Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 2
по дисциплине «Теория систем и системный анализ»

Тема: «Исследование метода случайного поиска экстремума функции одного переменного»

Вариант 10

Выполнила:
Минькова А.А., студент
группы ИУ8-31

Проверила: Коннова Н.С.,
доцент каф. ИУ8

г. Москва,
2020 г.

1. Цель работы

Изучение метода случайного поиска экстремума на примере унимодальной и мультимодальной функций одного переменного.

2. Условие задачи

1. На интервале [-2; 4] задана унимодальная функция одного переменного f(x) = (1-x)^2+exp(x). Используя метод случайного поиска осуществить поиск минимума $f(x)$ с заданной вероятностью попадания в окрестность экстремума P при допустимой длине интервала неопределенности ε. Определить необходимое число испытаний N. Численный эксперимент выполнить для значений P = 0,90, 0,91,..., 0,99 и значений ε = – ( ) b a q , где q = 0,005, 0,010,..., 0,100. Последовательность действий:
   - определить вероятность P_1 непопадания в ε-окрестность экстремума за одно испытание;
   - записать выражение для вероятности P_N непопадания в ε-окрестность экстремума за N испытаний;
   - из выражения для P_N определить необходимое число испытаний N в зависимости от заданных $P_N = P$ и ε .
2. При аналогичных исходных условиях осуществить поиск минимума *f(x)*, модулированной сигналом *sin(5x)*, т.е. мультимодальной функции *f(x)·sin(5x)*.

3. Ход работы

Построим графики заданных функций и определим их минимумы:

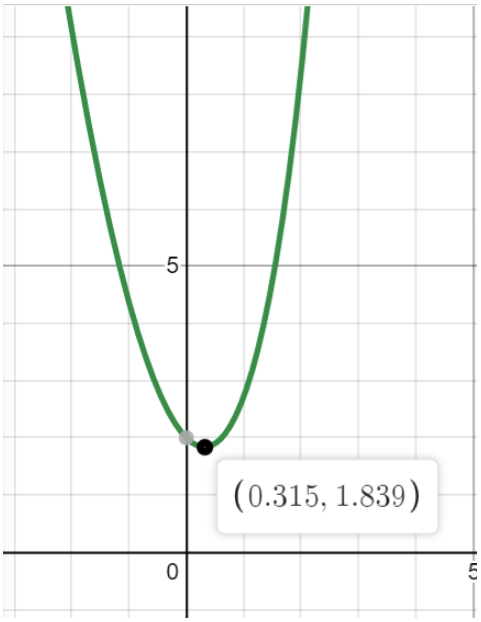

Рисунок 1 – График функции *f(x)* на интервале [-2; 4]

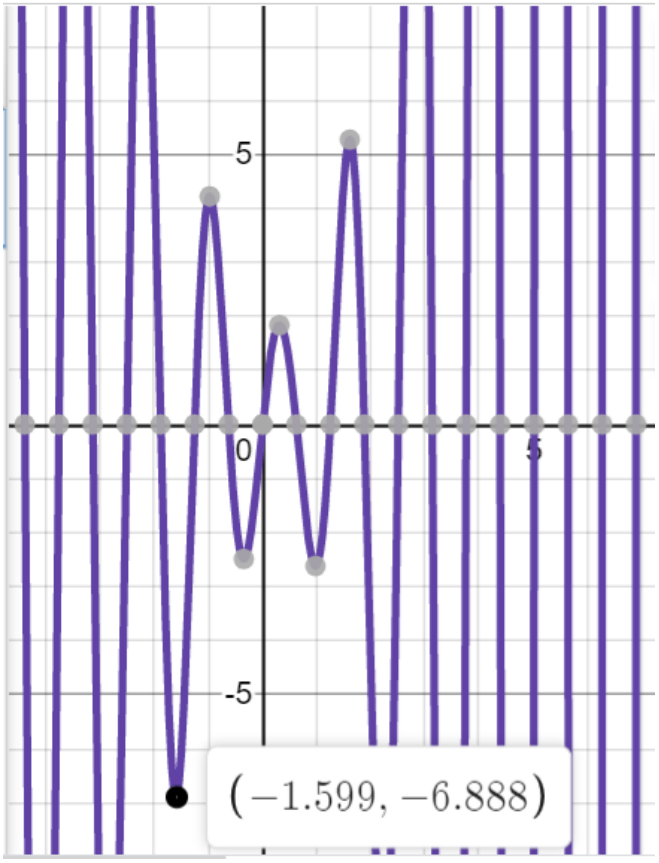

Рисунок 2 - График функции *f(x)*sin(5x)* на интервале [-2; 4]

