МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп'ютерні інформаційні технології»

**Звіт**

**з навчальної практики**

Виконав:

студент гр.ПЗ1911

Сафонов Д.Є.

Прийняла:

Шаповал І.В.

Дніпро, 2020

# **Постановка завдання**

Ознайомитись з методами визначення мінімуму функції однієї або декількох змінних, методами пошуку нуля (кореня) функції. Провести програмну реалізацію методів у відповідності до обраного рівня складності та оформити звіт. Програма повинна забезпечувати введення наступних даних: границь відрізку на якому розглядається функція (або початкового наближення для методів багатовимірної оптимізації), точності обчислень, вибір функції для розрахунків та метода розрахунку. Відповідь координати точки в якій досягається мінімум або нуль функції, значення мінімуму функції. Значення виводяться з точністю до шести знаків. В доповнення може будуватись графік функції з позначенням отриманого результату.

Перелік методів оптимізації:

Прості:

− метод перебору;

− метод ділення відрізку навпіл;

− метод золотого перерізу;

− метод Фібоначчі;

− метод дотичних;

Середньої складності:

− метод ламаних;

− метод парабол;

Підвищеної складності (методи багатовимірної оптимізації):

− метод градієнтного спуску;

− метод спряжених напрямків.

Перелік методів пошуку нуля функції:

− метод ділення відрізку навпіл;

− метод золотого перерізу;

− метод дотичних.

Рівні складності завдань визначаються наступним чином:

|  |  |  |
| --- | --- | --- |
| Рівень | Вимоги до оформлення програми | Вимоги до змісту завдання |
| А | В незалежності від типу інтерфейсу обов’язків контроль вхідної інформації з реалізацією методів і функцій в окремому файлі(ах). Функція для обчислень повинна передаватись в якості параметра (вказівник на функцію). | Для набору мінімум з трьох різних функцій може бути реалізований один з наступних варіантів: Всі прості методи та один метод середньої складності Всі методи середньої складності Один метод підвищеної складності. Один з методів визначення нуля функції |

# **2. Зовнішні специфікації**

2.1. Знаходження мінімума функції двох аргументів методом спряжених градієнтів:

2.1.1. Формат вхідних даних:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Назва | Умовне позначення | Вимоги до даних | Приклад |
| 1 | Перша координата початкового наближення | x0[0] | Дійсне число | 0.675 |
| 2 | Друга координата початкового наближення | x0[1] | Дійсне число | 7.9785 |
| 3 | Кількість знаків після коми(точність) | epsilon | Ціле число, більше нуля | 5 |
| 4 | Номер функції для мінімізації | n | Ціле число більше нуля та меньше кількості функцій | 2 |

2.1.2. Формат вихідних даних:

-Графік пошуку мінімума

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Назва | Умовне позначення | Вимоги до даних | Приклад |
| 1 | Перша координата мінімума | x0[0] | Дійсне число | 5 |
| 2 | Друга координата мінімума | x0[1] | Дійсне число | 4 |
| 3 | Мінімальне значення функції | func3d[n].Function(x0[0], x0[1]) | Дійсне число | 1 |

2.2. Знаходження мінімума функції методом Фібоначчі:

2.2.1. Формат вхідних даних:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Назва | Умовне позначення | Вимоги до даних | Приклад |
| 1 | Ліва границя пошуку локального мінімума | lim\_left | Дійсне число | 7.7 |
| 2 | Права границя пошуку локального мінімума | lim\_right | Дійсне число | 264.830 |
| 3 | Кількість знаків після коми(точність) | epsilon | Ціле число, більше нуля | 4 |
| 4 | Номер функції для мінімізації | n | Ціле число більше нуля та меньше кількості функцій | 1 |

2.2.2. Формат вихідних даних:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Назва | Умовне позначення | Вимоги до даних | Приклад |
| 1 | Мінімум функції | func2d[n].Function(Fibonacci\_minimization(func2d[n].Function, lim\_left, lim\_right, epsilon)) | Дійсне число | 67 |
| 2 | Аргумент мінімального значення функцій | Fibonacci\_minimization(func2d[n].Function, lim\_left, lim\_right, epsilon) | Дійсне число | 4 |

2.3. Знаходження нуля функції методом бісекції:

2.3.1. Формат вхідних даних:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Назва | Умовне позначення | Вимоги до даних | Приклад |
| 1 | Ліва границя пошуку нуля функції | lim\_left | Дійсне число | 7.7 |
| 2 | Права границя пошуку нуля функції | lim\_right | Дійсне число | 264.830 |
| 3 | Кількість знаків після коми(точність) | epsilon | Ціле число, більше нуля | 4 |
| 4 | Номер функції для пошуку кореня | n | Ціле число більше нуля та меньше кількості функцій | 1 |

2.3.2. Формат вихідних даних:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Назва | Умовне позначення | Вимоги до даних | Приклад |
| 1 | Корінь функції | bisection\_null(func2d[n].Function, lim\_left, lim\_right, epsilon) | Дійсне число | -7578.789 |

2.4. Функціональні вимоги до програми:

Програма повинна забезпечувати введення наступних даних: границь відрізку на якому розглядається функція (або початкового наближення для методів багатовимірної оптимізації), точності обчислень, вибір функції для розрахунків та метода розрахунку. Відповідь координати точки в якій досягається мінімум або нуль функції, значення мінімуму функції. Значення виводяться з точністю до шести знаків. В доповнення може будуватись графік функції з позначенням отриманого результату.

# **Вибір методу рішення задачи**

Для обраного рівню складності були запропоновані два методи пошуку локального екстремуму функцій двох аргументів:

− метод градієнтного спуску;

− метод спряжених напрямків.

Та три методи пошуку нуля функції:

− метод ділення відрізку навпіл;

− метод золотого перерізу;

− метод дотичних.

З методів пошуку локального екстремуму був обраний метод спряженних градієнтів. Обідва методи дуже схожі, через те, що метод спряжених градієнтів є модифікацією методу градієнтного спуску. Але метод спряженних градієнтів на порядок швидше, не зважаючи на трохи складніші математичні розрахунки. Обидва методи знаходять єкстремуми достатньо точно з плавними функціями, та показують відносно велику похибку при роботі з функціями, які різко спадають/зростають.

Метод градієнтного спуску складаеться з послідовного розрахунку:

Де – шаг спуску. Шаг підбирають різними способами, важливо шоб він був не занадто великим – тоді буде велика похибка розрахунків, і не занадто малим – знадобиться дуже багато ітерацій. Наприклад:

Метод градієнтного спуску складаеться з послідовного розрахунку:

Для обох методів – початкове наближення, відносно якого шукается найближчий локальний єкстремум. Умова зупинки – досягнення похідною значення заданої похибки, або меньше, та/або досягнення максимальної кількості ітерацій.

Також для методу спряжених градіентів потрібно було написати якийсь з методів одновимірної мінімізації(для пошуку альфа). Для цього був обраний метод Фибоначі.

Для пошуку нуля функції був обраний метод бісекції, через те що він не потребує використання похідної.

