Национальный исследовательский университет ИТМО

Факультет ПИиКТ

Лабораторная работа №2

по вычислительной математике

Работу выполнила:

Тройникова Вероника

Группа: Р3233

Преподаватель:

Перл О. В.

Санкт-Петербург

2022

1. Описание методов, расчетные формулы

Метод хорд является итерационным методом (неизвестная x ищется в течение некоторого числа итераций); используется для нахождения корней нелинейных уравнений. Данный метод основан на постепенном "сужении" заданного изначально интервала [a, b]. На каждой итерации находится точка пересечения хорды с осью Ox и выбирается один из двух получившихся интервалов. Чем больше произведено итераций, тем более точный ответ получится в итоге.

Для применения данного метода выбираются первоначальный отрезок [a, b] и погрешность вычислений. Условием использования метода является выбор верного интервала локализации:

$$f(a) * f(b) < 0$$

Для подсчета на каждой итерации нового значения x_i используется следующая формула (она получается из уравнения прямой, проходящей через две заданные точки):

$$\frac{x-a}{b-a} = \frac{y-f(a)}{f(b)-f(a)} \quad => \quad x = a - \frac{f(a)}{f(b)-f(a)} \cdot (b-a)$$

После подсчета нового значения x_i выбирается один из двух интервалов, который будет содержать искомый корень: [a, x_i] или [x_i, b] (выбор делается в пользу отрезка, значения функции на концах которого имеют противоположные знаки).

Скорость сходимости – линейная (но быстрее, чем метод половинного деления).

Условием окончания выполнения итераций является достижение заданной точности: $|x_i - x_{i-1}| \leq eps$

Метод касательных также является итерационным методом и используется для нахождения корней нелинейных уравнений (второе название – метод Ньютона). На каждой итерации находится точка пересечения касательной с осью Ox и эта точка становится следующим приближением x_i (касательная "строится" в точке x_{i-1}). Чем больше произведено итераций, тем более точный ответ получится в итоге.

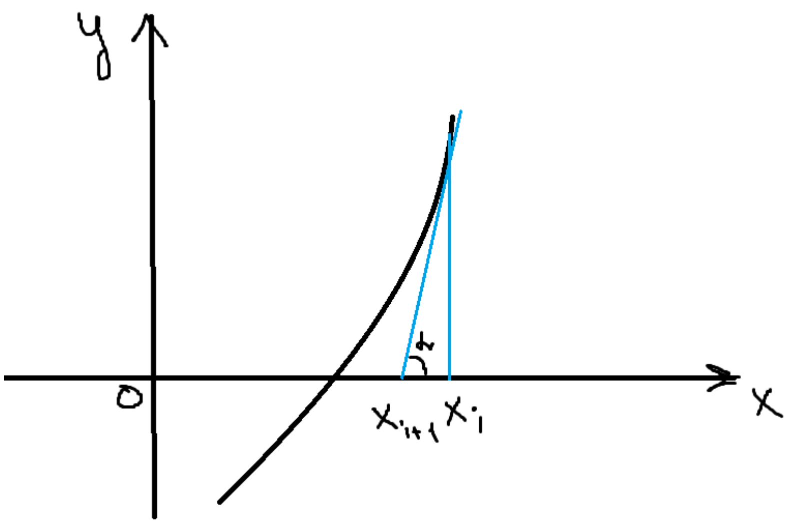
Для применения данного метода выбираются первоначальный отрезок [a, b] и погрешность вычислений. Начальное приближение x_0 выбирается из отрезка [a, b] по следующему условию (оно обеспечивает быструю сходимость):

$$f(x_0) * f''(x_0) > 0$$

Для подсчета на каждой итерации нового значения x_i используется следующая формула:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Вывод формулы:



В прямоугольном треугольнике имеем следующее соотношение:

$\text{tg}\,\alpha = \frac{f(x)}{x_i - x_{i+1}}$ => $x_{i+1} = x_i - \frac{f(x)}{tg\alpha}$ => $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$

Скорость сходимости – квадратичная (то есть данный метод быстрее метода хорд и метода половинного деления).

