МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ № 2**

**по дисциплине  
 «МЕТОДЫ ОПТИМИЗАЦИИ»**

Выполнил студент группы 34                                       В.В.Несветайлов

Отчет приняла преподаватель                                                                                А.С. Черная

Краснодар

2025 г.

**Вводные данные:**

f(x) =

M=10

**Задание:**

Найти минимум заданной функции аналитически и численно с использованием трех методов оптимизации:

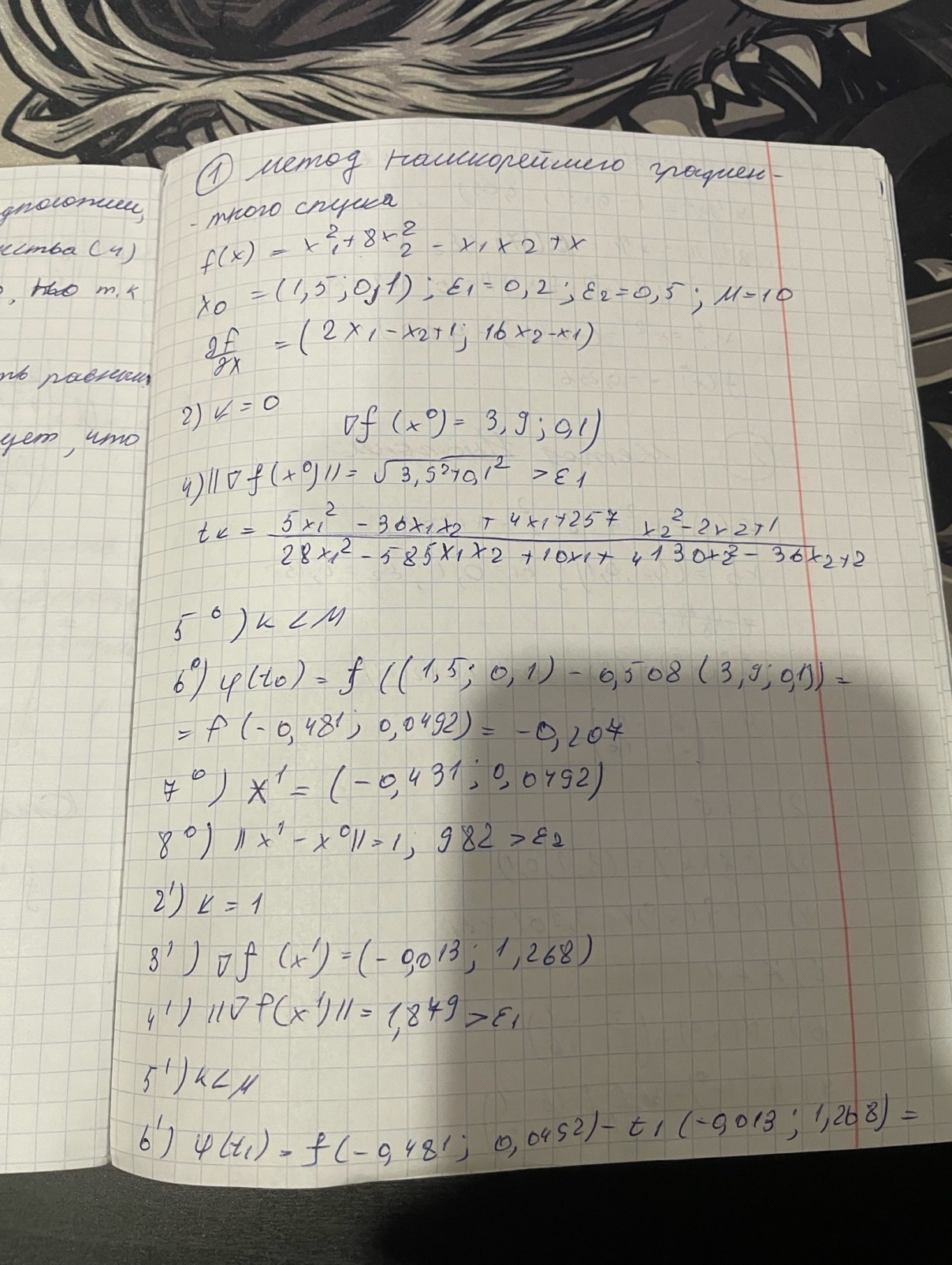
1. Метод наискорейшего градиента
2. Метод Ньютона
3. Метод Ньютона-Рассона
4. Метод Флетчера-Ривса

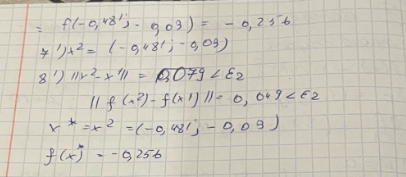
Реализовать программную часть для каждого метода и сравнить их.

**ХОД РАБОТЫ**

**Метод наискорейшего градиентного спуска**

**Аналитическое решение**





**Программное решение**

#include <iostream>

#include <cmath>

using namespace std;

double f(double x1, double x2) {

return x1 \* x1 + 8 \* x2 \* x2 - x1 \* x2 + x1;

}

double dfx1(double x1, double x2) {

return 2 \* x1 - x2 + 1;

}

double dfx2(double x1, double x2) {

return 16 \* x2 - x1;

}

double tk(double x1, double x2) {

return (5 \* x1 \* x1 - 36 \* x1 \* x2 + 4 \* x1 + 257 \* x2 \* x2 - 2 \* x2 + 1)

/ (28 \* x1 \* x1 - 585 \* x1 \* x2 + 10 \* x1 + 4130 \* x2 \* x2 - 36 \* x2 + 2);

}

int main() {

double x1 = 1.5, x2 = 0.1;

double E1 = 0.2, E2 = 0.5;

int M = 10;

int k = 0;

double newx1, newx2;

while (true) {

cout << "\nk = " << k;

cout << "\nx = (" << x1 << " ; " << x2 << ")\n";

double dx1 = dfx1(x1, x2);

double dx2 = dfx2(x1, x2);

if (sqrt(dx1 \* dx1 + dx2 \* dx2) < E1 || k >= M) {

cout << dx1 \* dx1 + dx2 \* dx2;

break;

}

newx1 = x1 - tk(x1, x2) \* dx1;

newx2 = x2 - tk(x1, x2) \* dx2;

if (sqrt(pow(newx1 - x1, 2) + pow(newx2 - x2, 2)) < E2) {

if (abs(f(newx1, newx2) - f(x1, x2)) < E2) {

x1 = newx1;

x2 = newx2;

break;

}

}

x1 = newx1;

x2 = newx2;

k = k + 1;

}

cout << "\nIterations: " << k;

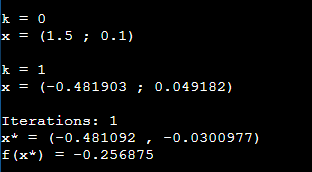
cout << "\nx\* = (" << x1 << " , " << x2 << ")\n";

cout << "f(x\*) = " << f(x1, x2) << "\n";

return 0;

