ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНЫХ РАБОТ

ПО ДИСЦИПЛИНЕ «ЧИСЛЕННЫЕ МЕТОДЫ»

ВАРИАНТ ЗАДАНИЯ № 24

Выполнила студент группы М8О-308Б-22

Былькова Кристина Алексеевна_____
подпись, дата

Проверил и принял

Гидаспов В.Ю._____
подпись, дата

с оценкой _____

Москва, 2025

Лабораторная работа № 1.1

Задание: 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.

24. $\begin{cases} -7 \cdot x_1 - 2 \cdot x_2 - x_3 - 4 \cdot x_4 = -12 \\ -4 \cdot x_1 + 6 \cdot x_2 - 4 \cdot x_4 = 22 \\ -8 \cdot x_1 + 2 \cdot x_2 - 9 \cdot x_3 - 3 \cdot x_4 = 51 \\ -7 \cdot x_3 + x_4 = 49 \end{cases}$

Описание решения: сначала строится матрица перестановок P, которая обеспечивает выбор максимального элемента в столбце для улучшения устойчивости метода. Затем выполняется LU-разложение матрицы PA, где L — нижняя треугольная матрица с единицами на диагонали, а U — верхняя треугольная матрица. Решение системы $Ax = b$ находится через решение двух систем: $Ly = Pb$ и $Ux = y$. Определитель матрицы A вычисляется как произведение диагональных элементов U с учетом четности перестановок. Обратная матрица находится путем решения систем для каждого столбца единичной матрицы. Также выполняется проверка путём перемножения исходной матрицы A и вычисленного решения x: в итоге получается вектор b. Также проверяется, верно ли была найдена обратная матрица A^{-1}: $A * A^{-1} = E$.

Текст программы:

```
def permutation_matrix(A):
    n = len(A)
    P = [[1 if i == j else 0 for j in range(n)] for i in range(n)]

    iter_count = 0
    # Находим максимальный элемент
    # Меняем первую строку со строкой с макс эл-том
    for i in range(n):
        max_row = max(range(i, n), key=lambda k: abs(A[k][i]))
        if max_row != i:
            P[i], P[max_row] = P[max_row], P[i]
            iter_count += 1
    return P, iter_count


def LU_decompose(PA):
    n = len(PA)
    # lower
    L = [[0 for _ in range(n)] for _ in range(n)]
    # upper
    U = [row[:] for row in PA]

    for i in range(n):
        # Заполняем матрицы L и U
        L[i][i] = 1
        for j in range(i + 1, n):
            if U[i][i] != 0:
                L[j][i] = U[j][i] / U[i][i]
                for k in range(i, n):
                    U[j][k] -= L[j][i] * U[i][k]

    return L, U

def solve(L, U, b):
    n = len(L)
    # L * y = b
    y = [0 for _ in range(n)]
    for i in range(n):
        y[i] = b[i] - sum(L[i][j] * y[j] for j in range(i)) / L[i][i]

    # U * x = y
    x = [0 for _ in range(n)]
    for i in range(n - 1, -1, -1):
        x[i] = (y[i] - sum(U[i][j] * x[j] for j in range(i + 1, n))) /
U[i][i]
    return x

def transpose(A):
    m = len(A)
    n = len(A[0])
    A_T = [[A[j][i] for j in range(n)] for i in range(m)]
    return A_T

def inverse_matrix(A):
    n = len(A)
    E = [[1 if (i == j) else 0 for j in range(n)] for i in range(n)]

    P, _ = permutation_matrix(A)
    PA = matrix_mult(P, A)
    L, U = LU_decompose(PA)

    A_inv = []
    for i in range(n):
```

```
        Pb = [sum(P[j][k] * E[k][i] for k in range(n)) for j in
range(n)]
        row_inv = solve(L, U, Pb)
        A_inv.append(row_inv)
    return transpose(A_inv)

def determinant(L, U):
    n = len(U)
    det = 1
    for i in range(n):
        det *= U[i][i]
    return det

def matrix_mult(A, B):
    n = len(A)
    return [[sum(A[i][k] * B[k][j] for k in range(n)) for j in
range(n)] for i in range(n)]

def matrix_vector_mult(A, x):
    n = len(A)
    m = len(x)
    return [sum(A[i][k] * x[k] for k in range(m)) for i in range(n)]

def format_matrix(matrix):
    return '\n'.join(' '.join(f"{0.00 if abs(elem) < 1e-10 else
elem:6.2f}" for elem in row) for row in matrix)

def main():
    with open('input.txt', 'r') as f:
        data = [list(map(float, line.split())) for line in
f.readlines()]
    A = data[:-1]
    b = data[-1]

    P, iter_count = permutation_matrix(A)
    PA = matrix_mult(P, A)
    L, U = LU_decompose(PA)

    det_A1 = determinant(L, U) * (-1) ** iter_count
    det_A2 = determinant(L, U)

    A_inv = inverse_matrix(A)

    b_T = [sum(P[i][j] * b[j] for j in range(len(b))) for i in
range(len(b))]
    x = solve(L, U, b_T)
    check = matrix_mult(A, A_inv)


    with open('output.txt', 'w') as f:
        f.write(f"Matrix A:\n{format_matrix(A)}\n\n")
        f.write(f"Vector b:\n{' '.join(f'{elem:6.2f}' for elem in
b)}\n\n")
        f.write(f"Matrix U:\n{format_matrix(U)}\n\n")
        f.write(f"Matrix L:\n{format_matrix(L)}\n\n")
        f.write(f"Solution x:\n{' '.join(f'{elem:6.2f}' for elem in
x)}\n\n")
        f.write(f"Determinant A: {det_A1:6.2f}\n")
        f.write(f"Determinant PA: {det_A2:6.2f}\n\n")
        f.write(f"Inverse matrix A^(-1):\n{format_matrix(A_inv)}\n\n")
        f.write(f"Check A * x = b:\n{' '.join(f'{elem:6.2f}' for elem
in (matrix_vector_mult(A, x)))}\n\n")
        f.write(f"Check A * A^(-1) = E:\n{format_matrix(check)}\n")

if __name__ == "__main__":
    main()
```

Входные данные:

```
-7 -2 -1 -4
-4 6 0 -4
-8 2 -9 -3
0 0 -7 1
-12 22 51 49
```

Результат работы программы:

```
Matrix A:
 -7.00  -2.00  -1.00  -4.00
 -4.00   6.00   0.00  -4.00
 -8.00   2.00  -9.00  -3.00
  0.00   0.00  -7.00   1.00

Vector b:
-12.00  22.00  51.00  49.00

Matrix U:
 -8.00   2.00  -9.00  -3.00
  0.00   5.00   4.50  -2.50
  0.00   0.00  10.25  -3.25
  0.00   0.00   0.00  -1.22

Matrix L:
  1.00   0.00   0.00   0.00
  0.50   1.00   0.00   0.00
  0.88  -0.75   1.00   0.00
  0.00   0.00  -0.68   1.00

Solution x:
  6.00   3.00  -8.00  -7.00

Determinant A: -500.00
Determinant PA: 500.00

Inverse matrix A^(-1):
  0.25   0.24  -0.46   0.56
 -0.21   0.04   0.16  -0.18
 -0.08  -0.06   0.10  -0.26
 -0.56  -0.42   0.70  -0.82

Check A * x = b:
-12.00  22.00  51.00  49.00

Check A * A^(-1) = E:
  1.00   0.00   0.00   0.00
  0.00   1.00   0.00   0.00
  0.00   0.00   1.00   0.00
  0.00   0.00   0.00   1.00
```

Лабораторная работа № 1.2

Задание: 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.

24. $$\begin{cases} -11 \cdot x_1 + 9 \cdot x_2 = -117 \\ -9 \cdot x_1 + 17 \cdot x_2 + 6 \cdot x_3 = -97 \\ 5 \cdot x_2 + 20 \cdot x_3 + 8 \cdot x_4 = -6 \\ -6 \cdot x_3 - 20 \cdot x_4 + 7 \cdot x_5 = 59 \\ 2 \cdot x_4 + 8 \cdot x_5 = -86 \end{cases}$$

Описание решения: сначала из файла считываются данные, представляющие компактную запись трехдиагональной матрицы (нижняя, главная и верхняя диагонали) и вектор правых частей. Затем строится полная матрица системы. Метод прогонки состоит из двух этапов: прямого хода, где вычисляются прогоночные коэффициенты p и q, и обратного хода, в котором находится решение системы. На прямом ходе последовательно выражаются отношения между соседними неизвестными, а на обратном — восстанавливаются значения всех неизвестных, начиная с последнего. После получения решения выполняется проверка путем умножения матрицы на найденный вектор решения. Также выполняется проверка путём перемножения исходной матрицы A и вычисленного решения x: в итоге получается вектор b.

Текст программы:

```
def read_tridiagonal_matrix(filename):
    with open(filename, 'r') as f:
        lines = f.readlines()

    matrix = []
    for line in lines[:-1]:
        row = list(map(float, line.split()))
        matrix.append(row)

    b = list(map(float, lines[-1].split()))

    n = len(matrix)
    A = [[0.0] * n for _ in range(n)]

    for i in range(n):
        if i == 0:
            A[i][i] = matrix[i][0]  # Главная диагональ
            A[i][i + 1] = matrix[i][1]  # Верхняя диагональ
        elif i == n - 1:
            A[i][i - 1] = matrix[i][0]  # Нижняя диагональ
```

```
            A[i][i] = matrix[i][1]  # Главная диагональ
        else:
            A[i][i - 1] = matrix[i][0]  # Нижняя диагональ
            A[i][i] = matrix[i][1]  # Главная диагональ
            A[i][i + 1] = matrix[i][2]  # Верхняя диагональ

    return A, b

def tridiagonal_matrix_algorithm(A, d):
    n = len(d)
    a_diag = [A[i][i - 1] if i > 0 else 0 for i in range(n)]
    b_diag = [A[i][i] for i in range(n)]
    c_diag = [A[i][i + 1] if i < n - 1 else 0 for i in range(n)]


    p = [0 for _ in range(n)]
    q = [0 for _ in range(n)]

    # Задаем начальные значения прогоночных коэффициентов
    p[0] = A[0][1] / -A[0][0]
    q[0] = d[0] / A[0][0]

    # Прямой ход
    for i in range(1, n - 1):
        p[i] = -c_diag[i] / (a_diag[i] * p[i - 1] + b_diag[i])
        q[i] = (d[i] - a_diag[i] * q[i - 1]) / (a_diag[i] * p[i-1] +
b_diag[i])

    q[n - 1] = (d[n - 1] - a_diag[n - 1] * q[n - 2]) / (b_diag[n - 1]
+ a_diag[n - 1] * p[n - 2])

    x = [0 for _ in range(n)]
    x[n - 1] = q[n - 1]

    # Обратный ход
    for i in range(n - 1, 0, -1):
        x[i - 1] = p[i - 1] * x[i] + q[i - 1]
    return x

def matrix_vector_mult(A, x):
    n = len(A)
    m = len(x)
    return [sum(A[i][k] * x[k] for k in range(m)) for i in range(n)]

def format_matrix(matrix):
    return '\n'.join(' '.join(f"{0.00 if abs(elem) < 1e-10 else
elem:6.2f}" for elem in row) for row in matrix)

def main():
    A, b = read_tridiagonal_matrix('input.txt')

    x = tridiagonal_matrix_algorithm(A, b)

    with open('output.txt', 'w') as f:
        f.write(f"Matrix A:\n{format_matrix(A)}\n\n")
        f.write(f"Vector b:\n{' '.join(f'{elem:6.2f}' for elem in
b)}\n\n")
        f.write(f"Solution x:\n{' '.join(f'{elem:6.2f}' for elem in
x)}\n\n")
        f.write(f"Check A * x = b:\n{' '.join(f'{elem:6.2f}' for elem
in (matrix_vector_mult(A, x)))}\n")


if __name__ == "__main__":
    main()
```

Входные данные:

```
-11 9
-9 17 6
5 20 8
-6 -20 7
2 8
-117 -97 -6 59 -86
```

Результат работы программы:

```
Matrix A:
-11.00   9.00   0.00   0.00   0.00
 -9.00  17.00   6.00   0.00   0.00
  0.00   5.00  20.00   8.00   0.00
  0.00   0.00  -6.00 -20.00   7.00
  0.00   0.00   0.00   2.00   8.00

Vector b:
-117.00 -97.00  -6.00  59.00 -86.00

Solution x:
  9.00  -2.00   3.00  -7.00  -9.00

Check A * x = b:
-117.00 -97.00  -6.00  59.00 -86.00
```

Лабораторная работа № 1.3

Задание: реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

24. $\begin{cases} -25 \cdot x_1 + 4 \cdot x_2 - 4 \cdot x_3 + 9 \cdot x_4 = 86 \\ -9 \cdot x_1 + 21 \cdot x_2 + 5 \cdot x_3 - 6 \cdot x_4 = 29 \\ 9 \cdot x_1 + 2 \cdot x_2 + 19 \cdot x_3 - 7 \cdot x_4 = 28 \\ -7 \cdot x_1 + 4 \cdot x_2 - 7 \cdot x_3 + 25 \cdot x_4 = 68 \end{cases}$

Описание решения: в методе простых итераций сначала вычисляются матрица коэффициентов $alpha$ и вектор $beta$, затем на каждом шаге новое приближение находится как линейная комбинация предыдущего приближения с добавлением $beta$. Норма матрицы $alpha$ используется для оценки погрешности, и итерации продолжаются, пока норма разности приближений не станет меньше заданной точности eps. Метод Зейделя использует разложение матрицы $alpha$ на нижнюю треугольную (B) и верхнюю (C), затем инвертирует матрицу ($E - B$) и вычисляет новые приближения с учетом уже обновленных значений на текущей итерации. Для инвертирования матрицы применяется LU-разложение с частичным выбором ведущего элемента. Также выполняется проверка для каждого из методов путём перемножения исходной матрицы A и вычисленного решения x: в итоге получается вектор b.

