**Министерство науки и высшего образования РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИС**

отчет

**по лабораторной работе №8**

**по дисциплине «Конструирование программ»**

Тема: Решение систем нелинейных уравнений методом Ньютона

|  |  |  |
| --- | --- | --- |
| Студент гр. 8363 |  | Нерсисян А.С. |
| Преподаватель |  | Копыльцов А.В. |

Санкт-Петербург

2020

**Цель работы.**

Написать программу, которая любым способом, решает методом Ньютона систему уравнений.

**Основные теоретические положения.**

Основная идея метода Ньютона состоит в выделении из уравнений системы  линейных частей, которые являются главными при малых приращениях аргументов. Это позволяет свести одну исходную задачу к решению последовательности задач для линейных систем.

Итерационная формула метода Ньютона для системы нелинейных уравнений (6.4.7) имеет вид . Необходимость обращения матрицы первых частных производных при каждой итерации сильно затрудняет решение. Эти затруднения чаще носят технический характер, тем не менее вместо уравнения (6.4.7) иногда решают систему линейных алгебраических уравнений вида (6.4.8):



По методу Ньютона итерационный процесс при наличии хорошего начального приближения сходится с квадратичной скоростью, то есть если

, то 

В пакете Mathcad для решения систем нелинейных уравнений служат конструкции **Given-Find** и **Given-MinErr**, разобранные в четвертой лабораторной работе. С помощью этих подпрограмм можно решать системы объемом до двухсот уравнений.

Рассмотрим **пример**. Пусть дана система



Вводим программу











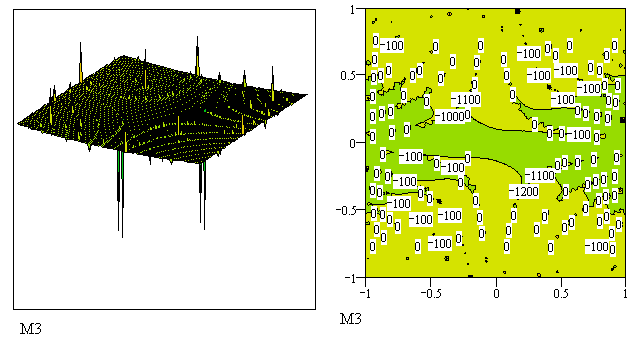
При задании иных начальных условий получаются другие точки экстремума. Например,





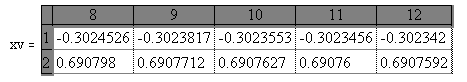




Это неудивительно, ибо поверхность, представляющая функцию, равную сумме уравнений исходной системы, имеет множество локальных экстремумов. Убедимся в этом, построив эту поверхность и ее линии уровня:

****

Найти решение упрощенным методом Ньютона по формулам (6.6.1), когда матрица  не перевычисляется на каждом шаге, можно с помощью нескольких операторов.

****

Наконец, подпрограммы, реализующие вычисления по методу Ньютона по формулам (6.4.7) и (6.4.8)-(6.4.9), могут быть такими:

Параметры этих подпрограмм одинаковы: , , - размерность системы, - вектор начального приближения,   
- заданная точность вычислений. Приведенные варианты подпрограмм рассчитаны на систему из двух уравнений с двумя неизвестными, однако легко переделываются для любого конечного числа неизвестных и уравнений. Для этого нужно лишь перечислить требуемое число аргументов в операторах, вычисляющих вектор  и матрицу  внутри подпрограмм.

Вводим конец программы:









- подпрограмма не может решить систему, начальное приближение плохое и процесс расходится.

**Экспериментальные результаты.**

**Задание № 1**

Любым способом, разобранным в этой лабораторной работе, решить методом Ньютона следующую систему уравнений:

**Дано:** Вариант 11

**Обработка результатов эксперимента.**

**Задание № 1. решение:**

#include <iostream>

#include <conio.h>

using namespace std;

double tg(double x)

{

return x + x\*x\*x / 3 + 2 \* x\*x\*x\*x\*x / 15 + 17 \* x\*x\*x\*x\*x\*x\*x / 315;

}

double func1(double x, double y)

{

return tg(x\*y + 0.2) - x\*x;

}

double func2(double x, double y)

{

return x\*x + y\*y - 1;

}

int main()

{

const double E = 0.001, h = 0.001;

double x[2], y[2], prev[2], F[2][2], det, temp;

cout << "tg(xy + 0.2) = x^2" << endl << "x^2 + y^2 = 1" << endl;

x[0] = 0.1;

x[1] = 0.1;

do

{

prev[0] = x[0];

prev[1] = x[1];

F[0][0] = (func1(prev[0] + h, prev[1]) - func1(prev[0] - h, prev[1])) / (2 \* h);

F[0][1] = (func1(prev[0], prev[1] + h) - func1(prev[0], prev[1] - h)) / (2 \* h);

F[1][0] = (func2(prev[0] + h, prev[1]) - func2(prev[0] - h, prev[1])) / (2 \* h);

F[1][1] = (func2(prev[0], prev[1] + h) - func2(prev[0], prev[1] - h)) / (2 \* h);

det = F[0][0] \* F[1][1] - F[0][1] \* F[1][0];

temp = F[0][0];

F[0][0] = F[1][1] / det;

F[1][1] = temp / det;

F[0][1] /= -det;

F[1][0] /= -det;

y[0] = func1(prev[0], prev[1]);

y[1] = func2(prev[0], prev[1]);

x[0] = prev[0] - F[0][0] \* y[0] - F[0][1] \* y[1];

x[1] = prev[1] - F[1][0] \* y[0] - F[1][1] \* y[1];

} while (((x[0] - prev[0]) > E) || ((x[1] - prev[1]) > E));

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

\_getch();

return 0;

}

**Выводы.**

В ходе выполнения данной лабораторной была написана программа, которая методом Ньютона решает систему уравнений.