Московский Авиационный Институт
(Национальный исследовательский университет)

Лабораторная работа №2

по курсу: «численные методы»

Выполнил: Коростелев Д.В.

Группа: М8О-308Б-18

Номер по списку: 10

г. Москва 2021

Скриншоты заданий

```
5 (1/10)  найти корни уравнений с точностью 0.01 , выполнив не менее 5 итераций
      Корни ищутся на отрезках длиной 1 и с целочисленными границами
      Методы касательных и итераций начинают попытку с левой границы
      При неудачной первой попытке - вычисления продолжить, начав с правой границы
        (  1 ;  2 ) метод половинного деления        2*x*x*x  + 2*x  - 8  = 0
        (  1 ;  2 ) метод секущих (хорд)               x*x*x  + 3*x  - 5  = 0
        (  0 ;  1 ) метод касательных (Ньютона)     0.5*x*x*x  + 11*x  - 6.5  = 0
        ( -2 ; -1 ) метод  итераций                 4*x*x*x  + 3*x  + 22  = 0

6 (  1 10)  методом линеаризации (Ньютона) решить систему
         3* x*x*x  -y*y*y  +   9 = 0
         x*y - 2* y -   5  = 0
      За начальные значения взять (  3;   4) и ( -1; -1)
      Сделать по  5 итераций
```

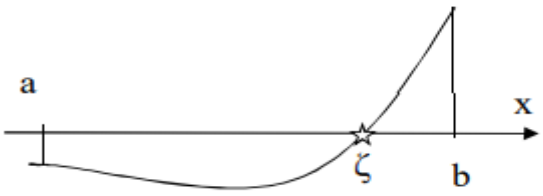
Метод половинного деления, метод секущих, метод касательных, метод итераций

Условие:

```
5 (1/10)  найти корни уравнений с точностью 0.01 , выполнив не менее 5 итераций
      Корни ищутся на отрезках длиной 1 и с целочисленными границами
      Методы касательных и итераций начинают попытку с левой границы
      При неудачной первой попытке - вычисления продолжить, начав с правой границы
        (  1 ;  2 ) метод половинного деления          2*x*x*x  + 2*x  - 8  = 0
        (  1 ;  2 ) метод секущих (хорд)                 x*x*x  + 3*x  - 5  = 0
        (  0 ;  1 ) метод касательных (Ньютона)      0.5*x*x*x  + 11*x  - 6.5  = 0
        ( -2 ; -1 ) метод  итераций                  4*x*x*x  + 3*x  + 22  = 0
```

Теория:

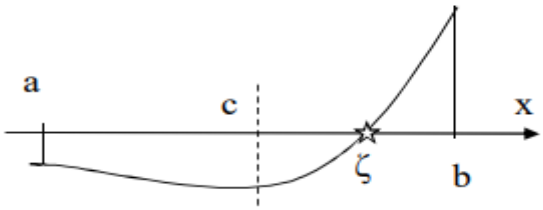
4.1. Метод половинного деления (дихотомии, бисекций)



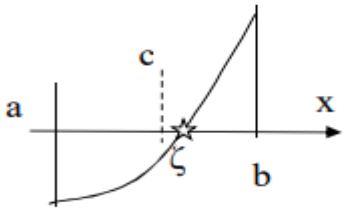
Пусть известен отрезок $[a\,;\,b]$, на котором непрерывная функция $y=f(x)$ меняет знак, т.е. в точках a и b функция имеет значения разных знаков: $f(a)\cdot f(b) \le 0$. Тогда существует точка $x=\zeta$ такая, что $f(\zeta) = 0$.

Разделим отрезок $[a\,;\,b]$ точкой c пополам : $c = (a+b)/2$.

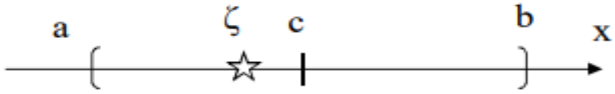
Точка c разделила отрезок $[a\,;\,b]$ на два отрезка : $[a\,;\,c]$ и $[c\,;\,b]$ (левый и правый). Но только на одном из них функция будет менять знак: на $[a\,;c]$ (если $f(a)\times f(c) < 0$) или на $[c\,;b]$ (если $f(c)\times f(b) < 0$).



Для данного конкретного рисунка выбираем правый отрезок. Теперь точка c стала новой точкой a, точка b осталась на месте.



