Лабораторная работа №1

1.1. Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.



*par\_and\_fun/Matrix.py* (только используемые функции)

def read\_matrix():

n, m = map(int, input("Введите 'n' и 'm': ").split())

print("Введите матрицу")

matrix = [[float(j) for j in input().split()] for \_ in range(n)]

if any(filter(lambda x: len(x) != m, matrix)):

raise ValueError("Количество элементов в строке матрицы не соответствует ожидаемому")

return matrix

def print\_matrix(matrix):

for i in matrix:

for j in i:

print(f"{j:.4f}", end=" ")

print()

def read\_vector():

n = int(input("Введите длину вектора n: "))

print("Введите значения вектора, каждый элемент вектора на новой строке")

vector = [[float(input())] for \_ in range(n)]

return vector

def transpose(matrix):

n, m = len(matrix), len(matrix[0])

result = [[matrix[j][i] for j in range(n)] for i in range(m)]

return result

def take\_column(matrix, index):

result = [[i[index]] for i in matrix]

return result

def matrix\_multiplication(matrix\_1, matrix\_2):

if len(matrix\_1[0]) != len(matrix\_2):

raise ValueError("Количество столбцов в первой матрице должно быть равно количеству строк второй матрице!")

result = [[0 for \_ in matrix\_2[0]] for \_ in matrix\_1]

for i in range(len(matrix\_1)):

for j in range(len(matrix\_2[0])):

for k in range(len(matrix\_2)):

result[i][j] += matrix\_1[i][k] \* matrix\_2[k][j]

if abs(result[i][j]) < 1e-12:

result[i][j] = 0.0

return result

def multiplying\_matrix\_number(number, matrix):

result = [[matrix[i][j] \* number for j in range(len(matrix[0]))] for i in range(len(matrix))]

return result

def matrix\_addition(matrix\_1, matrix\_2):

if len(matrix\_1) != len(matrix\_2) or len(matrix\_1[0]) != len(matrix\_2[0]):

raise ValueError("Матрицы должны быть одинакового размера!")

result = [[matrix\_1[i][j] + matrix\_2[i][j] for j in range(len(matrix\_1[0]))] for i in range(len(matrix\_1))]

return result

def matrix\_difference(matrix\_1, matrix\_2):

result = matrix\_addition(matrix\_1, multiplying\_matrix\_number(-1, matrix\_2))

return result

*Laba1/Laba1-1.py*

from par\_and\_func import matrix\_multiplication, read\_matrix, read\_vector, transpose, print\_matrix

def lup(matrix):

n = len(matrix)

if n <= 0:

raise ValueError("Матрица вырождена")

count\_swaps = 0

a\_matrix = [row[:] for row in matrix]

l\_matrix = [[0.0] \* n for \_ in range(n)]

u\_matrix = [[0.0] \* n for \_ in range(n)]

p\_matrix = [[1.0 if i == j else 0.0 for j in range(n)] for i in range(n)]

for k in range(n):

max\_row = k

max\_val = abs(a\_matrix[k][k])

for i in range(k + 1, n):

if abs(a\_matrix[i][k]) > max\_val:

max\_val = abs(a\_matrix[i][k])

max\_row = i

if max\_row != k:

count\_swaps += 1

a\_matrix[k], a\_matrix[max\_row] = a\_matrix[max\_row], a\_matrix[k]

p\_matrix[k], p\_matrix[max\_row] = p\_matrix[max\_row], p\_matrix[k]

for j in range(k):

l\_matrix[k][j], l\_matrix[max\_row][j] = l\_matrix[max\_row][j], l\_matrix[k][j]

if abs(a\_matrix[k][k]) < 1e-12:

raise ValueError("Матрица вырождена или близка к вырожденной")

l\_matrix[k][k] = 1.0

for j in range(k, n):

u\_matrix[k][j] = a\_matrix[k][j]

for m in range(k):

u\_matrix[k][j] -= l\_matrix[k][m] \* u\_matrix[m][j]

for i in range(k + 1, n):

l\_matrix[i][k] = a\_matrix[i][k]

for m in range(k):

l\_matrix[i][k] -= l\_matrix[i][m] \* u\_matrix[m][k]

l\_matrix[i][k] /= u\_matrix[k][k]

return l\_matrix, u\_matrix, p\_matrix, count\_swaps

def solve\_equation(matrix, vector):

n = len(matrix)

if (n != len(matrix[0])) or (n != len(vector)):

raise ValueError("Матрица должна быть квадратной и соответствовать количеству не известных")

l\_matrix, u\_matrix, p\_matrix, count\_swaps = lup(matrix)

b = matrix\_multiplication(p\_matrix, vector)

