**Министерство науки и высшего образования Российской Федерации**

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**им. Н.Э. Баумана**

Факультет: ГУИМЦ

Кафедра: Информационная безопасность(ИУ8)

**ТЕОРИЯ СИСТЕМ И СИСТЕМНЫЙ АНАЛИЗ**

**Лабораторная работа №1 на тему:**

**«**Исследование методов прямого поиска экстремума

унимодальной функции одного переменного**»**

Вариант 2

**Преподаватель:**

Коннова Н.С.

**Студент:**

Васильев М.Е.

**Группа:**

ИУ8ц-51

Москва 2019

**Цель работы:**

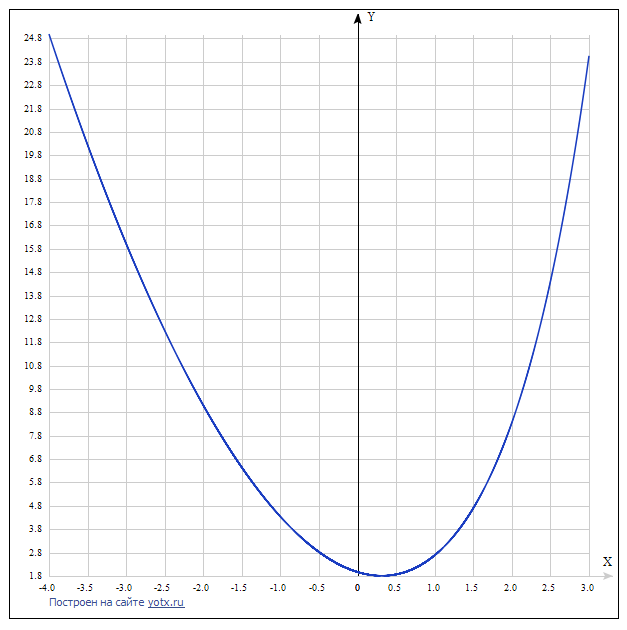
Исследовать функционирование и провести сравнительный анализ различных алгоритмов прямого поиска экстремума (пассивный поиск, метод золотого сечения) на примере унимодальной функции одного переменного.

**Постановка задачи:**

На интервале [a,b] (a=-4, b=3) задана унимодальная функция . Используя метод золотого сечения, найти интервал нахождения минимума f(x) при заданной наибольшей допустимой длине интервала неопределенности ε = 0,1. Провести сравнение с методом оптимального пассивного поиска. Результат, в зависимости от числа точек разбиения N представить в виде таблицы.

**Ход работы:**

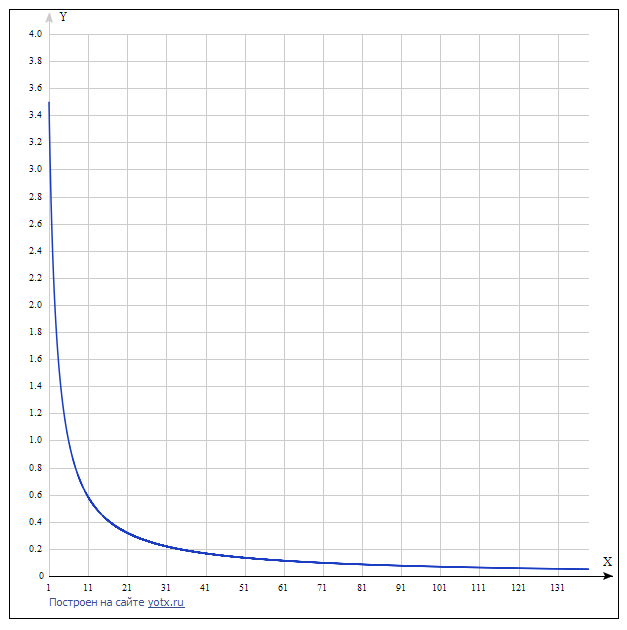
Строим график функции

1. Рассмотрим метод золотого сечения. В данном случае отношение золотого сечения имеет вид . На первом шаге выбираем две точки x11 = и x12 = и находим значения функции в данных точках. После чего выполняем процедуру исключения отрезка: если f(x11)>f(x12), то выбираем отрезок [x11, b] и берем x2 = x12, в противном случае — [a, x12], a x2 = x11. Для данной функции на первом шаге мы получаем отрезок [-1.32624, 3].
2. Так как длина полученного отрезка превышает интервал неопределенности ε = 0,1, то необходимо перейти ко второму шагу, в котором нам нужно найти только одну точку в отрезке по правилу золотого сечения и значение функции в ней, так как у нас в качестве второй точки, расположенной симметрично нашей, уже имеется x2. Далее выполняем те же операции, что и в предыдущем пункте, пока не получим длину отрезка, меньшую, чем ε — в этом случае мы прекращаем поиск.
3. Таблица полученных значений

