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 № 2**

по дисциплине «Вычислительная математика»

Системы нелинейных уравнений

Выполнил:

Гурьянов Кирилл Алексеевич

Группа: P32302

Преподаватель:

Перл Ольга Вячеславовна

Санкт-Петербург

2023

## 1. Описание методов, расчетные формулы

Метод деления пополам — численный метод для решения нелинейных уравнений. Он основан на теореме о промежуточных значениях, которая гласит, что если функция $f(x)$ непрерывна на отрезке [a, b] и принимает значения $f(a)$ и $f(b)$ с разными знаками, то существует точка $c \in (a, b)$, такая что $f(c) = 0$. Т.е. необходимым условием является: $f(a) * f(b) < 0$.

Метод заключается в следующем: на каждой итерации метода отрезок [a, b] делится пополам, и находится точка $c$, которая является серединой отрезка. Затем вычисляются значения функции $f(c)$ и $f(a)$ (или $f(b)$), и на основе знаков этих значений определяется, на каком из подотрезков [a, c] или [c, b] находится корень. Процесс продолжается до тех пор, пока не будет достигнута заданная точность или не будет найдено решение.

Метод простой итерации используется для приближенного решения нелинейных уравнений. Для этого исходное уравнение приводится к виду $x = f(x)$, где функция $f(x)$ определяется из исходного уравнения путем переноса всех слагаемых в правую часть. Функция должна быть непрерывной на интервале поиска корня. После этого проверяется условие сходимости, а именно производная функции на интервале [a, b] должна быть меньше 1 по модулю: $f'(a) < 1$ и $f'(b) < 1$. Если эти условия выполнены, то итерационный процесс сходится к корню функции, при этом скорость сходимости зависит от выбора начального приближения.

Затем выбирается начальное приближение $x_0$ и последовательно вычисляются новые значения $x_{i+1} = f(x_i)$ до тех пор, пока разность $|x_{i+1} - x_i|$ не станет меньше заданной точности ε.

В целом алгоритм метода простой итерации выглядит так:

- Привести исходное уравнение к виду $x = f(x)$

- Выбрать начальное приближение $x_0$
- Вычислить новое значение $x_{i+1} = f(x_i)$
- Если $|x_{i+1} - x_i| < \varepsilon$, то завершить итерационный процесс и вернуть $x_{i+1}$ как приближенное решение уравнения. Иначе перейти к шагу 3.

Также необходимо ограничить количество итераций, чтобы избежать бесконечного цикла, если метод не сходится.

Метод Ньютона - это итерационный численный метод решения нелинейных систем уравнений. Для решения системы уравнений методом Ньютона нужно начать с некоторого начального приближения и итеративно вычислять новые приближения до тех пор, пока не будет достигнута заданная точность или не будет достигнуто максимальное количество итераций.

Для подсчета на каждой итерации новых значений xi используется формула:

$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \Rightarrow x_{i+1} = x_i - \frac{f(x_i)}{J(x_i)}$, где $J$ – матрица Якоби

Главная сложность алгоритма заключается в решении СЛАУ:

$J(x_i) * (x_{i+1} - x_i) = - f(x_i)$

Скорость сходимости – квадратичная. Условием окончания выполнения итераций является достижение заданной точности: $|x_i^{k+1} - x_i^k| < \varepsilon$