# Ly=b

y = [i[:] for i in b]

for i in range(n):

for j in range(i):

y[i][0] -= y[j][0] \* l\_matrix[i][j]

# Ux = y

x = [i[:] for i in y]

for i in range(n - 1, -1, -1):

for j in range(n - 1, i, -1):

x[i][0] -= x[j][0] \* u\_matrix[i][j]

x[i][0] /= u\_matrix[i][i]

return x

def determinant(matrix):

l\_matrix, u\_matrix, p\_matrix, count\_swaps = lup(matrix)

result = -1 \*\* (count\_swaps % 2)

for i in range(len(matrix)):

result \*= u\_matrix[i][i]

return result

def inverse\_matrix(matrix):

det = determinant(matrix)

if abs(det) < 1e-12:

raise ValueError("Матрица вырожденная")

n = len(matrix)

l\_matrix, u\_matrix, p\_matrix, count\_swap = lup(matrix)

x = [[1.0 if i == j else 0.0 for j in range(n)] for i in range(n)]

for k in range(n):

for i in range(n):

for j in range(i):

x[i][k] -= x[j][k] \* l\_matrix[i][j]

if abs(x[i][k]) < 1e-10:

x[i][k] = 0.0

for k in range(n):

for i in range(n - 1, -1, -1):

for j in range(n - 1, i, -1):

x[i][k] -= x[j][k] \* u\_matrix[i][j]

x[i][k] /= u\_matrix[i][i]

if abs(x[i][k]) < 1e-10:

x[i][k] = 0.0

return matrix\_multiplication(x, p\_matrix)

def main():

matrix = read\_matrix()

n = len(matrix)

if len(matrix) != len(matrix[0]):

raise ValueError("Матрица должна быть квадратной")

b = read\_vector()

if len(b) != n:

raise ValueError("Вектор должен соответствовать количеству столбцов матрицы")

l\_matrix, u\_matrix, p\_matrix, count\_swap = lup(matrix)

print("L")

print\_matrix(l\_matrix)

print("U")

print\_matrix(u\_matrix)

print("p")

print\_matrix(p\_matrix)

result = transpose(solve\_equation(matrix, b))

print("Ответ: ", end="")

for i in range(len(result[0])):

print(f"x{i + 1} {result[0][i]:.4f}", end=" ")

print()

print(f"Определитель: {determinant(matrix):.4f}")

print("Обратная матрица:")

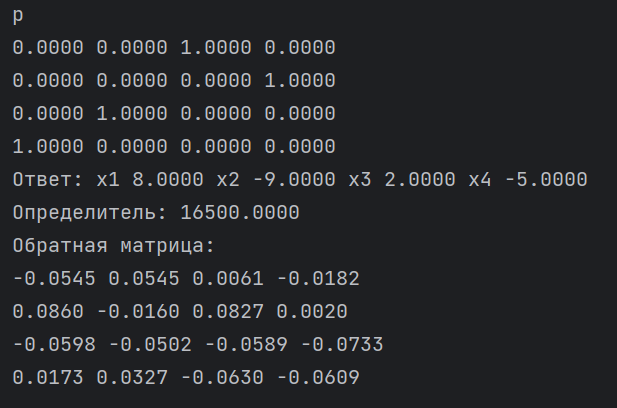
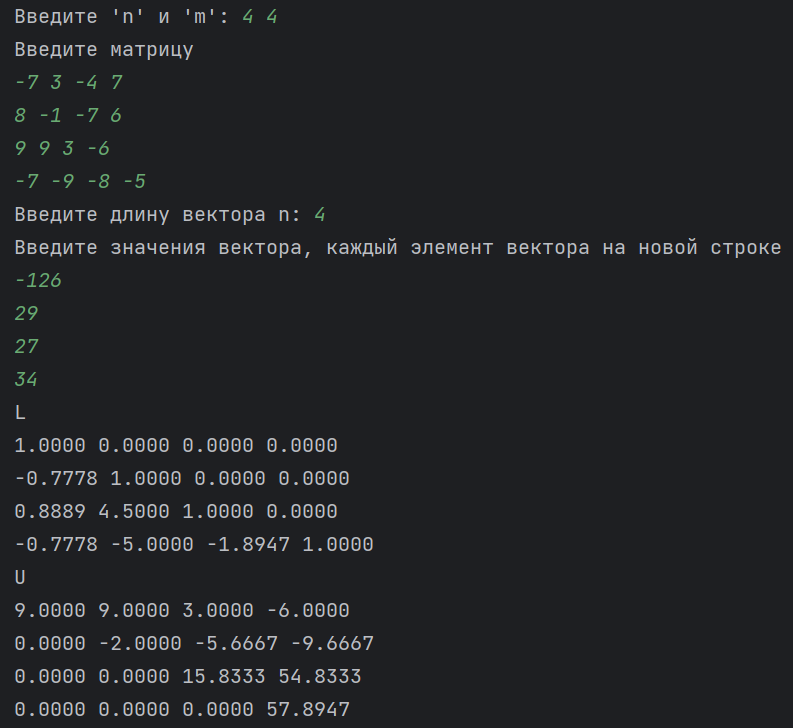
inv\_matrix = inverse\_matrix(matrix)

