Nama : Wahyu Ramadhani

NIM : 21120122140158

Metode Numerik kelas D

1. Metode Balikan

```
import numpy as np

# Fungsi untuk mencari solusi sistem persamaan linear
def solusi_persamaan_linear(A, b):
# Menghitung determinan matriks A
det_A = np.linalg.det(A)

    # Memeriksa apakah matriks A memiliki invers
    if det_A == 0:
        print("Matriks A tidak memiliki invers.")
        return None

    # Menghitung invers matriks A
    A_inv = np.linalg.inv(A)

    # Menghitung solusi x
    x = np.dot(A_inv, b)

    return x

# Contoh penggunaan
A = np.array([[1, -1, 2],
              [3, 0, 1],
              [1, 0, 2]])
b = np.array([5, 10, 5])

solusi = solusi_persamaan_linear(A, b)

if solusi is not None:
    print("Solusi sistem persamaan linear:")
    print(solusi)
```

Penjelasan :

```
import numpy as np
```

Baris ini mengimpor library NumPy yang menyediakan dukungan untuk operasi numerik seperti operasi matriks, aljabar linear, dan lain-lain.

```
def solusi_persamaan_linear(A, b):
```

Fungsi ini menerima dua argumen:

`A`: Matriks koefisien dari sistem persamaan linear, dalam bentuk numpy array dua dimensi

`b`: Vektor konstanta dari sistem persamaan linear, dalam bentuk numpy array satu dimensi.

```
det_A = np.linalg.det(A)
```

Determinan matriks A dihitung menggunakan fungsi np.linalg.det dari library NumPy. Determinan digunakan untuk memeriksa apakah matriks A memiliki invers atau tidak.

```
if det_A == 0:
    print("Matriks A tidak memiliki invers.")
    return None
```

Jika determinan matriks A sama dengan nol, maka matriks A tidak memiliki invers. Dalam kasus ini, fungsi akan mencetak pesan "Matriks A tidak memiliki invers." dan mengembalikan None (tidak ada solusi).

```
A_inv = np.linalg.inv(A)
```

Jika determinan matriks A tidak sama dengan nol, maka matriks A memiliki invers. Invers matriks A dihitung menggunakan fungsi np.linalg.inv dari library NumPy.

```
x = np.dot(A_inv, b)
```

Solusi x dari sistem persamaan linear dihitung dengan mengalikan invers matriks A dengan vektor b menggunakan operasi perkalian matriks np.dot dari library NumPy.

```
return x
```

Solusi x dikembalikan sebagai hasil dari fungsi.

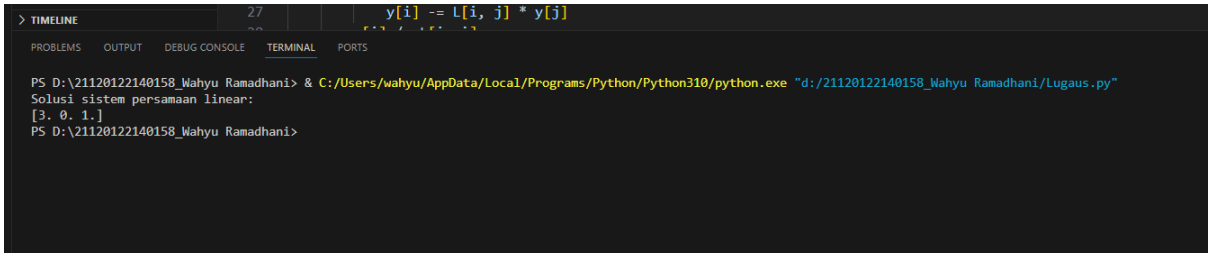
```
A = np.array([[1, -1, 2], [3, 0, 1], [1, 0, 2]])
b = np.array([5, 10, 5])
solusi = solusi_persamaan_linear(A, b)
if solusi is not None:
    print("Solusi sistem persamaan linear:")
    print(solusi)
```

Bagian ini memberikan contoh penggunaan fungsi solusi_persamaan_linear dengan memasukkan matriks koefisien A dan vektor konstanta b. Hasil solusi x akan dicetak jika matriks A memiliki invers.

