\*Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

****

Лабораторная работа №1 по дисциплине «Вычислительная математика»

Вариант 13

Выполнил:

Студент группы P3212

Метель Леонард Валерьевич

Преподаватель:

г. Санкт-Петербург

2025

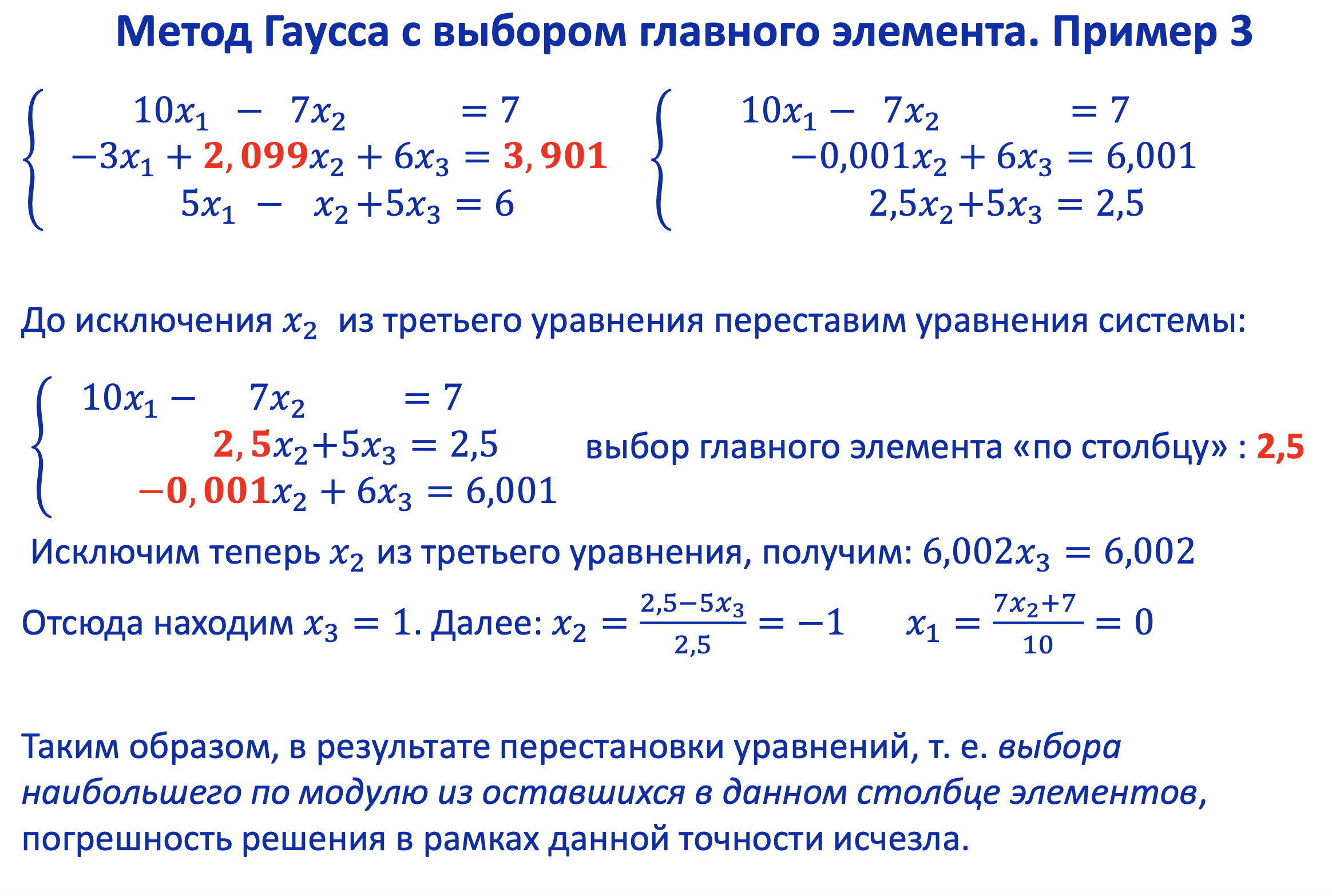
# Цель работы

Реализация метода Гаусса с выбором главного элемента по столбцам для решения СЛАУ на языке программирования Python