Результат работы программы

Variant 10:

Table with number of points for each p and q:

q/P	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
0.005	460	481	504	531	562	598	643	700	781	919
0.01	230	240	252	265	280	299	321	349	390	459
0.015	153	160	168	176	187	199	213	233	259	305
0.02	114	120	126	132	140	149	160	174	194	228
0.025	91	96	100	106	112	119	128	139	155	182
0.03	76	80	83	88	93	99	106	116	129	152
0.035	65	68	71	75	79	85	91	99	110	130
0.04	57	59	62	66	69	74	79	86	96	113
0.045	51	53	55	58	62	66	70	77	85	101
0.05	45	47	50	52	55	59	63	69	77	90
0.055	41	43	45	48	50	53	57	62	70	82
0.06	38	39	41	43	46	49	53	57	64	75
0.065	35	36	38	40	42	45	48	53	59	69
0.07	32	34	35	37	39	42	45	49	54	64
0.075	30	31	33	35	37	39	42	45	51	60
0.08	28	29	31	32	34	36	39	43	47	56
0.085	26	28	29	30	32	34	37	40	45	52
0.09	25	26	27	29	30	32	35	38	42	49
0.095	24	25	26	27	29	31	33	36	40	47
0.1	22	23	24	26	27	29	31	34	38	44

Table for function 1:

q/P	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
0.005	1.83954	1.83951	1.83955	1.83949	1.83949	1.83951	1.83949	1.8395	1.83963	1.83948
0.01	1.83954	1.83951	1.84427	1.84221	1.84238	1.84012	1.83953	1.83954	1.83949	1.83997
0.015	1.8396	1.83972	1.83951	1.8412	1.83953	1.83961	1.84374	1.84	1.83993	1.83951
0.02	1.83951	1.8418	1.84062	1.84034	1.83962	1.83956	1.84122	1.83956	1.83952	1.83949
0.025	1.83991	1.83979	1.8395	1.84839	1.84554	1.84029	1.84244	1.83962	1.83951	1.83961
0.03	1.84202	1.83998	1.84926	1.85552	1.83982	1.83959	1.83969	1.84076	1.83964	1.83963
0.035	1.84721	1.84168	1.84033	1.84038	1.83957	1.84054	1.83951	1.84051	1.83952	1.84151
0.04	1.83953	1.84255	1.84067	1.84495	1.84129	1.84426	1.84037	1.83951	1.84209	1.84733
0.045	1.85031	1.84034	1.84017	1.84484	1.84122	1.84152	1.84546	1.8404	1.84162	1.84149
0.05	1.84068	1.83948	1.84022	1.84688	1.83956	1.84551	1.85372	1.83959	1.86706	1.83949
0.055	1.84453	1.83951	1.96468	1.83948	1.87337	1.83954	1.85216	1.84052	1.85236	1.83967
0.06	1.85644	1.83959	2.08106	1.84882	1.8395	1.84204	1.83974	1.84113	1.85599	1.83985
0.065	1.84369	1.8517	1.84003	1.83992	1.84216	1.86271	1.83985	1.84469	1.84044	1.84011
0.07	1.83949	1.8473	1.9668	1.85148	1.84022	1.86646	1.84403	1.86653	1.8407	1.88796
0.075	1.84061	1.84037	1.83965	1.83959	1.86304	1.83991	1.85262	1.86864	1.85604	1.86001
0.08	1.85119	1.88544	1.84201	1.85263	1.94102	1.84122	1.84082	1.84405	1.84135	1.84057
0.085	1.8416	1.8395	1.8491	1.95648	1.85619	1.88049	1.83962	1.84228	1.84314	1.84031
0.09	1.8492	1.8779	1.92842	1.84353	1.92947	1.8454	1.84125	1.8395	1.83964	1.84006
0.095	1.85067	1.88134	1.89414	1.85747	1.83954	1.88836	1.8395	1.8473	1.83995	1.84662
0.1	1.84572	1.84605	1.86125	1.89634	1.98209	1.91803	1.92815	1.83995	1.83983	1.84851