Новый отрезок $[a\,;\,b]$ делится новой точкой $c = (a+b)/2$ на два отрезка : $[a\,;\,c]$ и $[c\,;\,b]$. И вновь из двух отрезков выбираем тот, на котором функция будет менять знак. Для данного конкретного рисунка видно, что это опять правый отрезок. Но при следующем делении может быть выбран уже левый отрезок. С каждым делением длина отрезка $[a\,;\,b]$ уменьшается в два раза и скоро станет меньше $2\cdot\varepsilon$ ($\varepsilon > 0$ и задаётся заранее) .

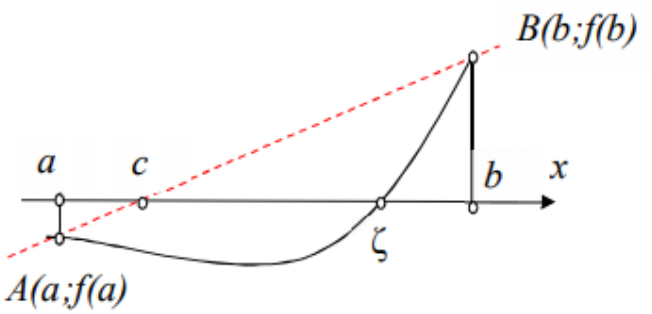


Тогда расстояние от ζ (ζ = неизвестное точное значение) до c не больше расстояния от c до a , т.е.

$$| c - \zeta | \le |(b - a) / 2 | \le \varepsilon \ . \qquad (4.1.1)$$

4.2. Метод хорд (секущих)

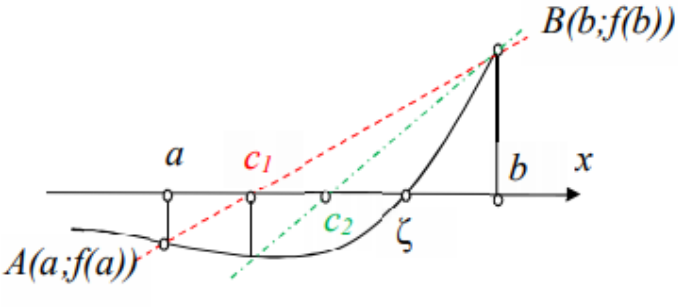
Пусть функция $f(x)$ (уравнения $f(x) = 0$) непрерывна на отрезке $[a;b]$ и имеет значения разных знаков на границе отрезка: $(f(a)\times f(b) < 0)$.



Через точки $A(a;f(a))$ и $B(b;f(b))$ проведём прямую линию, которая пересечёт ось ox в некоторой точке c. Точка c находится из соотношения

$$c = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}.$$

Точка c разделила отрезок $[a;b]$ на два отрезка, но разные знаки значения функции будут иметь только на одном из них: на $[a;c]$ (если $f(a)\times f(c) < 0$) или на $[c;b]$ (если $f(c)\times f(b) < 0$). В ситуации, которая представлена на рисунке, мы выбираем правый отрезок. В новом отрезке $[a;b]$ через точки $A(a;f(a))$ и $B(b;f(b))$ проведём прямую линию, которая пересечёт ось ox в некоторой новой точке c_2 (4.2.1). И так далее.



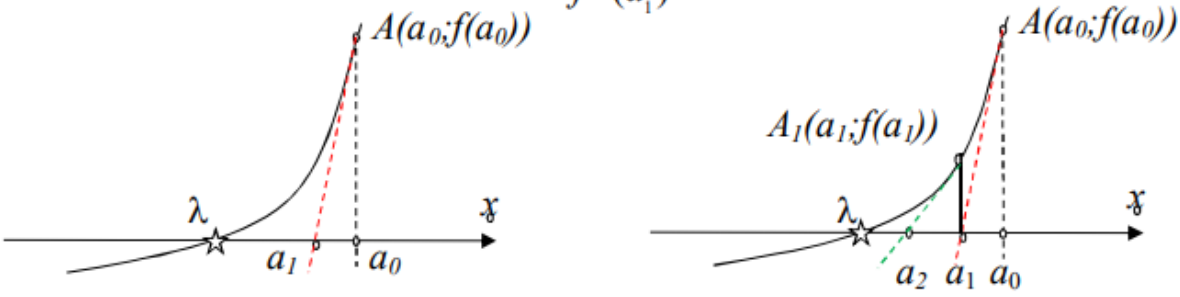
Особенность метода в том, что раз выбрав, например, правый отрезок, на каждом шаге итераций выбирать придётся тоже правый отрезок.

