**Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»**

**Факультет программной инженерии и компьютерной техники**

**Вычислительная математика**

**Лабораторная работа №2**

**Вариант 8**

**Студент: Коляда Анастасия, Тажибаева Ева**

**P3266**

**Преподаватель: Машина Екатерина Алексеевна**

**Оценка: \_\_\_\_\_\_\_\_\_\_**

**Подпись преподавателя: \_\_\_\_\_\_\_\_\_\_**

1. **Цели работы**

Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

1. **Описание метода, расчётные формулы**
2. **Метод половинного деления:**

Этот метод основан на принципе "деления пополам" отрезка, на котором меняет знак функция. Начиная с отрезка, на котором функция принимает значения с разными знаками, метод находит середину этого отрезка и проверяет знак функции в этой точке. Затем метод выбирает тот подотрезок, на котором функция сохраняет знак противоположный серединной точке, и повторяет процесс для этого подотрезка. Процесс повторяется до достижения требуемой точности.

Рабочая формула метода:

1. **Метод хорд:**

В этом методе линейная аппроксимация кривой используется для приближения корня функции. Начиная с двух начальных точек на кривой, метод строит линию, соединяющую эти точки (хорда). Затем эта хорда пересекается с осью абсцисс, и новая точка становится одним из концов следующей хорды. Процесс повторяется до достижения требуемой точности.

Рабочая формула метода:

1. **Метод Ньютона:**

Этот метод использует локальную линейную аппроксимацию функции для нахождения корней. Начиная с начального приближения, метод строит касательную к кривой в этой точке и находит точку пересечения касательной с осью абсцисс. Полученная точка становится новым приближением, и процесс повторяется до достижения заданной точности.

Рабочая формула метода:

1. **Метод простых итераций:**

Этот метод преобразует уравнение в форму x = g(x), где g(x) - некоторая функция. Начиная с начального приближения, новые значения x вычисляются путем подстановки предыдущего значения x в функцию g(x). Процесс повторяется до достижения требуемой точности.

Рабочая формула метода:

**Вычислительная часть**

**Часть 1**

**Изображение выглядит как линия, График

Автоматически созданное описание**

1. Метод хорд (крайний левый корень)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № шага | a | b | x | f(a) | f(b) | f(x) |  |
| 1 | -3 | -2.5 | -2.711362 | -12.55 | 9.19 | 1.399141 | 0.5 |
| 2 | -3 | -2.711362 | -2.740313 | -12.55 | 1.399141 | 0.177845 | 0.028951 |
| 3 | -3 | -2.740313 | -2.743941 | -12.55 | 0.177845 | 0.022069 | 0.003629 |

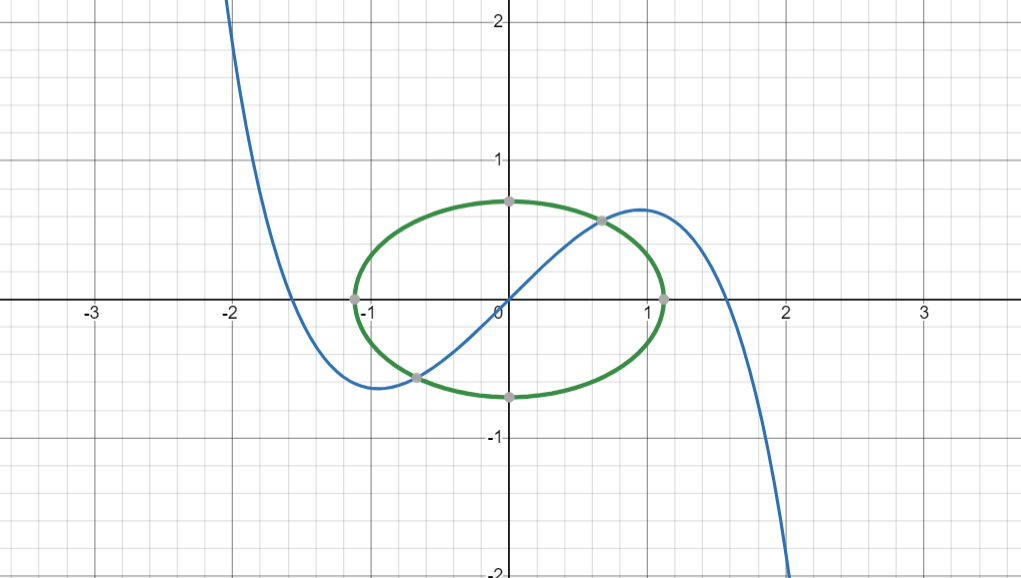
1. Метод Ньютона (центральный корень)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации |  | ) | ) |  |  |
| 1 | 0 | 6.89 | -15.42 | 0.446822 | 0.446822 |
| 2 | 0.446822 | 0.60703 | -12.103953 | 0.496974 | 0.050151 |
| 3 | 0.496974 | 0.014769 | -11.507444 | 0.498257 | 0.001283 |

