МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

# ОТЧЁТ

по лабораторной работе №2

" РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХУРАВНЕНИЙ"

по дисциплине

*Вычислительная математика*

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Панкратова А.З.\_\_\_

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_Халеев А.А. \_

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_\_21-ВМз-4\_\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород

2023

**Тема работы:**

Решение систем линейных уравнений с одной неизвестной.

**Цель работы**:

Изучить численные методы и алгоритмы решения систем линейных уравнений

**Постановка задачи:**

Реализовать изученные алгоритмы решения систем линейных уравнений и провести сравнение методов.

**Вариант №7:**

Решить систему линейных уравнений методом Гаусса, итерационным методом и методом Гаусса-Зейделя. При необходимости преобразовать систему к диагонально преобладающему виду. Сделать оценку количества итераций для итерационных методов, сравнить. Точность ε=0.001.

Дана система n алгебраических уравнений с n неизвестными:

Эту систему можно записать в матричном виде:

A - квадратная матрица коэффициентов, X - вектор-столбец неизвестных, B - вектор-столбец свободных членов.

Численные методы решения систем линейных уравнений делятся на прямые и итерационные.

Прямые методы используют конечные соотношения для вычисления неизвестных. Эти методы сравнительно просты и пригодны для широкого класса систем. Недостатки: требуют хранения в памяти ЭВМ сразу всей матрицы A. При больших порядках системы расходуется много места в памяти и накапливается вычислительная погрешность. Кроме того, существенно возрастает время вычисления вектора X. Поэтому прямые методы обычно применяют при небольших порядках системы (n<200). Примеры прямых методов - метод определителей Крамера, метод Гаусса. Первый из них применяется крайне редко, так как с ростом n алгоритм нахождения определителей резко возрастает. Метод Гаусса будет подробно рассмотрен в дальнейшем.

Итерационные методы основаны на последовательных приближениях. Задается некоторое приближенное значение вектора X – начальное приближение. Затем с помощью некоторого алгоритма проводится первый цикл вычислений – итерация, в результате которого получается новое приближение вектора X. Итерации проводятся до получения решения с заданной точностью. Алгоритм решения систем линейных уравнений здесь более сложен, чем у прямых методов. Не всегда выполняется условие сходимости. Однако, в ряде случаев итерационные методы предпочтительнее. Они требуют хранения в памяти ЭВМ не всей матрицы A, а лишь нескольких векторов. Вычислительная погрешность практически не накапливается. Поэтому итерационные методы применимы и для больших порядков системы. Примеры - метод простой итерации и метод Зейделя.

***Метод Гаусса***

Метод основан на приведении матрицы системы к треугольному виду. Это достигается последовательным исключением неизвестных из уравнений системы. Сначала с помощью первого уравнения исключается x1 из всех последующих уравнений. Затем с помощью второго уравнения исключается x2 из последующих и т.д. Этот процесс называется прямым ходом метода Гаусса и продолжается до тех пор, пока в левой части последнего n-го уравнения не останется лишь один член с неизвестным xn.

В результате прямого хода система принимает вид:

Обратный ход метода Гаусса состоит в последовательном вычислении искомых неизвестных, начиная с xn и кончая x1.

