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 №4

" ОПРЕДЕЛЕННЫЙ ИНТЕГРАЛ"

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:**

Численное интегрирование функций методами прямоугольников, трапеций, Симпсона.

**Цель работы**:

Изучить численные методы и алгоритмы численного интегрирования

**Постановка задачи:**

Реализовать изученные алгоритмы численного интегрирования и провести сравнение методов.

**Вариант №7:**

Вычислить интеграл по формулам центральных (средних) прямоугольников, трапеций и формуле Симпсона, при n=8 и n=20; оценить погрешность результата.

***Теория:***

Пусть на отрезке *a, b* задана функция элементарные отрезки (рис. 1).

*y* **** *f(x)* . Разобьем отрезок на

*x0* **** *a; xn* **** *b*

**y**

**a x1 xi-1 i xi b x**

**Рис. 1. Разбиение отрезка**

На каждом из этих отрезков выберем произвольную точку:

*xi******1* **** *εi* **** *xi* . Найдем

произведение *Si* значения функции в точке *εi*

на длину элементарного отрезка:

*Si* ****

*f(εi )(xi* **** *xi* *****1 )*

(1)

Составим сумму всех таких произведений:

*n*

*Sn* **** *S1* **** *S2* **** *...***** *Sn* ** ** *f(εi ) Δxi*

*i******1*

(2)

Sn – называется интегральной суммой.

Определенным интегралом от функции f(x) на [a;b] называется предел интегральной суммы при неограниченном увеличении числа точек разбиения, или при

*Δxi* **** *0*

(максимального из отрезков)

*b*

**** *f(x)dx* ****

*a*

*lim*

*max Δxi* *****0*

**** *f(εi ) Δxi*

*i*

(3)

***Геометрический смысл***

Выражение (1) при *i=*1, 2,…, *n* описывает площадь элементарных прямоугольников *S1, S2,…,Sn* а выражение (2) интегральной суммы – является суммой всей ступенчатой фигуры (рис. 2).

**y**

**Mn**

**f(i)**

**Mi**

**M2**

**M1**

**1**

**S**

**2**

**S**

**i**

**S**

**n**

**S**

**a 1 x1 2 x2**

1. **1i xi**

**xn-1 n b x**

**Рис. 2. Геометрический смысл**

При неограниченном увеличении числа точек деления или

*Δx* **** *0* , верхняя

граница фигуры (ломаная линия) переходит в кривую *y=f(x).* Площадь полученной фигуры (криволинейной трапеции) – определенный интеграл.

Во многих случаях, когда подынтегральная функция задана в аналитическом виде, определенный интеграл удается вычислить непосредственно с помощью определенного (с помощью первообразной) используя формулу Ньютона-Лейбница. Она состоит в том, что определенный интеграл равен приращению первообразной *F(x)* на отрезке интегрирования [*a;b*].

*b b*

**** *f(x)dx* **** *F(x)*

*a a*

**** *F(b)* **** *F(a)*

(4)

Однако, на практике этой формулой часто нельзя воспользоваться по двум основным причинам:

* + вид подынтегральной функции f(x) не допускает непосредственного интегрирования (т.е. первообразную нельзя выразить в элементарной функции);
  + значение f(x) задано на фиксированном множестве точек, т.е. в виде таблицы.

Тогда используются методы численного интегрирования. Они основаны на аппроксимации подынтегральной функции (замене ее некоторым более простым выражением).

В дальнейшем будем использовать кусочную (локальную) интерполяцию. Это позволяет приближенно заменить определенный интеграл (3) интегральной суммой (2). В зависимости от способа интерполяции подынтегральные функции различают разные методы численного интегрирования (методы прямоугольников, трапеций, парабол и др.).

***Метод центральных прямоугольников***

Метод непосредственно использует замену определенного интеграла интегральной суммой. В качестве точек может выбираться любая точка в промежутке . В зависимости от выбора этой точки различают методы левых, правых и центральных прямоугольников.

