МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ НЭТИ | Факультет прикладной математики и информатики

Кафедра прикладной математики и информатики

Лабораторная работа №4
по дисциплине «Методы оптимизации»

Статистические методы

Студенты	БЕГИЧЕВ АЛЕКСАНДР
	ШИШКИН НИКИТА
Бригада	08
Группа	ПМ-92
Преподаватель	ФИЛИППОВА Е.В.

Новосибирск, 2022

Цель работы

Ознакомиться со статистическими методами поиска при решении задач нелинейного программирования. Изучить методы случайного поиска при определении глобального экстремума функции.

Задание

Найти **максимум** заданной функции:

$$f(x,y) = \sum_{i=1}^{6} \frac{C_i}{1 + (x - a_i)^2 + (y - b_i)^2}$$

на области $-10 \leq x \leq 10$, $-10 \leq y \leq 10$.

C_1	C_2	C_4	C_5	C_6	a_1	a_2	a_3	a_4	a_5	a_6	b_1	b_2	b_3	b_3	b_4	b_5	b_6
2	1	7	2	8	4	5	2	-9	0	-3	-3	4	0	-6	-3	7	3

1. Разработать программу для решения задачи поиска глобального экстремума с использованием **метода простого случайного поиска и трех алгоритмов глобального поиска**.
2. Исследовать метод простого случайного поиска глобального экстремума при различных ε и P. Результат представить в таблице:

ε	P	N	$(\hat{x}, \hat{y})$	$f(\hat{x}, \hat{y})$

3. Исследовать алгоритмы поиска глобального экстремума. Сравнить результаты поиска по количеству вычислений функций и найденной точке экстремума. Исследование провести при различных значениях числа попыток m.
4. П.3 повторить при пяти разных начальных значениях ГСЧ. Сделать выводы об устойчивости различных алгоритмов.

Выполнение работы

С помощью подпрограммы на python оценим нашу функцию.

