

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ     
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**ДАЛЬНЕВОСТОЧНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**

**Институт математики и компьютерных технологий**

**Департамент математического и компьютерного моделирования**

**ОТЧЕТ**

**по лабораторной работе** **№ 5**

«Приближенные методы решения СЛАУ. Метод Якоби, метод Зейделя.»

Выполнил: студент гр. Б9120-02.03.01сцт

Пограничный К. О.

Проверил: преподаватель

Кузнецов К. С.

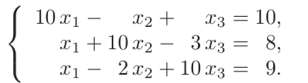
Владивосток 2023

**Постановка задачи**

Дана система линейных алгебраических уравнений в виде Ax=b. Требуется решить ее приближенными методами:

1. Метод Якоби (метод простой итерации)
2. Метод Зейделя в поэлементном виде

Приближенными методами решить системы с точностью



Сравнить число итераций, потребовавшихся для решения.

**Метод решения**

Задача была выполнена на языке программирования Python с использованием средств библиотеки NumPy.

Решения СЛАУ приближёнными методами предполагает постепенное изменение некоторого начального приближения по заданным формулам. Условием остановки итерационного процесса служит достижение необходимой точности. В данной лабораторной работе точность , то есть норма разности значений на соседних итерационных слоях не должна превышать . Полученное приближение считается решением заданной СЛАУ.

**Алгоритм решения**

Решим матрицу методом простой итерации - или по-другому - методом Якоби. Для реализации данного способа необходимо приближаться к необходимой точности по следующей формуле:

Где:

решение на новом итерационном шаге,

решение со старого итерационного шага,

матрица с компонентами:

вектор с компонентами

Также по условию нам нужно решить систему вида Ax=b методом Зейделя. Для данного способа необходимо приближаться к необходимой точности по следующей формуле, заданной в поэлементном виде:

…

Общий вид формулы Зейделя:

В ходе лабораторной работы будут применены именно эти два способа для нахождения решения СЛАУ.

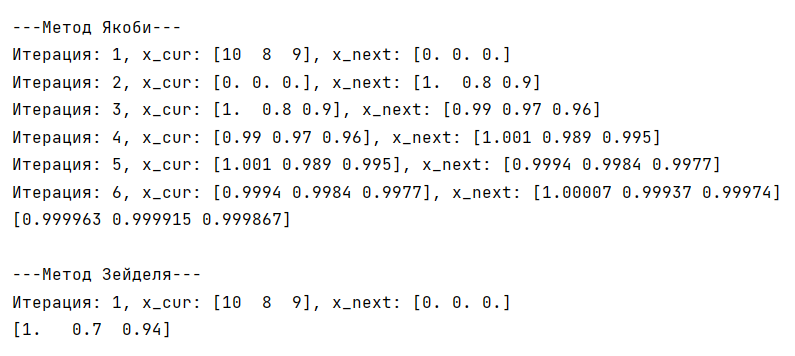
**Подробный анализ кода**

1. Создание матрицы A и вектора b по условию задачи.
2. Реализация класса “Jacobi”, в которой были созданы переменные Альфа и Бета, для уравнения
3. Далее была написан класс “Zeidel”, с помощью которого мы можем поэлементно приближаться к необходимой точности, с помощью формулы, описанной выше.

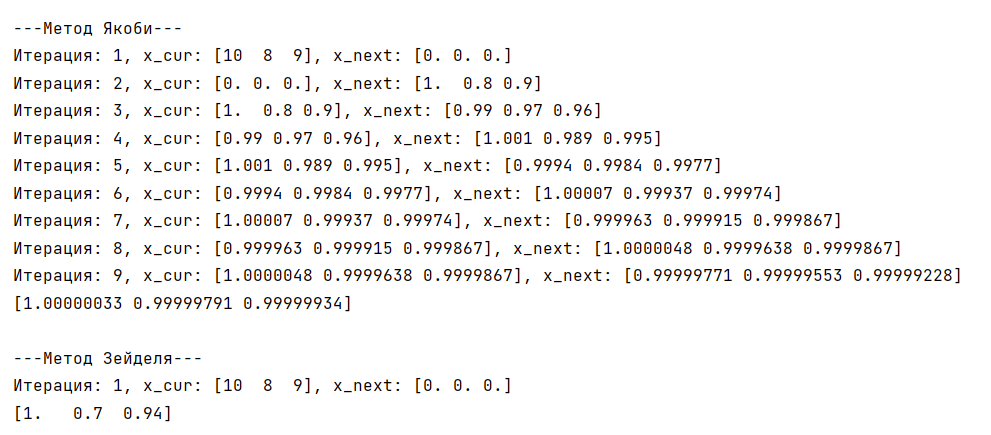
Для системы при помощи метода простой итерации и метода Зейделя находятся вектора x, которые сравниваются с решением, вычисленным непосредственно, для того чтобы убедиться в корректности алгоритма. Также в каждом из методов подсчитывается конечное количество шагов для возможности дальнейшего анализа.

