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

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №1

Вариант 3

Группа: P3269

Выполнили:

Грибкова В.Е

Долганова О.А

Проверил:

Машина Е. А.

Г. Санкт-Петербург

Цель работы

Используя известные методы вычислительной математики, написать программу, осуществляющий решение СЛАУ методом Гаусса. Вычислить определитель, треугольную матрицу, вектор неизвестных и вектор невязок.

Описание метода, расчётные формулы

Прямые методы. Метод Гаусса

Рассмотрим наиболее распространенную ***схему единственного деления***.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2, \\ &\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n. \end{aligned} \qquad (1)$$

Прямой ход:

Шаг 1 (считаем $a_{11} \neq 0$):

Исключим x_1 из второго уравнения: умножим первое уравнение на $(-{}^{a_{21}}/_{a_{11}})$ и прибавим ко второму.

Исключим x_1 из третьего уравнения: умножим первое уравнение на $(-{}^{a_{31}}/_{a_{11}})$ и прибавим к третьему...

Исключим x_1 из последнего уравнения: умножим первое уравнение на $(-{}^{a_{n1}}/_{a_{11}})$ и прибавим к последнему. Получим равносильную систему уравнений (2) :

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1, \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)}, \\ a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 + \ldots + a_{3n}^{(1)}x_n &= b_3^{(1)} \\ a_{n2}^{(1)}x_2 + a_{n3}^{(1)}x_3 + \ldots + a_{nn}^{(1)}x_n &= b_n^{(1)} \end{aligned} \qquad (2)$$

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}\,, i, j = 2,3 \ldots n$$

$$b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}} b_1\,, i = 2,3 \ldots n$$

Шаг 2:

Исключим x_2 из третьего уравнения: умножим второе уравнение на $(-{}^{a'_{32}}/_{a'_{22}})$ и прибавим к третьему (и т.д. для следующих уравнений)

Исключим x_2 из последнего уравнения: умножим второе уравнение на $(-{}^{a'_{n2}}/_{a'_{22}})$ и прибавим к последнему.

Получим:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}\,x_n &= b_1, \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}\,x_n &= b_2^{(1)}, \\ a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}\,x_n &= b_3^{(2)} \\ a_{n3}^{(2)}x_3 + \ldots + a_{nn}^{(2)}\,x_n &= b_n^{(2)} \end{aligned} \tag{3}$$

$$a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2j}^{(1)}, \quad i,j = 3,4 \ldots n \qquad b_i^{(2)} = b_i^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} b_2^{(1)}, \quad i = 3,4 \ldots n$$

Продолжим до тех пор, пока матрица системы (3) не примет треугольный вид (4):

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}\,x_n &= b_1, \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}\,x_n &= b_2^{(1)}, \\ a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}\,x_n &= b_3^{(2)} \\ a_{nn}^{(n-1)}x_n &= b_n^{(n-1)} \end{aligned} \tag{4}$$

Матрица системы (4) имеет треугольный вид $\to$ конец *прямого хода*.

Требование : Если в процессе исключения неизвестных, коэффициенты:

$$\boldsymbol{a_{11}, a_{22}^{1}, a_{33}^{2} \ldots = 0},$$

тогда необходимо соответственным образом переставить уравнения системы.

Перестановка уравнений должна быть предусмотрена в вычислительном алгоритме при его реализации на компьютере.

Определитель

Из курса линейной алгебры известно, что определитель треугольной матрицы равен произведению диагональных элементов.

Определитель **после** приведения матрицы А к треугольному виду вычисляется по формуле:

$$detA = (-1)^k \prod_{i=1}^{n} a_{ii}$$

k – число перестановок строк (или столбцов) матрицы при ее приведении к треугольному виду (для получения ненулевого или максимального по модулю ведущего элемента на каждом этапе исключения).

Знак определителя меняется на противоположный при перестановке его столбцов или строк.

Благодаря методу исключения можно вычислять определители порядка *n*>1000-го и объем вычислений будет значительно меньший, чем в проведенных ранее оценках.

