МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «Програмна інженерія та інформаційні технології управління»

Звіт з лабораторної роботи №2

з дисципліни «Дослідження операцій»

Виконав:

Студент групи КН-36а

Рубан Ю. Д.

Перевірив:

Проф. Гужва В. О.

Харків – 2018

**Ціль роботи:** Розв’язати задачу мінімізації методом Розенброка з дискретним кроком.

**Задача:**

Задана функція:

Характер екстремуму: мінімум.

Початкова точка: (0, 0)

Альфа = 1,5. Бета = -0,5. Точність = 0,0001. Крок = 0,001.

**Алгоритм досліджуваного метода:**

1. Покласти , в якості направлення обрати координатні направлення та обрати початкову довжину кроку, обрати початкову точку і покласти
2. Розрахувати , , якщо то і перейти на крок 3. Інакше і перейти на крок 3.
3. Якщо , то і перейти на крок 2, інакше на крок 4
4. Якщо то і перейти на крок 2. Інакше якщо то перейти на крок 5 інакше і перейти на крок 2.
5. то пошук завершено:

*.* Інакше розрахувати з відношення

1. Розрахувати та

і перейти на крок 2.

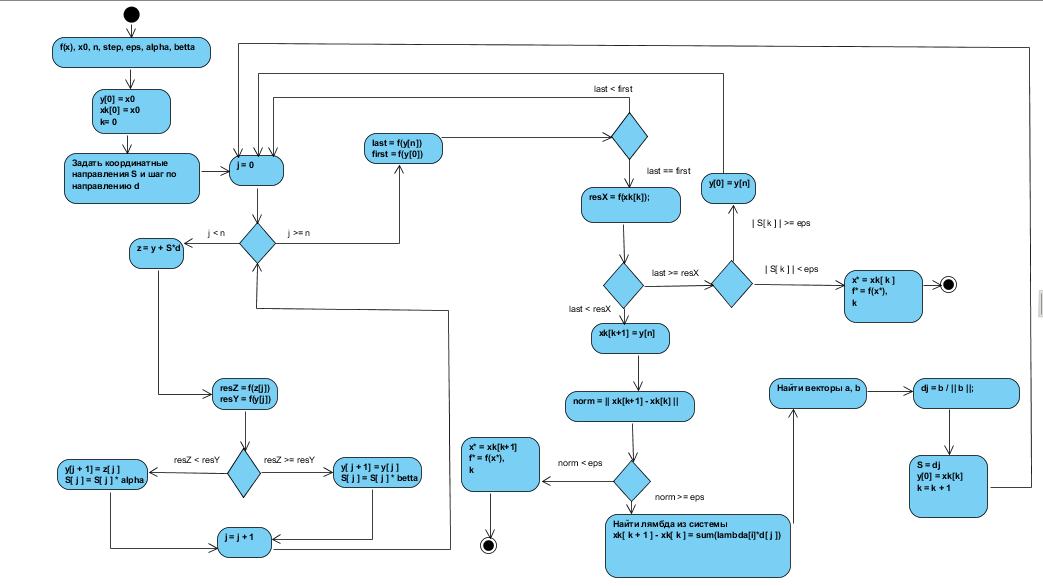
Алгоритм метода Розенброка показаний на рисунку 1.

Рисунок 1 – Блок схема алгоритму Розенброка

**Код розробленої програми:**

Код програми показаний у лістингу 1.