Table for function 2:

q/P	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
0.005	-38.4238	-38.4224	-38.3971	-38.4144	-38.421	-38.4238	-38.4226	-38.4236	-38.4238	-38.4205
0.01	-38.4228	-36.7164	-38.423	-38.4154	-38.2662	-38.4048	-38.3812	-38.3753	-38.3678	-38.3943
0.015	-38.3766	-38.4225	-38.2367	-38.3958	-38.3703	-37.9828	-38.3953	-38.4136	-38.3868	-38.4223
0.02	-33.9452	-38.256	-38.2049	-38.2884	-37.6962	-37.2557	-37.5797	-38.3632	-37.8274	-38.4219
0.025	-37.0183	-38.3349	-38.0939	-38.4042	-34.5837	-33.9343	-37.4387	-38.1875	-36.7521	-38.3482
0.03	-37.6077	-35.5787	-38.4017	-35.9029	-37.9687	-36.7646	-37.8731	-38.3635	-38.2934	-38.3514
0.035	-30.2516	-35.3261	-36.9969	-35.7573	-38.342	-34.2737	-37.1377	-37.2127	-38.1831	-38.3334
0.04	-38.2627	-37.6684	-27.4178	-35.529	-38.3022	-38.2578	-37.7367	-35.8063	-38.4232	-38.2283
0.045	-38.3446	-37.851	-37.9109	-37.1346	-38.423	-38.2488	-38.2355	-37.9631	-35.2301	-37.286
0.05	-32.3709	-35.5979	-33.6804	-23.7302	-38.4009	-33.8546	-38.4081	-32.442	-37.294	-36.7072
0.055	-34.3686	-38.2609	-27.2299	-38.4167	-37.9551	-35.1673	-30.073	-32.97	-38.4103	-38.4212
0.06	-29.7961	-31.8972	-33.408	-37.7864	-38.4016	-36.0542	-34.3867	-38.4012	-38.2834	-38.3645
0.065	-35.3663	-28.5313	-38.3797	-37.812	-38.3454	-33.9218	-31.6861	-36.1267	-38.4216	-32.3907
0.07	-36.0141	-31.1975	-26.4051	-38.2203	-35.1166	-10.2812	-38.0697	-38.4095	-35.9232	-38.3607
0.075	-36.7159	-38.3113	-37.2669	-38.1701	-38.199	-38.4182	-19.1165	-38.3013	-31.2412	-31.7401
0.08	-38.3289	-31.3544	-10.1855	-38.3732	-38.0497	-38.24	-38.4231	-24.8377	-36.6508	-28.4976
0.085	-29.6753	-38.3875	-34.8057	-37.8623	-20.6263	-38.4238	-28.8643	-38.2594	-33.3077	-38.298
0.09	-34.9289	-20.7959	-37.9272	-37.9618	-38.4015	-38.4162	-37.1055	-38.1443	-37.7688	-37.842
0.095	-37.3008	-31.9304	-21.1983	-25.3812	-31.5017	-37.2304	-38.1938	-35.8783	-35.7035	-37.9757
0.1	-8.72471	-37.9113	-31.6758	-27.2703	-16.9403	-29.7313	-37.8406	-37.8774	-37.1939	-38.4239

4. Выводы

Из полученных таблиц и графиков видно, что метод случайного поиска эффективен при поиске экстремума как унимодальной, так и мультимодальной функции одного переменного.

Ссылка на гит-репозиторий: https://github.com/AnnaMinkova/Tsisa_lab_02

Ответ на контрольный вопрос

В чем состоит сущность метода случайного поиска? Какова область применимости данного метода?

