|  |  |
| --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ | |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  «Пермский государственный национальный исследовательский университет» | |
| **ЧИСЛЕННЫЕ МЕТОДЫ** *Лабораторная работа №\_4\_*  **«**Прямые методы решения систем линейных алгебраических уравнений**»** | |
| Варианты: №3, №6 | |
|  | Работу выполнили студенты группы ПМИ-2-16  Мироненко Анастасия Олеговна,  Зимин Илья Владимирович |
| Оценка отчета   |  |  |  | | --- | --- | --- | |  | Баллы |  | | Конспект | 1 |  | | Опоздание с отчетом | -0.05 день |  | | Попытки | -0.5 попыт. |  | | Замечания к отчету | -0.25 зам. |  | | Программа | 1 |  | | Выводы  (Заключение) | 1 |  | | Защита | 1 |  | | ИТОГО: |  |  | | Проверил:  профессор, доктор физико-математических наук  С. В. Русаков  “\_\_\_\_” 2018 г. |
| Замечания:  Пермь 2018 | |

**Содержание**

[1. Задание. 3](#_Toc525161389)

[2. Исходные данные. 4](#_Toc525161390)

[3. Решение 5](#_Toc525161391)

[4. Краткие выводы. 6](#_Toc525161392)

[5. Текст программы. 7](#_Toc525161393)

1. **Задание**

**Тема:** «Решение нелинейных уравнений».

Найти корни системы нелинейных уравнений



с точностью .

1. Приближенно определить корни геометрически.
2. Уточнить корни методом:

* Простой итерации;
* Ньютона;
* Градиентного спуска, сведя к нахождению минимума функции



1. Провести анализ скорости сходимости и точности решения рассмотренными методами.

**Требования к отчету.**

1. Структура отчета:

* Задание;
* Исходные данные;
* Теоретическая справка;
* Решение (по пунктам задания);
* Краткие выводы.

1. Теоретическая справка - основные определения и расчетные формулы.
2. Для решения задачи написать программу на языке высокого уровня.
3. В процессе решения должны выводиться (с комментариями) все промежуточные результаты вычислений: приближенное решение, невязку, оценку погрешности, значения итерационных параметров и т.п.
4. Алгоритм вычислений должен быть организован максимально экономично, чтобы выполнялись полученные оценки по вычислительным затратам.
5. Отчет сдается в виде файла формата DOC. В приложение к отчету приводится листинг программы.

**Пример исходных данных**

****

1. **Исходные данные**

|  |  |
| --- | --- |
| Вариант №3: | Вариант №6 |
|  |  |

1. **Решение**

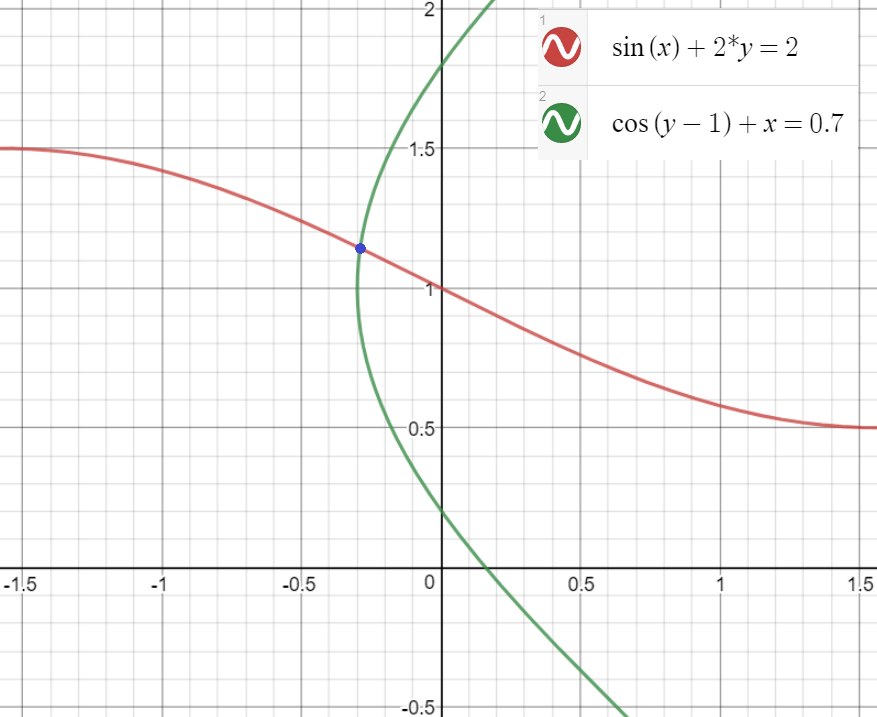
В данном разделе будут приведены результаты, а также подробные шаги выполнения программы, на примере двух наборов: №3 и №6.

Программа считывает оба уравнения, а также начальное приближение из файла «input.txt», затем автоматически генерирует файл «output.txt» и выводит в него все необходимые данные. Программа сработает корректно на любых входных данных, если уравнения имеют вид, подобный тому, как в наборах 3, 6.

# **Набор №3.**

# 

1. Воспользуемся возможностями сайта desmos.com и построим графики обоих уравнений и найдём их пересечение:



Можно заметить, что решение системы лежит в пределах: 

Поэтому выберем начальное приближение 

Файл «input.txt»:

sin(x)+2y=2

cos(y-1)+x=0,7

-0.3 1.15

Файл «output.txt»:

**ИСХОДНАЯ СИСТЕМА УРАВНЕНИЙ:**

sin(x) + 2y = 2

cos(y - 1) + x = 0,7

**МЕТОД ПРОСТОЙ ИТЕРАЦИИ:**

Представим уравнения системы в виде f(x,y) = 0:

sin(x) + 2y – 2 = 0

cos(y - 1) + x - 0,7 = 0

Преобразуем систему к эквивалентной:

y = 0.5\*(2 - sin(x))

x = 0.7 - cos(y - 1)

Начальное приближение: x = -0.3, y = 1.15

Якобиан:

0.5\*-cos(x) 0

0 sin(y-1)

Подставим начальное приближение в якобиан:

0.149438 0

0 -0.477668

Норма якобиана: 0.477668

Норма Норма

№ x y невязки F1 F2 якобиана

1 -0.2887711 1.1477601 0.0053832 1.0745793E-02 3.3224530E-04 0.479297

2 -0.2891033 1.1423872 0.0007929 -3.1847281E-04 7.7673351E-04 0.479250

3 -0.2898801 1.1425464 0.0003729 -7.4441291E-04 -2.2609241E-05 0.479139

4 -0.2898575 1.1429186 0.0000540 2.1666015E-05 -5.2945773E-05 0.479142

5 -0.2898045 1.1429078 0.0000254 5.0737521E-05 1.5429155E-06 0.479150

Число итераций: 5

Последнее приближение: **x** = -0.2898045; **y** = 1.1429078

**МЕТОД НЬЮТОНА**

Представим уравнения системы в виде f(x,y) = 0:

sin(x) + 2y – 2 = 0

cos(y - 1) + x - 0,7 = 0

Матрица производных имеет вид:

cos(x) 2

1 -sin(y-1)

Матрица, обратная матрице производных:

-sin(y-1) -2

-1 cos(x)

№ x y Норма невязки F1 F2

1 -0.2898317 1.1429030 1.2400106E-02 1.5110171E-05 -2.4909674E-05

2 -0.2898093 1.1428848 2.8821615E-05 7.1130213E-11 -1.6475110E-10

Число итераций: 2

Последнее приближение: **x** = -0.2898093; **y** = 1.1428848

**МЕТОД ГРАДИЕНТНОГО СПУСКА**

Выберем alpha = 0.5

Оптимальный параметр lyambda = 0.46

№ x y alpha Норма невязки F1 F2 FF k

1 -0.2930508 1.1393624 0.5000 1.2089540E-02 4.4797933E-03 -1.1228910E-02 1.4615697E-04 0

2 -0.2904137 1.1435770 0.1058 1.0514444E-02 -1.0149528E-02 -2.7460184E-03 1.1055354E-04 2

3 -0.2904452 1.1427898 0.2300 1.0690712E-03 8.0531105E-04 -7.0312682E-04 1.1429132E-06 1

4 -0.2901515 1.1431093 0.1058 1.0128971E-03 -7.9919285E-04 -6.2229523E-04 1.0259606E-06 2

5 -0.2898935 1.1428135 0.5000 3.9324286E-04 1.2118216E-04 -3.7410538E-04 1.5463995E-07 0

6 -0.2898326 1.1429057 0.1058 2.3499324E-04 -2.2304297E-04 -7.3984171E-05 5.5221823E-08 2

7 -0.2898292 1.1428859 0.2300 3.2755087E-05 1.9626143E-05 -2.6224230E-05 1.0728957E-09 1

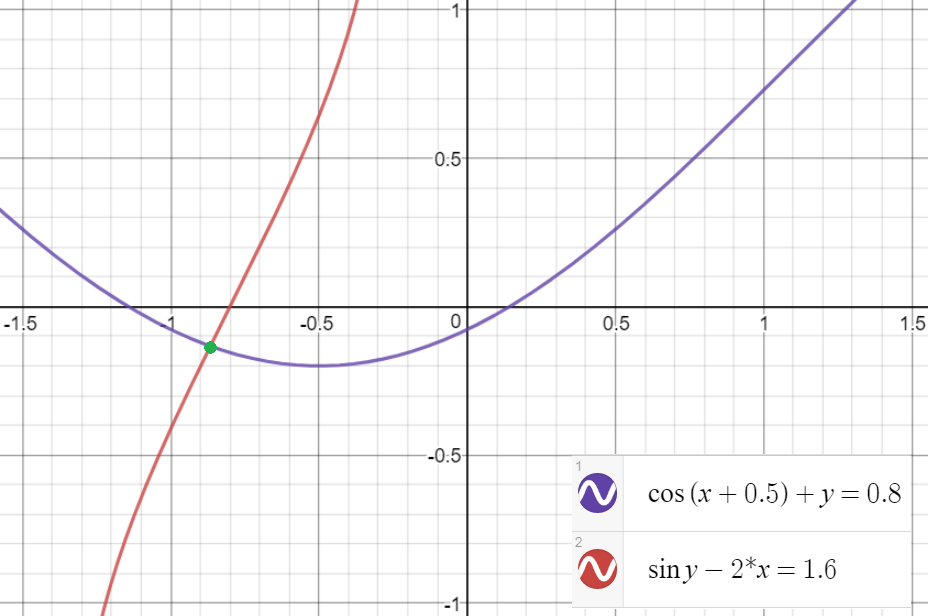
Число итераций: 7

Последнее приближение: **x** = -0.2898292; **y** = 1.1428859

# **Набор №6.**



1. Воспользуемся возможностями сайта desmos.com и построим графики обоих уравнений и найдём их пересечение:



Можно заметить, что решение системы лежит в пределах: 

Поэтому выберем начальное приближение 

Файл «input.txt»:

cos(x+0,5)+y=0,8

sin(y)+2\*x=1,6

-0.88 -0.13

Файл «output.txt»:

**ИСХОДНАЯ СИСТЕМА УРАВНЕНИЙ:**

cos(x + 0,5) + y = 0,8

sin(y) - 2x = 1,6

**МЕТОД ПРОСТОЙ ИТЕРАЦИИ:**

Представим уравнения системы в виде f(x,y) = 0:

cos(x + 0,5) + y - 0,8 = 0

sin(y) - 2x - 1,6 = 0

Преобразуем систему к эквивалентной (выразим x и y):

y = 0.8 - cos(x+0.5)

x= -0.5 \* (1.6 - sin(y))

Начальное приближение: x = -0.88, y = -0.13

Якобиан имеет вид:

sin(x+0.5) 0

1. -0.5\*cos(y)

Подставим начальное приближение в якобиан:

0.495781 0

0 -0.37092

Норма якобиана: 0.495781

Норма Норма

№ x y невязки F1 F2 якобиана

1 -0.8648171 -0.1286646 0.0055639 5.5244020E-03 1.3242235E-03 0.495867

2 -0.8641550 -0.1341890 0.0027485 2.3602240E-04 -5.4767521E-03 0.495505

3 -0.8668933 -0.1344250 0.0009858 -9.7880175E-04 -2.3389686E-04 0.495489

4 -0.8670103 -0.1334462 0.0004868 -4.1957806E-05 9.7003559E-04 0.495555

5 -0.8665253 -0.1334043 0.0001752 1.7392741E-04 4.1584887E-05 0.495557

6 -0.8665045 -0.1335782 0.0000865 7.4512626E-06 -1.7238003E-04 0.495546

7 -0.8665907 -0.1335857 0.0000311 -3.0890026E-05 -7.3848807E-06 0.495545

Число итераций: 7

Последнее приближение: **x** = -0.8665907; **y** = -0.1335857

**МЕТОД НЬЮТОНА**

Представим уравнения системы в виде f(x,y) = 0:

cos(x + 0,5) + y - 0,8 = 0

sin(y) - 2x - 1,6 = 0

Матрица производных имеет вид:

-sin(x+0.5) 1

-2 cos(y)

Матрица, обратная матрице производных

cos(y) -1

2 -sin(x+0.5)

№ x y Норма невязки F1 F2

1.3867019E-02

1 -0.8666162 -0.1336289 1.3867019E-02 -8.3320507E-05 8.6148351E-07

2 -0.8665808 -0.1335583 7.9009161E-05 -5.8582628E-10 3.3218117E-10

Число итераций: 2

Последнее приближение: **x** = -0.8665808; **y** = -0.1335583

**МЕТОД ГРАДИЕНТНОГО СПУСКА**

Выберем alpha = 0.5

Оптимальный параметр lyambda = 0.31

№ x y alpha Норма невязки F1 F2 FF k

1 -0.8610196 -0.1389200 0.155 3.0395182E-02 -1.3353763E-03 3.0365834E-02 9.2386708E-04 1

2 -0.8708385 -0.1328257 0.155 1.6778922E-02 -3.3828760E-03 -1.6434365E-02 2.8153221E-04 1

3 -0.8650187 -0.1354168 0.155 9.2762100E-03 -8.0196026E-04 9.2414788E-03 8.6048071E-05 1

4 -0.8679537 -0.1334885 0.155 5.1331529E-03 -1.2996894E-03 -4.9658902E-03 2.6349258E-05 1

5 -0.8661612 -0.1342222 0.155 2.8466318E-03 -4.2318252E-04 2.8150007E-03 8.1033127E-06 1

6 -0.8670324 -0.1336031 0.155 1.5827663E-03 -5.1361138E-04 -1.4971147E-03 2.5051490E-06 1

7 -0.8664769 -0.1338029 0.155 8.8343325E-04 -2.0673578E-04 8.5890315E-04 7.8045431E-07 1

8 -0.8667330 -0.1336003 0.155 4.9560108E-04 -2.0733294E-04 -4.5014829E-04 2.4562043E-07 1

9 -0.8665594 -0.1336511 0.155 4.7991457E-04 -9.6545629E-05 2.6273772E-04 7.8352167E-08 1

10 -0.8666335 -0.1335833 0.155 1.5946419E-04 -8.5095705E-05 -1.3486122E-04 2.5428829E-08 1

11 -0.8665786 -0.1335945 0.155 9.1821281E-05 -4.3863575E-05 8.0666812E-05 8.4311477E-09 1

Число итераций: 11

Последнее приближение: **x** = -0.8665786; **y** = -0.1335945

1. **Краткие выводы**

Для наглядности запишем полученные данные в виде таблиц, тем самым сравним результаты, полученные всеми четырьмя итерационными методами.

Вариант 3)

|  |  |  |  |
| --- | --- | --- | --- |
| Метод: | Число итераций | X | Y |
| Простой итерации | 5 | -0.2898045 | 1.1429078 |
| Ньютона | 2 | -0.2898093 | 1.1428848 |
| Градиентного спуска | 7 | -0.2898292 | 1.1428859 |

Вариант 6)

|  |  |  |  |
| --- | --- | --- | --- |
| Метод: | Число итераций | X | Y |
| Простой итерации | 7 | -0.8665907 | -0.1335857 |
| Ньютона | 2 | -0.8665808 | -0.1335583 |
| Градиентного спуска | 11 | -0.8665786 | -0.1335945 |

По количеству итераций лучшие результаты показывает метод Ньютона, худший – метод градиентного спуска.

