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}: $A * A^{-1} = E$.

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.

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.

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 $np.linalg.eig$.

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.

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.

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:

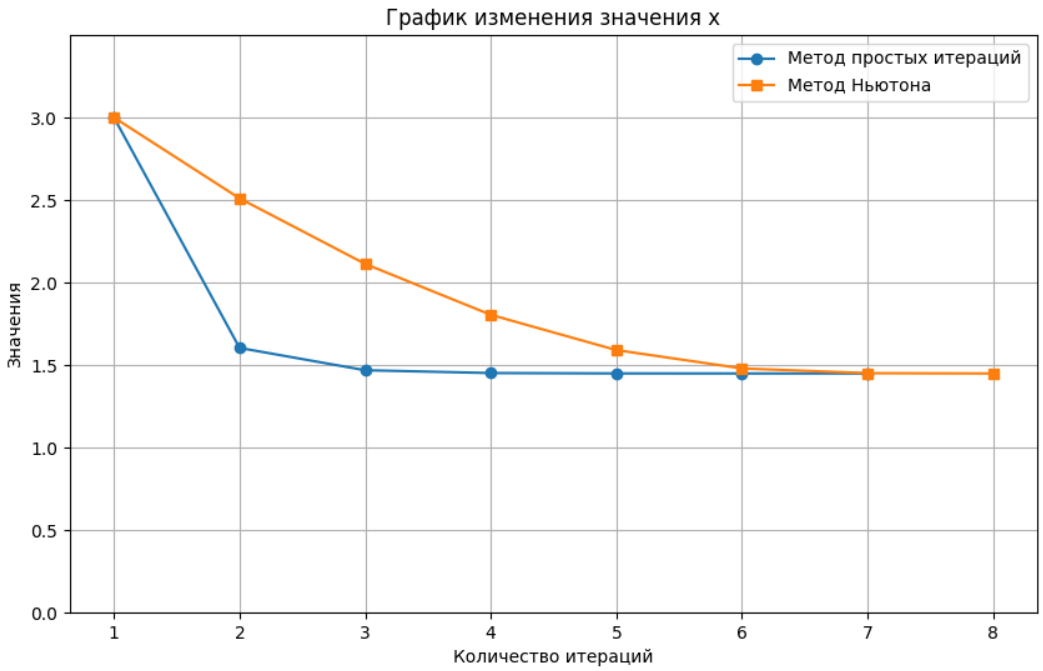
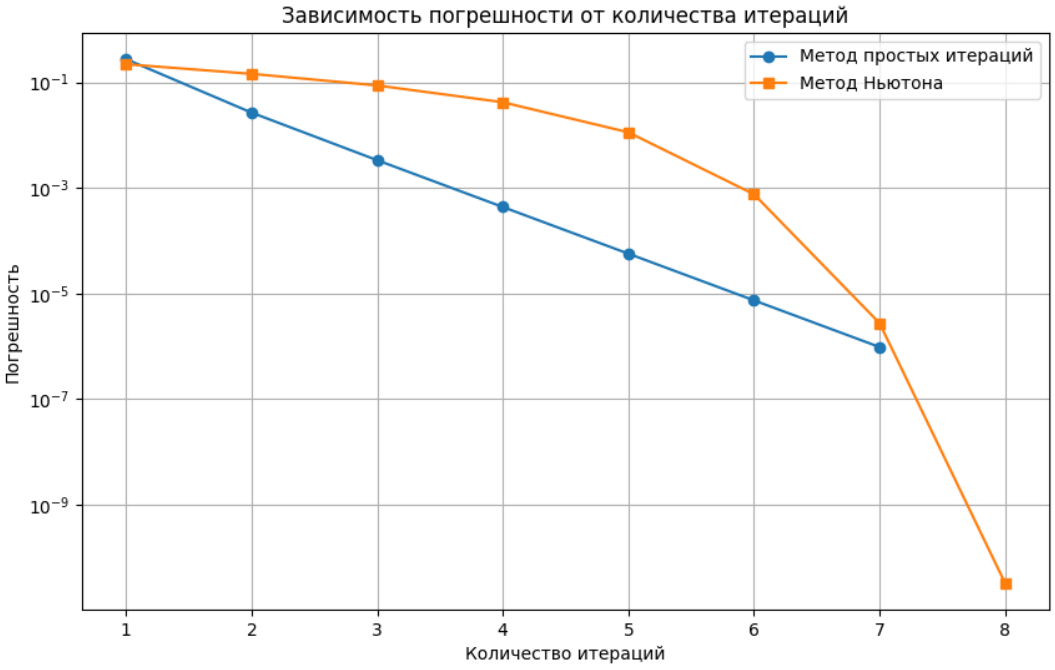


График изменения значения x
Метод простых итераций
Метод Ньютона
Значения
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7
8
Количество итераций


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.

