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

Факультет Программной Инженерии и Компьютерной Техники

Вычислительная математика

Лабораторная работа №2.

Метод Гаусса — Зейделя

Выполнил:

Маликов Александр Максимович

Группа № P3222

Преподаватель:

Перл Ольга Вячеславовна

г. Санкт-Петербург

2024

Оглавление

Задание

Решите систему линейных алгебраических уравнений, реализуя метод Гаусса-Зейделя.

Формат входных данных:

n

a11 a12 ... a1n b1

a21 a22 ... a2n b2

...

an1 an2 ... ann bn

Формат вывода:

x1

x2

...

xn

, где x1..xn - значения неизвестных.

Если для текущей матрицы нет диагонального преобладания, вам следует попытаться найти его путем перестановки столбцов или / и строк. Если после такой операции преобладание диагонали по-прежнему отсутствует, должно быть напечатано следующее сообщение:

"The system has no diagonal dominance for this method. Method of the Gauss-Seidel is not applicable.". Для этого задайте значение переменной isMethodApplicable и сообщение об ошибке.

Описание численного метода

Метод Гаусса – Зейделя

Метод решения СЛАУ Гаусса – Зейделя представляет собой один из методов итерационного решения систем уравнений. Этот метод является самым распространенным из тех, что используются для решения СЛАУ, в силу своей алгоритмической простоты и, как следствие, легкостью программирования.

У метода также присутствует условие выполнимости (сходимости), которое наследуется от метода простых итераций с некоторым дополнением. Условие – слабое диагональное доминирование

Слабое диагональное доминирование:

$$|a_{ii}| \geq \sum_{i \neq k} |a_{ik}| (i, k = 1, 2, \dots, n)$$

Обязательное условие хотя бы для одного уравнение в СЛАУ:

$$|a_{ii}| > \sum_{i \neq k} |a_{ik}| (i, k = 1, 2, \dots, n)$$

После того, как разобрались с условиями сходимости перейдём к описанию самого метода вычислений.

Суть заключается в итеративном вычислении приближений значений X-ов. На старте необходимо выбрать начальные значения для каждого из X, это условие также наследуется из метода простых итераций. Подходящими значениями могут быть:

1. 0
2. b_i
3. Предварительно рассчитанное значение для повышения точности.
4. Любое другое значение.

Далее в итеративном порядке вычисляем все последующие значения по формуле:

$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - a_{ii} x_1^{(k)} - \cdots - a_{ii-1} x_{i-1}^{(k)} - a_{ii+1} x_{i+1}^{(k-1)} - \cdots - a_{in} x_n^{(k-1)} \right)$, где i = 1, 2, …, n; k = 1, 2, …

Итерационный процесс продолжается до тех пор, пока абсолютно все значения $x_i^{(k)}$ не станут достаточно приближенными ко всем $x_i^{(k-1)}$. Отсюда выражается условие окончания итерационного процесса:

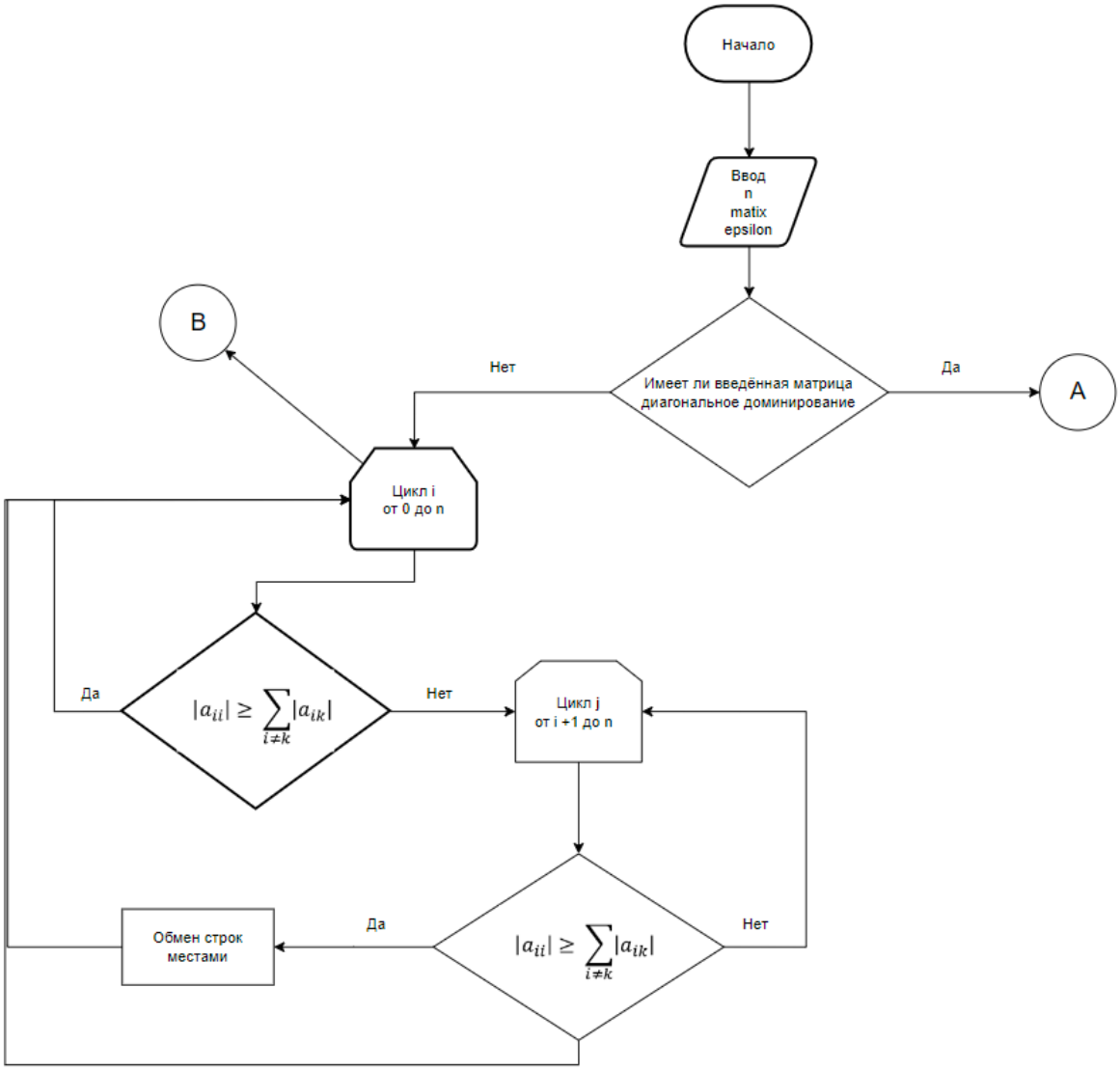
$$\forall i (i = 1,2, \dots, n) \forall \varepsilon > 0 : \left| x_i^{(k)} - x_i^{(k-1)} \right| \leq \varepsilon$$