По вычислительным затратам самым объемным является метод градиентного спуска, самым мало-затратным - метод простой итерации.

Если не брать во внимание парсинг уравнений и подсчет производных и значений функций, то легче всего было запрограммировать метод простой итерации, метод градиентного спуска, как ни странно отнял большую часть работы.

Сделать выводы о том, какой метод является более точным достаточно сложно, так как точные результаты не вычислялись.

1. **Текст программы**

Программа состоит из одного файла «Main.cpp» - исходный код программы

«Main.cpp»

#include <iostream>

#include <fstream>

#include <cmath>

#include <string>

#include <tuple>

#include <algorithm>

#include <iomanip>

using namespace std;

double const EPS = 0.0001;

// Структура уравнения

struct Equation

{

double multiplier; // Коэффициент при выражаемой переменной (х или у)

double multiplier\_1; // 1 / Коэффициент при выражаемой переменной (х или у)

char variable; // Выражаемая переменная

string trigFunc; // Тригонометрическая функция

char variableInTrigFunk; // Переменная в триг. функции

double addInTrigFunk; // Слагаемое в триг. Функции

char signForTrig = '+'; // Знак перед триг. функцией

double freeMember; // Свободный член (в правой части)

string f; // Вид функции f

} eq1, eq2;

// Разбиваем строку уравнения на части структуры

void Parser(string s, Equation &eq)

{

int i = 0;

string f = "";

while (s[i] != '\0')

{

if (s[i] == '=')

{

i++;

if (s[i] == '-')

{

f += '+';

i++;

}

else f += '-';

}

f += s[i];

i++;

}

f += "=0";

eq.f = f;

i = 0;

while (s[i] != '\0')

{

if (s[i] == 's' || s[i] == 'c')

{

if (i == 0)

{

eq.signForTrig = '-';

}

else if (s[i - 1] == '-')

{

eq.signForTrig = '+';

}

else eq.signForTrig = '-';

eq.trigFunc = s.substr(i, 3);

i+=4;

eq.variableInTrigFunk = s[i];

i++;

if (s[i] == ')')

{

eq.addInTrigFunk = 0;

i++;

}

else if (s[i+2] == ')')

{

eq.addInTrigFunk = stof(s.substr(i, 2));

i += 3;

}

else

{

cout <<s.substr(i, 4);

eq.addInTrigFunk = stof(s.substr(i, 4));

i += 5;

}

}

else if (s[i] <= '9' && s[i] >= '0')

{

if (s[i + 1] == 'x' || s[i + 1] == 'y')

{

if (i != 0 && s[i - 1] == '-')

eq.multiplier = stof(s.substr(i - 1, 2));

else eq.multiplier = stof(s.substr(i, 1));

i ++;

}

else

{

string freeMember = "";

while (s[i] != '\0')

{

freeMember += s[i];

i++;

}

eq.freeMember = stof(freeMember);

}

}

else if (s[i] == 'x' || s[i] == 'y')

{

eq.variable = s[i];

if (i != 0)

{

if (s[i - 1] == '-')

eq.multiplier = -1;

else if (s[i - 1] == '+')

eq.multiplier = 1;

}

else eq.multiplier = 1;

i++;

}

else i++;

}

eq.multiplier\_1 = 1 / eq.multiplier;

}

// 1-я норма вектора

double NormVecI(double x, double y)

{

return max(fabs(x), fabs(y));

}

// 2-я норма вектора

double NormVecII(double x, double y)

{

return fabs(x)+fabs(y);

}

// 3-я норма вектора

double NormVecIII(double x, double y)

{

return sqrt(x\*x+y\*y);

}

//Решение правой части уравнения при выраженном x или y

//(для метода простой итерации)

double SolveXY(Equation eq, double var)

