Министерство науки и высшего образования
Российской Федерации

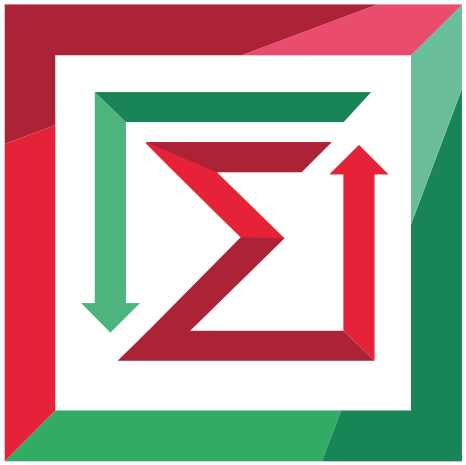
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ
НЭТИ

Кафедра теоретической и прикладной информатики

Лабораторная работа № 4

по дисциплине «Методы оптимизации»



Факультет:	ПМИ
Группа:	ПМ-71
Вариант:	4
Студент:	Баштовой Павел
	Бурдуков Вадим
	Востриков Вячеслав
Преподаватели:	Чимитова Е. В.

Новосибирск

2020

1. *Цель работы*

Ознакомиться со статистическими методами поиска при решении задач нелинейного программирования. Изучить методы случайного поиска при определении локальных и глобальных экстремумов функций.

2. *Задание*

1. Реализовать программу для решения задачи поиска глобального экстремума с использованием **метода простого случайного поиска, 1-3 алгоритмов глобального поиска**.
2. Исследовать метод простого случайного поиска при различных ε и P. Результат представить в таблице:

ε	P	N	(x^*, y^*)	$f(x^*, y^*)$

3. Провести сравнительный анализ 1-3 алгоритмов глобального поиска по точности получаемого решения и числу вычислений целевой функции. Исследование провести при различных значениях числа попыток m.

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

Целевая функция

№ варианта	C_1	C_2	C_3	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_4	b_5	b_6
4	4	9	1	7	5	6	7	-9	6	-8	-10	-2	9	-1	5	-2	-8	-4

3. Исследования:

1) Метод простого случайного поиска:

ε	P	N	(x^*, y^*)	$f(x^*, y^*)$
1	0,9	1001	(8.66321, 1.11816)	11.3666
0,1	0,9	92201	(8.90627, 1.1114)	11.7152
0,01	0,9	9210401	(8.89034, 1.10916)	11.7174
0,01	0,1	421501	(8.91029, 1.08454)	11.7101
0,01	0,5	2772601	(8.88268, 1.10703)	11.7168
0,01	0,9	9210401	(8.89034, 1.10916)	11.7174

Из таблицы видно, что с увеличением эпсилон точность увеличивается, но т.к область целевой функции очень большая для случайного поиска, то дальнейшее увеличение точности приводит к сильно большим вычислениям. Высокая вероятность нахождения минимума требует больших вычислений, но и точность решения возрастает.

2) 1-3 Алгоритмы глобального поиска:

Алгоритм номер	m	Количество вычислений	(x^*, y^*)	$f(x^*, y^*)$
1	10	5889	(8.89033951, 1.1091638)	11.7174
2	10	6446	(8.89033978, 1.1091634)	11.7174
3	10	10795	(8.89034012, 1.1091636)	11.7174
1	100	73967	(8.89033928, 1.1091627)	11.7174
2	100	74798	(8.89033973, 1.101614)	11.7174
3	100	142483	(8.89033981, 1.1091619)	11.7174
1	1000	642909	(8.89033908, 1.1091628)	11.7174
2	1000	647254	(8.89033970, 1.1091334)	11.7174
3	1000	1342383	(8.89033989, 1.1091636)	11.7174

Из таблицы видно, что от роста m напрямую зависит количество вычислений, а также точность увеличивается. Из 1-3 методов 3-й требует самое большое количество вычислений.