Все точки c приближаются к корню с одной стороны (имеет место односторонняя сходимость). Итерации можно остановить, когда точки c на соседних двух шагах почти одинаковы и значения функции $f(x)$ в точке c почти ноль ($\varepsilon>0$, $\varepsilon\approx 0$):

$$\begin{cases} |c_n - c_{n-1}| < \varepsilon \\ |f(c_n)| < \varepsilon \end{cases} \qquad (4.2.2)$$

4.3. Метод касательных (Ньютона)

Пусть функция $f(x)$ дважды непрерывно дифференцирована. Пусть в окрестности корня уравнения $f(x)=0$, в некоторой точке a_0 выполняется соотношение $f(a_0)\cdot f``(a_0)>0$. Из точки $A(a_0;f(a_0))$ проведём касательную к графику функции $y=f(x)$. Она пересечёт ось OX в точке a_1 : $a_1 = a_0 - \dfrac{f(a_0)}{f`(a_0)}$ . Из точки $A_1(a_1;f(a_1))$ проведём касательную к графику функции $y=f(x)$. Она пересечёт ось $OX$ в точке $a_2 = a_1 - \dfrac{f(a_1)}{f`(a_1)}$.



Таким образом, можно получить бесконечную последовательность точек a_n, которые быстро будут приближаться к корню λ. Преимущество метода в его высокой скорости, но есть и недостатки. Это необходимость уметь вычислять производную от функции $f(x)$, а это бывает не всегда возможно. Ещё желательно на каждом i-м шаге проверять выполнение неравенства $f(a_i)\cdot f``(a_i)>0$.

4.4. Метод итераций

Пусть уравнение $f(x)=0$ удалось преобразовать к равносильному уравнению $x=g(x)$, функция $g(x)$ которого обладает свойством: в окрестности корня $x=\lambda$

$$|g`(x)| \le q < 1 \ , \qquad (4.4.1)$$

то итерационный процесс $x_{n+1}=g(x_n)$ будет сходиться к корню λ уравнения $f(x)=0$. Причём, сходимость будет тем быстрее, чем меньше q. Величины $|x_n-\lambda|$ будут убывать не медленнее бесконечно–убывающей прогрессии со знаменателем q. Процесс останавливаем, если для некоторого малого положительного ε

$$\begin{cases} \dfrac{(x_n - x_{n-1})^2}{|2x_{n-1} - x_n - x_{n-2}|} < \varepsilon \\ |f(x_n)| < \varepsilon \end{cases} \qquad (4.4.2)$$

Скорость сходимости метода может быть разной (и хуже, чем у метода половинного деления, и почти такой же высокой, как у метода касательных). Сходимость может быть как односторонней, так и двухсторонней (точки x_n могут ложиться как с одной стороны от корня, так и поочерёдно с разных сторон).

Решение:

Файл Main.java

```
package second;

public class Main {
    public static void main(String[] args) {

        PolovinnoeDelenie pd = new PolovinnoeDelenie() {
            //задаем начальные условия и условия окончания
            @Override
            double f(double x) {
                return 2*x*x*x + 2*x - 8;
            }
            int iterations = 10;
            @Override
            boolean endCondition() {
                if(iterations > 0){
                    --iterations;
                    return true;
                }else {
                    return false;
                }
            }
            @Override
            double getA(){
                return 1;
            }
            @Override
            double getB(){
                return 2;
            }
        };
        System.out.println("Метод половинного деления");
        pd.exec();
        System.out.println();

        SekushieHordi sh = new SekushieHordi() {
            //задаем начальные условия и условия окончания
            @Override
            double getA() {
                return 1;
            }

            @Override
            double getB() {
                return 2;
            }

            @Override
            double f(double x) {
                return x*x*x + 3*x - 5;
            }

            int iterations  =10;
            @Override
            boolean endCondition() {
                if(iterations > 0){
                    --iterations;
                    return true;
                }else {
                    return false;
```

```
            }
        }
    };
    System.out.println("Метод секущих");
    sh.exec();
    System.out.println();
    System.out.println("Метод касательных");
    Kasatelnie kas = new Kasatelnie() {
        //задаем начальные условия и условия окончания
        @Override
        double getA() {
            return 0;
        }

        @Override
        double getB() {
            return 1;
        }

        int iterations  =10;
        @Override
        boolean endCondition() {
            if(iterations > 0){
                --iterations;
                return true;
            }else {
                return false;
            }
        }

        @Override
        double f(double x) {
            return 0.5*x*x*x + 11 * x - 6.5;
        }

        @Override
        double df(double x) {
            return 1.5*x*x + 11 ;
        }

        @Override
        double ddf(double x) {
            return 3*x;
        }
    };
    kas.exec();
    System.out.println("Метод итераций");
    Iterations iter = new Iterations() {
        //задаем начальные условия и условия окончания
        @Override
        double getA() {
            return -2;
        }

        @Override
        double getB() {
            return -1;
        }

        @Override
        double f(double x) {
            return 4.0*x*x*x + 3.0*x + 22.0;
        }
```

```
            @Override
            double g(double x) {
                return Math.cbrt((-3.0*x - 22.0)/4.0);
            }

            @Override
            double dg(double x) {
                return (1.0/3.0) * Math.pow((-3.0/4.0)*x + 22.0/4.0, -
2.0/3.0) * (-3.0/4.0);
            }

            int iterations  =10;
            @Override
            boolean endCondition() {
                if (iterations > 0) {
                    --iterations;
                    return true;
                } else {
                    return false;
                }
            }
        };
        iter.exec();
        System.out.println();
    }
}
```

