Санкт-Петербургский Национальный Исследовательский

Университет ИТМО

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная №1

Вариант: метод Гаусса

Выполнили:

Студентки группы P3266

Фурзикова Александра,

Савельева Елизавета

Преподаватель:

Машина Екатерина Алексеевна

г. Санкт-Петербург

2025 г.

Цель работы: Цель данной работы состоит в разработке программы для решения систем линейных уравнений с использованием метода Гаусса. Программа должна обеспечивать возможность ввода системы уравнений как вручную, так и через файл, а также выводить результаты в понятном виде, включая решения, определитель матрицы и невязки.

Описание метода, расчетные формулы:

Описание метода

В данной программе реализован метод Гаусса, который является классическим численным методом для решения систем линейных уравнений. Этот метод основывается на последовательном преобразовании системы уравнений с использованием элементарных преобразований строк, что позволяет получить верхнетреугольный вид матрицы. После этого решения для переменных вычисляются с помощью обратного подставления.

Ключевые этапы метода Гаусса:

1. Преобразование в верхнетреугольный вид:

   Метод включает в себя прямой ход, в ходе которого производится обнуление элементов ниже главной диагонали матрицы. Это достигается путем вычитания одной строки, умноженной на соответствующий коэффициент, из других строк.

2. Обратное подставление:

   После получения верхнетреугольной матрицы, осуществляется обратное подставление, при котором значения переменных вычисляются начиная с последней строки и подставляются в предыдущие уравнения для нахождения оставшихся неизвестных.

3. Определение определителя:

   Во время работы алгоритма также вычисляется определитель матрицы. Если определитель равен нулю, это указывает на то, что система не имеет единственного решения.

В дополнение к классическому методу Гаусса, в программе используется библиотека NumPy, которая обеспечивает эффективные численные вычисления с матрицами и векторами. Использование NumPy позволяет:

- Упрощать реализацию математических операций и ускорять вычисления.
- Применять встроенные функции для решения систем линейных уравнений и вычисления определителей, что значительно повышает производительность программы.

Сравним оба метода:

Сходство: Оба метода (реализованный метод Гаусса и метод, использующий numpy) должны давать одинаковые результаты для корректно определённых систем уравнений. Это связано с тем, что оба метода основаны на одних и тех же математических принципах.

Различие: Различия могут возникать из-за:

1. Точности вычислений: Метод Гаусса, реализованный вручную, может быть менее точным из-за накопления ошибок округления, особенно при большом количестве уравнений. Библиотека numpy использует оптимизированные алгоритмы, которые минимизируют такие ошибки.
2. Перестановок строк: В ручной реализации могут быть выполнены перестановки строк для избежания деления на ноль, что может повлиять на конечный результат.
   Обработки вырожденных систем: В ручной реализации может быть менее строгая проверка на вырожденность системы, что может привести к некорректным результатам.

Таким образом, сравнение результатов позволяет убедиться в корректности реализации метода Гаусса и оценить точность вычислений.

Формулы:

1. Преобразование системы уравнений

$$a_{ij} \to a_{ij} + k \cdot a_{ik}, \quad \text{где } k = -\frac{a_{mj}}{a_{ij}}$$