# **Розробка алгоритмів рішення програми**

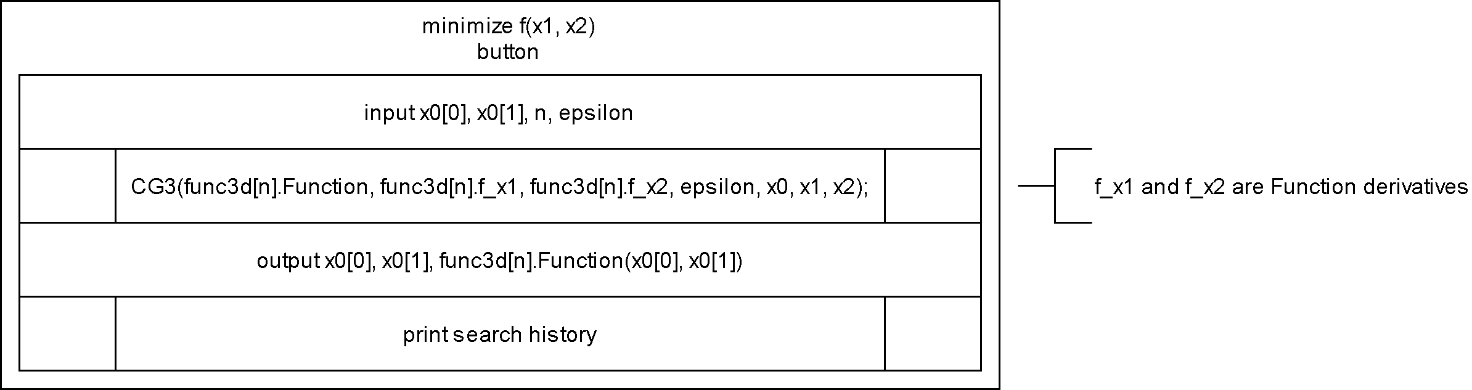


Рисунок 1

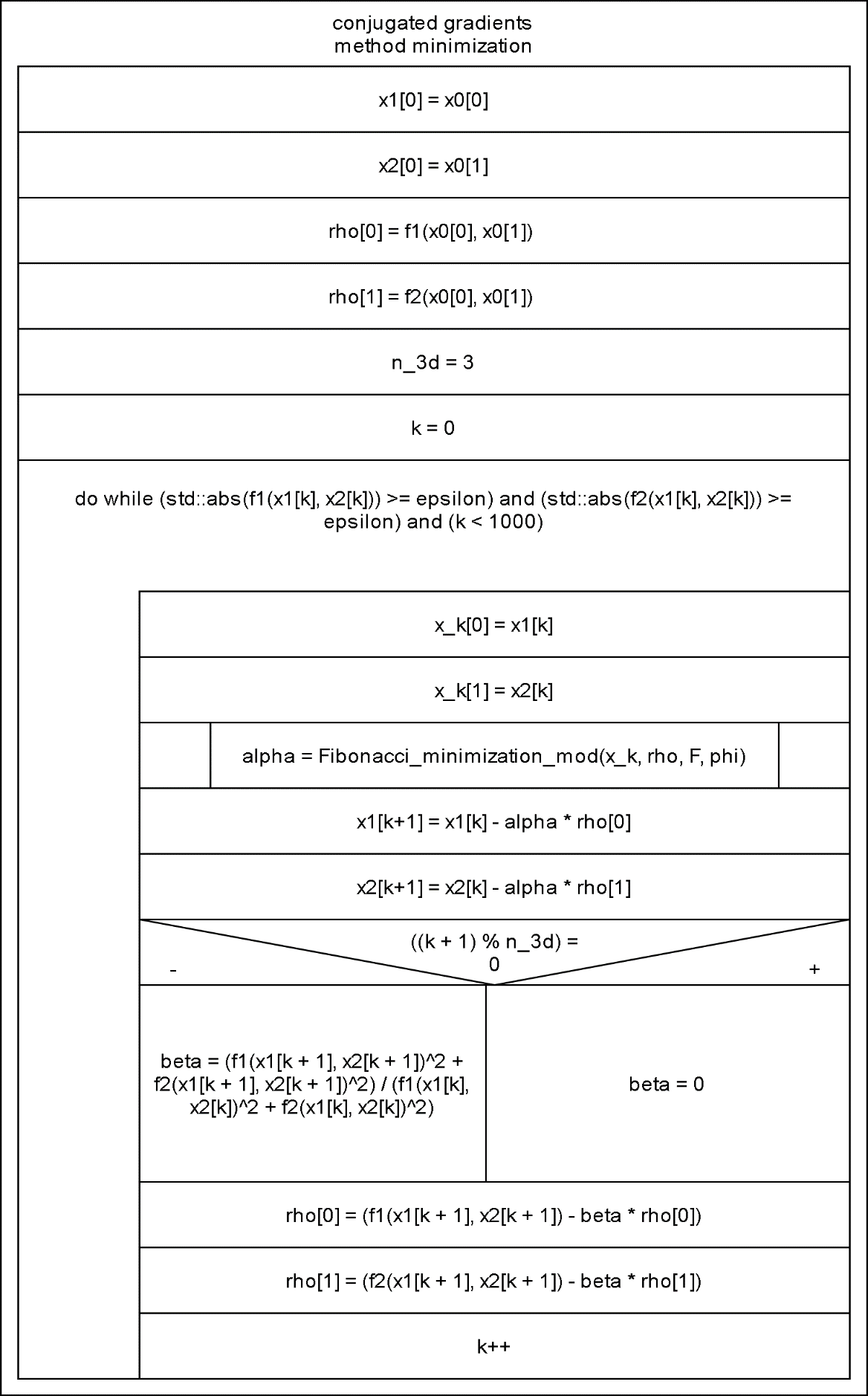


Рисунок 2

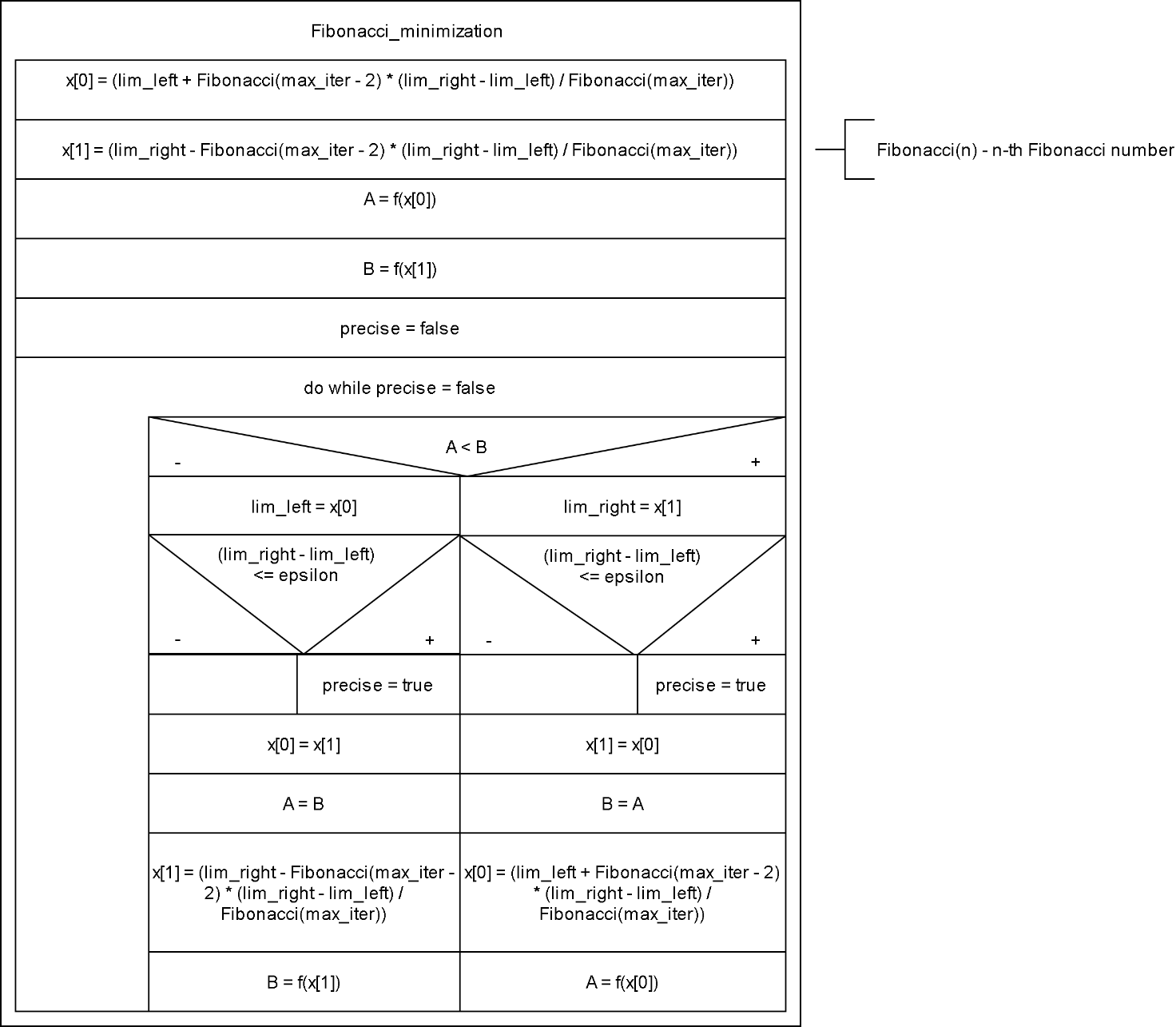


Рисунок 3

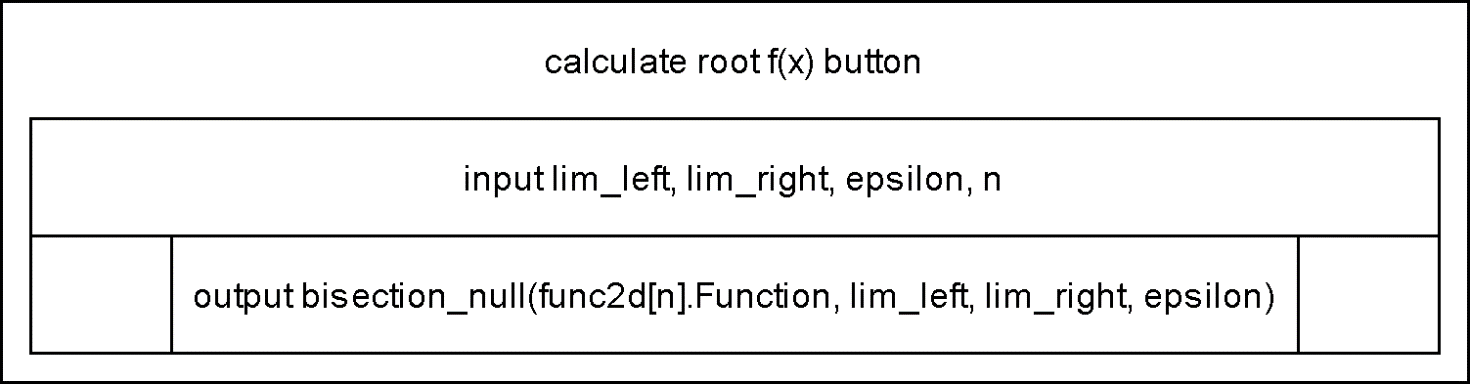


Рисунок 4

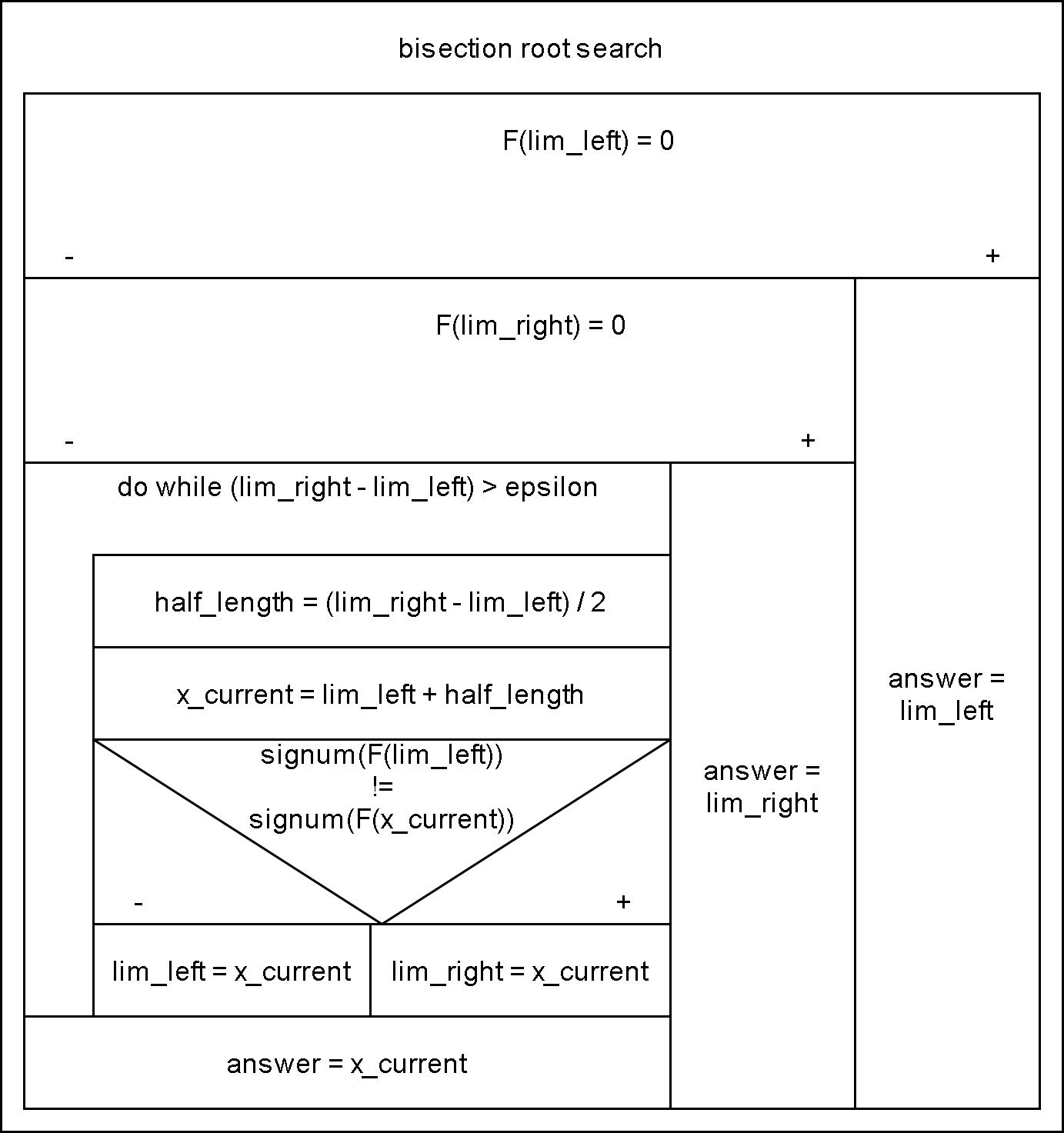


Рисунок 5

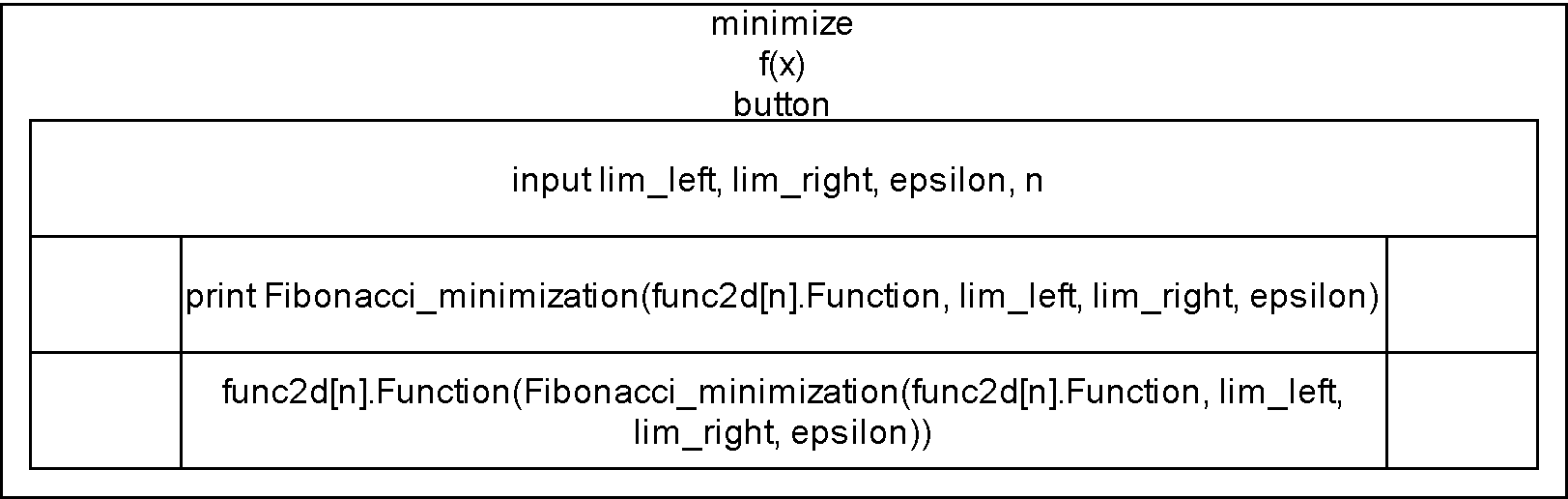


Рисунок 6

# **Текст програми**

“MyForm.cpp”

#include "MyForm.h"

#include "methods.h"

#include "math\_functions\_x.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void main(array<String^>^ arg) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

WEEK1::MyForm form;

Application::Run(% form);

}

“MyForm.h”//buttons only

#include "methods.h"

#include "math\_functions\_x.h"

#include <vector>

#include <string>

