

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 по Вычислительной Математике №1
Метод простых итераций
Вариант №10

Преподаватель: Малышева Татьяна Алексеева

Выполнил: Состанов Тимур Айратович

Группа: Р3214

Г. Санкт-Петербург

2024

Оглавление

Цель работы.....	3
Описание метода, расчётные формулы.....	4
Листинг программы.....	6
Примеры и результаты работы программы	7
Работа из файла	7
Работа из консоли.....	7
Результат работы программы одинаковый для обоих способов ввода исходных данных:	8
Вывод	8

Цель работы

Используя известные методы вычислительной математики, написать программу, осуществляющий решение СЛАУ итерационным методом. Проанализировать полученные результаты, оценить погрешность.

Описание метода, расчётные формулы

Итерационные методы. Метод простой итерации

Рассмотрим систему линейных уравнений с невырожденной матрицей ($\det A \neq 0$):

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases} \quad (5)$$

Приведем систему уравнений к виду (6), выразив неизвестные x_1, x_2, \dots, x_n соответственно из первого, второго и т.д. уравнений системы (5):

$$\begin{cases} x_1 = -\frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3 - \dots - \frac{a_{1n}}{a_{11}}x_n + \frac{b_1}{a_{11}} \\ x_2 = -\frac{a_{21}}{a_{22}}x_1 - \frac{a_{23}}{a_{22}}x_3 - \dots - \frac{a_{2n}}{a_{22}}x_n + \frac{b_2}{a_{22}} \\ \dots \dots \\ x_n = -\frac{a_{n1}}{a_{nn}}x_1 - \frac{a_{n2}}{a_{nn}}x_2 - \dots - \frac{a_{n-1n-1}}{a_{nn}}x_{n-1} + \frac{b_n}{a_{nn}} \end{cases} \quad (6)$$

Обозначим:

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

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

Тогда получим:

$$\begin{cases} x_1 = c_{11}x_1 + c_{12}x_2 + \dots + c_{1n}x_n + d_1 \\ x_2 = c_{21}x_1 + c_{22}x_2 + \dots + c_{2n}x_n + d_2 \\ \dots \dots \\ x_n = c_{n1}x_1 + c_{n2}x_2 + \dots + c_{nn}x_n + d_n \end{cases}$$

Или в векторно-матричном виде: $\mathbf{x} = \mathbf{Cx} + \mathbf{D}$, где \mathbf{x} – вектор неизвестных, \mathbf{C} – матрица коэффициентов преобразованной системы размерности $n \times n$, \mathbf{D} – вектор правых частей преобразованной системы.

Систему (6) представим в сокращенном виде:

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

$$c_{ij} = \begin{cases} 0, & \text{при } i = j \\ -\frac{a_{ij}}{a_{ii}}, & \text{при } i \neq j \end{cases} \quad d_i = \frac{b_i}{a_{ii}} \quad i = 1, 2, \dots, n$$

Рабочая формула метода простой итерации:

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

где k – номер итерации.

За начальное (нулевое) приближение выбирают вектор свободных членов: $x^{(0)} = D$ или нулевой вектор: $x^{(0)} = 0$

Следующее приближение: $\vec{x}^{(1)} = c\vec{x}^{(0)} + \vec{d}$, $\vec{x}^{(2)} = c\vec{x}^{(1)} + \vec{d} \dots$

