**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по практической работе №4**

**по дисциплине «Вычислительная математика»**

**Тема: Метод Ньютона и метод простых итераций.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1304 |  | Чернякова В.А. |
| Преподаватель |  | Попова Е.В. |

Санкт-Петербург

2022

## Цель работы.

## Целью работы является нахождение корня уравнения f(x)=0 методами Ньютона и простых итераций с заданной точностью Eps, исследование скорости сходимости и обусловленности метода.

## Основные теоретические положения.

**Метод Ньютона.** В случае, когда известно хорошее начальное приближение решения уравнения , эффективным методом повышения точности является метод Ньютона. Он состоит в построении итерационной последовательности (1)

(1)

где Последовательность сходится к корню уравнения . По теореме о сходимости метода Ньютона должен быть простым корнем уравнения в отсекающем промежутке этого корня функция – дважды непрерывно дифференцируема и

Для оценки погрешности n-го приближения корня предлагается пользоваться неравенством (2)

(2)

где - наибольшее значение модуля второй производной на отрезке [a,b]; - наименьшее значение модуля первой производной на отрезке [a,b]. Таким образом, если , то . Это означает, что при хорошем начальном приближении корня после каждой итерации число верных десятичных знаков в очередном приближении удваивается, т.е. процесс сходится очень быстро и имеет место квадратическая сходимость.

Если необходимо найти корень с точностью ε, то итерационный процесс можно прекращать, когда выполняется неравенство (3)

(3)

Если на (n-1)-м шаге очередное приближение не удовлетворяет условию окончания процесса, то вычисляются величины и следующие приближение корня . При выполнении условия (3) величина принимается за приближенное значение корня с, вычисленное с точностью ε.

**Метод простых итераций.** Метод *простых итераций* решения уравнения заключается в замене исходного уравнения эквивалентным ему уравнением и построении последовательности , сходящейся при к точному решению

Корень уравнения является точкой пересечения двух графиков и . Сходимость метода зависит от вида функции . В зависимости от величины модуля первой производной метод может сходиться и расходиться.

Достаточные условия сходимости метода *простых итераций* формулируются следующей теоремой:

Если функция определена, дифференцируема и принадлежит отрезку , то существует число, такое что на , и последовательность,сходится к единственному решению на уравнения при (1)

(1)

Если , то |, если , то |.

Рассмотрим один шаг итерационного процесса. Исходя из найденного на предыдущем шаге значения , вычисляется . Если , то полагается и выполняется очередная итерация. Если же , то вычисления заканчиваются и за приближенное значение корня принимается величина . Погрешность результата вычислений зависит от знака производной : при : погрешность определения корня составляет , а при , погрешность не превышает . Здесь - число, такое, что на отрезке [a,b]. Существование числа является условием сходимости метода в соответствии с отмеченной выше теоремой.

Для применения метода *простых итераций* определяющее значение имеет выбор функции , в уравнении , эквивалентном исходному. Функцию необходимо подбирать так, чтобы . Это обусловливается тем, что если , на отрезке [a,b], то последовательные приближения . будут колебаться около корня *c*, если же , то последовательные приближения будут сходиться к корню *c* монотонно.

Число обусловленности метода *простых итераций* (2)

## Постановка задачи.

**Метод Ньютона.** В практической работе предлагается, использовать программы - функции NEWTON (листинг 4) и Round (листинг 1).

Листинг 4 – Функция NEWTON

double NEWTON(double X, double Eps, int& N) {

// extern double F1(double);

double Y, Y1, DX, Eps0;

N = 0;

double m1 = 3.29; // наименьшее значение модуля 1-ой производной

double M2 = 16.424; // наибольшее значение модуля 2-ой производной

Eps0 = sqrt(2 \* m1 \* Eps / M2);

do {Y = F(X);

if (Y == 0.0) {

return X;}

Y1 = F1(X);

if (Y1 == 0.0) {

puts("Производная обратилась в ноль\n");

exit(1);}

DX = Y / Y1;

X -= DX;

N++;

} while (fabs(DX) >= Eps0);

return X;}

**Метод простых итераций.** В практической работе предлагается использовать программы функции ITER и PHI (листинг 5).

Листинг 5 - функции ITER и PHI

double ITER (double X0, double Eps, int& N, double a, double b) {

if (Eps <= 0.0) {

puts("Неверное задание точности\n");

exit(1); }

double X1 = PHI(X0, a, b);

double X2 = PHI(X1, a, b);

for (N = 2; (X1 - X2) \* (X1 - X2) > fabs((2 \* X1 - X0 - X2) \* Eps); N++) {

X0 = X1;

X1 = X2;

X2 = PHI(X1, a, b); }

return X2; }

double PHI(double x, double a, double b) {

if (x == 0) {

printf("деление на 0!");

exit(1); }

double min = min\_F1(a, b);

double max = max\_F1(a, b);

double s = x - 2 / (min + max) \* (F(x));

s = Round(s, delta);

return(s); }

## Этапы выполнения практической работы.

**Метод Ньютона.**

**I часть**:

1. Графически или аналитически отделить корень уравнения (найти отрезки [Left, Right], на которых функция . удовлетворяет условиям теоремы о сходимости метода Ньютона).//см лаба 1-3
2. Проверить функцию на выпуклость вверх или вниз, выбрать начальное приближение корня так чтобы.
3. Получить аналитическое выражение функций и Оценить снизу величину оценить сверху величину
4. По заданному Eps сосчитать условие окончания итерационного процесса Eps2=
5. Составить подпрограммы-функции вычисления , , предусмотрев округление их значений с заданной точностью Delta.
6. Составить головную программу, вычисляющую корень уравнения и содержащую обращение к подпрограммам , , Round, NEWTON и индикацию результатов.
7. Провести вычисления по программе. Исследовать скорость сходимости метода и чувствительность метода к ошибкам в исходных данных.

