Nama : Mazroha Anis Sugesti

NIM : 21120122120020

Matkul : Metode Numerik / C

Tugas :

Diinginkan aplikasi untuk mencari solusi sistem persamaan linear masing-masing menggunakan:

1. Metode matriks balikan
2. Metode dekomposisi LU Gauss
3. Metode dekomposisi Crout

Pembahasan :

1. *Source Code* persamaan linear dengan metode matriks balikan

|  |
| --- |
| import numpy as np  # Langkah 1: Mendefinisikan matriks koefisien A dan vektor konstanta B  A = np.array([[2, 3, -1],                [1, -2, 2],                [3, 2, -4]])  B = np.array([7, 3, 1]) |

Mengimpor pustaka numpy dan memberi alias sebagai 'np', yang merupakan konvensi umum. Lalu mendefinisikan matriks koefisien A dan vektor konstanta B dari sistem persamaan linear. Matriks A adalah matriks koefisien yang berisi koefisien-koefisien variabel, dan vektor B adalah vektor konstanta pada sisi kanan sistem persamaan linear.

|  |
| --- |
| # Menampilkan matriks koefisien A dan vektor konstanta B  print("Langkah 1: Matriks koefisien A dan vektor konstanta B")  print("Matriks koefisien A:")  print(A)  print("\nVektor konstanta B:")  print(B)  # Langkah 2: Mencari invers dari matriks A  A\_inv = np.linalg.inv(A)  # Menampilkan invers dari matriks A  print("\nLangkah 2: Mencari invers dari matriks A")  print("Invers dari matriks A:")  print(A\_inv) |

Langkah 1 untuk menampilkan matriks koefisien A dan vektor B untuk validasi. Langkah 2 menggunakan fungsi np.linalg.inv() dari Numpy untuk menghitung invers dari matriks A, dan mencetak invers dari matriks A

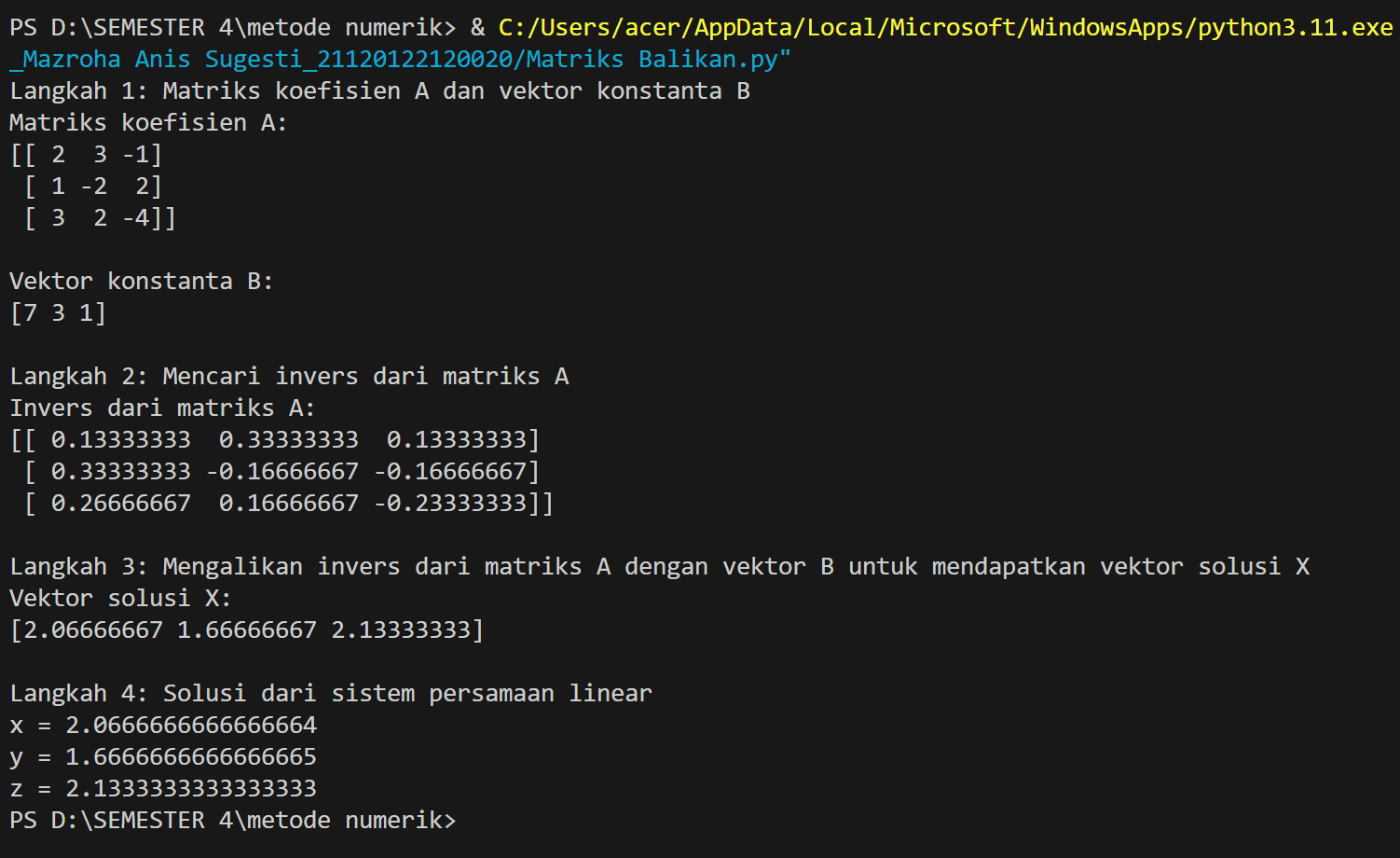
|  |
| --- |
| # Langkah 3: Mengalikan invers dari matriks A dengan vektor B untuk mendapatkan vektor solusi X  X = np.dot(A\_inv, B)  # Menampilkan vektor solusi X  print("\nLangkah 3: Mengalikan invers dari matriks A dengan vektor B untuk mendapatkan vektor solusi X")  print("Vektor solusi X:")  print(X) |