## 2. Блок-схемы методов

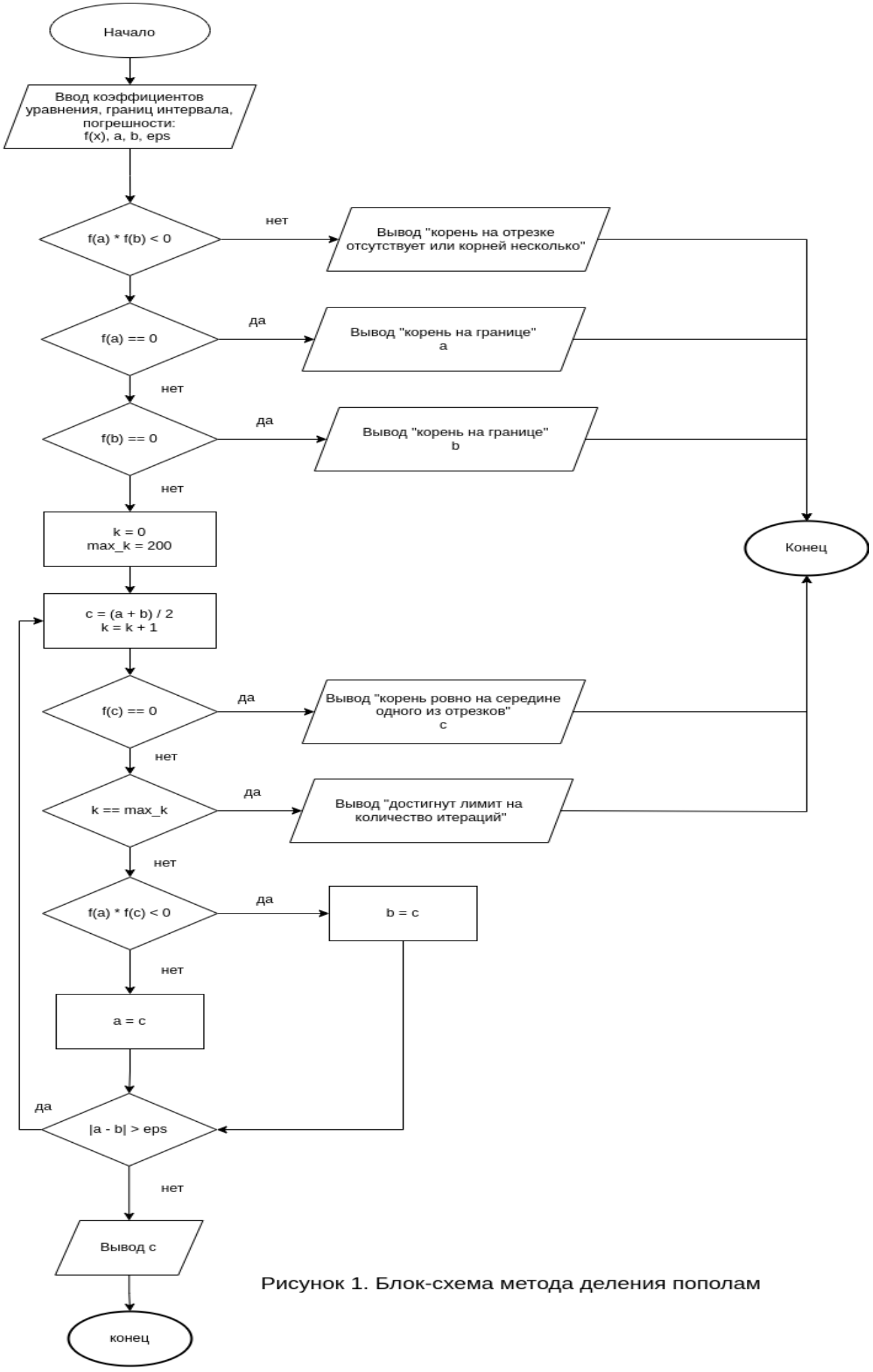


Рисунок 1. Блок-схема метода деления пополам

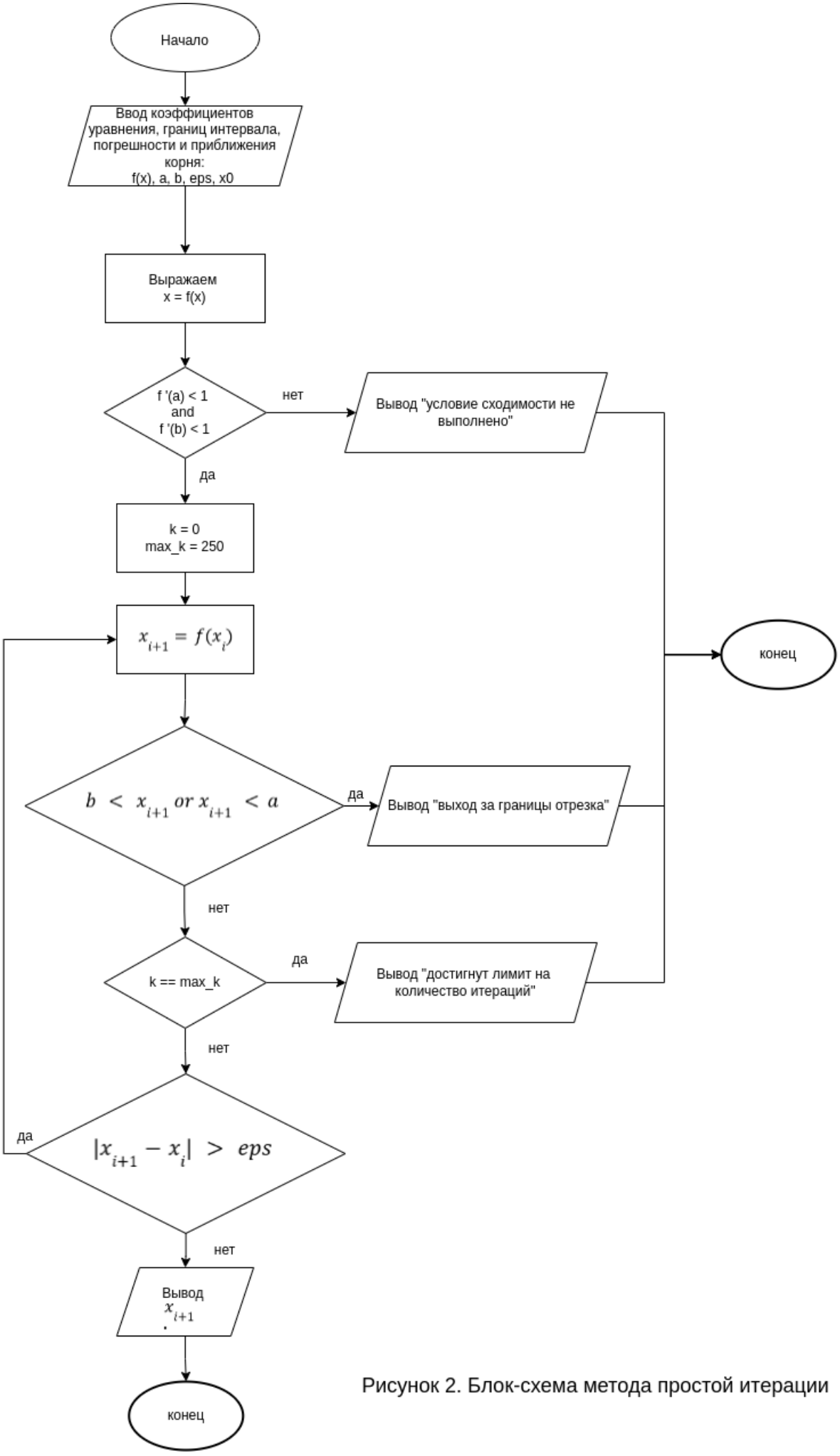


Рисунок 2. Блок-схема метода простой итерации

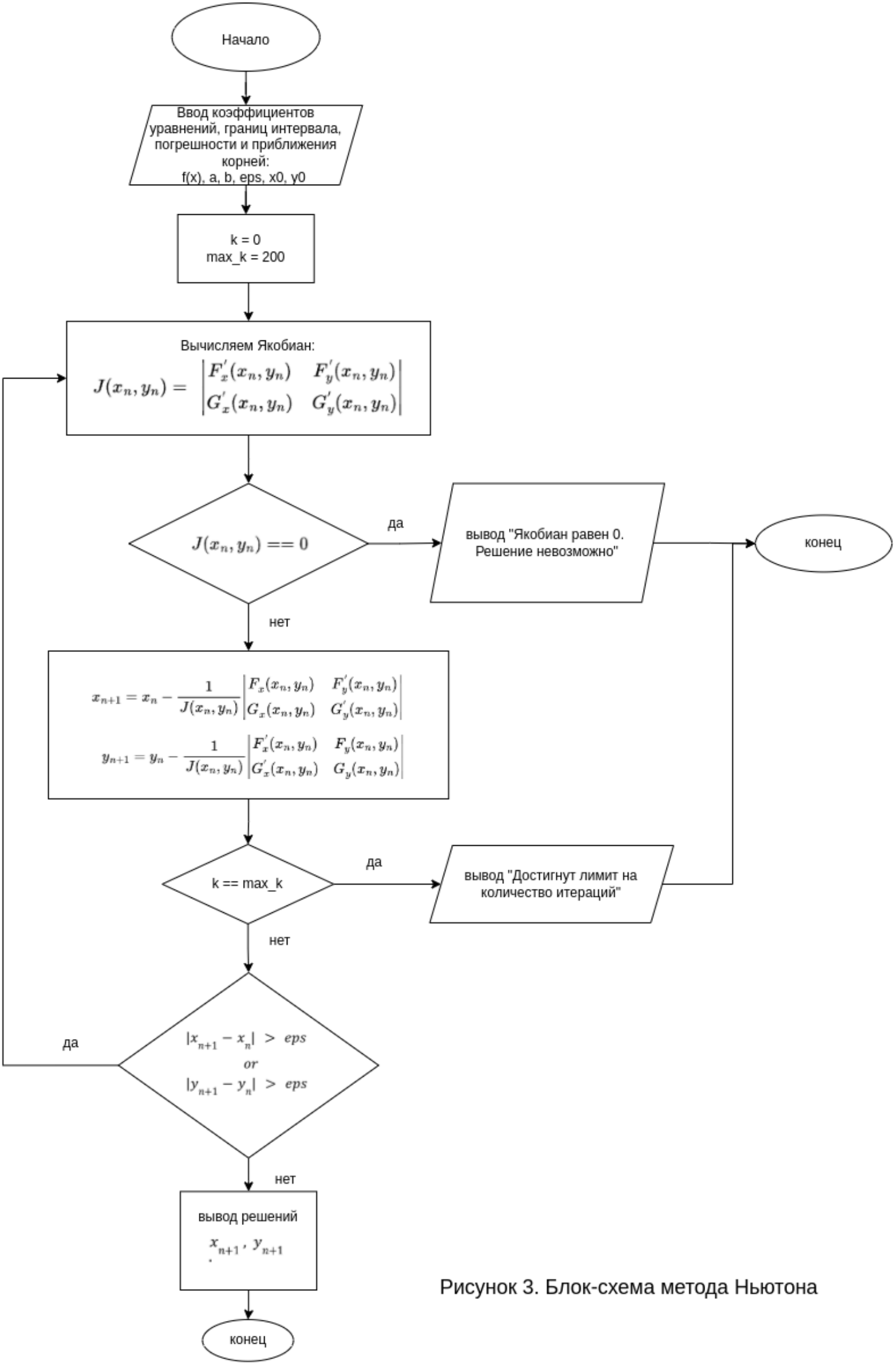


Рисунок 3. Блок-схема метода Ньютона

## 3. Листинг реализованных методов программы

### a. Метод половинного деления

```java
public class BisectionMethod {
    2 usages
    public Answer solveTheEquation(Equation equation, double error, double leftBoundary, double rightBoundary) {
        if (!isRootInsideTheSegment(equation, leftBoundary, rightBoundary))
            return new Answer( answer: null,  error: null,  numberOfIterations: 0,
                    errorMassage: "Корень на отрезке отсутствует или корней на отрезке несколько",  isCorrect: false);
        if (equation.calculateForBisection(leftBoundary) == 0)
