Университет ИТМО

Факультет программной инженерии и компьютерной техникиНаправление подготовки 09.03.04 Программная инженерия Дисциплина «Вычислительная математика»

Отчёт

Лабораторная работа №1

Вариант 10

Выполнил:

Коломиец Никита Сергеевич

P3208

Преподаватель:

Машина Екатерина Алексеевна

Санкт-Петербург, 2024 г

Цель работы

Разработать программу для решения СЛАУ методом простых итераций. В программе численный метод должен быть реализован в виде отдельной подпрограммы/метода/класса, в который исходные/выходные данные передаются в качестве параметров. Размерность матрицы n<=20 (задается из файла или с клавиатуры по выбору конечного пользователя). Должна быть реализована возможность ввода коэффициентов матрицы, как с клавиатуры, так и из файла (по выбору конечного пользователя). Для метода простыхитерация должно быть реализовано:

- Точность задается с клавиатуры/файла
- Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - выводить соответствующее сообщение.
- Вывод вектора неизвестных: x_1, x_2, …, x_n.
- Вывод количества итераций, за которое было найдено решение.
- Вывод вектора погрешностей: $|x_i^{(k)}-x_i^{(k-1)}|$.

Описание метода

Итерационные методы - это методы последовательных приближений. Задается некоторое начальное приближение. Далее с помощью определенногоалгоритма проводится один цикл вычислений - итерация. В результате итерациинаходят новое приближение. Итерации проводятся до получения решения с требуемой точностью.

В методе простых итераций сначала исходную СЛАУ:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}\,x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}\,x_n = b_2 \\ \qquad \dots\dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}\,x_n = b_n \end{cases}$$

представляют в виде:

$$\begin{cases} x_1 = \dfrac{a_{12}}{a_{11}}x_2 + \dfrac{a_{13}}{a_{11}}x_3 + \dots + \dfrac{a_{1n}}{a_{11}}x_n - \dfrac{b_1}{a_{11}} \\ x_2 = \dfrac{a_{21}}{a_{22}}x_1 + \dfrac{a_{23}}{a_{22}}x_3 + \dots + \dfrac{a_{2n}}{a_{22}}x_n - \dfrac{b_2}{a_{22}} \\ \qquad \dots\dots \\ x_n = \dfrac{a_{n1}}{a_{nn}}x_1 + \dfrac{a_{n2}}{a_{nn}}x_2 + \dots + \dfrac{a_{n-1n-1}}{a_{nn}}x_{n-1} - \dfrac{b_n}{a_{nn}} \end{cases}$$

Обозначив:

$$c_{ij} = \begin{cases} 0, & \text{при } i = j \\ -\dfrac{a_{ij}}{a_{ii}}, & \text{при } i \neq j \end{cases}$$

$$d_i = \frac{b_i}{a_{ii}} \quad i = 1, 2, \dots, n$$

Получим СЛАУ и запишем в сокращенном виде:

$$\boldsymbol{x_i = \sum_{j=1}^{n} c_{ij}x_j + d_i}\ , \qquad i = 1, 2, \dots, n$$

Рабочая формула метода простых итераций:

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{a_{ij}}{a_{ii}} x_j^k \ , \qquad i = 1, 2, \dots, n$$

С каждой k-ой итерацией решения будут приближаться к действительному решению СЛАУ. Важным критерием этого является то, что матрица коэффициентов СЛАУ должна иметь диагональное преобладание.

Листинг части программы, реализующей сам метод

```
# determinant.py
def minor(matrix, i, j) -> list[list[float]]:
    tmp = matrix
    tmp = tmp[:i] + tmp[i + 1:]
    for k in range(len(tmp)):
        tmp[k] = tmp[k][:j] + tmp[k][j + 1:]
    return tmp


def calculate(matrix, n) -> float:
    if n == 1: return matrix[0][0]
    if n == 2: return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]
    det = 0
    for i in range(n):
        tmp = minor(matrix, 0, i)
        det += (-1) ** i * matrix[0][i] * calculate(tmp, n - 1)
    return det
```

```
# matrix.py
def read() -> list[list[float]]:
    size_sle = int(input("Введите размерность СЛАУ: "))
    sle = []

    print("Введите матрицу ->")

    for row in range(size_sle):
        print(f"{row + 1}:", end=" ")
        sle.append([float(a) for a in input().split()])

    return sle


def read_from_file():
    filename = input("Введите имя файла: ")
    data = []
    try:
        file = open(filename, 'r')
        size = int(file.readline().strip())
        data = file.readlines()
    except FileNotFoundError:
        print("Ошибка! Нет такого файла")
        return

    matrix = []
    for i in range(size):
        matrix.append([float(a) for a in data[i].split()])
    print("Матрица успешно считана!")

    return matrix


def make_square(sle) -> list[list[float]]:
    tmp = []
    for i in range(len(sle)):
        tmp.append(sle[i][:len(sle)])
    return tmp


def print_matrix(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            print(matrix[i][j], end=" ")
        print()


def check_predominance(matrix: list[list[float]]):
    size = len(matrix)
```

```
    all_indexes = set()
    sort_matrix = []
    for i in range(size):
        all_indexes.add(matrix[i].index(max(matrix[i][:len(matrix[i]) - 1])))
        sort_matrix.append([i, matrix[i].index(max(matrix[i][:len(matrix[i]) -
1]))])

    if size == len(all_indexes):
        sort_matrix = sorted(sort_matrix, key=lambda x: x[-1], reverse=False)
    else:
        print("Не выполняется условие диагонального преобладания")
        return

    tmp = []
    for i in range(size):
        tmp.append(matrix[sort_matrix[i][0]])

    return tmp
```