Langkah 3 Ini mengalikan invers dari matriks A dengan vektor B menggunakan fungsi np.dot() untuk mendapatkan solusi dari sistem persamaan linear, dan mencetak Solusi X

|  |
| --- |
| # Menampilkan solusi dari sistem persamaan linear  print("\nLangkah 4: Solusi dari sistem persamaan linear")  print("x =", X[0])  print("y =", X[1])  print("z =", X[2]) |

Langkah ke 4 mencetaksolusi dari sistem persamaan linear dalam bentuk x, y, dan z.

*Output:*



1. *Source Code* persamaan linear dengan metode dekomposisi LU Gauss

|  |
| --- |
| 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):  L[i][i] = 1 # Mengisi bagian diagonal L dengan 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 |

Mengimpor pustaka numpy dan memberi alias sebagai 'np', yang akan digunakan untuk operasi matriks. Pertama, matriks L dan U dibuat dengan ukuran yang sama dengan matriks masukan, diinisialisasi dengan nilai nol. Kemudian, dilakukan iterasi melalui baris matriks. Di dalam loop, elemen diagonal matriks L diatur menjadi 1.Selanjutnya, elemen-elemen matriks U dihitung menggunakan metode eliminasi Gauss. Kemudian, elemen-elemen matriks L dihitung berdasarkan elemen-elemen U yang telah dihitung sebelumnya.

|  |
| --- |
| def solve\_lu\_decomposition(A, b):  L, U = lu\_decomposition\_gauss(A)  n = len(A)  y = np.zeros(n) # Substitusi maju untuk mencari y  for i in range(n):  y[i] = (b[i] - np.dot(L[i, :i], y[:i])) / L[i, i]  x = np.zeros(n) # Substitusi mundur untuk mencari x  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 |

Fungsi ini menerima matriks koefisien A dan vektor konstanta b, dan mengembalikan solusi sistem persamaan linear. Fungsi lu\_decomposition\_gauss() dipanggil untuk mendapatkan matriks L dan U. Dilakukan substitusi maju untuk mencari vektor y dengan memecahkan matriks Lb. Selanjutnya, dilakukan substitusi mundur untuk mencari solusi x dengan memecahkan matriks Uy.

|  |
| --- |
| A = np.array([[4, -2, 1], [-2, 5, 3], [1, 3, 6]])  b = np.array([8, 3, 9]) |

Mendefinisikan matriks koefisien A dan vektor konstanta b yang merupakan bagian dari sistem persamaan linear yang akan diselesaikan.