            return new Answer(leftBoundary,  error: 0.0d,  numberOfIterations: 0,  errorMassage: "Корень на границе.",  isCorrect: true);
        if (equation.calculateForBisection(rightBoundary) == 0)
            return new Answer(rightBoundary,  error: 0.0d,  numberOfIterations: 0,  errorMassage: "Корень на границе.",  isCorrect: true);

        int MAX_ITERATIONS = 200;
        int numberOfIterations = 0;
        double middle = 0;
        double func;
        double currentError;
        String message = "Все отлично!";
        boolean isCorrect = true;

        do {
            middle = (rightBoundary + leftBoundary) / 2.0d;
            func = equation.calculateForBisection(middle);
            numberOfIterations++;
            currentError = rightBoundary - leftBoundary;

            if (func == 0.0d) {
                currentError = 0.0d;
                message = "Корень попал ровно на середину одного из отрезков";
                break;
            }

            if (numberOfIterations == MAX_ITERATIONS) {
                message = "Достигнут лимит на количество итераций.";
                isCorrect = false;
                break;
            }

            if (isRootInsideTheSegment(equation, leftBoundary, middle)) rightBoundary = middle;
            else leftBoundary = middle;

        } while (currentError > error);
        return new Answer(middle, currentError, numberOfIterations, message, isCorrect);
    }

    2 usages
    private boolean isRootInsideTheSegment(Equation equation, double leftBoundary, double rightBoundary) {
        return equation.calculateForBisection(leftBoundary) * equation.calculateForBisection(rightBoundary) < 0;
    }
}
```