Метод случайного поиска представляет собой нахождение экстремума среди значений заданной функции в случайно сгенерированных точках, принадлежащих некоторому отрезку. Различают направленный и ненаправленный случайный поиск. Первый используют для нахождения локального экстремума, второй – для глобального. Этот метод используется при решении задач на областях со сложной геометрией. Обычно вписывают эту область в n-мерный параллелепипед, а далее генерируют в этом n-мерном параллелепипеде случайные точки по равномерному закону, оставляя только те, которые попадают в допустимую область.

Приложение 1. Исходный код программы

```
#include
<iostream>
             #include <cmath>
             #include <iomanip>
             #include <vector>
             #include <string>
             using namespace std;
             double myF(double x)
             {
                 return pow((1-x),2)+exp(x);
             }
             double F(const double x)
             {
                 return myF(x) * sin(5*x);
             }
             const double A=-2;
             const double B=4;
             const vector<double> P_VALUES = {0.9, 0.91, 0.92, 0.93, 0.94,
                                              0.95, 0.96, 0.97, 0.98, 0.99};
             const vector<double> Q_VALUES = {0.005, 0.01, 0.015, 0.02, 0.025,
                                              0.03, 0.035, 0.04, 0.045, 0.05,
                                              0.055, 0.06, 0.065, 0.07, 0.075,
                                              0.08, 0.085, 0.09, 0.095, 0.1};
             void printLine()
             {
                 cout << '+' << std::string(7, '-') << '+' << std::string(10, '-')
                      << '+' << std::string(10, '-') << '+' << std::string(10, '-')
                      << '+' << std::string(10, '-') << '+' << std::string(10, '-')
                      << '+' << std::string(10, '-') << '+' << std::string(10, '-')
                      << '+' << std::string(10, '-') << '+' << std::string(10, '-')
                      << '+' << std::string(10, '-') << '+' << '\n';
             }

             void TableHead(const vector<double>& p)
             {
             printLine();

                 cout << '|' << setw(5) << "q/P" << setw(3) << '|';
                 for(auto item : p) cout <<setw(9) << item << " |";
                 cout << '\n';

             printLine();
             }

             void Table(const vector<double>& p, const vector<double>& q,const
```

```
vector<vector<double>>& table)
{
TableHead(p);

    for (size_t i = 0; i < q.size(); ++i)
    {
        cout << '|' << setw(6) << q[i] << " |";
        for(size_t j = 0; j < p.size(); ++j)
        {
            cout << setw(9) << table[i][j] << " |";
        }
        cout << '\n';
    }
printLine();
}

vector<vector<double>> pointsNumber(const vector<double>& p,const vector<double>& q)
{
    vector<vector<double>> points(q.size());
    for(size_t i = 0; i < q.size(); ++i)
    {
        points[i].resize(p.size());
        for(size_t j = 0; j < p.size(); ++j)
        {
            points[i][j] = ceil(log(1 - p[j]) / log(1 - q[i]));
        }
    }
    return points;
}
double random(const double a, const double b)
{
    return a + rand() * 1./RAND_MAX * (b - a);
}
template<class F>
vector<vector<double>> rndSearch(const vector<vector<double>>& numbers,const double a,
const double b, F function)
{
    vector<vector<double>> table;
    table.resize(numbers.size());
    for(size_t i = 0; i < table.size(); ++i)
    {
        table[i].resize(numbers[i].size());
        for(size_t j = 0; j < table[i].size(); ++j)
        {
            table[i][j] = function(a);
            for(size_t k = 0; k < numbers[i][j]; ++k)
            {
```

```
                double newValue = function(random(a, b));
                if(newValue < table[i][j])
                {
                    table[i][j] = newValue;
                }
            }
        }
    }

    return table;
}

int main()
{
    cout << "Variant 10: "<<endl;
    cout << "Table with number of points for each p and q:\n";
    auto points = pointsNumber(P_VALUES, Q_VALUES);
    Table(P_VALUES, Q_VALUES, points);

    srand(time(nullptr));
    auto valuesForF = rndSearch(points, A, B, myF);
    cout << "Table for function 1:\n";
    Table(P_VALUES, Q_VALUES, valuesForF);
    auto valuesForF_ = rndSearch(points, A, B, F);
    cout << "Table for function 2:\n";
    Table(P_VALUES, Q_VALUES, valuesForF_);

    return 0;
}
```