**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт №3. Системы управления, информатика и электроэнергетика. Кафедра №304 Вычислительные машины, системы и комплексы

Отчет по лабораторной работе

по учебной дисциплине

теория оптимального планирования и управления

на тему

**«Методы одномерной безусловной оптимизации»**

Группа М30-207Б

Выполнили студенты:

Кривонос А.А.

Приняли:

Татарникова Е.М.

Давыдкина Е.А.

Москва, 2019г.

Содержание**:**

1. Задание………………………………………………………………......... 2

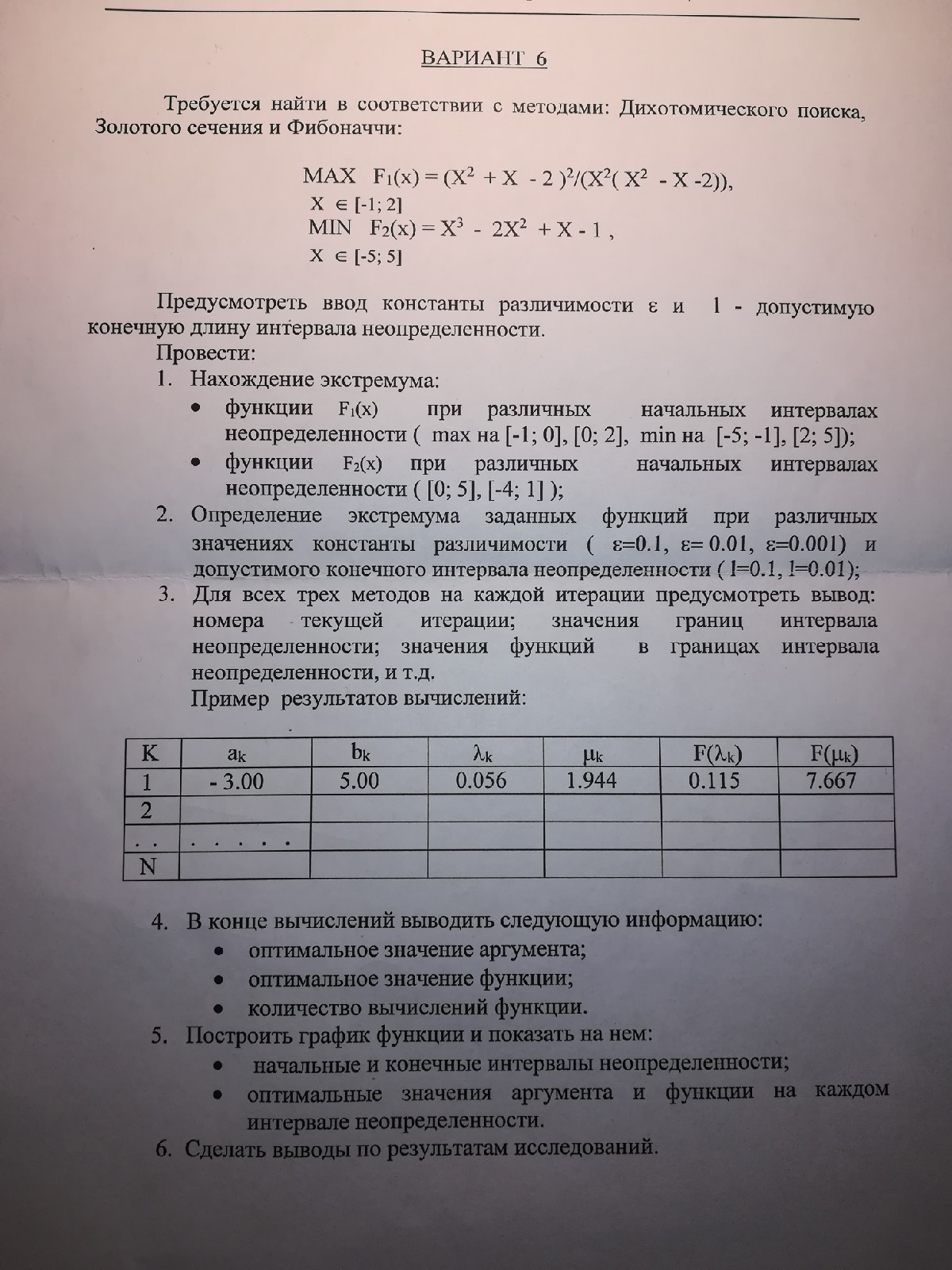
2. Текст программы……….………………………………………………… 3

