Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ

по лабораторной работе №2
«Системы нелинейных уравнений»

по дисциплине «**Вычислительная математика**»

Автор: Пряничников Кирилл Сергеевич

Факультет: ПИиКТ

Группа: P32202

Преподаватель: Перл Ольга Вячеславовна

Санкт-Петербург, 2023

Описание метода касательных

Метод Ньютона (касательных) – итерационный численный метод для решения нелинейных алгебраических уравнений.
Сначала уравнение приводится к виду $f(x) = 0$, затем задаётся приблизительное значение корня x_0 (нулевое приближение) и вычисляется функция в точке нулевого приближения $f(x_0)$. Далее нахождение значения сводится к итерационному процессу вычисления: $x_{n+1} = \frac{f(x_n)}{f'(x_n)}$
Условие сходимости метода: $f(x_n)f''(x_n) > 0$

Описание метода простой итерации

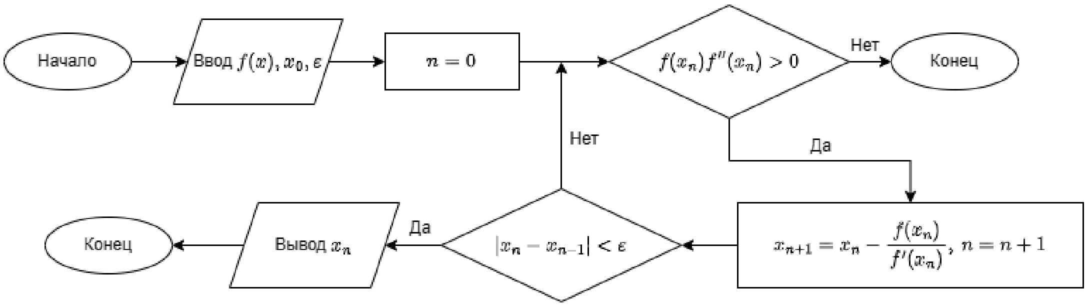
Метод простой итерации – итерационный численный метод для решения нелинейных алгебраических уравнений.
Как и в методе касательных, для метода простой итерации задаётся приблизительное значение корня x_0, затем уравнение $f(x)$ приводится к виду $x = \varphi(x)$. Далее нахождение значения сводится к итерационному процессу вычисления: $x_{n+1} = \varphi(x_n)$
Условие сходимости метода: $|\varphi'(x_n)| < 1$

Описание метода Ньютона для решения СНАУ

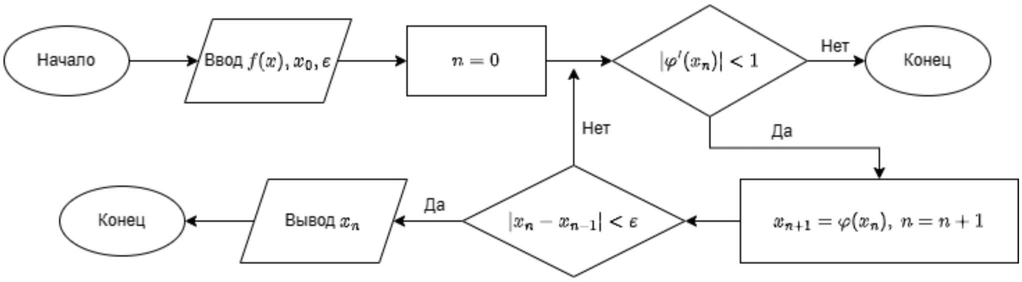
Метод Ньютона – итерационный численный метод для решения систем нелинейных алгебраических уравнений.
Для метода Ньютона задаётся начальное приближение X_0. Далее нахождение значения сводится к итерационному процессу вычисления $X_{n+1} = X_n - J^{-1}(X_n)F(X_n)$, где J – матрица Якоби.
Условие сходимости метода: $|\, J(X_n)\, | \neq 0$

Блок-схемы методов

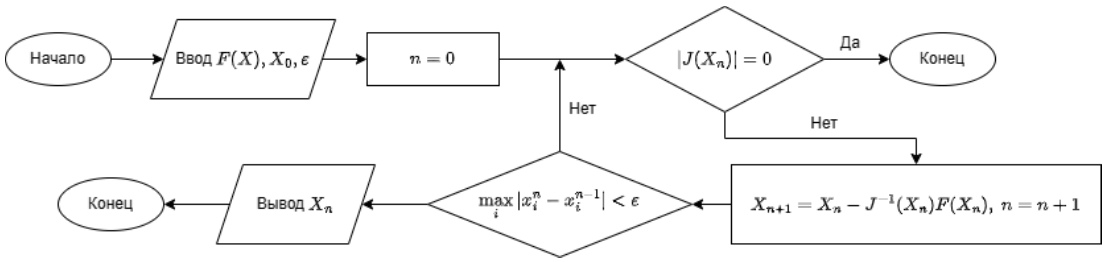
Метод касательных:



Метод простой итерации:



Метод Ньютона:



Функция, реализующая метод касательных на Java

```
public class TangentMethod {
    public static boolean isPossible(double x0) {
        List<Double> results = Equations.functions.apply(x0);
        return results.get(0) * results.get(2) > 0;
    }
    public static double solve(double x0, double epsilon) {
        List<Double> results = Equations.functions.apply(x0);
        double x = x0 - results.get(0) / results.get(1);
        if (Math.abs(x - x0) < epsilon) return x;
        if (isPossible(x)) return solve(x, epsilon);
        throw new UnsupportedOperationException("Метод касательных не сходится");
    }
}
```

Функция, реализующая метод простой итерации на Java

```
public class SimpleIterationMethod {
    public static boolean isPossible(double x0) {
        return Math.abs(Equations.functions.apply(x0).get(4)) < 1;
    }
    public static double solve(double x0, double epsilon) {
        double x = Equations.functions.apply(x0).get(3);
        if (Math.abs(x - x0) < epsilon) return x;
        if (isPossible(x)) return solve(x, epsilon);
        throw new UnsupportedOperationException("Метод простой итерации не сходится");
    }
}
```

Функция, реализующая метод Ньютона на Java

```
public class NewtonMethod {

    public static Double difference = 1.;
    public static List<Double> solve(int system_id, List<Function<List<Double>,
Double>> functions, List<Double> initial_approximations, double epsilon) {
        List<Double> previousApproximations = initial_approximations;
        List<Double> currentApproximations = setNewIteration(system_id, functions,
previousApproximations);
        while (difference > epsilon) {
            previousApproximations = currentApproximations;
            currentApproximations = setNewIteration(system_id, functions,
previousApproximations);
        }
        return currentApproximations;
    }
    public static List<Double> setNewIteration(int system_id,
List<Function<List<Double>, Double>> functions, List<Double> approximations) {

        Double[][] jacobian = getJacobian(system_id, approximations);

        Double[] B = new Double[functions.size()];
        for (int i = 0; i < functions.size(); i++) {
            B[i] = -functions.get(i).apply(approximations);
        }
        difference = 0.;
        Double[] delta = multiplyTwoMatrices(B, getInversedMatrix(jacobian));

        Double[] result = new Double[approximations.size()];
        for (int i = 0; i < approximations.size(); i++) {
            if (Math.abs(delta[i]) > difference) difference = Math.abs(delta[i]);
            result[i] = delta[i] + approximations.get(i);
        }
        return Arrays.stream(result).toList();
    }
    private static Double getDeterminant(Double[][] matrix) {
```

```
        double result;
        if (matrix.length == 1) {
            result = matrix[0][0];
        } else if (matrix.length == 2) {
            return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
        } else {
            result = 0;
            for (int i = 0; i < matrix[0].length; i++)
                result += Math.pow(-1, i) * matrix[0][i] *
getDeterminant(getSubmatrix(matrix, 0, i));
        }
        return result;
    }

    private static Double[][] getInversedMatrix(Double[][] matrix) {
        int n = matrix.length;
        Double[][] result = new Double[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                result[i][j] = Math.pow(-1, i + j) *
getDeterminant(getSubmatrix(matrix, i, j));
            }
        }
        if (getDeterminant(matrix) == 0) throw new
UnsupportedOperationException("Метод Ньютона не сходится в данной точке");
        double determinant = 1. / getDeterminant(matrix);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j <= i; j++) {
                double temp = result[i][j];
                result[i][j] = result[j][i] * determinant;
                result[j][i] = temp * determinant;
            }
        }
        return result;
    }

    private static Double[][] getSubmatrix(Double[][] matrix, int row, int column) {
        int n = matrix.length;
        Double[][] result = new Double[n - 1][n - 1];
        for (int i = 0; i < n; i++) {
            if (i == row) continue;
            for (int j = 0; j < n; j++)
                if (j != column)
                    result[i < row ? i : i - 1][j < column ? j : j - 1] =
matrix[i][j];
        }
        return result;
    }
    private static Double[] multiplyTwoMatrices(Double[] a, Double[][] b) {
        int n = a.length;
        Double[] result = new Double[n];