Если на i-ой итерации x_i вышел за пределы интервала [a, b], то следует считать метод касательных непрактичным для применения при заданных начальных условиях.

Условием окончания выполнения итераций является достижение заданной точности: $|x_i - x_{i-1}| \le eps$

Метод Ньютона – это метод для решения систем нелинейных уравнений, является обобщением метода касательных.

Для применения данного метода выбираются первоначальный приближения $\{x_i\}$ и погрешность вычислений. Чем больше произведено итераций, тем более точный ответ получится в итоге.

Для подсчета на каждой итерации новых значений x_i используется формула, полученная обобщением формулы для метода касательных:

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

Основная 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: $|x_i^{(k+1)} - x_i^{(k)}| \le eps$

2. Блок-схема численного метода

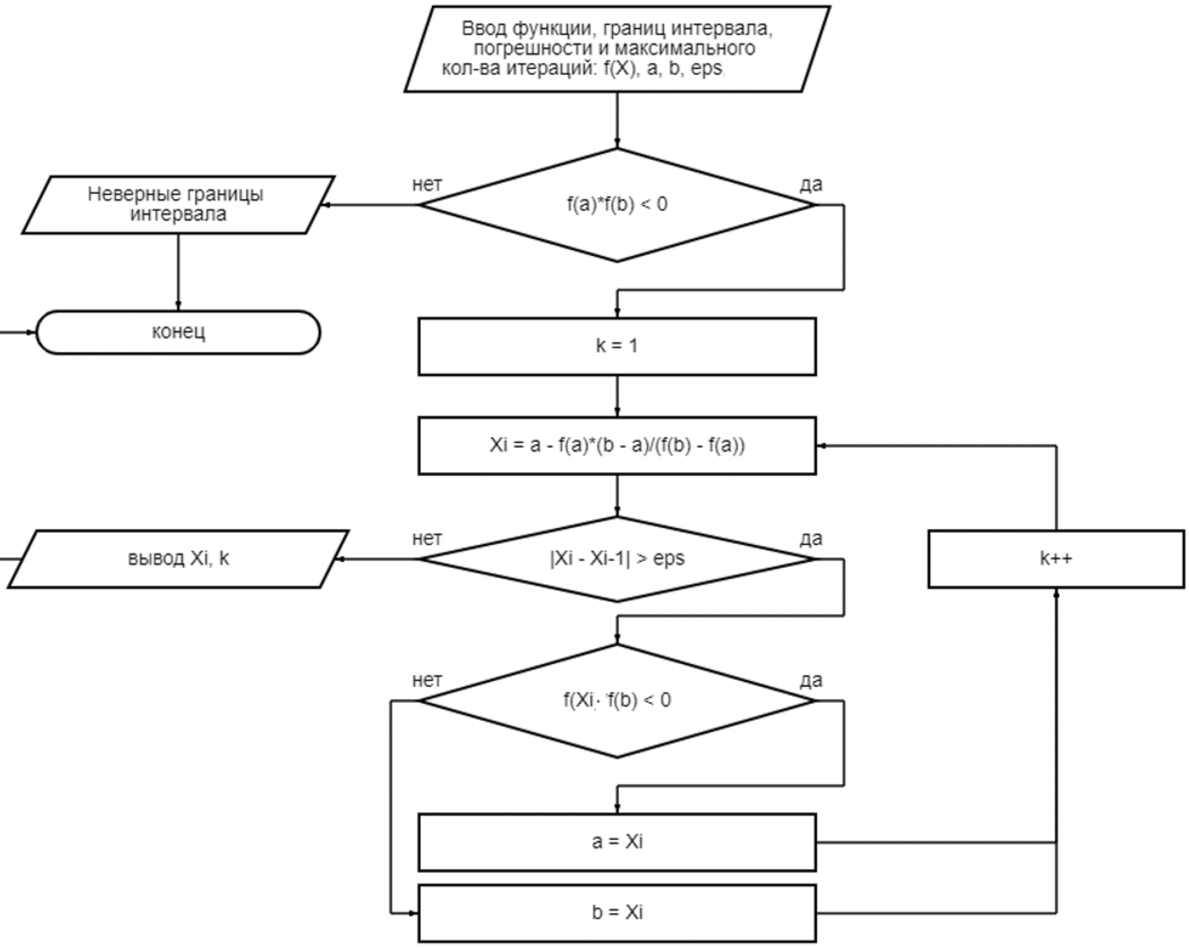


Рисунок 1 “Блок-схема метода хорд”

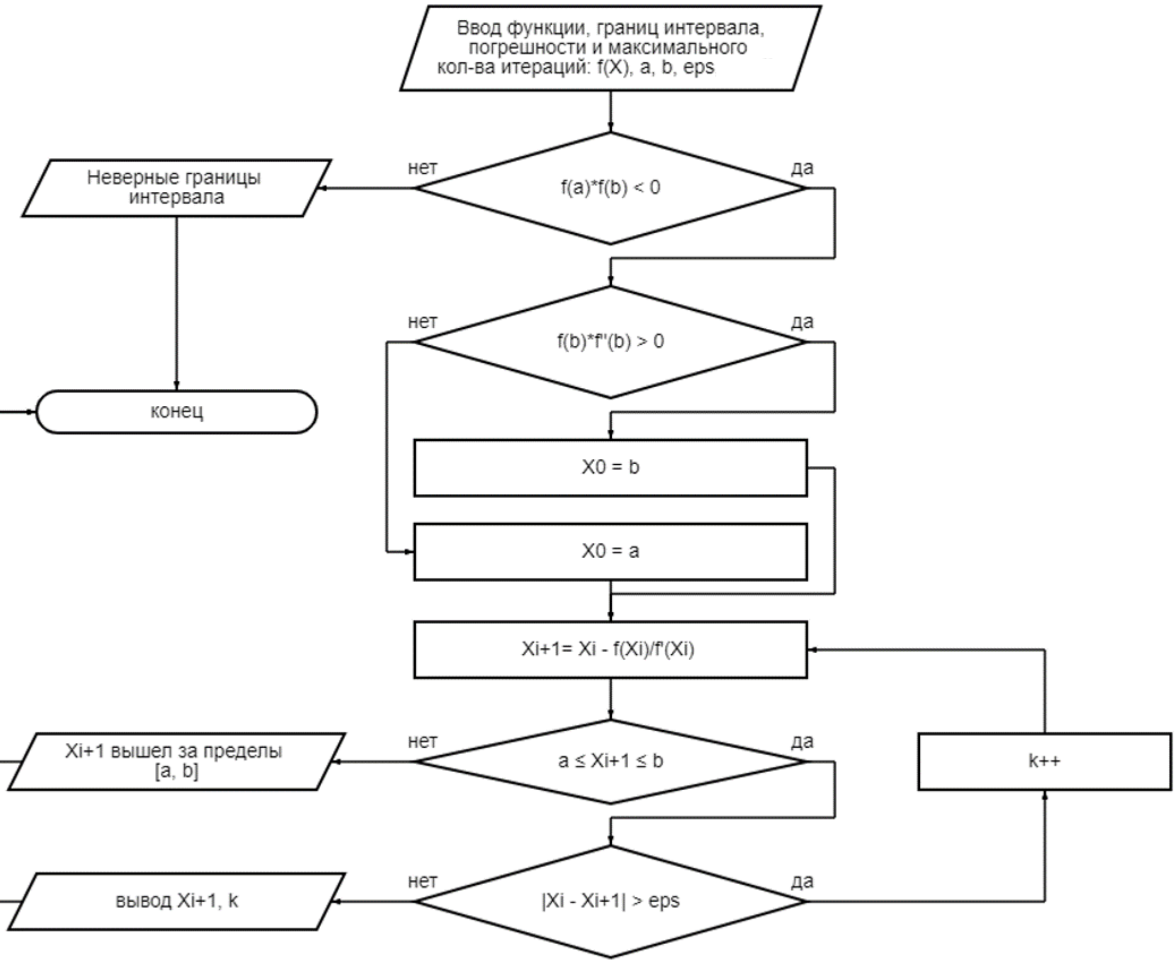


Рисунок 2 "Блок-схема метода касательных"