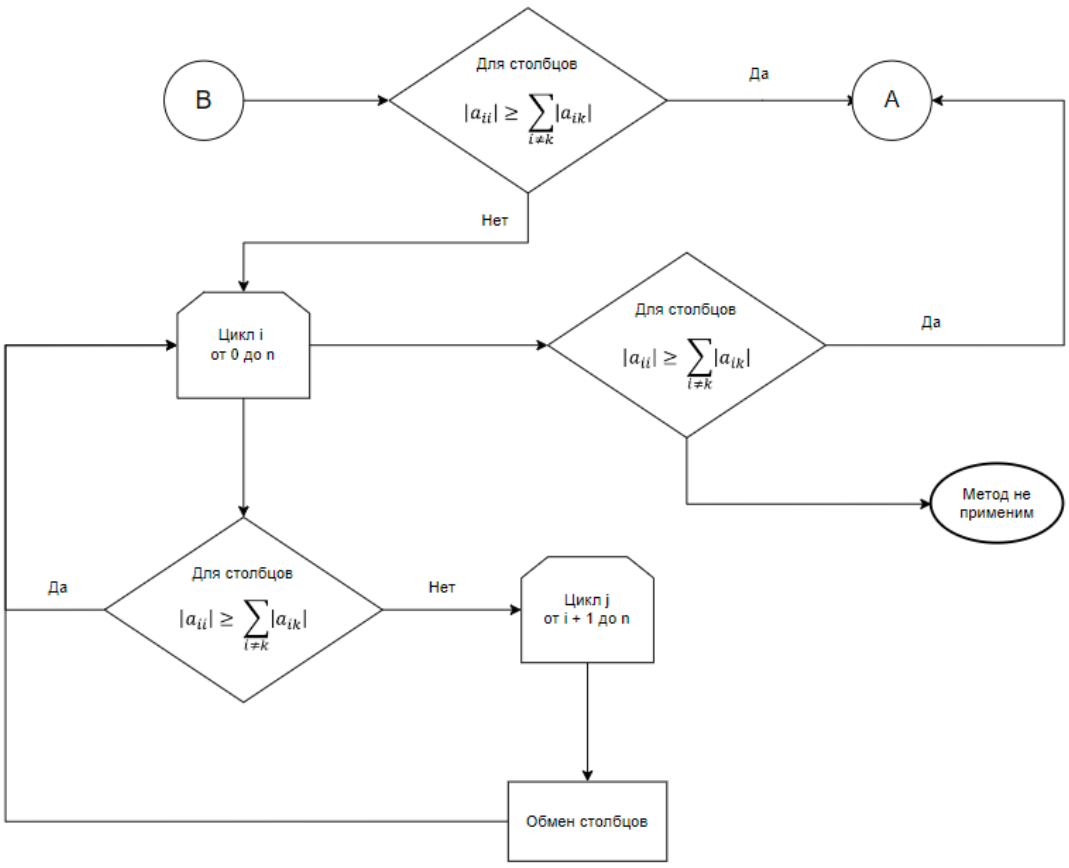
Из достоинств метода определённо можно выделить возможность увеличивать или уменьшать погрешность вычисление, для этого достаточно увеличить или уменьшить значение ε соответственно. Также следует отметить возможность параллельных вычисление, так как каждое уравнение текущей итерации независимо друг от друга.

