Министерство науки и высшего образования Российской Федерации
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 №13

Выполнил:

Студент группы P3213

Султанов А.Р.

Проверила:

Машина Е.А.

г. Санкт-Петербург

2024г.

Цель работы

Изучить методы решения систем линейных алгебраических уравнений. Реализовать один из них (по варианту - метод простых итераций) в виде программы.

Описание метода, расчетные формулы

Итерационные (приближенные) методы решения линейных систем позволяют получить последовательность векторов $\bar{x}^{(0)}, \bar{x}^{(1)}, \ldots, \bar{x}^{(k)}$, предел которой - точное решение $\bar{x}^{(*)}: \bar{x}^{(*)} = \lim_{i \to \infty} \bar{x}^{(i)}$. Выполнение (построение последовательности) заканчивается тогда, когда достигается желаемая (подается как один из входных параметров) точность.

Есть разные критерии для оценки точности, в данном случае используется критерий по абсолютным отклонениям:

$$max_{1 \le i \le n} |x_i^{(k)} - x_i^{(k-1)}| \le \varepsilon$$

Рабочая формула метода простой итерации:

$$x_i^{(k+1)} = \frac{bi}{aii} - \sum_{j=1, j \neq i}^{n} \frac{aii}{aij} x_j^k.$$

Листинг программы

Полный исходный код доступен здесь: https://github.com/sultanowskii/itmo-edu/tree/master/comp-maths/lab1/src.

Основная часть программы, реализующая логику решения (файл **solve.py**):

```
from error import print_error_and_exit
from matrix import (
    Matrix1D,
    MatrixSquare,
    create_1_n_matrix,
    create_n_n_matrix,
```

```
)


def find_predominant_element_index(m: Matrix1D) -> int | None:
    """Находит индекс преобладающего элемента в строке."""
    abs_m = [abs(e) for e in m]
    abs_m_sum = sum(abs_m)

    for i in range(len(m)):
        elem = abs_m[i]
        rest = abs_m_sum - elem
        if elem > rest:
            return i

    return None


def sort_by_diagonal_element_predominance(a: MatrixSquare, b: Matrix1D) ->
tuple[MatrixSquare, Matrix1D]:
    """Сортирует строки (и A, и B) так, чтобы получилась матрица с
диагональным преобладанием."""
    lines = [(a[i].copy(), b[i]) for i in range(len(a))]

    predominant_indexes = [find_predominant_element_index(row) for row in a]

    if any(map(lambda index: index is None, predominant_indexes)):
        print_error_and_exit('Диагональное преобладание не может быть
достигнуто для матрицы A.')

    if len(set(predominant_indexes)) != len(predominant_indexes):
        print_error_and_exit('Диагональное преобладание не может быть
достигнуто для матрицы A.')

    lines.sort(key=lambda line: find_predominant_element_index(line[0]))

    return list(map(lambda line: line[0], lines)), list(map(lambda line:
line[1], lines))


def solve(n: int, a: MatrixSquare, b: Matrix1D, accuracy: float) ->
tuple[Matrix1D, int]:
    """
```

```
    Решает СЛАУ (a, b) с заданной точностью.

    Помимо решения, возвращает число итераций.
    """
    c = create_n_n_matrix(n)
    d = create_1_n_matrix(n)

    a, b = sort_by_diagonal_element_predominance(a, b)

    # (5) -> (6)
    for i in range(n):
        for j in range(n):
            if i != j:
                c[i][j] = - (a[i][j] / a[i][i])

    for i in range(n):
        d[i] = b[i] / a[i][i]

    return _solve(n, c, d, accuracy)


def _solve(n: int, c: MatrixSquare, d: Matrix1D, accuracy: float, prev_x:
Matrix1D = None) -> tuple[Matrix1D, int]:
    """Итерация решения."""
    x = create_1_n_matrix(n)

    if not prev_x:
        prev_x = create_1_n_matrix(n)

    for i in range(n):
        x[i] = d[i]
        for j in range(n):
            x[i] += c[i][j] * prev_x[j]

    abs_deviations = create_1_n_matrix(n)

    for i in range(n):
        abs_deviations[i] = abs(x[i] - prev_x[i])

    criteria = max(abs_deviations)

    print(criteria)
```

```
    if criteria < accuracy:
        return x, 1

    solution, iterations = _solve(n, c, d, accuracy, x)
    return solution, iterations + 1
```

Примеры работы программы

```
❯ cat example.yaml
N: 3
accuracy: 0.0001
a:
- [2, 2, 10]
- [10, 1, 1]
- [2, 10, 1]
b: [14, 12, 13]
❯ python go.py -f example.yaml

n=3
accuracy=0.0001

A:
2 2 10
10 1 1
2 10 1

B:
14
12
13

вектор погрешностей:
1.4
0.5
0.13000000000000012
0.0384000000000001
0.010800000000000032
0.0030839999999999756
0.0008784000000000569
```

```
0.0002498400000001677
7.120800000004479e-05

число итераций: 9

решение:
1.000009936
1.000012528
1.000015768
```

Вывод

В результате выполнения лабораторной работы я ознакомился с различными методами решения СЛАУ и реализовал один из них - метод простых итераций. Одним из главных сравнительных преимуществ данного метода является меньшее потребление памяти программы и факт того, что погрешность не накапливается, т.к.точность на каждой итерации зависит от точности предыдущей итерации. Главным недостатком итерационного метода является его сложность.