4. *Вывод:*

Методы случайного поиска требуют много вычислений, чтобы достичь необходимой точности.
К преимуществам статистических методов можно отнести то, что при решении задач с ограничениями ищется минимум самой исходной функции, а не преобразованной, так как случайные числа генерируются только в исходной области.

Ненаправленные методы, например, метод простейшего случайного поиска, можно использовать для поиска глобального экстремума у функций, имеющих несколько локальных экстремумов.

5. *Текст программы:*

```
#include <iostream>
#include <math.h>
#include <random>
#include <omp.h>
#include <fstream>
using namespace std;
double eps = 1e-7;
int N = 1;
double fc1 = 0, fc2 = 0, fc3 = 0;
double m = 100;
double r = 1;
double A[2] = { -10, -10 };
double B[2] = { 10, 10 };
double Norm(double* x, double* y) { return sqrt(x[0] * y[0] + x[1] * y[1]); }
double Norm(pair<double, double> x, pair<double, double> y) { return sqrt(x.first * y.first
+ x.second * y.second); }
pair<double, double> Rand()
{
	pair<double, double> x;
	random_device rd;
	mt19937 gen(rd());
	uniform_real_distribution<> dis(-10, 10);
	x.first = dis(gen);
	x.second = dis(gen);
	return x;
}
double g(pair<double, double> x) { return -x.first - 10; }
double h(pair<double, double> x) { return x.first - 10; }
double p(pair<double, double> x) { return -x.second - 10; }
double q(pair<double, double> x) { return x.second - 10; }
double G(pair<double, double> x) { return pow(0.5 * (g(x) + abs(g(x))), 2); }
double H(pair<double, double> x) { return pow(0.5 * (h(x) + abs(h(x))), 2); }
double P(pair<double, double> x) { return pow(0.5 * (p(x) + abs(p(x))), 2); }
double Q(pair<double, double> x) { return pow(0.5 * (q(x) + abs(q(x))), 2); }
double r_correct(double r) { return r * 2; }
double f(pair<double, double> z)
{
	double f = 0;
	double C[6] = { 2,3,8,3,2,8 };
	double a[6] = { 3,-5,0,3,-4,6 };
	double b[6] = { -4,-6,-1,7,0,5 };
```

```
	double x = z.first, y = z.second;
	for (int i = 0; i < 6; i++)
	{
		f += C[i] / (1 + (x - a[i]) * (x - a[i]) + (y - b[i]) * (y - b[i]));
	}
	fc1++;
	fc2++;
	fc3++;
	return -f;
}
double ff(pair<double, double> z)
{
	double f = 0;
	double C[6] = { 4,9,1,7,5,6 };
	double a[6] = { 7,-9,6,-8,-10,-2 };
	double b[6] = { 9,-1,5,-2,-8,-4 };
	double x = z.first, y = z.second;
	for (int i = 0; i < 6; i++)
	{
		f += C[i] / (1 + (x - a[i]) * (x - a[i]) + (y - b[i]) * (y - b[i]));
	}
	fc1++;
	fc2++;
	fc3++;
	return -f;
}

pair<double, double> Easy_random_search()
{
	pair<double, double> x, x1;
	int k = 0;
	x = Rand();
	for (int i = 0; i < N; i++)
	{
		x1 = Rand();
		if (ff(x1) < ff(x))
			x = x1;
	}
	//cout << -x.first << '\t' << -x.second << '\t' << -ff(x) << endl;
	return x;
}
double Golden(pair<double, double> s, double* p, int& fc)
{
	const double r = (1 + sqrt(5)) / 2;
	double a = -10;
	double b = 10;
	double x0, x1, xf1, xf2;
	x0 = b - (b - a) / r;
	x1 = a + (b - a) / r;
	pair<double, double> t1, t2;
	t1.first = s.first + x0 * p[0];