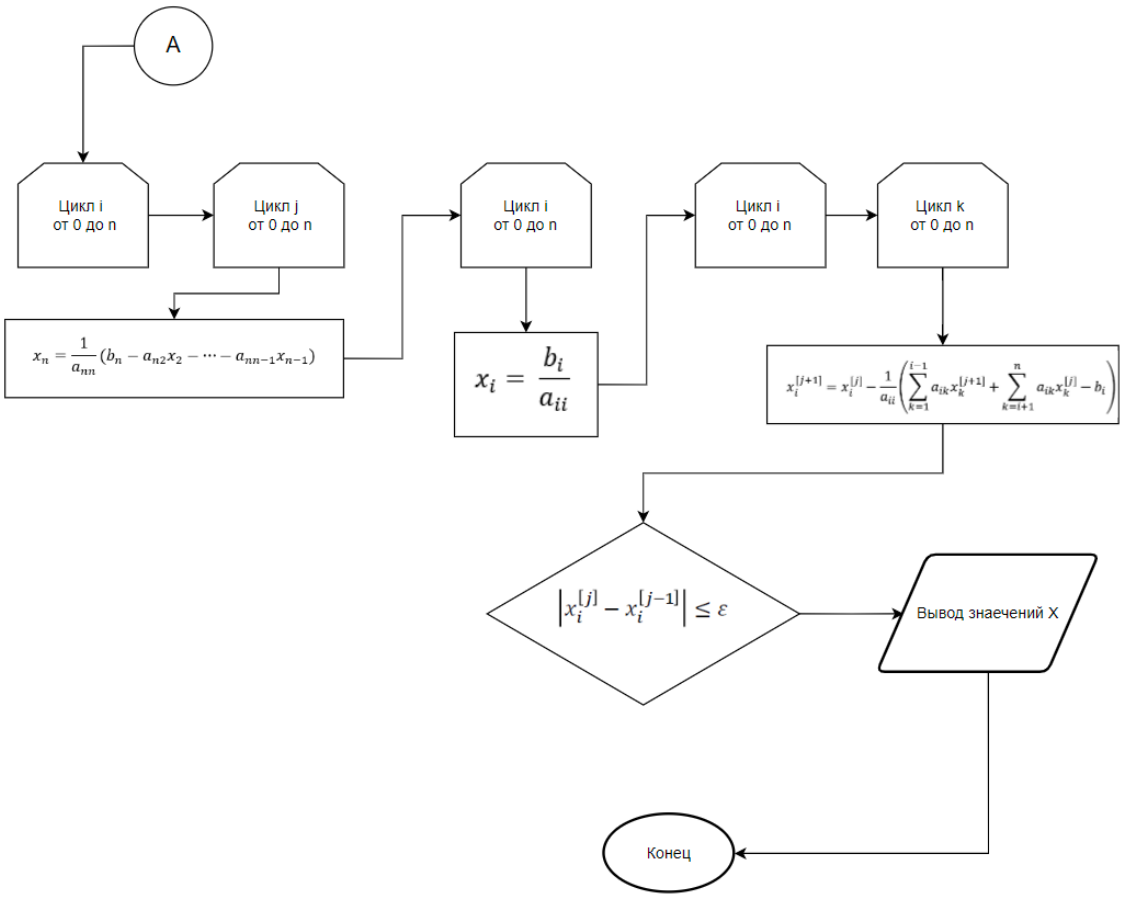
Также присутствуют несколько недостатков:

1. Жесткое условие применимости метода ограничивает пул матриц, которые могут быть применены.
2. Алгоритмическая сложность напрямую зависит от числа итераций, при количестве итераций l: $O(l * n^2)$

Блок – схемы


Начало
Ввод
n
matix
epsilon
Имеет ли введённая матрица диагональное доминирование
Нет
Да
A
B
Цикл i
от 0 до n
$|a_{ii}| \geq \sum_{i \neq k} |a_{ik}|$
Да
Нет
Цикл j
от i +1 до n
$|a_{ii}| \geq \sum_{i \neq k} |a_{ik}|$
Да
Нет
Обмен строк
местами


B
Для столбцов
$|a_{ii}| \geq \sum_{i \neq k} |a_{ik}|$
Да
A
Нет
Цикл i
от 0 до n
Для столбцов
$|a_{ii}| \geq \sum_{i \neq k} |a_{ik}|$
Да
Метод не применим
Да
Для столбцов
$|a_{ii}| \geq \sum_{i \neq k} |a_{ik}|$
Нет
Цикл j
от i + 1 до n
Обмен столбцов


A
Цикл i
от 0 до n
Цикл j
от 0 до n
$x_n = \frac{1}{a_{nn}}(b_n - a_{n2}x_2 - \cdots - a_{nn-1}x_{n-1})$
Цикл i
от 0 до n
$x_i = \frac{b_i}{a_{ii}}$
Цикл i
от 0 до n
Цикл k
от 0 до n
$x_i^{[j+1]} = x_i^{[j]} - \frac{1}{a_{ii}}\left(\sum_{k=1}^{i-1} a_{ik}x_k^{[j+1]} + \sum_{k=i+1}^{n} a_{ik}x_k^{[j]} - b_i\right)$
$\left|x_i^{[j]} - x_i^{[j-1]}\right| \leq \varepsilon$
Вывод знаечений X
Конец