        for (int i = 0; i < n; i++) {
            result[i] = 0.;
            for (int k = 0; k < n; ++k)
                result[i] += a[k] * b[i][k];
        }
        for (int i = 0; i < n; i++) {
            result[i] = result[i];
        }
        return result;
    }
    public static Double[][] getJacobian(int system_id, List<Double> approximations) {
        Double[][] result = new Double[approximations.size()][approximations.size()];
        for (int variableIndex = 0; variableIndex < approximations.size();
variableIndex++) {
            result[variableIndex] = getJacobianRow(system_id, approximations,
variableIndex);
        }
        Double[][] t_result = new
Double[approximations.size()][approximations.size()];
        for (int i = 0; i < result.length; i++) {
```

```
            for (int j = 0; j < result.length; j++) {
                t_result[i][j] = result[j][i];
            }
        }
        return t_result;
    }
    private static Double[] getJacobianRow(int system_id, List<Double> approximations, 
int variableIndex) {
        Double[] result = new Double[approximations.size()];
        switch (system_id) {
            case 1: {
                result[0] = variableIndex == 0 ? 
Math.cos(approximations.get(variableIndex)) : 0.;
                result[1] = approximations.get(variableIndex == 0 ? 1 : 0) / 2;
                break;
            }
            case 2: {
                result[0] = variableIndex == 0 ? approximations.get(1) / 
Math.pow(Math.cos(approximations.get(0) * approximations.get(1) + 0.4), 2) - 2 * 
approximations.get(0)
                        : approximations.get(0) / 
Math.pow(Math.cos(approximations.get(0) * approximations.get(1) + 0.4), 2);
                result[1] = variableIndex == 0 ? 1.8 * approximations.get(0) : 4 * 
approximations.get(1);
                break;
            }
            case 3: {
                result[0] = variableIndex == 0 ? approximations.get(1) / 
Math.pow(Math.cos(approximations.get(0) * approximations.get(1)), 2) - 2 * 
approximations.get(0)
                        : approximations.get(0) / 
Math.pow(Math.cos(approximations.get(0) * approximations.get(1)), 2);
                result[1] = variableIndex == 0 ?  approximations.get(0) : 4 * 
approximations.get(1);
                break;
            }
            case 4: {
                switch (variableIndex) {
                    case 0: {
                        result[0] = 2 * approximations.get(0);
                        result[1] = 4 * approximations.get(0);
                        result[2] = 6 * approximations.get(0);
                        break;
                    }
                    case 1: {
                        result[0] = 2 * approximations.get(1);
                        result[1] = 2 * approximations.get(1);
                        result[2] = -4.;
                        break;
                    }
                    case 2: {
                        result[0] = 2 * approximations.get(2);
                        result[1] = -4.;
                        result[2] = 2 * approximations.get(2);
                        break;
                    }
                }
                break;
            }
        }
        return result;
    }
}
```

Примеры работы программы

Пример 1:

Для функции $e^x - x = 0$

Введите x0:
0.5
Введите предельную погрешность:
0.00001
Значение, полученное методом касательных: 0.5671432904097811
Значение, полученное методом простой итерации: 0.5671407632698067

Пример 2:

Для функции $\ln(x) + x^3 - 2^x = 0$

Введите x0:
-1
Введите предельную погрешность:
0.00001
Метод касательных не применим для начального приближения
Метод простой итерации не применим для начального приближения

Пример 3:

Для системы уравнений $\tan(xy + 0.4) - x^2 = 0,\ 0.9x^2 + 2y^2 - 1 = 0$

Введите 2 значения для начального приближения
1 2
Введите предельную погрешность:
0.00001
Ответы:
    x1 = 0.9281399820811073
    x2 = 0.33518693015706447

Вывод

Методы касательных и простой итерации дают относительно быстрое и точное решение. Однако из-за того, что для обоих методов достаточно сложно подобрать начальное приближение из-за строгости их условий сходимости, для получения приближённого значения следует использовать методы половинного деления и хорд, а методы касательных и простой итерации – для того, чтобы сделать уже известное значение более точным.
Метод Ньютона для решения систем нелинейных алгебраических уравнений даёт точные значения в рамках данного приближения за достаточно короткий срок, что позволяет его использовать для большого диапазона задач.