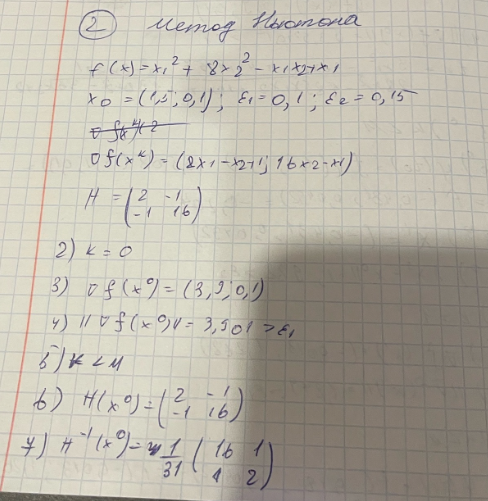
}

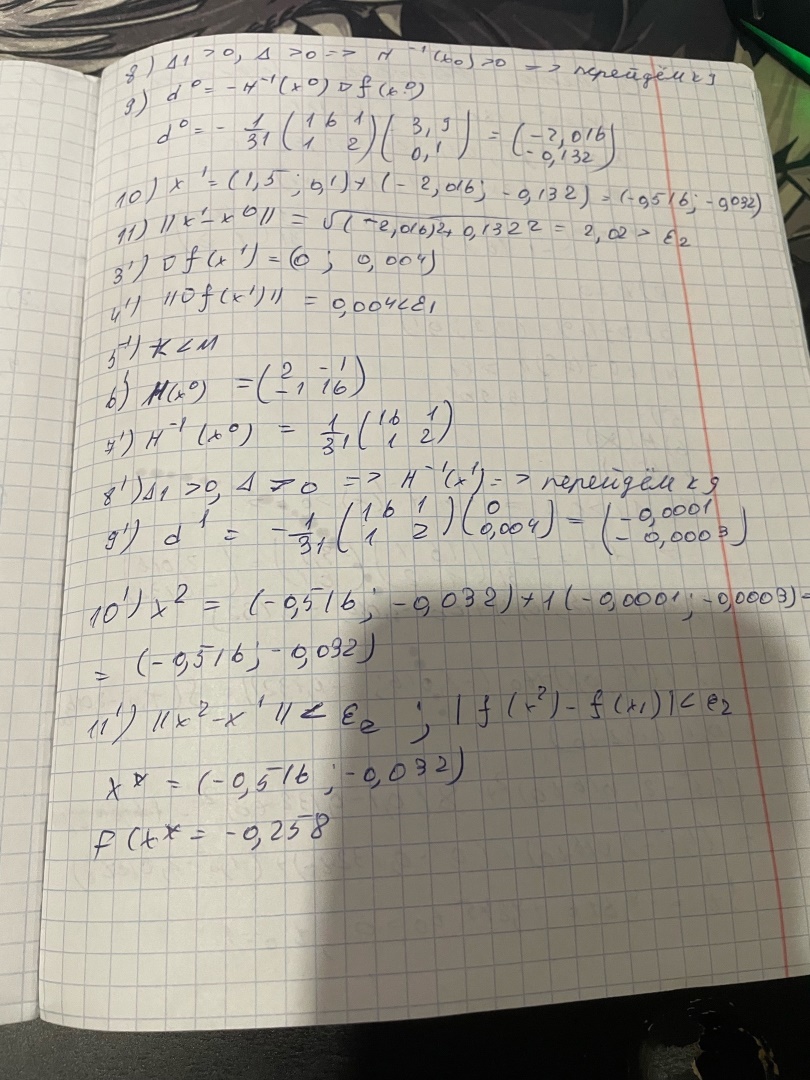
**Результат**



**Метод Ньютона**

**Аналитическое решение**





**Программное решение**

#include <iostream>

#include <cmath>

using namespace std;

double f(double x1, double x2) {

return x1 \* x1 + 8 \* x2 \* x2 - x1 \* x2 + x1; // Примерная функция

}

double dfx1(double x1, double x2) {

return 2 \* x1 - x2 + 1; // производная f по x1

}

double dfx2(double x1, double x2) {

return 16 \* x2 - x1; // производная f по x2

}

int main() {

double x[2] = {1.5, 0.1};

double dx[2], dk[2];

double newx[2];

double Hrev[4] = {0.51613, 0.03125, 0.03125, 0.0625};

double E1 = 0.1, E2 = 0.15;

int M = 10;

int k = 0, iter = -1;

bool flag = false;

while (true) {

cout << "Ank = " << k;

cout << " Anx" << k << " = (" << x[0] << ", " << x[1] << ")\n";

dx[0] = dfx1(x[0], x[1]);

dx[1] = dfx2(x[0], x[1]);

if (sqrt(dx[0] \* dx[0] + dx[1] \* dx[1]) < E1 || k >= M) {

if (flag == true && iter == k - 1)

break;

flag = true;

iter = k;

