**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**ОТЧЁТ**

по практической работе работе №2 на тему:   
**«Анализ алгоритмов и программ»**

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-114,Константинов А.Д.

«21» сентября 2022 г. \_\_\_\_\_\_\_\_\_\_\_/А.Д. Константинов/

Принял: к.ф.-м.н., доцент, И.А. Моисеев

«05» октября 2022 г. \_\_\_\_\_\_\_\_\_\_\_/ И.А. Моисеев /

**Содержание:**

[Цель работы: 3](#_Toc114692552)

[Листинг кода: 3](#_Toc114692553)

[Результат работы программы: 8](#_Toc114692554)

[Вывод: 8](#_Toc114692555)

[Приложение: 22](#_Toc114692556)

Цель работы:

Найти локальный минимум (максимум) функции вида f(x) = на заданном интервале [a,b] с заданной точностью ε > 0 одним из способов:

1) Методом «деления отрезка» пополам;

2) Методом «золотого сечения»;

3) Методом «Фибоначчи».

Провести анализ разработанного алгоритма и программы и сравнить с аналогичным решением с помощью алгоритма «пассивного поиска».

# Листинг кода:

#include <iostream>  
#include <cmath>  
#include <ctime>  
bool isMaxRequired = false;  
float func(float x) {  
 if(isMaxRequired)  
 return -(pow(x,3)-x+exp(-x));  
 return pow(x,3)-x+exp(-x);  
}  
void PassiveSearch(float left,float right, float e)  
{  
 srand(time(0));  
 int steps = 0;  
 float minPoint = 0;  
 float minVal = 99999999;  
 for (float i = left; i <= right; i+=e) {  
 steps++;  
 float tmp = func(i);  
 if(minVal > tmp) {  
 minVal = tmp;  
 minPoint = i;  
 }  
 }  
 if(isMaxRequired)  
 std::cout << "Result X:" << minPoint << " Y:" << -func(minPoint) <<" Steps:"<< steps<<" Time:" << clock()/1000.0<< std::endl;  
 else  
 std::cout << "Result X:" << minPoint << " Y:" << func(minPoint) <<" Steps:"<< steps<<" Time:" << clock()/1000.0<< std::endl;  
}  
void Task1\_SearchHalfPart(float a,float b,float e){  
 float x1;  
 float a1=0,b1 = 999999;  
 float a0 = a;  
 float b0 = b;  
 srand(time(0));  
 int steps = 0;  
 while(b1-a1 > e) {  
 float om = (b0 - a0) / 10;  
 float alp0 = (a0 + b0) / 2 - om;  
 float bet0 = alp0 + 2 \* om;  
 float fAlp = func(alp0);  
 float fBet = func(bet0);  
  
  
 if (fAlp <= fBet) {  
 x1 = alp0;  
 a1 = a0;  
 b1 = bet0;  
 } else {  
 a1 = alp0;  
 x1 = bet0;  
 b1 = b0;  
 }  
  
 a0 = a1;  
 b0 = b1;  
 steps++;  
 if(isMaxRequired)  
 std::cout << "x:" << (a1+b1)/2 << " y:" << -func((a1+b1)/2) << std::endl;  
 else  
 std::cout << "x:" << (a1+b1)/2 << " y:" << func((a1+b1)/2) << std::endl;  
 }  
 float res = (a1 + b1) / 2;  
 if(isMaxRequired)  
 std::cout << "Result X:" << res << " Y:" << -func(res) <<" Steps:"<< steps<<" Time:" << clock()/1000.0 << std::endl;  
 else  
 std::cout << "Result X:" << res << " Y:" << func(res) <<" Steps:"<< steps<<" Time:" << clock()/1000.0 << std::endl;  
}  
void Task1\_SearchGoldCut(float a,float b,float e){  
 float x1;  
 float a1=0,b1 = 999999;  
 float a0 = a;  
 float b0 = b;  
 srand(time(0));  
 int steps = 0;  
  
