ИТМО

ПИИКТ

Лабораторная работа №1

“Вычислительная математика”

Группа P3201

Метод простых итераций

Выполнил: Братчиков Иван Станиславович

Приняла: Перл Ольга Вячеславовна

Санкт-Петербург

2020

Описание метода:

Метод простых итераций – один из приближенных численных методов.

Пусть дана линейная система

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

Рассмотрим матрицы:

$$A = \begin{bmatrix} a_{11}\ a_{12} \dots\ a_{1n} \\ a_{21}\ a_{22} \dots\ a_{2n} \\ \dots\dots\dots\dots\dots\dots \\ a_{n1}\ a_{n2} \dots\ a_{nn} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_n \end{bmatrix}$$

Тогда систему можно записать в виде уравнения

$$\mathrm{Ax} = \mathrm{b}$$

Разрешим первое уравнение системы относительно x_1, второе – относительно x_2 и тд:

$$\begin{cases} x_1 = \beta_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ x_2 = \beta_2 + a_{21}x_1 + \cdots + a_{2n}x_n \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_n = \beta_n + a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn-1}x_{n-1} \end{cases}$$

Где

$$\beta_i = \frac{b_i}{a_{ii}}; \quad a_{ij} = \frac{a_{ij}}{a_{ii}} \quad \text{при } i \neq j$$

$$a_{ij} = 0 \text{ при } i = j \quad (i, j = 1,2,3, \dots, n)$$

Получаются матрицы

$$\alpha = \begin{bmatrix} a_{11}\ a_{12} \dots\ a_{1n} \\ a_{21}\ a_{22} \dots\ a_{2n} \\ \dots\dots\dots\dots\dots\dots \\ a_{n1}\ a_{n2} \dots\ a_{nn} \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ . \\ . \\ . \\ \beta_n \end{bmatrix}$$

Тогда систему можно записать в виде уравнения

$$x = \beta + \alpha x$$

Будем решать систему методом последовательных приближений. За нулевое приближение можно взять произвольные числа

$$x^{(1)} = \beta + \alpha x^{(0)}$$

$$x^{(2)} = \beta + \alpha x^{(1)}$$

И так далее

$$x^{(k+1)} = \beta + \alpha x^{(k)}$$

Если последовательность приближений имеет предел $x = \lim\limits_{k\to\infty} x^{(k)}$, то этот предел является решением системы

$$\lim_{k\to\infty} x^{(k+1)} = \beta + \alpha \lim_{k\to\infty} x^{(k)}$$

$$x = \beta + \alpha x$$

При этом процесс итерации сводится к единственному решению этой системы только при выполнении по меньшей мере одного из условий