std::string f\_ch1 = { "'f(x) = pow(x1, 4) + (2 \* pow(x2, 4)) + (x1 \* x1 \* x2 \* x2) + (2 \* x1) + x2'" };

std::string f\_ch2 = { "'f(x) = ((x1 - 5) \* (x1 - 5)) \* ((x2 - 4) \* (x2 - 4)) + ((x1 - 5) \* (x1 - 5)) + ((x2 - 4) \* (x2 - 4)) + 1'" };

std::string f\_ch3 = { "'f(x) = (x1 \* x1) + (2 \* x2 \* x2) + pow(exp(1), ((x1 \* x1) + (x2 \* x2))) - x1 + (2 \* x2)'" };

std::string f\_ch4 = { "'f(x) = (x1 \* x1) + (2 \* x2 \* x2) + pow(exp(1), ((x1 \* x1) + (x2 \* x2))) - x1 + (2 \* x2)'" };//

std::string f\_ch5 = { "'f(x) = (x1 \* x1) + (2 \* x2 \* x2) + pow(exp(1), ((x1 \* x1) + (x2 \* x2))) - x1 + (2 \* x2)'" };//

std::string f\_ch6 = { "'f(x) = (x1 \* x1) + (2 \* x2 \* x2) + pow(exp(1), ((x1 \* x1) + (x2 \* x2))) - x1 + (2 \* x2)'" };//

Function3d func3d[3]{ {F1\_3d, f1\_x1\_3d, f1\_x2\_3d, &f\_ch1},

{F2\_3d, f2\_x1\_3d, f2\_x2\_3d, &f\_ch2},

{F3\_3d, f3\_x1\_3d, f3\_x2\_3d, &f\_ch3}

};

Function2d func2d[3]{ {F\_sqr, &f\_ch4},

{F2d\_0, &f\_ch5},

{F2d\_1, &f\_ch6},

};

System::Void calculate\_Click(System::Object^ sender, System::EventArgs^ e) {

double x0[2]{Convert::ToDouble(maskedTextBox2->Text), Convert::ToDouble(maskedTextBox3->Text) };

int n = listBox->SelectedIndex;

double epsilon = pow(10, -Convert::ToDouble(maskedTextBox1->Text));

std::vector<double> x1;

std::vector<double> x2;

CG3(func3d[n].Function, func3d[n].f\_x1, func3d[n].f\_x2, epsilon, x0, x1, x2);

textBox4->Text = Convert::ToString(x0[0]);

textBox5->Text = Convert::ToString(x0[1]);

textBox6->Text = Convert::ToString(func3d[n].Function(x0[0], x0[1]));

this->chart1->Series["Series1"]->Points->Clear();

for (int i = 0; i < x1.size(); i++) {

this->chart1->Series["Series1"]->Points->AddXY(x1[i], x2[i]);