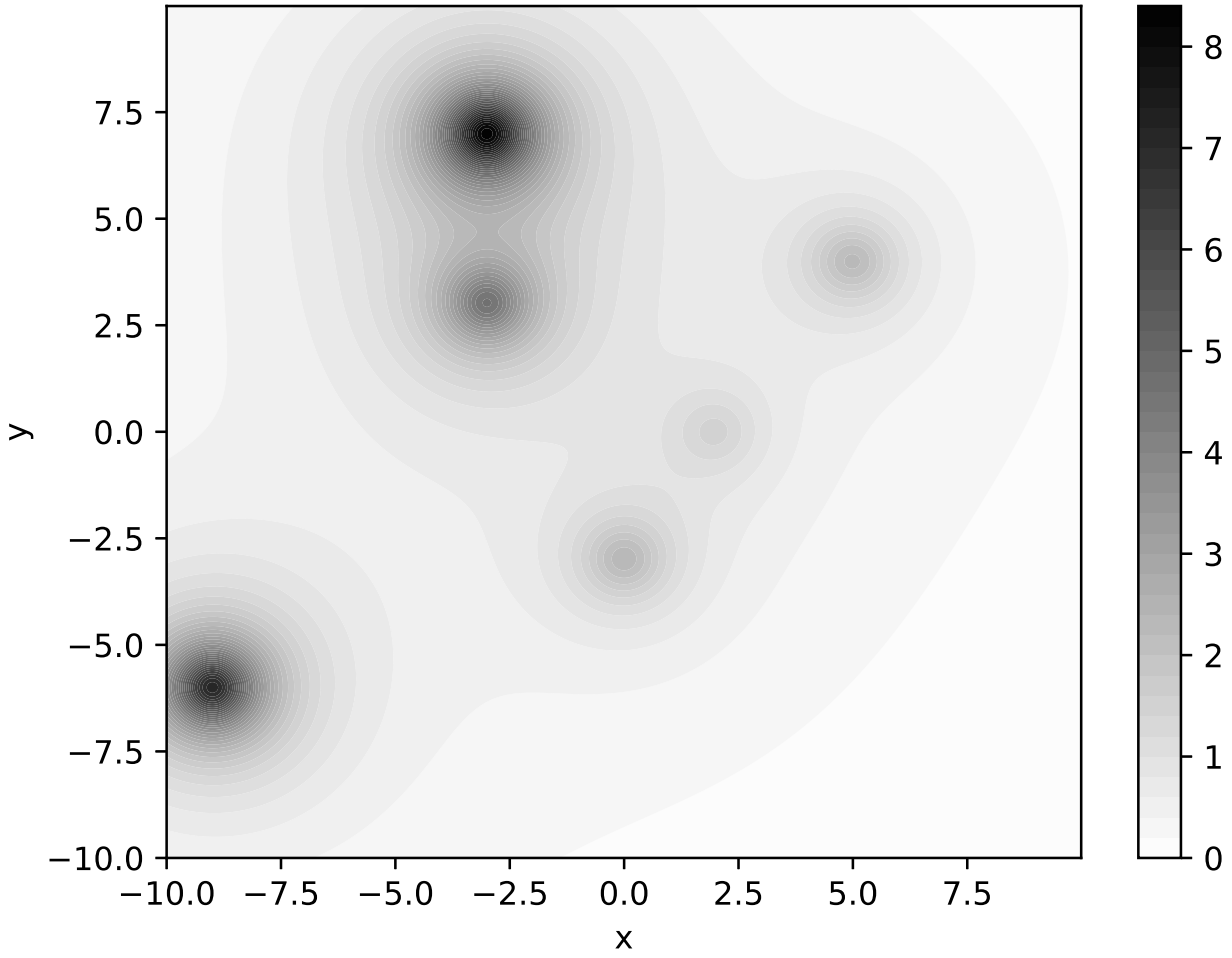

Рис. 1: Заданная функция

Наибольшее значение функции в заданной области:

$$\max_{\Omega} f(x, y) = -8.32829340$$

Точка максимума:

$$(x, y) = (-2.99958534, 6.99227345)$$

Исследование метода простого случайного поиска

ε	P	N	$(\hat{x}, \hat{y})$	$f(\hat{x}, \hat{y})$
0.1	0.6	36652	$(-2.9328, 7.0202)$	8.2865
0.025	0.6	586426	$(-2.9907, 6.9770)$	8.3258
0.01	0.6	3665163	$(-2.9911, 6.9870)$	8.3275
0.005	0.6	14660652	$(-2.9993, 6.9891)$	8.3282
0.1	0.7	48159	$(-3.0521, 6.9877)$	8.3060
0.025	0.7	770542	$(-3.0049, 7.0062)$	8.3265
0.01	0.7	4815891	$(-3.0023, 6.9914)$	8.3282
0.005	0.7	19263565	$(-2.9991, 6.9932)$	8.3282
0.1	0.8	64377	$(-2.9864, 6.9647)$	8.3208
0.025	0.8	1030040	$(-3.0071, 6.9951)$	8.3277
0.01	0.8	6437751	$(-2.9942, 6.9954)$	8.3279
0.005	0.8	25751006	$(-3.0007, 6.9936)$	8.3282

Вывод: метод простого случайного поиска напрямую зависит от значений ε и P. При увеличении вероятности P метод стремиться к истинному значению (глобальному экстремуму), а при увеличении значения ε вырастает точность найденного решения. Чем больше значение переменных P и ε, тем больше значение количества требуемых экспериментов N.

Исследование алгоритмов глобального поиска

Алгоритм №1

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-4}$	2,578	10	-2.9992	6.9946	8.3283
$1 \cdot 10^{-4}$	5,152	20	-3.0005	6.9935	8.3283
$1 \cdot 10^{-4}$	7,335	30	-2.9983	6.9916	8.3283
$1 \cdot 10^{-4}$	9,639	40	-3.0004	6.9909	8.3283

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-5}$	3,065	10	-2.99923	6.99195	8.32829
$1 \cdot 10^{-5}$	6,016	20	-3.00010	6.99218	8.32829
$1 \cdot 10^{-5}$	8,282	30	-2.99930	6.99199	8.32829
$1 \cdot 10^{-5}$	11,198	40	-2.99921	6.99245	8.32829

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-6}$	3,204	10	-2.999722	6.992299	8.328293
$1 \cdot 10^{-6}$	6,964	20	-2.999472	6.992277	8.328293
$1 \cdot 10^{-6}$	10,606	30	-2.999465	6.992157	8.328293
$1 \cdot 10^{-6}$	12,532	40	-2.999587	6.992410	8.328293

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,445	10	-2.9995696	6.9922169	8.3282934
$1 \cdot 10^{-7}$	7,389	20	-2.9996049	6.9922426	8.3282934
$1 \cdot 10^{-7}$	11,284	30	-2.9995624	6.9923165	8.3282934
$1 \cdot 10^{-7}$	14,408	40	-2.9995922	6.9922258	8.3282934

Алгоритм №2

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-4}$	1,870	10	-2.9957	3.0264	4.6347
$1 \cdot 10^{-4}$	3,880	20	-2.9989	3.0226	4.6347
$1 \cdot 10^{-4}$	5,905	30	-9.0005	-6.0015	7.1077
$1 \cdot 10^{-4}$	8,483	40	-2.9982	6.9916	8.3283

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-5}$	2,288	10	-2.99918	6.99251	8.32829
$1 \cdot 10^{-5}$	4,683	20	-8.99851	-5.99954	7.10778
$1 \cdot 10^{-5}$	6,667	30	-8.99973	-5.99898	7.10778
$1 \cdot 10^{-5}$	8,890	40	-2.99915	6.99263	8.32829

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-6}$	2,183	10	-8.999352	-5.999124	7.107785
$1 \cdot 10^{-6}$	5,233	20	-2.999653	6.992369	8.328293
$1 \cdot 10^{-6}$	8,050	30	-2.999328	6.992184	8.328293
$1 \cdot 10^{-6}$	10,400	40	-8.998974	-5.999024	7.107784

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,120	10	-2.9995358	6.9922084	8.3282934
$1 \cdot 10^{-7}$	5,755	20	-2.9995314	6.9922925	8.3282934
$1 \cdot 10^{-7}$	8,349	30	-8.9991412	-5.9990615	7.1077847
$1 \cdot 10^{-7}$	11,686	40	-2.9995703	6.9922296	8.3282934

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-8}$	3,378	10	-8.99917922	-5.99909334	7.10778472
$1 \cdot 10^{-8}$	6,777	20	-2.99958935	6.99229481	8.32829341
$1 \cdot 10^{-8}$	10,638	30	-2.99958301	6.99229391	8.32829341
$1 \cdot 10^{-8}$	14,540	40	-2.99958796	6.99229373	8.32829341

Хочется сразу сказать пару слов об этом алгоритме. В описании алгоритма говорится, что для начала должна быть получена точка, которая является локальным экстремумом, однако возможен случай, когда данная точка будет уже являться глобальным экстремумом и последующий поиск новой точки через алгоритм ненаправленного случайного поиска может не прийти к решению. Поэтому данный алгоритм неустойчив.

Алгоритм №3

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-4}$	4,573	10	-8.9974	-5.9984	7.1078
$1 \cdot 10^{-4}$	9,978	20	-3.0008	6.9933	8.3283
$1 \cdot 10^{-4}$	14,429	30	-2.9983	6.9922	8.3283
$1 \cdot 10^{-4}$	18,085	40	-8.9991	-5.9974	7.1078

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-5}$	5,070	10	-2.99946	6.99168	8.32829
$1 \cdot 10^{-5}$	11,599	20	-2.99958	6.99173	8.32829
$1 \cdot 10^{-5}$	16,008	30	-2.99912	6.99241	8.32829
$1 \cdot 10^{-5}$	20,378	40	-8.99926	-5.99967	7.10778

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-6}$	5,715	10	-8.999022	-5.999206	7.107785
$1 \cdot 10^{-6}$	11,730	20	-2.999483	6.992178	8.328293
$1 \cdot 10^{-6}$	17,970	30	-2.999435	6.992311	8.328293
$1 \cdot 10^{-6}$	24,350	40	-2.999430	6.992380	8.328293

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	6,824	10	-2.9996487	6.9923286	8.3282934
$1 \cdot 10^{-7}$	13,547	20	-2.9996233	6.9922558	8.3282934
$1 \cdot 10^{-7}$	20,412	30	-2.9995626	6.9922401	8.3282934
$1 \cdot 10^{-7}$	27,536	40	-2.9995865	6.9922356	8.3282934

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-8}$	7,848	10	-2.99959658	6.99227584	8.32829341
$1 \cdot 10^{-8}$	14,633	20	-2.99960549	6.99228217	8.32829341
$1 \cdot 10^{-8}$	23,015	30	-2.99958244	6.99226299	8.32829341
$1 \cdot 10^{-8}$	30,889	40	-2.99959341	6.99227933	8.32829341

Вывод: данные алгоритмы имеют прямую зависимость от точности ε и числа попыток m, так как при большем значении точности ε и количестве попыток m возрастает число оценки функции.

Исследование на устойчивость алгоритмов глобального поиска

Алгоритм №1

ГСЧ №1

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,686	10	-2.9995460	6.9922771	8.3282934
$1 \cdot 10^{-7}$	7,035	20	-2.9995460	6.9922771	8.3282934
$1 \cdot 10^{-7}$	10,459	30	-2.9995460	6.9922771	8.3282934
$1 \cdot 10^{-7}$	13,985	40	-2.9995460	6.9922771	8.3282934

ГСЧ №2

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,980	10	-2.9996332	6.9923047	8.3282934
$1 \cdot 10^{-7}$	7,376	20	-2.9996332	6.9923047	8.3282934
$1 \cdot 10^{-7}$	10,929	30	-2.9996103	6.9922708	8.3282934
$1 \cdot 10^{-7}$	14,550	40	-2.9996103	6.9922708	8.3282934

ГСЧ №3

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,783	10	-2.9995541	6.9923023	8.3282934
$1 \cdot 10^{-7}$	7,261	20	-2.9995541	6.9923023	8.3282934
$1 \cdot 10^{-7}$	10,664	30	-2.9995559	6.9923004	8.3282934
$1 \cdot 10^{-7}$	14,146	40	-2.9995559	6.9923004	8.3282934

ГСЧ №4

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,612	10	-2.9996030	6.9923224	8.3282934
$1 \cdot 10^{-7}$	7,369	20	-2.9996030	6.9923224	8.3282934
$1 \cdot 10^{-7}$	10,944	30	-2.9996030	6.9923224	8.3282934
$1 \cdot 10^{-7}$	14,471	40	-2.9996030	6.9923224	8.3282934