$$\sum_{j=1}^{n} |a_{ij}| < 1 \ (i = 1,2, \dots, n)$$

Или

$$\sum_{i=1}^{n} |a_{ij}| < 1 \ (j = 1,2, \dots, n)$$

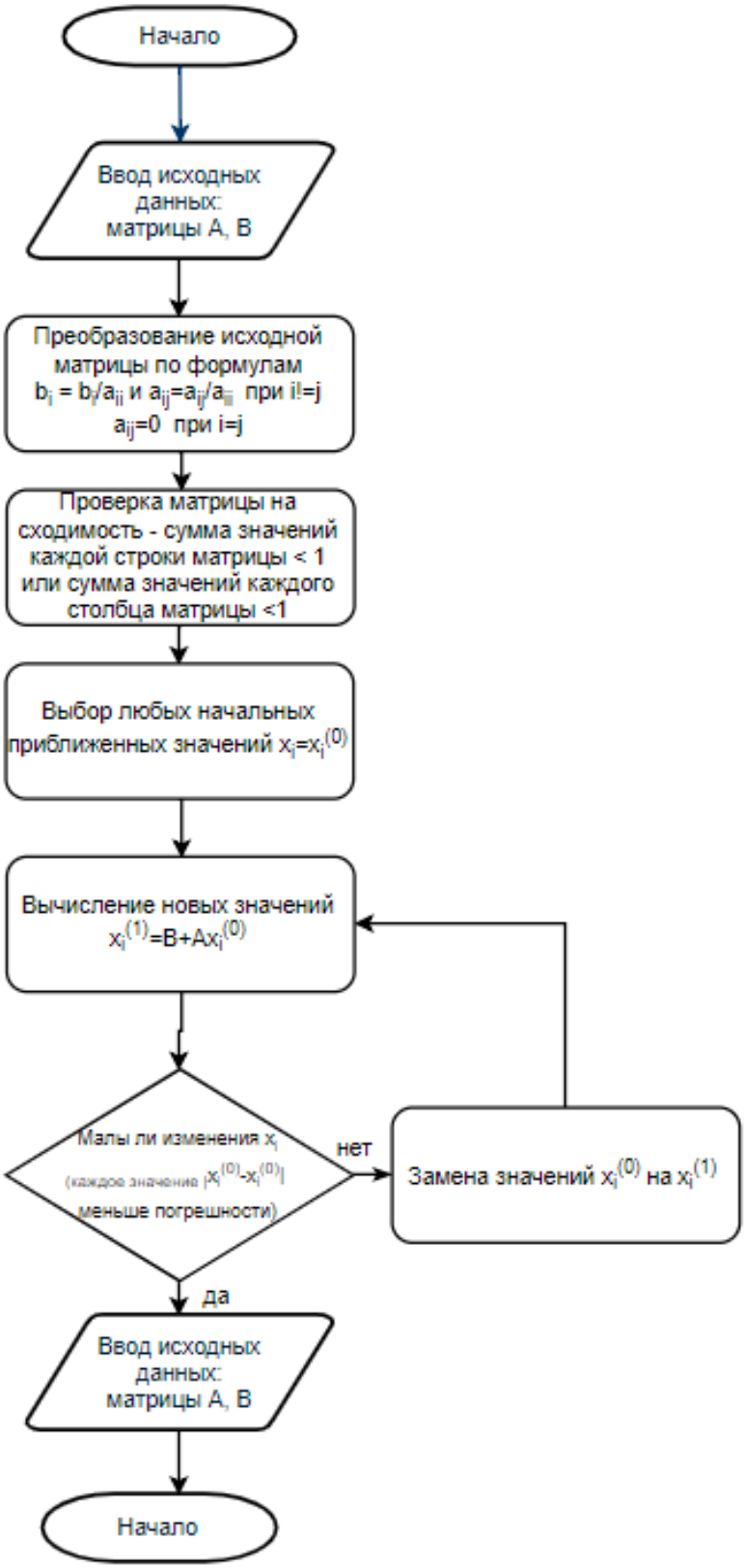
IterationMatrix.java

```java
public class IterationMatrix {
    private double[][] matrix;
    private double maxDeviation = 0;
    private int iterations = 1;
    private double[] approximation;
    private double[] previousApproximation;

    public IterationMatrix(double[][] initialMatrix) {
        this.matrix = initialMatrix;
    }

    /*
     * Applies the formula Cij = -Aij/Aii (if i != j) or Cij = 0 (if i == j)
     */
    public void transformMatrixToXFormed() {
        // Loop through every row in the array
        for (int i = 0; i < matrix.length; i++) {
            double Aii = matrix[i][i];
            // Loop through every element in the row
            for (int j = 0; j < matrix[i].length; j++) {
                if (j != i) {
                    if (matrix[i].length - 1 != j)
                        matrix[i][j] = matrix[i][j] / -Aii;
                    else
                        matrix[i][j] = matrix[i][j] / Aii;
                } else
                    matrix[i][j] = 0;
            }
        }
    }
    /*
     * Returns an array of constants terms of the matrix a.k.a Bn.
     */
    public double[] getConstantTermsVector() {
        double[] constants = new double[matrix.length];
        for (int i = 0; i < matrix.length; i++) {
            constants[i] = matrix[i][matrix[i].length-1];
```

```
    }
    return constants;
  }

  /*
   * Computes the X values (aka approximation) on the basis of the previously computed approximation. Also, sets the maximum deviation.
   */
  public double[] computeXUsingPreviousApproximation(final double[] previousApproximation) {
    double[] answer = new double[matrix.length];
    this.maxDeviation = 0;
    for (int i = 0; i < matrix.length; i++) {
      answer[i] = 0;
      //compute the Xk terms values
      for (int j = 0; j < matrix[i].length-1; j++) {
        answer[i] += matrix[i][j] * previousApproximation[j];
      }
      //compute the final Xk value of the row
      answer[i] += matrix[i][matrix[i].length-1];

      //Search for the absolute deviation criteria
      double deviation = Math.abs(answer[i] - previousApproximation[i]);
      if (deviation > this.maxDeviation)
        this.maxDeviation = deviation;
    }
    return answer;
  }

  public void iterateToTheGivenEpsilon() {
    // if it's the first iteration then use the Constant Terms Vector as an approximation
    if (iterations == 1) {
      approximation = computeXUsingPreviousApproximation(getConstantTermsVector());
      previousApproximation = approximation;
    } else {
      // otherwise use the previously computed approximation
      previousApproximation = approximation;
      approximation = computeXUsingPreviousApproximation(approximation);
    }
    iterations++;
  }

  /**
   * Check whether the matrix is Diagonally Dominant, if not makes it so.
   */
  public boolean transformToDominant(int r, boolean[] V, int[] R) {
    int n = matrix.length;
    // if moved all of the rows then change initial matrix
    if (r == matrix.length) {
      double[][] T = new double[n][n + 1];
      for (int i = 0; i < R.length; i++) {
        for (int j = 0; j < n + 1; j++)
          T[i][j] = matrix[R[i]][j];
      }
```

```
            matrix = T;
            return true;
        }
        //use recursion to move through all of the rows and search for the dominant elements
        for (int i = 0; i < n; i++) {
            if (V[i]) continue;

            double sum = 0;

            for (int j = 0; j < n; j++)
                sum += Math.abs(matrix[i][j]);

            if (2 * Math.abs(matrix[i][r]) > sum) {
                V[i] = true;
                R[r] = i;
                if (transformToDominant(r + 1, V, R))
                    return true;
                V[i] = false;
            }
        }
        return false;
    }

    /**
     * Check whether the matrix is Diagonally Dominant, if not makes it so.
     */
    public boolean makeDominant() {
        //boolean array for highlighting moved rows
        boolean[] visited = new boolean[matrix.length];
        int[] rows = new int[matrix.length];

        Arrays.fill(visited, false);

        return transformToDominant(0, visited, rows);
    }

    public double getMaxDeviation() {
        return maxDeviation;
    }
    public int getIterations() {
        return iterations;
    }
    public double[] getApproximation() {
        return approximation;
    }
    public double[] getPreviousApproximation() {
        return previousApproximation;
    }
}
```