Листинг программы

```
Usage: tsostanov

def simple_optimizations(transformed_matrix, start_approximation, accuracy):
    answer = 1

    print(" Метод простых итераций")
    print(" | ".join(["k ", *(f"x{i}" for i in range(1, len(start_approximation) + 1)),
                     "difference"]))
    print(" | ".join(
        ["0 "] + [str(item) + (" " * (18 - len(str(item)))) for sublist in start_approximation for item in sublist] + [
            "---"])))

    vector_of_right_parts = start_approximation[:]
    delta_approximation = [[] for _ in range(len(start_approximation))]
    factor_start = calculate_factor_for_array(start_approximation)

    for number, coefficients in enumerate(transformed_matrix):
        factor = max([10 ** max(
            (len(str(number_str)) - str(number_str).index('.') - 1) if '.' in str(number_str) else 0 for number_str in
            coefficients), factor_start])

        delta_approximation[number] = [
            (sum(int(start_approximation[i][0] * factor) * int(transformed_matrix[number][i] * factor)
                for i in range(len(start_approximation))) + int(vector_of_right_parts[number][0] * (factor ** 2))) / (
                factor ** 2)]

    factor_delta = calculate_factor_for_array(delta_approximation)
    factor_result = max([factor_delta, factor_start])

    accuracy_check = (max(
        abs(int(delta_approximation[i][0] * factor_result) - int(start_approximation[i][0] * factor_result)) for i in
        range(len(delta_approximation))) / factor_result)
    print(" | ".join(
        [str(answer) + " "] + [str(item) + (" " * (18 - len(str(item)))) for sublist in delta_approximation for item in
            sublist] + [str(accuracy_check)]))

    while accuracy_check >= accuracy:
        start_approximation = delta_approximation
        answer += 1

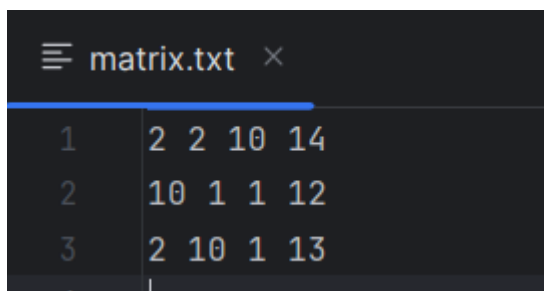
        delta_approximation = [[] for _ in range(len(start_approximation))]
        factor_start = calculate_factor_for_array(start_approximation)

        for number, coefficients in enumerate(transformed_matrix):
            factor = max([10 ** max(
                (len(str(number_str)) - str(number_str).index('.') - 1) if '.' in str(number_str) else 0 for number_str in
                coefficients), factor_start])
            delta_approximation[number] = [
                (sum(int(start_approximation[i][0] * factor) * int(transformed_matrix[number][i] * factor)
                    for i in range(len(start_approximation))) +
                    int(vector_of_right_parts[number][0] * (factor ** 2))) / (factor ** 2)]
        factor_delta = calculate_factor_for_array(delta_approximation)
        factor_result = max([factor_delta, factor_start])
        accuracy_check = (max(
            abs(int(delta_approximation[i][0] * factor_result) - int(start_approximation[i][0] * factor_result)) for i
            in range(len(delta_approximation))) / factor_result)
        print(" | ".join(
            [str(answer) + " "] + [str(item) + (" " * (18 - len(str(item)))) for sublist in delta_approximation for item
                in sublist] + [str(accuracy_check)]))

    return answer, delta_approximation
```

Примеры и результаты работы программы

Работа из файла:



1	2	2	10	14
2	10	1	1	12
3	2	10	1	13

Введите точность:

0.0001

Выберите режим работы: 1 - ввод с клавиатуры, 2 - работа с файлом

2

Введите название файла с матрицей:

matrix.txt

Работа из консоли:

Введите точность:

0.0001

Выберите режим работы: 1 - ввод с клавиатуры, 2 - работа с файлом

1

Введите количество строк матрицы:

3

Вводите матрицу:

2 2 10 14

10 1 1 12

2 10 1 13

Результат работы программы одинаковый для обоих способов ввода исходных данных:

Введенная матрица:

```
2.0  2.0 10.0 14.0
10.0  1.0  1.0 12.0
 2.0 10.0  1.0 13.0
```

Диагональная форма матрицы:

```
10.0  1.0  1.0 12.0
 2.0 10.0  1.0 13.0
 2.0  2.0 10.0 14.0
```

Матрица коэффициентов преобразованной системы:

```
0 -0.1 -0.1
-0.2  0 -0.1
-0.2 -0.2  0
```

Вектор правых частей преобразованной системы:

```
1.2
1.3
1.4
```

Метод простых итераций

k	x1	x2	x3	difference
0	1.2	1.3	1.4	---
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	1.00192	1.0024	0.0108
5	0.999569	0.99946	0.999318	0.003082
6	1.0001222	1.0001544	1.0001942	0.0008762
7	0.99996514	0.99995614	0.99994467999999	0.00024952000001
8	1.0000099180000008	1.0000125040000012	1.000015744	7.106400001e-05

Проверка

```
12.00012742800001
13.000160620000013
14.000202284000004
```

Невязка

```
0.0001274280000096
0.0001606200000128
0.0002022840000032
```

Вывод

В ходе выполнения лабораторной работы был реализован метод простых итераций, позволяющий решать СЛАУ итерационным методом. Метод показал быструю сходимость.