**II часть:**

(Аналогия с бисекцией): посчитать Xn и Yn с «чистой» функцией, внести возмущения в f(x), изменить условие цикла до 80 итераций.

**Новое:** подсчитать «разболтку» в столбце Е. Как только вычисления попадают в промежуток неопределенности. Определить номер, с которого начинается «разболтка». **на входе y, на выходе х**. Поэтому eps должен быть связан с х, delta c y (проверить код программы). В столбце I считаем и сравниваем с .

**Метод простых итераций.**

1) Графически или аналитически отделить корень уравнения и получить отрезок [Left, Right].

2) Сосчитать найти на отрезке [Left, Right] – минимальное значение – максимальное значение .

3) Преобразовать уравнение к виду, удобному для итераций .

3) Выбрать начальное приближение , лежащее на [Left, Right].

4) Составить подпрограммы для вычисления значений ,, предусмотрев округление вычисленных значений с точностью Delta.

5) Составить головную программу, вычисляющую корень уравнения и содержащую обращение к программам , PHI, ITER и индикацию результатов.

6) Провести вычисления по программе. Исследовать скорость сходимости и обусловленность метода.

Сделать сравнительную таблицу с результатами последних 4 методов.

## Выполнение работы.

**Вариант 28.** *f(x) = 1/cos(cx) + 1*

**Метод Ньютона.**

*Часть 1.*

1. Функция *f(x) = 1/cos(cx) + 1* тригонометрическая и, следовательно,

является периодической, то есть значения функции через регулярные промежутки времени повторяются.

В область допустимых значений данной функции не входят значения,

принадлежащие интервалу *(π/2c + πn/c),* где *n* – целое число, так как функция *cos(cx)* принимает значение равное 0.

При *с*, принимающем значения от 3 до 3.5, корень находится на отрезке [3; 4].

Пример графика функции при *с* = 3 см. рис. 1. Корни , где *n*

целое число.

Для дальнейшего выполнение практической работы зафиксируем значение с = 3 и сделаем отрезок локализации более точным, уменьшив его, [3.5;4].

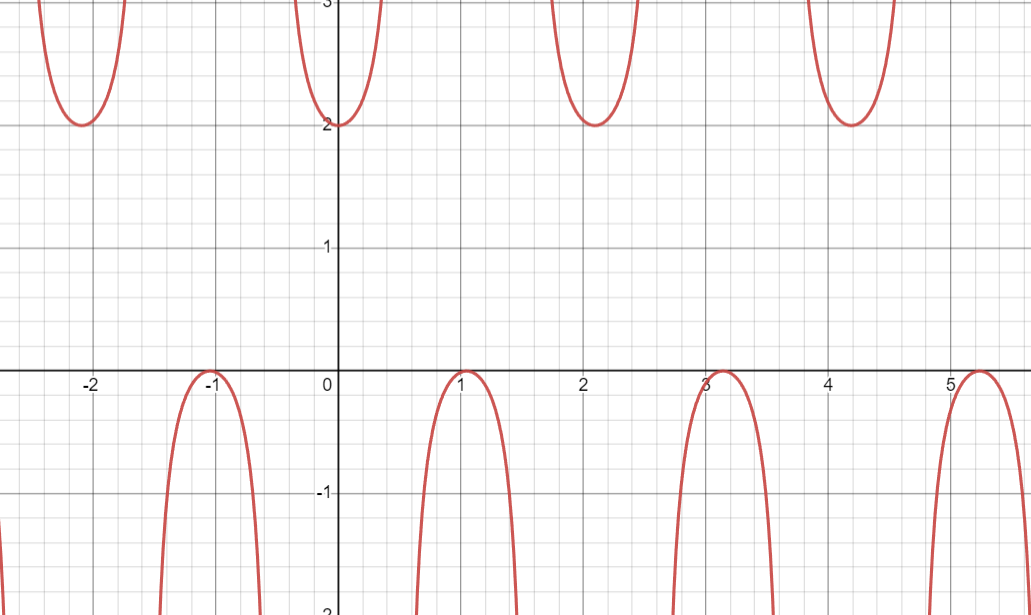
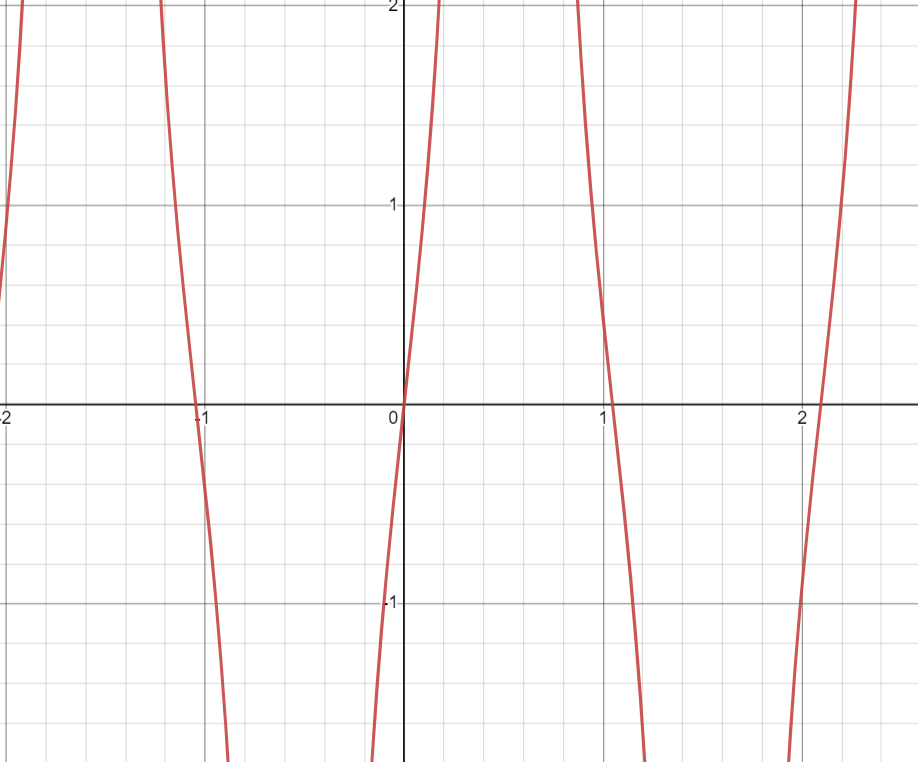