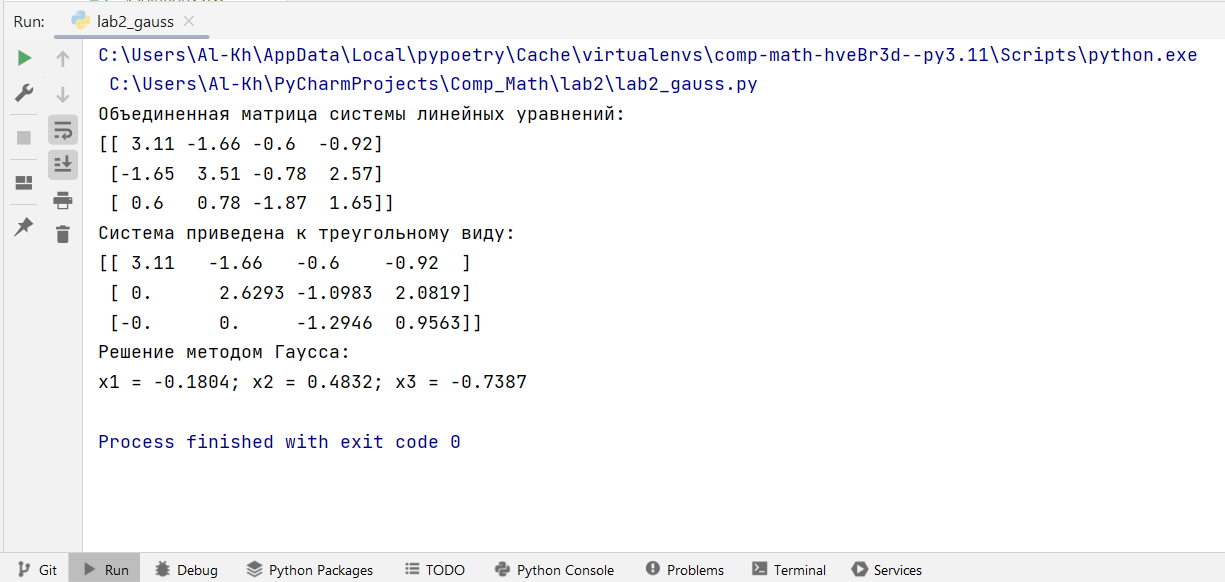
***Решение в Excel:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Решение системы методом Гаусса | | | | | |
| 3,1100 | -1,6600 | -0,6000 | -0,9200 |  |  |
| -1,6500 | 3,5100 | -0,7800 | 2,5700 |  |  |
| 0,6000 | 0,7800 | -1,8700 | 1,6500 |  |  |
|  |  |  |  |  |  |
| 1,0000 | -0,5338 | -0,1929 | -0,2958 |  |  |
| 0,0000 | 2,6293 | -1,0983 | 2,0819 |  |  |
| 0,0000 | 1,1003 | -1,7542 | 1,8275 |  |  |
|  |  |  |  |  |  |
| 1,0000 | -0,5338 | -0,1929 | -0,2958 | x1 | -0,1804 |
| 0,0000 | 1,0000 | -0,4177 | 0,7918 | x2 | 0,4832 |
| 0,0000 | 0,0000 | -1,2946 | 0,9563 | x3 | -0,7387 |
|  |  |  |  |  |  |

***Программа на Python:***

import numpy as np  
  
  
def gauss(a: np.array, b: np.array) -> np.ndarray:  
 *"""  
 Рассчитывает решение системы линейных уравнений с использованием метода Гаусса.  
  
 Параметры:  
 a (numpy.ndarray): Матрица коэффициентов системы линейных уравнений.  
 b (numpy.ndarray): Вектор правых частей системы линейных уравнений.  
  
 Возвращает:  
 x (numpy.ndarray): Вектор решения системы линейных уравнений.  
 """* n = len(a)  
 system = np.hstack([a, b.reshape(-1, 1)])  
 print(f"Объединенная матрица системы линейных уравнений:\n{system}")  
  
 for i in range(n):  
 *# нахождение индекса строки с максимальным абсолютным значением* max\_row\_index = np.abs(system[i:, i]).argmax() + i  
  
 *# перестановка строк* if i != max\_row\_index:  
 system[[i, max\_row\_index]] = system[[max\_row\_index, i]]  
  
 *# зануление i-тых элементов строк* for j in range(i + 1, n):  
 system[j] = system[j] - system[i] \* system[j, i] / system[i, i]  
 print(f"Система приведена к треугольному виду:\n{system.round(4)}")  
 x = np.zeros(n)  
 for i in range(n - 1, -1, -1):  
 x[i] = (system[i, -1] - np.dot(system[i, :-1], x)) / system[i, i]  
  
 return x  
  
  
def main() -> None:  
 *# Использование методов* A = np.array([[3.11, -1.66, -0.6],  
 [-1.65, 3.51, -0.78],  
 [0.6, 0.78, -1.87]])  
  
 B = np.array([-0.92, 2.57, 1.65])  
  
 solution = gauss(A, B)  
 print(f"Решение методом Гаусса:")  
 print("; ".join(f'x{i} = {x:.4f}' for i, x in enumerate(solution, 1)))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Вывод программы:



***Метод простой итерации (Якоби)***

Метод основан на последовательном сужении интервала, содержащего единственный корень уравнения 0 *f* (*x*) = до тех пор, пока не будет достигнута заданная точность ε. Пусть задан отрезок [*a*, *b*], содержащий один корень уравнения. Этот отрезок может быть предварительно найден с помощью шагового метода.