## б. Метод простой итерации

```java
public class SimpleIterationMethod {
    1 usage
    public Answer solveTheEquation(Equation equation, double error, double leftBoundary, double rightBoundary, double approximation) {
        double currentError;
        int MAX_ITERATIONS = 250;
        int numberOfIterations = 0;
        double oldX = approximation;
        double x;
        String message = "Все отлично!";
        boolean isCorrect = true;
        do {
            x = equation.calculateForIterationMethod(oldX);
            currentError = Math.abs(x - oldX);
            numberOfIterations++;
            oldX = x;
            if (leftBoundary > x || x > rightBoundary) {
                message = "Выход за пределы диапазона";
                isCorrect = false;
                break;
            }
            if (numberOfIterations == MAX_ITERATIONS) {
                message = "Достигнут лимит на количество итераций.";
                isCorrect = false;
                break;
            }
        } while (currentError > error);
        if (Double.isNaN(x)) {
            message = "К сожалению, не удалось вычистить корень.";
            isCorrect = false;
        }
        return new Answer(x, currentError, numberOfIterations, message, isCorrect);
    }
}
```

## в. Метод Ньютона

```java
public class NewtonMethod {
    1 usage
    public AnswerFromSystem solveTheSystem(SystemOfEquations system, double[] approximations, double error) throws ArithmeticException {
        int numberOfIterations = 0;
        int MAX_ITERATIONS = 200;
        double x;
        double y;
        double oldX = approximations[0];
        double oldY = approximations[1];
        String message = "Все отлично!";
        boolean isCorrect = true;

        do {
            x = oldX;
            y = oldY;
            numberOfIterations++;
            double[] coefficients = calcCoefficients(x, y, system);
            oldX = x + coefficients[0];
            oldY = y + coefficients[1];

            if (numberOfIterations == MAX_ITERATIONS) {
                message = "Достигнут лимит на количество итераций.";
                isCorrect = false;
                break;
            }
        } while (isErrorsOverThreshold(x, y, oldX, oldY, error));

        if (Double.isNaN(oldX) || Double.isNaN(oldY)) {
            message = "К сожалению, не удалось вычистить корни системы";
            isCorrect = false;
        }

        return new AnswerFromSystem(oldX, oldY, xError: oldX - x, yError: oldY - y, numberOfIterations, message, isCorrect);
    }
```

```java
    private double[] calcCoefficients(double x, double y, SystemOfEquations system) {
        double[] coefficients = new double[2];

        double[] derivativesX = calcDerivativeByX(system, x, y);
        double[] derivativesY = calcDerivativeByY(system, x, y);
        double func1 = system.calculateFirstEquation(x, y);
        double func2 = system.calculateSecondEquation(x, y);

        double det = derivativesY[1] * derivativesX[0] - derivativesX[1] * derivativesY[0];
        if (det != 0) {
            coefficients[0] = -(func1 * derivativesY[1] - func2 * derivativesY[0]) / det;
            coefficients[1] = -(func2 * derivativesX[0] - func1 * derivativesX[1]) / det;
        } else {
            throw new ArithmeticException("Определитель Якобиана равен 0. Решение невозможно.");
        }

        return coefficients;
    }

    1 usage
    private double[] calcDerivativeByX(SystemOfEquations system, double x, double y) {
        double h = 0.000001;
        double[] derivatives = new double[2];
        derivatives[0] = (system.calculateFirstEquation( x: x + h, y) - system.calculateFirstEquation(x, y)) / h;
        derivatives[1] = (system.calculateSecondEquation( x: x + h, y) - system.calculateSecondEquation(x, y)) / h;
        return derivatives;
    }

    1 usage
    private double[] calcDerivativeByY(SystemOfEquations system, double x, double y) {
        double h = 0.000001;
        double[] derivatives = new double[2];
        derivatives[0] = (system.calculateFirstEquation(x,  y: y + h) - system.calculateFirstEquation(x, y)) / h;
        derivatives[1] = (system.calculateSecondEquation(x, y) - system.calculateSecondEquation(x,  y: y + h)) / h;
        return derivatives;
    }
```

```java

    1 usage
    private boolean isErrorsOverThreshold(double x, double y, double oldX, double oldY, double error) {
        return Math.abs(x - oldX) > error || Math.abs(y - oldY) > error;
    }
}
```