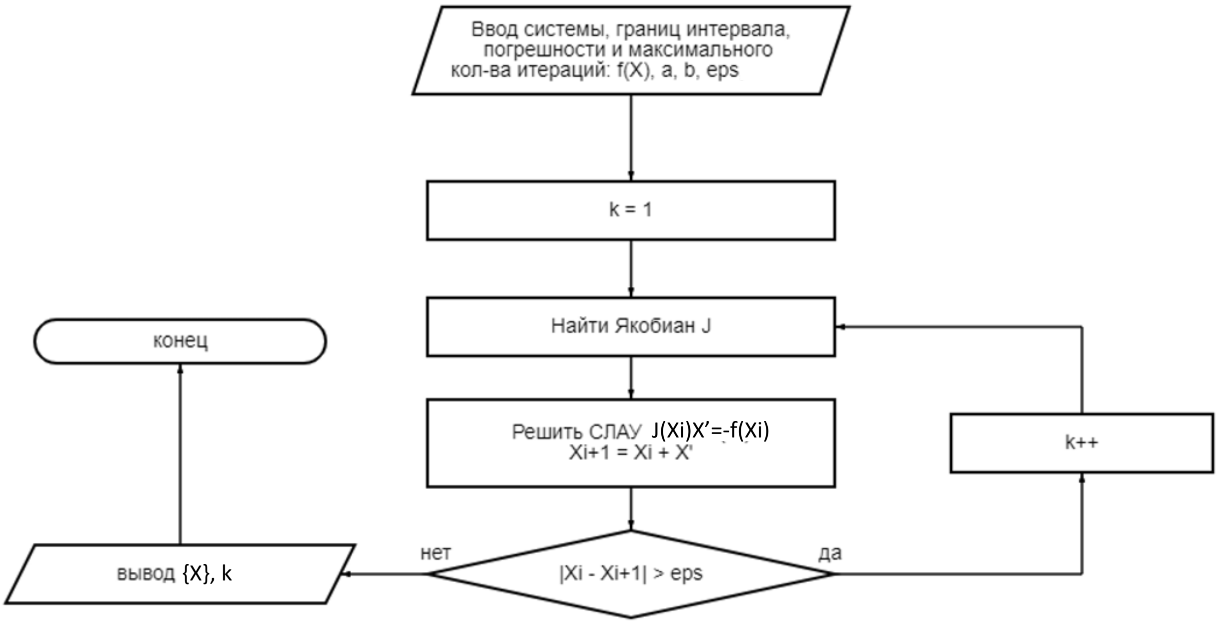


Рисунок 3 "Блок-схема метода Ньютона"