Текст программы:

```
import math

def l2_norm(x):
    n = len(x)
    l2_norm = 0
    for i in range(n):
        l2_norm += x[i] * x[i]
    return math.sqrt(l2_norm)

# Метод простых итераций
def simple_iteration_method(A, b, eps, max_iter):
    n = len(A)
    alpha = [[-A[i][j] / A[i][i] if i != j else 0 for j in range(n)]
for i in range(n)]
    beta = [b[i] / A[i][i] for i in range(n)]

    # Начальное приближение
    x_new = [beta[i] for i in range(n)]
```

```
    iterations = 0

    eps_k = 0
    alpha_norm = l2_norm([alpha[i][j] for i in range(n) for j in
range(n)])

    while iterations == 0 or eps_k > eps:
        x = x_new[:]
        x_new = [sum(alpha[i][j] * x[j] for j in range(n)) + beta[i]
for i in range(n)]
        iterations += 1
        if (alpha_norm >= 1):
            eps_k = l2_norm([x_new[i] - x[i] for i in range(n)])
        else:
            eps_k = alpha_norm / (1 - alpha_norm) * l2_norm([x_new[i]
- x[i] for i in range(n)])
        if (iterations >= max_iter):
            return x_new, iterations

    return x_new, iterations


# Метод Зейделя (используя подсчет обратной матрицы с помощью LU
разложения)
def permutation_matrix(A):
    n = len(A)
    P = [[1 if i == j else 0 for j in range(n)] for i in range(n)]
    for i in range(n):
        max_row = max(range(i, n), key=lambda k: abs(A[k][i]))
        if max_row != i:
            P[i], P[max_row] = P[max_row], P[i]
    return P


def LU_decompose(PA):
    n = len(PA)
    L = [[0 for _ in range(n)] for _ in range(n)]
    U = [row[:] for row in PA]

    for i in range(n):
        L[i][i] = 1
        for j in range(i + 1, n):
            if U[i][i] != 0:
                L[j][i] = U[j][i] / U[i][i]
                for k in range(i, n):
                    U[j][k] -= L[j][i] * U[i][k]

    return L, U

def solve(L, U, b):
    n = len(L)
    y = [0 for _ in range(n)]
    for i in range(n):
        y[i] = b[i] - sum(L[i][j] * y[j] for j in range(i)) / L[i][i]
    x = [0 for _ in range(n)]
    for i in range(n - 1, -1, -1):
        x[i] = (y[i] - sum(U[i][j] * x[j] for j in range(i + 1, n))) /
U[i][i]
    return x

def transpose(A):
    m = len(A)
    n = len(A[0])
    A_T = [[A[j][i] for j in range(n)] for i in range(m)]
    return A_T
```

```
def inverse_matrix(A):
    n = len(A)
    E = [[1 if (i == j) else 0 for j in range(n)] for i in range(n)]

    P = permutation_matrix(A)
    PA = matrix_mult(P, A)
    L, U = LU_decompose(PA)

    A_inv = []
    for i in range(n):
        Pb = [sum(P[j][k] * E[k][i] for k in range(n)) for j in
range(n)]
        row_inv = solve(L, U, Pb)
        A_inv.append(row_inv)
    return transpose(A_inv)

def seidel_method(A, b, eps, max_iter):
    n = len(A)
    alpha = [[-A[i][j] / A[i][i] if i != j else 0 for j in range(n)]
for i in range(n)]
    beta = [b[i] / A[i][i] for i in range(n)]

    # Разделяем матрицу alpha на нижнюю треугольную (B) и оставшуюся
часть (C)
    B = [[alpha[i][j] if j < i else 0 for j in range(n)] for i in
range(n)]
    C = [[alpha[i][j] if j >= i else 0 for j in range(n)] for i in
range(n)]

    # Инвертируем (E - B)
    E_minus_B = [[1 if i == j else -B[i][j] for j in range(n)] for i
in range(n)]
    inv_E_minus_B = inverse_matrix(E_minus_B)

    # Вычисляем tmp1 и tmp2
    tmp1 = matrix_mult(inv_E_minus_B, C)
    tmp2 = matrix_vector_mult(inv_E_minus_B, beta)

    # Начальное приближение
    x_new = [tmp2[i] for i in range(n)]
    iterations = 0

    eps_k = 0
    alpha_norm = l2_norm([alpha[i][j] for i in range(n) for j in
range(n)])

    while iterations == 0 or eps_k > eps:
        x = x_new[:]
        for i in range(n):
            x_new = [sum(tmp1[i][j] * x[j] for j in range(n)) +
tmp2[i] for i in range(n)]
        iterations += 1
        if (alpha_norm >= 1):
            eps_k = l2_norm([x_new[i] - x[i] for i in range(n)])
        else:
            eps_k = l2_norm([C[i][j] for i in range(n) for j in
range(n)]) / (1 - alpha_norm) * l2_norm([x_new[i] - x[i] for i in
range(n)])
        if (iterations >= max_iter):
            return x_new, iterations
    return x_new, iterations


def invert_matrix(matrix):
    # Прямой и обратный ход метода Гаусса
    size = len(matrix)
```

```
    identity = [[1 if i == j else 0 for j in range(size)] for i in
range(size)]
    for i in range(size):
        factor = matrix[i][i]
        for j in range(size):
            matrix[i][j] /= factor
            identity[i][j] /= factor
        for k in range(size):
            if k != i:
                factor = matrix[k][i]
                for j in range(size):
                    matrix[k][j] -= factor * matrix[i][j]
                    identity[k][j] -= factor * identity[i][j]
    return identity

def matrix_vector_mult(A, x):
    n = len(A)
    m = len(x)
    return [sum(A[i][k] * x[k] for k in range(m)) for i in range(n)]

def matrix_mult(A, B):
    n = len(A)
    return [[sum(A[i][k] * B[k][j] for k in range(n)) for j in
range(n)] for i in range(n)]

def format_matrix(matrix):
    return '\n'.join(' '.join(f"{0.00 if abs(elem) < 1e-10 else
elem:6.2f}" for elem in row) for row in matrix)

def main():
    with open('input.txt', 'r') as f:
        data = [list(map(float, line.split())) for line in
f.readlines()]

    A = data[:-2]
    b = data[-2]
    eps = data[-1][0]

    simple_iter_x, simple_iter_i = simple_iteration_method(A, b, eps,
100)

    seidel_x, seidel_i = seidel_method(A, b, eps, 100)

    with open('output.txt', 'w') as f:
        f.write(f"Matrix A:\n{format_matrix(A)}\n\n")
        f.write(f"Vector b:\n{' '.join(f'{elem:6.2f}' for elem in
b)}\n\n")

        f.write(f"Simple iterations method:\n\nSolution x:\n{'
'.join(f'{elem:6.2f}' for elem in simple_iter_x)}\n\n")
        #f.write(f"Simple iterations method:\n\nSolution x:\n{'
'.join(f'{elem}' for elem in simple_iter_x)}\n\n")
        f.write(f"Number of iterations: {simple_iter_i}\n\n")
        f.write(f"Check A * x = b:\n{' '.join(f'{elem:6.2f}' for elem
in (matrix_vector_mult(A, simple_iter_x)))}\n\n")
        f.write(f"Seidel method:\n\nSolution x:\n{'
'.join(f'{elem:6.2f}' for elem in seidel_x)}\n\n")
        #f.write(f"Seidel method:\n\nSolution x:\n{' '.join(f'{elem}'
for elem in seidel_x)}\n\n")
        f.write(f"Number of iterations: {seidel_i}\n\n")
        f.write(f"Check A * x = b:\n{' '.join(f'{elem:6.2f}' for elem
in (matrix_vector_mult(A, seidel_x)))}\n")


if __name__ == "__main__":
    main()
```

Входные данные:

```
-25 4 -4 9
-9 21 5 -6
9 2 19 -7
-7 4 -7 25
86 29 28 68
0.000001
```

Результат работы программы:

```
Matrix A:
-25.00   4.00  -4.00   9.00
 -9.00  21.00   5.00  -6.00
  9.00   2.00  19.00  -7.00
 -7.00   4.00  -7.00  25.00

Vector b:
 86.00  29.00  28.00  68.00

Simple iterations method:

Solution x:
 -3.00   0.00   4.00   3.00

Number of iterations: 29

Check A * x = b:
 86.00  29.00  28.00  68.00

Seidel method:

Solution x:
 -3.00   0.00   4.00   3.00

Number of iterations: 11

Check A * x = b:
 86.00  29.00  28.00  68.00
```

Лабораторная работа № 1.4

Задание: реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

$$24.\begin{pmatrix} -8 & -4 & 8 \\ -4 & -3 & 9 \\ 8 & 9 & -5 \end{pmatrix}$$

Описание решения: сначала определяется максимальный по модулю внедиагональный элемент матрицы, который будет обнуляться на текущей итерации. Затем вычисляется угол вращения phi, который зависит от разности диагональных элементов и значения выбранного внедиагонального элемента. Строится матрица вращения U, которая используется для преобразования исходной матрицы A_i в новую матрицу путем умножения на U и транспонированную U^T. Собственные векторы накапливаются как произведение всех матриц вращения. Процесс повторяется, пока норма внедиагональных элементов не станет меньше заданной точности eps. В результате на диагонали матрицы A_i получаются собственные значения, а накопленные произведения матриц вращения дают собственные векторы. Проверка выполняется с помощью библиотеки $np.linalg.eig$.

Текст программы:

```
import math
# import numpy as np

def find_max_upper_element(A):
    n = len(A)
    l, m = 0, 1
    max_elem = abs(A[0][1])
    for i in range(n):
        for j in range(i + 1, n):
            if abs(A[i][j]) > max_elem:
                max_elem = abs(A[i][j])
                l = i
                m = j
    return l, m

def matrix_norm(A):
    n = len(A)
    norm = 0
    for i in range(n):
        for j in range(i + 1, n):
            norm += A[i][j] * A[i][j]
    return math.sqrt(norm)
```

```python
def matrix_mult(A, B):
    n = len(A)
    return [[sum(A[i][k] * B[k][j] for k in range(n)) for j in
range(n)] for i in range(n)]


def transpose(A):
    m = len(A)
    n = len(A[0])
    A_T = [[A[j][i] for j in range(n)] for i in range(m)]
    return A_T


def rotation_method(A, eps, max_iter):
    n = len(A)
    A_i = [row[:] for row in A]
    eigen_vectors = [[1 if i == j else 0 for j in range(n)] for i in
range(n)] # создаем единичную матрицу
    iters = 0

    while matrix_norm(A_i) > eps:
        l, m = find_max_upper_element(A_i)
        if A_i[l][l] - A_i[m][m] == 0:
            phi = math.pi / 4
        else:
            phi = 0.5 * math.atan(2 * A_i[l][m] / (A_i[l][l] -
A_i[m][m]))

        # Матрица вращения
        U = [[1 if i == j else 0 for j in range(n)] for i in range(n)]
        U[l][l] = math.cos(phi)
        U[l][m] = -math.sin(phi)
        U[m][l] = math.sin(phi)
        U[m][m] = math.cos(phi)

        U_T = transpose(U)
        A_i = matrix_mult(matrix_mult(U_T, A_i), U)
        eigen_vectors = matrix_mult(eigen_vectors, U) # СВ - столбцы
        eigen_values = [A_i[i][i] for i in range(n)] # СЗ -
диагональные элементы
        iters += 1
        if (iters >= max_iter):
            return eigen_values, eigen_vectors, iters

    return eigen_values, eigen_vectors, iters


def format_matrix(matrix):
    return '\n'.join(' '.join(f"{0.00 if abs(elem) < 1e-10 else
elem:6.2f}" for elem in row) for row in matrix)

def format_eigen_vectors(eigen_vectors):
    formatted_vectors = []
    for i, row in enumerate(eigen_vectors, start=1):
        formatted_row = ' '.join(f"{elem:6.2f}" for elem in row)
        formatted_vectors.append(f"eigen vector num {i}:
{formatted_row}")

    return '\n'.join(formatted_vectors)

def main():
    with open('input.txt', 'r') as f:
        data = [list(map(float, line.split())) for line in
f.readlines()]
```

```
    A = data[:-1]
    eps = data[-1][0]

    eigen_values, eigen_vectors, iters = rotation_method(A, eps, 100)
    eigen_vectors = transpose(eigen_vectors) # в столбцах наши СВ =>
транспонируем, чтобы теперь СВ были в строках

    # проверка через numpy
    # eigenvalues, eigenvectors = np.linalg.eig(A)
    # eigenvectors = transpose(eigen_vectors)

    with open('output.txt', 'w') as f:
        f.write(f"Matrix A:\n{format_matrix(A)}\n\n")
        f.write(f"Eigen values:\n{' '.join(f'{elem:6.2f}' for elem in
eigen_values)}\n\n")
        f.write(f"Eigen
vectors:\n{format_eigen_vectors(eigen_vectors)}\n\n")
        f.write(f"Number of iterations: {iters}\n\n")
        # f.write(f"Eigen values:\n{' '.join(f'{elem:6.2f}' for elem
in eigenvalues)}\n\n")
        # f.write(f"Eigen
vectors:\n{format_eigen_vectors(eigenvectors)}\n\n")

if __name__ == "__main__":
    main()
```

Входные данные:

```
-8 -4 8
-4 -3 9
8 9 -5
```

Результат работы программы:

```
Matrix A:
 -8.00  -4.00   8.00
 -4.00  -3.00   9.00
  8.00   9.00  -5.00

Eigen values:
 -2.03   5.63 -19.60

Eigen vectors:
eigen vector num 1:   0.76  -0.59   0.28
eigen vector num 2:   0.24   0.65   0.73
eigen vector num 3:  -0.60  -0.49   0.63

Number of iterations: 8
```

Лабораторная работа № 1.5

Задание: реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

$$24.\begin{pmatrix} -3 & 1 & -1 \\ 6 & 9 & -4 \\ 5 & -4 & -8 \end{pmatrix}$$

Описание решения: на каждой итерации выполняется QR-разложение матрицы (ортогональная Q и верхнетреугольная R) с использованием матрицы Хаусхолдера, затем матрица пересчитывается как произведение RQ. В ходе итераций проверяется норма поддиагональных элементов — если она становится меньше заданной точности eps , это означает, что соответствующий диагональный элемент можно считать найденным собственным значением. Для блоков 2×2 , которые не удаётся диагонализировать, собственные значения вычисляются через характеристическое уравнение квадратной подматрицы. Алгоритм продолжается, пока все собственные значения не будут извлечены с диагонали или из блоков 2×2 , последовательно уменьшая размер обрабатываемой подматрицы после нахождения каждого собственного значения. Критерием остановки служит малость всех поддиагональных элементов, что свидетельствует о достижении треугольной формы матрицы.