Лістинг 1. Код алгоритму Розенброка з дискретним кроком

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AproximationMethodsLib

{

public class RosenbrockMethod : IMultipleExtremumMethod

{

private double eps, alpha, betta,step;

public IEnumerable<double> StartPoint { get; set; }

public double Eps { get { return eps; } set

{

eps = value > 0 ? value : throw new InvalidOperationException("Епсилон должно быть положительно");

}

}

public double Alpha { get { return alpha; } set

{

alpha = value > 1 ? value : throw new InvalidOperationException("Альфа должна быть больше 1");

} }

public double Betta { get { return betta; } set

{

betta = ((value > -1) && (value < 0)) ? value : throw new InvalidOperationException("Бетта должна быть в диапазоне (-1; 0)");

} }

public double Step { get { return step; } set

{

step = value > 0 ? value : throw new InvalidOperationException("Шаг должен быть больше нуля");

} }

public RosenbrockMethod(IEnumerable<double> startPoint, double step,double eps, double alpha, double betta)

{

this.StartPoint = startPoint;

this.Eps = eps;

this.Alpha = alpha;

this.Betta = betta;

this.Step = step;

}

private double Norm(IEnumerable<double> a)

{

double res = 0;

foreach(double e in a)

{

res += e \* e;

}

return Math.Sqrt(res);

}

private double VecMul(List<double> a, List<double> b)

{

double res = 0;

int n = a.Count;

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

{

res += a[i] \* b[i];

}

return res;

}

public IEnumerable<double> FindExtremum(Func<IEnumerable<double>, double> func, int n, ExtremumType extremumType, Logger logger = null)

{

//begin

var f = extremumType == ExtremumType.Min ? func : (x) => -func(x);

logger?.Log("Начальная точка X0 =");

for (int c = 0; c < StartPoint.Count(); c++)

{

logger?.Log("x"+(c+1).ToString()+": "+StartPoint.ElementAt(c));

}

logger?.Log("Точность: "+Eps);

logger?.Log("Альфа: " + Alpha);

logger?.Log("Бетта: " + Betta);

logger?.Log("Шаг по всем направлениям: " + Step);

logger?.Log("Установка начальных значений");

List<Queue<double>> S = new List<Queue<double>>(n);

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

{

S.Add(new Queue<double>(n));

for (int q = 0; q < n; q++)

{

S[i].Enqueue(i == q ? 1 : 0);

}

}

int k = 0;

List<List<double>> y = new List<List<double>>

{

StartPoint.Select(x => x).ToList()

};

List<List<double>> xk = new List<List<double>>

{

StartPoint.Select(x => x).ToList()

};

List<List<double>> dS = new List<List<double>>();

dS.AddRange(S.Select(x => x.Select(r =>r).ToList()));

List<double> d = new List<double>();

d.AddRange(Enumerable.Range(0, n).Select(x => Step));

List<List<double>> z = new List<List<double>>(n);

logger?.Log("Начинается выполнение основного этапа");

int stepOneIter=0, stepThreeIter=0, stepFourIter=0;

//second

Func<IEnumerable<double>> StepOneThree = () =>

{

stepThreeIter++;

Action StepOne = () =>

{

stepOneIter++;

z = new List<List<double>>(n);

for (int j = 0; j < n; j++)

{

//first

z.Add(new List<double>(n));

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

{

z[j].Add(y[j][i] + dS[j][i] \* d[i]);

}

double resZ = f(z[j]);

double resY = f(y[j]);

if (resZ < resY)

{

y.Add(z[j]);

dS[j] = dS[j].Select(x => Alpha \* x).ToList();

}

else

{

y.Add(y[j]);

dS[j] = dS[j].Select(x => Betta \* x).ToList();

}

}

};

StepOne();

//third

while (true)

{

double last = f(y[n]);

double first = f(y[0]);

if (last < first)

{

var t = y[n];

LogVector(logger, f, t, "y");

y = new List<List<double>>();

y.Add(t);

StepOne();

}

else if (first == last)

{

double resXK = f(xk[k]);

if (last <= resXK)

{

return null;

}

else

{

if (dS[k].All(x => Math.Abs(x) < Eps))

{

logger?.Log("Шаг по всем направлениям меньше Eps");

return xk[k];//end

}

else

{

var t = y[n];

LogVector(logger, f, t, "y");

y = new List<List<double>>();

y.Add(t);

StepOne();

}

}

}

}

};

var result = StepOneThree();

while (result == null)

{

stepFourIter++;

//step 4

xk.Add(y[n]);

LogVector(logger, f, xk.Last(), "x");

List<double> tempDif = new List<double>();

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

{

tempDif.Add(xk[k+1][i] - xk[k][i]);

}

double norm = Norm(tempDif);

if (norm < Eps)

{

logger?.Log("||X(k+1) - X(k)|| < Eps");

logger?.Log("Алгоритм завершен!");

logger?.Log("Итераций на первом шаге: " + stepOneIter);

logger?.Log("Итераций на третьем шаге: " + stepThreeIter);

logger?.Log("Итераций на четвертом шаге: " + stepFourIter);

logger?.Log("Всего итераций: " + (stepOneIter+stepFourIter+stepThreeIter));

logger?.Log("X\* = ");

for (int i = 0; i < xk.Last().Count(); i++)

{

logger?.Log("x" + (i + 1).ToString() + ": " + xk.Last().ElementAt(i));

}

logger?.Log("f\*(X\*) = " + f(xk.Last()));

return xk.Last();

}

else

{

logger?.Log("Поиск значений лямбда");

List<double> lambda = SOLE.Solution(dS, tempDif).ToList();

List<List<double>> a = new List<List<double>>();

List<List<double>> b = new List<List<double>>();

logger?.Log("Вычисление матриц a, b");

for (int i = 0; i < lambda.Count; i++)

{

if (lambda[i] == 0)

{

a.Add(dS[i]);

}

else

{

List<double> res = new List<double>();

List<List<double>> buffer = new List<List<double>>();

for (int q = i; q < n; q++)

{

buffer.Add(dS[q].Select(x => x \* lambda[q]).ToList());

}

for (int o = 0; o < n; o++)

{

double sum = 0;

for (int u = 0; u < buffer.Count; u++)

{

sum += buffer[u][o];

}

res.Add(sum);

}

a.Add(res);

}

}//find a

b.Add(a[0]);

for (int i = 1; i < n; i++)

{

List<List<double>> buffer = new List<List<double>>();

List<double> res = new List<double>();

for (int j = 0; j < i - 1; j++)

{

double m = VecMul(a[j], dS[i]);

buffer.Add(dS[i].Select(x => x \* m).ToList());

}

for (int q = 0; q < n; q++)

{

double sum = 0;

for (int w = 0; w < buffer.Count; w++)

{

sum += buffer[w][q];

}

res.Add(a[i][q] - sum);

}

b.Add(res);

}

List<List<double>> dj = new List<List<double>>(n);

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

{

double bNorm = Norm(b[i]);

dj.Add(new List<double>(n));

for (int o = 0; o < n; o++)

{

dj[i].Add(b[i][o] / bNorm);

}

}

dS = dj;

y[0] = xk[k];

k++;

//go to step 1

}

result = StepOneThree();

}

logger?.Log("Алгоритм завершен!");

logger?.Log("Итераций на первом шаге: " + stepOneIter);

logger?.Log("Итераций на третьем шаге: " + stepThreeIter);

logger?.Log("Итераций на четвертом шаге: " + stepFourIter);

logger?.Log("Всего итераций: " + (stepOneIter + stepFourIter + stepThreeIter));

logger?.Log("X\* = ");

for (int i = 0; i < result.Count(); i++)

{

logger?.Log("x" + (i + 1).ToString() + ": " + result.ElementAt(i));

}

logger?.Log("f\*(X\*) = " + f(result));

return result;

}

private void LogVector(Logger logger,Func<IEnumerable<double>,double> f, IEnumerable<double>vec, string varName)

{

logger?.Log($"{varName.ToUpper()} = ");

for (int i = 0; i < vec.Count(); i++)

{

logger?.Log($"{varName.ToLower()}" + (i + 1).ToString() + ": " + vec.ElementAt(i));

}

logger?.Log($"f({varName.ToUpper()}) = " + f(vec));

}

}

}

**Результати роботи програми:**

Результати роботи програми показані на рисунках 5 – 6.