2. Metode Dekomposisi LU Gauss

```
import numpy as np

def lu_decomposition(A):
    n = A.shape[0]
    L = np.eye(n, dtype=np.double)
    U = A.copy()

    for k in range(n-1):
        if U[k, k] == 0.0:
            raise ValueError("Matriks A tidak memiliki solusi unik atau
   sistem persamaan linear tidak konsisten.")
```

```
        for i in range(k+1, n):
            if U[i, k] != 0.0:
                lam = U[i, k] / U[k, k]
                L[i, k] = lam
                U[i, k:n] -= lam * U[k, k:n]

    return L, U

def forward_substitution(L, b):
    n = L.shape[0]
    y = np.zeros_like(b, dtype=np.double)

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

    return y

def backward_substitution(U, y):
    n = U.shape[0]
    x = np.zeros_like(y, dtype=np.double)

    for i in range(n-1, -1, -1):
        if U[i, i] == 0.0:
            raise ValueError("Matriks A tidak memiliki solusi unik atau
   sistem persamaan linear tidak konsisten.")

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

    return x

def solve_linear_system(A, b):
    try:
        L, U = lu_decomposition(A)
        y = forward_substitution(L, b)
        x = backward_substitution(U, y)
        return x
    except ValueError as e:
        print(e)
        return None

if __name__ == '__main__':
    A = np.array([[1, -1, 2], [3, 0, 1], [1, 0, 2]], dtype=np.double)
    b = np.array([5, 10, 5], dtype=np.double)

    x = solve_linear_system(A, b)
    if x is not None:
        print("Solusi sistem persamaan linear:")
        print(x)
```



Penjelasan :

```
import numpy as np
```

Baris ini mengimpor modul NumPy, yang digunakan untuk operasi numerik dan manipulasi array dalam Python.

```
def lu_decomposition(A):
```

Ini adalah definisi fungsi lu_decomposition yang menerima matriks koefisien A sebagai argumen.

```
n = A.shape[0]
```

Baris ini mendapatkan jumlah baris (atau kolom) dari matriks A dan menyimpannya dalam variabel n.

```
      L = np.eye(n, dtype=np.double)
U = A.copy()
```

Dua baris ini menginisialisasi matriks L sebagai matriks identitas berukuran n x n dengan tipe data double, dan matriks U sebagai salinan dari matriks A.

```
      for k in range(n-1):
          if U[k, k] == 0.0:
        raise ValueError("Matriks A tidak memiliki solusi unik atau sistem
persamaan linear tidak konsisten.")
```

Loop ini iterasi melalui diagonal utama matriks U. Jika elemen diagonal U[k, k] sama dengan nol, maka dilempar ValueError yang menunjukkan bahwa matriks A tidak memiliki solusi unik atau sistem persamaan linear tidak konsisten.

```
      for i in range(k+1, n):
          if U[i, k] != 0.0:
              lam = U[i, k] / U[k, k]
              L[i, k] = lam
        U[i, k:n] -= lam * U[k, k:n]
```

Loop bersarang ini melakukan operasi baris pada matriks U untuk menghasilkan matriks segitiga atas U dan matriks segitiga bawah L. Jika elemen U[i, k] tidak sama dengan nol, maka dihitung faktor lam yang digunakan untuk mengupdate elemen L[i, k] dan mengurangi baris U[i, k:n] dengan lam kali baris U[k, k:n].

```
return L, U
```

Baris ini mengembalikan matriks L dan U hasil dari dekomposisi LU.

```
def forward_substitution(L, b):
```

Ini adalah definisi fungsi forward_substitution yang menerima matriks segitiga bawah L dan vektor konstanta b sebagai argumen.

```
      n = L.shape[0]
y = np.zeros_like(b, dtype=np.double)
```

Baris ini mendapatkan ukuran matriks L dan menginisialisasi vektor y sebagai vektor nol dengan tipe data double dan ukuran yang sama dengan b.

```
      for i in range(n):
          y[i] = b[i]
          for j in range(i):
              y[i] -= L[i, j] * y[j]
    y[i] /= L[i, i]
```

Loop ini melakukan substitusi maju untuk menghitung elemen-elemen vektor y dari persamaan Ly = b. Setiap elemen y[i] dihitung dengan mengurangi b[i] dengan kombinasi linear dari elemen-elemen L dan y yang telah dihitung sebelumnya, lalu membaginya dengan elemen diagonal L[i, i].

```
return y
```

Baris ini mengembalikan vektor y hasil dari substitusi maju.

```
def backward_substitution(U, y):
```

Ini adalah definisi fungsi backward_substitution yang menerima matriks segitiga atas U dan vektor y sebagai argumen.

```
      n = U.shape[0]
x = np.zeros_like(y, dtype=np.double)
```

Baris ini mendapatkan ukuran matriks U dan menginisialisasi vektor solusi x sebagai vektor nol dengan tipe data double dan ukuran yang sama dengan y.

```
      for i in range(n-1, -1, -1):
          if U[i, i] == 0.0:
              raise ValueError("Matriks A tidak memiliki solusi unik atau
      sistem persamaan linear tidak konsisten.")
          x[i] = y[i]
          for j in range(i+1, n):
              x[i] -= U[i, j] * x[j]
    x[i] /= U[i, i]
```