 while(b1-a1 > e) {  
 float om = b0 - a0;  
 float alp0 =a0+(2\*om)/(3+sqrt(5));  
 float bet0 = a0+(2\*om)/(1+sqrt(5));  
 float fAlp = func(alp0);  
 float fBet = func(bet0);  
  
  
 if (fAlp <= fBet) {  
 x1 = alp0;  
 a1 = a0;  
 b1 = bet0;  
 } else {  
 a1 = alp0;  
 x1 = bet0;  
 b1 = b0;  
 }  
  
 a0 = a1;  
 b0 = b1;  
 steps++;  
 if(isMaxRequired)  
 std::cout << "x:" << (a1+b1)/2 << " y:" << -func((a1+b1)/2) << std::endl;  
 else  
 std::cout << "x:" << (a1+b1)/2 << " y:" << func((a1+b1)/2) << std::endl;  
 }  
 float res = (a1 + b1) / 2;  
 if(isMaxRequired)  
 std::cout << "Result X:" << res << " Y:" << -func(res) <<" Steps:"<< steps<<" Time:" << clock()/1000.0 << std::endl;  
 else  
 std::cout << "Result X:" << res << " Y:" << func(res) <<" Steps:"<< steps<<" Time:" << clock()/1000.0 << std::endl;  
}  
float Fib(int n)  
{  
 if(n == 0) return 1;  
 if(n== 1) return 1;  
 return Fib(n-1) + Fib(n-2);  
}  
void Task1\_SearchFib(float a, float b, float e)  
{  
 int f0 = 1;  
 int f1=1;  
 float a0 = a;  
 float b0 = b;  
 float d0 = b0-a0;  
 int N = 1;  
 srand(time(0));  
 int steps = 0;  
 while(d0/Fib(N) >= e)  
 {  
 N++;  
 }  
  
  
 float x1=0;  
 float a1=0,b1 = 999999;  
 int k =0;  
 while(b1-a1 > e) {  
 float om = b0 - a0;  
 float alp0 = a0 + Fib(N-k-2)/Fib(N-k)\*om;  
 float bet0 = a0 + Fib(N-k-1)/Fib(N-k)\*om;  
 float fAlp = func(alp0);  
 float fBet = func(bet0);  
  
 if (fAlp <= fBet) {  
 x1 = alp0;  
 a1 = a0;  
 b1 = bet0;  
 } else {  
 a1 = alp0;  
 x1 = bet0;  
 b1 = b0;  
 }  
  
 a0 = a1;  
 b0 = b1;  
 k++;  
 steps++;  
 if(isMaxRequired)  
 std::cout << "x:" << (a1+b1)/2 << " y:" << -func((a1+b1)/2) << std::endl;  
 else  
 std::cout << "x:" << (a1+b1)/2 << " y:" << func((a1+b1)/2) << std::endl;  
 }  
 float res = (a1 + b1) / 2;  
 if(isMaxRequired)  
 std::cout << "Result X:" << res << " Y:" << -func(res) <<" Steps:"<< steps<<" Time:" << clock()/1000.0 << std::endl;  
 else  
 std::cout << "Result X:" << res << " Y:" << func(res) <<" Steps:"<< steps<<" Time:" << clock()/1000.0 << std::endl;  
}  
int main() {  
 float left,right,e;  
// std::cout << "Current function y=x^3-x+e^(-x)" << std::endl;  
// std::cout << "Type left limit:";  
// std::cin >> left;  
// std::cout << "Type right limit:";  
// std::cin >> right;  
// std::cout << "Type accuracy:";  
// std::cin >> e;  
// std::cout<<std::endl;  
//  
// PassiveSearch(left,right,e);  
 e= 0.000001;  
 std::cout<< "Task 2 Functions(accuracy = 0.001)" << std::endl;  
 std::cout << std::endl<<"Part 1: Searching extremum in range[-5,-3]" << std::endl;  
 left = -5;  
 right = -3;  
 std::cout <<std::endl<< "Method 1: Classic search: "<< std::endl; PassiveSearch(left,right,e);  
 std::cout <<std::endl<< "Method 2: Like-binary search: "<< std::endl;Task1\_SearchHalfPart(left,right,e);  
 std::cout <<std::endl<< "Method 3: Gold-cut search: "<< std::endl;Task1\_SearchGoldCut(left,right,e);  
 std::cout <<std::endl<< "Method 4: Fib search: "<< std::endl;Task1\_SearchFib(left,right,e);  
  