*Алгоритм метода*:  
1. Определить новое приближение корня x в середине отрезка [a, b]: x = (a + b) / 2.  
2. Найти значения функции в точках a и x: f(a) и f(x).  
3. Проверить условие f(a)\*f(x)<0. Если условие выполнено, то корень расположен на отрезке [a, x]. В этом случае необходимо точку b переместить в точку x (b = x). Если условие не выполнено, то корень расположен на отрезке [x, b]. В этом случае необходимо точку a переместить в точку x (a=x).  
4. Перейти к пункту 1 и вновь поделить отрезок пополам. Алгоритм продолжить до тех пор, пока не будет выполнено условие *f* (*x*) < ε .

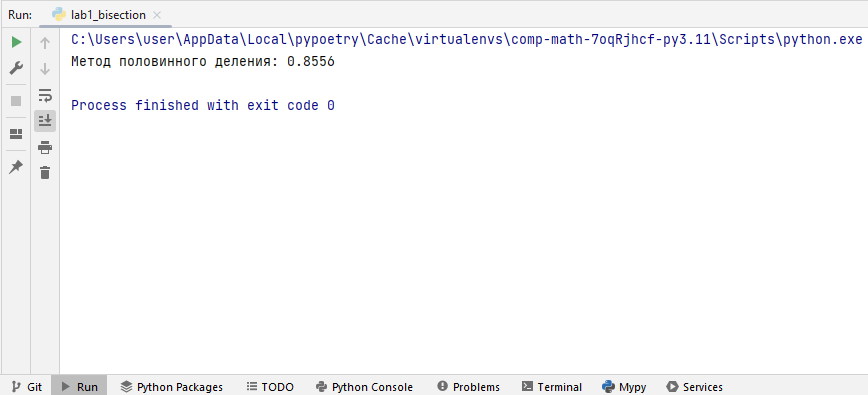
***Решение в Excel:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод половинного деления | | | | | |
| Начальное значение | 0,85 | | | | |
| Шаг табуляции | нет | | | | |
| Точность | 0,001 | | | | |
| a | x | b | f(a) | f(x) | f(a)\*f(x)<0 |
| 0,8500 | 0,8550 | 0,8600 | -0,0164 | -0,0013 | НЕТ |
| 0,8550 | 0,8575 | 0,8600 | -0,0013 | 0,0063 | ДА |
| 0,8550 | 0,8563 | 0,8575 | -0,0013 | 0,0025 | ДА |
| 0,8550 | 0,8556 | 0,8563 | -0,0013 | 0,0006 | СТОП |

***Программа на Python:***

def f(x: float) -> float:  
 *"""  
 Вычисляет значение функции f(x) = x \*\* 3 + 0.2 \* x \*\* 2 + 0.5 \* x - 1.2  
  
 Параметры:  
 x: Значение аргумента функции.  
  
 Возвращаемое значение:  
 Значение функции в точке x.  
 """* return x \*\* 3 + 0.2 \* x \*\* 2 + 0.5 \* x - 1.2  
  
  
def step\_method(func: callable, start: float, step: float) -> tuple[int, int]:  
 *"""  
 Находит отрезок, содержащий корень функции.  
  
 Параметры:  
 func: Функция f(x), корень которой необходимо найти.  
 start: Начало отрезка поиска.  
 step: Шаг при переборе точек на отрезке.  
  
 Возвращаемое значение:  
 Кортеж с началом и концом сегмента, содержащего корень функции.  
 """* x0, x1 = start, start + step  
 while func(x0) \* func(x1) >= 0:  
 x0, x1 = x1, x1 + step  
 return round(x0, 2), round(x1, 2)  
  
  
def bisection\_method(func: callable, a: float, b: float, epsilon: float) -> float:  
 *"""  
 Реализует метод половинного деления для численного решения уравнения  
 f(x) = 0 на заданном отрезке [a, b].  
  