print\_matrix(inv\_matrix)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Результат задания:



1.2. Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.



*par\_and\_fun/Matrix.py*

def read\_matrix():  
 n, m = map(int, input("Введите 'n' и 'm': ").split())  
 print("Введите матрицу")  
 matrix = [[float(j) for j in input().split()] for \_ in range(n)]  
 if any(filter(lambda x: len(x) != m, matrix)):  
 raise ValueError("Количество элементов в строке матрицы не соответствует ожидаемому")  
 return matrix  
  
def read\_vector():  
 n = int(input("Введите длину вектора n: "))  
 print("Введите значения вектора, каждый элемент вектора на новой строке")  
 vector = [[float(input())] for \_ in range(n)]  
 return vector

*Laba1/Laba1-2.py*

from par\_and\_func import read\_matrix, read\_vector  
  
  
def main():  
 stability = 1  
 matrix = read\_matrix()  
 n = len(matrix)  
 if len(matrix) != len(matrix[0]):  
 raise ValueError("Матрица должна быть квадратной")  
 d = read\_vector()  
 pq = []  
  
 # Прямой ход метода  
 if abs(matrix[0][0]) < 10e-12:  
 raise ZeroDivisionError("Элемент b1 == 0")  
 stability \*= abs(matrix[0][0]) >= abs(matrix[0][1])  
  
 pq.append((-matrix[0][1] / matrix[0][0], d[0][0] / matrix[0][0]))  
 for i in range(1, n - 1):  
 chasn = matrix[i][i] + matrix[i][i - 1] \* pq[i - 1][0]  
 if abs(chasn) < 10e-12:  
 raise ZeroDivisionError("Деление на 0")  
 stability \*= abs(matrix[i][i]) >= (abs(matrix[i][i + 1]) + abs(matrix[i][i - 1]))  
 pq.append((-matrix[i][i + 1] / chasn, (d[i][0] - matrix[i][i - 1] \* pq[i - 1][1]) / chasn))  
 chasn = matrix[n - 1][n - 1] + matrix[n - 1][n - 2] \* pq[n - 2][0]  
 if abs(chasn) < 10e-12:  
 raise ZeroDivisionError("Деление на 0")  
 pq.append((0, (d[n - 1][0] - matrix[n - 1][n - 2] \* pq[n - 2][1]) / chasn))

# Обратный ход метода  
 result = [0] \* n  
  
 result[-1] = pq[-1][1]  
 for i in range(n - 2, -1, -1):  
 result[i] = pq[i][0] \* result[i + 1] + pq[i][1]  
 if not stability:  
 print("Предупреждение. Результат может быть посчитан с большой погрешностью!")  
  
 print("Ответ: ", end=" ")  
 for i in range(1, n + 1):  
 print(f"x{i} = {result[i - 1]}", end=" ")  
 print()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Результат решения данной системы:

