Задание №3. Метод Ньютона

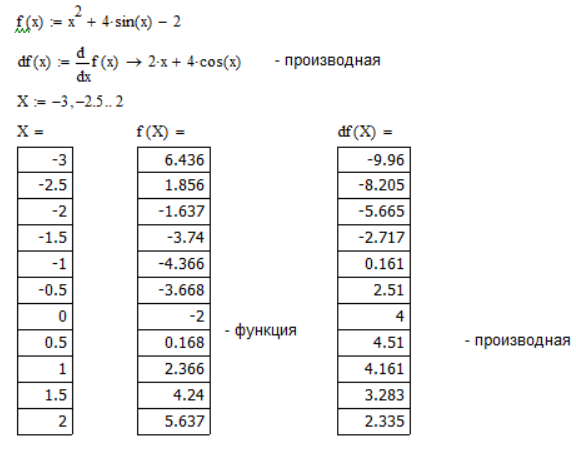
Вариант №10

***Цель задания***: *практическое освоение метода Ньютона для решения нелинейных уравнений и систем.*

1. **Программная реализация метода Ньютона для решения нелинейных уравнений:**

* Локализовать один любой корень уравнения из *Приложения №1* (номер варианта = номер в списке группы) методом последовательного перебора (определить начальный интервал локализации [a0, b0]).
* Реализовать метод Ньютона (в связке с методом половинного деления или методом хорд) для уточнения корня на выбранном интервале локализации [a0, b0] с точностью *e* = 10-4.
* Для тестирования реализованного метода выбрать начальное приближение хо из интервала локализации [a0, b0] такое, что x1 (или любое другое приближение) «вылетает» из текущего интервала локализации [ak, bk].
* Интервал [ak, bk] пересчитывать для каждой k-й итерации, независимо от того, вызывался ли вспомогательный метод (половинного деления или хорд) или же работал только «чистый» метод Ньютона.

Отделим корни (в Mathcad):



Будем искать положительный корень на отрезке [0; 1].

Метод Ньютона (касательных)

Пока не выполнено условие остановки, в качестве которого можно взять

(то есть погрешность в нужных пределах), вычисляем новое приближение:

Производную можно вычислить аналитически:

Метод хорд

Итерационная формула имеет вид:

Программа на С++:

#include <iostream>

#include <cmath>

#include<iomanip>

#include<conio.h>

using namespace std;

int k = 0;

float f(float x) //функция по которой вычисляем нужные нам корни

{

float y;

y = x\*x+4\*sin(x)-2;

return y;

}

float f1(float x) { //первая производная

float z;

z = 2\*x+4\*cos(x);

return z;

}

float f2(float x) { //вторая производная

float u;

u = 2;

return u;

}

float function(float a, float b, float e) {// функция вычисляет по методу хорд и касательных

float c;

do {

cout<<"Итерация №"<< k<<" :"<< "[" <<a<<" ; "<<b<<"]"<<endl;

if (f(a) \* f2(a) < 0) {//Условие начальной точки для метода хорд

a = a + (b - a) / (f(a) - f(b))\*f(a); //формулы расчета по методу хорд

b = b - f(b) / f1(b);

}

else if (f(a) \* f2(a) > 0) {//Условие начальной точки для метода касательных

a = a - f(a) / f1(a); //формулы расчета по методу касательных

b = b + (b - a) / (f(b) - f(a))\*f(b);

}

else

{

cout << "can't solve\n";

break;

}

k++;

}

while (fabs(b - a) > e);//Построение хорд и касательных продолжается до достижения необходимой точности решения е

cout<<"Итерация №"<< k<<" :"<< "[" <<a<<" ; "<<b<<"]"<<endl;

return (a + b) / 2.0;

}

int main()

{

setlocale(LC\_ALL, "rus");

double a, b, e = 0.0001, F, pr = 1;

cout << "Комбинированный метод Ньютона и хорд" << endl;

cout << endl;

while (pr != 0) {

cout << "Введите начало отрезка a" << endl;

cin >> a;

cout << "Введите конец отрезка b" << endl;

cin >> b;

if (f(a)\*f(b) < 0)//отрезок на концах которых функция имеет разный знак

{

pr = 0;//находим значение функции от а и b

cout << setw(40) << "f(a) = " << "f(" << a << ") = " << f(a) << endl;

cout << setw(40) << "f(b) = " << "f(" << b << ") = " << f(b) << endl;

cout << endl;