1. Метод простой итерации (крайний правый корень)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № итерации |  |  | ) |  |
| 1 | 1 | 1,315527 | -3,623389 | 0,315527 |
| 2 | 0,315527 | 1,516045 | -2,126753 | 0,200517 |
| 3 | 1,516045 | 1,612581 | -0,975143 | 0,096536 |
| 4 | 1,612581 | 1,653214 | -0,400979 | 0,040634 |
| 5 | 1,653214 | 1,669357 | -0,157744 | 0,016143 |
| 6 | 1,669357 | 1,675623 | -0,060989 | 0,006266 |

**2 часть**

****

**Метод Ньютона**

1. -2x

=

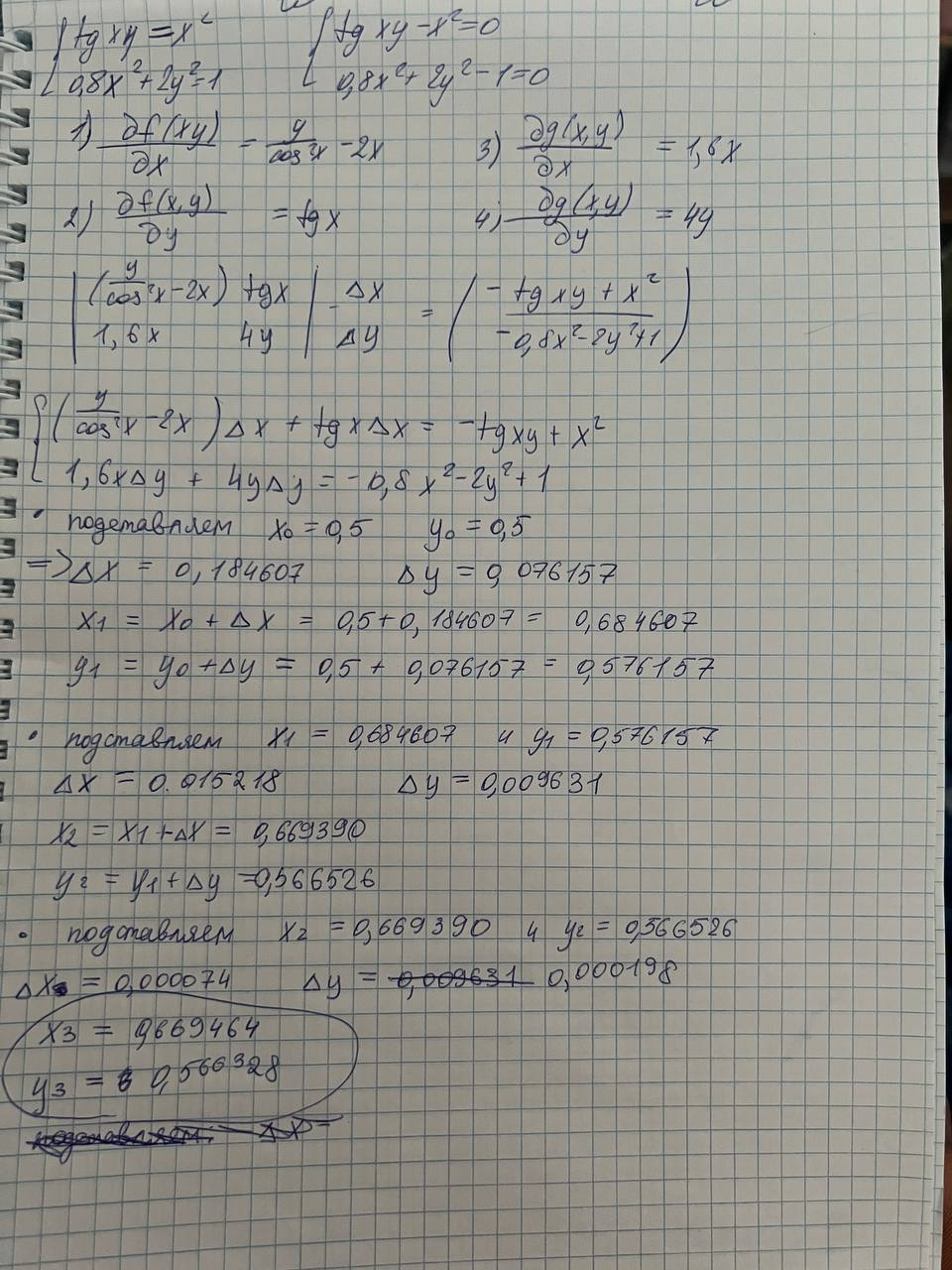
Подставляем x0=0,5, y0=0,5

/delta x = 0,184607 /delta y=0,076157

x1=x0+ /delta x = 0.5 + 0.184607 = 0.684607

y1=y0+ /delta y = 0.5 + 0.076157 = 0.0576157

получаем /delta x, /delta y



**Листинг программы**

Код написан на языке java в редакторе InteliJIIDEA. Код программы:

1. Метод половинного деления
2. protected void startCalculation(CalcParams params) {  
    //Запоминаем начальный отрезок  
    double a = params.xa;  
    double b = params.xb;  
    double x;  
    // Очищаем список итерацийи счетчик итераций  
    steps.clear();  
    \_iterations = 0;  
    do {  
    \_iterations++;  
    // вычисляем текущий х  
    x = (a + b) / 2;  
    //Вычисляем функцию от x и границ текущего отрезка  
    var f0 = \_function.functionOf(x);  
    var fa = \_function.functionOf(a);  
    var fb = \_function.functionOf(b);  
     
    //Запоминаем текущую итерацию  
    steps.add(new StepDescription(a,b,x,fa,fb,f0));  
    //Проверяем условие завершения  
    if(Math.*abs*(b-a) < params.precision){  
    break;  
    }  
    if(Math.*abs*(f0) < params.precision){  
    break;  
    }  
    //Переходим на следующую итерацию меняя или левую или правую границу,  
    // в зависимости от того, с какой стороны находится корень  
    if(f0 \* fa < 0) b = x;  
    else a = x;  
     
    } while(\_iterations < 101);  
    // Проверяем, есть ли решение  
    if(\_iterations == 101)  
    throw new CalcErrorException(String.*format*("Не смогли найти решение за %d итераций",\_iterations));  
    //Записываем найденное решение  
    \_foundX = x;  
   }

2. Метод Ньютона

private void checkConvergence(CalcParams params) {  
 if(\_function.functionOf(params.xa)\* \_function.functionOf(params.xb) >= 0)  
 throw new CalcErrorException("Не соблюдается условие сходимости - разные знаки функции на начальных границах");  
 double step = 0.05;  
 for(double x = params.xa+step; x <= params.xb;x += step){  
 if((\_function.f1Of(x) > 0) != (\_function.f1Of(x-step) > 0))  
 throw new CalcErrorException("Нет сходимости - первая производная меняет знак");  
 if((\_function.f2Of(x) > 0) != (\_function.f2Of(x-step) > 0))  
 throw new CalcErrorException("Нет сходимости - вторая производная меняет знак");  
 if(\_function.f1Of(x) == 0) throw new CalcErrorException("Нет сходиомости - первая производная равна 0");  
 }  
  
}  
  
*/\*\*  
 \* Расчет методом Ньютона  
 \* @param params параметры расчета  
 \*/*@Override  
protected void startCalculation(CalcParams params) {  
  
 double x0;  
 double x, f1x0, fx0;  
 // Выбираем начальное приближение  
 if(\_function.functionOf(params.xa) \* \_function.f2Of(params.xa) > 0) x0 =params.xa;  
 else if(\_function.functionOf(params.xb) \* \_function.f2Of(params.xb) > 0) x0 = params.xb;  
 else throw new CalcErrorException("Не определить начальное значение");  
 do {  
 fx0 = \_function.functionOf(x0);  
 f1x0 = \_function.f1Of(x0);  
 x = x0 - fx0 / f1x0;  
  
 log(String.*format*("Xn %f fXn %f f'Xn %f Xn+1 %f |Xn+1 - Xn| %f\n",  
 x0, fx0, f1x0, x, Math.*abs*(x - x0)));  
  
 if (Math.*abs*(x - x0) < params.precision) break;  
 if (Math.*abs*(fx0 / f1x0) < params.precision) break;  
 if (Math.*abs*(fx0) < params.precision) break;  
  
 x0 = x;  
  
 } while (true);  
 \_foundX = x;  
}

1. Метод простой итерации
2. protected void startCalculation(CalcParams params) {  
    //Запрашиваем подходящую эквивалентную функию - функция имеет возможность  
    // передать эквивалентную функцию методу на проверку годности,  
    // для этого методо передается первым параметром  
    IFunction phi = \_function.getPhiFunction(this,params.xa,params.xb);  
    //Проверяем условие сходимости полученной эквивалентной функции  
    if (Math.*abs*(phi.f1Of(params.xa)) >= 1 || Math.*abs*(phi.f1Of(params.xb)) >= 1)  
    throw new CalcErrorException(String.*format*("Не соблюдается условие сходимости на отрезке %f %f",  
    params.xa, params.xb));  
     