**Результаты тестов**

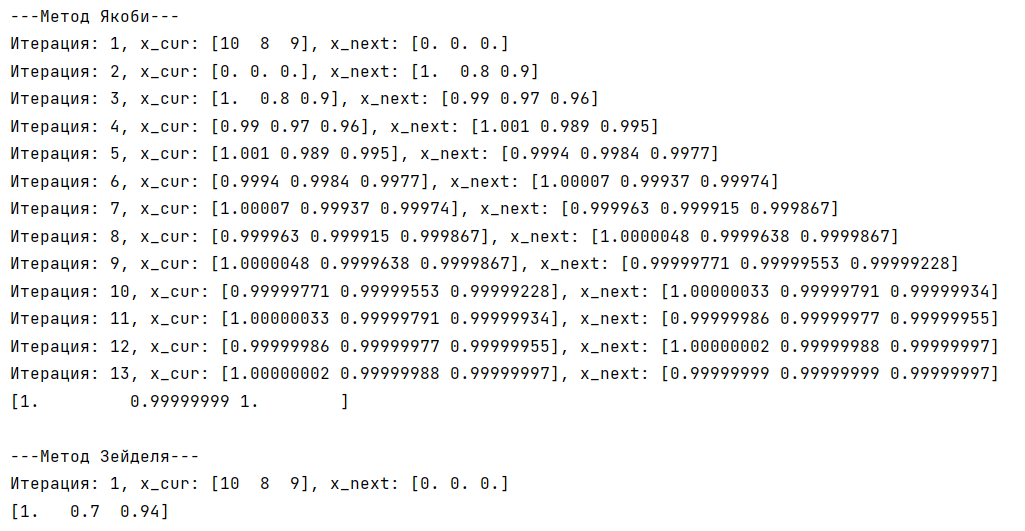
1. При



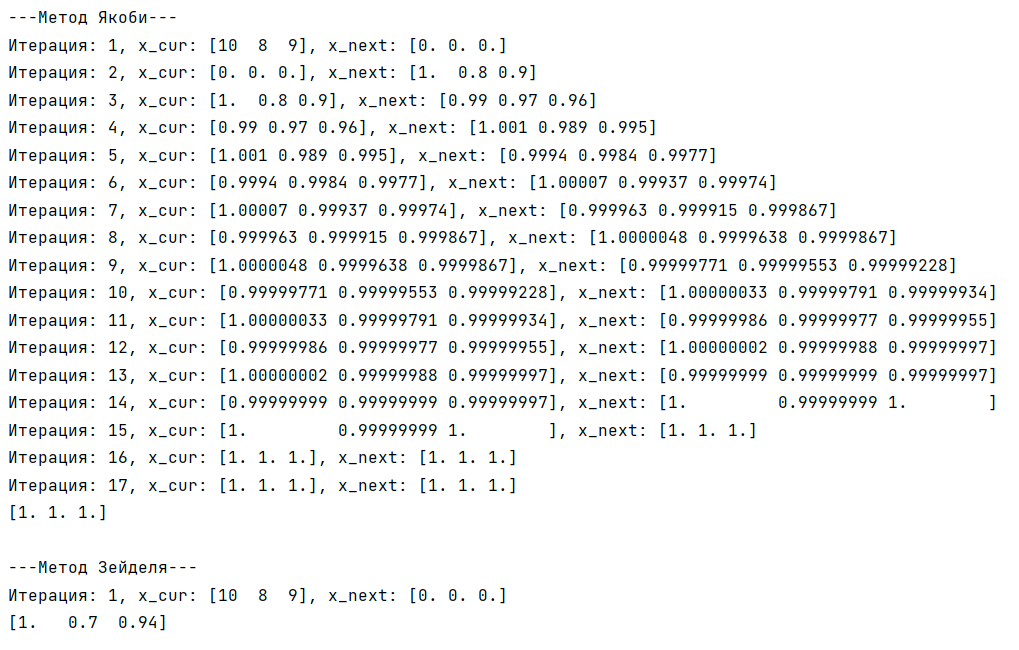
1. При



1. При



1. При



**Вывод**

Методы Якоби и Зейделя позволяют приблизиться к точному решению систем линейных алгебраических уравнений с необходимой точностью, не требуя при этом сложных вычислений.

Метод Зейделя несколько сложнее по своей формуле, но благодаря этому позволяет достичь необходимой точности быстрее, чем метод простой итерации, буквально за одну итерацию.

Таким образом, приближённые методы позволяют упростить поиск решения СЛАУ, но требуют большего числа операций.

**Приложение**

Листинг программы:

import numpy as np  
  
  
def define\_alpha\_and\_beta(dim, a, b):  
 alpha = np.zeros(dim)  
 beta = np.zeros(dim[0])  
  
 for i in range(dim[0]):  
 for j in range(dim[0]):  
 if i != j:  
 alpha[i][j] = - a[i][j] / a[i][i]  
 if i == j:  
 beta[i] = b[i] / a[i][i]  
 return alpha, beta  
  
  
def error(x):  
 vect\_sum = 0  
 for elem in x:  
 vect\_sum += np.power(elem, 2)  
  
 return np.sqrt(vect\_sum)  
  
  
class Jacobi:  
 def \_\_init\_\_(self, a: np.array, b: np.array, eps: float):  
 self.a = a  
 self.b = b  
 self.dim = a.shape  
 self.eps = eps  
 self.alpha, self.beta = define\_alpha\_and\_beta(self.dim, self.a, self.b)  
  
 def get\_solution(self):  
 x\_current = self.b  
 x\_next = self.alpha.dot(x\_current) + self.beta  
 x\_next = np.zeros(self.dim[0])  