 std::cout << std::endl<<"Part 2: Search Max-points in range [-3,0]"; std::cout << std::endl;  
 left = -3;  
 right = 0;  
 isMaxRequired = true;  
 std::cout <<std::endl<< "Method 1: Classic search: "<< std::endl; PassiveSearch(left,right,e);  
 std::cout <<std::endl<< "Method 2: Like-binary search: "<< std::endl;Task1\_SearchHalfPart(left,right,e);  
 std::cout <<std::endl<< "Method 3: Gold-cut search: "<< std::endl;Task1\_SearchGoldCut(left,right,e);  
 std::cout <<std::endl<< "Method 4: Fib search: "<< std::endl;Task1\_SearchFib(left,right,e);  
 isMaxRequired = false;  
  
 std::cout <<std::endl << "Part 3: Search Min-points in range [0,3]"; std::cout << std::endl;  
 left = 0;  
 right = 3;  
 std::cout <<std::endl<< "Method 1: Classic search: "<< std::endl; PassiveSearch(left,right,e);  
 std::cout <<std::endl<< "Method 2: Like-binary search: "<< std::endl;Task1\_SearchHalfPart(left,right,e);  
 std::cout <<std::endl<< "Method 3: Gold-cut search: "<< std::endl;Task1\_SearchGoldCut(left,right,e);  
 std::cout <<std::endl<< "Method 4: Fib search: "<< std::endl;Task1\_SearchFib(left,right,e);  
  
 std::cout <<std::endl<< "End. Thanks for using our software" << std::endl;  
 return 0;  
}

