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: **9**

Преподаватель:

Выполнил: Пронкин Алексей
Группа: P3215

Санкт-Петербург, 2024 г

Цель работы

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

Описание метода

Итерационные методы дают возможность для системы (1) построить последовательность векторов $x^{(0)}, x^{(1)}, \dots, x^{(k)}$, пределом которой должно быть точное решение $x^{(*)}$: $x^{(*)} = \lim_{k\to\infty} x^{(k)}$ Построение последовательности заканчивается, как только достигается желаемая точность.

Приведем систему уравнений, выразив неизвестные $x_1, x_2, \dots, x_n$ соответственно из первого, второго и т.д. уравнений системы.

Листинг программы

```
import math
import sys
import random
from itertools import permutations


def print_augmented_matrix(A, b):
    """
    Функция для вывода расширенной матрицы [A|b].
    """
    n = len(A)
    print("Расширенная матрица [A|b]:")
    for i in range(n):
        row_str = ""
        for j in range(n):
            row_str += f"{A[i][j]:10.4f} "
        row_str += f"| {b[i]:10.4f}"
        print(row_str)
    print()


def euclidian_norm(v):
    """
    Функция вычисления евклидовой нормы вектора.
    """
    return math.sqrt(sum(val * val for val in v))


def has_diagonal_dominance(A):
    """
    Проверяет, есть ли у матрицы A диагональное преобладание.
    Возвращает True, если преобладание есть, иначе False.
    """
    n = len(A)
    for i in range(n):
        diagonal_element = abs(A[i][i])
        row_sum = sum(abs(A[i][j]) for j in range(n) if j != i)
        if diagonal_element < row_sum:  # Преобладание нарушено
            return False
    return True


def swap_rows(A, b, row1, row2):
    """
    Меняет местами две строки row1 и row2 в матрице A и векторе b.
```

```
    """
    A[row1], A[row2] = A[row2], A[row1]
    b[row1], b[row2] = b[row2], b[row1]


def make_diagonally_dominant(A, b):
    """
    Пытается привести матрицу A к диагонально преобладающей перестановкой
строк.
    Если это невозможно, выводит ошибку.
    """
    n = len(A)
    for perm in permutations(range(n)):  # Перебираем возможные перестановки
строк
        A_permuted = [A[i] for i in perm]  # Переставленная матрица
        b_permuted = [b[i] for i in perm]  # Переставленный вектор b

        if has_diagonal_dominance(A_permuted):
            for i in range(n):
                A[i] = A_permuted[i]
                b[i] = b_permuted[i]
            return True  # Удалось переставить строки

    print("Невозможно привести матрицу к диагонально преобладающей форме.")
    return False  # Не удалось


def build_Bc(A, b):
    """
    Функция формирования матрицы B и вектора c
    для метода простых итераций (x = Bx + c).
    Предполагается, что на главной диагонали нет нулевых элементов.
    """
    n = len(A)
    B = [[0.0 for _ in range(n)] for _ in range(n)]
    c = [0.0 for _ in range(n)]

    for i in range(n):
        diag = A[i][i]
        # Предполагаем, что A[i][i] != 0.
        c[i] = b[i] / diag
        for j in range(n):
            if j != i:
                B[i][j] = -A[i][j] / diag
    return B, c


def simple_iteration_method(B, c, eps, max_iter):  # можно sys.maxsize на
max_iter повесить
    """
    Основная функция итерационного процесса.
    Возвращает кортеж (x, num_iterations), где:
    x - найденный вектор решения,
    num_iterations - реальное число итераций.
    """
    n = len(B)
    x = [0.0 for _ in range(n)]
    x_new = [0.0 for _ in range(n)]

    for k in range(max_iter):
        for i in range(n):
```

```
            temp_sum = 0.0
            for j in range(n):
                temp_sum += B[i][j] * x[j]
            x_new[i] = temp_sum + c[i]

        # Вычисляем вектор погрешностей: |x^(k) - x^(k-1)|
        error_vector = [abs(x_new[i] - x[i]) for i in range(n)]


        # Вывод вектора погрешности
        print(f"Итерация {k}, вектор погрешностей:")
        print(" ".join(f"{e:10.6f}" for e in error_vector))

        print("результат сейчас")
        print(" ".join(f"{x:10.6f}" for x in x_new))


        # Проверяем критерий остановки: если все абсолютные ошибки < eps
        if all(error < eps for error in error_vector):
            return x_new, k, error_vector  # Возвращаем найденное решение и
количество итераций

        # x = x_new.copy()
        x = x_new[:]  # Обновляем x

    return x, max_iter, error_vector


def read_matrix_from_stdout():
    """

    :return:
    """
    # размер матрицы n
    while True:
        try:
            n = int(input("Введите размер матрицы (от 1 до 20): "))
            if 1 <= n <= 20:
                break
            else:
                print("Число вне диапазона [1..20]. Повторите ввод.")
        except ValueError:
            print("Некорректный ввод. Попробуйте ещё раз.")

    # матрица A и вектор b
    print("Введите элементы матрицы A и вектора b (A|b):")
    A = []
    b = []
    for i in range(n):
        while True:
            try:
                row = list(map(float, input().split()))
                if len(row) != n + 1:
                    raise ValueError
                A.append(row[:-1])  # Первые n чисел — матрица A
                b.append(row[-1])  # Последнее число — вектор b
                break
            except ValueError:
                print("Некорректный ввод. Введите ровно", n + 1, "чисел.")

    # точность
```

```
    while True:
        try:
            eps = float(input("Введите желаемую точность: "))
            break
        except ValueError:
            print("Некорректный ввод. Введите число.")

    # максимальное число итераций
    while True:
        try:
            line = input("Введите максимальное число итераций(нажмите ENTER
для неограниченных итераций): ")
            if line == "":
                max_iter = sys.maxsize
                break
            max_iter = int(line)
            if max_iter <= 0:
                raise ValueError
            break
        except ValueError:
            print("Некорректный ввод. Введите натуральное число.")
    return A, b, eps, max_iter


def read_matrix_from_file(filename):
    """
    Функция для чтения матрицы A и вектора b из файла.
    Формат файла:
    n
    A[0][0] A[0][1] ... A[0][n-1] b[0]
    A[1][0] A[1][1] ... A[1][n-1] b[1]
    ...
    A[n-1][0] A[n-1][1] ... A[n-1][n-1] b[n-1]
    eps (точность вычислений)
    max_iter (необязательный параметр, если отсутствует, ставим
"бесконечность")
    :param filename: имя файла, который читаем
    :return: кортеж из
                A - матрицы
                b - вектора
                eps - точности
                max_iter - максимального кол-ва итераций
    """
    with open(filename, "r") as file:
        lines = file.readlines()

    # Считываем размер матрицы n
    n = int(lines[0].strip())
    print(f"Размер данной матрицы: {n}")

    A = []
    b = []

    # матрица A и вектор b
    for i in range(1, n + 1):
        values = list(map(float, lines[i].split()))

        if len(values) != n + 1:
            raise ValueError(f"Ошибка: строка {i} должна содержать {n + 1}
чисел, а не {len(values)}.")
```

```
            A.append(values[:-1])  # Первые n чисел — это строка матрицы A
            b.append(values[-1])  # Последнее число — элемент вектора b

    # точность
    eps = float(lines[n + 1].strip())
    # кол-во итераций
    if len(lines) > n + 2:
        max_iter = int(lines[n + 2].strip())
    else:
        max_iter = sys.maxsize
    return A, b, eps, max_iter


def generate_random_matrix(n, min_value=-10, max_value=10):
    """
    Генерирует случайную матрицу A размером n x n с диагональным
преобладанием
    и случайный вектор b длиной n.
    min_value и max_value — диапазон случайных чисел.
    :param n:
    :param min_value:
    :param max_value:
    :return:
    """
    A = [[random.uniform(min_value, max_value) for _ in range(n)] for _ in
range(n)]
    b = [random.uniform(min_value, max_value) for _ in range(n)]

    # Создаем диагональное преобладание
    for i in range(n):
        A[i][i] = sum(abs(A[i][j]) for j in range(n) if i != j) +
random.uniform(1, 5)  # Диагональ > суммы остальных

    return A, b


def main():
    # Пользователь выбирает способ ввода данных
    choice = 0
    while choice != 1 and choice != 2:
        try:
            choice = int(input("Введите '1' для ввода с клавиатуры, '2' для
чтения из файла: ").strip())
        except ValueError:
            print("Попробуйте ещё раз!")

    if choice == 1:
        A, b, eps, max_iter = read_matrix_from_stdout()
    else:
        filename = input("Введите имя файла (например, matrix.txt):
").strip()
        try:
            A, b, eps, max_iter = read_matrix_from_file(filename)
        except Exception as e:
            print(f"Ошибка чтения файла: {e}")
            return

    # выведем матрицу
    print_augmented_matrix(A, b)

    # диагональное преобразование
```

```
    if not has_diagonal_dominance(A):
        print("Матрица не обладает диагональным преобладанием. Пробуем
перестановку строк...")
        if not make_diagonally_dominant(A, b):
            print("Не удалось достигнуть диагонального преобладания.
Завершаем работу программы...")
            return  # Завершаем программу, если не удалось переставить
строки
        else:
            print("Матрица с диагональным преобладанием:")
            # выведем матрицу
            print_augmented_matrix(A, b)

    # Строим матрицу B и вектор c
    B, c_vec = build_Bc(A, b)

    # Запускаем метод простых итераций
    x_solution, real_iter, error_vector = simple_iteration_method(B, c_vec,
eps, max_iter)

    # Вывод результата
    print("\nРезультат (метод простых итераций):")
    for i, val in enumerate(x_solution):
        print(f"x[{i}] = {val:12.6f}")
    # print(f"Норма матрицы: {}")
    print(f"Число итераций: {real_iter}")
    print("Вектор погрешности:", " ".join(f"{e:10.6f}" for e in
error_vector))


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

Примеры работы программы

Введите '1' для ввода с клавиатуры, '2' для чтения из файла: 1

Введите размер матрицы (от 1 до 20): 3

Введите элементы матрицы A и вектора b (A|b):

2 2 10 14

10 1 1 12

2 10 1 13

Введите желаемую точность: 0.01

Введите максимальное число итераций(нажмите ENTER для неограниченных итераций):

Расширенная матрица [A|b]:

2.0000 2.0000 10.0000 | 14.0000

10.0000 1.0000 1.0000 | 12.0000

2.0000 10.0000 1.0000 | 13.0000

Матрица не обладает диагональным преобладанием. Пробуем перестановку строк...

Матрица с диагональным преобладанием:

Расширенная матрица [A|b]:

10.0000 1.0000 1.0000 | 12.0000

2.0000 10.0000 1.0000 | 13.0000

2.0000 2.0000 10.0000 | 14.0000

Результат (метод простых итераций):

x[0] = 0.999568

x[1] = 0.999460

x[2] = 0.999316

Число итераций: 5

Вектор погрешности: 0.001932 0.002460 0.003084

Вывод:

В результате выполнения данной лабораторной работой я познакомился с численными методами решения математических задач на примере систем алгебраических уравнений, реализовав на языке программирования Java метод простых итераций.