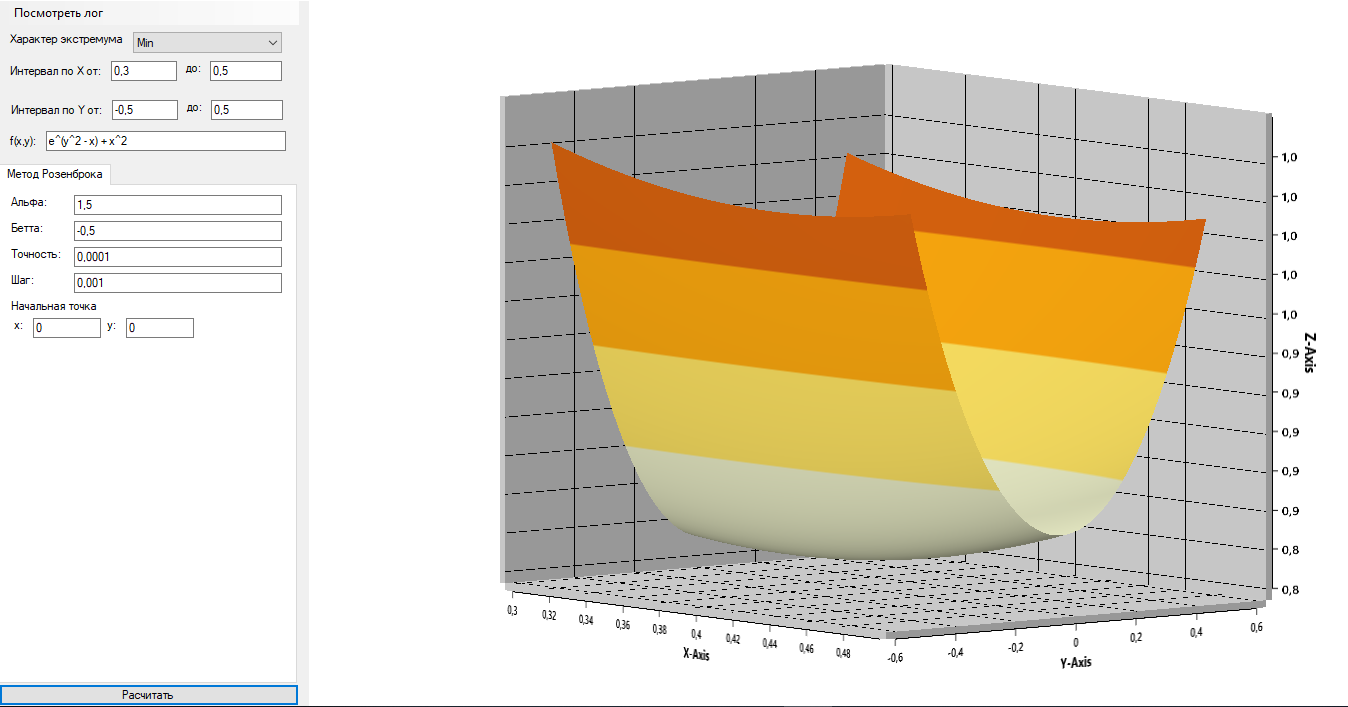


Рисунок 5 – Інтерфейс програми

Результати виводу програми:

===========================================

Алгоритм поиска экстремума: Метод Розенброка

Начало поиска экстремума для функции: e^(y^2 - x) + x^2

Характер экстремума: Min

Начальная точка X0 =

x1: 0

x2: 0

Точность: 0,0001

Альфа: 1,5

Бетта: -0,5

Шаг по всем направлениям: 0,001

Установка начальных значений

Начинается выполнение основного этапа

Y =

y1: 0,001

y2: 0

f(Y) = 0,999001499833375

Y =

y1: 0,0025

y2: 0

f(Y) = 0,99750937239746

Y =

y1: 0,00475

y2: 0

f(Y) = 0,995283825909212

Y =

y1: 0,008125

y2: 0

f(Y) = 0,991973934222632

Y =

y1: 0,0131875

y2: 0

f(Y) = 0,987072984251225

Y =

y1: 0,02078125

y2: 0

f(Y) = 0,979865052499808

Y =

y1: 0,032171875

y2: 0

f(Y) = 0,969375163855933

Y =

y1: 0,0492578125

y2: 0

f(Y) = 0,954362009235017

Y =

y1: 0,07488671875

y2: 0

f(Y) = 0,93345660886843

Y =

y1: 0,113330078125

y2: 0

f(Y) = 0,905699605878331

Y =

y1: 0,1709951171875

y2: 0

f(Y) = 0,872065018923348

Y =

y1: 0,25749267578125

y2: 0

f(Y) = 0,839289765901725

Y =

y1: 0,387239013671875

y2: 0

f(Y) = 0,828882855975923

X =

x1: 0,387239013671875

x2: 0

f(X) = 0,828882855975923

Поиск значений лямбда

Вычисление матриц a, b

Y =

y1: 0,387239013671875

y2: 0

f(Y) = 0,828882855975923

Y =

y1: 0,387239013671875

y2: 0

f(Y) = 0,828882855975923

Y =

y1: 0,386989013671875

y2: 0

f(Y) = 0,828859052387947

Y =

y1: 0,386614013671875

y2: 0

f(Y) = 0,828823660966161

Y =

y1: 0,386051513671875

y2: 0

f(Y) = 0,828771280314781

Y =

y1: 0,385207763671875

y2: 0

f(Y) = 0,828694299160029

Y =

y1: 0,383942138671875

y2: 0

f(Y) = 0,828582405336698

Y =

y1: 0,382043701171875

y2: 0

f(Y) = 0,828422617628573

Y =

y1: 0,379196044921875

y2: 0

f(Y) = 0,828201064613437

Y =

y1: 0,374924560546875

y2: 0

f(Y) = 0,827909555575339

Y =

y1: 0,368517333984375

y2: 0

f(Y) = 0,827564243997656

Y =

y1: 0,358906494140625

y2: 0

f(Y) = 0,827253528034937

X =

x1: 0,358906494140625

x2: 0

f(X) = 0,827253528034937

Поиск значений лямбда

Вычисление матриц a, b

Y =

y1: 0,358906494140625

y2: 0

f(Y) = 0,827253528034937

Y =

y1: 0,357406494140625

y2: 0

f(Y) = 0,827227504174896

Y =

y1: 0,355156494140625

y2: 0

f(Y) = 0,827199857588975

Y =

y1: 0,351781494140625

y2: 0

f(Y) = 0,827184029213797

X =

x1: 0,351781494140625

x2: 0

f(X) = 0,827184029213797

Поиск значений лямбда

Вычисление матриц a, b

Y =

y1: 0,351781494140625

y2: 0

f(Y) = 0,827184029213797

Y =

y1: 0,351781494140625

y2: 0

f(Y) = 0,827184029213797

Y =

y1: 0,351781494140625

y2: 0

f(Y) = 0,827184029213797

Y =

y1: 0,351781494140625

y2: 0

f(Y) = 0,827184029213797

Y =

y1: 0,351781494140625

y2: 0

f(Y) = 0,827184029213797

Y =

y1: 0,351687744140625

y2: 0

f(Y) = 0,827184028983716

X =

x1: 0,351687744140625

x2: 0

f(X) = 0,827184028983716

||X(k+1) - X(k)|| < Eps

Алгоритм завершен!

Итераций на первом шаге: 39

Итераций на третьем шаге: 4

Итераций на четвертом шаге: 4

Всего итераций: 47

X\* =

x1: 0,351687744140625

x2: 0

f\*(X\*) = 0,827184028983716

===========================================

**Результат обчислень:**

k = 4

**Висновки:**

У даній роботі було досліджено алгоритм Розенброка з дискретним кроком, було програмно реалізовано цей алгоритм, та мінімізовано функцію індивідуального завдання.