# Результат работы программы:

# /Users/andrew/Projects/testproj/cmake-build-debug/testproj

# Task 2 Functions(accuracy = 0.001)

# Part 1: Searching extremum in range[-5,-3]

# Method 1: Classic search:

# Result X:-3.67908 Y:-6.50965 Steps:2097153 Time:151.726

# Method 2: Like-binary search:

# x:-3.6 y:-6.45776

# x:-3.84 y:-6.25763

# x:-3.696 y:-6.50706

# x:-3.6096 y:-6.46937

# x:-3.66144 y:-6.507

# x:-3.69254 y:-6.508

# x:-3.67388 y:-6.50943

# x:-3.68508 y:-6.50931

# x:-3.67836 y:-6.50964

# x:-3.68239 y:-6.50954

# x:-3.67997 y:-6.50964

# x:-3.67852 y:-6.50965

# x:-3.67939 y:-6.50965

# x:-3.67887 y:-6.50965

# x:-3.67856 y:-6.50965

# x:-3.67874 y:-6.50965

# x:-3.67863 y:-6.50965

# x:-3.67856 y:-6.50965

# x:-3.6786 y:-6.50965

# x:-3.67863 y:-6.50965

# x:-3.67864 y:-6.50965

# x:-3.67865 y:-6.50965

# x:-3.67866 y:-6.50965

# x:-3.67866 y:-6.50965

# x:-3.67866 y:-6.50965

# x:-3.67866 y:-6.50965

# x:-3.67866 y:-6.50965

# x:-3.67866 y:-6.50965

# x:-3.67866 y:-6.50965

# Result X:-3.67866 Y:-6.50965 Steps:29 Time:151.879

# Method 3: Gold-cut search:

# x:-3.61803 y:-6.47841

# x:-3.8541 y:-6.2089

# x:-3.7082 y:-6.50198

# x:-3.61803 y:-6.47841

# x:-3.67376 y:-6.50942

# x:-3.7082 y:-6.50198

# x:-3.68692 y:-6.50908

# x:-3.67376 y:-6.50942

# x:-3.68189 y:-6.50957

# x:-3.67687 y:-6.50961

# x:-3.67997 y:-6.50964

# x:-3.67805 y:-6.50964

# x:-3.67924 y:-6.50965

# x:-3.67851 y:-6.50964

# x:-3.67896 y:-6.50965

# x:-3.67868 y:-6.50965

# x:-3.67885 y:-6.50965

# x:-3.67896 y:-6.50965

# x:-3.67889 y:-6.50965

# x:-3.67893 y:-6.50965

# x:-3.67891 y:-6.50965

# x:-3.67893 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# Result X:-3.67892 Y:-6.50965 Steps:30 Time:152.03

# Method 4: Fib search:

# x:-3.61803 y:-6.47841

# x:-3.8541 y:-6.2089

# x:-3.7082 y:-6.50198

# x:-3.61803 y:-6.47841

# x:-3.67376 y:-6.50942

# x:-3.7082 y:-6.50198

# x:-3.68692 y:-6.50908

# x:-3.67376 y:-6.50942

# x:-3.68189 y:-6.50957

# x:-3.67687 y:-6.50961

# x:-3.67997 y:-6.50964

# x:-3.67805 y:-6.50964

# x:-3.67924 y:-6.50965

# x:-3.67851 y:-6.50964

# x:-3.67896 y:-6.50965

# x:-3.67868 y:-6.50965

# x:-3.67885 y:-6.50965

# x:-3.67896 y:-6.50965

# x:-3.67889 y:-6.50965

# x:-3.67893 y:-6.50965

# x:-3.67891 y:-6.50965

# x:-3.67893 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# x:-3.67892 y:-6.50965

# Result X:-3.67892 Y:-6.50965 Steps:30 Time:287.31

# Part 2: Search Max-points in range [-3,0]

# Method 1: Classic search:

# Result X:-1.20015 Y:2.79212 Steps:3087674 Time:469.766

# Method 2: Like-binary search:

# x:-0.9 y:2.6306

# x:-1.26 y:2.78505

# x:-1.044 y:2.74666

# x:-1.1736 y:2.79077

# x:-1.25136 y:2.78695

# x:-1.2047 y:2.79207

# x:-1.17671 y:2.79107

# x:-1.19351 y:2.79203

# x:-1.20358 y:2.79209

# x:-1.19754 y:2.7921

# x:-1.20117 y:2.79211

# x:-1.19899 y:2.79211

# x:-1.20029 y:2.79212

# x:-1.19951 y:2.79212

# x:-1.19998 y:2.79212