{

double res = eq.freeMember;

double trig = eq.addInTrigFunk + var;

if (eq.trigFunc[0] == 's')

trig = sin(trig);

else trig = cos(trig);

if (eq.signForTrig == '-')

trig = 0 - trig;

res += trig;

res \*= eq.multiplier\_1;

return res;

}

//Решение правой части уравнения при выраженном x или y

double fSolve(Equation eq, double var1, double var2)

{

double res = eq.multiplier\*var2;

double trig = eq.addInTrigFunk + var1;

if (eq.trigFunc[0] == 's')

trig = sin(trig);

else trig = cos(trig);

if (eq.signForTrig == '+')

trig = 0 - trig;

res += trig;

res -= eq.freeMember;

return res;

}

//Вычисление якобиана в заданной точке

double SolveJacobian(Equation eq, double var)

{

double trig = eq.addInTrigFunk + var;

if (eq.trigFunc[0] == 's')

trig = cos(trig);

else trig = 0-sin(trig);

if (eq.signForTrig == '-')

trig = 0 - trig;

return eq.multiplier\_1\*trig;

}

//Печать строчки якобиана в файл

void PrintJacobian(Equation eq, ofstream & fout)

{

if (abs(eq.multiplier\_1) != 1) fout << eq.multiplier\_1 << "\*";

if (eq.trigFunc == "cos")

{

if (eq.signForTrig == '-' && eq.multiplier\_1 < 0

|| eq.signForTrig == '+' && eq.multiplier\_1 > 0)

fout << "-sin(" << eq.variableInTrigFunk;

else fout << "sin(" << eq.variableInTrigFunk;

}

else

{

if (eq.signForTrig == '-' && eq.multiplier\_1 < 0

|| eq.signForTrig == '+' && eq.multiplier\_1 > 0)

fout << "cos(" << eq.variableInTrigFunk;

else fout << "-cos(" << eq.variableInTrigFunk;

}

if (eq.addInTrigFunk != 0)

if (eq.addInTrigFunk < 0)

fout << eq.addInTrigFunk;

else fout << "+" << eq.addInTrigFunk;

fout << ')';

}

//Печать матрицы якобиана

void Jacobian(ofstream & fout)

{

fout << "Якобиан: \n\n";

PrintJacobian(eq1, fout);

fout << setw(10) << 0 << endl << 0 << setw(15);

PrintJacobian(eq2, fout);

}

//Печать функции фи (для метода простой итерации)

void PrintPhi(Equation eq, ofstream &fout)

{

fout << eq.variable << "=";

if (eq.multiplier\_1 != 1)

fout << eq.multiplier\_1 << "\*" << "(";

fout << eq.freeMember << eq.signForTrig << eq.trigFunc << "("

<< eq.variableInTrigFunk;

if (eq.addInTrigFunk != 0)

if (eq.addInTrigFunk < 0)

fout << eq.addInTrigFunk;

else fout << "+" << eq.addInTrigFunk;

fout << ")";

if (eq.multiplier\_1 != 1)

fout << ")";

}

//Метод простой итерации

tuple<double, double> SimpleIteration(ofstream & fout, double x0, double y0)

{

fout << "\n\nМЕТОД ПРОСТОЙ ИТЕРАЦИИ: \n";

fout << "Представим уравнения системы в виде f(x,y) = 0: \n";

fout << "\nУр. №1: " << eq1.f << "\nУр. №2: " << eq2.f << endl;

fout << "\n\nПреобразуем систему к эквивалентной:";

fout << "\nУр. №1: ";

PrintPhi(eq1, fout);

fout << "\nУр. №2: ";

PrintPhi(eq2, fout);

double x, y, xNext(x0), yNext(y0), f1, f2;

int countItr = 0;

fout << "\n\nНачальное приближение: x = " << x0 << ", y = " << y0 << "\n";

Jacobian(fout);

fout << "\nПодставим начальное приближение в якобиан: \n\n";

double x\_proizv = SolveJacobian(eq2, y0), y\_proizv = SolveJacobian(eq1, x0);

fout << setw(10) << x\_proizv << setw(10) << 0 << endl

<< setw(10) << 0 << setw(10) << y\_proizv;

double Norm = max(abs(x\_proizv), abs(y\_proizv));

fout << "\n\nНорма якобиана: " << Norm << endl << endl;

do {

countItr++;

x = xNext;

y = yNext;

xNext = SolveXY(eq2, y);

yNext = SolveXY(eq1, x);

f1 = fSolve(eq1, xNext, yNext);

f2 = fSolve(eq2, yNext, xNext);