Код программы

```
class Result:
    errorMessage = ""
    isMethodApplicable = True

    @staticmethod
    def solveByGaussSeidel(n, matrix, epsilon):
        indexes = []

        for i in range(n):
            indexes.append(i)

        for i in range(n):
            if len(matrix[i]) != n + 1 or matrix[i][i] == 0:
                Result.error_message = "The system has no diagonal dominance
for this method. Method of the " \
                                       "Gauss-Seidel is not applicable."
                Result.isMethodApplicable = False
                return []

        if not check_on_diagonally_dominant(matrix):
            for i in range(len(matrix)):
                if matrix[i][i] > abs_sum_row(matrix[i]) - matrix[i][i]:
                    continue
                elif matrix[i][i] == abs_sum_row(matrix[i]) - matrix[i][i]:
                    continue
                else:
                    for j in range(i + 1, len(matrix)):
                        if matrix[j][i] >= abs_sum_row(matrix[j]) -
matrix[j][i]:
                            matrix[i], matrix[j] = matrix[j], matrix[i]
                            break
            if not check_on_diagonally_dominant(matrix):
                for i in range(len(matrix)):
                    column = []
                    for j in range(len(matrix)):
                        column.append(matrix[j][i])
                    if column[i] > abs_sum(column) - column[i]:
                        continue
                    elif column[i] == abs_sum(column) - column[i]:
                        continue
                    else:
                        for j in range(i + 1, len(matrix)):
                            new_column = []
                            for k in range(len(matrix)):
                                new_column.append(matrix[k][j])
                            if column[i] < abs_sum(column) - column[i]:
                                matrix = change_columns(matrix, i, j)
                                indexes[i], indexes[j] = indexes[j],
indexes[i]
                                break
        if not check_on_diagonally_dominant(matrix):
            Result.isMethodApplicable = False
            Result.errorMessage = "The system has no diagonal dominance for
this method. Method of the " \
                                  "Gauss-Seidel is not applicable."
            return []

        vector = []
        for i in range(n):
            vector.append(matrix[i][-1])
            matrix[i].pop(-1)
```

```
        transform_matrix, transform_vector = transformation(matrix, vector)

        x_n_0 = zero_approximation(transform_vector, vector)

        x_n_1 = [0] * len(transform_matrix)

        x_n_1 = n_approximation(x_n_0, x_n_1, transform_matrix)

        while calc(x_n_0, x_n_1, epsilon):
            x_n_0 = x_n_1
            x_n_1 = [0] * len(transform_matrix)
            x_n_1 = n_approximation(x_n_0, x_n_1, transform_matrix)

        result = []
        for index in indexes:
            result.append(x_n_1[index])
        return result


def abs_sum(line):
    sum = 0
    for i in range(len(line)):
        sum += abs(line[i])
    return sum


def abs_sum_row(row):
    sum = 0
    for i in range(len(row) - 1):
        sum += abs(row[i])
    return sum


def change_columns(matrix, row_1, row_2):
    for i in range(len(matrix)):
        if row_1 != len(matrix) and row_2 != len(matrix):
            matrix[i][row_1], matrix[i][row_2] = matrix[i][row_2],
matrix[i][row_1]
    return matrix


def transformation(input_matrix, input_vector):
    buffer = input_vector.copy()
    for i in range(len(input_matrix[0])):
        buffer[i] = input_matrix[i][i]
        input_matrix[i][i] = input_vector[i]
        for j in range(len(input_matrix[i])):
            if input_matrix[i][j] != input_vector[i]:
                input_matrix[i][j] = input_matrix[i][j] * -1
            input_matrix[i][j] = input_matrix[i][j] / buffer[i]
    return input_matrix, buffer


def check_on_diagonally_dominant(matrix):
    diagonally_dominant = False
    counter = 0
    for i in range(len(matrix)):
        sum = 0
        for j in range(len(matrix)):
            if i != j:
                sum += abs(matrix[i][j])
        if abs(matrix[i][i]) > sum:
            diagonally_dominant = True
        if abs(matrix[i][i]) >= sum:
```

```
            counter += 1

    if diagonally_dominant and counter == len(matrix):
        return True
    else:
        return False


def zero_approximation(transform_vector, vector):
    return_arr = []
    for i in range(len(vector)):
        return_arr.append(vector[i] / transform_vector[i])
    return return_arr


def n_approximation(x_n_1, x_n_2, transform_matrix):
    return_arr = []
    for i in range(len(transform_matrix)):
        sum = 0
        for j in range(len(transform_matrix)):
            if i == j:
                new_x = transform_matrix[i][j]
            elif j > i:
                new_x = transform_matrix[i][j] * x_n_1[j]
            else:
                new_x = transform_matrix[i][j] * x_n_2[j]
            sum += new_x
        x_n_2[i] = sum
        return_arr.append(sum)
    return return_arr


def calc(x_n_0, x_n_1, epsilon):
    for i in range(len(x_n_0)):
        if x_n_1[i] == 0:
            continue
        calc = (x_n_1[i] - x_n_0[i]) / x_n_1[i]
        if abs(calc) > epsilon:
            return True
    return False
```

