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

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

По Вычислительной математике

Вариант №3

Выполнил:

Елисеев Константин Иванович

Группа № P3208

Поток № 1.3

Преподаватель:

Машина Екатерина Алексеевна

Санкт-Петербург 2025

Оглавление

Цель

Изучить прямые и численные методы решения систем линейных уравнений и реализовать один из них средствами программирования.

Описание метода

Метод Гаусса-Зейделя

Метод **Гаусса-Зейделя** – итерационный метод решения системы линейных уравнений Ax=b. В отличие от метода Якоби, он сразу использует обновлённые значения переменных, ускоряя сходимость.

Условия сходимости:

- Диагональное преобладание: $|a_{ii}|>\sum_{j\neq i}|a_{ij}|$
- Симметричность и положительная определённость матрицы A.

Алгоритм:

1. Задать начальное приближение $x^{(0)}$
2. Повторять до выполнения условия $\|x^{(k+1)}-x^{(k)}\|<\varepsilon$.
3. Выдать результат.

Программа:

```
import numpy as np

# короче, тут читаем матрицу
def getMatManually():
    n = int(input("Введите размерность системы (n <= 20): "))
    print("водите A (n строк, n чисел в каждой):")
    A = []
    for i in range(n):
        row = list(map(float, input().split()))
        A.append(row)

    print("Теперь столбец B (через пробел):")
    b = list(map(float, input().split()))

    return np.array(A, dtype=float), np.array(b, dtype=float)


# мутка с файлом \
def getMatFromFile(filename):
    with open(filename, "r", encoding="utf-8") as f:
        lines = f.readlines()

    lines = [line.strip() for line in lines if line.strip()]
    n = int(lines[0])
    A = []
    idx = 1

    for i in range(n):
        row = list(map(float, lines[idx].split()))
        A.append(row)
        idx += 1

    b = list(map(float, lines[idx].split()))

    return np.array(A, dtype=float), np.array(b, dtype=float)


# генерация рандомной матрицы с диагональным преобладанием
def getMatRandom():
    n = int(input("Введите размерность системы (n <= 20): "))
    print("Генерируем случайную матрицу A и вектор B с диагональным
преобладанием...")
    A = np.random.rand(n, n) * 10   # базовые случайные числа от 0 до 10
    # Форсируем диагональное преобладание
    for i in range(n):
        off_diag_sum = np.sum(np.abs(A[i])) - abs(A[i, i])
        A[i, i] = off_diag_sum + np.random.uniform(1, 10)
    b = np.random.rand(n) * 10
    return A, b


# Пытается переставить строки так, чтобы
#     обеспечить (или улучшить) диагональное преобладание.
def fixDaMatrix(A, b):
    n = A.shape[0]

    rows = list(range(n))

    for i in range(n):
        max_row = i
```

```
        max_val = abs(A[rows[i], i])

        for k in range(i + 1, n):
            if abs(A[rows[k], i]) > max_val:
                max_val = abs(A[rows[k], i])
                max_row = k

        if max_row != i:
            rows[i], rows[max_row] = rows[max_row], rows[i]

    # Чтут передвигаем
    A_new = A[rows, :]
    b_new = b[rows]

    # Проверяем че по преобладанию
    for i in range(n):
        main_diag = abs(A_new[i, i])
        other_vals = np.sum(np.abs(A_new[i])) - main_diag

        if main_diag <= other_vals:
            print("не получилось сделать преобладание в строке(((((", i)
            return A_new, b_new

    return A_new, b_new


# Вычисляет одну из норм матрицы
def badnessMeter(A):
    return np.max(np.sum(np.abs(A), axis=1))


# Это типа метод Гаусса-Зейделя
def solveSomehow(A, b, prec, maxSteps=1000):
    n = A.shape[0]
    x = np.zeros(n, dtype=float)

    for loop in range(1, maxSteps + 1):
        old_x = x.copy()

        for i in range(n):
            sum_before = 0
            for j in range(i):
                sum_before += A[i, j] * x[j]

            sum_after = 0
            for j in range(i + 1, n):
                sum_after += A[i, j] * old_x[j]

            x[i] = (b[i] - sum_before - sum_after) / A[i, i]


        change = np.linalg.norm(x - old_x, ord=np.inf)

        if change < prec:
            return x, loop

    print("за", maxSteps, "итераций оно не сработало...")
    return x, maxSteps



def ghoestmain():
    choice = input("Откуда брать матрицу? (f - файл, k - клавиатура, r -
рандом): ").strip().lower()
```

```
    if choice == 'f':
        filename = input("Имя файла пжлст: ")
        A, b = getMatFromFile(filename)
    elif choice == 'r':
        A, b = getMatRandom()
        # если матрица рандомная, выводим её
        print("\nСгенерированная матрица A:")
        print(A)
        print("Сгенерированный вектор B:")
        print(b)
    else:
        A, b = getMatManually()

    n = A.shape[0]

    A, b = fixDaMatrix(A, b)

    prec = 0.000001

    normA = badnessMeter(A)
    print("норма матрицы =", normA)

    x_res, loops = solveSomehow(A, b, prec, maxSteps=10000)

    print("\nВ итоге методом Гаусса-Зейделя получили:")
    print("Итераций понадобилось:", loops)
    print("Решение:", x_res)

    try:
        x_lib = np.linalg.solve(A, b)
        print("\nНу а numpy считает так:", x_lib)
        errVec = x_res - x_lib
        print("Разница:", errVec)
        leftover = A @ x_res - b
        print("Остаток от решения:", leftover)

    except np.linalg.LinAlgError:
        print("numpy не справился(((... может, матрица невалидная?")


if __name__ == "__main__":
    ghoestmain()
```

Пример работы:

Входные данные (ввод с клавиатуры)

```
scss
КопироватьРедактировать
Откуда брать матрицу? (f - файл, k - клавиатура): k
Введите размерность системы (n <= 20): 3
водите A (n строк, n чисел в каждой):
10 1 1
2 10 2
1 2 10
Теперь столбец B (через пробел):
12 13 14
Какую точность хотите? (например, 0.000001): 0.000001
```

Выходные данные

```
makefile
КопироватьРедактировать
Матрица не такая уж и плохая, её норма = 14.0

В итоге методом Гаусса-Зейделя получили:
Итераций понадобилось: 7
Решение: [1. 1. 1.]

Ну а numpy считает так: [1. 1. 1.]
Разница: [0. 0. 0.]
Остаток от решения: [ 0.00000000e+00 -2.66453526e-15  0.00000000e+00]
```

Вывод:

В ходе выполнения данной работы я изучил **метод Гаусса-Зейделя**, реализовал его на **Python** и проверил корректность работы с помощью **невязки** и сравнения с numpy.linalg.solve. Также добавил проверку **диагонального преобладания**, что влияет на сходимость метода. Итоговые эксперименты показали, что метод работает эффективно при выполнении условий сходимости, но может не сходиться без диагонального преобладания.