Norm = NormVecIII(x - xNext, y - yNext);

fout << setw(3) << countItr << setw(13) << fixed << setprecision(7)

<< xNext << setw(13) << yNext << setw(13)

<< Norm << setw(20) << uppercase << scientific << f1

<< setw(20) << f2;

fout.copyfmt(ios(NULL));

fout << setw(13) << NormVecI(SolveJacobian(eq2, yNext),

SolveJacobian(eq1, xNext)) << "\n";

} while (NormVecIII(x-xNext, y-yNext) >= EPS);

fout << "Число итераций: " << countItr << endl;

fout << "Последнее приближение: x = " << fixed << setprecision(7)

<< xNext << " y = " << setw(13) << yNext << endl;

return tuple<double, double>(xNext, yNext);

}

//Печать частной производной в файл

void PrintDf(Equation eq, ofstream & fout)

{

if (eq.trigFunc[0] == 's') {

if (eq.signForTrig == '+')

fout << '-';

fout << "cos(";

}

else {

if (eq.signForTrig == '-')

fout << '-';

fout << "sin(";

}

fout << eq.variableInTrigFunk;

if (eq.addInTrigFunk > 0)

fout << '+';

if (eq.addInTrigFunk != 0)

fout << eq.addInTrigFunk;

fout << ')';

}

//Вычисление частной производной в точке

double SolveDf(Equation eq, double var1)

{

double d;

if (eq.trigFunc[0] == 's')

d = cos(var1 + eq.addInTrigFunk);

else

d = 0 - sin(var1 + eq.addInTrigFunk);

if (eq.signForTrig == '+')

d = 0 - d;

return d;

}

//Метод Ньютона

tuple<double, double> Newton(ofstream & fout, double x0, double y0)

{

fout << "\n\nМЕТОД НЬЮТОНА: \n";

fout << "Представим уравнения системы в виде f(x,y) = 0: \n";

fout << "\nУр. №1: " << eq1.f << "\nУр. №2: " << eq2.f << endl;

fout << "Матрица производных имеет вид: \n";

fout.copyfmt(ios(NULL));

df(eq1, fout); fout << "\t" << eq1.multiplier << endl;

fout << eq2.multiplier << "\t";

df(eq2, fout); fout << endl;

double x, y, xNext(x0), yNext(y0), f1, f2;

double dxf1, dxf2, dyf1, dyf2, det, Norm;

int countItr = 0;

dxf2 = eq2.multiplier;

dyf1 = eq1.multiplier;

f1 = fSolve(eq1, x0, y0);

f2 = fSolve(eq2, y0, x0);

do

{

countItr++;

x = xNext;

y = yNext;

dxf1 = SolveDf(eq1, x);

dyf2 = SolveDf(eq2, y);

det = dxf1 \* dyf2 - dxf2 \* dyf1;

xNext = x - (f1 \* dyf2 - f2 \* dyf1) / det;

yNext = y - (f2 \* dxf1 - f1 \* dxf2) / det;

f1 = fSolve(eq1, xNext, yNext);

f2 = fSolve(eq2, yNext, xNext);

Norm = NormVecIII(x - xNext, y - yNext);

fout << setw(3) << countItr << setw(13) << fixed

<< setprecision(7) << xNext << setw(13) << yNext << setw(20)

<< uppercase << scientific << Norm << setw(20)

<< f1 << setw(20) << f2 << endl;

} while (Norm >= EPS);

fout << "Число итераций: " << countItr << endl;

fout << "Последнее приближение: x = " << fixed << setprecision(7)

<< xNext << " y = " << setw(13) << yNext << endl;

return tuple<double, double>(xNext, yNext);

}

// Вычисление ф-и F = f1\*f1 + f2\*f2

double F(double x, double y)

