УНИВЕРСИТЕТ ИТМО

Факультет ПИиКТ

Дисциплина: вычислительная математика

Лабораторная работа №3

“Численные методы решения нелинейных уравнений”

Выполнил: Тарасов Александр

Группа: Р3212

Преподаватель: Перл О.В.

Вариант: аб1

Метод половинного деления
метод хорд
метод Ньютона

Санкт-Петербург, 2020 г.

Описание метода

Метод половинного деления - численный метод приближенного нахождения корня нелинейного уравнения с заданной точностью. Основная идея метода – в выполнении итерационного процесса, в котором заданный интервал поиска корней [a;b] мы разбиваем пополам и оцениваем, в каком из двух новых интервалов содержится корень. При достижении заданной точности очередная середина интервала и будет решением

уравнения: $x_i = \frac{a_i+b_i}{2}$

Критерий окончания итерационного процесса: $|b_n - a_n| \le \varepsilon$ или $|f(x_n)| \le \varepsilon$.

Для использования данного метода должно выполняться условие существования корней на заданном интервале: $f(a) \cdot f(b) < 0$,

Метод хорд схож с методом половинного деления, только в данном случае в качестве приближенного значения на каждой итерации выбирается точка пересечения хорды, связывающей левую и правую границы интервала с осью абсцисс. То есть, при условии f(a)*f(b) < 0 мы разбиваем отрезок [a; b] на два отрезка с помощью хорды и выбираем новый от точки пересечения хорды с осью абсцисс до неподвижной точки, на котором функция меняет знак. При этом должны выполняться следующие условия:

$$x_1 = a - \frac{(b-a)}{f(b)-f(a)} f(a) \qquad x_{i+1} = x_i - \frac{(b-x_i)}{f(b)-f(x_i)} f(x_i)$$

или

$$x_1 = b - \frac{(a-b)}{f(a)-f(b)} f(b) \qquad x_{i+1} = x_i - \frac{(a-x_i)}{f(a)-f(x_i)} f(x_i)$$

Критерий окончания итерационного процесса:

$$|x_n - x_{n-1}| \le \varepsilon \text{ или } |f(x_n)| \le \varepsilon.$$

Метод Ньютона для систем нелинейных уравнений является обобщением метода Ньютона решения нелинейных уравнений, который основан на идее построения последовательных приближений.

Пусть дана система нелинейных уравнений:

$$\left.\begin{array}{l} f_1(x_1, x_2, \ldots, x_n) = 0, \\ f_2(x_1, x_2, \ldots, x_n) = 0, \\ \cdots\cdots\cdots\cdots\cdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{array}\right\}$$

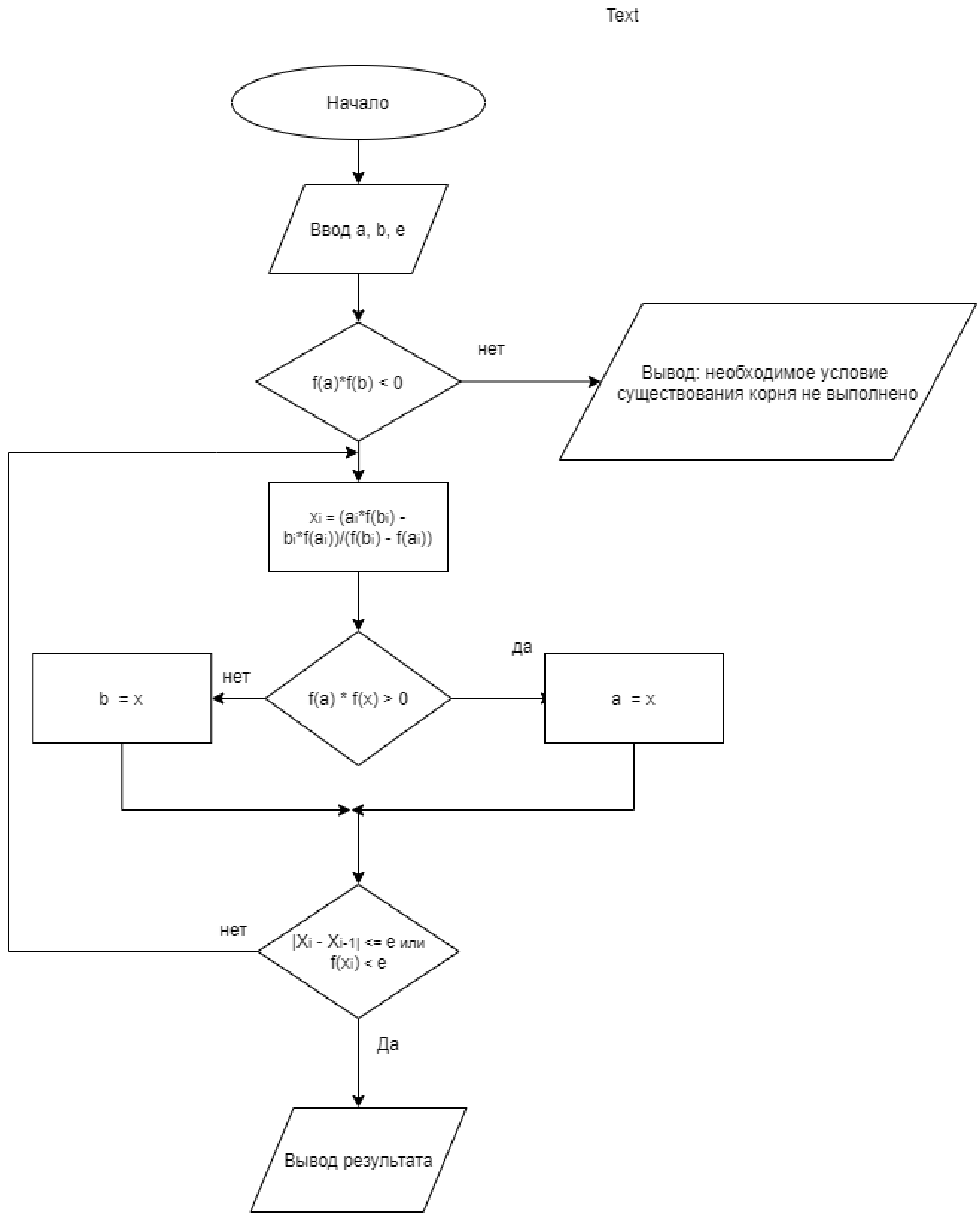
Приняв за начальное приближение некоторые числа, можно получить приближенное значение с заданной точностью с помощью значений, вычисленных на предыдущих итерациях. Тогда формула нахождения очередного приближения выглядит так:

$$x^{(p+1)} = x^{(p)} - W^{-1}(x^{(p)}) f(x^{(p)}) \quad (p = 0, 1, 2, \ldots)$$

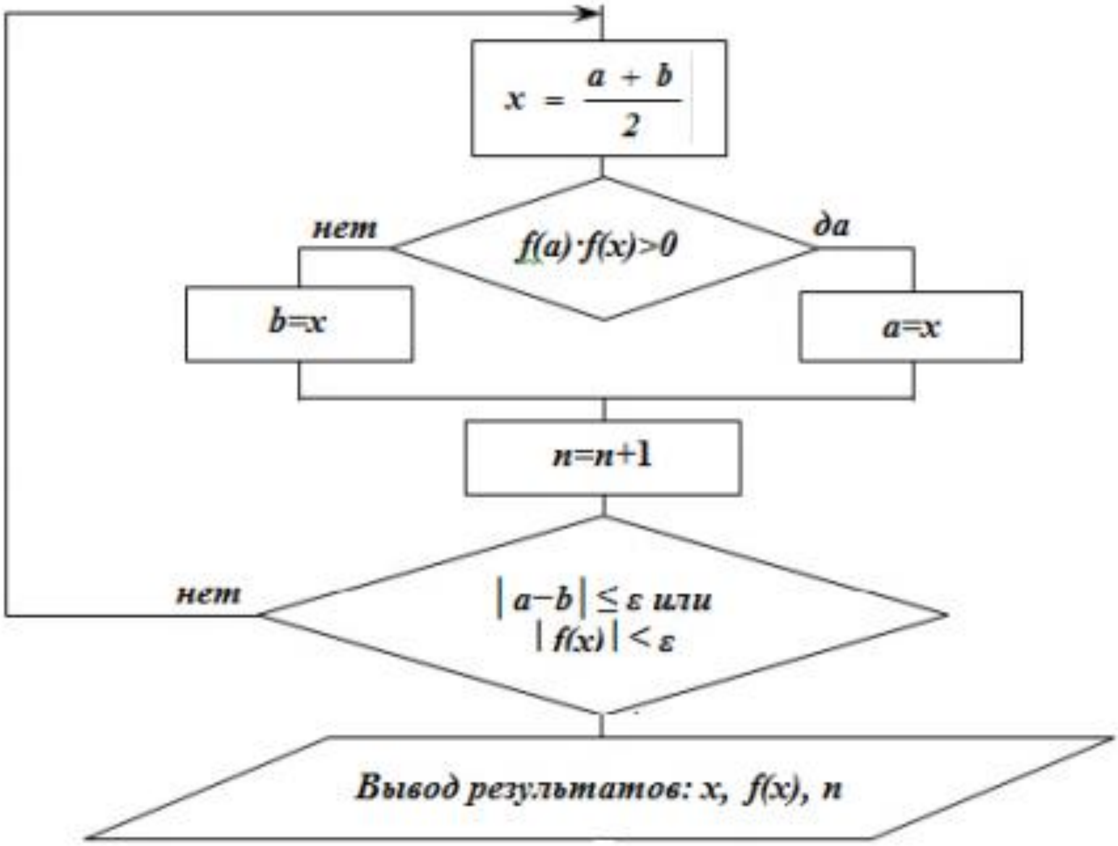
Где $W(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$ - матрица Якоби.