Примеры выполнения

Пример 1

Ввод:

3

1.7 2.8 11.9 0.7

12.1 1.8 1.3 1.1

4.2 -11.7 1.3 2.8

0.0001

Результат:

0.10975340532381547

-0.1901524908000699

0.08788581175319793

На ввод передаётся матрица, а которой для диагонализированного доминирования необходимо поменять некоторые строки местами.

Пример 2

Ввод:

5

1 1 0 0 0 1

1 1 1 0 0 4

0 1 1 1 0 -3

0 0 1 1 1 2

0 0 0 1 1 -1

0.001

Вывод программы:

The system has no diagonal dominance for this method. Method of the Gauss-Seidel is not applicable.

Пример 3

Ввод:

3

15.8 -4.1 -1.7 2.8

1.3 22.8 3.3 6.89

2.66 0.9 15.3 7.1

0.0001

Вывод программы:

0.2796504861773621

0.22806124636314032

0.40201794673473135

Входная матрица уже представляла собой диагонально доминирующую, так что никакие строки и столбцы менять не пришлось.

Пример 4

Ввод:

4

9 1 1 1 10

1 9 1 1 10

1 1 9 1 10

1 1 1 9 10

0.000000000001

Вывод программы:

0.8333333333333441

0.8333333333333348

0.8333333333333311

0.8333333333333323

Решения должны быть одинаковыми, но так как метод является методом приближения, то при уменьшении погрешности значения будут все больше подходить друг другу.

Пример 5

Ввод:

2

1 0 5

0 1 7

0.0001

Вывод программы:

5.0

7.0

Тривиальная матрица

Вывод

В ходе выполнения лабораторной работы был реализован алгоритм решения СЛАУ методом Гаусса – Зейделя, который успешно справляется с матрицами диагонально доминирующего вида.

Обращая внимание на примеры запуска программы и полученные результаты, можно сделать вывод, что алгоритм успешно справляется с решением поставленной задачей, также имеет возможность привести матрицу к диагонально доминирующему виду и продолжить вычисления. Следует отметить, что корректно обрабатываются случаи, когда метод не применим, выводя соответствующее сообщение.

Если сравнивать метод Гаусса – Зейделя с методом простых итераций, то можно выделить несколько особенностей:

1. Поскольку используются свежеполученные значения неизвестных в одной итерации, то тем самым увеличивается скорость сходимости метода.
2. Выбор начальных элементов идентичен, и чем выбранные элементы ближе к результирующим значениям, тем выше скорость сходимости.
3. Данный метод сложнее параллелизуется, в силу первого пункта: все уравнения зависят друг от друга в пределах одной итерации.

Алгоритмическая сложность напрямую зависит от числа итераций, при количестве итераций l: $O(l * n^2)$

Анализ численной ошибки: точность вычислений легко регулируется вводимым значением ε, при ее уменьшении – количество итераций метода будет больше, из чего следует, что точность будет выше, а ошибка – меньше.