}

}

System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

double lim\_left = Convert::ToDouble(maskedTextBox5->Text);

double lim\_right = Convert::ToDouble(maskedTextBox6->Text);

double epsilon = pow(10, -Convert::ToDouble(maskedTextBox4->Text));

int n = listBox1->SelectedIndex;

textBox7->Text = Convert::ToString(bisection\_null(func2d[n].Function, lim\_left, lim\_right, epsilon));

}

System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

double lim\_left, lim\_right, epsilon;

lim\_left = Convert::ToDouble(maskedTextBox8->Text);

lim\_right = Convert::ToDouble(maskedTextBox7->Text);

epsilon = pow(10, -Convert::ToDouble(maskedTextBox9->Text));

int n = listBox1->SelectedIndex;

double x\_min = Fibonacci\_minimization(func2d[n].Function, lim\_left, lim\_right, epsilon);

textBox1->Text = Convert::ToString(x\_min);

textBox2->Text = Convert::ToString(func2d[n].Function(x\_min));

}

“Math\_functions\_x.h”

#ifndef math\_functions\_x\_H

#define math\_functions\_x\_H

#include <string>

typedef double (\*pointFunc3d)(double, double);

typedef double (\*pointFunc2d)(double);

typedef double (\*pointFuncPhi)(double, double\*, double\*, pointFunc3d);

class Function3d {

public:

pointFunc3d Function;

pointFunc3d f\_x1;

pointFunc3d f\_x2;

std::string\* Function\_char;

};

class Function2d {

public:

pointFunc2d Function;

std::string\* Function\_char;

};

double phi(double alpha, double\* x\_k, double\* rho\_k, pointFunc3d F);

double F2d(double x);

double F1\_3d(double x1, double x2);

double f1\_x1\_3d(double x1, double x2);

double f1\_x2\_3d(double x1, double x2);

double F2\_3d(double x1, double x2);

double f2\_x1\_3d(double x1, double x2);

double f2\_x2\_3d(double x1, double x2);

double F3\_3d(double x1, double x2);

double f3\_x1\_3d(double x1, double x2);

double f3\_x2\_3d(double x1, double x2);

double F\_sqr(double x);

double F2d\_0(double x);

double F2d\_1(double x);

double signum(double x);

#endif

“Math\_functions\_x.cpp”

#include "math\_functions\_x.h"

#include <cmath>

double F1\_3d(double x1, double x2) {

return (pow(x1, 4) + (2 \* pow(x2, 4)) + (x1 \* x1 \* x2 \* x2) + (2 \* x1) + x2);

}

double f1\_x1\_3d(double x1, double x2) {

return ((4 \* pow(x1, 3)) + (2 \* x1) + 2);

}

double f1\_x2\_3d(double x1, double x2) {

return ((8 \* pow(x1, 3)) + (2 \* x2) + 1);

}

double phi(double alpha, double\* x\_k, double\* rho\_k, pointFunc3d F) {

return F((x\_k[0] - (alpha \* rho\_k[0])), (x\_k[1] - (alpha \* rho\_k[1])));

}

double F2d(double x) {

return((2 \* x) + (1 / x));

}

double F2\_3d(double x1, double x2) {

return(((x1 - 5) \* (x1 - 5)) \* ((x2 - 4) \* (x2 - 4)) + ((x1 - 5) \* (x1 - 5)) + ((x2 - 4) \* (x2 - 4)) + 1);

}

double f2\_x1\_3d(double x1, double x2) {

return ((2 \* x1) + (((2 \* x1) - 10) \* (x2 - 4) \* (x2 - 4)) - 10);

}

double f2\_x2\_3d(double x1, double x2) {

return((2 \* x2) + ((x1 - 5) \* (x1 - 5) \* ((2 \* x2) - 8)) - 8);

}

double F3\_3d(double x1, double x2) {

return((x1 \* x1) + (2 \* x2 \* x2) + pow(exp(1), ((x1 \* x1) + (x2 \* x2))) - x1 + (2 \* x2));

}

double f3\_x1\_3d(double x1, double x2) {

return ((2 \* x1 \* pow(exp(1), ((x1 \* x1) + (x2 \* x2)))) + (2 \* x1) - 1);

}

double f3\_x2\_3d(double x1, double x2) {

return((2 \* x2 \* pow(exp(1), ((x1 \* x1) + (x2 \* x2)))) + (4 \* x2) + 2);

}

double F\_sqr(double x) {

return((3 \* x \* x) + (6 \* x) - 8);

}

double F2d\_0(double x) {

return((x - 5) / (x + 3));

}

double F2d\_1(double x) {

return(pow(x, 1/3) + (x \* x) - 17);

}

“methods.h”

#ifndef methods\_H

#define methods\_H

#include "math\_functions\_x.h"

#include <vector>

void CG(pointFunc3d F, pointFunc3d f1, pointFunc3d f2, double epsilon, double\* x0);

void CG2(pointFunc3d F, pointFunc3d f1, pointFunc3d f2, double epsilon, double\* x0);

void CG3(pointFunc3d F, pointFunc3d f1, pointFunc3d f2, double epsilon, double\* x0, std::vector<double> &x1, std::vector<double> &x2);

double Fibonacci\_minimization(pointFunc2d f, double lim\_left = 0, double lim\_right = 1, double epsilon = 0.001, int max\_iter = 12);

double Fibonacci\_minimization\_mod(double\* x\_k, double\* rho\_k, pointFunc3d F, pointFuncPhi Phi, double lim\_left = 0, double lim\_right = 1, double epsilon = 0.001, int max\_iter = 12);//or golden ratio, idk

int Fibonacci(int n);

double bisection\_null(pointFunc2d F, double lim\_left, double lim\_right, double epsilon);

