Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №1 по дисциплине «Вычислительная математика»

"Решение системы линейных алгебраических уравнений СЛАУ"

Выполнил: Дау Конг Туан Ань

Группа: P32151

Преподаватель: Машина Е.А

г. Санкт-Петербург

2023

Цель работы:

Научиться искать решение СЛАУ при помощи численных методов, написать программу, которая будет совершать приближенные вычисления и находить решение, получая на вход матрицу из файла или консоли.

Задание лабораторной работы:

Вариант 9 => метода Гаусса-Зейделя

1. № варианта определяется как номер в списке группы согласно ИСУ. (**9**)
2. В программе численный метод должен быть реализован в виде отдельной подпрограммы или класса, в который входные/выходные данные передаются в качестве параметров.
3. Размерность матрицы n <=20 (задается из файла или с клавиатуры - по выбору конечного пользователя).
4. Должна быть реализована возможность ввода коэффициентов матрицы, как с клавиатуры, так и из файла (по выбору конечного пользователя).

Для итерационных методов 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: x1, x2, …, xn
- Вывод количества итераций, за которое было найдено решение.
- Вывод вектора погрешностей: |xi(k)-xik-1|

Описание метода, расчетные формулы:

Метод Гаусса-Зейделя является итерационным методом и находит конечные значение переменных последовательным приближением после каждой итерации. Так, имея в исходную матрицу A*X = B строится матрица с вынесенными переменными в каждой строчке (так в первой строке $x_1 = f(x_2, x_3 \ldots) + b_1/a_{11}$, для второй строчки $x_2 = f(x_1, x_3, \ldots) + b_2/a_{22}$ и т.д.)

Отсюда конечные формулы для итераций:

- Начальное приближение берется как свободные коэффициенты правой части, то есть
  $x_1^0 = d_1 = b_1 / a_{11}$,

$x_2^0 = d_2 = b_2 / a_{22}$,

…

- На каждой итерации вычисляются новые значения для всех переменных x_1, x_2 … методом подстановки в уравнение для соответствующей переменной вместо других переменных их последние значения (так для x_1^1 будут использоваться значения x_2^0, x_3^0, … x_n^0, когда как для вычисления x_2^1 вместо x_1^0 будет подставляться x_1^1 найденное на прошлом шаге этой итерации и так далее – это и есть отличие данного метода от метода простых итераций).

Тогда приближения к решению системы методом Зейделя определяются следующей системой равенств:

$$x_1^{(k+1)} = c_{11}x_1^{(k)} + c_{12}x_2^{(k)} + \cdots + c_{1n}x_n^{(k)} + d_1$$
$$x_2^{(k+1)} = c_{21}x_1^{(k+1)} + c_{22}x_2^{(k)} + \cdots + c_{2n}x_n^{(k)} + d_2$$
$$x_3^{(k+1)} = c_{31}x_1^{(k+1)} + c_{32}x_2^{(k+1)} + c_{33}x_3^{(k)} \ldots + c_{3n}x_n^{(k)} + d_3$$

……………………………………………………………………

$$x_n^{(k+1)} = c_{n1}x_1^{(k+1)} + c_{n2}x_2^{(k+1)} + \cdots + c_{n\,n-1}x_{n-1}^{(k+1)} + c_{nn}x_n^{(k)} + d_n$$

Рабочая формула метода Гаусса-Зейделя:

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

- Сам итерационный процесс продолжается до тех пор, пока значения разниц вычисленных переменных от их значений на прошлом шаге не будет меньше, чем заданное заранее значение погрешности ε

$$|x_1^{(k)} - x_1^{(k-1)}| \le \varepsilon, \quad |x_2^{(k)} - x_2^{(k-1)}| \le \varepsilon, \quad |x_3^{(k)} - x_3^{(k-1)}| \le \varepsilon$$