Текст программы:

```
import math

def sign(x):
    return -1 if x < 0 else (1 if x > 0 else 0)

def l2_norm(x):
    n = len(x)
    l2_norm = 0
    for i in range(n):
        l2_norm += x[i] * x[i]
    return math.sqrt(l2_norm)

def matrix_mult(A, B):
    n = len(A)
    return [[sum(A[i][k] * B[k][j] for k in range(n)) for j in
range(n)] for i in range(n)]
```

```
def transpose(A):
    m = len(A)
    n = len(A[0])
    A_T = [[A[j][i] for j in range(n)] for i in range(m)]
    return A_T

def matrix_subtract(A, B):
    n = len(A)
    return [[A[i][j] - B[i][j] for j in range(len(A[0]))] for i in
range(n)]

def householder(A, col):
    n = len(A)
    a = [A[i][col] for i in range(n)]
    v = [0.0] * n

    v[col] = a[col] + sign(a[col]) * l2_norm(a[col:])

    for i in range(col + 1, n):
        v[i] = a[i]

    E = [[1.0 if i == j else 0.0 for j in range(n)] for i in range(n)]
    scalar = 2.0 / sum(v[i] * v[i] for i in range(n))
    H = matrix_subtract(E, [[scalar * v[i] * v[j] for j in range(n)]
for i in range(n)])

    return H

def get_QR(A):
    n = len(A)
    Q = [[1.0 if i == j else 0.0 for j in range(n)] for i in range(n)]
    R = [row.copy() for row in A]

    for i in range(n-1):
        H = householder(R, i)
        Q = matrix_mult(Q, H)
        R = matrix_mult(H, R)

    return Q, R

def solve_quad(a, b, c):
    discr = b * b - 4 * a * c

    if discr < 0:
        real = -b / (2 * a)
        imag = math.sqrt(-discr)/(2 * a)
        return [complex(real, imag), complex(real, -imag)]
    else:
        root1 = (-b + math.sqrt(discr)) / (2 * a)
        root2 = (-b - math.sqrt(discr)) / (2 * a)
        return [root1, root2]

def get_roots(A, i):
    n = len(A)
    a11 = A[i][i]
    a12 = A[i][i + 1] if i + 1 < n else 0.0
    a21 = A[i + 1][i] if i + 1 < n else 0.0
    a22 = A[i + 1][i + 1] if i + 1 < n else 0.0

    return solve_quad(1.0, -a11 - a22, a11 * a22 - a12 * a21)

def subdiag_norm(A, i):
    return math.sqrt(sum(A[row][i] ** 2 for row in range(i + 1,
len(A))))

def eigenvals_QR(A, eps, max_iter):
```

```
    n = len(A)
    A_k = [row.copy() for row in A]
    eigen = []
    i = 0
    iterations = 0

    while i < n:
        while True:
            iterations += 1
            if iterations > max_iter:
                return eigen, iterations

            Q, R = get_QR(A_k)
            A_k = matrix_mult(R, Q)

            if subdiag_norm(A_k, i) <= eps:
                eigen.append(A_k[i][i])
                i += 1
                break
            elif i + 1 < n and subdiag_norm(A_k, i + 1) <= eps:
                roots = get_roots(A_k, i)
                eigen.extend(roots)
                i += 2
                break

    return eigen, iterations

def format_matrix(matrix):
    return '\n'.join(' '.join(f"{0.00 if abs(elem) < 1e-10 else
elem:6.2f}" for elem in row) for row in matrix)

def main():
    with open('input.txt', 'r') as f:
        data = [list(map(float, line.split())) for line in
f.readlines()]

    A = data[:-1]
    eps = data[-1][0]

    eigenvalues, iters = eigenvals_QR(A, eps, 100)

    with open('output.txt', 'w') as f:
        f.write(f"Matrix A:\n{format_matrix(A)}\n\n")
        f.write(f"Eigen values:\n{' '.join(f'{elem:6.2f}' for elem in
eigenvalues)}\n\n")
        f.write(f"Number of iterations: {iters}\n")

if __name__ == "__main__":
    main()
```

Входные данные:

```
-3 1 -1
6 9 -4
5 -4 -8
0.000001
```

Результат работы программы:

```
Matrix A:
 -3.00   1.00  -1.00
  6.00   9.00  -4.00
  5.00  -4.00  -8.00

Eigen values:
 10.31  -7.86  -4.45

Number of iterations: 27
```

Лабораторная работа № 2.1

Задание: реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

24. $x^6 - 5x - 2 = 0$.

Описание решения: программа считывает из входного файла границы интервала и заданную точность eps. Метод простых итераций преобразует уравнение к виду $x = \varphi(x)$ и последовательно подставляет значения, пока изменения не станут достаточно малы. Скорость сходимость линейная, важно, чтобы модуль производной функции $\varphi'(x)$ был меньше 1. Метод Ньютона использует касательные к графику функции для поиска корня. На каждом шаге приближение улучшается с помощью формулы $x_{n+1} = x_n - f(x_n)/f'(x_n)$. Данный метод сходится квадратично, если начальное приближение выбрано хорошо и производная функции не равна 0.

Текст программы:

```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x**6 - 5 * x - 2

def df(x):
    return 6 * x**5 - 5

def phi(x):
        return (5 * x + 2)**(1/6)

def dphi(x):
        return 5 / 6 / (5 * x + 2)**(5/6)

def ddphi(x):
     return - 125 / 36 / (5 * x + 2)**(11/6)

def simple_iteration_method(phi, a, b, epsilon, max_iter=1000):
    q = max([abs(dphi(x)) for x in np.arange(a, b, epsilon)])
    if q >= 1:
        print("Не удовлетворяется условие q < 1")
        exit()

    errors = []
    x_s = []
    x_prev = (a + b) / 2

    iterations = 0
    while True:
```

```
        x_next = phi(x_prev)
        error = q / (1 - q) * abs(x_next - x_prev)
        errors.append(error)
        x_s.append(x_prev)
        iterations += 1
        if error < epsilon or iterations >= max_iter:
            break
        x_prev = x_next
    return x_next, iterations, errors, x_s

def newton_method(f, df, a, b, epsilon, max_iter=1000):
    M2 = max([abs(ddphi(x)) for x in np.arange(a, b, epsilon)])
    m1 = min([abs(dphi(x)) for x in np.arange(a, b, epsilon)])

    errors = []
    x_s = []
    x_prev = (a + b) / 2

    iterations = 0
    while True:
        x_next = x_prev - f(x_prev) / df(x_prev)
        error = M2 / (2 * m1) * (x_next - x_prev)**2
        errors.append(error)
        x_s.append(x_prev)
        iterations += 1
        if error < epsilon or iterations >= max_iter:
            break
        x_prev = x_next
    return x_next, iterations, errors, x_s

def draw_graph_errors(iter_si, errors_si, iter_n, errors_n):
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, iter_si + 1), errors_si, label='Метод простых
итераций', marker='o')
    plt.plot(range(1, iter_n + 1), errors_n, label='Метод Ньютона',
marker='s')
    plt.xlabel('Количество итераций')
    plt.ylabel('Погрешность')
    plt.title('Зависимость погрешности от количества итераций')
    plt.yscale('log')
    plt.grid(True)
    plt.legend()
    plt.show()

def draw_graph_x(iter_si, x_vals_si, iter_n, x_vals_n):
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, iter_si + 1), x_vals_si, label='Метод простых
итераций', marker='o')
    plt.plot(range(1, iter_n + 1), x_vals_n, label='Метод Ньютона',
marker='s')
    plt.xlabel('Количество итераций')
    plt.ylabel('Значения')
    plt.title('График изменения значения x')
    plt.ylim(1, 3.5)
    plt.yticks(np.arange(0, 3.1, 0.5))
    plt.grid(True)
    plt.legend()
    plt.show()


def main():
    with open('input.txt', 'r') as file:
        line = file.readline()
        a, b = map(float, line.split())
        epsilon = float(file.readline())
```

```
    if (f(a) * f(b) >= 0):
        print("Выбраны неподходящие границы a и b")
        exit()

    # Метод простых итераций
    x_si, iter_si, errors_si, x_vals_si = simple_iteration_method(phi,
a, b, epsilon)

    # Метод Ньютона
    x_n, iter_n, errors_n, x_vals_n = newton_method(f, df, a, b,
epsilon)

    with open('output.txt', 'w') as file:
        file.write(f"Nonlinear equation:\nf(x) = x^6 - 5x - 2\n")
        file.write(f"Interval: [{a}, {b}]\n")
        file.write(f"Epsilon: {epsilon}\n")
        file.write(f"\nSimple iterations method:\n")
        file.write(f"Root: {x_si}\nNumber of iterations:
{iter_si}\nf(x) = {f(x_si)}\nError: {errors_si[-1]}\n")
        file.write(f"\nNewton method:\n")
        file.write(f"Root: {x_n}\nNumber of iterations: {iter_n}\nf(x)
= {f(x_n)}\nError: {errors_n[-1]}")

    draw_graph_x(iter_si, x_vals_si, iter_n, x_vals_n)
    draw_graph_errors(iter_si, errors_si, iter_n, errors_n)

if __name__ == "__main__":
    main()
```

Входные данные:

```
1.0 5.0
1e-6
```

Результат работы программы:

```
Nonlinear equation:
f(x) = x^6 - 5x - 2
Interval: [1.0, 5.0]
Epsilon: 1e-06

Simple iterations method:
Root: 1.448678795609901
Number of iterations: 7
f(x) = 2.4604634581315565e-05
Error: 9.6995264333122e-07

Newton method:
Root: 1.4486780564352173
Number of iterations: 8
f(x) = 2.3415100969259584e-09
Error: 3.248840796951551e-11
```

Графики:

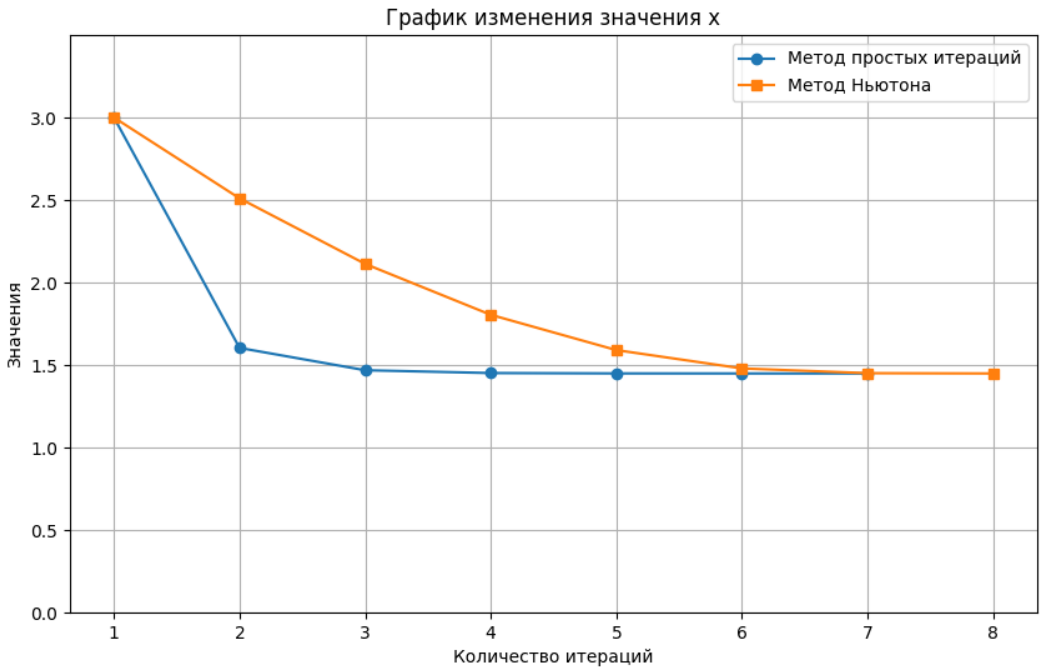
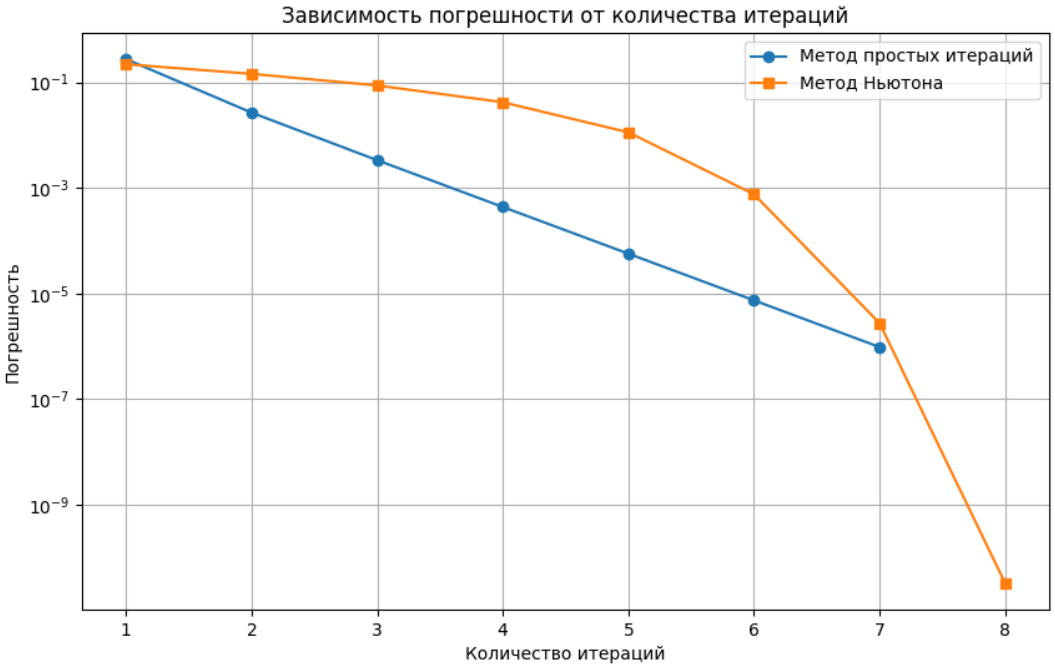