Рисунок 1 – график функции f(x) = 1/cos(cx) + 1 при с = 3.

1. Проверим функцию на выпуклость вверх или вниз и выберем начальное приближение корня так чтобы.



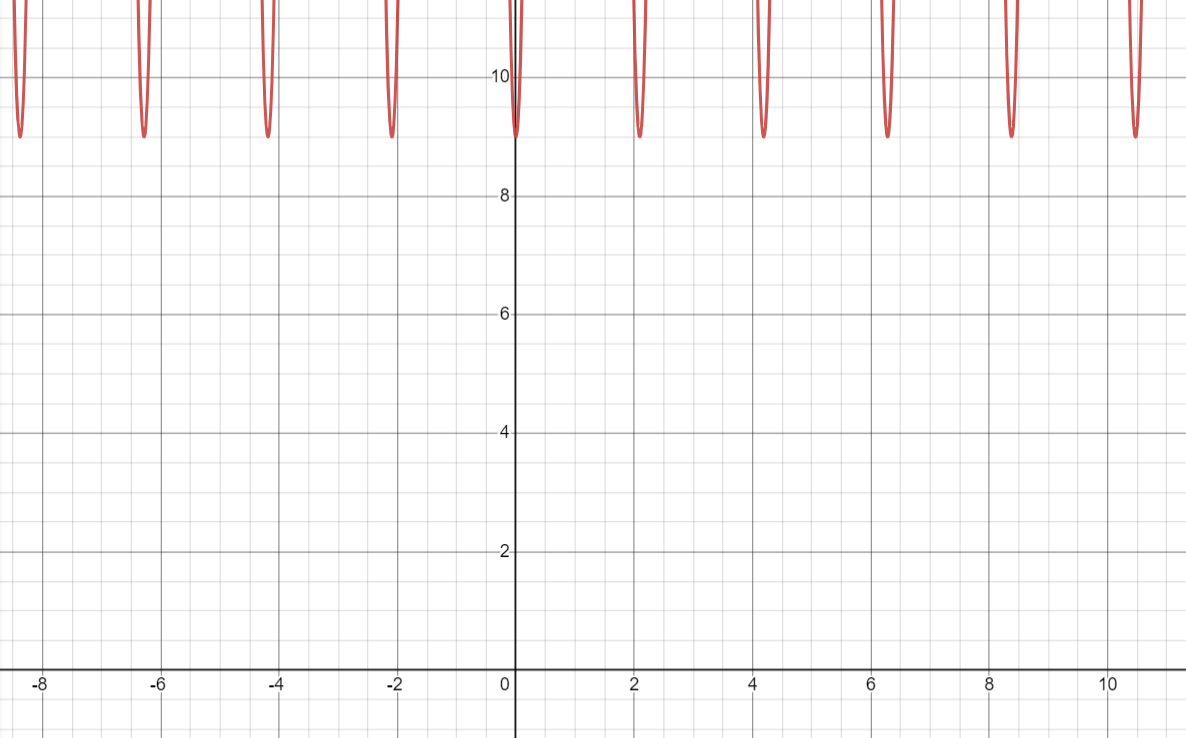
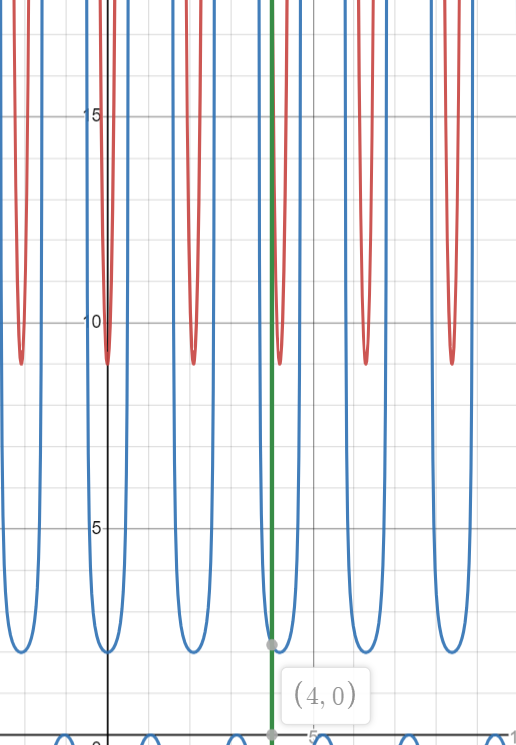
Рисунок 2 – график первой производной функции f(x) = 1/cos(cx) + 1 при с = 3.