}

if (Hrev[0] \* Hrev[3] - Hrev[2] \* Hrev[1] > 0) {

dk[0] = -1 \* (Hrev[0] \* dx[0] + Hrev[1] \* dx[1]);

dk[1] = -1 \* (Hrev[2] \* dx[0] + Hrev[3] \* dx[1]);

} else {

dk[0] = -1 \* dx[0];

dk[1] = -1 \* dx[1];

}

newx[0] = x[0] + dk[0];

newx[1] = x[1] + dk[1];

if (sqrt(pow(newx[0] - x[0], 2) + pow(newx[1] - x[1], 2)) < E2) {

if (abs(f(newx[0], newx[1]) - f(x[0], x[1])) < E2) {

x[0] = newx[0];

x[1] = newx[1];

if (flag == true && iter == k - 1)

break;

flag = true;

iter = k;

}

}

x[0] = newx[0];

x[1] = newx[1];

k = k + 1;

}

cout << "\niterations: " << k;

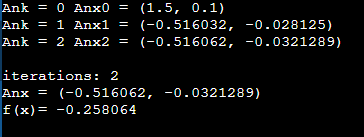
cout << "\nAnx = (" << x[0] << ", " << x[1] << ")\n";

cout << "f(x)= " << f(x[0], x[1]);

return 0;

}

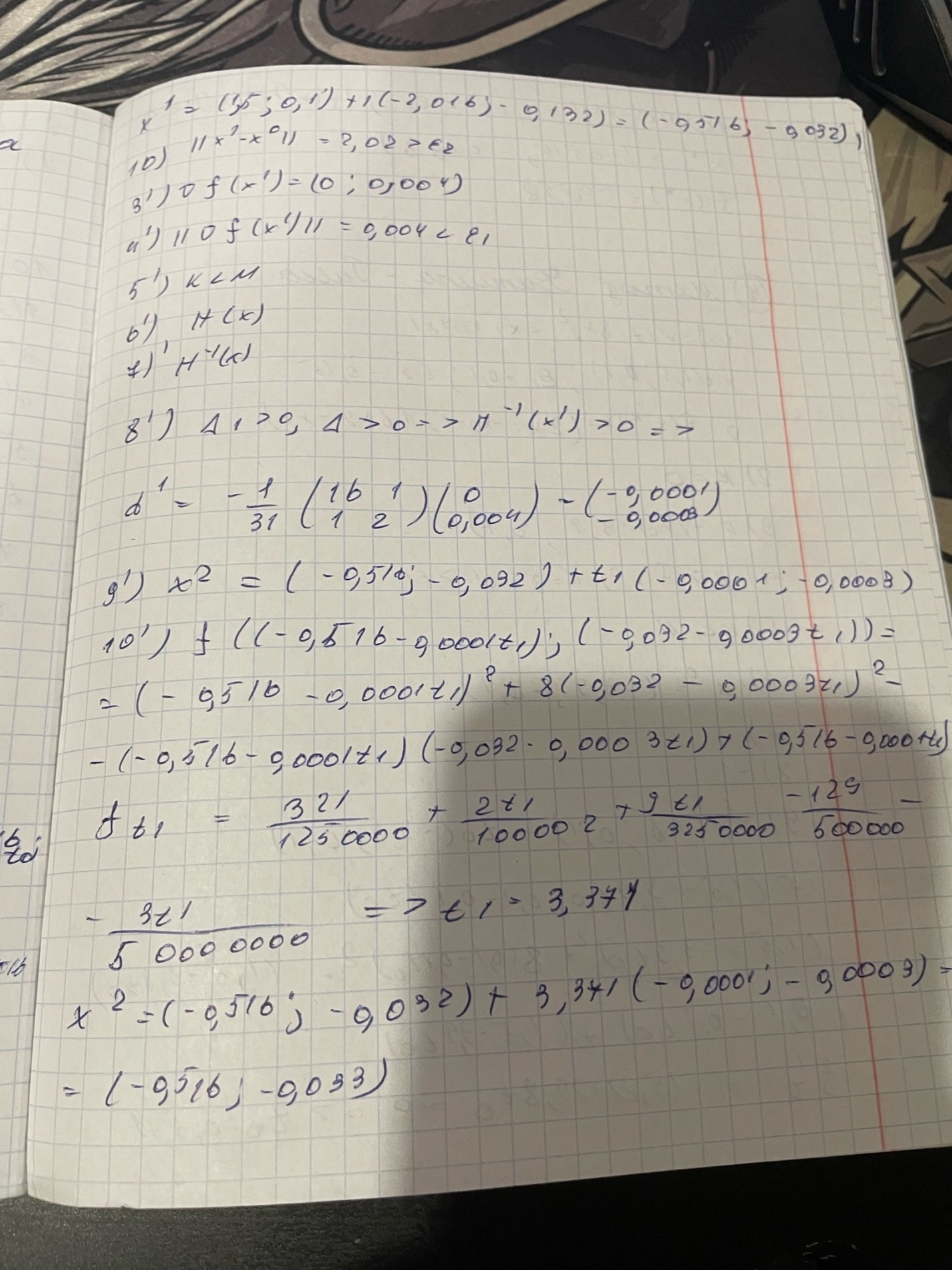
**Результат**

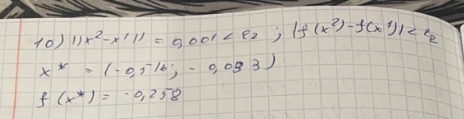


**Метод Ньютона-Рафсона**

**Аналитическое решение**







**Программное решение**

#include <iostream>

#include <cmath>

using namespace std;

double f(double x1, double x2) {

return x1 \* x1 + 8 \* x2 \* x2 - x1 \* x2 + x1;

}

double dfx1(double x1, double x2) {

return 2 \* x1 - x2 + 1;

}

double dfx2(double x1, double x2) {

return 16 \* x2 - x1;

}

double t\_k(double x1, double x2, double d1, double d2) {

return (-2 \* x1 \* d1 + x2 \* d1 - d1 - 16 \* x2 \* d2 + x1 \* d2) / (2 \* d1 \* d1 + 16 \* d2 \* d2 - 2 \* d1 \* d2);

}

int main() {

double x[2] = {1.5, 0.1};

double dx[2], dk[2];

double newx[2];

double Hrev[4] = {0.51613, 0.03125, 0.03125, 0.0625};

double E1 = 0.1, E2 = 0.15;

int M = 10;

int k = 0, iter = -2;

bool flag = false;

while (true) {

cout << " Ank = " << k;

cout << "Anx" << k << " = (" << x[0] << ", " << x[1] << ")\n";

dx[0] = dfx1(x[0], x[1]);

dx[1] = dfx2(x[0], x[1]);

if (sqrt(dx[0] \* dx[0] + dx[1] \* dx[1]) < E1 || k >= M) {

if (flag == true && iter == k - 1)

break;

flag = true;

iter = k;

}

if (Hrev[0] \* Hrev[3] - Hrev[2] \* Hrev[1] > 0) {

dk[0] = -1 \* (Hrev[0] \* dx[0] + Hrev[1] \* dx[1]);

dk[1] = -1 \* (Hrev[2] \* dx[0] + Hrev[3] \* dx[1]);

} else {

dk[0] = -1 \* dx[0];

dk[1] = -1 \* dx[1];