Файл PolovinnoeDelenie.java

```
package second;

abstract public class PolovinnoeDelenie {
    abstract double getA();
    abstract double getB();
    abstract double f(double x);
    void exec() throws RuntimeException{
        double a,b;
        a = getA();
        b = getB();
        int iteration =1;
        System.out.println("\t\t a \t\t\t\t c \t\t\t\t b \t\t\t\t f(a)
\t\t\t\t f(c) \t\t\t\t f(b)");
        while(endCondition()){
            double fa = f(a);
            double fb = f(b);
            double c = (a + b)/2;
            double fc = f(c);

            printRes(iteration, a,c,b,fa,fc,fb);
            if(fa * fc < 0) {
                b = c;
            } else if(fc*fb < 0){
                a = c;
            } else{
                throw new RuntimeException();
            }
            ++iteration;
        }
    }
    void printRes(int iteration, double a, double c, double b, double fa,
double fc, double fb){
        System.out.println("[" + iteration+"] : \t" +
```

```
String.format("%.10f\t%.10f\t%.10f\t%.10f\t%.10f\t%.10f\t",
a,c,b,fa,fc,fb));
    }
    abstract boolean endCondition();
}
```

Вывод консоли:

```
Метод половинного деления
         a               c               b               f(a)              f(c)              f(b)
[1] :   1,0000000000    1,5000000000    2,0000000000    -4,0000000000    1,7500000000    12,0000000000
[2] :   1,0000000000    1,2500000000    1,5000000000    -4,0000000000    -1,5937500000    1,7500000000
[3] :   1,2500000000    1,3750000000    1,5000000000    -1,5937500000    -0,0507812500    1,7500000000
[4] :   1,3750000000    1,4375000000    1,5000000000    -0,0507812500    0,8159179688    1,7500000000
[5] :   1,3750000000    1,4062500000    1,4375000000    -0,0507812500    0,3743286133    0,8159179688
[6] :   1,3750000000    1,3906250000    1,4062500000    -0,0507812500    0,1597366333    0,3743286133
[7] :   1,3750000000    1,3828125000    1,3906250000    -0,0507812500    0,0539712906    0,1597366333
[8] :   1,3750000000    1,3789062500    1,3828125000    -0,0507812500    0,0014687777    0,0539712906
[9] :   1,3750000000    1,3769531250    1,3789062500    -0,0507812500    -0,0246877521    0,0014687777
[10] :  1,3769531250    1,3779296875    1,3789062500    -0,0246877521    -0,0116173718    0,0014687777
```

Файл SekushieHordi.java

```
package second;
abstract public class SekushieHordi {
    abstract double getA();
    abstract double getB();
    abstract double f(double x);
    void exec() throws RuntimeException{
        double a,b;
        a = getA();
        b = getB();
        int iteration =1;
        System.out.println("\t\t a \t\t\t\t c \t\t\t\t b \t\t\t\t f(a)
\t\t\t\t f(c) \t\t\t\t f(b)");
        while(endCondition()){
            double fa = f(a);
            double fb = f(b);
            double c = (a*fb - b * fa)/(fb-fa);
            double fc = f(c);

            printRes(iteration, a,c,b,fa,fc,fb);
            if(fa * fc < 0) {
                b = c;
            } else if(fc*fb < 0){
                a = c;
            } else{
                throw new RuntimeException();
            }
            ++iteration;
        }
    }
    void printRes(int iteration, double a, double c, double b, double fa,
double fc, double fb){
        System.out.println("[" + iteration+"] : \t" +
String.format("%.10f\t%.10f\t%.10f\t%.10f\t%.10f\t%.10f\t",
a,c,b,fa,fc,fb));
    }
```

```
    abstract boolean endCondition();
}
```