График изменения значения x
Метод простых итераций
Метод Ньютона
Значения
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7
8
Количество итераций


Зависимость погрешности от количества итераций
Метод простых итераций
Метод Ньютона
Погрешность
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
1
2
3
4
5
6
7
8
Количество итераций

Лабораторная работа № 2.2

Задание: реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

22	1
23	2
24	3

$$\begin{cases} a x_1^2 - x_1 + x_2^2 - 1 = 0, \\ x_2 - \operatorname{tg} x_1 = 0. \end{cases}$$

Описание решения: решаем систему из двух нелинейных уравнений двумя методами: простых итераций и Ньютона. Метод простых итераций постепенно улучшает приближение, заменяя переменные по определённым формулам, пока значения не перестанут сильно меняться. Важно, чтобы изменения на каждом шаге не росли — это проверяется через матрицу производных. Метод Ньютона строит касательные в каждой точке и делает более точный шаг к решению. Он сходится быстрее, особенно когда мы уже близки к правильному ответу, но требует вычисления производных и решения линейной системы на каждом шаге. Оба метода начинают движение от средних значений заданных интервалов и останавливаются, когда достигнута нужная точность или превышено число допустимых итераций.

Текст программы:

```
import math
import numpy as np
import matplotlib.pyplot as plt

def f1(x1, x2):
    return 3 * x1**2 - x1 + x2**2 - 1

def f2(x1, x2):
    return x2 - math.tan(x1)

def df1_dx1(x1, x2): return 6 * x1 - 1
def df1_dx2(x1, x2): return 2 * x2
def df2_dx1(x1, x2): return -1 / (math.cos(x1))**2
def df2_dx2(x1, x2): return 1

def phi1(x1, x2):
    return x1 - f1(x1, x2) / 10

def phi2(x1, x2):
    return math.tan(x1)
```

```
def dphi1_dx1(x1, x2): return 1 - (6 * x1 - 1) / 10
def dphi1_dx2(x1, x2): return - (2 * x2) / 10
def dphi2_dx1(x1, x2): return 1 / (math.cos(x1))**2
def dphi2_dx2(x1, x2): return 0

def simple_iteration_method(intervals, eps, max_iter=100):
    a1, b1 = intervals[0][0], intervals[0][1]
    a2, b2 = intervals[1][0], intervals[1][1]
    x1 = (a1 + b1) / 2
    x2 = (a2 + b2) / 2
    iters = 0
    errors = []
    history = []

    J = np.array([[dphi1_dx1(x1, x2), dphi1_dx2(x1, x2)],
                  [dphi2_dx1(x1, x2), dphi2_dx2(x1, x2)]])

    q = np.linalg.norm(J, ord=np.inf)
    if (q < 1):
        while True:
            x1_new, x2_new = phi1(x1, x2), phi2(x1, x2)
            error = q / (1 - q) * max(abs(x1_new - x1), abs(x2_new -
x2))
            errors.append(error)
            history.append((x1, x2))
            iters += 1
            if error < eps or iters >= max_iter:
                break
            x1, x2 = x1_new, x2_new
    else:
        while True:
            x1_new, x2_new = phi1(x1, x2), phi2(x1, x2)
            error = max(abs(x1_new - x1), abs(x2_new - x2))
            errors.append(error)
            history.append((x1, x2))
            iters += 1
            if error < eps or iters >= max_iter:
                break
            x1, x2 = x1_new, x2_new
    return [x1, x2], iters, errors, history


def newton_method(intervals, eps, max_iter=100):
    a1, b1 = intervals[0][0], intervals[0][1]
    a2, b2 = intervals[1][0], intervals[1][1]
    x1 = (a1 + b1) / 2
    x2 = (a2 + b2) / 2
    iters = 0
    errors = []
    history = []

    while True:
        J = np.array([[df1_dx1(x1, x2), df1_dx2(x1, x2)],
                      [df2_dx1(x1, x2), df2_dx2(x1, x2)]])
        F = np.array([f1(x1, x2), f2(x1, x2)])

        delta = np.linalg.solve(J, -F)
        x1_new, x2_new = x1 + delta[0], x2 + delta[1]
        error = max(abs(x1_new - x1), abs(x2_new - x2))
        iters += 1
        errors.append(error)
        history.append((x1, x2))
        if error < eps or iters >= max_iter:
            break
        x1, x2 = x1_new, x2_new
    return [float(x1), float(x2)], iters, errors, history
```

```python
def draw_graph_errors(iter_si, errors_si, iter_n, errors_n):
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, iter_si + 1), errors_si, label='Метод простых
итераций', marker='o')
    plt.plot(range(1, iter_n + 1), errors_n, label='Метод Ньютона',
marker='s')
    plt.xlabel('Количество итераций')
    plt.ylabel('Погрешность')
    plt.title('Зависимость погрешности от количества итераций')
    plt.yscale('log')
    plt.grid(True)
    plt.legend()
    plt.show()

def draw_graph_system(iter_si, history_si, iter_n, history_n):
    x1_si = [point[0] for point in history_si]
    x2_si = [point[1] for point in history_si]

    x1_n = [point[0] for point in history_n]
    x2_n = [point[1] for point in history_n]

    plt.figure(figsize=(10, 6))

    plt.subplot(1, 2, 1)
    plt.plot(range(1, iter_si + 1), x1_si, label='Метод простых
итераций', marker='o')
    plt.plot(range(1, iter_n + 1), x1_n, label='Метод Ньютона',
marker='s')
    plt.xlabel('Количество итераций')
    plt.ylabel('Значения')
    plt.title('График изменения значения x1')
    plt.ylim(1, 2)
    plt.yticks(np.arange(0, 2.1, 0.25))
    plt.grid(True)
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(range(1, iter_si + 1), x2_si, label='Метод простых
итераций', marker='o')
    plt.plot(range(1, iter_n + 1), x2_n, label='Метод Ньютона',
marker='s')
    plt.xlabel('Количество итераций')
    plt.ylabel('Значения')
    plt.title('График изменения значения x2')
    plt.ylim(1, 2)
    plt.yticks(np.arange(0, 2.1, 0.25))
    plt.grid(True)
    plt.legend()

    plt.tight_layout()
    plt.show()

def main():
    with open('input.txt', 'r') as file:
        line = file.readline()
        a1, b1 = map(float, line.split())
        line = file.readline()
        a2, b2 = map(float, line.split())
        epsilon = float(file.readline())

    intervals = [(a1, b1), (a2, b2)]

    # Метод простых итераций
    x_si, iter_si, errors_si, history_si =
simple_iteration_method(intervals, epsilon)
```

```
    # Метод Ньютона
    x_n, iter_n, errors_n, history_n = newton_method(intervals,
epsilon)

    with open('output.txt', 'w') as file:
        file.write(f"System of nonlinear equations:\n | 3*x1^2 - x1 +
x2^2 - 1 = 0\n | x2 - tg(x1) = 0\n")
        file.write(f"Intervals: [{a1}, {b1}], [{a2}, {b2}]\n")
        file.write(f"Epsilon: {epsilon}\n")
        file.write(f"\nSimple iterations method:\n")
        file.write(f"Root: {x_si}\nNumber of iterations: {iter_si}\n")
        file.write(f"f1(x1, x2) = {f1(*x_si)}\nf2(x1, x2) =
{f2(*x_si)}\nError: {errors_si[-1]}\n")
        file.write(f"\nNewton method:\n")
        file.write(f"Root: {x_n}\nNumber of iterations: {iter_n}\n")
        file.write(f"f1(x1, x2) = {f1(*x_n)}\nf2(x1, x2) =
{f2(*x_n)}\nError: {errors_n[-1]}\n")

    draw_graph_errors(iter_si, errors_si, iter_n, errors_n)
    draw_graph_system(iter_si, history_si, iter_n, history_n)


if __name__ == "__main__":
    main()
```

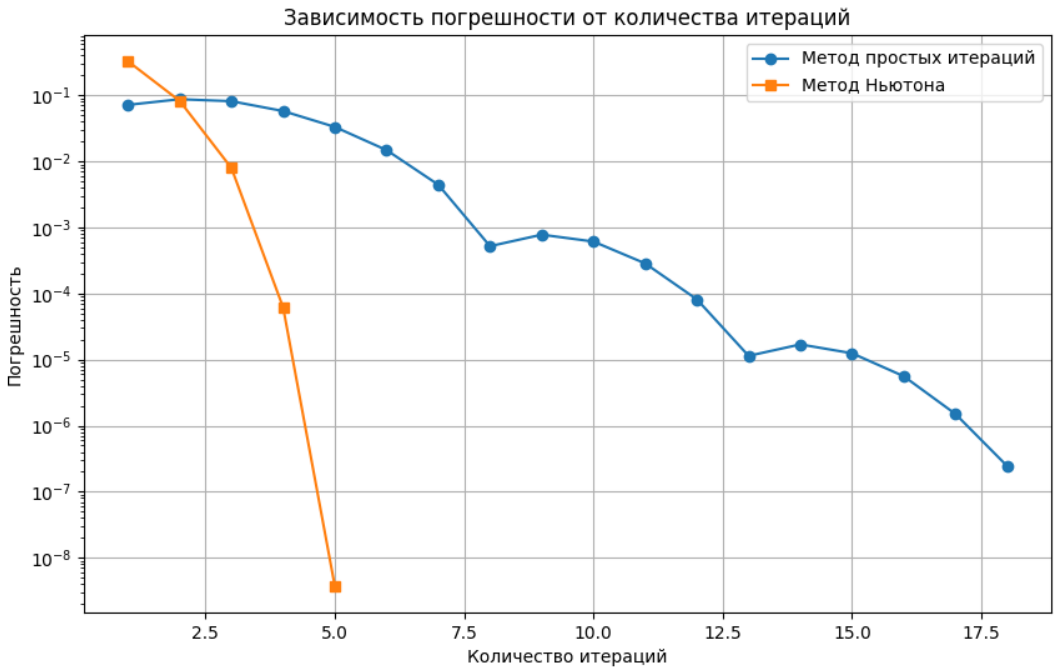
Входные данные:

```
0.2 0.6
0.4 0.5
1e-6
```

Результат работы программы:

```
System of nonlinear equations:
 | 3*x1^2 - x1 + x2^2 - 1 = 0
 | x2 - tg(x1) = 0
Intervals: [0.2, 0.6], [0.4, 0.5]
Epsilon: 1e-06

Simple iterations method:
Root: [0.611098309086366, 0.7005550070076721]
Number of iterations: 18
f1(x1, x2) = 2.4388618007353813e-06
f2(x1, x2) = 6.874343805307603e-08
Error: 2.438861800291292e-07

Newton method:
Root: [0.6110978201770642, 0.7005542054291679]
Number of iterations: 5
f1(x1, x2) = 1.204290689393872e-08
f2(x1, x2) = -3.98046529070939e-09
Error: 3.7053128343345065e-09
```

Графики:


Зависимость погрешности от количества итераций
Метод простых итераций
Метод Ньютона
Погрешность
Количество итераций

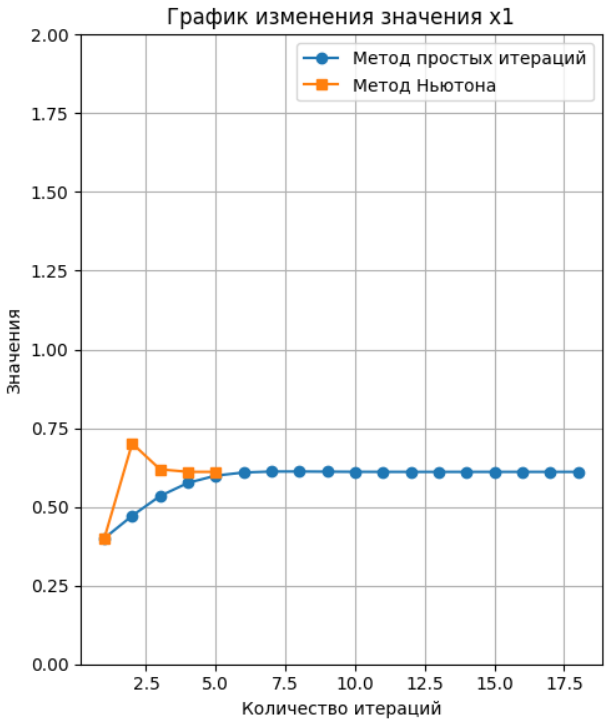


График изменения значения x1
Метод простых итераций
Метод Ньютона
Значения
Количество итераций

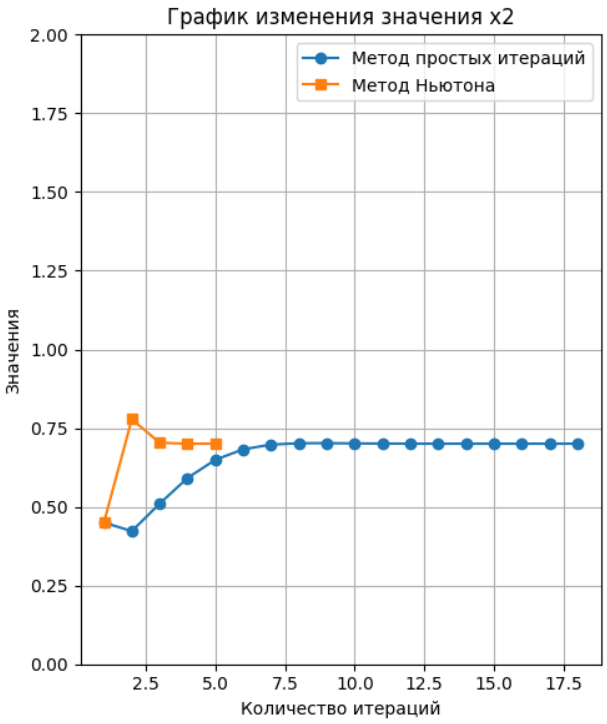


График изменения значения x2
Метод простых итераций
Метод Ньютона
Значения
Количество итераций

Лабораторная работа № 3.1

Задание: используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках X_i, $i = 0, \ldots, 3$ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки $\{X_i, Y_i\}$. Вычислить значение погрешности интерполяции в точке X^*.