# Описание метода











# Листинг программы

import numpy as np  
import os  
  
class GaussMethod:  
 def \_\_init\_\_(self, matrix, vector\_b=None):  
 *"""  
 Инициализация метода Гаусса  
 :param matrix: Матрица коэффициентов системы  
 :param vector\_b: Вектор правых частей (если None, берется последний столбец матрицы)  
 """* if vector\_b is None:  
 # Если вектор b не передан, считаем, что последний столбец матрицы - это вектор b  
 self.A = np.array(matrix[:, :-1], dtype=float)  
 self.b = np.array(matrix[:, -1], dtype=float)  
 else:  
 self.A = np.array(matrix, dtype=float)  
 self.b = np.array(vector\_b, dtype=float)  
   
 self.n = len(self.b)  
 self.x = np.zeros(self.n)  
 self.residuals = np.zeros(self.n)  
 self.determinant = 1.0  
   
 # Сохраняем исходные данные для вычисления невязок  
 self.original\_A = self.A.copy()  
 self.original\_b = self.b.copy()  
   
 # Создаем расширенную матрицу для метода Гаусса  
 self.augmented = np.column\_stack((self.A, self.b))  
   
 def solve(self):  
 *"""  
 Решение системы методом Гаусса с выбором главного элемента  
 """* # Прямой ход метода Гаусса с выбором главного элемента  
 for i in range(self.n):  
 # Поиск максимального элемента в текущем столбце  
 max\_row = i  
 max\_val = abs(self.augmented[i, i])  
   
 for k in range(i + 1, self.n):  
 if abs(self.augmented[k, i]) > max\_val:  
 max\_val = abs(self.augmented[k, i])  
 max\_row = k  
   
 # Обмен строк, если найден больший элемент  
 if max\_row != i:  
 self.augmented[[i, max\_row]] = self.augmented[[max\_row, i]]  
 self.determinant \*= -1 # При обмене строк определитель меняет знак  
   
 # Если главный элемент равен нулю, матрица вырожденная  
 if abs(self.augmented[i, i]) < 1e-10:  
 self.determinant = 0  
 return False  
   
 # Обновление определителя  
 self.determinant \*= self.augmented[i, i]  
   
 # Вычитание из всех строк ниже (без нормализации текущей строки)  
 pivot = self.augmented[i, i]  
 for j in range(i + 1, self.n):  
 factor = self.augmented[j, i] / pivot  
 self.augmented[j] -= factor \* self.augmented[i]  
   
 # Обратный ход метода Гаусса  
 for i in range(self.n - 1, -1, -1):  
 self.x[i] = self.augmented[i, -1]  
 for j in range(i + 1, self.n):  
 self.x[i] -= self.augmented[i, j] \* self.x[j]  
 self.x[i] /= self.augmented[i, i] # Делим на диагональный элемент  
   
 # Вычисление невязок  
 self.calculate\_residuals()  
   
 return True  
   
 def calculate\_residuals(self):  
 *"""  
 Вычисление вектора невязок r = Ax - b  
 """* for i in range(self.n):  
 self.residuals[i] = -self.original\_b[i]  
 for j in range(self.n):  
 self.residuals[i] += self.original\_A[i, j] \* self.x[j]  
   
 def get\_triangular\_matrix(self):  
 *"""  
 Возвращает треугольную матрицу после прямого хода метода Гаусса  
 """* return self.augmented[:, :-1]  
   
 def get\_determinant(self):  
 *"""  
 Возвращает определитель матрицы  
 """* return self.determinant  
   
 def get\_solution(self):  
 *"""  
 Возвращает вектор решения  
 """* return self.x  
   
 def get\_residuals(self):  
 *"""  
 Возвращает вектор невязок  
 """* return self.residuals  
  
  
def read\_matrix\_from\_file(filename):  
 *"""  
 Чтение матрицы из файла  
 """* with open(filename, 'r') as file:  
 lines = file.readlines()  
   
 matrix = []  
 for line in lines:  
 row = [float(x) for x in line.strip().split()]  
 matrix.append(row)  
   
 return np.array(matrix)  
  