3. Листинг реализованного численного метода программы

```java
public class MethodSecant {

    public MethodAnswer findAnswer(OneArgEquation equation, double eps, double a, double b, int maxItr) {
        double funcAtA = equation.calculate(a);
        double funcAtB = equation.calculate(b);
        int iterationCount = 0;
        double x = 0;
        double currentX = 0;
        String errorMessage = "";
        if (funcAtA * funcAtB < 0) {
            boolean continueFlag;
            do {
                x = currentX;
                iterationCount++;
                currentX = a - ((b - a) * funcAtA) / (funcAtB - funcAtA);
                continueFlag = abs(currentX - x) > eps;
                if (equation.calculate(currentX) * funcAtA < 0) b = currentX;
                if (equation.calculate(currentX) * funcAtB < 0) a = currentX;
                funcAtA = equation.calculate(a);
                funcAtB = equation.calculate(b);
            } while (continueFlag && maxItr > iterationCount);
        } else errorMessage = "Неверный интервал локализации.";
        double[] answer = new double[]{currentX};
        double[] inaccuracy = new double[]{currentX - x};
        if (maxItr == iterationCount) errorMessage = "Достигнуто максимально допустимое количество итераций!";
        return new MethodAnswer(answer, inaccuracy, iterationCount, errorMessage);
    }
}
```

```java
public class MethodTangent {
    public MethodAnswer findAnswer(OneArgEquation equation, double eps, double a, double b, int maxItr) {
        int iterationCount = 0;
        double x = a;
        if (equation.calculate(b) * MathUtils.getDerivativeInPoint( level: 2, equation, b) > 0) x = b;
        double currentX = 0;
        String errorMessage = "";
        if (equation.calculate(a) * equation.calculate(b) < 0) {
            boolean continueFlag;
            do {
                x = currentX;
                iterationCount++;
                currentX = x - equation.calculate(x) / MathUtils.getDerivativeInPoint( level: 1, equation, x);
                continueFlag = abs(x - currentX) > eps;
                if (!(currentX >= a && currentX <= b)) {
                    continueFlag = false;
                    errorMessage = "Xi вышел за пределы [a,b].";
                }
            } while (continueFlag && maxItr > iterationCount);
        } else errorMessage = "Неверный интервал локализации.";
        double[] answer = new double[]{currentX};
        double[] inaccuracy = new double[]{currentX - x};
        if (maxItr == iterationCount) errorMessage = "Достигнуто максимально допустимое количество итераций!";
        return new MethodAnswer(answer, inaccuracy, iterationCount, errorMessage);
    }
}
```

```java
public class MethodNewton {
    public MethodAnswer findAnswer(TwoFuncSystem system, double eps, double currentX, double currentY, int maxItr) {
        int iterationCount = 0;
        double x;
        double y;
        String errorMessage = "";
        boolean continueFlag;
        do {
            x = currentX;
            y = currentY;
            iterationCount++;
            double[] coefficients = calculateCoefficients(x, y, system);
            currentX = x + coefficients[0];
            currentY = y + coefficients[1];
            continueFlag = (abs(x - currentX) > eps || abs(y - currentY) > eps);
        } while (continueFlag && maxItr > iterationCount);
        double[] answer = new double[]{currentX, currentY};
        double[] inaccuracy = new double[]{currentX - x, currentY - y};
        if (maxItr == iterationCount) errorMessage = "Достигнуто максимально допустимое количество итераций!";
        if (Double.isNaN(currentX) || Double.isNaN(currentY)) errorMessage = "Нет корней";
        return new MethodAnswer(answer, inaccuracy, iterationCount, errorMessage);
    }

    private double[] calculateCoefficients(double x, double y, TwoFuncSystem system) {
        double[] derivativesX = MathUtils.getDerivativeInPointByX(system, x, y);
        double[] derivativesY = MathUtils.getDerivativeInPointByY(system, x, y);
        double func1 = system.calculateFirstFunc(x, y);
        double func2 = system.calculateSecondFunc(x, y);
        double[] answers = new double[2];
        answers[0] = (func2 * derivativesY[0] - func1 * derivativesY[1]) / (derivativesY[1] * derivativesX[0] -
                derivativesX[1] * derivativesY[0]);
        answers[1] = (func1 * derivativesX[1] - func2 * derivativesX[0]) / (derivativesY[1] * derivativesX[0] -
                derivativesX[1] * derivativesY[0]);
        return answers;
    }
}
```