# x:-1.20026 y:2.79212

# x:-1.20043 y:2.79212

# x:-1.20033 y:2.79212

# x:-1.20039 y:2.79212

# x:-1.20043 y:2.79212

# x:-1.20045 y:2.79212

# x:-1.20046 y:2.79212

# x:-1.20046 y:2.79212

# x:-1.20046 y:2.79212

# x:-1.20046 y:2.79212

# x:-1.20047 y:2.79212

# x:-1.20047 y:2.79212

# x:-1.20047 y:2.79212

# x:-1.20047 y:2.79212

# x:-1.20047 y:2.79212

# Result X:-1.20047 Y:2.79212 Steps:30 Time:470.055

# Method 3: Gold-cut search:

# x:-0.927051 y:2.65737

# x:-1.28115 y:2.77912

# x:-1.06231 y:2.75653

# x:-1.19756 y:2.7921

# x:-1.28115 y:2.77912

# x:-1.22949 y:2.79042

# x:-1.19756 y:2.79211

# x:-1.21729 y:2.79154

# x:-1.2051 y:2.79207

# x:-1.19756 y:2.7921

# x:-1.20222 y:2.79211

# x:-1.19934 y:2.79212

# x:-1.20112 y:2.79211

# x:-1.20002 y:2.79212

# x:-1.2007 y:2.79212

# x:-1.20028 y:2.79212

# x:-1.20002 y:2.79212

# x:-1.20018 y:2.79212

# x:-1.20028 y:2.79212

# x:-1.20022 y:2.79212

# x:-1.20026 y:2.79212

# x:-1.20028 y:2.79212

# x:-1.20029 y:2.79212

# x:-1.2003 y:2.79212

# x:-1.2003 y:2.79212

# x:-1.2003 y:2.79212

# x:-1.2003 y:2.79212

# x:-1.2003 y:2.79212

# x:-1.20031 y:2.79212

# x:-1.20031 y:2.79212

# x:-1.20031 y:2.79212

# x:-1.20031 y:2.79212

# Result X:-1.20031 Y:2.79212 Steps:32 Time:470.2

# Method 4: Fib search:

# x:-0.927051 y:2.65737

# x:-1.28115 y:2.77912

# x:-1.06231 y:2.75653

# x:-1.19756 y:2.79211

# x:-1.28115 y:2.77912

# x:-1.22949 y:2.79042

# x:-1.19756 y:2.79211

# x:-1.21729 y:2.79154

# x:-1.2051 y:2.79207

# x:-1.19756 y:2.79211

# x:-1.20222 y:2.79211

# x:-1.19934 y:2.79212

# x:-1.20112 y:2.79211

# x:-1.20002 y:2.79212

# x:-1.2007 y:2.79212

# x:-1.20028 y:2.79212

# x:-1.20002 y:2.79212

# x:-1.20018 y:2.79212

# x:-1.20008 y:2.79212

# x:-1.20014 y:2.79212

# x:-1.20018 y:2.79212

# x:-1.20016 y:2.79212

# x:-1.20017 y:2.79212

# x:-1.20018 y:2.79212

# x:-1.20019 y:2.79212

# x:-1.20019 y:2.79212

# x:-1.20019 y:2.79212

# x:-1.20019 y:2.79212

# x:-1.20019 y:2.79212

# x:-1.20019 y:2.79212

# x:-1.20019 y:2.79212

# Result X:-1.20019 Y:2.79212 Steps:31 Time:699.993

# Part 3: Search Min-points in range [0,3]

# Method 1: Classic search:

# Result X:0.705581 Y:0.13951 Steps:3087675 Time:869.641

# Method 2: Like-binary search:

# x:0.9 y:0.23557

# x:0.54 y:0.200212

# x:0.756 y:0.145622

# x:0.6264 y:0.153898

# x:0.70416 y:0.139515

# x:0.750816 y:0.144419

# x:0.722822 y:0.140213

# x:0.706026 y:0.139511

# x:0.695949 y:0.139732

# x:0.701995 y:0.139542

# x:0.705623 y:0.13951

# x:0.7078 y:0.139521

# x:0.706494 y:0.139512

# x:0.70571 y:0.13951

# x:0.70524 y:0.139511

# x:0.705522 y:0.13951

# x:0.705691 y:0.13951

# x:0.70559 y:0.13951

# x:0.705529 y:0.13951

# x:0.705565 y:0.13951

# x:0.705543 y:0.13951

# x:0.70553 y:0.13951

# x:0.705522 y:0.13951

# x:0.705518 y:0.13951

# x:0.705515 y:0.13951

# x:0.705517 y:0.13951

# x:0.705516 y:0.13951

# x:0.705516 y:0.13951

# x:0.705516 y:0.13951