ГСЧ №5

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,432	10	-2.9995929	6.9923198	8.3282934
$1 \cdot 10^{-7}$	6,977	20	-2.9995929	6.9923198	8.3282934
$1 \cdot 10^{-7}$	10,403	30	-2.9995929	6.9923198	8.3282934
$1 \cdot 10^{-7}$	13,827	40	-2.9995441	6.9922683	8.3282934

Алгоритм №2

ГСЧ №1

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	2,975	10	-2.9995194	6.9922965	8.3282934
$1 \cdot 10^{-7}$	6,085	20	-2.9995194	6.9922965	8.3282934
$1 \cdot 10^{-7}$	9,395	30	-2.9995194	6.9922965	8.3282934
$1 \cdot 10^{-7}$	12,905	40	-2.9995194	6.9922965	8.3282934

ГСЧ №2

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,191	10	-2.9996103	6.9922708	8.3282934
$1 \cdot 10^{-7}$	6,320	20	-2.9996103	6.9922708	8.3282934
$1 \cdot 10^{-7}$	9,669	30	-2.9996103	6.9922708	8.3282934
$1 \cdot 10^{-7}$	13,179	40	-2.9996103	6.9922708	8.3282934

ГСЧ №3

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	2,958	10	-8.9991403	-5.9990645	7.1077847
$1 \cdot 10^{-7}$	5,958	20	-8.9991403	-5.9990645	7.1077847
$1 \cdot 10^{-7}$	9,360	30	-2.9995252	6.9922672	8.3282934
$1 \cdot 10^{-7}$	12,900	40	-2.9995252	6.9922672	8.3282934

ГСЧ №4

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,129	10	-2.9996547	6.9923179	8.3282934
$1 \cdot 10^{-7}$	6,359	20	-2.9996547	6.9923179	8.3282934
$1 \cdot 10^{-7}$	9,789	30	-2.9996547	6.9923179	8.3282934
$1 \cdot 10^{-7}$	13,419	40	-2.9996547	6.9923179	8.3282934

ГСЧ №5

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	3,238	10	-2.9996161	6.9923195	8.3282934
$1 \cdot 10^{-7}$	6,558	20	-2.9996161	6.9923195	8.3282934
$1 \cdot 10^{-7}$	10,078	30	-2.9996161	6.9923195	8.3282934
$1 \cdot 10^{-7}$	13,798	40	-2.9996161	6.9923195	8.3282934

Алгоритм №3

ГСЧ №1

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	6,659	10	-2.9995194	6.9922965	8.3282934
$1 \cdot 10^{-7}$	13,443	20	-2.9995934	6.9922339	8.3282934
$1 \cdot 10^{-7}$	20,545	30	-2.9995934	6.9922339	8.3282934
$1 \cdot 10^{-7}$	27,705	40	-2.9995934	6.9922339	8.3282934

ГСЧ №2

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	6,863	10	-2.9995406	6.9922968	8.3282934
$1 \cdot 10^{-7}$	13,791	20	-2.9995599	6.9923082	8.3282934
$1 \cdot 10^{-7}$	20,879	30	-2.9995599	6.9923082	8.3282934
$1 \cdot 10^{-7}$	27,804	40	-2.9995680	6.9922987	8.3282934

ГСЧ №3

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	6,412	10	-8.9992036	-5.9990731	7.1077847
$1 \cdot 10^{-7}$	12,848	20	-2.9995956	6.9922256	8.3282934
$1 \cdot 10^{-7}$	19,659	30	-2.9995956	6.9922256	8.3282934
$1 \cdot 10^{-7}$	26,322	40	-2.9995956	6.9922256	8.3282934

ГСЧ №4

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	6,943	10	-2.9996449	6.9922706	8.3282934
$1 \cdot 10^{-7}$	13,819	20	-2.9996154	6.9922954	8.3282934
$1 \cdot 10^{-7}$	20,949	30	-2.9996154	6.9922954	8.3282934
$1 \cdot 10^{-7}$	28,073	40	-2.9996154	6.9922954	8.3282934

ГСЧ №5

ε	calls	m	x	y	$f(\overline{x})$
$1 \cdot 10^{-7}$	6,876	10	-2.9995435	6.9922949	8.3282934
$1 \cdot 10^{-7}$	13,727	20	-2.9995435	6.9922949	8.3282934
$1 \cdot 10^{-7}$	21,110	30	-2.9995435	6.9922949	8.3282934
$1 \cdot 10^{-7}$	27,962	40	-2.9996221	6.9922696	8.3282934

Вывод: из проведенного исследования можно сказать, что алгоритмы №1 и №3 являются устойчивыми, так как при различных ГСЧ приходят к истинному решению, однако алгоритм №3 основан на случайном направлении и может прийти в локальный экстремум, что мы наблюдаем при ГСЧ №3. Про алгоритм №2 было сказано в исследовании выше, но при ограничении генерации точек в алгоритме ненаправленного случайного поиска он сходится к нужному решению.

Листинг

Program.cs

```
using oMethods_4;

Solver solver = Solver.CreateBuilder()
.SetFunction(new Function())
.SetMethod(new SimpleRandomSearch(0.1, 0.999))
.SetArea(Rectangle.ReadJson("area.json")!.Value);

solver.Compute();
```

Solver.cs

```
namespace oMethods_4;

public class Solver {
    public class SolverBuilder {
        private readonly Solver _solver = new();

        public SolverBuilder SetFunction(IFunction function) {
            _solver._function = function;
            return this;
        }

        public SolverBuilder SetMethod(ISearchMethod2D method) {
            _solver._method = method;
            return this;
        }

        public SolverBuilder SetArea(Rectangle rectangle) {
            _solver._rectangle = rectangle;
            return this;
        }

        public static implicit operator Solver(SolverBuilder builder)
            => builder._solver;
    }

    private IFunction _function = default!;
    private ISearchMethod2D _method = default!;
    private Rectangle _rectangle = default!;

    public void Compute() {
        try {
            ArgumentNullException.ThrowIfNull(_function,
                                              $"{nameof(_function)} cannot be null,
                                              ↪  set the function");

             ArgumentNullException.ThrowIfNull(_rectangle,
                                              $"{nameof(_rectangle)} cannot be null,
                                              ↪  set the area");

            ArgumentNullException.ThrowIfNull(_method,
                                              $"{nameof(_method)} cannot be null, set
                                              ↪  the method of minimization");

            _method.Compute(_rectangle, _function);
        } catch (Exception ex) {
```

```
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }

    public static SolverBuilder CreateBuilder()
        => new();
}
```

IMethods.cs

```
namespace oMethods_4;

public interface IMinMethod2D {
    public Vector2D? Min { get; }
    public double Eps { get; init; }

    public void Compute(IFunction function, Vector2D initPoint);
}

public interface ISearchMethod2D {
    public Vector2D? Min { get; }
    public double Eps { get; init; }

    public void Compute(Rectangle rectangle, IFunction function);
}
```

Function.cs

```
namespace oMethods_4;

public interface IFunction {
    public double Value(Vector2D point);
}

public class Function : IFunction {
    public double Value(Vector2D point)
        => -((2.0 / (1 + ((point.X - 5) * (point.X - 5)) + ((point.Y - 4) * (point.Y - 4)))) +
           (1.0 / (1 + ((point.X - 2) * (point.X - 2)) + (point.Y * point.Y))) +
           (7.0 / (1 + ((point.X + 9) * (point.X + 9)) + ((point.Y + 6) * (point.Y + 6)))) +
           (2.0 / (1 + (point.X * point.X) + ((point.Y + 3) * (point.Y + 3)))) +
           (8.0 / (1 + ((point.X + 3) * (point.X + 3)) + ((point.Y - 7) * (point.Y - 7)))) +
           (4.0 / (1 + ((point.X + 3) * (point.X + 3)) + ((point.Y - 3) * (point.Y - 3)))));
}
```