    //Устанавливаем начальное приближение  
    double x0 = params.xa;  
    double x,fix,fx;  
    //Вычисляем функцию от начального приближения  
    fix = phi.functionOf(x0);  
    int counter = 0;  
    do{  
    //Далее вычисляем следующий x от эквивалентной функции  
    x = fix;  
    fix = phi.functionOf(x);  
    fx = \_function.functionOf(x); //Значение самой функции нам надо для проверки на то,  
    // что решение удовлетворяет точности  
    log(String.*format*("Xi %f Xi+1 %f Phi(Xi+1) %f f(Xi+1) %f |Xi+1 - Xi| %f)\n",  
    x0,x,fix,fx,Math.*abs*(x - x0)));  
    if(Math.*abs*(fx) < params.precision) break;  
    if(Math.*abs*(x - x0) < params.precision) break;  
    x0 = x;  
    }while(++counter < 100);  
     
    if(counter == 100){  
    throw new CalcErrorException("За 100 итераций результат не определен!");  
    }  
    \_foundX = x;  
   }
3. Метод простой итерации (для системы)

public void Calculate(ISysFunctions funcs, CalcParams params) {  
 *checkParams*(params);  
 *checkConverge*(funcs, params);  
 double x0 = params.iaX;  
 double y0 = params.iaY;  
 ISysFunction f1 = (ISysFunction)funcs.getSysFunctions()[0];  
 ISysFunction f2 = (ISysFunction)funcs.getSysFunctions()[1];  
  
 int count = 0;  
 double x,y;  
 do {  
 count++;  
 x = f1.phi(x0,y0);  
 y = f2.phi(x0,y0);  
  
 log(String.*format*("Xi %f Xi+1 %f |Xi+1 - Xi| %f Yi %f Yi+1 %f |Yi+1 - Yi| %f\n",  
 x0,x,Math.*abs*(x-x0),y0,y,Math.*abs*(y-y0)));  
  
 if(Math.*abs*(x -x0) <= params.precision && Math.*abs*(y - y0) <= params.precision) break;  
 y0=y;  
 x0=x;  
  
 } while(count <=100);  
 if(count > 100) throw new CalcErrorException("Количество итераций превысило 100");  
 log("Результат расчета методом простых итераций:\n");  
 log(String.*format*("Количество итераций: %d\n",count));  
 log(String.*format*("Вектор решений: (%f,%f)\n",x,y));  
 log(String.*format*("Вектор погрешностей: (%f,%f)\n",Math.*abs*(x-x0),Math.*abs*(y-y0)));  
 log("Проверочный расчет уравнений:\n");  
 log(String.*format*("Уравнение %s при x=%f,y=%f результат %f\n",f1.toString(),x,y,f1.originalFunc(x,y)));  
 log(String.*format*("Уравнение %s при x=%f,y=%f результат %f\n",f2.toString(),x,y,f2.originalFunc(x,y)));  
  
}  
  
*/\*\*  
 \* Проверяет введенные параметры  
 \* @param params  
 \*/*private static void checkParams(CalcParams params){  
 if(params.precision <= 0) throw new CalcErrorException("Недопустимое значение требуемой точности");  
 if(params.asMaxX <= params.asMinX) throw new CalcErrorException("Недопустимые значения области решения по х");  
 if(params.asMaxY <= params.asMinY) throw new CalcErrorException("Недопустимое значение области решения по y");  
  
}  
  
*/\*\*  
 \* Проверяет условие сходимости  
 \* @param funcs  
 \* @param params  
 \*/*private static void checkConverge(ISysFunctions funcs,CalcParams params){  
 double step = 0.1;  
 ISysFunction f1 = (ISysFunction)funcs.getSysFunctions()[0];  
 ISysFunction f2 = (ISysFunction)funcs.getSysFunctions()[1];  
 for(double x = params.asMinX; x <= params.asMaxX; x += step){  
 for(double y = params.asMinY; y <= params.asMaxY; y+= step){  
 if((Math.*abs*(f1.phi1X(x,y)) + Math.*abs*(f1.phi1Y(x,y))) >= 1)  
 throw new CalcErrorException("На указанном отрезке решений не соблюдается условие сходимости.");  
 }  
 }  
}

1. **Примеры и результаты работы программы**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, диаграмма

Автоматически созданное описание**

1. **Вывод**

В данной лабораторной работе мы изучитли численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов половинного деления, итераций и Ньютона.