Рисунок 3 – график второй производной функции f(x) = 1/cos(cx) + 1 при с = 3.

Рисунок 4 – график функции f(x) = 1/cos(cx) + 1, ее второй производной и прямой x = 4

Как происходит поиск x0:

* x0 принадлежит [a, b] => на данном этапе предположим, что x0 любое число от 3,5 до 4 согласно выбранному отрезку локализации
* также должно выполняться условие . Напишем функцию searchx0(double a, double b)

void searchx0(double a, double b){

for (double i = a; i <=b; i+=0.1){

if (F(i)\*SecondDerivative(i) > 0)

std::cout<<i<<endl;

}

}

Рисунок 5 – результат работы функции по поиску x0.

Таким образом выбираем значение x0 = 4

1. Найдем и оценим снизу величину сверху величину

Для нахождения m и M были написаны функции m\_min и M\_maxNEWTON. Обе функции принимают границы отрезка и шаг для вычислений. Внутри функций в цикле находится для m – первая производная, для M – вторая производная и сравнивается со значением, хранящимся в переменной min или max соответственно. Если полученное значение меньше/больше, то оно записывается в переменную. Функция возвращает min или max соответственно.

Функция вычисления m:

*double m\_min(double a, double b, double delta){*

*double min\_ = fabs(Derivative(a));*

*for (double i = a; i <= b ;){*

*double elem = fabs(Derivative(i));*

*if (elem < min\_){*

*min\_ = elem;*

*}*

*i += delta;*

*}*

*return min\_;*

*}*

Функция вычисления M:

*double M\_maxNEWTON(double a, double b, double delta){*

*double max\_ = fabs(SecondDerivative(a));*

*for (double i = a; i <= b;){*

*double elem = fabs(SecondDerivative(i));*

*if(elem > max\_ && elem != INFINITY ){*

*max\_ = elem;*

*}*

*i += delta;*

*}*

*return max\_;*

*}*

При c = 3, eps = 0.1, delta = 0.1, x0 = 4, a = 3.5, b = 4 значения получаются следующие:

m = 2.26056

M = 15807.1

1. По заданному Eps сосчитаем условие окончания итерационного процесса Eps2=

В теле функции данное значение будет рассчитываться следующим образом:

*Eps0 = sqrt(2 \* m1 \* Eps / M2),*

где m1 = *m\_min(3.5, 4, delta),* а M2 = *M\_maxNEWTON(3.5, 4, delta).*

При eps = 0.1 и delta = 0.1 значение получается следующее:

Eps0 = 0.00534806

1. Составлены подпрограммы вычисления f(x) и f’(x)

f(x):

*double F(double x){*

*return (1/(cos(c\*x)))+1;*

*}*

f’(x):

*double Derivative(double x) {*

*return (c\*sin(c\*x))/(pow(cos(c\*x),2));*

*}*

1. Составлена головная программа, вычисляющая корень уравнения и содержащая обращение к подпрограммам , , Round, NEWTON и индикацию результатов. Разработанный код смотри в приложении.
2. Проведение вычислений по программе:

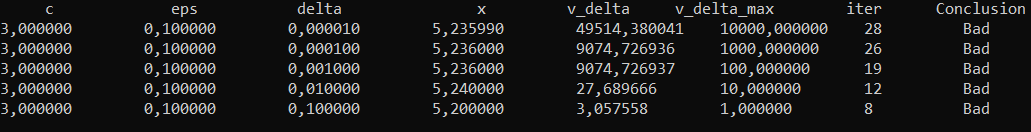
c = 3, eps = 0.1, delta варьируется от 0.00001 до 0.1.

Рисунок 5 – результат работы программы.

Вывод: обусловленность всегда плохая.