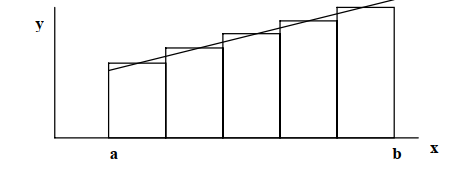
Середина интервала, метод центральных прямоугольников:

Обычно, когда рассматривают метод прямоугольников, разбивают на n равных отрезков:

Заметим, что в пределах одного шага подынтегральная функция заменяется  
(аппроксимация) отрезком горизонтальной прямой т.е. первым членом полинома

Коэффициент ищется из условия прохождения кривой через точки .  
Широко распространенным и более точным является вид формулы  
прямоугольников, использующий значения функции в средних точках элементарных  
отрезков, т.е.

В этом случае формула прямоугольника имеет вид:



***Исходные данные:***

***n1 = 8; n2 = 20***

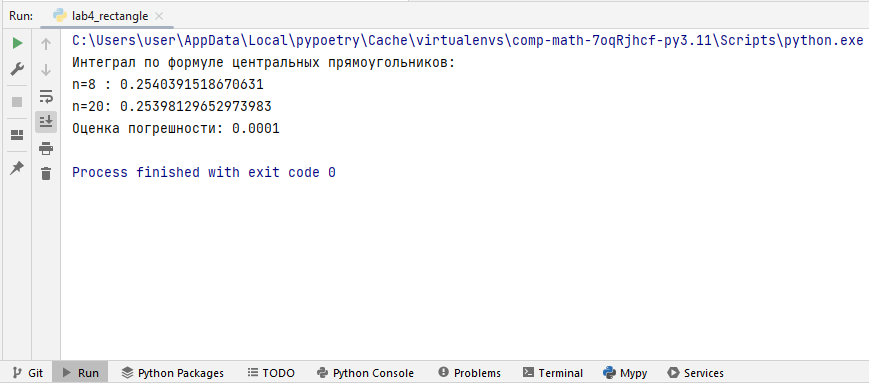
***Решение в Excel:***

|  |  |  |  |
| --- | --- | --- | --- |
| Вычисление интеграла методом центральных прямоугольников | | | |
| Нижний предел | Верхний предел | n | h |
| 0,80 | 1,60 | 8,00 | 0,10 |
| k | x | Xi-1/2 | Fk(x) |
| 1 | 0,90 | 0,85 | 0,027783439 |
| 2 | 1,00 | 0,95 | 0,029402596 |
| 3 | 1,10 | 1,05 | 0,030736762 |
| 4 | 1,20 | 1,15 | 0,031822237 |
| 5 | 1,30 | 1,25 | 0,03269311 |
| 6 | 1,40 | 1,35 | 0,033380293 |
| 7 | 1,50 | 1,45 | 0,033911158 |
| 8 | 1,60 | 1,55 | 0,034309557 |
|  |  | |= | 0,254039152 |

***Программа на Python:***

import math  
  
  
def func(x: float) -> float:  
 return math.log10(x \*\* 2 + 1) / x *# подынтегральная функция*def rectangular\_rule(a: float, b: float, n: int) -> float:  
 *"""  
 Вычисляет определенный интеграл функции с использованием метода центральных прямоугольников.  
  
 Аргументы:  
 a (float): Нижний предел интегрирования.  
 b (float): Верхний предел интегрирования.  
 n (int): Количество интервалов, на которые разбивается область интегрирования.  
  
 Возвращает:  
 float: Приближенное значение определенного интеграла.  
 """* h = (b - a) / n *# шаг* integral = 0 *# аккумулятор значения интеграла* for i in range(n):  
 x = a + h \* (i + 0.5) *# вычисление значения Xi-1/2 с учетом 0-индексированного цикла* integral += h \* func(x) *# аккумуляция значения интеграла* return integral  
  
  
def main() -> None:  
 *# Вычисление интегралов* integral\_rectangular\_8 = rectangular\_rule(a=0.8, b=1.6, n=8)  
 integral\_rectangular\_20 = rectangular\_rule(a=0.8, b=1.6, n=20)  
 error\_rectangular = abs(integral\_rectangular\_20 - integral\_rectangular\_8)  
 print("Интеграл по формуле центральных прямоугольников:")  
 print(f"n=8 : {integral\_rectangular\_8}\nn=20: {integral\_rectangular\_20}")  
 print(f"Оценка погрешности: {round(error\_rectangular, 4)}")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Вывод программы:



***Метод простых итераций (Якоби)***

Решение систем линейных уравнений с помощью метода простой итерации сводится

к следующему алгоритму.