## 4. Примеры и результаты работы программы на разных данных

```
Решение систем нелинейных уравнений и систем нелинейных уравнений.
Вы уверены, что оно вам нужно? Если нет, то нажмите сочетание клавиш Ctrl+C
Выберите что вы хотите решить: 0 - нелинейное уравнение, 1 - систему нелинейных уравнений, 2 - выход
0
Выберите тип уравнения, которое хотите решить
ax^3 + bx^2 + cx + d = 0 - введите 0
ax^3 + b * e^{-x} = 0 - введите 1
a * sin(x) + b * x + c - введите 2
0
Введите коэффициенты уравнения через пробел в формате: a b c ...:
1 2 -3 -6
Введите левую границу интервала:
1,5
Введите правую границу интервала:
2
Введите погрешность вычислений от 0,000001 до 0,1:
0,000001
Введите приближение для x:
2
------- Метод простой итерации -------
Корень уравнения: 1.7320505758018325
Все отлично!
------- Метод деления попалам -------
Корень уравнения: 1.7320504188537598
Все отлично!

Разница между методами: 1.569480727603434E-7
```

```
Решение систем нелинейных уравнений и систем нелинейных уравнений.
Вы уверены, что оно вам нужно? Если нет, то нажмите сочетание клавиш Ctrl+C
Выберите что вы хотите решить: 0 - нелинейное уравнение, 1 - систему нелинейных уравнений, 2 - выход
0
Выберите тип уравнения, которое хотите решить
ax^3 + bx^2 + cx + d = 0 - введите 0
ax^3 + b * e^{-x} = 0 - введите 1
a * sin(x) + b * x + c - введите 2
1
Введите коэффициенты уравнения через пробел в формате: a b c ...:
5 -2
Введите левую границу интервала:
0
Введите правую границу интервала:
1
Введите погрешность вычислений от 0,000001 до 0,1:
0,00001
Введите приближение для x:
0,5
------- Метод простой итерации -------
Корень уравнения: 0.6027041236155175
Все отлично!
------- Метод деления попалам -------
Корень уравнения: 0.6027030944824219
Все отлично!

Разница между методами: 1.0291330956313516E-6
```

```
Решение систем нелинейных уравнений и систем нелинейных уравнений.
Вы уверены, что оно вам нужно? Если нет, то нажмите сочетание клавиш Ctrl+C
Выберите что вы хотите решить: 0 - нелинейное уравнение, 1 - систему нелинейных уравнений, 2 - выход
0
Выберите тип уравнения, которое хотите решить
ax^3 + bx^2 + cx + d = 0 - введите 0
ax^3 + b * e^{-x} = 0 - введите 1
a * sin(x) + b * x + c - введите 2
2
Введите коэффициенты уравнения через пробел в формате: a b c ...:
-2 4 10,5
Введите левую границу интервала:
-3
Введите правую границу интервала:
-2
Введите погрешность вычислений от 0,000001 до 0,1:
0,00001
Введите приближение для x:
-2,5
------- Метод простой итерации -------
Корень уравнения: -2.7948947040737564
Все отлично!
------- Метод деления попалам -------
Корень уравнения: -2.7948951721191406
Все отлично!

Разница между методами: 4.680453842276222E-7
```

```
Решение систем нелинейных уравнений и систем нелинейных уравнений.
Вы уверены, что оно вам нужно? Если нет, то нажмите сочетание клавиш Ctrl+C
Выберите что вы хотите решить: 0 - нелинейное уравнение, 1 - систему нелинейных уравнений, 2 - выход
1
Выберите тип системы, которую хотите решить
Введите 0:
a_1 * x + b_1 * y + c_1 = 0
a_2 * x^2 + b_2 * y^2 + c_2 = 0
------------------------------
Введите 1:
a_1 * x + b_1 * lg(x) + c_1 * y^2 + d_1 = 0
a_2 * x^2 + b_2 * x * y + c_2 * x + d_2 = 0
------------------------------
0
Введите коэффициенты первого уравнения через пробел в формате: a b c ...:
4 -3 6
Введите коэффициенты второго уравнения через пробел в формате: a b c ...:
2 3 -5
Введите приближение для x:
-1,5
Введите приближение для y:
0
Введите погрешность вычислений от 0,000001 до 0,1:
0,000001
x = -1.5762306988208812
y = -0.10164093176117492
Количество итераций: 10
Погрешности:
Для x = 5.725781269116936E-7
Для y = 7.634375018966688E-7
Все отлично!
```