*Часть 2.*

Для вычисления функции с учетом возмущения была написана функция Round\_Truncation().

*double Round\_Truncation(double Y) {*

*if (Y >= 0) {*

*return floor(Y);//round down*

*} else {*

*return ceil(Y);//round up*

*}*

*}*

Возмущение 1 до 3 знаков после запятой.

Возмущение 2 до 2 знаков после запятой.

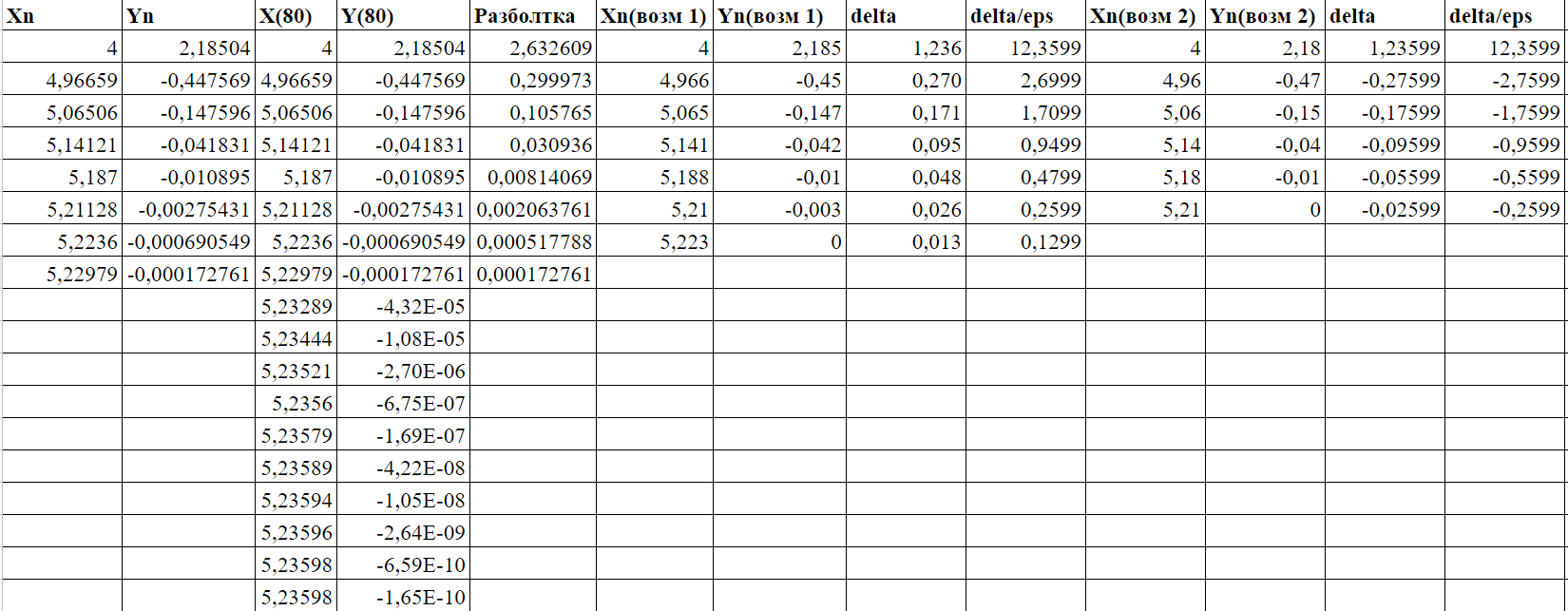
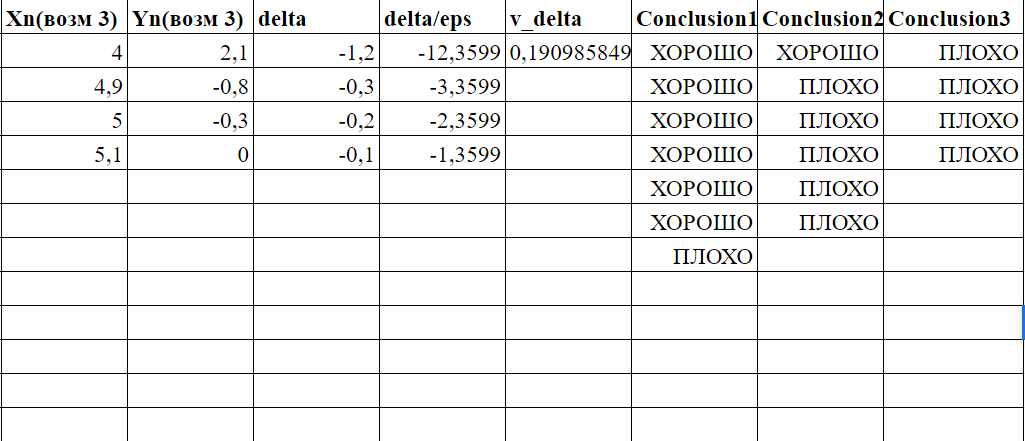
Возмущение 3 до 1 знаков после запятой.

Таблица 1 – вычисления с учетом возмущений.

