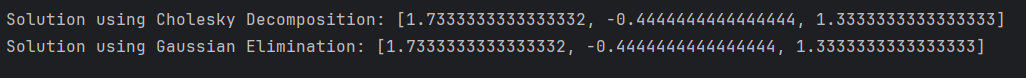
from lab2\_gauss\_method import gaussian\_elimination  
  
  
# Custom square root function using Newton's method  
def custom\_sqrt(x, tolerance=1e-10):  
 if x < 0:  
 raise ValueError("Cannot compute square root of a negative number")  
 if x == 0:  
 return 0  
 guess = x  
 while True:  
 new\_guess = 0.5 \* (guess + x / guess)  
 if abs(new\_guess - guess) < tolerance:  
 return new\_guess  
 guess = new\_guess  
  
  
# Check if matrix is symmetric  
def is\_symmetric(matrix):  
 n = len(matrix)  
 for i in range(n):  
 for j in range(n):  
 if matrix[i][j] != matrix[j][i]:  
 return False  
 return True  
  
  
# Cholesky decomposition to decompose A into L \* L^T  
def cholesky\_decomposition(matrix):  
 n = len(matrix)  
 L = [[0.0] \* n for \_ in range(n)] # Initialize an empty matrix for L  
  
 for i in range(n):  
 for j in range(i + 1):  
 if i == j:  
 # Diagonal elements  
 sum\_k = sum(L[i][k] \*\* 2 for k in range(j))  
 L[i][j] = custom\_sqrt(matrix[i][i] - sum\_k)  
 else:  
 # Off-diagonal elements  
 sum\_k = sum(L[i][k] \* L[j][k] for k in range(j))  
 L[i][j] = (matrix[i][j] - sum\_k) / L[j][j]  
  
 return L  
  
  
def forward\_substitution(L, B):  
 n = len(L)  
 y = [0] \* n  
  
 for i in range(n):  
 sum\_k = sum(L[i][k] \* y[k] for k in range(i))  
 y[i] = (B[i] - sum\_k) / L[i][i]  
  
 return y  
  
  
def backward\_substitution(L, y):  
 n = len(L)  
 x = [0] \* n  
  
 for i in range(n - 1, -1, -1):  
 sum\_k = sum(L[k][i] \* x[k] for k in range(i + 1, n))  
 x[i] = (y[i] - sum\_k) / L[i][i]  
  
 return x  
  
  
def solve\_using\_cholesky(A, B):  
 if not is\_symmetric(A):  
 raise ValueError("Matrix A is not symmetric")  
  
 # Step 1: Perform Cholesky decomposition to get L  
 L = cholesky\_decomposition(A)  
  
 # Step 2: Solve Ly = B (forward substitution)  
 y = forward\_substitution(L, B)  
  
 # Step 3: Solve L^T x = y (backward substitution)  
 x = backward\_substitution(L, y)  
  
 return x  
  
  
A = [  
 [25, 15, -5],  
 [15, 18, 0],  
 [-5, 0, 11]  
]  
  
B = [30, 18, 6]  
  
cholesky\_solution = solve\_using\_cholesky([row[:] for row in A], B[:])  
print("Solution using Cholesky Decomposition:", cholesky\_solution)  
gaussian\_solution = gaussian\_elimination([row[:] for row in A], B[:])  
print("Solution using Gaussian Elimination:", gaussian\_solution)

2. 

3. 