 Параметры:  
 a: Начало отрезка.  
 b: Конец отрезка.  
 epsilon: Точность решения.  
  
 Возвращаемое значение:  
 Приближенное значение корня уравнения f(x) = 0.  
 """*  
  
 while abs(func(x := ((a + b) / 2))) >= epsilon:  
 if func(x) \* func(a) < 0:  
 b = x  
 else:  
 a = x  
 return x  
  
  
def main():  
 *# Использование методов* a, b = step\_method(func=f, start=0.8, step=0.01)  
 root\_bisection = bisection\_method(f,a, b, epsilon=0.001)  
 print(f"Метод половинного деления: {root\_bisection:.4f}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вывод программы:



Достоинство метода: более быстрая сходимость к заданной точности, чем у шагового. Недостаток: если на отрезке [a, b] содержится более одного корня, то метод не работает.

***Метод Ньютона (метод касательных)***

Задан отрезок [a, b], содержащий корень *f* (*x*) = 0. Уточнение значения корня производится путем использования уравнения касательной. В качестве начального приближения задается тот из концов отрезка [a, b], где значение функции и ее второй производной имеют одинаковые знаки (т.е. выполняется условие f(x0) \* f ``(x0) > 0). В точке f(x0) строится касательная к кривой y= F(x) и ищется ее пересечение с осью x. Точка пересечения принимается за новую итерацию. Итерационная формула имеет вид:  
   
Итерационный процесс продолжается до тех пор, пока не будет выполнено условие:

|f(x)| < ε, где ε - заданная точность.

***Решение в Excel:***

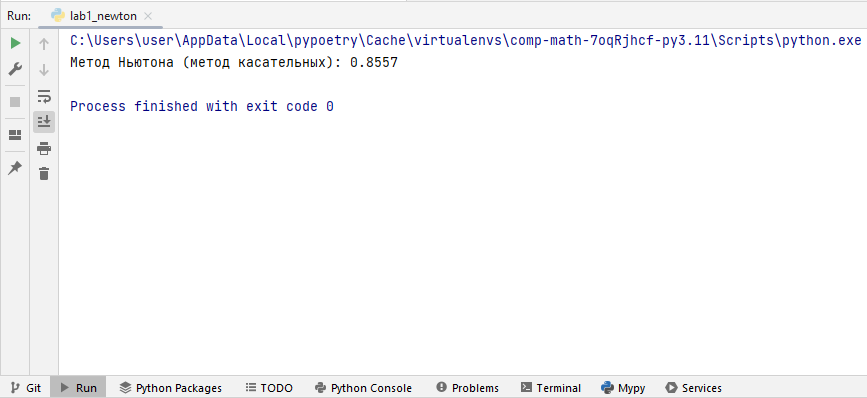
|  |  |  |
| --- | --- | --- |
| X | 0,8 | 0,9 |
| |  | | --- | |  | | -0,16 | 0,141 |
| |  | | --- | |  | | 2,74 | 3,29 |
| |  | | --- | |  | | 5,2 | 5,8 |
| Так как в правой точке отрезка вторая производная и значение функции имеют одинаковые знаки, то начальной точкой выбрана правая граница отрезка, содержащего корень: Х0 = 0.9 | | |
|  |

|  |  |
| --- | --- |
| Метод Ньютона | |
| Начальное значение | 0,9 |
| Шаг табуляции | нет |
| Точность | 0,001 |
| |  | | --- | |  | | | |
|
| |  | | --- | |  | |  |
| 0,9000 | 0,1410 |
| 0,8571 | 0,0052478 |
| **0,8554** | **0,0000082** |

***Программа на Python:***

from scipy.misc import derivative  
import warnings  
  
warnings.filterwarnings("ignore", category=DeprecationWarning)  
  
  
def f(x: float) -> float:  
 *"""  
 Вычисляет значение функции f(x) = x \*\* 3 + 0.2 \* x \*\* 2 + 0.5 \* x - 1.2  
  
 Параметры:  
 x: Значение аргумента функции.  
  