}

newx[0] = x[0] + t\_k(x[0], x[1], dk[0], dk[1]) \* dk[0];

newx[1] = x[1] + t\_k(x[0], x[1], dk[0], dk[1]) \* dk[1];

if (sqrt(pow(newx[0] - x[0], 2) + pow(newx[1] - x[1], 2)) < E2) {

if (abs(f(newx[0], newx[1]) - f(x[0], x[1])) < E2) {

x[0] = newx[0];

x[1] = newx[1];

if (flag == true && iter == k - 1)

break;

flag = true;

iter = k;

}

}

x[0] = newx[0];

x[1] = newx[1];

k = k + 1;

}

cout << "\niterations: " << k;

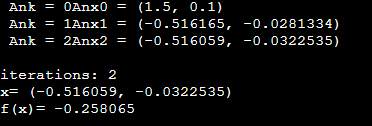
cout << "\nx= (" << x[0] << ", " << x[1] << ")\n";

cout << "f(x)= " << f(x[0], x[1]);

return 0;

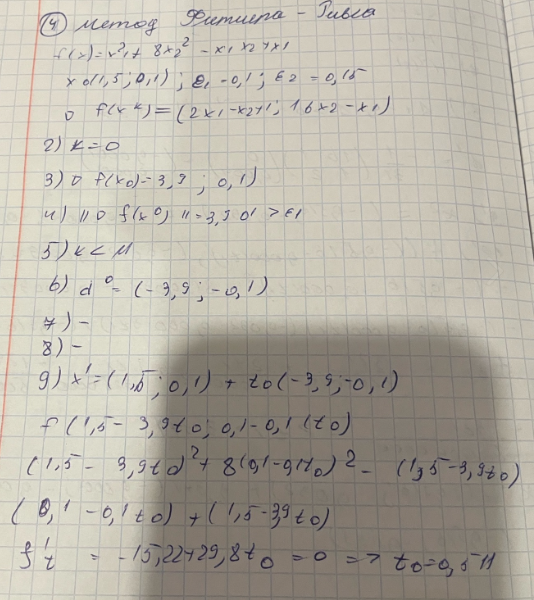
}

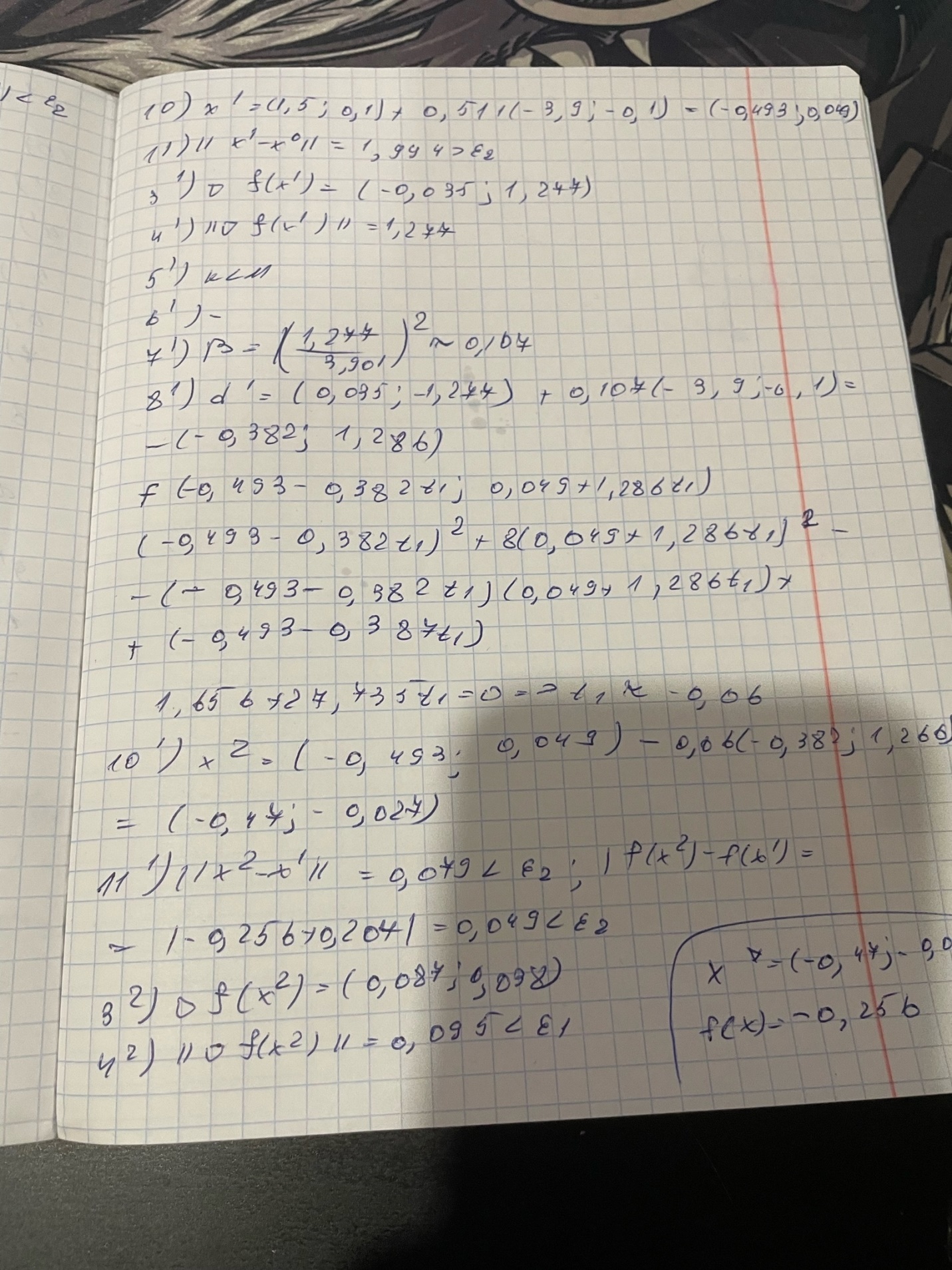
**Результат**



**Метод Флетчера-Ривса**

**Аналитическое решение**





**Программное решение**

#include <iostream>

#include <cmath>

using namespace std;

double f(double x1, double x2) {

return x1 \* x1 + 8 \* x2 \* x2 - x1 \* x2 + x1;

}

double dfx1(double x1, double x2) {

return 2 \* x1 - x2 + 1;