Блок-схема численных методов

Метод хорд



Метод половинного деления



Листинг численных методов

BsectionMethod.java

```
public String solve(){

    if(checkSuffCondition(leftBorder, rightBorder)) {
        while (Math.abs(rightBorder - leftBorder) > accuracy) {
            currentX = (leftBorder + rightBorder) / 2;
            if (function.solve(leftBorder, 0) * function.solve(currentX, 0) > 0)
                leftBorder = currentX;
            else rightBorder = currentX;
            steps++;
        }
        return String.format("%4.10f ", currentX);
    }else return "уравнене не имеет корней или имеет более одного корня на данном интервале";
}
```

ChordMethod.java

```
public String solve() {

    if(checkSuffCondition(leftBorder, rightBorder)) {
        currentX = leftBorder;
        while (Math.abs(currentX - previousX) > accuracy || Math.abs(function.solve(currentX,0)) > accuracy){
            previousX = currentX;
            currentX = (leftBorder * function.solve(rightBorder,0) - rightBorder * function.solve(leftBorder,0))/
                        (function.solve(rightBorder,0) - function.solve(leftBorder,0));
            if (function.solve(leftBorder,0) * function.solve(currentX,0) > 0)
                leftBorder = currentX;
            else rightBorder = currentX;
            steps++;
        }
        return String.format("%4.10f ", currentX);
    }else return "уравнене не имеет корней \nили имеет более одного корня на данном интервале";
}

public double getDerivativeFunction(double x) {
    switch (funstionStr){
        case "x^2 + 25x = 0":
            return 2*x + 25;
        case "x^3+23x-56 = 0":
            return 3*x*x+23;
        case "sin(x)":
            return Math.cos(x);
        case "x^3 - x + 4":
            return 3*x*x - 1;
    }
    return  0;
}

public double getSecondDerivativeFunction(double x){
    switch (funstionStr){
        case "x^2 + 25x = 0":
            return 2;
        case "x^3+23x-56 = 0":
            return 6*x;
        case "sin(x)":
```

```java
                return -Math.sin(x);
            case "x^3 - x + 4":
                return 6*x;
        }
        return  0;
    }
```

NewtonMethod.java

```java
    public String solve() {

        while (firstFunc.solve(currentX, currentY) > accuracy ||
    secondFunc.solve(currentX, currentY) > accuracy){
            iterCount++;
            previousX = currentX;
            previousY = currentY;
            initMatrix(previousX, previousY);

            currentX = previousX - hlpX;
            currentY = previousY - hlpY;

            if (iterCount >= 1000000)
                return "превышен лимит итераций";
        }

        return "X =  " + String.format("%4.10f ",currentX) + "   Y =   " + currentY;
    }

    private void initMatrix(double x, double y){
        switch (firtsFuncStr){
            case "y - x^3 - 4 = 0":
                jacobiMatrix[0][0] = -3.0 * x * x;
                jacobiMatrix[0][1] = 1.0;
                break;
            case  "y + e^x + 1 = 0":
                jacobiMatrix[0][0] = Math.exp(x);
                jacobiMatrix[0][1] = 1.0;
                break;
            case "x^2 + y = 0":
                jacobiMatrix[0][0] = 2.0 * x;
                jacobiMatrix[0][1] = 1.0;
                break;
        }
        switch (secondFuncStr){
            case "y - x^3 - 4 = 0":
                jacobiMatrix[1][0] = -3.0 * x * x;
                jacobiMatrix[1][1] = 1.0;
                break;
            case  "y + e^x + 1 = 0":
                jacobiMatrix[1][0] = Math.exp(x);
                jacobiMatrix[1][1] = 1.0;
                break;
            case "x^2 + y = 0":
                jacobiMatrix[1][0] = 2.0 * x;
                jacobiMatrix[1][1] = 1.0;
                break;
        }

        det = jacobiMatrix[0][0] * jacobiMatrix[1][1] - jacobiMatrix[0][1] *
```

```
jacobiMatrix[1][0];

        hlpX = (firstFunc.solve(x,y)* jacobiMatrix[1][1] - secondFunc.solve(x,y) *
jacobiMatrix[0][1])/det;
        hlpY = (jacobiMatrix[0][0] * secondFunc.solve(x,y) - jacobiMatrix[1][0] *
firstFunc.solve(x,y))/det;
```

Вывод:

Невозможность решать уравнения на интервалах, содержащих несколько корней - явный недостатки данных методов.
Метод половинного деления обладает абсолютной сходимостью, что является плюсом.
Метод хорд требует проверки сходимости – это минус.

Выбор начального приближения влияет на скорость сходимости процессов в данных методах.

В методе Ньютона на каждой итерации приходится считать матрицу Якоби и её детерминант, что увеличивает кол-во операций.