  
def input\_matrix\_manually():  
 *"""  
 Ввод матрицы с клавиатуры  
 """* n = int(input("Введите размерность матрицы (n <= 20): "))  
 if n > 20:  
 print("Размерность матрицы должна быть не более 20")  
 return input\_matrix\_manually()  
   
 print(f"Введите {n} строк матрицы (включая столбец свободных членов):")  
 matrix = []  
 for i in range(n):  
 while True:  
 try:  
 row = list(map(float, input(f"Строка {i+1}: ").strip().split()))  
 if len(row) != n + 1:  
 print(f"Строка должна содержать {n+1} чисел")  
 continue  
 matrix.append(row)  
 break  
 except ValueError:  
 print("Ошибка ввода. Введите числа, разделенные пробелами")  
   
 return np.array(matrix)  
  
  
def main():  
 print("Решение СЛАУ методом Гаусса с выбором главного элемента")  
 print("=" \* 60)  
 input\_choice = input("Введите имя файла или нажмите Enter для ручного ввода: ")  
 matrix = None  
 if input\_choice.strip():  
 while True:  
 try:  
 matrix = read\_matrix\_from\_file(input\_choice)  
 break  
 except FileNotFoundError:  
 input\_choice = input("Файл не найден. Попробуйте снова или нажмите Enter для ручного ввода: ")  
 if not input\_choice.strip():  
 break  
 except Exception as e:  
 print(f"Ошибка при чтении файла: {e}")  
 input\_choice = input("Попробуйте снова или нажмите Enter для ручного ввода: ")  
 if not input\_choice.strip():  
 break  
   
 if matrix is None:  
 matrix = input\_matrix\_manually()  
   
 # Решение системы  
 gauss = GaussMethod(matrix)  
 success = gauss.solve()  
   
 if not success:  
 print("Матрица вырожденная, решение не существует")  
 return  
   
 # Вывод результатов  
 print("\nРезультаты:")  
 print("-" \* 60)  
   
 print("\nТреугольная матрица:")  
 triangular = gauss.get\_triangular\_matrix()  
 for row in triangular:  
 print(" ".join(f"{x:10.6f}" for x in row))  
   
 print("\nОпределитель матрицы:")  
 print(f"{gauss.get\_determinant():10.6f}")  
   
 print("\nВектор неизвестных:")  
 solution = gauss.get\_solution()  
 for i, x in enumerate(solution):  
 print(f"x{i+1} = {x:10.6f}")  
   
 print("\nВектор невязок:")  
 residuals = gauss.get\_residuals()  
 for i, r in enumerate(residuals):  
 print(f"r{i+1} = {r:10.6e}")  
   
 # Сравнение с библиотечным решением  
 print("\nСравнение с библиотечным решением:")  
 print("-" \* 60)  
   
 A = matrix[:, :-1]  
 b = matrix[:, -1]  
   
 try:  
 # Решение с помощью numpy  
 numpy\_solution = np.linalg.solve(A, b)  
 numpy\_det = np.linalg.det(A)  
   
 print("\nРешение с помощью numpy:")  
 for i, x in enumerate(numpy\_solution):  
 print(f"x{i+1} = {x:10.6f}")  
   
 print(f"\nОпределитель (numpy): {numpy\_det:10.6f}")  
   
 # Сравнение решений  
 diff = np.linalg.norm(solution - numpy\_solution)  
 print(f"\nРазница между решениями: {diff:10.6e}")  
   
 if diff < 1e-10:  
 print("Решения совпадают с высокой точностью")  
 else:  
 print("Есть различия в решениях")  
   
 except np.linalg.LinAlgError:  
 print("Библиотека numpy не смогла решить систему (вырожденная матрица)")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# Примеры работы программы

Ввод с консоли:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

### Работа программы при вводе из файла input.txt матрицы без диагонального преобладания:

Содержимое файла:

Изображение выглядит как часы, снимок экрана, текст, число

Автоматически созданное описание

Работа программы:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

# Вывод

В процессе выполнения данной лабораторной работы я смог реализовать прямой метод Гаусса с выбором главного элемента по столбцам для решения СЛАУ на языке программирования Python.