Loop ini melakukan substitusi mundur untuk menghitung elemen-elemen vektor solusi x dari persamaan Ux = y. Jika elemen diagonal U[i, i] sama dengan nol, maka dilempar ValueError yang menunjukkan bahwa matriks A tidak memiliki solusi unik atau sistem persamaan linear

tidak konsisten. Setiap elemen x[i] dihitung dengan mengurangi y[i] dengan kombinasi linear dari elemen-elemen U dan x yang telah dihitung sebelumnya, lalu membaginya dengan elemen diagonal U[i, i].

```
return x
```

Baris ini mengembalikan vektor solusi x hasil dari substitusi mundur.

```
    def solve_linear_system(A, b):
        try:
            L, U = lu_decomposition(A)
            y = forward_substitution(L, b)
            x = backward_substitution(U, y)
            return x
        except ValueError as e:
            print(e)
      return None
```

Fungsi solve_linear_system menerima matriks koefisien A dan vektor konstanta b sebagai argumen. Fungsi ini mencoba melakukan dekomposisi LU pada matriks A menggunakan lu_decomposition(A), kemudian melakukan substitusi maju pada L dan b menggunakan forward_substitution(L, b) untuk mendapatkan vektor y, lalu melakukan substitusi mundur pada U dan y menggunakan backward_substitution(U, y) untuk mendapatkan vektor solusi x. Jika terjadi ValueError pada tahap dekomposisi LU atau substitusi mundur, maka pesan error akan dicetak dan fungsi mengembalikan None. Jika tidak ada error, maka vektor solusi x dikembalikan.

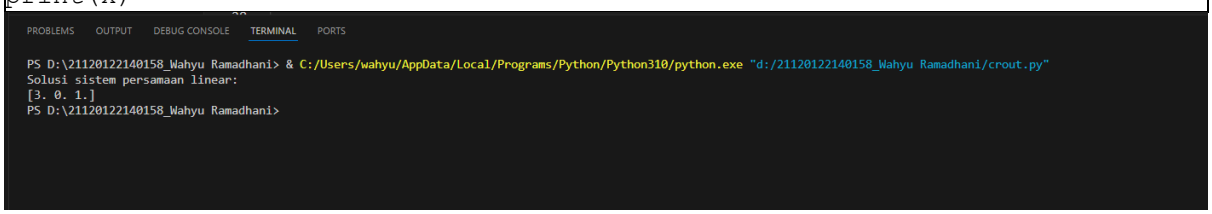
```
    if __name__ == '__main__':
        A = np.array([[1, -1, 2], [3, 0, 1], [1, 0, 2]], dtype=np.double)
        b = np.array([5, 10, 5], dtype=np.double)
        x = solve_linear_system(A, b)
        if x is not None:
            print("Solusi sistem persamaan linear:")
      print(x)
```

Bagian ini merupakan bagian utama (main) dari program. Jika file ini dijalankan secara langsung, maka kode ini akan dieksekusi. Matriks koefisien A dan vektor konstanta b didefinisikan sebagai contoh. Fungsi solve_linear_system(A, b) dipanggil untuk menyelesaikan sistem persamaan linear Ax = b. Jika vektor solusi x tidak None (tidak terjadi error), maka vektor solusi x akan dicetak ke layar.

Dengan demikian, kode ini mengimplementasikan dekomposisi LU, substitusi maju, dan substitusi mundur untuk menyelesaikan sistem persamaan linear Ax = b. Fungsi utama `solve_

3. Metode Dekomposisi Crout

```
import numpy as np

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

    # Dekomposisi Crout
    for j in range(n):
        U[0][j] = A[0][j]

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

    for i in range(1, n):
        for j in range(1, n):
            if i > j:
                L[i][j] = A[i][j] - sum(L[i][k] * U[k][j] for k in range(j))
            else:
                U[i][j] = A[i][j] - sum(L[i][k] * U[k][j] for k in range(i))

    # Solusi dengan substitusi maju dan substitusi mundur
    y = np.zeros(n)
    y[0] = b[0] / L[0][0]

    for i in range(1, n):
        y[i] = (b[i] - sum(L[i][j] * y[j] for j in range(i))) / L[i][i]

    x = np.zeros(n)
    x[n-1] = y[n-1] / U[n-1][n-1]

    for i in range(n-2, -1, -1):
        x[i] = (y[i] - sum(U[i][j] * x[j] for j in range(i+1, n))) / U[i][i]

    return x

# Contoh penggunaan
A = np.array([[1, -1, 2],
              [3, 0, 1],
              [1, 0, 2]])
b = np.array([5, 10, 5])
x = crout_decomposition(A, b)
print("Solusi sistem persamaan linear:")
print(x)
```



