Nama : Edi Wicoro

NIM : 21120122130073

Mata Kuliah : Metode Numerik B

## 1. Metode Matriks Balikan

|  |
| --- |
| # Nama : Edi Wicoro  # NIM : 21120122130073  # Kelas : Metode Numerik B / 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 |

Dua library yang diimpor adalah NumPy untuk operasi matriks dan unittest untuk pengujian unit.

1. Fungsi ‘invers\_matrix(matrix)’

|  |
| --- |
| def inverse\_matrix(matrix):  try:  inverse = np.linalg.inv(matrix)  return inverse  except np.linalg.LinAlgError:  return None |

Ini adalah fungsi yang menerima input berupa matriks dan mengembalikan matriks balikan menggunakan fungsi np.linalg.inv() dari NumPy. Jika matriks tersebut singular (tidak memiliki balikan), fungsi akan menangkap LinAlgError dan mengembalikan None.

1. 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) |

Determinan matriks A dihitung terlebih dahulu. Jika determinan A tidak sama dengan 0 (artinya matriks A memiliki balikan), maka matriks adjoin A dihitung terlebih dahulu dengan dikalikan dengan determinan. Selanjutnya, fungsi inverse\_matrix() dipanggil untuk mencari matriks balikan A.

1. 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)) |

Ini adalah definisi kelas pengujian unit. Terdapat dua metode pengujian, yaitu test\_inverse() untuk matriks yang memiliki balikan, dan test\_singular\_matrix() untuk matriks yang tidak memiliki balikan.

## 2. Metode Dekomposisi LU Gausss

|  |
| --- |
| # Nama : Edi Wicoro  # NIM : 21120122130073  # Kelas : Metode Numerik B / Teknik Komputer  import numpy as np  # 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("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.")  # Menyelesaikan sistem persamaan linear  solution = solve\_lu\_decomposition(A, b)  print("\nSolusi:")  print("x =", solution[0])  print("y =", solution[1])  print("z =", solution[2]) |

**Penjelasan Kode**

1. Import Library

|  |
| --- |
| import numpy as np |

NumPy digunakan untuk operasi matriks dan vektor dalam penyelesaian sistem persamaan linear.

1. 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 |

Ini adalah fungsi yang mengimplementasikan dekomposisi LU dengan menggunakan metode eliminasi Gauss. Fungsi ini menerima matriks koefisien sebagai input dan mengembalikan matriks L (lower triangular) dan U (upper triangular).

1. 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 |

Ini adalah fungsi untuk menyelesaikan sistem persamaan linear menggunakan dekomposisi LU. Fungsi ini menerima matriks koefisien A dan vektor hasil b sebagai input, dan mengembalikan vektor solusi.

1. 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.

1. 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.

## 3. Metode Dekomposisi Crout

|  |
| --- |
| # Nama : Edi Wicoro  # NIM : 21120122130073  # Kelas : Metode Numerik B / 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 |

NumPy digunakan untuk operasi matriks, sedangkan unittest digunakan untuk melakukan pengujian unit.

1. 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 |

Fungsi ini mengimplementasikan algoritma dekomposisi Crout. Fungsi ini menerima matriks A sebagai input dan mengembalikan matriks segitiga bawah (L) dan matriks segitiga atas (U).

1. Contoh Penggunaan

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

Ini adalah matriks A yang akan dipecah menggunakan dekomposisi Crout

1. 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.

1. Cetak Hasil

|  |
| --- |
| print("Matrix L:")  print(L)  print("Matrix U:")  print(U) |

Matriks segitiga bawah (L) dan matriks segitiga atas (U) dicetak untuk melihat hasil dekomposisi.

1. 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) |

Kelas ini berisi metode pengujian unit untuk memastikan bahwa fungsi crout\_decomposition() memberikan hasil yang benar. Dalam metode test\_decomposition(), matriks L dan U yang dihasilkan dibandingkan dengan matriks yang diharapkan.