1. Проверка условия сходимости. Для сходимости метода необходимо и достаточно,

чтобы в матрице A абсолютные значения всех диагональных элементов были больше

суммы модулей всех остальных элементов в соответствующей строке:

Недостатком итерационных методов является это достаточно жесткое условие

сходимости, которое выполняется далеко не для всех систем.

2. Если условие сходимости выполнено, то на следующем этапе необходимо задать

начальное приближение вектора неизвестных, в качестве которого обычно выбирается

нулевой вектор:

3. Затем организуется циклический вычислительный процесс, каждый цикл которого

представляет собой одну итерацию. В результате каждой итерации получается новое

значение вектора неизвестных. Для организации итерационного процесса запишем

систему в приведенном виде. При этом слагаемые, стоящие на главной диагонали

нормируются и остаются слева от знака равенства, а остальные переносятся в правую

часть. Приведенная система уравнений имеет вид:

4. Итерационный процесс заканчивается, если для каждой i-й компоненты вектора

неизвестных будет выполнено условие достижения точности:

где k - номер итерации, ε - заданная точность.

***Решение:***

Исходная система:

***Проверка сходимости:***

|3.11| < |-1.66| + |-0.60|+|-0.92| = |3.18| **- нет**

|3.51| < |-1.65| + |-0.78|+|2.57| = |5.00| - **нет**

|-1.87| < |0.60| + |0.78| +|1.65| = |3.03| - **нет**

**Сходимости нет.**

**Хотя данная система не имеет сходимости при любом начальном приближении, можно попробовать достигнуть заданной точности за некоторое количество итераций.**

Выбор начального приближения:

Запись приведенной системы уравнений:

***Решение в Excel:***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Решение методом простой итерации | | | | | | | |
| k | x1 | x2 | x3 | Погрешность | | | Выполнено условие точности (ε=0.001) |
| 0 | 0 | 0 | 0 | x1 | x2 | x3 |  |
| 1 | -0,2958 | 0,7322 | -0,8824 | 0,2958 | 0,7322 | 0,8824 | НЕТ |
| 2 | -0,0752 | 0,3971 | -0,6719 | 0,2206 | 0,3351 | 0,2105 | НЕТ |
| 3 | -0,2135 | 0,5475 | -0,7409 | 0,1383 | 0,1505 | 0,0690 | НЕТ |
| 4 | -0,1465 | 0,4672 | -0,7225 | 0,0670 | 0,0803 | 0,0184 | НЕТ |
| 5 | -0,1858 | 0,5028 | -0,7345 | 0,0393 | 0,0356 | 0,0120 | НЕТ |
| 6 | -0,1692 | 0,4816 | -0,7323 | 0,0167 | 0,0212 | 0,0022 | НЕТ |
| 7 | -0,1800 | 0,4899 | -0,7357 | 0,0109 | 0,0083 | 0,0035 | НЕТ |
| 8 | -0,1762 | 0,4841 | -0,7358 | 0,0038 | 0,0059 | 0,0000 | НЕТ |
| 9 | -0,1794 | 0,4858 | -0,7370 | 0,0031 | 0,0018 | 0,0012 | НЕТ |
| 10 | -0,1787 | 0,4841 | -0,7373 | 0,0007 | 0,0018 | 0,0003 | НЕТ |
| **11** | **-0,1797** | **0,4844** | **-0,7378** | **0,0010** | **0,0003** | **0,0005** | **ДА** |
| 12 | -0,1796 | 0,4838 | -0,7380 | 0,0000 | 0,0006 | 0,0002 | ДА |
| 13 | -0,1800 | 0,4838 | -0,7382 | 0,0003 | 0,0000 | 0,0002 | ДА |
| 14 | -0,1800 | 0,4836 | -0,7383 | 0,0001 | 0,0002 | 0,0001 | ДА |
| 15 | -0,1802 | 0,4835 | -0,7384 | 0,0001 | 0,0001 | 0,0001 | ДА |
| 16 | -0,1802 | 0,4834 | -0,7385 | 0,0000 | 0,0001 | 0,0001 | ДА |
| 17 | -0,1803 | 0,4834 | -0,7385 | 0,0001 | 0,0000 | 0,0001 | ДА |
| 18 | -0,1803 | 0,4833 | -0,7386 | 0,0000 | 0,0000 | 0,0000 | ДА |
| 19 | -0,1803 | 0,4833 | -0,7386 | 0,0000 | 0,0000 | 0,0000 | ДА |
| 20 | -0,1803 | 0,4833 | -0,7386 | 0,0000 | 0,0000 | 0,0000 | ДА |