SimpleRandomSearches.cs

```
namespace oMethods_4;

public class UndirectedSimpleRandomSearch : ISearchMethod2D {
    private Vector2D? _min;
    public Vector2D? Min => _min;
    public double Eps { get; init; }
    public double Probability { get; init; }

    public UndirectedSimpleRandomSearch(double eps, double probability)
```

```
        => (Eps, Probability) = (eps, probability);

    public void Compute(Rectangle rectangle, IFunction function) {
        double functionMaxValue = function.Value(rectangle.LeftBottom);

        int tests = 1;
        double neighbourhood = Eps * Eps;
        double probabilityEps = neighbourhood / rectangle.Square;

        while (tests < Math.Log(1.0 - Probability) / Math.Log(1.0 - probabilityEps)) {
            tests++;
        }

        for (int itest = 0; itest < tests; itest++) {
            double newX = new Random().NextDouble(rectangle.LeftBottom.X,
            ↪  rectangle.RightBottom.X);
            double newY = new Random().NextDouble(rectangle.LeftBottom.Y,
            ↪  rectangle.LeftTop.Y);

            double temp;

            if ((temp = function.Value(new(newX, newY))) < functionMaxValue) {
                _min = new(newX, newY);
                functionMaxValue = temp;
            }
        }

        Console.WriteLine($"N: {tests}");
        Console.WriteLine($"P: {Probability}");
        Console.WriteLine($"Extremum: {_min}");
        Console.WriteLine($"f(extremum) = {-functionMaxValue}");
    }
}

public class SimplexMethod : IMinMethod2D {
    private Vector2D? _min;
    public Vector2D? Min => _min;
    public int MaxIters { get; init; }
    public double Step { get; init; }
    public double Eps { get; init; }

    public SimplexMethod(int maxIters, double step, double eps) {
        MaxIters = maxIters;
        Step = step;
        Eps = eps;
    }

    public void Compute(IFunction function, Vector2D initPoint) {
        Vector2D[] points = new Vector2D[3];

        double d1 = Step * (Math.Sqrt(2 + 1) + 2 - 1) / (2 * Math.Sqrt(2));
        double d2 = Step / (2 * Math.Sqrt(2) * (Math.Sqrt(2 + 1) - 1));

        points[0] = initPoint;
        points[1] = new(d1, d2);
        points[2] = new(d2, d1);

        for (int iter = 0; iter < MaxIters; iter++) {
            Vector2D xG;
            Vector2D xR;
```

```
            Vector2D xC;
            Vector2D xE;

            double sumX = 0;
            double sumY = 0;

            points = points.OrderBy(point => function.Value(point)).ToArray();

            for (int i = 0; i < 2; i++) {
                sumX += points[i].X;
                sumY += points[i].Y;
            }

            xG = new(sumX / 2.0, sumY / 2.0);

            if (Criteria(points, xG, function)) {
                _min = points[0];
                break;
            }

            xR = Reflection(points, xG);

            double value = function.Value(xR);

            if (function.Value(points[0]) <= value
                && value < function.Value(points[1])) {
                points[2] = xR;
            } else if (value < function.Value(points[0])) {
                xE = Expansion(xG, xR);

                if (function.Value(xE) < value) {
                    points[2] = xE;
                } else {
                    points[2] = xR;
                }
            } else if (value < function.Value(points[2])) {
                xC = OutsideContraction(xG, xR);

                if (function.Value(xC) < value) {
                    points[2] = xC;
                } else {
                    Shrink(points);
                }
            } else {
                xC = InsideContraction(points, xG);

                if (function.Value(xC) < function.Value(points[2])) {
                    points[2] = xC;
                } else {
                    Shrink(points);
                }
            }
        }

        _min = points[0];
    }

    private bool Criteria(Vector2D[] points, Vector2D xG, IFunction function) {
        double sum = 0.0;
        double valueXG = function.Value(xG);
```

```
128
129         for (int i = 0; i < 3; i++) {
130             double valuePoint = function.Value(points[i]);
131
132             sum += (valuePoint - valueXG) * (valuePoint - valueXG);
133         }
134
135         return Math.Sqrt(sum / (2 + 1)) < Eps;
136     }
137
138     private static Vector2D Reflection(Vector2D[] points, Vector2D xG)
139         => new(xG.X - points[2].X + xG.X, xG.Y - points[2].Y + xG.Y);
140
141     private static Vector2D Expansion(Vector2D xG, Vector2D xR)
142         => new(xG.X + (2 * (xR.X - xG.X)), xG.Y + (2 * (xR.Y - xG.Y)));
143
144     private static Vector2D OutsideContraction(Vector2D xG, Vector2D xR)
145         => new(xG.X + (0.5 * (xR.X - xG.X)), xG.Y + (0.5 * (xR.Y - xG.Y)));
146
147     private static Vector2D InsideContraction(Vector2D[] points, Vector2D xG)
148          => new(xG.X + (0.5 * (points[2].X - xG.X)), xG.Y + (0.5 * (points[2].Y -
          ↪  xG.Y)));
149
150     private static void Shrink(Vector2D[] points) {
151         for (int i = 1; i <= 2; i++)
152             points[i] = points[0] + (0.5 * (points[i] - points[0]));
153     }
154 }
155
156 // public class DirectedSimpleRandomSearch : ISearchMethod2D { // алгоритм парной
    ↪  пробы
157 //     private Vector2D? _min;
158 //     public Vector2D? Min => _min;
159 //     public double Eps { get; init; }
160 //     public double Probability { get; init; }
161
162 //     public DirectedSimpleRandomSearch(double eps, double probability)
163 //         => (Eps, Probability) = (eps, probability);
164
165 //     public void Compute(Rectangle rectangle, IFunction function) {
166 //         double functionMaxValue = function.Value(rectangle.LeftBottom);
167
168 //         int tests = 1;
169 //         const double step = 1.0;
170 //         double neighbourhood = Eps * Eps;
171 //         double probabilityEps = neighbourhood / rectangle.Square;
172
173 //         while (tests < Math.Log(1.0 - Probability) / Math.Log(1.0 -
    ↪  probabilityEps)) {
174 //             tests++;
175 //         }
176
177 //         for (int i = 0; i < tests; i++) {
178 //             double newX = new Random().NextDouble(rectangle.LeftBottom.X,
    ↪  rectangle.RightBottom.X);
179 //             double newY = new Random().NextDouble(rectangle.LeftBottom.Y,
    ↪  rectangle.LeftTop.Y);
180
181 //             double temp;
182
```

```
//             if ((temp = function.Value(new(newX, newY))) > functionMaxValue) {
//                 _min = new(newX, newY);
//                 functionMaxValue = temp;

//                 Vector2D unitVector = new(1.0 / (rectangle.RightBottom.X - rectangle.LeftBottom.X),
//                                           1.0 / (rectangle.LeftTop.Y - rectangle.LeftBottom.Y));

//                 unitVector = unitVector.Normalize();

//                 if (function.Value(_min.Value + unitVector) > function.Value(_min.Value - unitVector)) {
//                     _min = new(_min.Value.X + (step * unitVector.X), _min.Value.Y + (step * unitVector.Y));
//                 } else {
//                     _min = new(_min.Value.X - (step * unitVector.X), _min.Value.Y - (step * unitVector.Y));
//                 }
//             }
//         }
//     }
// }
```