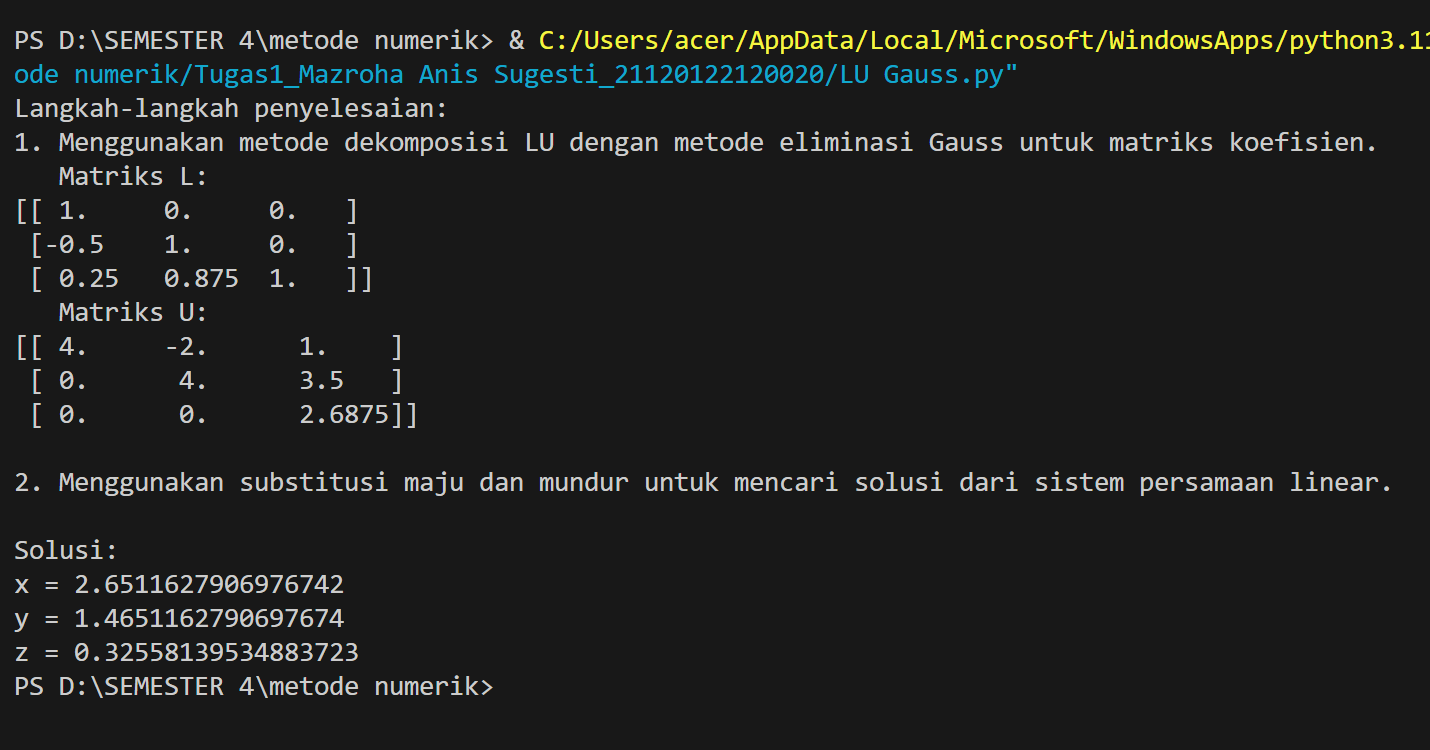
|  |
| --- |
| print("Langkah-langkah penyelesaian:")  print("1. Menggunakan metode dekomposisi LU dengan metode eliminasi Gauss untuk matriks koefisien.")  L, U = lu\_decomposition\_gauss(A)  print(" Matriks L:")  print(L)  print(" Matriks U:")  print(U)  print("\n2. Menggunakan substitusi maju dan mundur untuk mencari solusi dari sistem persamaan linear.") |