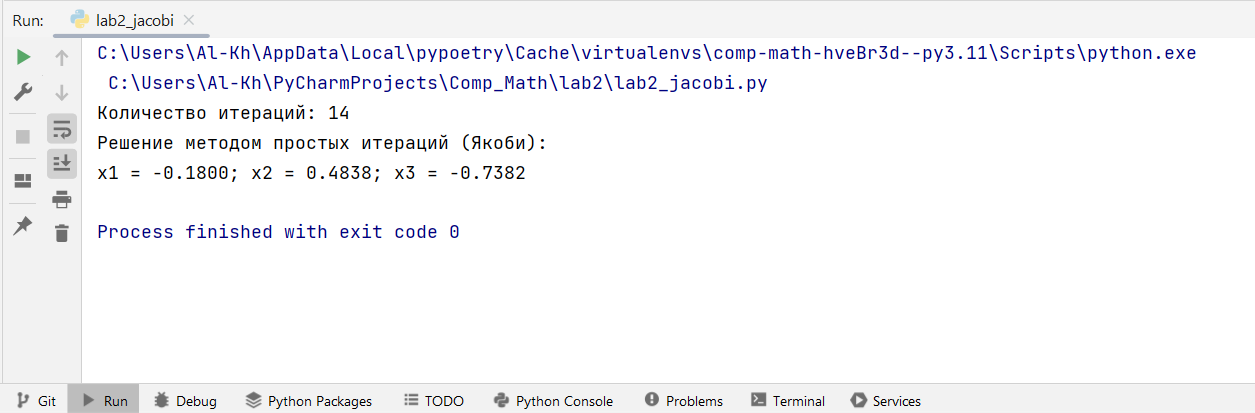
***Программа на Python:***

import numpy as np  
def jacobi(a: np.ndarray, b:np.ndarray, epsilon:float, max\_iterations:int=1000) -> np.ndarray:  
 *"""  
 Рассчитывает решение системы линейных уравнений с использованием метода Якоби.  
  
 Параметры:  
 a (numpy.ndarray): Матрица коэффициентов системы линейных уравнений.  
 b (numpy.ndarray): Вектор правых частей системы линейных уравнений.  
 epsilon (float, опционально): Точность сходимости. По умолчанию 0.0001.  
 max\_iterations (int, опционально): Максимальное количество итераций.*

*По умолчанию 1000.  
  
 Возвращает:  
 x (numpy.ndarray): Вектор решения системы линейных уравнений.  
 """* n = len(a)  
 x = np.zeros(n) *# начальное приближение (нулевой вектор)* it\_counter = 0 *# счетчик итераций*  
 for \_ in range(max\_iterations):  
 x\_new = np.copy(x)  
 it\_counter += 1  
  
 for i in range(n):

*# произведение векторов (срезов, чтобы исключить диагональные элементы)*  
 s1 = np.dot(a[i, :i], x[:i])  
 s2 = np.dot(a[i, i + 1:], x[i + 1:])  
 x\_new[i] = (b[i] - s1 - s2) / a[i, i] *# вычисление значений нового вектора*  
  
 if np.allclose(x, x\_new, rtol=epsilon): *# проверка на соответствие заданной точности*  
 break  
  
 x = x\_new  
 print(f"Количеcтво итераций: {it\_counter}")  
 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 = jacobi(A, B, epsilon=0.001)  
 print(f"Решение методом простых итераций (Якоби):")  
 print("; ".join(f'x{i} = {x:.4f}' for i, x in enumerate(solution, 1)))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Вывод программы:



***Метод Гаусса-Зейделя***

Отличие метода Зейделя от метода простой итерации заключается в том, что при  
вычислении очередного приближения вектора неизвестных используются уже уточненные  
значения на этом же шаге итерации. Это обеспечивает более быструю сходимость метода  
Зейделя. Алгоритм метода Зейделя весьма похож на алгоритм предыдущего метода.  
Первые два пункта (проверка условия сходимости и выбор начального приближения), а  
также четвертый пункт (проверка достижения заданной точности) остаются без  
изменения.  
Отличается здесь только третий пункт алгоритма. При вычислении x1 используется  
информация об остальных неизвестных, найденных на предыдущей итерации. При  
вычислении x2 используется значение x1, найденное на текущей итерации и значения  
остальных переменных, найденные на предыдущей итерации и т.д. Наконец, при  
вычислении последней компоненты вектора неизвестных xn используется информация об  
остальных компонентах, найденных на текущей итерации. Приведенная система  
уравнений имеет вид:

***Решение:***

Исходная система:

***Проверка сходимости:***

|3.11| < |-1.66| + |-0.60|+|-0.92| = |3.18| **- нет**

|3.51| < |-1.65| + |-0.78|+|2.57| = |5.00| - **нет**

|-1.87| < |0.60| + |0.78| +|1.65| = |3.03| - **нет**

**Сходимости нет.**

**Хотя данная система не имеет сходимости при любом начальном приближении, можно попробовать достигнуть заданной точности за некоторое количество итераций.**

Выбор начального приближения:

Запись приведенной системы уравнений:

***Решение в Excel:***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Решение методом Гаусса - Зейделя | | | | | | | |
| k | x1 | x2 | x3 | Погрешность | | | Выполнено условие точности (ε=0.001) |
| 0 | 0 | 0 | 0 | x1 | x2 | x3 |  |
| 1 | -0,2958 | 0,5931 | -0,7299 | 0,2958 | 0,5931 | 0,7299 | НЕТ |
| 2 | -0,1200 | 0,5136 | -0,7067 | 0,1758 | 0,0796 | 0,0232 | НЕТ |
| 3 | -0,1580 | 0,5009 | -0,7241 | 0,0380 | 0,0127 | 0,0175 | НЕТ |
| 4 | -0,1682 | 0,4922 | -0,7310 | 0,0102 | 0,0087 | 0,0069 | НЕТ |
| 5 | -0,1741 | 0,4879 | -0,7347 | 0,0059 | 0,0043 | 0,0037 | НЕТ |
| 6 | -0,1771 | 0,4857 | -0,7366 | 0,0030 | 0,0022 | 0,0019 | НЕТ |
| 7 | -0,1787 | 0,4845 | -0,7376 | 0,0016 | 0,0012 | 0,0010 | НЕТ |
| **8** | **-0,1795** | **0,4839** | **-0,7381** | **0,0008** | **0,0006** | **0,0005** | **ДА** |
| 9 | -0,1799 | 0,4836 | -0,7384 | 0,0004 | 0,0003 | 0,0003 | ДА |
| 10 | -0,1802 | 0,4834 | -0,7385 | 0,0002 | 0,0002 | 0,0001 | ДА |
| 11 | -0,1803 | 0,4833 | -0,7386 | 0,0001 | 0,0001 | 0,0001 | ДА |
| 12 | -0,1803 | 0,4833 | -0,7386 | 0,0001 | 0,0000 | 0,0000 | ДА |
| 13 | -0,1804 | 0,4833 | -0,7386 | 0,0000 | 0,0000 | 0,0000 | ДА |
| 14 | -0,1804 | 0,4833 | -0,7387 | 0,0000 | 0,0000 | 0,0000 | ДА |
| 15 | -0,1804 | 0,4833 | -0,7387 | 0,0000 | 0,0000 | 0,0000 | ДА |
| 16 | -0,1804 | 0,4833 | -0,7387 | 0,0000 | 0,0000 | 0,0000 | ДА |
| 17 | -0,1804 | 0,4833 | -0,7387 | 0,0000 | 0,0000 | 0,0000 | ДА |
| 18 | -0,1804 | 0,4833 | -0,7387 | 0,0000 | 0,0000 | 0,0000 | ДА |
| 19 | -0,1804 | 0,4832 | -0,7387 | 0,0000 | 0,0000 | 0,0000 | ДА |
| 20 | -0,1804 | 0,4832 | -0,7387 | 0,0000 | 0,0000 | 0,0000 | ДА |