Погрешности решения

Решения, получаемые с помощью прямых методов, обычно содержат погрешности. Они возникают из-за погрешностей округлений при выполнении операций над числами с плавающей точкой , связанных с ограниченностью разрядной сетки машины. В ряде случаев эти погрешности могут быть значительными.

Существуют две величины, характеризующие степень отклонения полученного решения от точного:

абсолютная погрешность $\Delta x = x - x^*$, где x – точное решение, x^* – решение, вычисленное по методу Гаусса.

невязка $r = Ax^* - b$, разность между левой и правой частями уравнений при подстановке в них решения x^* .

В практических расчетах, если система не является плохо обусловленной, контроль точности решения осуществляется с помощью невязки (погрешность же обычно вычислить невозможно, поскольку неизвестно точное решение).

Можно отметить, что метод Гаусса с выбором главного элемента в этих случаях дает малые невязки.

Листинг программы

```
from random import randint

def read_data_from_file():
    # Функция для чтения данных из файла 'input.txt'
    with open('input.txt') as file:
        size = int(file.readline())  # Чтение размерности матрицы
        file.readline()  # Пропуск пустой строки
        # Чтение матрицы A
        matrix_a = [
            list(map(float, file.readline().split()))
            for _ in range(size)
        ]
        file.readline()  # Пропуск пустой строки
        # Чтение вектора B
        vector_b = list(map(float, file.readline().split()))
        return size, matrix_a, vector_b


def generate_random_matrix(size):
    # Функция для генерации случайной квадратной матрицы размера
size и случайного вектора
    min_value = -1000  # Минимальное значение для случайных чисел
    max_value = 1000  # Максимальное значение для случайных чисел

    # Генерация случайной матрицы A
    matrix_a = [
        [randint(min_value, max_value) for _ in range(size)]
        for _ in range(size)
    ]

    # Генерация случайного вектора B
    vector_b = [randint(min_value, max_value) for _ in range(size)]

    return matrix_a, vector_b


def get_user_input():
    # Функция для получения данных от пользователя и генерации
матрицы и вектора

    random_flag = 'RANDOM'  # Ключевое слово для генерации
случайных данных

    # Запрос у пользователя размерности матрицы
    size_input = input('Введите размерность матрицы (n): ').strip()
```

```
    while not size_input.isdigit() or int(size_input) < 1 or
int(size_input) > 20:
        print('-' * 50)
        print('n должно быть числом на отрезке [1;20]')
        size_input = input('Введите размерность матрицы (n):
').strip()

    size = int(size_input)

    # Запрос у пользователя, хочет ли он сгенерировать случайную
матрицу
    random_choice = input(f'Введите "{random_flag}" для генерации
случайной матрицы (A и B): ')
    if random_choice == 'RANDOM':
        # Генерация случайной матрицы и вектора
        matrix_a, vector_b = generate_random_matrix(size)
        print('Сгенерированная матрица: ')
        for i in range(size):
            print(*matrix_a[i], '|', vector_b[i], sep='\t')
        print('-' * 50)
    else:
        # Ввод матрицы A от пользователя
        print('Введите элементы матрицы A:')
        matrix_a = []
        for i in range(size):
            valid_input = False
            while not valid_input:
                try:
                    # Запрос строки матрицы
                    print(f'Введите строку ({i}) (n чисел): ')
                    row = list(map(float, input().split()))
                    assert len(row) == size  # Проверка, что
введено нужное количество элементов
                    valid_input = True
                except Exception:
                    print("Ошибка ввода. Попробуйте еще раз.")
                    pass
            matrix_a.append(row)

        # Ввод вектора B от пользователя
        print('Введите элементы вектора B:')
        valid_input = False
        while not valid_input:
            try:
                # Запрос строки вектора
                print('Введите строку (n чисел): ')