}

double dfx2(double x1, double x2) {

return 16 \* x2 - x1;

}

double t\_k(double x1, double x2, double d1, double d2) {

return (-2 \* x1 \* d1 + x2 \* d1 - d1 - 16 \* x2 \* d2 + x1 \* d2) / (2 \* d1 \* d1 + 16 \* d2 \* d2 - 2 \* d1 \* d2);

}

int main() {

double x[2] = {1.5, 0.1};

double dx[2], dk[2], olddx[2];

double newx[2];

double E1 = 0.1, E2 = 0.15;

double B;

int M = 10;

int k = 0, iter = -1;

bool flag = false;

while (true) {

cout << " Ank = " << k;

cout << "Anx" << k << " = (" << x[0] << ", " << x[1] << ")\n";

olddx[0] = dx[0];

olddx[1] = dx[1];

dx[0] = dfx1(x[0], x[1]);

dx[1] = dfx2(x[0], x[1]);

if (sqrt(dx[0] \* dx[0] + dx[1] \* dx[1]) < E1 || k >= M) {

if (flag == true && iter == k - 1)

break;

flag = true;

iter = k;

}

if (k == 0) {

dk[0] = -1 \* dx[0];

dk[1] = -1 \* dx[1];

} else {

double temp = (sqrt(dx[0] \* dx[0] + dx[1] \* dx[1]) / sqrt(olddx[0] \* olddx[0] + olddx[1] \* olddx[1]));