GlobalSearchAlgorithms.cs

```
namespace oMethods_4;

public class AlgorithmA : ISearchMethod2D {
    private Vector2D? _min;
    public Vector2D? Min => _min;
    public double Eps { get; init; }
    public int Trying { get; init; }

    public AlgorithmA(double eps, int trying)
        => (Eps, Trying) = (eps, trying);

    public void Compute(Rectangle rectangle, IFunction function) {
        IMinMethod2D method = new SimplexMethod(1000, 1.0, Eps);
        _min = new(0.0, 0.0);

        for (int i = 0; i < Trying; i++) {
            double newX = new Random().NextDouble(rectangle.LeftBottom.X, rectangle.RightBottom.X);
            double newY = new Random().NextDouble(rectangle.LeftBottom.Y, rectangle.LeftTop.Y);

            method.Compute(function, new(newX, newY));

            if (function.Value(method.Min!.Value) < function.Value(_min.Value)) {
                _min = method.Min;
            }
        }

        Console.WriteLine($"Extremum: {_min}");
        Console.WriteLine($"f(extremum) = {-function.Value(_min!.Value)}");
    }
}

public class AlgorithmB : ISearchMethod2D {
```

```
    private Vector2D? _min;
    public Vector2D? Min => _min;
    public double Eps { get; init; }
    public int Trying { get; init; }

    public AlgorithmB(double eps, int trying)
        => (Eps, Trying) = (eps, trying);

    public void Compute(Rectangle rectangle, IFunction function) {
        IMinMethod2D simplexMethod = new SimplexMethod(1000, 1.0, Eps);
        Vector2D point;

        double temp;

        _min = new(0.0, 0.0);

        double functionValue = function.Value(_min!.Value);

        for (int i = 0; i < Trying; i++) {
            int index = 0;
            simplexMethod.Compute(function, _min.Value);

            if ((temp = function.Value(simplexMethod.Min!.Value)) < functionValue) {
                _min = simplexMethod.Min;
                functionValue = temp;
            }

            do {
                index++;
                double newX = new Random().NextDouble(rectangle.LeftBottom.X,
                ↪  rectangle.RightBottom.X);
                double newY = new Random().NextDouble(rectangle.LeftBottom.Y,
                ↪  rectangle.LeftTop.Y);
                point = new(newX, newY);
            } while ((temp = function.Value(point)) >= functionValue && index <
            ↪  Trying);

            if (temp < functionValue) {
                _min = point;
                functionValue = temp;
            }
        }

        Console.WriteLine($"Extremum: {_min}");
        Console.WriteLine($"f(extremum) = {-function.Value(_min!.Value)}");
    }
}

public class AlgorithmC : ISearchMethod2D {
    private Vector2D? _min;
    public Vector2D? Min => _min;
    public double Eps { get; init; }
    public int Trying { get; init; }

    public AlgorithmC(double eps, int trying)
        => (Eps, Trying) = (eps, trying);

    public void Compute(Rectangle rectangle, IFunction function) {
        IMinMethod2D simplexMethod = new SimplexMethod(1000, 1.0, Eps);

```

```
90          double newX = new Random().NextDouble(rectangle.LeftBottom.X,
            ↪  rectangle.RightBottom.X);
91          double newY = new Random().NextDouble(rectangle.LeftBottom.Y,
            ↪  rectangle.LeftTop.Y);
92
93          double temp;
94          double functionValueX2;
95
96          Vector2D x1, x2;
97          Vector2D initPoint = new(newX, newY);
98          _min = new(newX, newY);
99
100         double functionValueX1 = function.Value(_min.Value);
101
102         for (int i = 0; i < Trying; i++) {
103             double step = 1.0;
104
105             simplexMethod.Compute(function, initPoint);
106             x1 = simplexMethod.Min!.Value;
107
108             if ((temp = function.Value(simplexMethod.Min!.Value)) < functionValueX1) {
109                 _min = simplexMethod.Min;
110                 functionValueX1 = temp;
111             }
112
113             newX = new Random().NextDouble(rectangle.LeftBottom.X,
                ↪  rectangle.RightBottom.X);
114             newY = new Random().NextDouble(rectangle.LeftBottom.Y,
                ↪  rectangle.LeftTop.Y);
115
116             Vector2D randomVector = new(newX, newY);
117             Vector2D direction = simplexMethod.Min!.Value + (step *
                ↪  (simplexMethod.Min.Value - randomVector));
118             double functionValueDirection = function.Value(direction);
119
120             while (functionValueDirection >= function.Value(simplexMethod.Min.Value)
                ↪  && rectangle.Inside(direction)) {
121                 direction = simplexMethod.Min!.Value + (step *
                    ↪  (simplexMethod.Min.Value - randomVector));
122                 functionValueDirection = function.Value(direction);
123                 step++;
124             }
125
126             simplexMethod.Compute(function, direction);
127             functionValueX2 = function.Value(simplexMethod.Min.Value);
128
129             x2 = simplexMethod.Min.Value;
130
131             if (functionValueX2 < functionValueX1) {
132                 _min = simplexMethod.Min;
133             }
134
135             if (functionValueX2 < functionValueX1) {
136                 initPoint = x2;
137             } else {
138                 initPoint = x1;
139             }
140         }
141
142         Console.WriteLine($"Extremum: {_min}");
```

```
        Console.WriteLine($"f(extremum) = {-function.Value(_min!.Value)}");
    }
}
```