Вывод консоли:

```
Метод секущих
         a               c               b              f(a)              f(c)              f(b)
[1] :   1,0000000000    1,1000000000    2,0000000000    -1,0000000000    -0,3690000000    9,0000000000
[2] :   1,1000000000    1,1354466859    2,0000000000    -0,3690000000    -0,1297975921    9,0000000000
[3] :   1,1354466859    1,1477379702    2,0000000000    -0,1297975921    -0,0448680510    9,0000000000
[4] :   1,1477379702    1,1519657087    2,0000000000    -0,0448680510    -0,0154155864    9,0000000000
[5] :   1,1519657087    1,1534157745    2,0000000000    -0,0154155864    -0,0052852985    9,0000000000
[6] :   1,1534157745    1,1539126438    2,0000000000    -0,0052852985    -0,0018107788    9,0000000000
[7] :   1,1539126438    1,1540828404    2,0000000000    -0,0018107788    -0,0006202315    9,0000000000
[8] :   1,1540828404    1,1541411324    2,0000000000    -0,0006202315    -0,0002124249    9,0000000000
[9] :   1,1541411324    1,1541610966    2,0000000000    -0,0002124249    -0,0000727519    9,0000000000
[10] :  1,1541610966    1,1541679339    2,0000000000    -0,0000727519    -0,0000249160    9,0000000000
```

Файл Kasatelnie.java

```
package second;
abstract public class Kasatelnie {
    abstract double getA();
    abstract double getB();
    abstract boolean endCondition();
    abstract double f(double x);
    //первая производная
    abstract double df(double x);
    //вторая производная
    abstract double ddf(double x);
    void exec() throws RuntimeException{
        double prev, cur;
        double a = getA();
        double b = getB();
        double fa = f(a);
        double ddfa = ddf(a);
        if(fa * ddfa > 0){
            prev = a;
        } else {
            prev = b;
        }
        int iteration = 1;
        System.out.println("\t\t a[n-1] \t\tf(a[n-1]) \t\t df(a[n-1]) \t
ddf(a[n-1]) \t a[n]");
        while(endCondition()){
            cur = prev - (f(prev)/df(prev));
            printRes(iteration, prev, f(prev), df(prev), ddf(prev), cur);
            ++iteration;
            prev = cur;
        }
    }
    void printRes(int iteration, double prev, double fprev, double dfprev,
double ddfprev, double cur){
        System.out.println("[" + iteration+"] : \t" +
String.format("%.10f\t%.10f\t%.10f\t%.10f\t%.10f\t", prev,fprev,dfprev,
ddfprev,cur));
    }
}
```

Вывод консоли:

```
Метод касательных
          a[n-1]          f(a[n-1])         df(a[n-1])       ddf(a[n-1])       a[n]
[1] :   1,0000000000    5,0000000000    12,5000000000   3,0000000000    0,6000000000
[2] :   0,6000000000    0,2080000000    11,5400000000   1,8000000000    0,5819757366
[3] :   0,5819757366    0,0002894589    11,5080436369   1,7459272097    0,5819505838
[4] :   0,5819505838    0,0000000006    11,5079997230   1,7458517515    0,5819505838
[5] :   0,5819505838    0,0000000000    11,5079997229   1,7458517513    0,5819505838
[6] :   0,5819505838    0,0000000000    11,5079997229   1,7458517513    0,5819505838
[7] :   0,5819505838    0,0000000000    11,5079997229   1,7458517513    0,5819505838
[8] :   0,5819505838    0,0000000000    11,5079997229   1,7458517513    0,5819505838
[9] :   0,5819505838    0,0000000000    11,5079997229   1,7458517513    0,5819505838
[10] :  0,5819505838    0,0000000000    11,5079997229   1,7458517513    0,5819505838
```