# x:0.705516 y:0.13951

# Result X:0.705516 Y:0.13951 Steps:30 Time:869.869

# Method 3: Gold-cut search:

# x:0.927051 y:0.265397

# x:0.572949 y:0.178993

# x:0.791796 y:0.157644

# x:0.656541 y:0.145101

# x:0.740133 y:0.14236

# x:0.688471 y:0.140203

# x:0.7204 y:0.140028

# x:0.700666 y:0.139569

# x:0.712862 y:0.139634

# x:0.705325 y:0.13951

# x:0.709983 y:0.139555

# x:0.707104 y:0.139515

# x:0.705325 y:0.13951

# x:0.706425 y:0.139512

# x:0.705745 y:0.13951

# x:0.705325 y:0.13951

# x:0.705585 y:0.13951

# x:0.705745 y:0.13951

# x:0.705646 y:0.13951

# x:0.705585 y:0.13951

# x:0.705547 y:0.13951

# x:0.70557 y:0.13951

# x:0.705556 y:0.13951

# x:0.705564 y:0.13951

# x:0.705559 y:0.13951

# x:0.705556 y:0.13951

# x:0.705558 y:0.13951

# x:0.705556 y:0.13951

# x:0.705556 y:0.13951

# x:0.705556 y:0.13951

# x:0.705556 y:0.13951

# x:0.705556 y:0.13951

# Result X:0.705556 Y:0.13951 Steps:32 Time:870.04

# Method 4: Fib search:

# x:0.927051 y:0.265397

# x:0.572949 y:0.178993

# x:0.791796 y:0.157644

# x:0.656541 y:0.145101

# x:0.740133 y:0.14236

# x:0.688471 y:0.140203

# x:0.7204 y:0.140028

# x:0.700667 y:0.139569

# x:0.712862 y:0.139634

# x:0.705325 y:0.13951

# x:0.709983 y:0.139555

# x:0.707104 y:0.139515

# x:0.705325 y:0.13951

# x:0.706425 y:0.139512

# x:0.705745 y:0.13951

# x:0.705325 y:0.13951

# x:0.705585 y:0.13951

# x:0.705745 y:0.13951

# x:0.705646 y:0.13951

# x:0.705585 y:0.13951

# x:0.705547 y:0.13951

# x:0.70557 y:0.13951

# x:0.705556 y:0.13951

# x:0.705564 y:0.13951

# x:0.705559 y:0.13951

# x:0.705556 y:0.13951

# x:0.705558 y:0.13951

# x:0.705559 y:0.13951

# x:0.705558 y:0.13951

# x:0.705559 y:0.13951

# x:0.705558 y:0.13951

# Result X:0.705558 Y:0.13951 Steps:31 Time:1084.14

# End. Thanks for using our software

# Process finished with exit code 0

# Вывод:

В ходе выполнения практической работы была написана программа, которая находит локальные минимумы(максимумы) функции на трех отрезках – [-5,-3];[-3,0];[0,3]. Каждый промежуток решается при помощи четырех различных методов – пассивный поиск(последовательный перебор чисел), поиск делением отрезка пополам, поиск золотым сечение, поиск методом Фибоначчи. Самым тяжелым для исполнения является пассивный поиск – он выполняет 300000 операций для поиска подходящего числа, самый быстрый алгоритм - поиск методом деления пополам – 30 итераций, что в среднем на одну итерацию алгоритма меньше чем остальные, что является чрезвычайно хорошим результатом. По точности все алгоритмы выдают заданные значения – 0.000001 – максимальное различие ответов. Вся программа поделена на 6 функций и работает на вывод в текстовом режиме. В результате выводит все расчеты и заканчивает свою работу.

# Приложение:

Тестовые файлы(a.txt;b.txt). и файл исходного кода main.cpp можно найти по адресу <https://github.com/StSiRe/Algoritms/tree/master/Algorithms%20and%20Data%20Structures/Second%20task/second>