#endif

“methods.cpp”

#include "methods.h"

#include "math\_functions\_x.h"

#include <cmath>

#include <vector>

#include <iostream>

void CG(pointFunc3d F, pointFunc3d f1, pointFunc3d f2, double epsilon, double\* x0) {//x0[2]

std::vector<double\*> x{ x0 };//current estimate

double rho[]{ f1(x0[0], x0[1]), f2(x0[0], x0[1]) };//current gradient

double alpha;//current search

double beta;

int n\_3d = 3;

double\* x\_new = new double[2];

for (int k = 0; (std::abs(f1(x[k][0], x[k][1])) >= epsilon) && (std::abs(f2(x[k][0], x[k][1])) >= epsilon) && (k < 1000); k++) {

alpha = Fibonacci\_minimization\_mod(x[k], rho, F, phi);

x\_new[0] = x[k][0] - alpha \* rho[0];

x\_new[1] = x[k][1] - alpha \* rho[1];

x.push\_back(x\_new);

if (((k + 1) % n\_3d) == 0)

beta = 0;

else

beta = (pow(f1(x[k + 1][0], x[k + 1][1]), 2) + pow(f2(x[k + 1][0], x[k + 1][1]), 2)) / (pow(f1(x[k][0], x[k][1]), 2) + pow(f2(x[k][0], x[k][1]), 2));

rho[0] = (f1(x[k + 1][0], x[k + 1][1]) - beta \* rho[0]);

rho[1] = (f2(x[k + 1][0], x[k + 1][1]) - beta \* rho[1]);

}

x0[0] = x.back()[0];

x0[1] = x.back()[1];

}

void CG2(pointFunc3d F, pointFunc3d f1, pointFunc3d f2, double epsilon, double\* x0) {

std::vector<double\*> x{ x0 };//current estimate

double d[]{ -f1(x0[0], x0[1]), -f2(x0[0], x0[1]) };//test

double rho[]{ d[0], d[1] };//current antigradient

double alpha;//current search

double beta;

double\* d\_n = new double[2];//test

double\* x\_new = new double[2];

int n = 3;//for 3d

for (int k = 0; (std::abs(rho[0]) >= epsilon) && (std::abs(rho[1]) >= epsilon) && (k < 1000); k++) {

d\_n[0] = -d[0];

d\_n[1] = -d[1];

alpha = Fibonacci\_minimization\_mod(x[k], d\_n, F, phi);

x\_new[0] = x[k][0] + alpha \* d[0];

x\_new[1] = x[k][1] + alpha \* d[1];

x.push\_back(x\_new);

rho[0] = -f1(x[k + 1][0], x[k + 1][1]);

rho[1] = -f2(x[k + 1][0], x[k + 1][1]);

if (((k + 1) % n) == 0)

beta = 0;

else

beta = -(pow(f1(x[k + 1][0], x[k + 1][1]), 2) + pow(f2(x[k + 1][0], x[k + 1][1]), 2)) / (pow(f1(x[k][0], x[k][1]), 2) + pow(f2(x[k][0], x[k][1]), 2));

d[0] = rho[0] + beta \* d[0];

d[1] = rho[1] + beta \* d[1];

}

x0[0] = x.back()[0];

x0[1] = x.back()[1];

}

void CG3(pointFunc3d F, pointFunc3d f1, pointFunc3d f2, double epsilon, double\* x0, std::vector<double>& x1, std::vector<double>& x2) {//x0[2]

std::vector<double\*> x{ x0 };//current estimate

x1.push\_back(x0[0]);

x2.push\_back(x0[1]);

double rho[]{ f1(x0[0], x0[1]), f2(x0[0], x0[1]) };//current gradient

double alpha;//current search

double beta;

int n\_3d = 3;

double\* x\_new = new double[2];

for (int k = 0; (std::abs(f1(x1[k], x2[k])) >= epsilon) && (std::abs(f2(x1[k], x2[k])) >= epsilon) && (k < 1000); k++) {

double x\_k[2]{x1[k], x2[k]};///

alpha = Fibonacci\_minimization\_mod(x\_k, rho, F, phi);

x\_new[0] = x1[k] - alpha \* rho[0];

x\_new[1] = x2[k] - alpha \* rho[1];

x.push\_back(x\_new);

x1.push\_back(x\_new[0]);

x2.push\_back(x\_new[1]);

if (((k + 1) % n\_3d) == 0)

beta = 0;

else {

beta = (pow(f1(x1[k + 1], x2[k + 1]), 2) + pow(f2(x1[k + 1], x2[k + 1]), 2)) / (pow(f1(x[k][0], x[k][1]), 2) + pow(f2(x[k][0], x[k][1]), 2));//error in denominator

//beta = (pow(f1(x1[k + 1], x2[k + 1]), 2) + pow(f2(x1[k + 1], x2[k + 1]), 2)) / (pow(f1(x1[k], x2[k]), 2) + pow(f2(x1[k], x2[k]), 2));///

//std::cout << x[k][0] << " " << x[k][0] << " \* " << x1[k] << " " << x2[k] << std::endl;

}

rho[0] = (f1(x1[k + 1], x2[k + 1]) - beta \* rho[0]);///

rho[1] = (f2(x1[k + 1], x2[k + 1]) - beta \* rho[1]);///

}

x0[0] = x1.back();

x0[1] = x2.back();