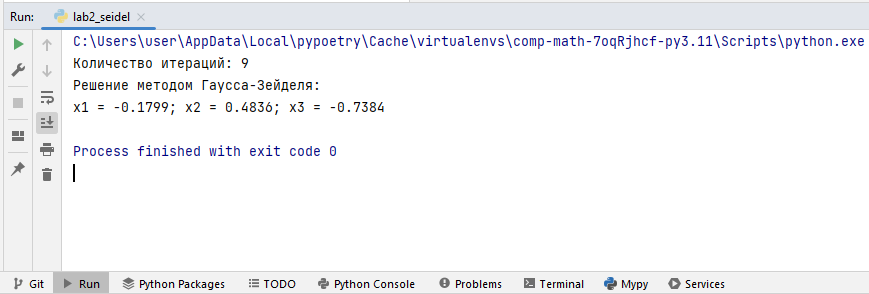
***Программа на Python:***

import numpy as np  
  
  
def gauss\_seidel(a, b, epsilon=0.001, max\_iterations=1000):  
 *"""  
 Рассчитывает решение системы линейных уравнений с использованием метода Гаусса-Зейделя.  
  
 Параметры:  
 a (numpy.ndarray): Матрица коэффициентов системы линейных уравнений.  
 b (numpy.ndarray): Вектор правых частей системы линейных уравнений.  
 epsilon (float, опционально): Точность сходимости. По умолчанию 0.0001.  
 max\_iterations (int, опционально): Максимальное количество итераций.*

*По умолчанию 1000.  
  
 Возвращает:  
 x (numpy.ndarray): Вектор решения системы линейных уравнений.  
 """* n = len(a)  
 x = np.zeros(n) *# начальное приближение (нулевой вектор)* it\_counter = 0 *# счетчик итераций*  
 for \_ in range(max\_iterations):  
 it\_counter += 1  
 for i in range(n):

*# произведение векторов (срезов, чтобы исключить диагональные элементы)*  
 s1 = np.dot(a[i, :i], x[:i]) s2 = np.dot(a[i, i + 1:], x[i + 1:])  
 x[i] = (b[i] - s1 - s2) / a[i, i] *# обновление значений вектора* if np.allclose(np.dot(a, x), b, rtol=epsilon): *# проверка заданной точности* break  
 print(f"Количеcтво итераций: {it\_counter}")  
 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\_seidel(A, B, epsilon=0.001)  
 print(f"Решение методом Гаусса-Зейделя:")  
 print("; ".join(f'x{i} = {x:.4f}' for i, x in enumerate(solution, 1)))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Вывод программы:



**Вывод:**

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

- метод Гаусса

- метод простых итераций (Якоби)

- метод Гаусса-Зейделя

Проблемы итерационных методов (Якоби и Гаусса-Зейделя) в необходимости выполнения достаточно жесткого условия сходимости, а также достаточно большого количества итераций для достижения необходимой точности. Точный результат не может быть достигнут с использованием итерационных методов.

Полученные количества итераций для достижения заданной точности (ε=0.001):

метод Якоби: 14

метод Гаусса – Зейделя: 9

На основе полученных данных можно сделать заключение о том, что метод Гаусса-Зейделя выполняется быстрее и использует меньшее количество итераций, за счет использования сразу вычисленных на текущей итерации значений вектора.