24. $y = \frac{1}{x} + x$, а) $X_i = 0.1, 0.5, 0.9, 1.3$; б) $X_i = 0.1, 0.5, 1.1, 1.3$; $X^* = 0.8$.

Описание решения: строим интерполяционные многочлены по заданным точкам с помощью методов Лагранжа и Ньютона. Метод Лагранжа строит многочлен как сумму дробей, каждая из которых обращается в ноль во всех точках, кроме одной, где принимает нужное значение. Результат — единая формула, которая проходит через все заданные точки. Метод Ньютона использует разделенные разности для построения того же многочлена, но делает это итерационно, добавляя к результату по одному слагаемому за шаг. Оба метода аппроксимируют функцию $f(x) = \frac{1}{x} + x$ на заданных точках, строя многочлен, который точно проходит через эти точки.

Текст программы:

```
import numpy as np

def f(x):
    return 1/x + x

def prod(points, x, i):
    res = 1
    for j in range(len(points)):
        if j != i:
            res *= (x - points[j])
    return res

def prod_to_print(points, i):
    prod = ""
    for j in range(len(points)):
        if i != j:
            prod += f"(x - {points[j]})"
    return prod

def Lagrange_interpolation(points, x):
    res = 0
    res_str = "L(x) = "
    for i in range(len(points)):
        f_prod = f(points[i]) / prod(points, points[i], i)
        res += f_prod * prod(points, x, i)

        sign = " + " if f_prod > 0 else ""
        res_str += f"{sign} {f_prod:.2f}*" + prod_to_print(points, i)

    return res, res_str
```

```
def Newton_interpolation(points, x):
    y = [f(p) for p in points]

    coefs = [y[i] for i in range(len(points))]
    for j in range(1, len(points)):
        for i in range(len(points) - 1, j - 1, -1):
            coefs[i] = float(coefs[i] - coefs[i - 1]) / float(points[i]
- points[i - j])


    res = coefs[0]
    res_str = f"P(x) = {coefs[0]:.2f}"
    current_terms = []

    for i in range(1, len(coefs)):
        current_terms.append(f"(x - {points[i-1]:.2f})")
        term_str = "*".join(current_terms)
        res += coefs[i] * np.prod([x - points[j] for j in range(i)])

        sign = " + " if coefs[i] >= 0 else " - "
        res_str += f"{sign}{abs(coefs[i]):.2f}*{term_str}"

    return res, res_str



def main():
    with open('input.txt', 'r') as file:
        data = [list(map(float, line.split())) for line in
file.readlines()]
    points = data[:-1]
    x = data[-1][0]


    val_lagrange1, str_lagrange1  = Lagrange_interpolation(points[0],
x)
    abs_err_lag1 = abs(f(x) - val_lagrange1)
    val_newton1, str_newton1 = Newton_interpolation(points[0], x)
    abs_err_new1 = abs(f(x) - val_newton1)


    val_lagrange2, str_lagrange2  = Lagrange_interpolation(points[1],
x)
    abs_err_lag2 = abs(f(x) - val_lagrange2)
    val_newton2, str_newton2 = Newton_interpolation(points[1], x)
    abs_err_new2 = abs(f(x) - val_newton2)

    with open('output.txt', 'w') as file:
        file.write(f"Function: y = 1/x + x\n")
        file.write(f"x* = {x}, y = {f(x)}\n\n\n")

        file.write(f"Points: {points[0]}\n\n")
        file.write("-Lagrange interpolation:\n")
        file.write(str_lagrange1)
        file.write(f"\nValue: {val_lagrange1}\n")
        file.write(f"Error: {abs_err_lag1}\n")
        file.write("-Newton interpolation:\n")
        file.write(str_newton1)
        file.write(f"\nValue: {val_newton1}\n")
        file.write(f"Error: {abs_err_new1}\n\n\n")

        file.write(f"Points: {points[1]}\n\n")
        file.write("-Lagrange interpolation:\n")
        file.write(str_lagrange2)
        file.write(f"\nValue: {val_lagrange2}\n")
        file.write(f"Error: {abs_err_lag2}\n")
```

```
            file.write("-Newton interpolation:\n")
            file.write(str_newton2)
            file.write(f"\nValue: {val_newton2}\n")
            file.write(f"Error: {abs_err_new2}\n")

if __name__ == "__main__":
    main()
```

Входные данные:

```
0.1 0.5 0.9 1.3
0.1 0.5 1.1 1.3
0.8
```

Результат работы программы:

```
Function: y = 1/x + x
x* = 0.8, y = 2.05


Points: [0.1, 0.5, 0.9, 1.3]

-Lagrange interpolation:
L(x) =  -26.30*(x - 0.5)(x - 0.9)(x - 1.3) +  19.53*(x - 0.1)(x -
0.9)(x - 1.3) -15.71*(x - 0.1)(x - 0.5)(x - 1.3) +  5.39*(x - 0.1)(x -
0.5)(x - 0.9)
Value: 1.8256410256410254
Error: 0.22435897435897445
-Newton interpolation:
P(x) = 10.10 - 19.00*(x - 0.10) + 22.22*(x - 0.10)*(x - 0.50) -
17.09*(x - 0.10)*(x - 0.50)*(x - 0.90)
Value: 1.8256410256410267
Error: 0.22435897435897312


Points: [0.1, 0.5, 1.1, 1.3]

-Lagrange interpolation:
L(x) =  -21.04*(x - 0.5)(x - 1.1)(x - 1.3) +  13.02*(x - 0.1)(x -
1.1)(x - 1.3) -16.74*(x - 0.1)(x - 0.5)(x - 1.3) +  10.78*(x - 0.1)(x
- 0.5)(x - 1.1)
Value: 1.4993006993006994
Error: 0.5506993006993004
-Newton interpolation:
P(x) = 10.10 - 19.00*(x - 0.10) + 18.18*(x - 0.10)*(x - 0.50) -
13.99*(x - 0.10)*(x - 0.50)*(x - 1.10)
Value: 1.4993006993007012
Error: 0.5506993006992986
```

Лабораторная работа № 3.2

Задание: построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x = x_0$ и $x = x_4$. Вычислить значение функции в точке $x = X^*$.

24. $X^* = 0.8$

i	0	1	2	3	4
x_i	0.1	0.5	0.9	1.3	1.7
f_i	100.00	4.0	1.2346	0.59172	0.34602

Описание решения: строим сплайн-интерполяцию, то есть приближаем функцию, заданную набором точек, кусочно-полиномиальной функцией — кубическим сплайном. Сначала вычисляются коэффициенты для каждого участка сплайна с помощью решения трёхдиагональной системы уравнений, затем по этим коэффициентам находим значение функции в любой точке между узлами.

Текст программы:

```
import numpy as np
import matplotlib.pyplot as plt

from typing import List

def tri_diagonal_matrix_algorithm(matrix: list, d: list, shape: int) ->
list:
    a, b, c = zip(*matrix)
    p = [-c[0] / b[0]]
    q = [d[0] / b[0]]
    x = [0] * (shape + 1)
    for i in range(1, shape):
        p.append(-c[i] / (b[i] + a[i] * p[i - 1]))
        q.append((d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]))

    for i in reversed(range(shape)):
        x[i] = p[i] * x[i + 1] + q[i]

    return x[:-1]


def spline(x: list, f: list) -> List[list]:
    n = len(x)
    h = [x[i] - x[i - 1] for i in range(1, n)]
    tridiag_matrix = [[0, 2 * (h[0] + h[1]), h[1]]]
    b = [3 * ((f[2] - f[1]) / h[1] - (f[1] - f[0]) / h[0])]
    for i in range(1, n - 3):
        tridiag_row = [h[i], 2 * (h[i] + h[i + 1]), h[i + 1]]
        tridiag_matrix.append(tridiag_row)
        b.append(3 * ((f[i + 2] - f[i + 1]) / h[i + 1] - (f[i + 1] -
f[i]) / h[i]))

    tridiag_matrix.append([h[-2], 2 * (h[-2] + h[-1]), 0])
    b.append(3 * ((f[-1] - f[-2]) / h[-1] - (f[-2] - f[-3]) / h[-2]))
```

```
    c = tri_diagonal_matrix_algorithm(tridiag_matrix, b, n - 2)
    a = []
    b = []
    d = []
    c.insert(0, 0)
    for i in range(1, n):
        a.append(f[i - 1])
        if i < n - 1:
            d.append((c[i] - c[i - 1]) / (3 * h[i - 1]))
            b.append((f[i] - f[i - 1]) / h[i - 1] - h[i - 1] * (c[i] +
2 * c[i - 1]) / 3)

    b.append((f[-1] - f[-2]) / h[-1] - 2 * h[-1] * c[-1] / 3)
    d.append(-c[-1] / (3 * h[-1]))
    return a, b, c, d


def interpolate(x: list, x_0: float, coef: list) -> float:
    k = 0
    for i, j in zip(x, x[1:]):
        if i <= x_0 <= j:
            break
        k += 1

    a, b, c, d = coef
    diff = x_0 - x[k]
    return a[k] + b[k] * diff + c[k] * diff ** 2 + d[k] * diff ** 3

def draw_plot(x_test, res, x, f, coef):
    x_vals = np.linspace(x[0], x[-1])
    y = [interpolate(x, val, coef) for val in x_vals]
    plt.figure(figsize=(12, 8))
    plt.plot(x_vals, y, color='b')
    plt.scatter(x, f, color='r')
    plt.scatter(x_test, res, color='g')
    plt.grid()
    plt.show()

def main():
    with open('input.txt', 'r') as file:
        data = [list(map(float, line.split())) for line in
file.readlines()]
    points = data[:-1]
    x_test = data[-1][0]

    x = points[0]
    y = points[1]

    coef = spline(x, y)
    a, b, c, d = coef
    res = interpolate(x, x_test, coef)

    with open('output.txt', 'w') as file:
        for i in range(5):
            file.write(f"x = {x[i]}, y = {y[i]}\n")
        file.write(f"_\n")
        for i in range(len(x) - 1):
            file.write(f'[{x[i]}; {x[i + 1]}):\n')
            file.write(f's(x) = {a[i]} + {b[i]}(x - {x[i]}) + {c[i]}(x
- {x[i]})^2 + {d[i]}(x - {x[i]})^3\n')
        file.write(f'_\ns(x*) = s({x_test}) = {res}\n')

    draw_plot(x_test, res, x, y, coef)

if __name__ == "__main__":
    main()
```

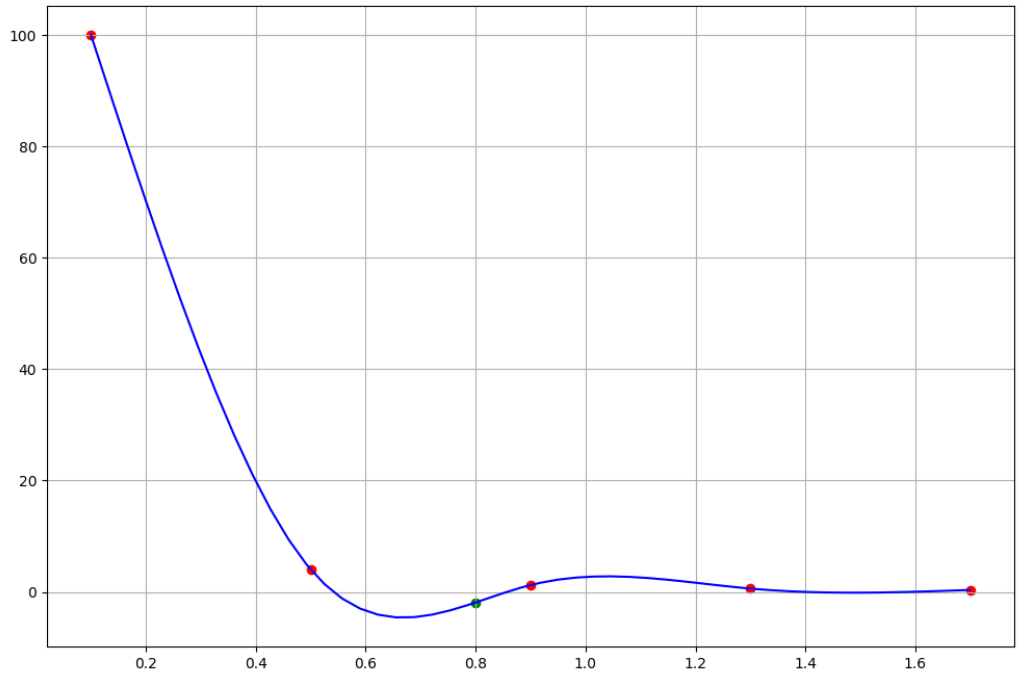
Входные данные:

```
0.1 0.5 0.9 1.3 1.7
100.00 4.0 1.2346 0.59172 0.34602
0.8
```