2. Обратное подстановление

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}}$$

3. Вычисление определителя

$$\det(A) = \prod_{i=1}^{n} a_{ii}$$

Листинг программы:

```
import numpy as np


# Функция для округления чисел
def round_number(num, digits=3):
    return f'{num:.{digits}f}'


# Функция для чтения системы уравнений из файла
def read_from_file(file_path):
    try:
        n = 0
        equations = []
        with open(file_path, 'r', encoding='utf-8') as file:
            for line in file:
                if line.strip():  # Пропускаем пустые строки
                    n += 1
        if n > 20:
            print('Файл содержит более 20 уравнений. Пожалуйста, уменьшите
количество уравнений и попробуйте снова.')
            return
        with open(file_path, 'r', encoding='utf-8') as file:
            for row in file:
                if row.strip():  # Пропускаем пустые строки
                    parts = row.split()
                    if parts[-2] != '|' or len(parts) - 2 != n:
                        print('Ошибка формата файла. Пожалуйста, проверьте
формат и попробуйте снова.')
                        return
                    equations.append(parts)
        solver = EquationSolver(n, prepare_equations(equations, n))
        solver.solve()
    except FileNotFoundError:
        print('Файл не найден по указанному пути:', file_path)


# Функция для ввода системы уравнений вручную
def input_manually():
    try:
        n = int(input('Введите количество уравнений (не более 20): '))
        if 1 < n <= 20:
            equations = []
            print('Введите коэффициенты уравнений в формате:')
            print('\t', 'a1 a2 ... an | b')
            for i in range(n):
                while True:
                    line = input(f'{i + 1}: ').split()
                    if len(line) - 2 != n or line[-2] != '|':
                        print('Количество коэффициентов не соответствует
количеству уравнений или неправильный формат.')
                        print('Пожалуйста, попробуйте снова.')
                    else:
```

```
                        equations.append(line)
                        break
            solver = EquationSolver(n, prepare_equations(equations, n))
            solver.solve()
        else:
            print('Некорректный ввод! Количество уравнений должно быть от 2
до 20.')
    except ValueError:
        print('Некорректный ввод! Пожалуйста, введите целое число.')


# Преобразуем строки в числа
def prepare_equations(arr, n):
    for i in range(n):
        for j in range(n):
            arr[i][j] = float(arr[i][j])
        arr[i][-1] = float(arr[i][-1])
    return arr


# Решение системы уравнений с использованием numpy
def solve_with_numpy(equations, n):
    # Преобразуем уравнения в матрицу коэффициентов и вектор свободных членов
    A = np.array([equations[i][:n] for i in range(n)], dtype=float)
    B = np.array([equations[i][-1] for i in range(n)], dtype=float)

    # Решаем систему уравнений
    solutions = np.linalg.solve(A, B)

    # Вычисляем определитель матрицы
    determinant = np.linalg.det(A)

    return solutions, determinant


class EquationSolver:
    def __init__(self, n, equations):
        self.n = n
        self.original_equations = [row.copy() for row in equations]  #
Сохраняем исходную матрицу
        self.equations = equations  # Матрица для преобразований
        self.solutions = []
        self.swaps = 0

    def solve(self):
        try:
            print('\nИсходная система уравнений:')
            self.print_equations()

            self.convert_to_triangle()
            print('\nТреугольная матрица:')
            self.print_equations()

            print('\nКоличество перестановок:', self.swaps)

            self.calculate_determinant()

            self.find_solutions()
            self.print_solutions()

            self.print_residuals()

            # Передаем исходную матрицу в numpy, а не модифицированную
            numpy_solutions, numpy_determinant =
```

```
solve_with_numpy(self.original_equations, self.n)
            print('\nРешения системы через numpy:')
            for i in range(self.n):
                print(f'\tx[{i}] = {round_number(numpy_solutions[i])}')
            print(f'\nОпределитель матрицы через numpy:
{round_number(numpy_determinant)}')

            # Сравнение результатов
            print('\nСравнение результатов:')
            print(f'\tРазница в решениях:
{np.linalg.norm(np.array(self.solutions) - numpy_solutions)}')
            print(f'\tРазница в определителях: {abs(self.determinant -
numpy_determinant)}')

        except ZeroDivisionError:
            print('Ошибка: деление на ноль!')
        except ArithmeticError:
            print('Ошибка: система не имеет решений!')
        except np.linalg.LinAlgError as e:
            print(f'Ошибка при решении системы через numpy: {e}')

    def check_diagonal(self, i):
        for j in range(i, self.n):
            if self.equations[j][i] != 0:
                self.equations[i], self.equations[j] = self.equations[j],
self.equations[i]
                self.swaps += 1
                return
        print('Система не имеет решений!')
        raise ArithmeticError

    def convert_to_triangle(self):
        for i in range(self.n):
            if self.equations[i][i] == 0:
                self.check_diagonal(i)
            for m in range(i + 1, self.n):
                factor = -(self.equations[m][i] / self.equations[i][i])
                for j in range(i, self.n):
                    self.equations[m][j] += factor * self.equations[i][j]
                self.equations[m][-1] += factor * self.equations[i][-1]

    def print_equations(self):
        for i in range(self.n):
            equation = ' '.join([f'{round_number(self.equations[i][j])} *
x[{j}]' for j in range(self.n)])
            print(f'{equation} | {round_number(self.equations[i][-1])}')

    def calculate_determinant(self):
        self.determinant = 1
        for i in range(self.n):
            self.determinant *= self.equations[i][i]
        if self.swaps % 2 == 1:
            self.determinant *= -1
        print(f'\nОпределитель матрицы: {round_number(self.determinant)}\n')
        if self.determinant == 0:
            print('Система вырождена и не имеет решений.')
            raise ArithmeticError

    def find_solutions(self):
        self.solutions = [0] * self.n
        for i in range(self.n - 1, -1, -1):
            self.solutions[i] = self.equations[i][-1] / self.equations[i][i]
            for j in range(i - 1, -1, -1):
                self.equations[j][-1] -= self.equations[j][i] *
```

```
self.solutions[i]

    def print_solutions(self):
        print('Решения системы:')
        for i in range(self.n):
            print(f'\tx[{i}] = {round_number(self.solutions[i])}')

    def print_residuals(self):
        print('\nВеличины невязок:')
        for i in range(self.n):
            residual = sum(self.equations[i][j] * self.solutions[j] for j in
range(self.n)) - self.equations[i][-1]
            print(f'\tНевязка для уравнения {i + 1}:
{round_number(abs(residual))}')


# Основное меню программы
def main():
    print('Решение систем линейных уравнений методом Гаусса')

    while True:
        try:
            print('\nДоступные действия:')
            print('\t1: Загрузить систему уравнений из файла.')
            print('\t2: Ввести систему уравнений вручную.')
            print('\t3: Выйти из программы.')
            choice = int(input('Выберите действие: '))

            if choice == 1:
                print('Загрузка системы уравнений из файла.')
                print('Формат файла должен быть следующим (максимум 20
уравнений):')
                print('\ta11 a12 ... a1n | b1')
                print('\ta21 a22 ... a2n | b2')
                print('\t... ... ... ... | ..')
                print('\tan1 an2 ... ann | bn')
                file_path = input('Введите путь к файлу: ').strip()
                read_from_file(file_path)
            elif choice == 2:
                print('Ручной ввод системы уравнений.')
                input_manually()
            elif choice == 3:
                print('Программа завершена. До свидания!')
                break
            else:
                print('Некорректный выбор. Пожалуйста, выберите действие от 1
до 3.')
        except KeyboardInterrupt:
            print('\nПрограмма была прервана пользователем.')
        except Exception as e:
            print(f'Произошла ошибка: {e}')


if __name__ == '__main__':
    main()
```