## 5. Вывод

Метод половинного деления для решения уравнений основан на теореме о промежуточных значениях, а метод простой итерации для решения основан на принципе последовательной замены. Если попробовать сравнить эти методы по скорости сходимости, то ответ нас ждем весьма неоднозначный. Ответ на этот вопрос зависит от конкретной функции и начального приближения, поэтому нет однозначного ответа на вопрос о том, какой метод быстрее сходится для решения нелинейных уравнений.

Однако в целом можно сказать, что метод простой итерации имеет более быструю сходимость, чем метод половинного деления. Это связано с тем, что метод простой итерации позволяет использовать более точные начальные приближения и более гибко подстраиваться под форму функции, что приводит к более быстрой сходимости. Однако, метод половинного деления имеет гарантированную сходимость для любой монотонной функции с известными значениями на концах интервала, что делает его более надежным в некоторых случаях.

Алгоритмическая сложность метода половинного деления O(log n). Это означает, что время выполнения алгоритма увеличивается логарифмически в зависимости от количества итераций n, необходимых для достижения заданной точности решения.

Метод простой итерации для решения нелинейных уравнений состоит в поочередном приближении к решению путем применения к текущему приближению некоторой итерационной формулы. На каждой итерации выполняется O(1) операций (вычисление значений функции и производной в текущей точке, арифметические операции и т.д.). В таком случае общая алгоритмическая сложность метода простой итерации составляет O(k), где k - количество итераций, необходимых для достижения заданной точности решения.

Метод половинного деления применяется в тех случаях, когда уравнение имеет один корень и функция является монотонной на данном интервале. Также метод применим в тех случаях, когда корень уравнения известен на данном интервале и нужно найти его с

высокой точностью. Метод простой итерации применяется в тех случаях, когда уравнение имеет несколько корней или когда корень находится в окрестности начального приближения.

Таким образом, выбор метода зависит от свойств уравнения, начального приближения и требуемой точности. В некоторых случаях может быть эффективно комбинировать различные методы для достижения наилучшего результата.

Алгоритмическая сложность метода Ньютона для решения систем нелинейных уравнений зависит от размерности системы и скорости сходимости метода. Обычно метод Ньютона имеет квадратичную сходимость, что означает, что на каждой итерации удваивается количество верных цифр в ответе. Следовательно, для достижения заданной точности метод Ньютона требует O(sqrt(N)) итераций для системы из N уравнений. Однако, скорость сходимости может зависеть от выбранной начальной точки, а в некоторых случаях метод может расходиться.

Метод простой итерации следует применять в случаях, когда легко задать итерационную формулу и при этом достаточно быстро достигается сходимость. Этот метод часто используется для решения нелинейных уравнений, когда корни легко определяются из уравнения.

Метод Ньютона следует применять в случаях, когда сложно или невозможно задать итерационную формулу, но есть возможность вычислить производную функции. Этот метод может сходиться к решению гораздо быстрее, чем метод простой итерации, но может быть менее устойчивым и может не сходиться при некоторых значениях начальных условий. Этот метод часто используется для решения систем нелинейных уравнений.