{

double f1 = fSolve(eq1, x, y), f2 = fSolve(eq2, y, x);

return f1 \* f1 + f2 \* f2;

}

// Вычисление частной производной по x

double dFdx(double x, double y)

{

return 2 \* fSolve(eq1, x, y)\*SolveDf(eq1, x) + 2 \* fSolve(eq2, y, x)\*eq2.multiplier;

}

// Вычисление частной производной по y

double dFdy(double x, double y)

{

return 2 \* fSolve(eq1, x, y)\*eq1.multiplier + 2 \* fSolve(eq2, y, x)\*SolveDf(eq2, y);

}

// Поиск оптимального значения lyambda

double SearchLyambda(double x0, double y0)

{

double xNext(x0), yNext(y0), a, a0(0.5), l(0.01), l1, l0(0.01), x(x0), y(y0);

int iter = 0, min(INT\_MAX);

for (int i = 0; i < 80; i++)

{

xNext = x;

yNext = y;

iter = 0;

do {

iter++;

x0 = xNext;

y0 = yNext;

a = a0;

while (F(x0 - a \* dFdx(x0, y0), y0 - a \* dFdy(x0, y0)) >= F(x0, y0)) {

a \*= l;

}

xNext = x0 - a \* dFdx(x0, y0);

yNext = y0 - a \* dFdy(x0, y0);

} while (NormVecIII(x0 - xNext, y0 - yNext) >= EPS);

if (iter < min)

{

l1 = l;

min = iter;

}

l += l0;

}

return l1;

}

// Метод градиентного спуска

tuple<double, double> Gradient(ofstream &fout, double x0, double y0)//Градиентный спуск

{

fout << "\n\nМЕТОД ГРАДИЕНТНОГО СПУСКА: \n";

double xNext(x0), yNext(y0), a, a0(0.5), l, k;

l = SearchLyambda(x0, y0);

fout << "\nОптимальный параметр lambda = " << l << endl;

int countItr = 0;

do {

countItr++;

x0 = xNext;

y0 = yNext;

a = 0.5;

k = 0;

while (F(x0 - a \* dFdx(x0, y0), y0 - a \* dFdy(x0, y0)) >= F(x0, y0))

{

a \*= l;

k++;

}

double f1 = fSolve(eq1, x0, y0), f2 = fSolve(eq2, y0, x0);

xNext = x0 - a \* dFdx(x0, y0);

yNext = y0 - a \* dFdy(x0, y0);

fout << setw(3) << countItr << setw(13) << fixed << setprecision(7) << xNext << setw(13) << yNext << setw(13) << a << setw(20)

<< uppercase << scientific << NormVecIII(f1, f2) << setw(20) << f1 << setw(20) <<f2<<setw(20)<< NormVecIII(f1,f2)\*NormVecIII(f1,f2) << setw(3) << k << endl;

} while (NormVecIII(x0 - xNext, y0 - yNext) >= EPS);

fout << "Число итераций: " << countItr << endl;

fout << "Последнее приближение: x = " << fixed << setprecision(7) << xNext

<< " y = " << setw(13) << yNext << endl;

return tuple<double, double>(xNext, yNext);

}

// Основная функция

int main()

{

setlocale(LC\_ALL, "Russian");

// Номер набора

int num = 3;

// Начальное приближение

double x0, y0;

// Файл для считывания

ifstream fin;

// Файл для записи

ofstream fout;

string strEq1, strEq2;

// Открываем файл для считывания данных

fin.open("input" + to\_string(num) + ".txt");

fin >> strEq1 >> strEq2;

fin >> x0 >> y0;

fout.open("output.txt");// +to\_string(num) + ".txt");

Parser(strEq1, eq1);

Parser(strEq2, eq2);

fout << "ИСХОДНАЯ СИСТЕМА УРАВНЕНИЙ: ";

fout << "\nУр. №1: " << strEq1;

fout << "\nУр. №2: " << strEq2;

// Считаем методом простой итерации

SimpleIteration(fout, x0, y0);

// Считаем методом Ньютона

Newton(fout, x0, y0);

// Считаем методом градиентного спуска

Gradient(fout, x0, y0);

return 0;

}