| k | ak | bk | lk | x1 | x2 | f(x1) | f(x2) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | -1.32624 | 3 | 4.32624 | -1.32624 | 0.326238 | 5.67686 | 1.8397 |
| 2 | -1.32624 | 1.34752 | 2.67376 | 0.326238 | 1.34752 | 1.8397 | 3.96866 |
| 3 | -0.304952 | 1.34752 | 1.65248 | -0.304952 | 0.326238 | 2.44006 | 1.8397 |
| 4 | -0.304952 | 0.716335 | 1.02129 | 0.326238 | 0.716335 | 1.8397 | 2.12738 |
| 5 | 0.0851449 | 0.716335 | 0.63119 | 0.0851449 | 0.326238 | 1.92583 | 1.8397 |
| 6 | 0.0851449 | 0.475242 | 0.390097 | 0.326238 | 0.475242 | 1.8397 | 1.88377 |
| 7 | 0.234149 | 0.475242 | 0.241093 | 0.234149 | 0.326238 | 1.85036 | 1.8397 |
| 8 | 0.234149 | 0.383152 | 0.149004 | 0.326238 | 0.383152 | 1.8397 | 1.8474 |
| 9 | 0.291063 | 0.383152 | 0.0920893 | 0.291063 | 0.326238 | 1.84044 | 1.8397 |

1. Рассмотрим метод оптимального пассивного поиска. В данном случае на отрезке [-4, 3] равномерно располагаем N точек на одинаковом расстоянии друг от друга. После чего вычисляется значение функции в каждой точке и среди них определяется предполагаемый минимум. Максимальное количество точек определяется с помощью значения точности поиска. В данном случае N = 139.
2. На первом шаге N=1. Вычисляем расстояние между точками . l1=3.5. Т.к. у нас одна точка, то её размещают в середине отрезка x=-0.5. Вычисляется значение в этой точке f(x) = 2.85653. Производится сравнение с остальными точками, включая концы отрезка и отбирается минимальное значение с указанием точности, равной расстоянию между точкой оптимума и соседней точкой.
3. На втором шаге количество точек увеличивается N=2. Точки равномерно распределяются по отрезку на расстоянии l2=2.33333. Над ними воспроизводятся те же операции, что и в предыдущем пункте. После нахождения минимума при данном количестве точек и при условии, что расстояние между точками превосходит необходимую точность поиска, операции повторяются. Поиск прекращается при нахождении минимума среди 139 точек, так как при таком количестве l139=0.05, что дает нам необходимую точность поиска.
4. Таблица полученных значений