Примеры и результаты работы программы:

Если вводим в ручную:

Решение систем линейных уравнений методом Гаусса

Доступные действия:

1: Загрузить систему уравнений из файла.

2: Ввести систему уравнений вручную.

3: Выйти из программы.

Выберите действие: 2

Ручной ввод системы уравнений.

Введите количество уравнений (не более 20): 2

Введите коэффициенты уравнений в формате:

a1 a2 ... an | b

1: 1 5 | 7

2: 3 -2 | 4

Исходная система уравнений:

1.000 * x[0] 5.000 * x[1] | 7.000

3.000 * x[0] -2.000 * x[1] | 4.000

Треугольная матрица:

1.000 * x[0] 5.000 * x[1] | 7.000

0.000 * x[0] -17.000 * x[1] | -17.000

Количество перестановок: 0

Определитель матрицы: -17.000

Решения системы:

x[0] = 2.000

x[1] = 1.000

Величины невязок:

Невязка для уравнения 1: 5.000

Невязка для уравнения 2: 0.000

Решения системы через numpy:

x[0] = 2.000

x[1] = 1.000

Определитель матрицы через numpy: -17.000

Сравнение результатов:

Разница в решениях: 0.0

Разница в определителях: 0.0

Если считываем данные из файла:

Доступные действия:

1: Загрузить систему уравнений из файла.

2: Ввести систему уравнений вручную.

3: Выйти из программы.

Выберите действие: 1

Загрузка системы уравнений из файла.

Формат файла должен быть следующим (максимум 20 уравнений):

a11 a12 ... a1n | b1

a21 a22 ... a2n | b2

... ... | ..

an1 an2 ... ann | bn

Введите путь к файлу: C:\Users\Redmi\PycharmProjects\lab\2.txt

Исходная система уравнений:

1.000 * x[0] 5.000 * x[1] | 7.000

3.000 * x[0] -2.000 * x[1] | 4.000

Треугольная матрица:

1.000 * x[0] 5.000 * x[1] | 7.000

0.000 * x[0] -17.000 * x[1] | -17.000

Количество перестановок: 0

Определитель матрицы: -17.000

Решения системы:

x[0] = 2.000

x[1] = 1.000

Величины невязок:

Невязка для уравнения 1: 5.000

Невязка для уравнения 2: 0.000

Решения системы через numpy:

x[0] = 2.000

x[1] = 1.000

Определитель матрицы через numpy: -17.000

Сравнение результатов:

Разница в решениях: 0.0

Разница в определителях: 0.0

Выводы

В результате работы программы была успешно реализована система для решения линейных уравнений методом Гаусса. Программа позволяет как вводить данные вручную, так и загружать их из файла. Метод Гаусса был реализован с возможностью проверки условий, таких как наличие решений и корректность формата входных данных. Тестирование программы показало, что она корректно обрабатывает различные системы уравнений, и выводит результаты, включая решения и информацию о невязках.