Файл Iterations.java

```
package second;

abstract public class Iterations {
    abstract double getA();
    abstract double getB();
    abstract double f(double x);
    abstract double g(double x);
    //первая производная g(x)
    abstract double dg(double x);

    void exec() {
        double prev, current;
        prev = getA();
        int iteration = 1;
        System.out.println("\t\t x \t\t\t\tf(x) \t\t\t g(x) \t\t\t dg(x)");
        while(endCondition()){
            current = g(prev);
            printRes(iteration, current, f(current), g(current),
dg(current));
            ++iteration;
            prev = current;
        }
    }
    void printRes(int iteration, double x, double fx, double gx, double
dgx){
        System.out.println("[" + iteration+"] : \t" +
String.format("%.10f\t%.10f\t%.10f\t%.10f", x, fx, gx, dgx));
    }
    abstract boolean endCondition();
}
```

Вывод консоли:

```
Метод итераций
          x              f(x)              g(x)             dg(x)
[1] :   -1,5874010520   1,2377968441    -1,6273235980   -0,0704096352
[2] :   -1,6273235980   -0,1197676381   -1,6235459721   -0,0702003482
[3] :   -1,6235459721   0,0113328776    -1,6239041790   -0,0702200851
[4] :   -1,6239041790   -0,0010746207   -1,6238702194   -0,0702182130
[5] :   -1,6238702194   0,0001018787    -1,6238734390   -0,0702183905
[6] :   -1,6238734390   -0,0000096587   -1,6238731338   -0,0702183737
[7] :   -1,6238731338   0,0000009157    -1,6238731627   -0,0702183753
[8] :   -1,6238731627   -0,0000000868   -1,6238731600   -0,0702183751
[9] :   -1,6238731600   0,0000000082    -1,6238731602   -0,0702183751
[10] :  -1,6238731602   -0,0000000008   -1,6238731602   -0,0702183751
```

Ответ:

Ответ методом половинного деления: 1,3779296875

Ответ методом секущих: 1,1541679339

Ответ методом касетльных: 0,5819505838

Ответ методом итераций: − 1,6238731602

Метод линеаризации (Ньютона)

Условие:

```
6 (  1 10)  методом линеаризации (Ньютона) решить систему
          3* x*x*x  -y*y*y  +   9 = 0
          x*y - 2* y -   5  = 0
       За начальные значения взять (  3;   4) и ( -1; -1)
       Сделать по  5 итераций
```

Теория:

5.1. Метод Ньютона (линеаризации)

Пусть известно приближение $\overline{\chi}^{(n)} = (x_1^{(n)}, x_2^{(n)}, \ldots, x_k^{(n)})^T$ к вектору-решению системы (5.1), вычислим его уточнение по формуле

$$\overline{\chi}^{(n+1)} = \overline{\chi}^{(n)} - J_{(n)}^{-1} \cdot \overline{f}^{(n)}(\overline{\chi}^{(n)}) \quad . \qquad (5.1.1)$$

Тут $J_{(n)}^{-1}$ = обратная матрица для матрицы Якоби $J_{(n)}$ (матрица частных производных, состоящая из элементов $j_{pq}^{(n)} = \frac{\partial f_p}{\partial x_q}$, вычисленных в точке $\overline{\chi}^{(n)}$); $\overline{f}^{(n)}(\overline{\chi}^{(n)})$ = вектор значений функций, стоящих в левой части (5.1).

Чем ближе выберем нулевое (начальное) приближение (например, по графику, или по каким-то другим соображениям), тем быстрее итерационный процесс (5.1-1) сойдётся к решению системы (5-1).

Решение:

Файл SimpleNuton.java

```
package second;


abstract public class SimpleNuton {
    abstract boolean isEnd();

    abstract double f1(double x, double y);

    abstract double f2(double x, double y);
    //производная f1 по x
    abstract double df1x(double x, double y);
    //производная f1 по y
    abstract double df1y(double x, double y);
    //производная f2 по x
    abstract double df2x(double x, double y);
    //производная f2 по y
    abstract double df2y(double x, double y);
```

```
    abstract Point getStartPoint();

    private Point function(Point p) {
        Point result = new Point();
        result.x = f1(p.x, p.y);
        result.y = f2(p.x, p.y);
        return result;
    }

    private double[][] getJakobyMatrix(Point p) {
        double[][] res = new double[2][2];
        res[0][0] = df1x(p.x, p.y);
        res[0][1] = df1y(p.x, p.y);
        res[1][0] = df2x(p.x, p.y);
        res[1][1] = df2y(p.x, p.y);
        return res;
    }

    private double[][] getReverse(double[][] matrix) {
        double det = (matrix[0][0] * matrix[1][1] - matrix[0][1] *
matrix[1][0]);
        double[][] reverse = new double[2][2];
        reverse[0][0] = matrix[1][1] / det;
        reverse[0][1] = -matrix[0][1] / det;
        reverse[1][0] = -matrix[1][0] / det;
        reverse[1][1] = matrix[0][0] / det;
        return reverse;
    }

    private Point mult(double[][] m, Point p) {
        Point res = new Point();
        res.x = m[0][0] * p.x + m[0][1] * p.y;
        res.y = m[1][0] * p.x + m[1][1] * p.y;
        return res;
    }

    public void exec() {
        Point currentPoint = getStartPoint();
        System.out.printf("(%.9f, %.9f)\n", currentPoint.x, currentPoint.y);
        printRes(0, currentPoint, f1(currentPoint.x, currentPoint.y),
f2(currentPoint.x, currentPoint.y));
        int iter = 1;
        while (!isEnd()) {
            double[][] reverseJakoby =
getReverse(getJakobyMatrix(currentPoint));
            Point nextPoint = new Point();
            Point funcPoint = function(currentPoint);
            Point multPoint = mult(reverseJakoby, funcPoint);
            nextPoint.x = currentPoint.x - multPoint.x;
            nextPoint.y = currentPoint.y - multPoint.y;
            currentPoint = nextPoint;
            printRes(iter, currentPoint, f1(currentPoint.x, currentPoint.y),
f2(currentPoint.x, currentPoint.y));
            ++iter;
        }
    }

    private void printRes(int iter, Point cur, double f1p, double f2p) {
        System.out.printf("%d : x: %.9f\ty: %.9f\tf1: %.9f\tf2:
%.9f\n",iter,  cur.x, cur.y, f1p, f2p);
    }
}

class Point{
```

```
    Point(){}
    Point(double X, double Y){
        x = X;
        y = Y;
    }
    double x, y;
}
```