Входные данные:

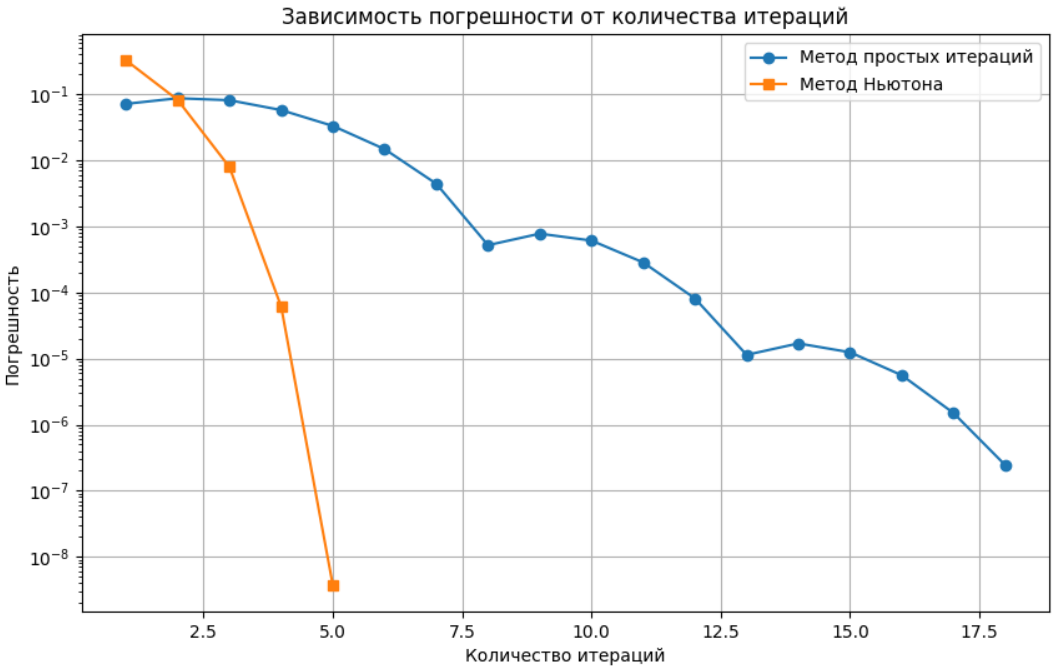
```
0.2 0.6
0.4 0.5
1e-6
```

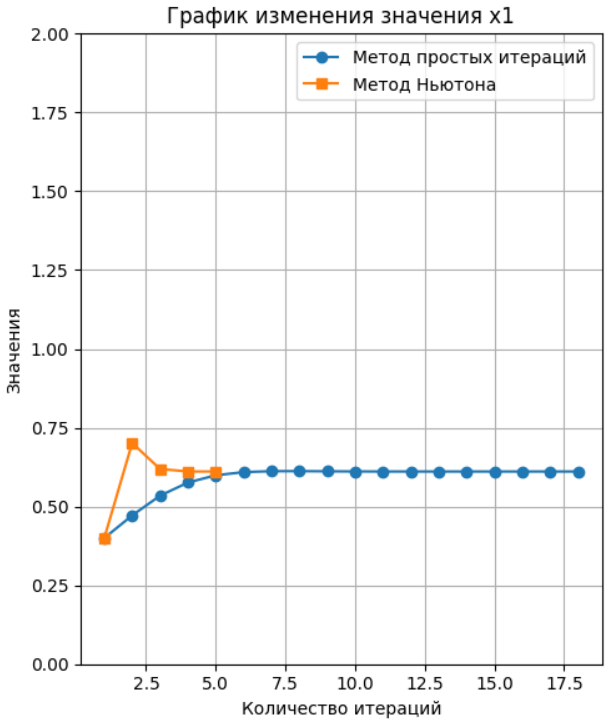
Результат работы программы:

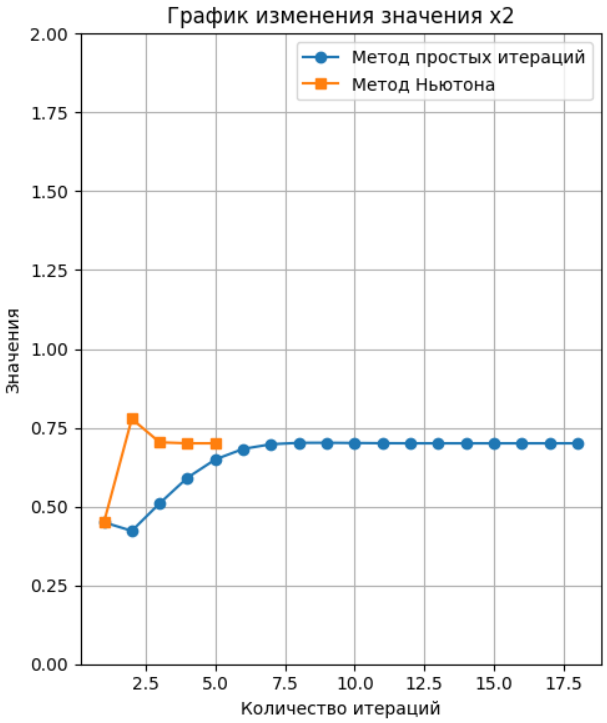
```
System of nonlinear equations:
 | 3*x1^2 - x1 + x2^2 - 1 = 0
 | x2 - tg(x1) = 0
Intervals: [0.2, 0.6], [0.4, 0.5]
Epsilon: 1e-06

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:







Лабораторная работа № 3.1

Задание: используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках X_i, $i = 0, \dots, 3$ 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.

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.

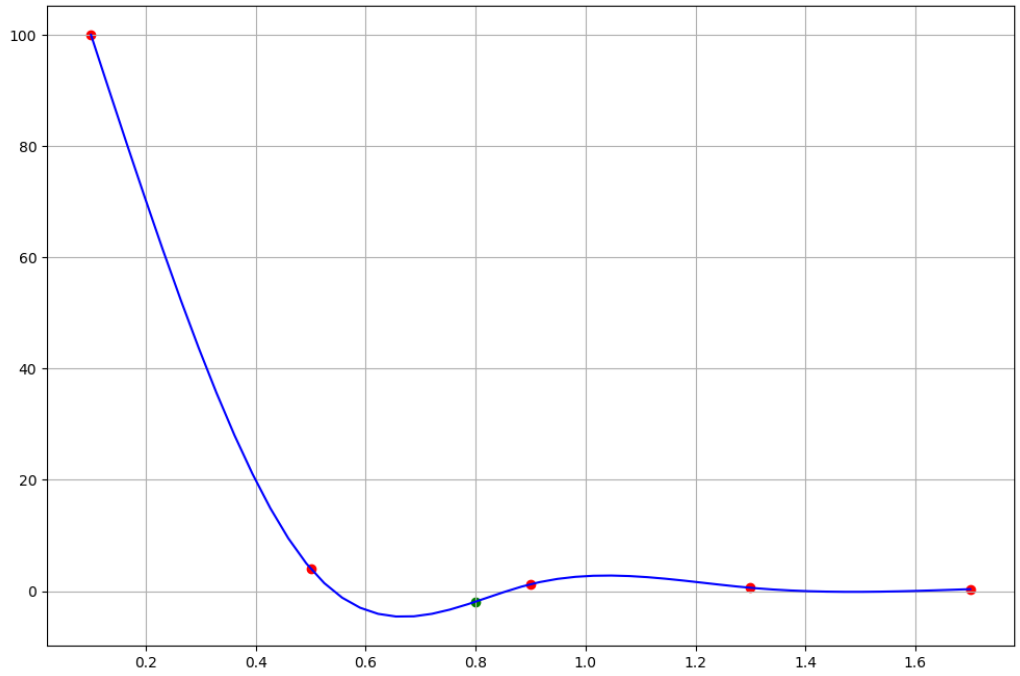
Входные данные:

```
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.