}

else cout << setw(75) << "КОРНЕЙ НЕТ! ПОПЫТАЙТЕСЬ ВВЕСТИ ДРУГИЕ ЗНАЧЕНИЯ НАЧАЛА И КОНЦА ОТРЕЗКА!" << endl;

cout << endl;

}

F = function(a, b, e);

cout << "РЕЗУЛЬТАТ X = " << F << endl;//выводим результат на экран

cout << "Количество итераций: " << k << endl;

cout << "Проверка f(x) = " ;

cout << setprecision(10) << f(F);

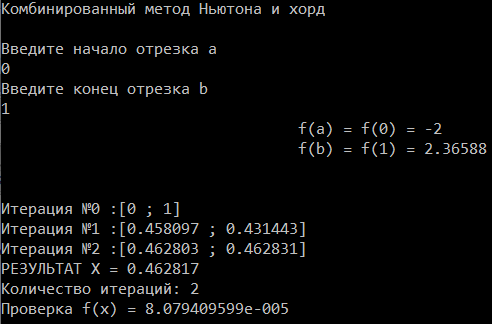
cout << endl;

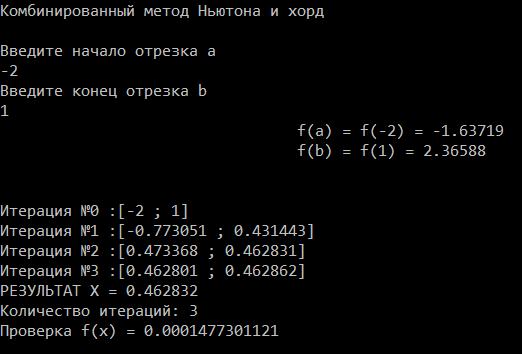
\_getch();

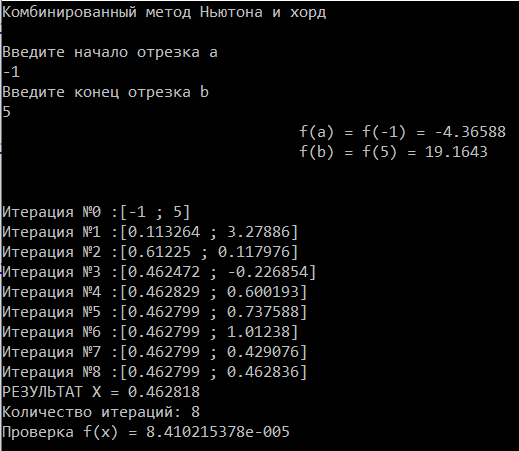
return 0;

}

Результаты выполнения:





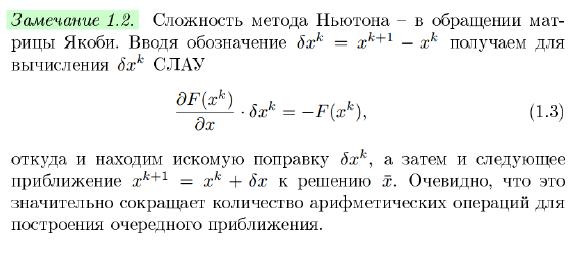


Таким образом, если отрезок локализации более широкий, то требуется большее число итераций для достижения заданной точности.

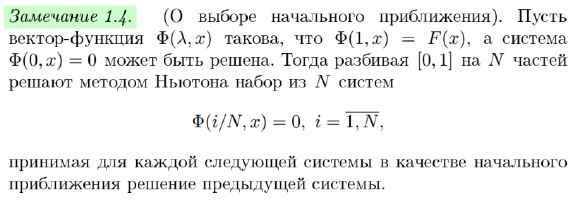
При соблюдении условий сходимости метода Ньютона ()) выхода за пределы отрезка локализации не происходит.