Вывод: Функция с первым возмущением почти всегда является хорошо обусловленной. Функция с учетом второго возмущения со 2 итерации является плохо обусловленной. При третьем возмущении функция всегда плохо обусловлена. «Разболтка» начинается с первого номера.

**Метод простых итераций.**

1. Функция *f(x) = 1/cos(cx) + 1* тригонометрическая и, следовательно,

является периодической, то есть значения функции через регулярные промежутки времени повторяются.

В область допустимых значений данной функции не входят значения,

принадлежащие интервалу *(π/2c + πn/c),* где *n* – целое число, так как функция *cos(cx)* принимает значение равное 0.

При *с*, принимающем значения от 3 до 3.5, корень находится на отрезке [3; 4].

Пример графика функции при *с* = 3 см. рис. 1. Корни , где *n*

целое число.

Для дальнейшего выполнение практической работы зафиксируем значение с = 3 и сделаем отрезок локализации более точным, уменьшив его, [3.5;4].

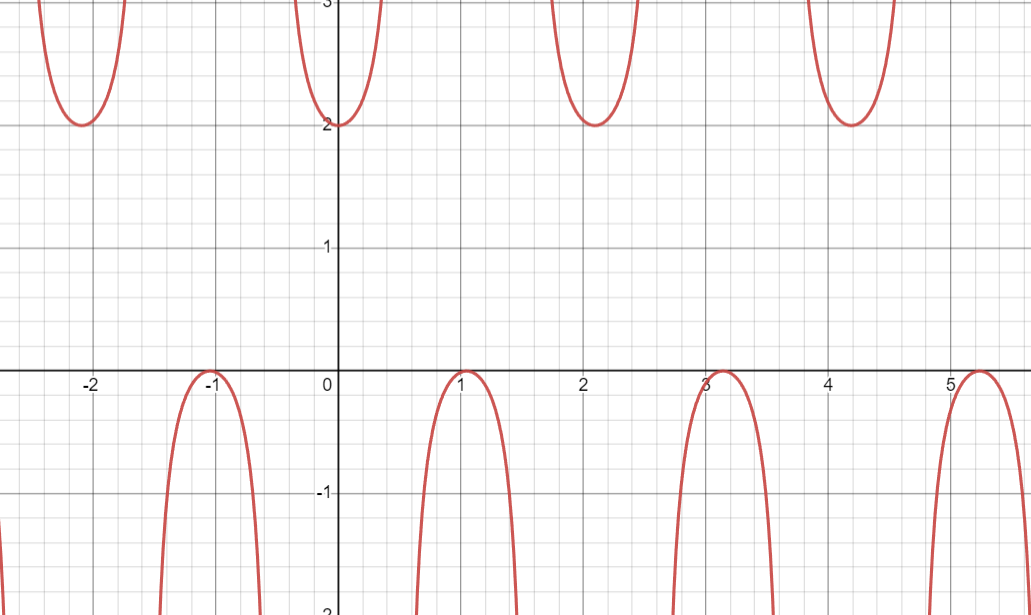
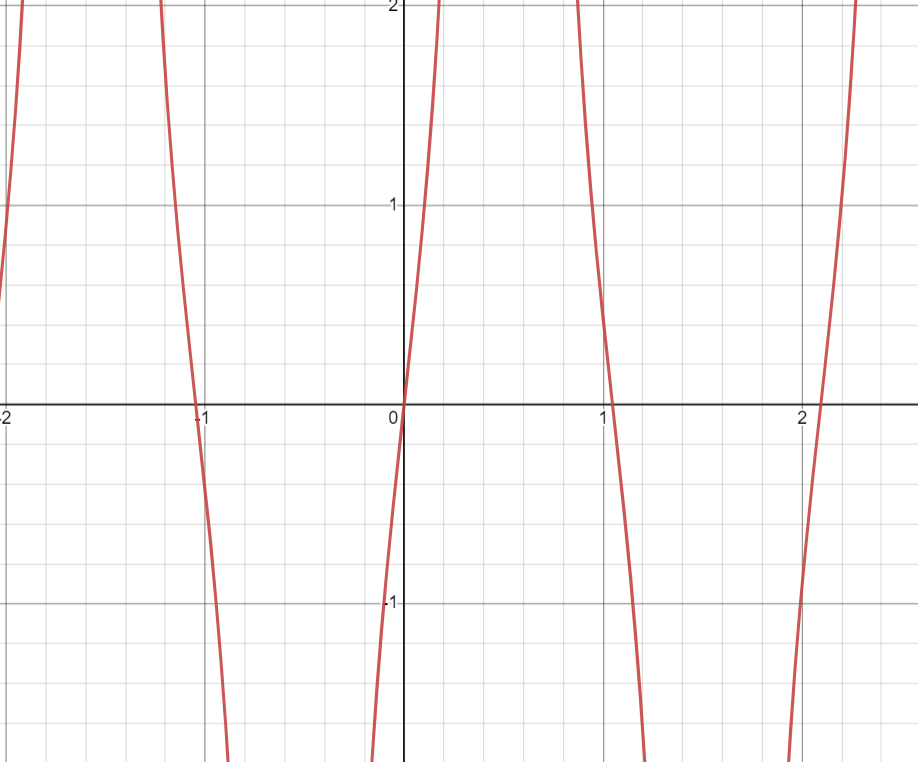


Рисунок 6 – график функции f(x) = 1/cos(cx) + 1 при с = 3.