Результат работы программы:

```
x = 0.1, y = 100.0
x = 0.5, y = 4.0
x = 0.9, y = 1.2346
x = 1.3, y = 0.59172
x = 1.7, y = 0.34602

[0.1; 0.5):
s(x) = 100.0 + -302.07259375(x - 0.1) + 0(x - 0.1)^2 +
387.9537109374999(x - 0.1)^3
[0.5; 0.9):
s(x) = 4.0 + -115.8548125(x - 0.5) + 465.54445312499996(x - 0.5)^2 + -
482.97792968749997(x - 0.5)^3
[0.9; 1.3):
s(x) = 1.2346 + 24.751343750000007(x - 0.9) + -114.02906250000001(x -
0.9)^2 + 120.3317578125(x - 0.9)^3
[1.3; 1.7):
s(x) = 0.59172 + -8.7126625(x - 1.3) + 30.369046875000006(x - 1.3)^2 +
-25.30753906250001(x - 1.3)^3

s(x*) = s(0.8) = -1.8978470703124994
```

График:



Лабораторная работа № 3.3

Задание: для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

24.

i	0	1	2	3	4	5
x_i	-1.0	0.0	1.0	2.0	3.0	4.0
y_i	0.86603	1.0	0.86603	0.50	0.0	-0.50

Описание решения: решаем систему линейных уравнений с помощью LU-разложения, а потом строим многочлен методом наименьших квадратов, чтобы наилучшим образом приблизить заданные точки. Сначала матрица системы разбивается на нижнюю (L) и верхнюю (U) треугольные матрицы, затем система решается по шагам: сначала с L, потом с U (из первой лабораторной работы). Для аппроксимации данных строится система уравнений из сумм степеней x, решается она через LU-разложение, и получаются коэффициенты многочлена, который минимизирует сумму квадратов ошибок между предсказанными и заданными значениями y.

Текст программы:

```
import matplotlib.pyplot as plt

def LU_decompose(A):
    n = len(A)
    # lower
    L = [[0 for _ in range(n)] for _ in range(n)]
    # upper
    U = [row[:] for row in A]

    for k in range(1, n):
        for i in range(k - 1, n):
            for j in range(i, n):
                L[j][i] = U[j][i] / U[i][i]

        for i in range(k, n):
            for j in range(k - 1, n):
                U[i][j] = U[i][j] - L[i][k - 1] * U[k - 1][j]

    return L, U

def solve_system(L, U, b):
    # L * y = b
    n = len(L)
    y = [0 for _ in range(n)]
    for i in range(n):
        s = 0
        for j in range(i):
            s += L[i][j] * y[j]
```

```
            y[i] = (b[i] - s) / L[i][i]

        # U * x = y
        x = [0 for _ in range(n)]
        for i in range(n - 1, -1, -1):
            s = 0
            for j in range(n - 1, i - 1, -1):
                s += U[i][j] * x[j]
            x[i] = (y[i] - s) / U[i][i]
        return x

    def least_squares(x, y, n):
        assert len(x) == len(y)
        A = []
        b = []
        for k in range(n + 1):
            A.append([sum(map(lambda x: x ** (i + k), x)) for i in range(n
    + 1)])
            b.append(sum(map(lambda x: x[0] * x[1] ** k, zip(y, x))))
        L, U = LU_decompose(A)
        return solve_system(L, U, b)

    def P(coefs, x):
        return sum([c * x**i for i, c in enumerate(coefs)])

    def sum_squared_errors(x, y, ls_coefs):
        y_ls = [P(ls_coefs, x_i) for x_i in x]
        return sum((y_i - y_ls_i)**2 for y_i, y_ls_i in zip(y, y_ls))

    def main():
        with open('input.txt', 'r') as file:
            data = [list(map(float, line.split())) for line in
    file.readlines()]
        x = data[0]
        y = data[1]

        plt.scatter(x, y, color='r')
        ls1 = least_squares(x, y, 1)
        plt.plot(x, [P(ls1, x_i) for x_i in x], color='b', label='degree =
    1')
        ls2 = least_squares(x, y, 2)
        plt.plot(x, [P(ls2, x_i) for x_i in x], color='g', label='degree =
    2')
        plt.legend()
        plt.show()

        with open('output.txt', 'w') as file:
            file.write("Least squares method, degree = 1:\n")
            file.write(f"P(x) = {ls1[0]} + {ls1[1]} * x\n")
            file.write(f"Sum of squared errors = {sum_squared_errors(x, y,
    ls1)}\n\n")
            file.write("Least squares method, degree = 2:\n")
            file.write(f"P(x) = {ls2[0]} + {ls2[1]} * x + {ls2[2]} *
    x^2\n")
            file.write(f"Sum of squared errors = {sum_squared_errors(x, y,
    ls2)}")

    if __name__ == "__main__":
        main()
```

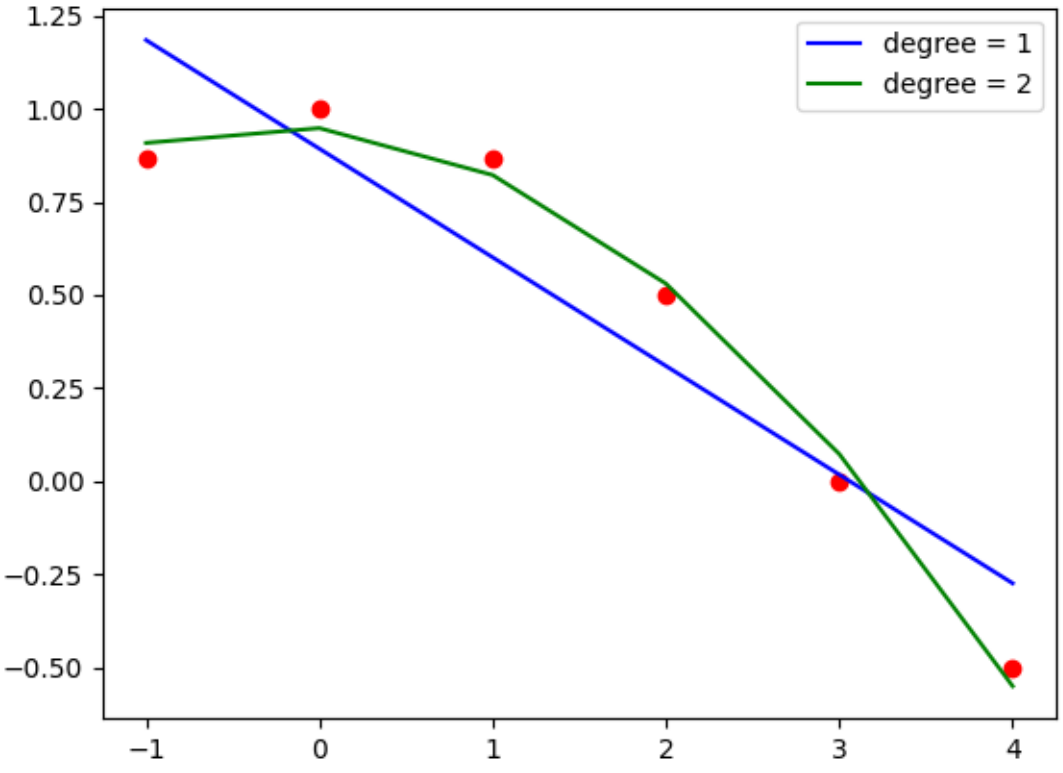
Входные данные:

```
-1.0 0.0 1.0 2.0 3.0 4.0
0.86603 1.0 0.86603 0.5 0.0 -0.5
```

Результат работы программы:

```
Least squares method, degree = 1:
P(x) = 0.8923224761904761 + -0.2913194285714285 * x
Sum of squared errors = 0.2708179489276191

Least squares method, degree = 2:
P(x) = 0.9474887857142856 + -0.04307103571428535 * x + -
0.08274946428571439 * x^2
Sum of squared errors = 0.01517892558357144
```

График:



Лабораторная работа № 3.4

Задание: вычислить первую и вторую производную от таблично заданной функции $y_i = f(x_i),\ i = 0,1,2,3,4$ в точке $x = X^*$.

24. $X^* = 1.4$

i	0	1	2	3	4
x_i	1.0	1.2	1.4	1.6	1.8
y_i	2.0	2.1344	2.4702	2.9506	3.5486

Описание решения: приближённо вычисляем первую и вторую производные функции, заданной таблично, с помощью кусочно-квадратичной интерполяции. Для точки, где нужно найти производную, выбирается подходящий интервал между узлами, затем строится квадратичный сплайн — по нему вычисляются первая и вторая производные, учитывающие наклон и кривизну функции на этом участке. Проверка выполняется с помощью библиотеки *scipy*.

Текст программы:

```
from scipy.interpolate import CubicSpline
import numpy as np

def df(x_test, x, y):
    assert len(x) == len(y)
    for interval in range(len(x)):
        if x[interval] <= x_test < x[interval + 1]:
            i = interval
            break

    a1 = (y[i + 1] - y[i]) / (x[i + 1] - x[i])
    # наклон линейной интерполяции
    a2 = ((y[i + 2] - y[i + 1]) / (x[i + 2] - x[i + 1]) - a1) / (x[i + 
2] - x[i]) * (2 * x_test - x[i] - x[i + 1])
    # учитываем кривизну сплайна
    return a1 + a2

def d2f(x_test, x, y):
    assert len(x) == len(y)
    for interval in range(len(x)):
        if x[interval] <= x_test < x[interval + 1]:
            i = interval
            break

    num = (y[i + 2] - y[i + 1]) / (x[i + 2] - x[i + 1]) - (y[i + 1] - 
y[i]) / (x[i + 1] - x[i])
    return 2 * num / (x[i + 2] - x[i])

def check_with_scipy(x_test, x, y):
    cs = CubicSpline(x, y)
    df_scipy = cs(x_test, 1)
    d2f_scipy = cs(x_test, 2)
    return df_scipy, d2f_scipy

def main():
    with open('input.txt', 'r') as file:
```

```
        data = [list(map(float, line.split())) for line in
file.readlines()]
    points = data[:-1]
    x_test = data[-1][0]
    x = points[0]
    y = points[1]

    x_np, y_np = np.array(x), np.array(y)
    df_scipy, d2f_scipy = check_with_scipy(x_test, x_np, y_np)

    with open('output.txt', 'w') as file:
        file.write(f"First derivative:\ndf({x_test}) = {df(x_test, x,
y)}\n")
        file.write(f"Check:\ndf_scipy({x_test}) = {df_scipy}\n\n")
        file.write(f"Second derivative:\nd2f({x_test}) = {d2f(x_test,
x, y)}\n")
        file.write(f"Check:\nd2f_scipy({x_test}) = {d2f_scipy}")

if __name__ == '__main__':
    main()
```

Входные данные:

```
1.0 1.2 1.4 1.6 1.8
2.0 2.1344 2.4702 2.9506 3.5486
1.4
```

Результат работы программы:

```
First derivative:
df(1.4) = 2.107999999999996
Check:
df_scipy(1.4) = 2.075416666666666

Second derivative:
d2f(1.4) = 2.9400000000000106
Check:
d2f_scipy(1.4) = 3.4287499999999684
```

Лабораторная работа № 3.5

Задание: вычислить определенный интеграл $F = \int_{X_0}^{X_1} ydx$, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя метод Рунге-Ромберга.

24. $y = \sqrt{16 - x^2}$, $X_0 = -2$, $X_k = 2$, $h_1 = 1.0$, $h_2 = 0.5$;

Описание решения: метод прямоугольников приближает интеграл, разбивая область под графиком функции на прямоугольники. Высота каждого прямоугольника берётся в середине отрезка. Площадь всех прямоугольников складывается, чтобы получить приближённое значение интеграла. Метод трапеций использует не прямоугольники, а трапеции для оценки площади под графиком. На каждом шаге вычисляются значения функции на концах отрезка, и строится трапеция между ними. Сумма их площадей даёт приближение интеграла. Метод Симпсона аппроксимирует функцию не прямыми линиями, а параболами на парах соседних отрезков. Для этого требуется чётное число шагов. Метод учитывает значения функции в начале, конце и середине каждого двойного интервала, что делает его особенно эффективным для гладких функций. Метод Рунге–Ромберга используется для оценки погрешности и уточнения результата. Он сравнивает два значения интеграла, вычисленных с разными шагами, и на основе порядка точности конкретного метода уточняет результат, уменьшая ошибку.