|  |  |  |
| --- | --- | --- |
| N | Xmin +- lN | f(xmin) |
| 1 | -0.5 +- 3.5 | 2.85653 |
| 2 | 0.666667 +- 2.33333 | 2.05885 |
| 3 | -0.5 +- 1.75 | 2.85653 |
| 4 | 0.2 +- 1.4 | 1.8614 |
| 5 | 0.666667 +- 1.16667 | 2.05885 |
| 6 | 0 +- 1 | 2 |
| 7 | 0.375 +- 0.875 | 1.84562 |
| 8 | 0.666667 +- 0.777778 | 2.05885 |
| 9 | 0.2 +- 0.7 | 1.8614 |
| 10 | 0.454545 +- 0.636364 | 1.87298 |
| 11 | 0.0833333 +- 0.583333 | 1.92718 |
| 12 | 0.307692 +- 0.538462 | 1.83957 |
| 13 | 0.5 +- 0.5 | 1.89872 |
| 14 | 0.2 +- 0.466667 | 1.8614 |
| 15 | 0.375 +- 0.4375 | 1.84562 |
| 16 | 0.117647 +- 0.411765 | 1.90339 |
| 17 | 0.277778 +- 0.388889 | 1.8418 |
| 18 | 0.421053 +- 0.368421 | 1.85874 |
| 19 | 0.2 +- 0.35 | 1.8614 |
| 20 | 0.333333 +- 0.333333 | 1.84006 |
| 21 | 0.454545 +- 0.318182 | 1.87298 |
| 22 | 0.26087 +- 0.304348 | 1.84437 |
| 23 | 0.375 +- 0.291667 | 1.84562 |
| 24 | 0.2 +- 0.28 | 1.8614 |
| 25 | 0.307692 +- 0.269231 | 1.83957 |
| 26 | 0.407407 +- 0.259259 | 1.85408 |
| 27 | 0.25 +- 0.25 | 1.84653 |
| 28 | 0.344828 +- 0.241379 | 1.841 |
| 29 | 0.2 +- 0.233333 | 1.8614 |
| 30 | 0.290323 +- 0.225806 | 1.8405 |
| 31 | 0.375 +- 0.21875 | 1.84562 |
| 32 | 0.242424 +- 0.212121 | 1.84826 |
| 33 | 0.323529 +- 0.205882 | 1.83961 |
| 34 | 0.4 +- 0.2 | 1.85182 |
| 35 | 0.277778 +- 0.194444 | 1.8418 |
| 36 | 0.351351 +- 0.189189 | 1.84173 |
| 37 | 0.236842 +- 0.184211 | 1.84965 |
| 38 | 0.307692 +- 0.179487 | 1.83957 |
| 39 | 0.375 +- 0.175 | 1.84562 |
| 40 | 0.268293 +- 0.170732 | 1.84313 |
| 41 | 0.333333 +- 0.166667 | 1.84006 |
| 42 | 0.395349 +- 0.162791 | 1.85051 |
| 43 | 0.295455 +- 0.159091 | 1.84012 |
| 44 | 0.355556 +- 0.155556 | 1.84228 |
| 45 | 0.26087 +- 0.152174 | 1.84437 |
| 46 | 0.319149 +- 0.148936 | 1.83951 |
| 47 | 0.375 +- 0.145833 | 1.84562 |
| 48 | 0.285714 +- 0.142857 | 1.84092 |
| 49 | 0.34 +- 0.14 | 1.84055 |
| 50 | 0.254902 +- 0.137255 | 1.84551 |
| 51 | 0.307692 +- 0.134615 | 1.83957 |
| 52 | 0.358491 +- 0.132075 | 1.8427 |
| 53 | 0.277778 +- 0.12963 | 1.8418 |
| 54 | 0.327273 +- 0.127273 | 1.83974 |
| 55 | 0.375 +- 0.125 | 1.84562 |
| 56 | 0.298246 +- 0.122807 | 1.83995 |
| 57 | 0.344828 +- 0.12069 | 1.841 |
| 58 | 0.271186 +- 0.118644 | 1.84269 |
| 59 | 0.316667 +- 0.116667 | 1.83949 |
| 60 | 0.360656 +- 0.114754 | 1.84303 |
| 61 | 0.290323 +- 0.112903 | 1.8405 |
| 62 | 0.333333 +- 0.111111 | 1.84006 |
| 63 | 0.265625 +- 0.109375 | 1.84355 |
| 64 | 0.307692 +- 0.107692 | 1.83957 |
| 65 | 0.348485 +- 0.106061 | 1.84139 |
| 66 | 0.283582 +- 0.104478 | 1.84113 |
| 67 | 0.323529 +- 0.102941 | 1.83961 |
| 68 | 0.362319 +- 0.101449 | 1.84329 |
| 69 | 0.3 +- 0.1 | 1.83986 |
| 70 | 0.338028 +- 0.0985915 | 1.84039 |
| 71 | 0.277778 +- 0.0972222 | 1.8418 |
| 72 | 0.315068 +- 0.0958904 | 1.83948 |
