposisi Crout

```python
#Annastasha Prunnel Anjelina
#NIM 21120122140096
#Metode Numerik Kelas C

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

# Test
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()
```

Faktorisasi Crout adalah teknik penting dalam aljabar linear yang memecah matriks persegi menjadi dua matriks segitiga, yaitu segitiga bawah (L) dan segitiga atas (U). Dalam tugas ini, dalam kode akan mengimplementasikan algoritma ini menggunakan Python dengan bantuan pustaka NumPy. Kode juga akan melakukan pengujian unit untuk memastikan kebenaran implementasi algoritma.

Kode yang digunakan dalam tugas ini terdiri dari beberapa bagian utama:

1. Fungsi `crout_decomposition(A)`: Ini adalah implementasi algoritma faktorisasi Crout. Fungsi ini menerima matriks A sebagai input dan mengembalikan dua matriks L dan U yang merupakan hasil faktorisasi Crout.

2. Pengujian Unit: Pengujian unit dilakukan menggunakan modul `unittest`. Kami menentukan kelas `TestCroutDecomposition` yang berisi metode `test_decomposition()`. Metode ini menguji apakah hasil faktorisasi Crout dari fungsi implementasi sama dengan hasil yang diharapkan.

3. Eksekusi Utama: Blok kode `if __name__ == '__main__':` digunakan untuk mengeksekusi unit test ketika skrip dijalankan sebagai skrip utama.