fungsi lu\_decomposition\_gauss() untuk mendapatkan matriks L dan U, dan kemudian disusul dengan pemanggilan fungsi solve\_lu\_decomposition() untuk mencari solusi sistem persamaan linear.

|  |
| --- |
| solution = solve\_lu\_decomposition(A, b)  print("\nSolusi:")  print("x =", solution[0])  print("y =", solution[1])  print("z =", solution[2]) |

Mencetak solusi dari sistem persamaan linear yang telah ditemukan.

*Output:*



1. *Source Code* persamaan linear dengan metode dekomposisi Crout

|  |
| --- |
| import numpy as np  A = np.array([[2, 3, 1],  [4, 7, 2],  [-2, 5, 2]])  B = np.array([10, 23, 4]) |

Mendefinisikan matriks koefisien A dan vektor konstanta B yang membentuk matriks augmented.

|  |
| --- |
| def crout\_decomposition(A):  n = len(A)  L = np.zeros((n, n))  U = np.zeros((n, n))  for j in range(n):  for i in range(j, n):  if i == j:  L[i][j] = 1  else:  L[i][j] = A[i][j] - sum(L[i][k] \* U[k][j] for k in range(j))  for i in range(j, n):  if i == j:  U[i][j] = A[i][j] - sum(L[i][k] \* U[k][j] for k in range(j))  else:  U[j][i] = (A[j][i] - sum(L[j][k] \* U[k][i] for k in range(j))) / L[j][j]  return L, U  L, U = crout\_decomposition(A) |

Fungsi crout\_decomposition ini menerima matriks koefisien 𝐴 dan mengembalikan matriks 𝐿 dan 𝑈 dari dekomposisi Crout. Matriks 𝐿 dan 𝑈 diinisialisasi dengan nol. Dilakukan iterasi melalui kolom matriks. Di dalam loop, dihitung nilai untuk matriks 𝐿 dan 𝑈 sesuai dengan aturan dekomposisi Crout.

|  |
| --- |
| def forward\_substitution(L, B):  n = len(B)  y = np.zeros(n)  for i in range(n):  y[i] = (B[i] - np.dot(L[i, :i], y[:i])) / L[i, i]  return y  y = forward\_substitution(L, B) |

Fungsi forward\_substitution ini melakukan substitusi maju untuk mencari solusi dari sistem persamaan linear 𝐿𝑦=𝐵Ly=B, dimana 𝐿L adalah matriks segitiga bawah. Ini dilakukan dengan mencari nilai 𝑦y yang memenuhi persamaan tersebut.

|  |
| --- |
| def backward\_substitution(U, y):  n = len(y)  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  x = backward\_substitution(U, y) |

Fungsi backward\_substitution ini melakukan substitusi mundur untuk mencari solusi dari sistem persamaan linear Ux=y, dimana 𝑈 adalah matriks segitiga atas. Ini dilakukan dengan mencari nilai 𝑥 yang memenuhi persamaan tersebut.

|  |
| --- |
| print("Langkah 1: Matriks augmented [A|B]:")  print(np.column\_stack((A, B)))  print("\nLangkah 2: Dekomposisi Crout:")  print("Matriks L:")  print(L)  print("Matriks U:")  print(U)  print("\nLangkah 3: Menyelesaikan Ly = B untuk mencari y:")  print("Nilai y:")  print(y)  print("\nLangkah 4: Menyelesaikan Ux = y untuk mencari x:")  print("Nilai x:")  print(x) |

Mencetak langkah-langkah penyelesaian, termasuk matriks augmented, matriks L dan U hasil dekomposisi Crout, nilai y hasil substitusi maju, dan nilai x hasil substitusi mundur.

*Output:*