| 73 | 0.351351 +- 0.0945946 | 1.84173 |
| 74 | 0.293333 +- 0.0933333 | 1.84027 |
| 75 | 0.328947 +- 0.0921053 | 1.83982 |
| 76 | 0.272727 +- 0.0909091 | 1.84247 |
| 77 | 0.307692 +- 0.0897436 | 1.83957 |
| 78 | 0.341772 +- 0.0886076 | 1.8407 |
| 79 | 0.2875 +- 0.0875 | 1.84075 |
| 80 | 0.320988 +- 0.0864198 | 1.83955 |
| 81 | 0.353659 +- 0.0853659 | 1.84203 |
| 82 | 0.301205 +- 0.0843373 | 1.8398 |
| 83 | 0.333333 +- 0.0833333 | 1.84006 |
| 84 | 0.282353 +- 0.0823529 | 1.84126 |
| 85 | 0.313953 +- 0.0813953 | 1.83949 |
| 86 | 0.344828 +- 0.0804598 | 1.841 |
| 87 | 0.295455 +- 0.0795455 | 1.84012 |
| 88 | 0.325843 +- 0.0786517 | 1.83969 |
| 89 | 0.277778 +- 0.0777778 | 1.8418 |
| 90 | 0.307692 +- 0.0769231 | 1.83957 |
| 91 | 0.336957 +- 0.076087 | 1.8403 |
| 92 | 0.290323 +- 0.0752688 | 1.8405 |
| 93 | 0.319149 +- 0.0744681 | 1.83951 |
| 94 | 0.347368 +- 0.0736842 | 1.84127 |
| 95 | 0.302083 +- 0.0729167 | 1.83976 |
| 96 | 0.329897 +- 0.0721649 | 1.83986 |
| 97 | 0.285714 +- 0.0714286 | 1.84092 |
| 98 | 0.313131 +- 0.0707071 | 1.83949 |
| 99 | 0.34 +- 0.07 | 1.84055 |
| 100 | 0.29703 +- 0.0693069 | 1.84002 |
| 101 | 0.323529 +- 0.0686275 | 1.83961 |
| 102 | 0.281553 +- 0.0679612 | 1.84135 |
| 103 | 0.307692 +- 0.0673077 | 1.83957 |
| 104 | 0.333333 +- 0.0666667 | 1.84006 |
| 105 | 0.292453 +- 0.0660377 | 1.84033 |
| 106 | 0.317757 +- 0.0654206 | 1.8395 |
| 107 | 0.342593 +- 0.0648148 | 1.84078 |
| 108 | 0.302752 +- 0.0642202 | 1.83973 |
| 109 | 0.327273 +- 0.0636364 | 1.83974 |
| 110 | 0.288288 +- 0.0630631 | 1.84068 |
| 111 | 0.3125 +- 0.0625 | 1.83949 |
| 112 | 0.336283 +- 0.0619469 | 1.84026 |
| 113 | 0.298246 +- 0.0614035 | 1.83995 |
| 114 | 0.321739 +- 0.0608696 | 1.83956 |
| 115 | 0.344828 +- 0.0603448 | 1.841 |
| 116 | 0.307692 +- 0.0598291 | 1.83957 |
| 117 | 0.330508 +- 0.059322 | 1.83989 |
| 118 | 0.294118 +- 0.0588235 | 1.84021 |
| 119 | 0.316667 +- 0.0583333 | 1.83949 |
| 120 | 0.338843 +- 0.0578512 | 1.84045 |
| 121 | 0.303279 +- 0.057377 | 1.83971 |
| 122 | 0.325203 +- 0.0569106 | 1.83966 |
| 123 | 0.290323 +- 0.0564516 | 1.8405 |
| 124 | 0.312 +- 0.056 | 1.8395 |
| 125 | 0.333333 +- 0.0555556 | 1.84006 |
| 126 | 0.299213 +- 0.0551181 | 1.8399 |
| 127 | 0.320313 +- 0.0546875 | 1.83953 |
| 128 | 0.341085 +- 0.0542636 | 1.84064 |
| 129 | 0.307692 +- 0.0538462 | 1.83957 |
| 130 | 0.328244 +- 0.0534351 | 1.83978 |
| 131 | 0.295455 +- 0.0530303 | 1.84012 |
| 132 | 0.315789 +- 0.0526316 | 1.83949 |
| 133 | 0.335821 +- 0.0522388 | 1.84022 |
| 134 | 0.303704 +- 0.0518519 | 1.8397 |
| 135 | 0.323529 +- 0.0514706 | 1.83961 |
| 136 | 0.291971 +- 0.0510949 | 1.84037 |
| 137 | 0.311594 +- 0.0507246 | 1.8395 |
| 138 | 0.330935 +- 0.0503597 | 1.83992 |
| 139 | 0.3 +- 0.05 | 1.83986 |