                vector_b = list(map(float, input().split()))
```

```
                assert len(vector_b) == size  # Проверка, что
введено нужное количество элементов
                valid_input = True
            except Exception:
                print("Ошибка ввода. Попробуйте еще раз.")
                pass

    return size, matrix_a, vector_b


def calculate_determinant(triangular_matrix, swaps):
    # Функция для вычисления определителя треугольной матрицы с
учетом перестановок строк
    determinant = 1
    for i in range(len(triangular_matrix)):
        determinant *= triangular_matrix[i][i]  # Произведение
диагональных элементов
    # Учет количества перестановок строк: каждая перестановка
меняет знак определителя
    return (-1) ** swaps * determinant


def gaussian_elimination(matrix_a, vector_b):
    # Функция для решения системы уравнений методом Гаусса
    size = len(matrix_a)
    x = [None] * size  # Вектор решения
    swap_count = 0  # Счетчик перестановок строк

    # Прямой ход метода Гаусса: приведение к треугольному виду
    for i in range(0, size - 1):
        # Проверка и перестановка строк для исключения нулевого
ведущего элемента
        while matrix_a[i][i] == 0:
            matrix_a = matrix_a[:i] + matrix_a[i + 1:] +
[matrix_a[i]]  # Перестановка строк
            vector_b = vector_b[:i] + vector_b[i + 1:] +
[vector_b[i]]  # Перестановка элементов вектора
            swap_count += 1  # Увеличение счетчика перестановок

        # Обнуление элементов ниже главной диагонали
        for k in range(i + 1, size):
            factor = matrix_a[k][i] / matrix_a[i][i]  # Вычисление
коэффициента для обнуления
            matrix_a[k][i] = 0  # Обнуление элемента
            for j in range(i + 1, size):
                matrix_a[k][j] = matrix_a[k][j] - factor *
matrix_a[i][j]  # Обновление строки
```

```
            vector_b[k] = vector_b[k] - factor * vector_b[i]  #
Обновление элемента вектора

    # Обратный ход метода Гаусса: нахождение решений системы
    for i in range(size - 1, -1, -1):
        sum_ax = 0  # Сумма произведений коэффициентов и найденных
решений
        for j in range(i + 1, size):
            sum_ax += matrix_a[i][j] * x[j]  # Вычисление суммы
        x[i] = (vector_b[i] - sum_ax) / matrix_a[i][i]  #
Вычисление неизвестного

    return x, swap_count, matrix_a, vector_b


def calculate_residuals(matrix_a, vector_b, solution):
    # Функция для вычисления вектора невязок
    size = len(matrix_a)
    residuals = [0] * size
    for i in range(size):
        for j in range(size):
            residuals[i] += matrix_a[i][j] * solution[
                j]  # Вычисление скалярного произведения строки
матрицы и вектора решения
        residuals[i] -= vector_b[i]  # Вычитание соответствующего
элемента вектора B
    return residuals


# Загрузка данных из файла
n, a, b = read_data_from_file()

# Сохранение первоначальных значений для вычисления невязок
original_a = [row[:] for row in a]  # Копирование матрицы A
original_b = b[:]  # Копирование вектора B

# Решение системы уравнений методом Гаусса
solution, swap_count, triangular_a, updated_b =
gaussian_elimination(a, b)

# Вычисление определителя треугольной матрицы
determinant = calculate_determinant(triangular_a, swap_count)
print('Определитель: ', determinant)

# Вывод треугольной матрицы и обновленного вектора B
print('Треугольная матрица: ')
for i in range(n):
    print(*triangular_a[i], '|', updated_b[i], sep='\t')
```

```
print('-' * 50)

# Вывод решения системы уравнений
print('Вектор неизвестных: ', *map(lambda xi: round(xi, 5),
solution))

# Вычисление и вывод вектора невязок
residuals = calculate_residuals(original_a, original_b, solution)
print('Вектор невязок: ', *residuals)
```

Примеры и результаты работы программы

Ввод:

```
3

1 3 5
1 3 6
9 8 5

7 4 12
```

Вывод:

```
Определитель:  19.0
Треугольная матрица:
1.0 3.0 5.0 7.0
0 -19.0 -40.0 -51.0
0 0 1.0 -3.0
Вектор неизвестных:  -5.0 9.0 -3.0
Вектор невязок:  0.0 0.0 0.0
```

Вывод

В ходе работы реализован метод Гаусса, позволяющий решать СЛАУ.