Penjelasan :

```
import numpy as np
```

Baris ini mengimpor modul NumPy, yang digunakan untuk operasi numerik dan manipulasi array dalam Python.

```
def crout_decomposition(A, b):
```

Ini adalah definisi fungsi crout_decomposition yang menerima dua argumen: A (matriks koefisien) dan b (vektor konstanta) dari sistem persamaan linear yang ingin diselesaikan.

```
n = len(A)
```

Baris ini mendapatkan ukuran matriks A (jumlah baris atau kolom) dan menyimpannya dalam variabel n.

```
L = np.eye(n)
U = np.zeros((n, n))
```

Kedua baris ini menginisialisasi matriks L sebagai matriks identitas berukuran n x n menggunakan np.eye(n), dan matriks U sebagai matriks nol berukuran n x n menggunakan np.zeros((n, n)).

```
for j in range(n):
U[0][j] = A[0][j]
```

Loop ini mengisi elemen-elemen pada baris pertama matriks U dengan elemen-elemen yang sesuai dari baris pertama matriks A.

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

Loop ini mengisi elemen-elemen pada kolom pertama matriks L dengan membagi elemen-elemen kolom pertama matriks A dengan elemen pertama U[0][0].

```
for i in range(1, n):
       for j in range(1, n):
            if i > j:
                 L[i][j] = A[i][j] - sum(L[i][k] * U[k][j] for k
     irange(j))
           else:
            U[i][j] = A[i][j] - sum(L[i][k] * U[k][j] for k in range(i))
```

Dua loop bersarang ini melakukan dekomposisi Crout untuk menghitung elemen-elemen lainnya pada matriks L dan U. Jika i > j, maka dihitung elemen L[i][j] di bawah diagonal utama dengan mengurangi A[i][j] dengan kombinasi linear dari elemen-elemen L dan U yang telah dihitung sebelumnya. Jika i <= j, maka dihitung elemen U[i][j] di atas atau pada diagonal utama dengan mengurangi A[i][j] dengan kombinasi linear dari elemen-elemen L dan U yang telah dihitung sebelumnya.

```
y = np.zeros(n)
y[0] = b[0] / L[0][0]
```

Dua baris ini menginisialisasi vektor y sebagai vektor nol, kemudian menghitung elemen pertama y[0] dengan membagi b[0] dengan elemen pertama L[0][0].

```
for i in range(1, n):
y[i] = (b[i] - sum(L[i][j] * y[j] for j in range(i))) / L[i][i]
```

Loop ini melakukan substitusi maju untuk menghitung elemen-elemen lainnya pada vektor y dengan mengurangi b[i] dengan kombinasi linear dari elemen-elemen L dan y yang telah dihitung sebelumnya, lalu membaginya dengan elemen diagonal L[i][i].

```
x = np.zeros(n)
x[n-1] = y[n-1] / U[n-1][n-1]
```

Dua baris ini menginisialisasi vektor solusi x sebagai vektor nol, kemudian menghitung elemen terakhir x[n-1] dengan membagi y[n-1] dengan elemen diagonal terakhir U[n-1][n-1].

```
      for i in range(n-2, -1, -1):
    x[i] = (y[i] - sum(U[i][j] * x[j] for j in range(i+1, n))) / U[i][i]
```

Loop ini melakukan substitusi mundur untuk menghitung elemen-elemen lainnya pada vektor solusi x dengan mengurangi y[i] dengan kombinasi linear dari elemen-elemen U dan x yang telah dihitung sebelumnya, lalu membaginya dengan elemen diagonal U[i][i].

```
return x
```

Baris ini mengembalikan vektor solusi x sebagai hasil dari fungsi crout_decomposition.

```
      A = np.array([[1, -1, 2], [3, 0, 1], [1, 0, 2]])
      b = np.array([5, 10, 5])
      x = crout_decomposition(A, b)
      print("Solusi sistem persamaan linear:")
print(x)
```

Bagian ini memberikan contoh penggunaan fungsi crout_decomposition dengan matriks koefisien A dan vektor konstanta b yang diberikan. Vektor solusi x dari sistem persamaan linear Ax = b dihitung dengan memanggil fungsi crout_decomposition(A, b), dan hasilnya dicetak ke layar menggunakan print(x).

4. Metode dekomposisi crout