1. График зависимости точности определения от количества точек:

Листинг программы:

#include <iostream>

#include <cmath>

#include <iomanip>

#include <vector>

using namespace std;

double function(double x)//возвращает значение функции в точке

{

return (1 - x) \* (1 - x) + exp(x);

}

void print\_numbers(double k, double a, double b, double x1, double x2, double fa, double fb)//печать таблицы значений для метода золотого сечения

{

cout << "| " << left << setw(2) << k << " | " << left << setw(10) << a << " | ";

cout << left << setw(10) << b << " | " << left << setw(10) << b-a << " | " << left << setw(10) << x1 << " | " << left << setw(10) << x2 << " | ";

cout << left << setw(10) << fa << " | " << left << setw(10) << fb << " |\n";

}

void print\_functions(double k, double x, double length, double minimum)//печать таблицы значений для метода пассивного поиска

{

cout << "| " << left << setw(3) << k << " | " << left << setw(10) << x << " +- " << left << setw(10)<< length << " | "

<< left << setw(10) << minimum <<" |\n";

}

void golden\_section(double a, double b, double epsilon)//метод золотого сечения

{

double tau, c, d, f1, f2;

tau = (1 + sqrt(5)) / 2;//золотое отношение

int k = 1;

c = b - (b - a) / tau;//первая точка по правилу ЗС

d = (b - a) / tau + a;//вторая точка по правилу ЗС

while ((b - a) >= epsilon)

{

f1 = function(c);

f2 = function(d);

if (f1 < f2)

{

b = d;

print\_numbers(k, a, b, c, d, f1, f2);

d = c;

c = b - (b - a) / tau;

}

else

{

a = c;

print\_numbers(k, a, b, c, d, f1, f2);

c = d;

d = (b - a) / tau + a;

}

k++;

}

}

double search\_minimum(vector<double> function)//поиск минимального значения среди найденных

{

double find;

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

{

if (i == 0)

{

find = function[i];

}

else

{

if (find > function[i]) (find = function[i]);

}

}

return find;

}

double argum\_min(double fmin, vector<double> functions, vector<double> arguments)//возвращает аргумент, соответствующий минимальному значению функции

{

int index;

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

{

if (fmin == functions[i])

{

index = i;

break;

}

}

return arguments[index];

}

void optimal\_passive\_search(double a, double b, double epsilon)//оптимальный пассивный поиск

{

int N = static\_cast<int>(((b - a) / (epsilon/2)) - 1);//определение количества точек для необходимой точности

double length, f, fminimal;

vector <double> points;//массив точек

vector <double> argum;//массив аргументов

points.resize(2);

argum.resize(2);

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

{

length = (b - a) / (i + 1);//расстояние между точками

points.resize(points.size()+1);

argum.resize(points.size() + 1);

double step = a;

int iter = 0;

while (iter<points.size())//находим значения функции в точках

{

f = function(step);

points[iter] = f;

argum[iter] = step;

step += length;

iter++;

}

fminimal = search\_minimum(points);

double x\_minimal = argum\_min(fminimal, points, argum);

print\_functions(i, x\_minimal, length, fminimal);

}

}

int main()

{

double a = -4, b = 3, epsilon = 0.1;//дано: отрезок [-4, 3], интервал неопределенности 0.1

cout << "\nUnimodal function: (1-x)^2+exp(x);\nRange: a=" << a << ", b=" << b;

cout << ";\nThe interval of uncertanity: " << epsilon << endl;

cout << "\n----\nMethod of golden section\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "| k | a\_k | b\_k | l\_k | x1 | x2 | f(x1) | f(x2) |\n";

golden\_section(a, b, epsilon);

cout << "\n----\nMethod of optimal passive search\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "| N | x\_min | f(min) |\n";

optimal\_passive\_search(a, b, epsilon);

}

Вывод: На примере унимодальной функции скорость сходимости метода золотого сечения превышает скорость сходимости метода пассивного поиска при условии заданного интервала неопределенности. Также метод золотого сечения предполагает собой сильную оптимизацию вычислений в плане уменьшения их количества а также количества шагов-повторений для достижения необходимой точности.