Блок схема:


Начало
Ввод исходных данных: матрицы A, B
Преобразование исходной матрицы по формулам
$b_i = b_i/a_{ii}$ и $a_{ij}=a_{ij}/a_{ii}$ при i!=j
$a_{ij}=0$ при i=j
Проверка матрицы на сходимость - сумма значений каждой строки матрицы < 1 или сумма значений каждого столбца матрицы <1
Выбор любых начальных приближенных значений $x_i=x_i^{(0)}$
Вычисление новых значений $x_i^{(1)}=B+Ax_i^{(0)}$
Малы ли изменения x_i (каждое значение $|x_i^{(1)}-x_i^{(0)}|$ меньше погрешности)
нет
Замена значений $x_i^{(0)}$ на $x_i^{(1)}$
да
Ввод исходных данных: матрицы A, B
Начало

Пример:

```
"epsilon": "0.000000001",
"matrix": [
"2 2 10 2 2 14",
"10 1 1 2 2 12",
"2 10 1 2 2 13",
"3 10 2 50 2 3",
"2 1 1 1 20 10"
]
```

Точность – 1.0E-9
Количество итераций: 32
Результат:
x1: 0.9888792527697281
x2: 0.990114891303152
x3: 0.9912132366582859
x4: -0.24958526033103978
x5: 0.31452493120620395
Вектор погрешностей:
x1^(32)-x1^(31): 5.147853254783286E-10
x2^(32)-x2^(31): 6.001125152366171E-10
x3^(32)-x3^(31): 6.995828361056056E-10
x4^(32)-x4^(31): 3.7679981357285897E-10
x5^(32)-x5^(31): 2.6882840398201324E-10

```
"epsilon": "0.0000000000000001",
"matrix": [
"100 1 2 3 4 5",
"6 200 7 8 9 10",
"11 12 300 13 14 15",
"16 17 18 400 19 20",
"21 22 23 24 500 25"
]
```

Точность – 1.0E-16
Количество итераций: 20
Результат:
x1: 0.04575092285414497
x2: 0.043537399584321554
x3: 0.04277650096031975
x4: 0.04239206788588952
x5: 0.042160277355718354
Вектор погрешностей:
x1^(20)-x1^(19): 1.3877787807814457E-17
x2^(20)-x2^(19): 2.0816681711721685E-17
x3^(20)-x3^(19): 2.7755575615628914E-17
x4^(20)-x4^(19): 2.7755575615628914E-17
x5^(20)-x5^(19): 2.7755575615628914E-17

Вывод:

Метод простых итераций подходит для систем любого размера, в том числе при n>200, так как количество вычислений относительно невелико для каждой итерации. Также методом достигается низкая погрешность за счет итераций, по сравнению с прямыми методами, из-за большого количества последовательных вычислений и ограниченности разрядной сетки, в которых, итоговый результат получается неточным. Однако для выполнения метода должны выполняться строгие условия сходимости, поэтому не для всех систем данный метод будет работать. Также преимуществом метода можно считать его понятность и простоту реализации.