1. Сосчитаем

f’(x) = csin(cx)/cos*2*(cx)

Рисунок 7 – график первой производной функции f(x) = 1/cos(cx) + 1 при с = 3.

Была составлена подпрограмма для вычисления f’(x):

*double Derivative(double x) {*

*return (c\*sin(c\*x))/(pow(cos(c\*x),2));*

*}*

Для нахождения m и M были написаны функции m\_min и M\_max. Обе функции принимают границы отрезка и шаг для вычислений. Внутри функций в цикле находится для m и M – первая производная и сравнивается со значением, хранящимся в переменной min или max соответственно. Если полученное значение меньше/больше, то оно записывается в переменную. Функция возвращает min или max соответственно.

Функция вычисления m:

*double m\_min(double a, double b, double delta){*

*double min\_ = fabs(Derivative(a));*

*for (double i = a; i <= b ;){*

*double elem = fabs(Derivative(i));*

*if (elem < min\_){*

*min\_ = elem;*

*}*

*i += delta;*

*}*

*return min\_;*

*}*

Функция вычисления M:

*double M\_max(double a, double b, double delta){*

*double max\_ = fabs(Derivative(3, a));*

*for (double i = a; i <= b;){*

*double elem = fabs(Derivative(3, i));*

*if(elem > max\_ && elem != INFINITY ){*

*max\_ = elem;*

*}*

*i += delta;*

*}*

*return max\_;*

*}*

При c = 3, eps = 0,1, delta = 0,1, x0 = 4, a = 3, b = 4 значения следующие:

m = 0.379237

M = 274.608

1. Преобразуем уравнение к виду, удобному для итераций .

Таким образом +1).

1. Найдем x0

Так как по условию ∀ x0 принадлежит [a,b]. Тогда пусть x0 = 4.

1. Составим подпрограмму для вычисления

Функция для вычисления

*double fi(double x,double delta){*

*return x-(2/(m\_min(3,4,delta)+ M\_max(3,4,delta)))\* F(x);*

*}*

Так как =>

1 – 2csin(cx)/(m+M)cos*2*(cx)

Функция:

*double fi\_(double x,double delta){*

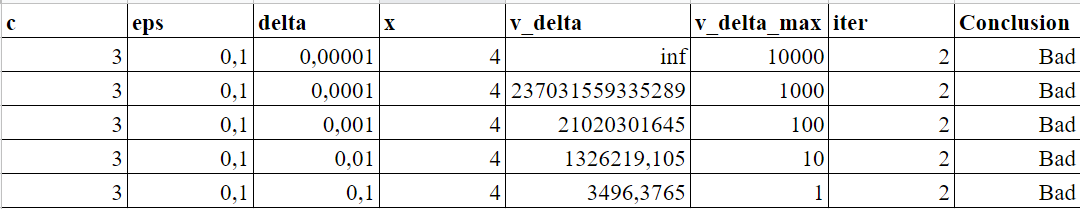
*return 1-(2/(m\_min(3,4,delta)+ M\_max(3,4,delta)))\* Derivative(x);*

*}*

1. Составлена головная программа, вычисляющая корень уравнения и содержащую обращение к программам , PHI, ITER и индикацию результатов.
2. Проведем вычисление по программе.

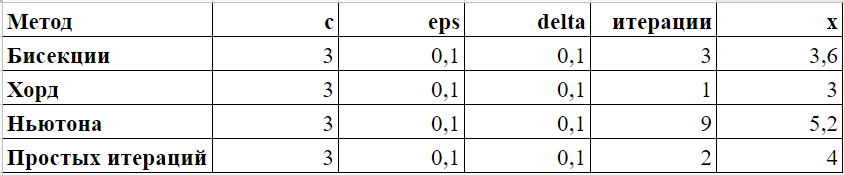
c = 3, eps = 0,1, delta варьируется от 0,00001 до 0,1.

Число обусловленности для данного метода рассчитывается по следующей формуле:

Таблица 2 – результат работы программы.

Вывод: функция всегда плохо обусловлена.

**Сравнение методов.**

Таблица 3 – сравнение методов.

Вывод: метод хорд осуществляет работу за самое наименьшее число итераций, метод Ньютона – за наибольшее.