 Возвращаемое значение:  
 Значение функции в точке x.  
 """* return x \*\* 3 + 0.2 \* x \*\* 2 + 0.5 \* x - 1.2  
  
  
def step\_method(func: callable, start: float, step: float) -> tuple[int, int]:  
 *"""  
 Находит отрезок, содержащий корень функции.  
  
 Параметры:  
 func: Функция f(x), корень которой необходимо найти.  
 start: Начало отрезка поиска.  
 step: Шаг при переборе точек на отрезке.  
  
 Возвращаемое значение:  
 Кортеж с началом и концом отрезка, содержащего корень функции.  
 """* x0, x1 = start, start + step  
 while func(x0) \* func(x1) >= 0:  
 x0, x1 = x1, x1 + step  
 return x0, x1  
  
  
def newton\_method(func: callable, a: float, b: float, epsilon: float) -> float:  
 *"""  
 Реализует метод Ньютона (метод касательных)  
 для численного решения уравнения f(x) = 0 на заданном отрезке [a, b].  
  
 Параметры:  
 func: Функция f(x), корень которой необходимо найти.  
 a: Начало отрезка.  
 b: Конец отрезка.  
 epsilon: Точность решения.  
 Возвращаемое значение:  
 Приближенное значение корня уравнения f(x) = 0.  
 """* x = a if func(a) \* derivative(func, x0=a, n=2) > 0 else b *# начальное приближение* next\_value = lambda x: x - func(x) / derivative(func, x0=x) *# итерационная формула* while func(x) >= epsilon:  
 x = next\_value(x)  
  
 return x  
  
  
def main():  
 *# Использование методов* a, b = step\_method(func=f, start=0.8, step=0.01)  
 root\_newton = newton\_method(f, a, b, epsilon=0.001)  
 print(f"Метод Ньютона (метод касательных): {root\_newton:.4f}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вывод программы:



Достоинство метода: очень быстрая сходимость к заданной точности.

Недостаток: громоздкий алгоритм, на каждой итерации необходимо вычислять значение функции и ее первой производной.

***Метод простой итерации (Якоби)***

Метод основан на замене исходного уравнения на эквивалентное

Функция выбирается таким образом, чтобы на обоих концах отрезка [a, b] выполнялось условие сходимости . В этом случае в качестве начального приближения можно выбрать любой из концов отрезка.

Итерационная формула имеет вид:

Итерационный процесс продолжается до тех пор, пока не будет выполнено условие:

На первом этапе нам необходимо выбрать функцию ϕ(x), удовлетворяющую условию сходимости.

Исходное уравнение:

Запишем исходное уравнение в виде:

Тогда:

Условие сходимости выполнено, поскольку

Следовательно, итерационная формула имеет вид:

В качестве начального приближения можно выбрать любой из концов отрезка, например:

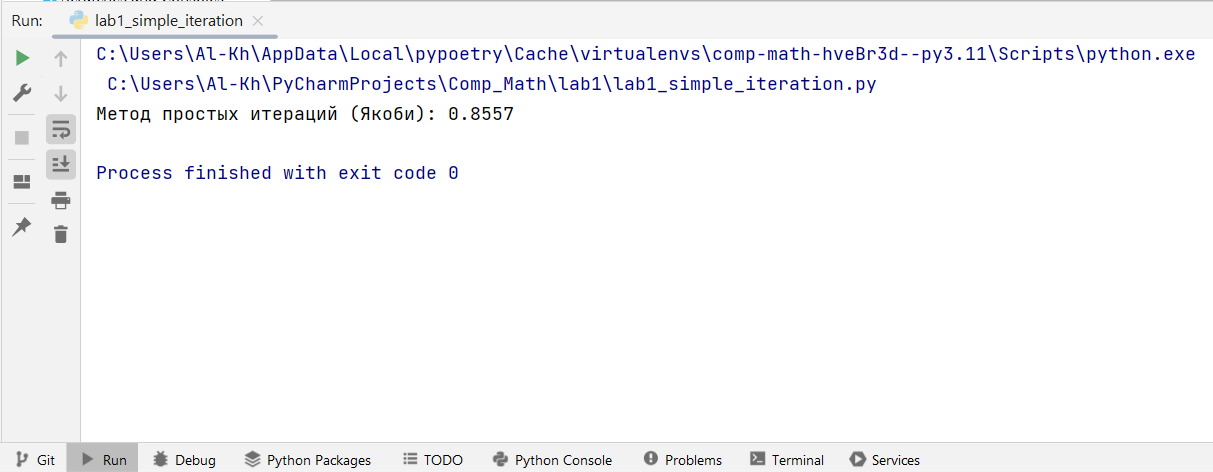
***Решение в Excel:***

|  |  |
| --- | --- |
| Метод простой итерации (Якоби) | |
| Начальное значение | 0,8 |
| Шаг табуляции | нет |
| Точность | 0,001 |
| Эквивалентная формула | |
|  |
|  |  |  |
| 0,8759 | 0,0634 |  |
| 0,8474 | -0,0240 |  |
| 0,8585 | 0,0093 |  |
| 0,8542 | -0,0035 |  |
| 0,8559 | 0,0014 |  |
| **0,8552** | **-0,0005** |  |

***Программа на Python:***

from scipy.misc import derivative  
import warnings  
  
warnings.filterwarnings("ignore", category=DeprecationWarning)  
  
  
def f(x: float) -> float:  
 *"""  
 Вычисляет значение функции f(x) = x \*\* 3 + 0.2 \* x \*\* 2 + 0.5 \* x - 1.2  
  
 Параметры:  
 x: Значение аргумента функции.  
  
 Возвращаемое значение:  
 Значение функции в точке x.  
 """* return x \*\* 3 + 0.2 \* x \*\* 2 + 0.5 \* x - 1.2  
  
  
def step\_method(func: callable, start: float, step: float) -> tuple[int, int]:  
 *"""  
 Находит отрезок, содержащий корень функции.  
  
 Параметры:  
 func: Функция f(x), корень которой необходимо найти.  
 start: Начало отрезка поиска.  
 step: Шаг при переборе точек на отрезке.  
  
 Возвращаемое значение:  
 Кортеж с началом и концом сегмента, содержащего корень функции.  
 """* x0, x1 = start, start + step  
 while func(x0) \* func(x1) >= 0:  
 x0, x1 = x1, x1 + step  
 return round(x0, 2), round(x1, 2)  
  
  
def simple\_iteration\_method(func: callable, a: float, b: float, epsilon: float) -> float:  
 *"""  
 Реализует метод простых итераций для численного решения уравнения  
 f(x) = 0 на заданном отрезке [a, b].  
 Реализовано для функции f(x) = x \*\* 3 + 0.2 \* x \*\* 2 + 0.5 \* x - 1.2 = 0  
 Тогда g(x) = (1.2 - 0.2 \* x \*\* 2 - 0.5 \* x) \*\* (1 / 3) является эквивалентной,  
 так как g'(x) меньше 1 для обоих концов отрезка  
  
 Параметры:  
 a: Начало отрезка.  
 epsilon: Точность решения.  
  
 Возвращаемое значение:  
 Приближенное значение корня уравнения f(x) = 0.  
 """* g = lambda x: (1.2 - 0.2 \* x \*\* 2 - 0.5 \* x) \*\* (1 / 3)  
 if any(derivative(g, x0=point) >= 1 for point in [a, b]):  
 raise ValueError("Условие сходимости не выполнено на заданном отрезке")  
 xi = g(a)  
 while abs(func(xi)) >= epsilon:  
 xi = g(xi)  
 return xi  
  
  
def main():  
 *# Использование методов* a, b = step\_method(func=f, start=0.8, step=0.01)  
 root\_jacobi = simple\_iteration\_method(f, a, b, epsilon=0.001)  
 print(f"Метод простых итераций (Якоби): {root\_jacobi:.4f}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вывод программы:



Достоинство метода: простота алгоритма.

Недостатки: возможные сложности с выбором функции ϕ(x); более медленное достижение заданной точности, чем у других методов уточнения.

**Вывод:**

В данной лабораторной работе были изучены следующие методы численного решения нелинейных уравнений:

- шаговый метод

- метод половинного деления

- метод Ньютона

- метод простых итераций (Якоби)

Также были проведены программные вычисления с помощью MS Excel и языка программирования Python для уравнения, полученного согласно варианту.

Все решения сходятся на заданной точности. В процессе решения были проанализированы особенности, плюсы и минусы каждого из использованных методов.