Федеральное агентство связи  
Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
 «Сибирский государственный университет  
телекоммуникаций и информатики»

Кафедра ПМиК

Лабораторная работа №4 по дисциплине

«Вычислительная математика» по теме:

«Многомерный вариант метода Ньютона»

Выполнил:

ст. гр. ИВ-823

Шиндель Э. Д.

Проверила:

Ассистент

Кафедры ПМиК

Петухова Я. В.

Новосибирск, 2020

**Содержание:**

1. Изложение метода3

2. Пример решения методом Ньютона4

3. Пример работы программы5

4. Листинг6

**1. Изложение метода**

МПД и МХ применимы только для решения НУ, метод Ньютона может быть легко видоизменен, и его можно применять для решения СНУ.

Рассмотрим СНУ  n на n (n – уравнений, n – неизвестных):

F(X) = 0, X =

При решении СНУ поступаем таким же образом, как и при решении НУ:

1) Выбираем стартовую точку , достаточно близкую к корню.

2) В одномерном варианте мы заменяли функцию на касательную и приравнивали её к нулю. Аналогичным образом поступаем и для функции многих переменных, только там заменяем на дифференциал, т.е.:

Решаем данное уравнение относительно X:

W – матрица частных производных (матрица Якоби)

умножим на матрицу обратную матрице W слева:

Окончательный вид формулы многомерного варианта метода Ньютона:

**2. Пример решения методом Ньютона**

e = 0,0001 (точность)

Приводим к виду F(X)=0:

,

в качестве стартовой точки возьмем:

Сделаем один шаг по многомерному методу Ньютона:

=

=

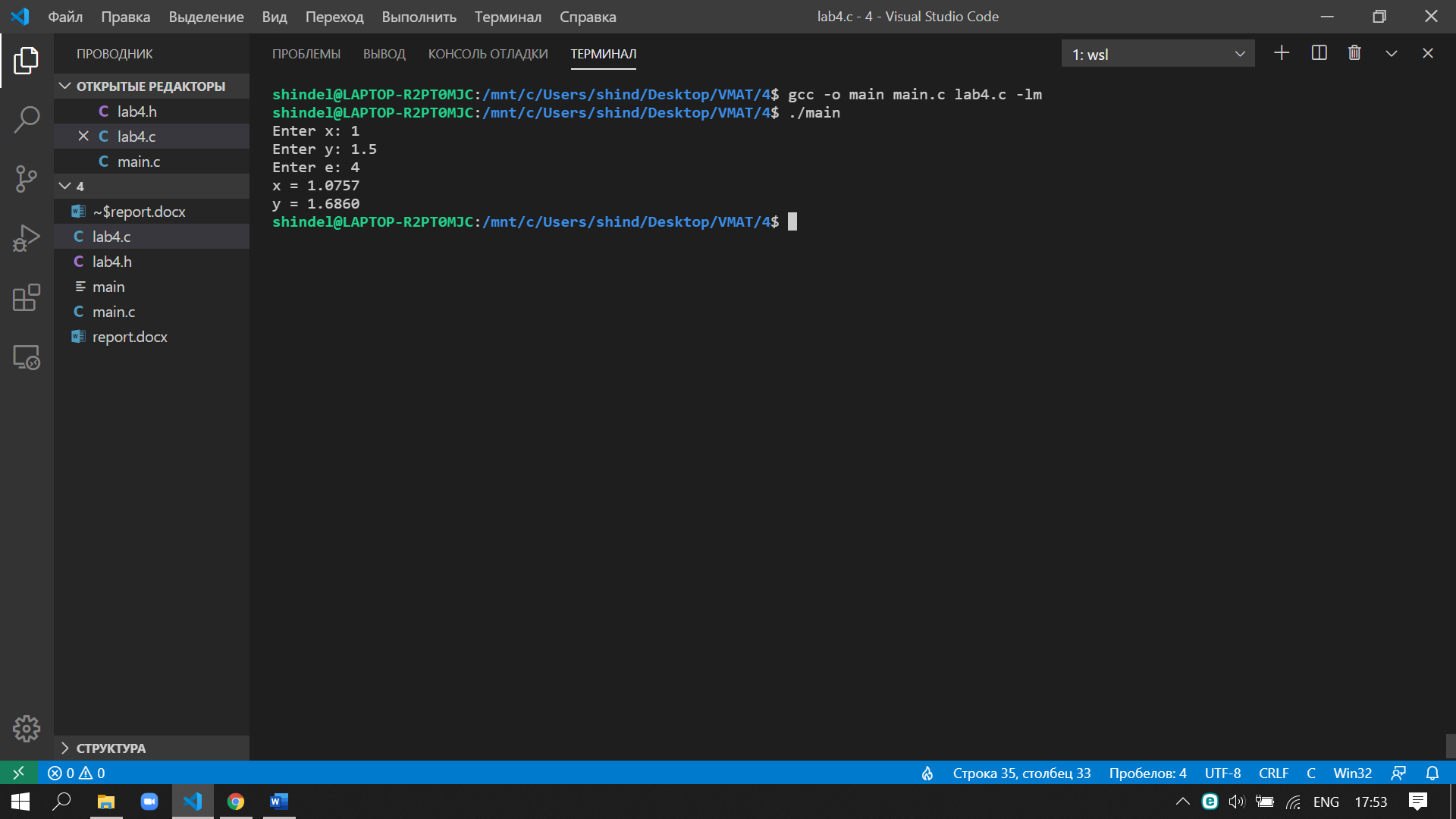
Отсюда последовательно получаем:

.

Ответ: x = 1,0757

y = 1,6869.

3. Пример работы программы



**4. Листинг**

#include "lab4.h"

double f1(double x, double y, int i) {

double f;

if (i == 1) f = 2 \* (x \* x) + y - 4;

else f = x \* x + y \* y - 4;

return f;

}

double f2(double x, double y, int i, double dif) {

double f2;

if (i == 1) f2 = (f1(x + dif, y, 1) - f1(x - dif, y, 1)) / (2 \* dif);

else if (i == 2) f2 = (f1(x, y + dif, 1) - f1(x, y - dif, 1)) / (2 \* dif);

else if (i == 3) f2 = (f1(x + dif, y, 2) - f1(x - dif, y, 2)) / (2 \* dif);

else f2 = (f1(x, y + dif, 2) - f1(x, y - dif, 2)) / (2 \* dif);

return f2;

}

void inverse\_matrix(double \*\*m) {

double dop, opr = m[0][0] \* m[1][1] - m[0][1] \* m[1][0];

dop = m[0][0];

m[0][0] = m[1][1] / opr;

m[1][1] = dop / opr;

m[0][1] = -m[0][1] / opr;

m[1][0] = -m[1][0] / opr;

}

double aim(int e) {

double dif = 1.0;

for (int i = 0; i < e; i++) dif /= 10;

return dif;

}

void Newton(int e, double x, double y) {

double x0, y0, dif = aim(e);

double \*\*W = malloc(2 \* sizeof(double \*));

W[0] = malloc(2 \* sizeof(double));

W[1] = malloc(2 \* sizeof(double));

do {

x0 = x;

y0 = y;

W[0][0] = f2(x, y, 1, dif);

W[0][1] = f2(x, y, 2, dif);

W[1][0] = f2(x, y, 3, dif);

W[1][1] = f2(x, y, 4, dif);

inverse\_matrix(W);

x = x - (W[0][0] \* f1(x, y, 1) + W[0][1] \* f1(x, y, 1));

y = y - (W[1][0] \* f1(x, y, 2) + W[1][1] \* f1(x, y, 2));

} while ((fabs(x - x0) > dif) || (fabs(y - y0) > dif));

printf("x = %.\*f\ny = %.\*f\n", e, x, e, y);

}