RandomExtensions.cs

```
namespace oMethods_4;

public static class RandomExtensions {
    public static double NextDouble(this Random RandGenerator, double MinValue,
    ↪  double MaxValue)
        => (RandGenerator.NextDouble() * (MaxValue - MinValue)) + MinValue;
}
```

Rectangle.cs

```
namespace oMethods_4;

public readonly record struct Rectangle {
    [JsonProperty("Left bottom")]
    public Vector2D LeftBottom { get; init; }

    [JsonProperty("Right top")]
    public Vector2D RightTop { get; init; }

    [JsonIgnore]
    public Vector2D RightBottom { get; init; }

    [JsonIgnore]
    public Vector2D LeftTop { get; init; }

    [JsonIgnore]
    public double Square { get; init; }

    [JsonConstructor]
    public Rectangle(Vector2D leftBottom, Vector2D rightTop) {
        LeftBottom = leftBottom;
        RightTop = rightTop;
        RightBottom = new(rightTop.X, leftBottom.Y);
        LeftTop = new(leftBottom.X, rightTop.Y);
        Square = LeftTop.Distance(rightTop) * leftBottom.Distance(RightBottom);
    }

    public bool Inside(Vector2D point)
        => point.X >= LeftBottom.X && point.X <= RightBottom.X && point.Y >=
        ↪  LeftBottom.Y && point.Y <= LeftTop.Y;

    public static Rectangle? ReadJson(string jsonPath) {
        try {
            if (!File.Exists(jsonPath))
                throw new Exception("File does not exist");

            var sr = new StreamReader(jsonPath);
            using (sr) {
                return JsonConvert.DeserializeObject<Rectangle>(sr.ReadToEnd());
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
            return null;
```

```
        }
    }
}
```

Vector2D.cs

```
namespace oMethods_4;

public readonly record struct Vector2D(double X, double Y) {
    [JsonIgnore]
    public double Norm => Math.Sqrt((X * X) + (Y * Y));

    public static Vector2D operator +(Vector2D a, Vector2D b)
    => new(a.X + b.X, a.Y + b.Y);

    public static Vector2D operator -(Vector2D a, Vector2D b)
        => new(a.X - b.X, a.Y - b.Y);

    public static Vector2D operator /(Vector2D a, double v)
        => new(a.X / v, a.Y / v);

    public static Vector2D operator *(Vector2D a, double v)
        => new(a.X * v, a.Y * v);

    public static Vector2D operator *(double v, Vector2D a)
        => new(v * a.X, v * a.Y);

    public static double Distance(Vector2D a, Vector2D b)
        => (a - b).Norm;

    public double Distance(Vector2D vector)
        => Distance(this, vector);

    public Vector2D Normalize() => this / Norm;

    public static Vector2D Zero
        => new(0.0, 0.0);

    public override string ToString()
        => $"{X} {Y}";
}
```

max.py

```
from pylab import figure, cm

import matplotlib.pyplot as plt
import numpy as np


def f(x1,x2):
    return ((2.0 / (1 + ((x1 - 5) * (x1 - 5)) + ((x2 - 4) * (x2 - 4)))) +
            (1.0 / (1 + ((x1 - 2) * (x1 - 2)) + (x2 * x2))) +
            (7.0 / (1 + ((x1 + 9) * (x1 + 9)) + ((x2 + 6) * (x2 + 6)))) +
            (2.0 / (1 + (x1 * x1) + ((x2 + 3) * (x2 + 3)))) +
            (8.0 / (1 + ((x1 + 3) * (x1 + 3)) + ((x2 - 7) * (x2 - 7)))) +
            (4.0 / (1 + ((x1 + 3) * (x1 + 3)) + ((x2 - 3) * (x2 - 3)))))


```

```
x1_min = -10.0
x1_max = 10.0
x2_min = -10.0
x2_max = 10.0

x1, x2 = np.meshgrid(np.arange(x1_min,x1_max, 0.1), np.arange(x2_min,x2_max, 0.1))

y = f(x1,x2)

plt.imshow(y,extent=[x1_min,x1_max,x2_min,x2_max], cmap=cm.jet, origin='lower')

plt.colorbar()


plt.show()
```