	t1.second = s.second + x0 * p[1];
	t2.first = s.first + x1 * p[0];
	t2.second = s.second + x1 * p[1];
	xf1 = f(t1);
	xf2 = f(t2);
	fc += 2;
	do {

		if (xf1 >= xf2)
		{
			a = x0;
			x0 = x1;
			xf1 = xf2;
			x1 = a + (b - a) / r;
```

```
                        t2.first = s.first + x1 * p[0];
                        t2.second = s.second + x1 * p[1];
                        xf2 = f(t2);
                        fc++;
                }
                else
                {
                        b = x1;
                        x1 = x0;
                        xf2 = xf1;
                        x0 = b - (b - a) / r;
                        t1.first = s.first + x0 * p[0];
                        t1.second = s.second + x0 * p[1];
                        xf1 = f(t1);
                        fc++;
                }
        } while (abs(b - a) > eps);
        return (a + b) / 2;
}
pair<double, double> Roz(pair<double, double> x)
{
        int fc = 0, i = 0;
        pair<double, double> x_old;
        double a1[2], a2[2];
        double s1[2] = { 1 , 0 }, s2[2] = { 0, 1 };
        double lambda1;
        double lambda2;
        do {

                x_old.first = x.first;
                x_old.second = x.second;
                lambda1 = Golden(x, s1, fc);
                x.first = x.first + s1[0] * lambda1;
                x.second = x.second + s1[1] * lambda1;
                lambda2 = Golden(x, s2, fc);
                x.first = x.first + s2[0] * lambda2;
                x.second = x.second + s2[1] * lambda2;

                a1[0] = s1[0] * lambda1 + s2[0] * lambda2;
                a1[1] = s1[1] * lambda1 + s2[1] * lambda2;
                a2[0] = s2[0] * lambda2;
                a2[1] = s2[1] * lambda2;

                s1[0] = a1[0] / Norm(a1, a1);
                s1[1] = a1[1] / Norm(a1, a1);

                s2[0] = (a2[0] * Norm(a1, a1) * Norm(a1, a1) - a1[0] * Norm(a2, a2) * Norm(a2,
a2)) /
                        (Norm(a1, a1) * Norm(a2, a2) * sqrt(Norm(a1, a1) * Norm(a1, a1) -
Norm(a2, a2) * Norm(a2, a2)));
                s2[1] = (a2[1] * Norm(a1, a1) * Norm(a1, a1) - a1[1] * Norm(a2, a2) * Norm(a2,
a2)) /
                        (Norm(a1, a1) * Norm(a2, a2) * sqrt(Norm(a1, a1) * Norm(a1, a1) -
Norm(a2, a2) * Norm(a2, a2)));
                i++;
                r = r_correct(r);
        } while (abs(f(x_old) - f(x)) > eps);
        return x;
}
pair<double, double> Random_vec()
{
        pair<double, double> x;
        random_device rd;
        mt19937 gen(rd());
        uniform_real_distribution<> dis(-0.1, 0.1);
```

```
	x.first = dis(gen);
	x.second = sqrt(1 - x.first * x.first);
	return x;
}
pair<double, double> Global_search_1()
{
	pair<double, double> x, x1;
	int flag = 0, q = 0;
	x = { 5, 5 };
	while (flag < m)
	{
		x1 = Rand();
		x1 = Roz(x1);
		if (ff(x1) < ff(x))
		{
			x = x1;
			q = 1;
		}
		else
		{
			flag++;
			q = 0;
		}
		if (q == 1)
			flag = 0;
	}
	//cout << -x.first << '\t' << -x.second << '\t' << -ff(x) << endl;