Файл Main.java

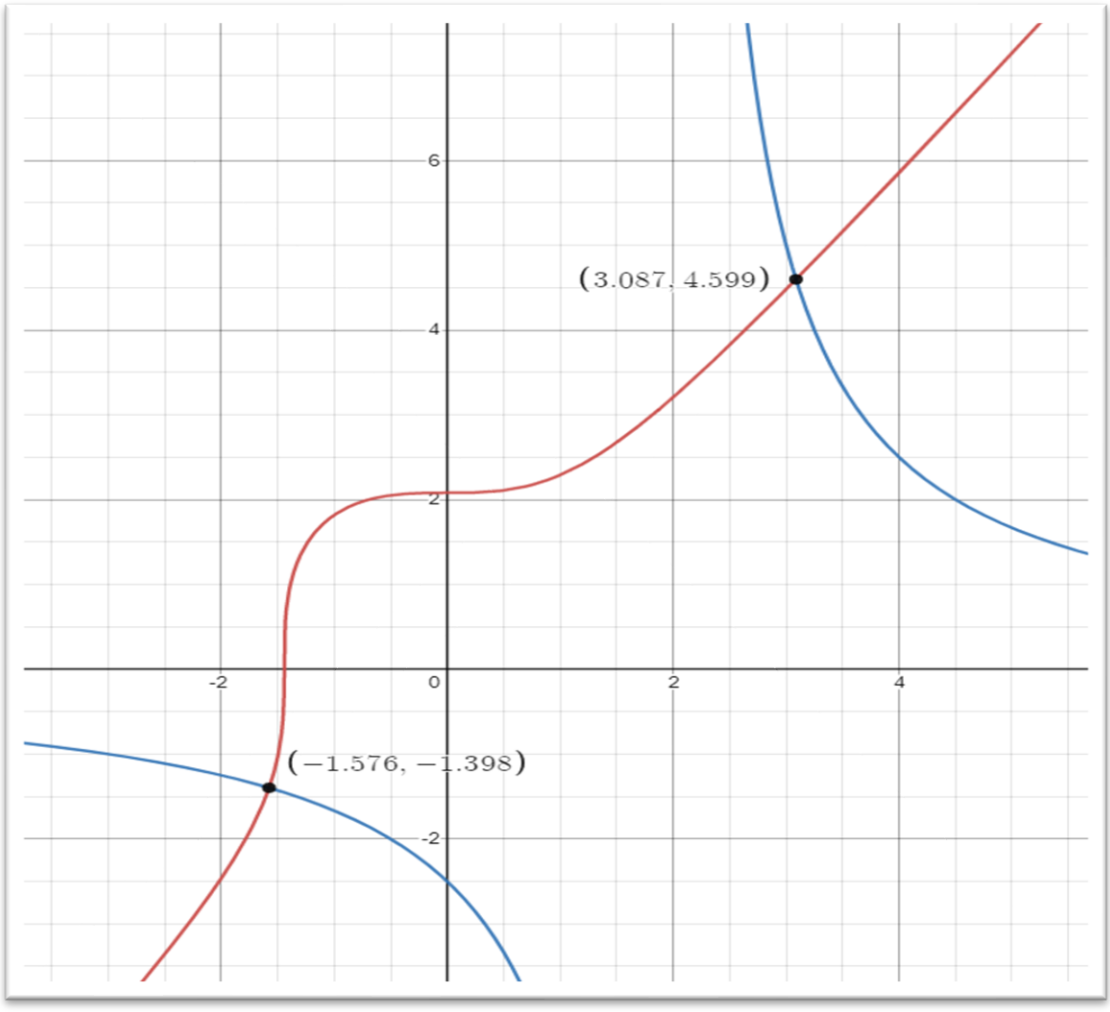
```
package second;

public class Main {
    public static void main(String[] args) {
        System.out.println("Метод линеризации");
        SimpleNuton nuton1 = new SimpleNuton() {
            int iterrations = 5;
            @Override
            boolean isEnd() {
                if(iterrations > 0){
                    --iterrations;
                    return false;
                }
                return true;
            }

            @Override
            double f1(double x, double y) {
                return 3*x*x*x-y*y*y + 9;
            }

            @Override
            double f2(double x, double y) {
                return x*y -2*y-5;
            }

            @Override
            double df1x(double x, double y) {
                return 9*x*x;
            }

            @Override
            double df1y(double x, double y) {
                return -3*y*y;
            }

            @Override
            double df2x(double x, double y) {
                return y;
            }

            @Override
            double df2y(double x, double y) {
                return x-2;
            }

            @Override
            Point getStartPoint() {
                return new Point(3,4);
            }
```

```
        };
        nuton1.exec();
        SimpleNuton nuton2 = new SimpleNuton() {
            int iterrations = 5;
            @Override
            boolean isEnd() {
                if(iterrations > 0){
                    --iterrations;
                    return false;
                }
                return true;
            }

            @Override
            double f1(double x, double y) {
                return 3*x*x*x-y*y*y + 9;
            }

            @Override
            double f2(double x, double y) {
                return x*y -2*y-5;
            }

            @Override
            double df1x(double x, double y) {
                return 9*x*x;
            }

            @Override
            double df1y(double x, double y) {
                return -3*y*y;
            }

            @Override
            double df2x(double x, double y) {
                return y;
            }

            @Override
            double df2y(double x, double y) {
                return x-2;
            }

            @Override
            Point getStartPoint() {
                return new Point(-1,-1);
            }
        };
        nuton2.exec();
    }
}
```

Вывод консоли:

```
Метод линеризации
(3,000000000, 4,000000000)
0 : x: 3,000000000  y: 4,000000000  f1: 26,000000000    f2: -1,000000000
1 : x: 3,080586081  y: 4,677655678  f1: -5,644886616    f2: 0,054609615
2 : x: 3,086884298  y: 4,599854835  f1: -0,083369674    f2: -0,000490007
3 : x: 3,087200289  y: 4,598968352  f1: -0,000008070    f2: -0,000000280
4 : x: 3,087200358  y: 4,598968318  f1: 0,000000000 f2: -0,000000000
5 : x: 3,087200358  y: 4,598968318  f1: 0,000000000 f2: 0,000000000
(-1,000000000, -1,000000000)
0 : x: -1,000000000 y: -1,000000000 f1: 7,000000000 f2: -2,000000000
1 : x: -1,900000000 y: -1,366666667 f1: -9,024370370    f2: 0,330000000
2 : x: -1,624310108 y: -1,378660560 f1: -1,236231185    f2: -0,003306595
3 : x: -1,576806613 y: -1,397642874 f1: -0,031169763    f2: -0,000901726
4 : x: -1,575602873 y: -1,398365341 f1: -0,000018369    f2: -0,000000870
5 : x: -1,575602185 y: -1,398365853 f1: -0,000000000    f2: -0,000000000
```

График:



Ответ:

$$x_1 = (3.087200358;\ 4.598968318)$$

$$x_2 = (-1.575602185; -1.398365853)$$