}

double Fibonacci\_minimization(pointFunc2d f, double lim\_left, double lim\_right, double epsilon, int max\_iter) {//or golden ratio, idk

double x[2];

x[0] = (lim\_left + Fibonacci(max\_iter - 2) \* (lim\_right - lim\_left) / Fibonacci(max\_iter));

x[1] = (lim\_right - Fibonacci(max\_iter - 2) \* (lim\_right - lim\_left) / Fibonacci(max\_iter));

double A = f(x[0]);

double B = f(x[1]);

bool precise = false;

do {

if (A < B) {

lim\_right = x[1];

if ((lim\_right - lim\_left) <= epsilon)

precise = true;

x[1] = x[0];

B = A;

x[0] = (lim\_left + Fibonacci(max\_iter - 2) \* (lim\_right - lim\_left) / Fibonacci(max\_iter));

A = f(x[0]);

}

else {

lim\_left = x[0];

if ((lim\_right - lim\_left) <= epsilon)

precise = true;

x[0] = x[1];

A = B;

x[1] = (lim\_right - Fibonacci(max\_iter - 2) \* (lim\_right - lim\_left) / Fibonacci(max\_iter));

B = f(x[1]);

}

} while (!precise);

return ((lim\_left + lim\_right) / 2);

}

int Fibonacci(int n) {

int Fib[2]{ 1, 1 };

for (int i = 0; i < (n - 2); i++) {

int temp = Fib[0] + Fib[1];

Fib[0] = Fib[1];

Fib[1] = temp;

}

return Fib[1];

}

double Fibonacci\_minimization\_mod(double\* x\_k, double\* rho\_k, pointFunc3d F, pointFuncPhi Phi, double lim\_left, double lim\_right, double epsilon, int max\_iter) {//or golden ratio, idk

double x[2];

x[0] = (lim\_left + Fibonacci(max\_iter - 2) \* (lim\_right - lim\_left) / Fibonacci(max\_iter));//alpha

x[1] = (lim\_right - Fibonacci(max\_iter - 2) \* (lim\_right - lim\_left) / Fibonacci(max\_iter));//alpha

double A = Phi(x[0], x\_k, rho\_k, F);

double B = Phi(x[1], x\_k, rho\_k, F);

bool precise = false;

do {

if (A < B) {

lim\_right = x[1];

if ((lim\_right - lim\_left) <= epsilon)

precise = true;

x[1] = x[0];

B = A;

x[0] = (lim\_left + Fibonacci(max\_iter - 2) \* (lim\_right - lim\_left) / Fibonacci(max\_iter));

A = Phi(x[0], x\_k, rho\_k, F);

}

else {

lim\_left = x[0];

if ((lim\_right - lim\_left) <= epsilon)

precise = true;

x[0] = x[1];

A = B;

x[1] = (lim\_right - Fibonacci(max\_iter - 2) \* (lim\_right - lim\_left) / Fibonacci(max\_iter));

B = Phi(x[1], x\_k, rho\_k, F);

}

} while (!precise);

return ((lim\_left + lim\_right) / 2);

}

double bisection\_null(pointFunc2d F, double lim\_left, double lim\_right, double epsilon) {

double answer;

if (F(lim\_left) == 0)

answer = lim\_left;

else if (F(lim\_right) == 0)

answer = lim\_right;

else {

double half\_length;

double x\_current;

while ((lim\_right - lim\_left) > epsilon) {

half\_length = (lim\_right - lim\_left) / 2;

x\_current = lim\_left + half\_length;

if (signum(F(lim\_left)) != signum(F(x\_current)))

lim\_right = x\_current;

else

lim\_left = x\_current;

}

answer = x\_current;

}

return answer;

}

double signum(double x) {

if (x == 0)

return 0;

if (x < 0)

return -1;

return 1;//if x>0

}

# **Контрольний приклад**

Метод спряжених градієнтів.

Real answer:

Метод Фібоначі.

A<B

A>B

A<B

A>B

A<B (on 4th digit after point)

={0.38, 0.33, 0.355}

Real answer = 0.3863…

Метод бісекції.

on [0;9], 1 digit after point

Answer = 5.0625

Real = 5

# **Тестування програми**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва | Вхідні дані | Очікуваний результат |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |

# **Аналіз результатів тестування і роботи програми**

Усі градіентні методи грунтуються на методі градіентного спуску та є його оптімізаціями. Обраний мною метод спряжених градієнтів – ітеративна модифікація методу градіентного спуску. Цей метод, як і усі інші базує свої розрахунки на послідовному додаванні/відніманні градієнту функції, помноженному на шаг, доки градіент більший за точність та/або до максимальної кількості ітерацій. Але для функцій, які зростастають/спадають дуже швидко наявна дуже велика похибка розрахунку, яку можливо компенсувати зменьшенням шагу. Але при використанні формули, яка описана раніше, з кількістю ітерацій росте й похибка, не зважаючі на обнулення градіенту кожні N ітерацій. Загалом, якщо функція достатньо плавна, локальний єкстремум знаходиться правильно відповідно до заданої точності.

Щодо методу бісекції, він порівнює знаки початку та середини і відрізає праву половину, якщо знаки однакові, якщо ні – ліву; і так доки довжина відрізку не стане меньшою потрібної точності. Належно вказувати відрізок, який не має розривів, бо вони можуть розпізнаватися цим методом, як нулі. Також треба перевіряти відрізки на наявність коренів.

# **Література**

1. «Численные методы анализа» Б.П. Медович, И.А. Марон, Э.З. Шувалова.
2. <https://habr.com/ru/post/413853/>
3. <https://basegroup.ru/community/articles/conjugate>