	return x;
}

pair<double, double> Global_search_2()
{
	pair<double, double> x, x1;
	int flag = 0, q = 0, i = 0;;
	x = Rand();
	x = Roz(x);
	do {
		flag = 0;
		do
		{
			x1 = Easy_random_search();
			flag++;
		} while (ff(x1) > ff(x) && flag < m);
		if (flag < m) {
			x1 = Roz(x1);
			x = Rand();
			x = Easy_random_search();
		}
		else
			q = 1;
		i++;
	} while (q != 1);
	return x;
}



pair<double, double>Best_prob(pair<double, double> x)
{
	pair<double, double> e, x1, x2;
	e = Random_vec();

	return x;
}
double Sign(double x)
```

```
{
	if (x > 0) return 1;
	if (x < 0) return -1;
	return 0;
}
pair<double, double> Global_search_3()
{
	pair<double, double> x1 = { 0,0 }, x2, e;
	int i = 0, f = m;
	do {
		x1 = Roz(x1);
		x2 = x1;
		do
		{
			e = Random_vec();
			x2.first += e.first;
			x2.second += e.second;
			i++;
		} while (ff(x2) <= ff(x1) && fabs(x2.first) < 10 && fabs(x2.second) < 10);
		if (fabs(x2.first) > 10 && fabs(x2.second) > 10)
		{
			x2.first -= e.first;
			x2.second -= e.second;
		}
		x2 = Roz(x2);
		if (ff(x2) < ff(x1))
			x1 = x2;
		else
			f--;
	} while (f > 0);
	return x2;
}
int main()
{
	pair<double, double> x;
	ofstream fout;
	fout.open("out.txt");
		eps = 1;
		for(int j = 0; j < 3; j++)
		{
			eps = 1 / pow(10, j);
			double P = 0.9;
			double V = (B[0] - A[0]) * (B[1] - A[1]);
			double Veps = eps * eps;
			double Peps = Veps / V;
			double a = log(1 - P) / log(1 - Peps);
			while (N < log(1 - P) / log(1 - Peps))
				N += 100;
			x = Easy_random_search();
			fout << eps << '\t' << P << '\t' << N << '\t' << -x.first << '\t' << -
x.second << '\t' << -ff(x) << endl;
			cout << "iter №" << j << endl;
		}
	//eps = 1e-2;
	//for (int i = 0; i < 3; i++)
	//{
	//	double P = 0.1 + 0.4 * i;
	//	//double P = 0.9;
	//	double V = (B[0] - A[0]) * (B[1] - A[1]);
	//	double Veps = eps * eps;
	//	double Peps = Veps / V;
	//	double a = log(1 - P) / log(1 - Peps);
	//	while (N < log(1 - P) / log(1 - Peps))
	//		N += 100;
	//	x = Easy_random_search();
```

```
	//	fout << eps << '\t' << P << '\t' << N << '\t' << -x.first << '\t' << -x.second
<< '\t' << -ff(x) << endl;
	//	cout << "iter" <<i<< endl;
	//}
	cout << "Poschitano!" << endl;
	//m = 1;
	//for (int i = 0; i < 3; i++)
	//{
	//	m *= 10;
	//	fc1 = 0;
	//	fc2 = 0;
	//	fc3 = 0;
	//	fout.precision(8);
	//	x = Global_search_1();
	//	fout << "1" << '\t';
	//	fout << m << '\t' << fc1 << '\t' << -x.first << '\t' << -x.second << '\t' << -
ff(x) << endl;
	//	x = Global_search_2();
	//	fout << "2" << '\t';
	//	fout << m << '\t' << fc2 << '\t' << -x.first << '\t' << -x.second << '\t' << -
ff(x) << endl;
	//	x = Global_search_3();
	//	fout << "3" << '\t';
	//	fout << m << '\t' << fc3 << '\t' << -x.first << '\t' << -x.second << '\t' << -
ff(x) << endl;
	//
	//}
	//m = 100;
	//
	//x = Global_search_1();
	//	fout << "1" << '\t';
	//	cout << m << '\t' << -x.first << '\t' << -x.second << '\t' << -ff(x) << endl;
	fout.close();

	return 0;
}
```