 iterator = 1  
  
 while error(x\_next - x\_current) >= self.eps:  
 print(f"Итерация: {iterator}, x\_cur: {x\_current}, x\_next: {x\_next}")  
 x\_current = x\_next  
 x\_next = self.alpha.dot(x\_current) + self.beta  
 iterator += 1  
  
 return x\_next, iterator  
  
  
class Seidel:  
 def \_\_init\_\_(self, a: np.array, b: np.array, eps: float):  
 self.a = a  
 self.b = b  
 self.dim = a.shape  
 self.eps = eps  
 self.alpha, self.beta = define\_alpha\_and\_beta(self.dim, self.a, self.b)  
  
 def get\_solution(self):  
 x\_current = self.b  
 x\_next = np.zeros(self.dim[0])  
  
 iterator = 1  
  
 while error(x\_next - x\_current) >= self.eps:  
 sum\_dim = 0  
 sum\_iter = 0  
 print(f"Итерация: {iterator}, x\_cur: {x\_current}, x\_next: {x\_next}")  
 x\_current = x\_next  
 for i in range(self.dim[0]):  
 sum\_dim = 0  
 sum\_iter = 0  
 for j in range(i, self.dim[0]):  
 sum\_dim += self.alpha[i][j] \* x\_current[j]  
 for j in range(i):  
 sum\_iter += self.alpha[i][j] \* x\_next[j]  
 x\_next[i] = (sum\_iter + sum\_dim) + self.beta[i]  
 iterator += 1  
  
 return x\_next, iterator  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 A = np.array([  
 [10, -1, 1],  
 [1, 10, -3],  
 [1, -2, 10]  
 ])  
 b = np.array([10, 8, 9])  
  
 print("\n---Метод Якоби---")  
 print(Jacobi(A, b, 1e-3).get\_solution()[0])  
 print("\n---Метод Зейделя---")  
 print(Seidel(A, b, 1e-3).get\_solution()[0])