# приложение

#include <iostream>

#include <cmath>

using namespace std;

double delta, eps, c;

int n;

double Round\_Truncation(double Y) {

if (Y >= 0) {

return floor(Y);//round down

} else {

return ceil(Y);//round up

}

}

double F(double x){

return (1/(cos(c\*x)))+1;

}

double Derivative(double x) {

return (c\*sin(c\*x))/(pow(cos(c\*x),2));

}

double SecondDerivative(double x) {

return (2\*pow(c,2)\*pow(sin(c\*x),2)+pow(c,2)\*pow(cos(c\*x),2))/(pow(cos(c\*x),3));

}

double Round(double X, double Delta) {

if (Delta <= 1E-9){

puts("Неверно задана точность округления\n");

exit(1);

}

if (X>0.0)

return (Delta\*(long((X / Delta) + 0.5)));

else

return (Delta\*(long((X / Delta) - 0.5)));

}

double m\_min(double a, double b, double delta){

double min\_ = fabs(Derivative(a));

for (double i = a; i <= b ;){

double elem = fabs(Derivative(i));

if (elem < min\_){

min\_ = elem;

}

i += delta;

}

return min\_;

}

double M\_max(double a, double b, double delta){

double max\_ = fabs(Derivative(a));

for (double i = a; i <= b;){

double elem = fabs(Derivative(i));

if(elem > max\_ && elem != INFINITY ){

max\_ = elem;

}

i += delta;

}

return max\_;

}

double NEWTON(double X, double Eps, double delta, int& N) {

// extern double F1(double);

double Y, Y1, DX, Eps0;

N = 0;

double m1 = m\_min(3, 4, delta); // наименьшее значение модуля 1-ой производной

double M2 = M\_max(3, 4, delta); // наибольшее значение модуля 2-ой производной

Eps0 = sqrt(2 \* m1 \* Eps / M2);

do {

Y = F(X);

//std::cout<<"X"<<N<<": "<<Round\_Truncation(X\*1000)/1000<<" f(X): "<<Round\_Truncation(F(X)\*1000)/1000<<"\n";

//std::cout<<"X"<<N<<": "<<Round\_Truncation(X\*100)/100<<" f(X) :"<<Round\_Truncation(F(X)\*100)/100<<"\n";

//std::cout<<"X"<<N<<": "<<Round\_Truncation(X\*10)/10<<" F(X): "<<Round\_Truncation(F(x)\*10)/10<<"\n";

//std::cout<<"X"<<N<<": "<<X<<" F(X): "<<Y<<"\n";

if (Y == 0.0) {

return X;

}

Y1 = Derivative(X);

if (Y1 == 0.0) {

puts("Производная обратилась в ноль\n");

exit(1);

}

DX = Y / Y1;

X -= DX;

N++;

} //while (N<80);

while (fabs(DX) >= Eps0);

//std::cout<<"X"<<N<<": "<<X<<" F(X): "<<Y<<"\n";

return X;

}

double fi(double x,double delta){

return x-(2/(m\_min(3,4,delta)+ M\_max(3,4,delta)))\* F(x);

}

double fi\_(double x,double delta){

return 1-(2/(m\_min(3,4,delta)+ M\_max(3,4,delta)))\* Derivative(x);

}

double PHI(double x, double a, double b, double delta) {

if (x == 0) {

printf("деление на 0!");

exit(1);

}

double min = m\_min(a, b, delta);

//std::cout << min << endl;

double max = M\_max(a, b, delta);

//std::cout<<max;

double s = x - 2 / (min + max) \* (F(x));

s = Round(s, delta);

return(s);

}

double ITER (double X0, double Eps, int& N, double a, double b, double delta) {

if (Eps <= 0.0) {

puts("Неверное задание точности\n");

exit(1);

}

double X1 = PHI(X0, a, b, delta);

double X2 = PHI(X1, a, b, delta);

for (N = 2; (X1 - X2) \* (X1 - X2) > fabs((2 \* X1 - X0 - X2) \* Eps); N++) {

X0 = X1;

X1 = X2;

X2 = PHI(X1, a, b, delta);

}

return X2;

}

int main(){

setlocale(LC\_ALL, "Russian");

c = 3;

eps = 0.1;

printf("%6s\t%14s\t%14s\t%14s\t%14s\t%14s\t%10s\t%10s\n", "c", "eps", "delta", "x", "v\_delta", "v\_delta\_max", "iter", "Conclusion");

for(delta = 0.00001; delta <= 0.1; delta \*= 10){

double x = Round(NEWTON(4, eps, delta, n), delta);

double derivative = fabs(Derivative(x));//вычисление абсолютного значения

double v = 1/derivative;

printf("%6f\t%6f\t%6f\t%6f\t%6lf\t%6f\t%d\t", c, eps, delta, x, v, eps/delta, n);

if(v < eps/delta)

printf("%6s\n", "Good");

else

printf("%6s\n", "Bad");

}

//delta = 0.1;

//NEWTON(4, eps, delta, n);

printf("%6s %14s %14s %14s %14s %14s %10s %10s\n", "c", "eps", "delta", "x", "v\_delta", "v\_delta\_max", "iter", "Conclusion");

for(delta = 0.00001; delta <= 0.1; delta \*= 10){

double x = Round(ITER(4, eps, n, 3, 4, delta), delta);

double derivative = fabs(1 - fi\_(x, delta));//вычисление абсолютного значения

double v = 1/derivative;

printf("%6f %6f %6f %6f %6lf %6f %d ", c, eps, delta, x, v, eps/delta, n);

if(v < eps/delta)

printf("%6s\n", "Good");

else

printf("%6s\n", "Bad");

}

//delta = 0.1;

//ITER(4, eps, n, 3, 4, delta);

return 0;

}