B = pow(temp, 2);

dk[0] = -1 \* dx[0] + B \* dk[0];

dk[1] = -1 \* dx[1] + B \* dk[1];

}

newx[0] = x[0] + t\_k(x[0], x[1], dk[0], dk[1]) \* dk[0];

newx[1] = x[1] + t\_k(x[0], x[1], dk[0], dk[1]) \* dk[1];

if (sqrt(pow(newx[0] - x[0], 2) + pow(newx[1] - x[1], 2)) < E2) {

if (abs(f(newx[0], newx[1]) - f(x[0], x[1])) < E2) {

x[0] = newx[0];

x[1] = newx[1];

if (flag == true && iter == k - 1)

break;

flag = true;

iter = k;

}

}

x[0] = newx[0];

x[1] = newx[1];

k = k + 1;

}

cout << "\niterations: " << k;

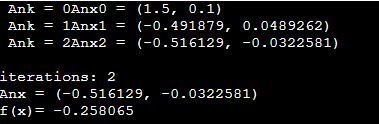
cout << "\nAnx = (" << x[0] << ", " << x[1] << ")\n";

cout << "f(x)= " << f(x[0], x[1]);

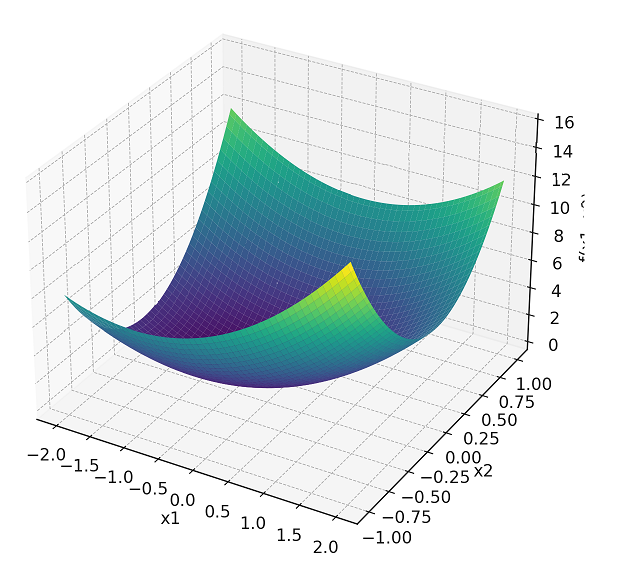
return 0;

}

**Результат**



**Вывод графика функции**



**Вывод**

В рамках лабораторной работы были рассмотрены и проанализированы четыре метода оптимизации: метод наискорейшего градиентного спуска, метод Ньютона, метод Ньютона-Рафсона и метод Флетчера–Ривса. Каждый из них обладает своими достоинствами и особенностями применения.

1. **Метод наискорейшего градиентного спуска** — отличается простотой реализации и хорошей наглядностью, однако чувствителен к выбору шага. При неправильных параметрах может сходиться медленно, особенно в задачах с плохо обусловленной функцией.
2. **Метод Ньютона** — использует информацию о второй производной (гессиан), что обеспечивает быструю сходимость. Однако вычисление и инверсия гессиана может быть ресурсоёмкой задачей в случае многомерных функций.
3. **Метод Ньютона–Рафсона** — представляет собой улучшенную версию метода Ньютона, учитывающую направление предыдущих шагов. Он обеспечивает высокую скорость сходимости при условии удачного выбора начальной точки.
4. **Метод Флетчера–Ривса** — не требует вычисления гессиана, что делает его более практичным для задач с большим числом переменных. Он демонстрирует стабильную и достаточно быструю сходимость.

Из всех исследованных методов наиболее универсальным является **метод Флетчера–Ривса**. Он удачно сочетает преимущества как градиентных, так и ньютоновских методов, оставаясь эффективным при отсутствии информации о второй производной. Благодаря этому он подходит для решения широкого круга оптимизационных задач, в том числе с негладкими или сложными функциями, и может использоваться в различных практических приложениях.