Текст программы:

```
def f(x):
    return (16 - x**2) ** (1/2)

def integrate_rectangle_method(f, l, r, h):
    result = 0
    cur_x = l
    while cur_x < r:
        result += h * f((cur_x + cur_x + h) * 0.5)
        cur_x += h
    return result

def integrate_trapeze_method(f, l, r, h):
    result = 0
    cur_x = l
    while cur_x < r:
        result += h * 0.5 * (f(cur_x + h) + f(cur_x))
        cur_x += h
    return result

def integrate_simpson_method(f, l, r, h):
    result = 0
    cur_x = l + h
```

```
    while cur_x < r:
        result += f(cur_x - h) + 4 * f(cur_x) + f(cur_x + h)
        cur_x += 2 * h
    return result * h / 3

def runge_rombert_method(h1, h2, integral1, integral2, p):
    return (integral1 - integral2) / ((h2 / h1)**p - 1), integral1 +
(integral1 - integral2) / ((h2 / h1)**p - 1)

def main():
    with open('input.txt', 'r') as file:
        data = [list(map(float, line.split())) for line in
file.readlines()]
    l, r = data[0][0], data[0][1]
    h1, h2 = data[1][0], data[1][1]

    int_rectangle_h1 = integrate_rectangle_method(f, l, r, h1)
    int_rectangle_h2 = integrate_rectangle_method(f, l, r, h2)
    rectangle_err, rectangle_rr = runge_rombert_method(h1, h2,
int_rectangle_h1, int_rectangle_h2, 2)

    int_trapeze_h1 = integrate_trapeze_method(f, l, r, h1)
    int_trapeze_h2 = integrate_trapeze_method(f, l, r, h2)
    trapeze_err, trapeze_rr = runge_rombert_method(h1, h2,
int_trapeze_h1, int_trapeze_h2, 2)

    int_simpson_h1 = integrate_simpson_method(f, l, r, h1)
    int_simpson_h2 = integrate_simpson_method(f, l, r, h2)
    simpson_err, simpson_rr = runge_rombert_method(h1, h2,
int_simpson_h1, int_simpson_h2, 4)

    with open('output.txt', 'w') as file:
        file.write(f"Rectangle method:\n")
        file.write(f"Step = {h1}: integral = {int_rectangle_h1}\n")
        file.write(f"Step = {h2}: integral = {int_rectangle_h2}\n")
        file.write(f"Error rate: = {abs(rectangle_err)}\n")
        file.write(f"More accurate integral (runge_rombert): =
{rectangle_rr}\n\n")

        file.write(f"Trapeze method:\n")
        file.write(f"Step = {h1}: integral = {int_trapeze_h1}\n")
        file.write(f"Step = {h2}: integral = {int_trapeze_h2}\n")
        file.write(f"Error rate: = {abs(trapeze_err)}\n")
        file.write(f"More accurate integral (runge_rombert): =
{trapeze_rr}\n\n")

        file.write(f"Simpson method:\n")
        file.write(f"Step = {h1}: integral = {int_simpson_h1}\n")
        file.write(f"Step = {h2}: integral = {int_simpson_h2}\n")
        file.write(f"Error rate: = {abs(simpson_err)}\n")
        file.write(f"More accurate integral (runge_rombert): =
{simpson_rr}")

if __name__ == '__main__':
    main()
```

Входные данные:

```
-2 2
1.0 0.5
```

Результат работы программы:

```
Rectangle method:
Step = 1.0: integral = 15.353452420289436
Step = 0.5: integral = 15.317782947920461
Error rate: = 0.04755929649196607
More accurate integral (runge_rombert): = 15.30589312379747

Trapeze method:
Step = 1.0: integral = 15.21006830755259
Step = 0.5: integral = 15.281760363921011
Error rate: = 0.09558940849122877
More accurate integral (runge_rombert): = 15.305657716043818

Simpson method:
Step = 1.0: integral = 15.304023333311614
Step = 0.5: integral = 15.30565771604382
Error rate: = 0.0017433415810198009
More accurate integral (runge_rombert): = 15.305766674892634
```

Лабораторная работа № 4.1

Задание: реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге-Ромберга и путем сравнения с точным решением.

24	$x^2y''+(x+1)y'-y=0,$ $y(1)=2+e,$ $y'(1)=1,$ $x\in[1,2], h=0.1$	$y=x+1+xe^{1/x}$

Описание решения: метод Эйлера заменяет производную на конечную разность и делает шаг вперёд по наклону функции. Для системы это означает, что сначала вычисляется новое значение производной z, а затем с её помощью находится новое значение y. Этот метод имеет первый порядок точности, то есть ошибка уменьшается линейно при уменьшении шага. Метод Рунге-Кутты вместо одного значения наклона использует средневзвешенное четырех промежуточных оценок наклона на каждом шаге. Для каждого шага вычисляются четыре коэффициента (K_1, K_2, K_3, K_4), которые затем усредняются с весами. Метод имеет четвёртый порядок точности, то есть ошибка уменьшается пропорционально четвёртой степени шага. Метод Адамса строит приближение на основе уже известных значений функции: первые 4 точки, найденные методом Рунге–Кутты. Здесь применяется формула Адамса–Башфорта четвёртого порядка точности, которая использует значения правых частей уравнений на предыдущих шагах. Метод Рунге-Ромберга позволяет оценить погрешность численного решения. Он сравнивает два решения, полученных с разными шагами сетки, и на основе порядка точности метода вычисляет приближённую ошибку.

Текст программы:

```
from math import e, exp
import numpy as np
import matplotlib.pyplot as plt

'''
Given:
x**2 * y'' + (x + 1) * y' - y = 0
y(1) = 2 + e
y'(1) = 1
```

```
Converted:
y' = g(x, y, z) = z
z' = f(x, y, z) = (y - (x + 1) * y') / x**2
y(1) = 2 + e
z(1) = 1

Exact solution:
y = x + 1 + x * e**(1/x)
'''

def f(x, y, z):
    return (y - (x + 1) * z) / x**2

def g(x, y, z):
    return z

def exact_solution(x):
    return x + 1 + x * exp(1/x)

def euler_method(f, g, y0, z0, interval, h):
    l, r = interval
    x = [i for i in np.arange(l, r + h, h)]
    y = [y0]
    z = z0
    for i in range(len(x) - 1):
        z += h * f(x[i], y[i], z)
        y.append(y[i] + h * g(x[i], y[i], z))
    return x, y

def runge_kutta_method(f, g, y0, z0, interval, h, return_z=False):
    l, r = interval
    x = [i for i in np.arange(l, r + h, h)]
    y = [y0]
    z = [z0]
    for i in range(len(x) - 1):
        K1 = h * g(x[i], y[i], z[i])
        L1 = h * f(x[i], y[i], z[i])
        K2 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * K1, z[i] + 0.5 * L1)
        L2 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * K1, z[i] + 0.5 * L1)
        K3 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * K2, z[i] + 0.5 * L2)
        L3 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * K2, z[i] + 0.5 * L2)
        K4 = h * g(x[i] + h, y[i] + K3, z[i] + L3)
        L4 = h * f(x[i] + h, y[i] + K3, z[i] + L3)
        delta_y = (K1 + 2 * K2 + 2 * K3 + K4) / 6
        delta_z = (L1 + 2 * L2 + 2 * L3 + L4) / 6
        y.append(y[i] + delta_y)
        z.append(z[i] + delta_z)

    if not return_z:
        return x, y
    else:
        return x, y, z

def adams_method(f, g, y0, z0, interval, h):
    x_runge, y_runge, z_runge = runge_kutta_method(f, g, y0, z0,
interval, h, return_z=True)
    x = x_runge
    y = y_runge[:4]
    z = z_runge[:4]
    for i in range(3, len(x_runge) - 1):
        z_i = z[i] + h * (55 * f(x[i], y[i], z[i]) -
                          59 * f(x[i - 1], y[i - 1], z[i - 1]) +
                          37 * f(x[i - 2], y[i - 2], z[i - 2]) -
                          9 * f(x[i - 3], y[i - 3], z[i - 3])) / 24
        z.append(z_i)
        y_i = y[i] + h * (55 * g(x[i], y[i], z[i]) -
```

```
                            59 * g(x[i - 1], y[i - 1], z[i - 1]) +
                            37 * g(x[i - 2], y[i - 2], z[i - 2]) -
                            9 * g(x[i - 3], y[i - 3], z[i - 3])) / 24
            y.append(y_i)
    return x, y

def runge_romberg_method(h1, h2, y1, y2, p):
    assert h1 == h2 * 2
    norm = 0
    for i in range(len(y1)):
        norm += (y1[i] - y2[i * 2]) ** 2
    return norm ** 0.5 / (2**p - 1)

def main():
    with open('input.txt', 'r') as file:
        data = [list(map(float, line.split())) for line in
file.readlines()]
    y0 = data[0][0] + e
    dy0 = data[1][0]
    interval = (data[2][0], data[2][1])
    h = data[3][0]

    x_euler, y_euler = euler_method(f, g, y0, dy0, interval, h)
    _, y_euler2 = euler_method(f, g, y0, dy0, interval, h/2)

    x_runge, y_runge = runge_kutta_method(f, g, y0, dy0, interval, h)
    _, y_runge2 = runge_kutta_method(f, g, y0, dy0, interval, h/2)

    x_adams, y_adams = adams_method(f, g, y0, dy0, interval, h)
    _, y_adams2 = adams_method(f, g, y0, dy0, interval, h/2)

    x_exact = [i for i in np.arange(interval[0], interval[1] + h, h)]
    y_exact = [exact_solution(x_i) for x_i in x_exact]

    plt.plot(x_euler, y_euler, label=f'Euler, step={h}')
    plt.plot(x_runge, y_runge, label=f'Runge-Kutta, step={h}')
    plt.plot(x_adams, y_adams, label=f'Adams, step={h}')
    plt.plot(x_exact, y_exact, label='Exact solution')
    plt.legend()
    plt.show()

    with open('output.txt', 'w') as file:
        file.write(f"Given:\nx**2 * y'' + (x + 1) * y' - y = 0\ny(1) =
2 + e\ny\'(1) = 1\n")
        file.write(f"Converted:\ny' = g(x, y, z) = z\nz' = f(x, y, z)
= (y - (x + 1) * y') / x**2\ny(1) = 2 + e\nz(1) = 1\n")
        file.write(f"Exact solution:\ny = x + 1 + x * e**(1/x)\n\n")
        file.write(f"Euler method:\n")
        file.write(f"Solution (step = {h}):\n{[float(y) for y in
y_euler]}\n")
        file.write(f"Error rate (runge_romberg) =
{runge_romberg_method(h, h/2, y_euler, y_euler2, 1)}\n\n")
        file.write(f"Runge-Kutta method:\n")
        file.write(f"Solution (step = {h}):\n{[float(y) for y in
y_runge]}\n")
        file.write(f"Error rate (runge_romberg) =
{runge_romberg_method(h, h/2, y_runge, y_runge2, 4)}\n\n")
        file.write(f"Adams method:\n")
        file.write(f"Solution (step = {h}):\n{[float(y) for y in
y_adams]}\n")
        file.write(f"Error rate (runge_romberg) =
{runge_romberg_method(h, h/2, y_adams, y_adams2, 4)}\n\n")
        file.write(f"Exact solution:\n{[float(y) for y in y_exact]}")

if __name__ == '__main__':
    main()
```

Входные данные:

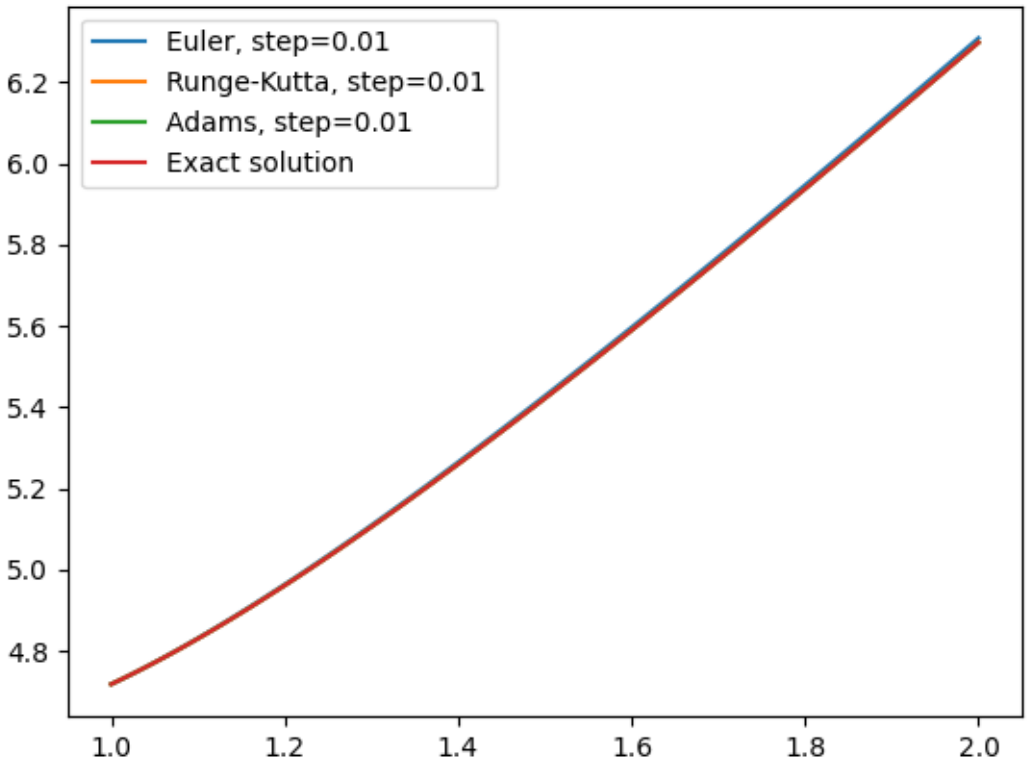
```
2
1
1 2
0.01
```

Результат работы программы:

```
Given:
x**2 * y'' + (x + 1) * y' - y = 0
y(1) = 2 + e
y'(1) = 1
Converted:
y' = g(x, y, z) = z
z' = f(x, y, z) = (y - (x + 1) * y') / x**2
y(1) = 2 + e
z(1) = 1
Exact solution:
y = x + 1 + x * e**(1/x)

Euler method:
Solution (step = 0.01):
[4.718281828459045, 4.728553656641891, ..., 6.28999371843019,
6.308312136145728]
Error rate (runge_romberg) = 0.03376715465452441

Runge-Kutta method:
Solution (step = 0.01):
[4.718281828459045, 4.728415953911749, ..., 6.279209300914516,
6.297442542299936]
Error rate (runge_romberg) = 5.02916105876833e-10

Adams method:
Solution (step = 0.01):
[4.718281828459045, 4.728415953911749, ..., 6.279208898969781,
6.297442135277896]
Error rate (runge_romberg) = 1.271288999607182e-07

Exact solution:
[4.718281828459045,    4.728415953843372,    ...,    6.279209300015206,
6.297442541400258]
```

(Шаг заменен на 0.01 для большей точности. Также показаны не все значения, промежуточные заменены на …)

График:


Euler, step=0.01
Runge-Kutta, step=0.01
Adams, step=0.01
Exact solution
6.2
6.0
5.8
5.6
5.4
5.2
5.0
4.8
1.0
1.2
1.4
1.6
1.8
2.0

Лабораторная работа № 4.2

Задание: реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге-Ромберга и путем сравнения с точным решением.