Программа (часть, реализующая вычисления):

```
public static double[][] seidelMethod(Matrix matrix, double epsilon) {
    rearrangeMatrix(matrix);
    System.out.println("Matrix after rearrange: ");
    matrix.printMatrix();
    int dimension = matrix.getDimension();
    Matrix copyMatrix = matrix.cloneMatrix();
    copyMatrix.toTriangular(0);
```

```
    System.out.println("Triangular matrix: ");
    copyMatrix.printMatrix();
    if(copyMatrix.getDeterminate() == 0) {
        return null;
    } else {
        System.out.println("Determinate of matrix: " +
copyMatrix.getDeterminate());
    }
    double[][] C = new double[dimension][dimension];
    double[] D = new double[dimension];
    double[][] matrix_1 = matrix.getMatrix();
     for(int i = 0 ;i < dimension; ++i) {
        for(int j = 0 ; j < dimension; ++j) {
            if(i != j) {
                C[i][j] = -matrix_1[i][j] /matrix_1[i][i];
            }
        }
        D[i] = matrix_1[i][dimension] /matrix_1[i][i];
        C[i][i] = 0;
    }
     System.out.println("C matrix: ");
     for(int i = 0 ;i < dimension; ++i) {
         for(int j = 0 ; j < dimension; ++j) {
             System.out.print(C[i][j] + " ");
         }

         System.out.println();
     }

     System.out.println("D vector: ");
     for(int i = 0 ; i < dimension; ++i) {
         System.out.print(D[i] + " ");
     }
     System.out.println();
     double[] newX = new double[dimension];
     double[] oldX = new double[dimension];
     for(int i = 0 ;i < dimension; ++i) {
         newX[i] = D[i];
         oldX[i] = D[i];
     }

    for(int i = 0 ;i < dimension; ++i) {
        newX[i] = 0;
        for(int j = 0 ;j < i; ++j) {
            newX[i] += C[i][j]* newX[j];
        }

        for(int j = i + 1; j < dimension; ++j) {
            newX[i] += C[i][j] * newX[j];
        }

        newX[i] += D[i];
    }
    while (!checkIfLessThanEpsilon(dimension, newX, oldX, epsilon)){
         for(int i = 0 ;i < dimension; ++i) {
             oldX[i] = newX[i];
         }
```

```
        for(int i = 0 ;i < dimension; ++i) {
            newX[i] = 0;
            for(int j = 0 ;j < i; ++j) {
                newX[i] += C[i][j]* newX[j];
            }

            for(int j = i + 1; j < dimension; ++j) {
                newX[i] += C[i][j] * newX[j];
            }

           newX[i] += D[i];
            newX[i] = ((double)((int)(newX[i] * 100))) / 100;
        }
       System.out.println();
        for(int i = 0 ;i < dimension; ++i) {
            System.out.print(newX[i] + " ");
        }
       System.out.println();
    };

    double[][] result = new double[2][dimension];
    for(int i = 0 ; i < dimension; ++i) {
        result[0][i] = newX[i];
        result[1][i] = Math.abs(newX[i] - D[i]);
    }

    return result;
}
```

*Epsilon: default equals to 0.1

Link github: [Click here]

Примеры и результат работы программы:

```
 Do you want to get data from console(1) or file(2) ?
2
Matrix after rearrange:
Print matrix:
9.0 8.0 5.0 14.0
1.0 3.0 5.0 12.0
1.0 3.0 6.0 13.0
Triangular matrix:
Print matrix:
9.0 8.0 5.0 14.0
0.0 2.111111111111111 4.444444444444445 10.444444444444445
0.0 0.0 1.0 1.0
Determinate of matrix: 19.0
C matrix:
0.0 -0.8888888888888888 -0.5555555555555556
-0.3333333333333333 0.0 -1.6666666666666667
-0.16666666666666666 -0.5 0.0
D vector:
1.5555555555555556 4.0 2.1666666666666665
Solution set of equations: -1.52 2.85 0.99
Residual vector :3.0755555555555556 1.15 1.1766666666666665
System finished!

 Do you want to get data from console(1) or file(2) ?
1
Please type dimension of matrix: 2
Please type matrix:
1 2 3
2 5 8
Triangular matrix:
Print matrix:
1.0 2.0 3.0
0.0 1.0 2.0
Determinate of matrix: 1.0
C matrix:
0.0 -2.0
-0.4 0.0
D vector:
3.0 1.6
Solution set of equations: -0.968 1.987
Residual vector :3.968 0.387
System finished!

Process finished with exit code 0
```

Выводы по работе:

При помощи вычислительных устройств можно вычислять приближённые решения различных математических задач различными способами, которые сам человек вычислял бы достаточно долго. Реализован метод Гаусса-Зейделя для решения СЛАУ.