1. **Программная реализация метода Ньютона для решения систем уравнений:**

- Решить систему нелинейных уравнений из *Приложения №2* (номер варианта = номер в списке гуппы) методом Ньютона с точностью *e* = 10-4, используя Замечание 1.2 методического пособия



найдя начальное приближение двумя способами: 1) графическим методом и 2) используя Замечание 1.4 методического пособия:



Имеем систему:

Составим матрицу Якоби:

На каждой итерации получаем:

Т.е. решаем систему:

1) Выбираем начальное приближение графически. Строим и .



В качестве начального приближения можно выбрать точку .

2) Выбираем используя Замечание 1.4 методического пособия.

При получаем

Разбиваем на 4 части.

При получаем:

При получаем:

При получаем:

При получаем:

В качестве начального приближения выбираем точку .

Программа на С++:

#include <iostream>

#include <math.h>

#include <locale>

using namespace std;

double eps,J[2][2],Jm[2][2],x[2],x1[2],F[2],delta;

bool err,err1,err2;

int n;

int main()

{

setlocale(LC\_ALL, "russian");

cout<<"Введите точность"<<endl;

cin>>eps;

cout<<"Введите начальное приближение x0"<<endl;

cin>>x[0];

cout<<"Введите начальное приближение y0"<<endl;

cin>>x[1];

err=1;

n=0;

while (err)

{

//система

F[0]=sin(x[1]-1)+x[0]-1.3;

F[1]=x[1]-sin(x[0]+1)-0.8;

//матрица Якоби

J[0][0]=1;J[0][1]=cos(x[1]-1);

J[1][0]=-cos(x[0]+1);J[1][1]=1;

//определитель матрицы Якоби

delta=(J[0][0]\*J[1][1]-J[0][1]\*J[1][0]);

//Обратная матрица

Jm[0][0]=J[1][1]/delta;

Jm[1][1]=J[0][0]/delta;

Jm[1][0]=-J[1][0]/delta;

Jm[0][1]=-J[0][1]/delta;

//по формуле метода Ньютона

x1[0]=x[0]-(Jm[0][0]\*F[0]+Jm[0][1]\*F[1]);

x1[1]=x[1]-(Jm[1][0]\*F[0]+Jm[1][1]\*F[1]);

//пока не достигнем необходимой точности

err1=fabs(x1[0]-x[0])>eps;

err2=fabs(x1[1]-x[1])>eps;

err=err1||err2;

//принимаем за новое приближение вычисленное ранее

for (int i=0;i<2;i++)

{

x[i]=x1[i];

}

n++;//счётчик числа итераций

cout <<"Итерация №"<<n<<": "<<"x = "<<x[0]<<" y = "<<x[1]<<endl;

}

cout <<"Получено решение с заданной точностью!"<<endl<<endl;

cout <<"Число итераций равно "<<n<<endl;

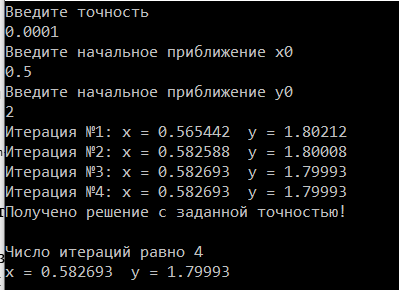
cout <<"x = "<<x[0]<<" y = "<<x[1]<<endl;

return 0;

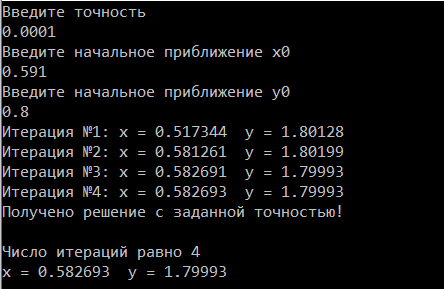
}

Результаты:

В качестве начального приближения можно выбрать точку :



В качестве начального приближения выбираем точку :



Таким образом, получили решения за 4 итерации как при определении начального приближения графическим методом, так и используя Замечание 1.4 методического пособия.