24	$(x^2+1)y''-2y=0$ $y'(0)=2$ $y(1)= 3+\frac{\pi}{2}$	$y(x)=x^2+x+1+$ $+(x^2+1)\text{arctg}(x)$

Описание решения: метод стрельбы превращает краевую задачу в задачу Коши. Подбирается неизвестное начальное условие так, чтобы при интегрировании через метод Рунге–Кутты решение достигло нужного значения на другом конце области. Это похоже на "выстрел" с одного конца, чтобы попасть в цель на другом. Используется линейная интерполяция или метод секущих для подбора правильного начального условия. Конечно-разностный метод заменяет производные в дифференциальном уравнении их конечно-разностными аналогами. Таким образом, уравнение преобразуется в систему линейных алгебраических уравнений, которая затем решается методом прогонки (трёхдиагональной матрицы). Этот метод позволяет сразу найти значения решения во всех точках сетки. С помощью метода Рунге-Ромберга оценивается погрешность численного решения: сравниваются результаты, полученные с разными шагами сетки, и вычисляется приближённая ошибка.

Текст программы:

```
from math import atan, pi
import numpy as np
import matplotlib.pyplot as plt

def runge_kutta_method(f, g, y0, z0, interval, h):
    l, r = interval
    x = [i for i in np.arange(l, r + h, h)]
    y = [y0]
    z = [z0]
    for i in range(len(x) - 1):
        K1 = h * g(x[i], y[i], z[i])
        L1 = h * f(x[i], y[i], z[i])
        K2 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * K1, z[i] + 0.5 * L1)
        L2 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * K1, z[i] + 0.5 * L1)
        K3 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * K2, z[i] + 0.5 * L2)
        L3 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * K2, z[i] + 0.5 * L2)
        K4 = h * g(x[i] + h, y[i] + K3, z[i] + L3)
```

```
        L4 = h * f(x[i] + h, y[i] + K3, z[i] + L3)
        delta_y = (K1 + 2 * K2 + 2 * K3 + K4) / 6
        delta_z = (L1 + 2 * L2 + 2 * L3 + L4) / 6
        y.append(y[i] + delta_y)
        z.append(z[i] + delta_z)

    return x, y, z

def tridiagonal_solve(A, b):
    n = len(A)
    # Step 1. Forward
    v = [0 for _ in range(n)]
    u = [0 for _ in range(n)]
    v[0] = A[0][1] / -A[0][0]
    u[0] = b[0] / A[0][0]
    for i in range(1, n-1):
        v[i] = A[i][i+1] / (-A[i][i] - A[i][i-1] * v[i-1])
        u[i] = (A[i][i-1] * u[i-1] - b[i]) / (-A[i][i] - A[i][i-1] *
v[i-1])
    v[n-1] = 0
    u[n-1] = (A[n-1][n-2] * u[n-2] - b[n-1]) / (-A[n-1][n-1] - A[n-
1][n-2] * v[n-2])

    # Step 2. Backward
    x = [0 for _ in range(n)]
    x[n-1] = u[n-1]
    for i in range(n-1, 0, -1):
        x[i-1] = v[i-1] * x[i] + u[i-1]
    return x

def f(x, y, z):
    return 2 * y / (x**2 + 1)

def g(x, y, z):
    return z

# y'' + p_fd(x)y' + q_fd(x)y = f_fd(x)

def p_fd(x):
    return 0

def q_fd(x):
    return -2 / (x**2 + 1)

def f_fd(x):
    return 0

def exact_solution(x):
    return x**2 + x + 1 + (x**2 + 1) * atan(x)

def shooting_method(f, g, dy0, y1_target, interval, h, eps=1e-6,
max_iter=100):
    n_prev = 0.0
    n = 1.0
    for iter_count in range(max_iter):
        x, y1, z1 = runge_kutta_method(f, g, n_prev, dy0, interval, h)
        val1 = y1[-1]  # y(1)

        x, y2, z2 = runge_kutta_method(f, g, n, dy0, interval, h)
        val2 = y2[-1]

        if abs(val1 - y1_target) < eps:
            return x, y1, z1, n_prev, iter_count
        if abs(val2 - y1_target) < eps:
            return x, y2, z2, n, iter_count
```

```
            slope = (val2 - val1) / (n - n_prev)
            n_new = n - (val2 - y1_target) / slope

            n_prev, n = n, n_new
            val1, val2 = val2, val1

        x, y, z = runge_kutta_method(f, g, n, dy0, interval, h)
        return x, y, z, n, max_iter

    def finite_difference_method(p, q, f, y0, yn, interval, h):
        A = []
        B = []
        rows = []
        a, b = interval
        x = np.arange(a, b + h, h)
        n = len(x)

        # Creating tridiagonal matrix
        for i in range(n):
            if i == 0:
                rows.append(1)
            else:
                rows.append(0)
        A.append(rows)
        B.append(y0)

        for i in range(1, n - 1):
            rows = []
            B.append(f(x[i]))
            for j in range(n):
                if j == i - 1:
                    rows.append(1 / h ** 2 - p(x[i]) / (2 * h))
                elif j == i:
                    rows.append(-2 / h ** 2 + q(x[i]))
                elif j == i + 1:
                    rows.append(1 / h ** 2 + p(x[i]) / (2 * h))
                else:
                    rows.append(0)
            A.append(rows)

        rows = []
        B.append(yn)
        for i in range(n):
            if i == n - 1:
                rows.append(1)
            else:
                rows.append(0)

        A.append(rows)
        y = tridiagonal_solve(A, B)
        return x, y

    def runge_romberg_method(h1, h2, y1, y2, p):
        assert h1 == h2 * 2
        norm = 0
        for i in range(len(y1)):
            norm += (y1[i] - y2[i * 2]) ** 2
        return norm ** 0.5 / (2 ** p - 1)

    def main():
        with open('input.txt', 'r') as file:
            data = [list(map(float, line.split())) for line in
    file.readlines()]
        y0 = data[0][0]
        y1 = data[1][0] + pi / 2
        interval = (data[2][0], data[2][1])
```

```
    h = data[3][0]

    x_shooting, y_shooting, z_shooting, y01, iter_count1 =
shooting_method(f, g, y0, y1, interval, h)
    plt.plot(x_shooting, y_shooting, label=f'shooting method,
step={h}')
    x_shooting2, y_shooting2, z_shooting2, y02, iter_count2 =
shooting_method(f, g, y0, y1, interval, h / 2)

    x_fd, y_fd = finite_difference_method(p_fd, q_fd, f_fd, y01, y1,
interval, h)
    plt.plot(x_fd, y_fd, label=f'finite difference method, step={h}')
    x_fd2, y_fd2 = finite_difference_method(p_fd, q_fd, f_fd, y02, y1,
interval, h / 2)

    x_exact = [i for i in np.arange(interval[0], interval[1] + h, h)]
    y_exact = [exact_solution(x_i) for x_i in x_exact]
    plt.plot(x_exact, y_exact, label='exact solution')

    with open('output.txt', 'w') as file:
        file.write(f"Given:\n(x**2 + 1) * y'' -2 * y = 0\ny\'(0) =
2\ny(1) = 3 + pi/2\n")
        file.write(f"Converted:\ny' = g(x, y, z) = z\nz' = f(x, y, z)
= 2 * y / (x**2 + 1) / x**2\ny(1) = 3 + pi/2\nz(0) = 2\n")
        file.write(f"Exact solution:\nx**2 + x + 1 + (x**2 + 1) *
arctg(x)\n\n")

        file.write(f"Shooting:\n")
        file.write(f"X values (step = {h}):\n{[float(x) for x in
x_shooting]}\n")
        file.write(f"Solution (step = {h}):\n{[float(y) for y in
y_shooting]}\n")
        file.write(f"Iter_count: {iter_count1}\n")
        file.write(f"Error rate (runge_romberg) =
{runge_romberg_method(h, h/2, y_shooting, y_shooting2, 4)}\n\n")

        file.write(f"Finite Difference method:\n")
        file.write(f"X values (step = {h}):\n{[float(x) for x in
x_fd]}\n")
        file.write(f"Solution (step = {h}):\n{[float(y) for y in
y_fd]}\n")
        file.write(f"Error rate (runge_romberg) =
{runge_romberg_method(h, h/2, y_fd, y_fd2, 2)}\n\n")

        file.write(f"Exact solution:\n{[float(y) for y in y_exact]}")

    plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

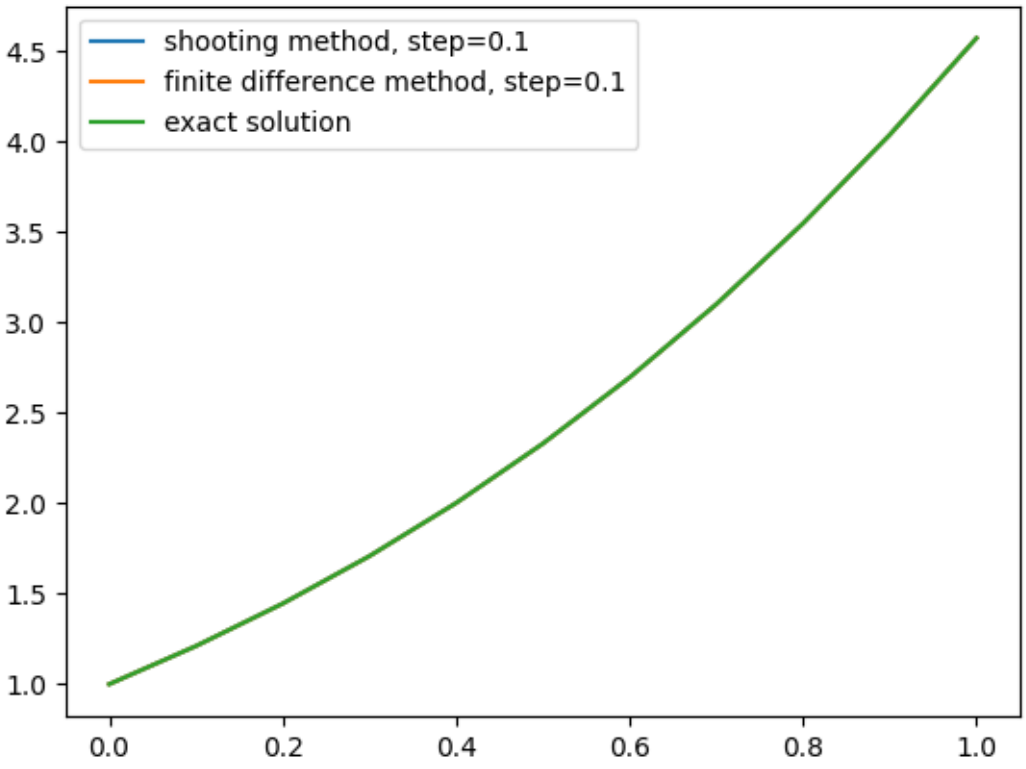
Входные данные:

```
2
3
0 1
0.1
```

Результат работы программы:

```
Given:
(x**2 + 1) * y'' -2 * y = 0
y'(0) = 2
y(1) = 3 + pi/2
Converted:
y' = g(x, y, z) = z
z' = f(x, y, z) = 2 * y / (x**2 + 1) / x**2
y(1) = 3 + pi/2
z(0) = 2
Exact solution:
x**2 + x + 1 + (x**2 + 1) * arctg(x)

Shooting:
X values (step = 0.1):
[0.0, 0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6000000000000001,
0.7000000000000001, 0.8, 0.9, 1.0]
Solution (step = 0.1):
[1.0000045177864505, 1.210669567120591, 1.445295310048066,
1.7076914976417905, 2.0013906045753176, 2.3295622836409953,
2.694972811942927, 3.0999834572229794, 3.5465763583065044,
4.036395956843359, 4.5707963267948974]
Iter_count: 1
Error rate (runge_romberg) = 6.764074242819345e-07

Finite Difference method:
X values (step = 0.1):
[0.0, 0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6000000000000001,
0.7000000000000001, 0.8, 0.9, 1.0]
Solution (step = 0.1):
[1.0000045177864505, 1.21056887186776, 1.445104886778134,
1.7074313802803955, 2.001086889934591, 2.3292438976911063,
2.6946688078106793, 3.0997212003980565, 3.5463805889639315,
4.036288521297659, 4.570796326794897]
Error rate (runge_romberg) = 0.00017783867961230332

Exact solution:
[1.0,   1.2106653390160738,   1.445291382243876,   1.7076879059808754,
2.0013873974503436,       2.3295595112510075,       2.694970520367995,
3.0999816869399215,       3.546575145246627,       4.0363953342335765,
4.570796326794897]
```

График:


shooting method, step=0.1
finite difference method, step=0.1
exact solution
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0