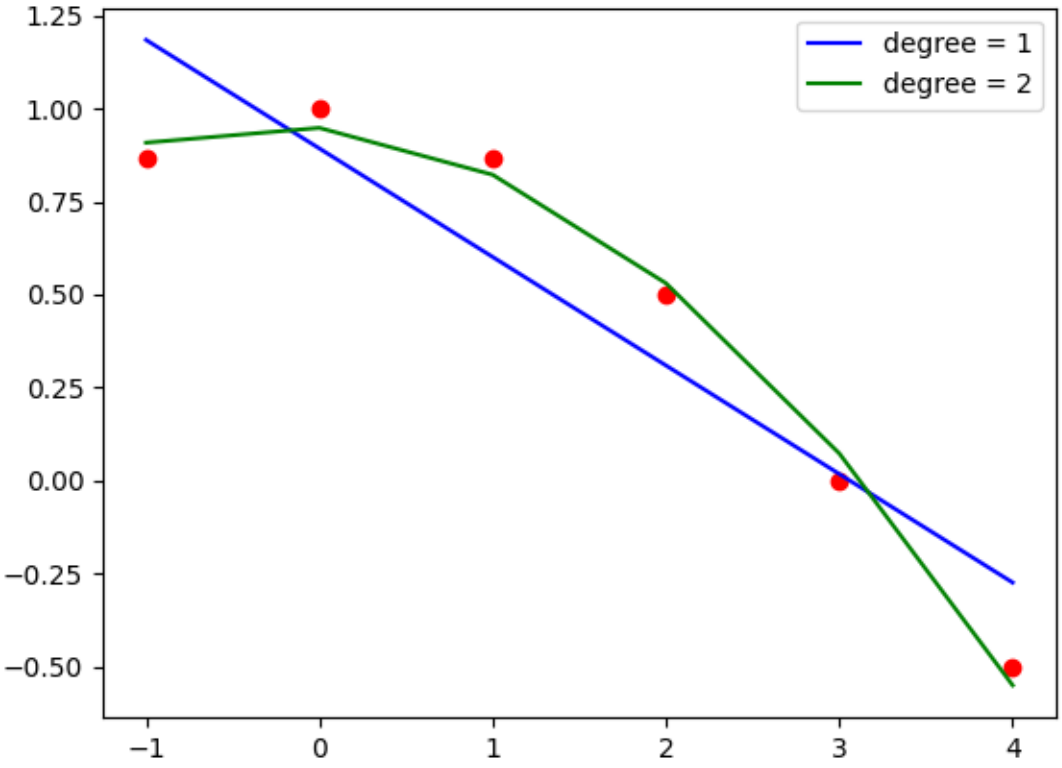
Входные данные:

```
-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 $scipy$.

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

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.

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: исходное дифференциальное уравнение преобразуем, заменяя y' на z. 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.

Входные данные:

```
2
1
1 2
0.01
```

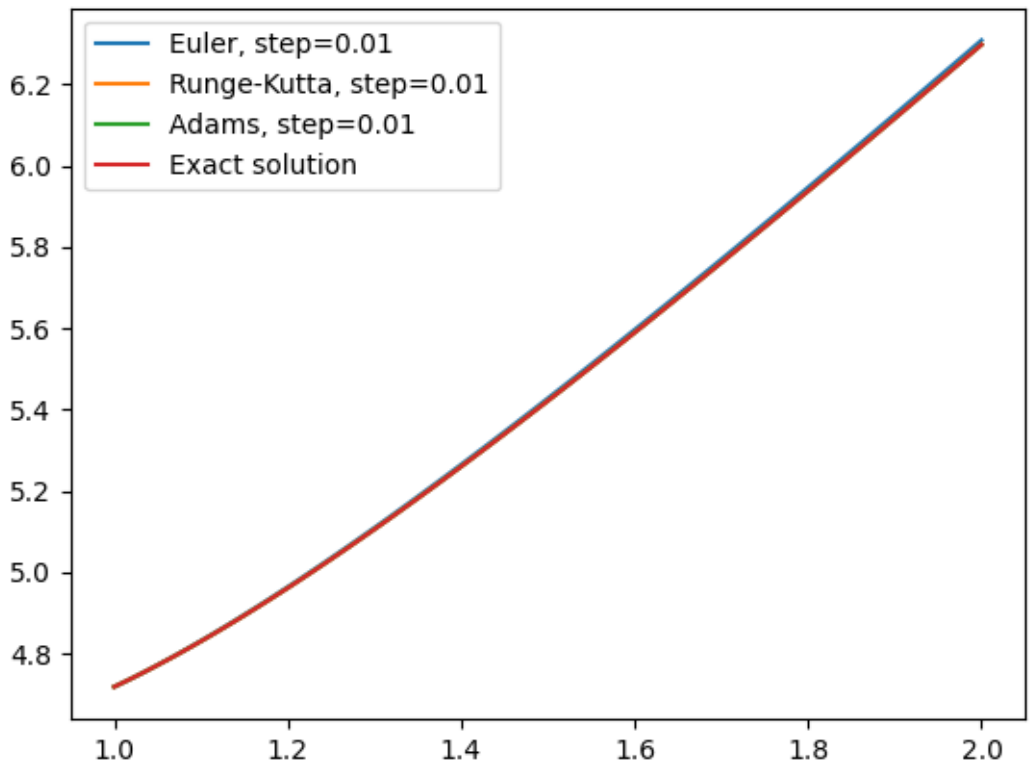
Результат работы программы:

```
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) = 0.0337671546545244 1

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:



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: исходное дифференциальное уравнение преобразуем, заменяя y' на z. 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.

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]
```

<u>График:</u>