iteration_mode.py

```
from prettytable import PrettyTable
import matrix


def run(sle, det):
    if det == 0:
        print("СЛАУ имеет вырожденную матрицу")
        return
    else:
        sle = matrix.check_predominance(sle)
        if sle is None:
            return
        epsilon = float(input("Введите точность решения: "))
        # sle = sorted(sle, key=lambda x: x[-1], reverse=False)
        result = express_unknown(sle)
        # matrix.print_matrix(result)
        zero_solve = get_zero_solve(result)
        solving_iterations(0, epsilon, result, zero_solve,
get_zero_solve(result), PrettyTable())


def express_unknown(sle) -> list[list[float]]:
    tmp = []
    for i in range(len(sle)):
        row = []
        for j in range(len(sle[i])):
            if i == j:
                row.append(0)
                continue
            row.append((-1) * (sle[i][j] / sle[i][i]))
        row[-1] *= -1
        tmp.append(row)

    return tmp


def print_iteration(count_iter, solve, solve_epsilon, table) -> PrettyTable:
    if count_iter == 0:
        th = ["k"]
        for i in range(len(solve)):
            th.append("x_" + str(i + 1))
        th.append("max|x_k - x_(k-1)|")
        table = PrettyTable(th)
        td = [count_iter]
        for i in solve:
            td.append(i)
        td.append(0)
        table.add_row(td)
        return table
    else:
```

```
        td = [count_iter]
        for i in solve:
            td.append(i)
        td.append(max(solve_epsilon))
        table.add_row(td)
    return table


def get_zero_solve(sle) -> list[float]:
    solve = []
    for i in range(len(sle)):
        solve.append(sle[i][-1])
    return solve


def solving_iterations(count_iter, epsilon, sle, solve, solve_epsilon, table):
    table = print_iteration(count_iter, solve, solve_epsilon, table)

    if max(solve_epsilon) < epsilon and count_iter != 0:
        print(table)
        return

    if count_iter > 20:
        print(table)
        return

    tmp_solve = []

    for i in range(len(solve)):
        x = sle[i][-1]
        for j in range(len(sle[i]) - 1):
            x += sle[i][j] * solve[j]
        tmp_solve.append(round(x, 4))

    for i in range(len(solve)):
        solve_epsilon[i] = abs(round(tmp_solve[i] - solve[i], 4))
        solve[i] = tmp_solve[i]

    return solving_iterations(count_iter + 1, epsilon, sle, solve,
solve_epsilon, table)
```

run.py

```
import determinant
import matrix
import iteration_method

variant = input("Выберите формат ввода данных:\n-> 1. Файл\n-> 2. Консоль\n-> ")
while variant != "exit":
    if variant == "1" or variant == "2":
        test = matrix.read() if variant == "2" else matrix.read_from_file()
        square_matrix = matrix.make_square(test)
        det = determinant.calculate(square_matrix, len(test))
        iteration_method.run(test, det)
    variant = input("Выберите формат ввода данных:\n-> 1. Файл\n-> 2. Консоль\n-
> ")
print("До свидания!")
```

Пример работы программы

```
Выберите формат ввода данных:
-> 1. Файл
-> 2. Консоль
-> 1
Введите имя файла: test
Матрица успешно считана!
Введите точность решения: 0.01
+---+--------+--------+--------+--------------------+
| k |  x_1   |  x_2   |  x_3   | max|x_k - x_(k-1)| |
+---+--------+--------+--------+--------------------+
| 0 |  1.2   |  1.3   |  1.4   |         0          |
| 1 |  0.93  |  0.92  |  0.9   |        0.5         |
| 2 | 1.018  | 1.024  |  1.03  |        0.13        |
| 3 | 0.9946 | 0.9934 | 0.9916 |       0.0384       |
| 4 | 1.0015 | 1.0019 | 1.0024 |       0.0108       |
| 5 | 0.9996 | 0.9995 | 0.9993 |       0.0031       |
+---+--------+--------+--------+--------------------+

Выберите формат ввода данных:
-> 1. Файл
-> 2. Консоль
-> 2
Введите размерность СЛАУ: 3
Введите матрицу ->
1: 2 2 10 14
2: 10 1 1 12
3: 2 10 1 13
Введите точность решения: 0.01
+---+--------+--------+--------+--------------------+
| k |  x_1   |  x_2   |  x_3   | max|x_k - x_(k-1)| |
+---+--------+--------+--------+--------------------+
| 0 |  1.2   |  1.3   |  1.4   |         0          |
| 1 |  0.93  |  0.92  |  0.9   |        0.5         |
| 2 | 1.018  | 1.024  |  1.03  |        0.13        |
| 3 | 0.9946 | 0.9934 | 0.9916 |       0.0384       |
| 4 | 1.0015 | 1.0019 | 1.0024 |       0.0108       |
| 5 | 0.9996 | 0.9995 | 0.9993 |       0.0031       |
+---+--------+--------+--------+--------------------+
```

Вывод

В результате выполнения данной лабораторной работы был изучен метод простых итераций для решения СЛАУ. Основное преимущество данного метода: он прост в реализации на ЭВМ и является универсальным. Однако он имеет и недостатки: в результате получается неточное решение (точность задается пользователем), итераций может быть очень много, матрица коэффициентов СЛАУ должна иметь диагональное преобладание.