4. Примеры и результаты работы программы на разных данных

```
----Решение нелинейных уравнений и систем нелинейных уравнений----
Вводить коэфициенты следует построчно, через пробел.
Пример ввода: a b c d...
Введите вид задачи (0 - нелинейное уравнение, 1 - система нелинейных уравнений, 2 - выход):
0
Уравнения/Системы:
1) y = a * sin(x) + b * cos(x) + x^c + d
2) y = a * x^3 + b * x^2 + c * x + d
3) y = a * x * e ^ b - ln(x + c) + d
Введите номер выбранного уравнения (системы):
2
Введите 4 коэффициента(ов):
1 1 1 1
Введите точность (больше 0 и меньше 1):
0.001
Введите левую границу интервала:
-2
Введите правую границу интервала:
0
---- Результат метода хорд ----
Получены ответы:
x = -0.9986753501170818

Количество итераций: 14

Погрешности:
для x: -8.818021053582648E-4

---- Результат метода касательных ----
Получены ответы:
x = -1.000000000200003

Количество итераций: 2

Погрешности:
для x: -1.000019619845638E-5
---- Сравнение методов ----
Разница между ответами: 0.0013246500829211527
```

```
----Решение нелинейных уравнений и систем нелинейных уравнений----
Вводить коэффициенты следует построчно, через пробел.
Пример ввода: a b c d...
Введите вид задачи (0 - нелинейное уравнение, 1 - система нелинейных уравнений, 2 - выход):
1
Уравнения/Системы:
1) a * sin(x) + b * cos(y) + y^c + d = 0
a * y * x + b * x^2 + c * y + d = 0
2) a * x * e ^ b - ln(y + c) + d = 0
a * y * x + b * x + c * y + d = 0
Введите номер выбранного уравнения (системы):
1
--- Ввод коэффициентов для первого уравнения ---
Введите 4 коэффициента(ов):
1 1 1 1
--- Ввод коэффициентов для второго уравнения ---
Введите 4 коэффициента(ов):
1 1 1 1
Введите точность (больше 0 и меньше 1):
0.001
Введите приближение для x:
-1
Введите приближение для y:
-1\
Неверный формат данных, повторите ввод:
Введите приближение для y:
-1
---- Результат метода Ньютона ----
Получены ответы:
x = -0.15124226960550777
y = -1.206047912534124

Количество итераций: 28

Погрешности:
для x: 8.868655363434286E-4
для y: -4.5327689160967743E-4
```

5. Вывод

В ходе выполнения данной лабораторной работы были изучены следующие итерационные методы нахождения корней нелинейных уравнений: метод хорд и метод касательных. Метод хорд основан на нахождении точки пересечения хорды с осью Ox и выбора одного из двух получившихся интервалов; метод касательных основан на нахождении точки пересечения касательной с осью Ox (эта точка становится следующим приближением x_i). Скорость сходимости метода касательных выше, чем метода хорд, но при этом условия применимости метода касательных строже чем у метода хорд.

Сравнение скоростей сходимости методов и условий применимости:

- метод половинного деления и метод хорд имеют линейную скорость сходимости (но при этом метод хорд быстрее метода половинного деления); условия, необходимые для применения методов: f(x) – непрерывная, f(a)*f(b) < 0. Данные методы не обобщаются на системы уравнений.
- метод простых итераций имеет линейную скорость сходимости. Достаточное условие сходимости метода: $|\varphi'(x)| \leq q < 1$ на [a, b] и $\varphi(x)$ имеет производную на [a, b]. Данный метод обобщается на системы уравнений – метод итераций для решения систем уравнений, который также имеет линейную скорость сходимости.
- метод касательных имеет квадратичную скорость сходимости; условия, необходимые для применения метода: f(x) – определена и дважды дифференцируема на [a, b], f'(x) != 0, f(a)*f(b) < 0, $f(x_0) * f''(x_0) > 0$ (для обеспечения быстрой сходимости). Данный метод обобщается на системы уравнений – метод Ньютона, который тоже имеет квадратичную скорость сходимости.

Сравнивая между собой метод Ньютона и метод итераций, стоит отметить, что метод Ньютона выигрывает по скорости сходимости, но при этом имеет более сложные вычисления в процессе итераций (нахождение матрицы и решение СЛАУ).