3. Графики исходных функций…….……..................................................... 11

4. Тесты……………………………………………….................................... 12

5. Вывод………………….………………………………...............................24

Задание



Текст программы

#include <cmath>

#include <stdio.h>

#include <iostream>

#include <iomanip>

using namespace std;

//метод дихотомии

void dichotomy(double eps, double l, double a, double b, double &aOpt, int &kCount, double &bOpt, double(\*func)(double));

//метод золотого сечения

void goldenRatio(double l, double a, double b, double &aOpt, double &bOpt, int &kCount, double(\*func)(double));

//метод Фибоначи

void Fibonachi(double eps, double l, double a, double b, double &aOpt, int &kCount, double &bOpt, double(\*func)(double));

//первая функция

double f1(double x);

//вторая функция

double f2(double x);

//число Фибоначи

double Fib(int i);

//вывод шапки

void shapka();

//вывод низа

void niz();

//инициализация переменных

void Initialization(double &eps, double &l, double &A, double &B);

//вывод информации после вычислений

void afterFunc(double aX, double bX, int Count, double(\*func)(double x));

int main()

{

double eps = 0.; //константа различимости (точность)

double l = 0.; //длина интервала неопределенности

double A = 0.; //начало интервала

double B = 0.; //конец интервала

double aX = 0.; //оптимальное значение начала интервала

double bX = 0.; //оптимальное значение конца интервала

int Count = 0; //счетчик вычисления функций

cout << "FOR F1(X)" << endl;

Initialization(eps, l, A, B); //инициализация переменных для функции 1

cout << "\nTHE DICHOTOMY METHOD\n";

dichotomy(eps, l, A, B, aX, Count, bX, (\*f1)); //дихотомический метод

afterFunc(aX, bX, Count, (\*f1)); //вывод информации после вычислений

cout << "\nTHE GOLDEN RATIO METHOD\n";

goldenRatio(l, A, B, aX, bX, Count, (\*f1)); //метод золотого сечения

afterFunc(aX, bX, Count, (\*f1)); //вывод информации после вычислений

cout << "\nTHE FIBONACCI METHOD\n";

Fibonachi(eps, l, A, B, aX, Count, bX, (\*f1)); //метод Фибоначи

afterFunc(aX, bX, Count, (\*f1)); //вывод информации после вычислений

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "FOR F2(X)" << endl;

Initialization(eps, l, A, B);

cout << "\nTHE DICHOTOMY METHOD\n";

dichotomy(eps, l, A, B, aX, Count, bX, (\*f2)); //дихотомический метод

afterFunc(aX, bX, Count, (\*f2)); //вывод информации после вычислений

cout << "\nTHE GOLDEN RATIO METHOD\n";

goldenRatio(l, A, B, aX, bX, Count, (\*f2)); //метод золотого сечения

afterFunc(aX, bX, Count, (\*f2)); //вывод информации после вычислений

cout << "\nTHE FIBONACCI METHOD\n";

Fibonachi(eps, l, A, B, aX, Count, bX, (\*f2)); //метод Фибоначи

afterFunc(aX, bX, Count, (\*f2)); //вывод информации после вычислений

system("pause");

return 0;

}

double f1(double x){ //задание функции 1

double f;

f =(-1)\*((x\*x + x - 2)\*(x\*x + x - 2)) / (x\*x\*x\*x - x \* x\*x - 2); //т.к. ищем max

return f;

}

double f2(double x) { //задание функции 2

double f;

f = x \* x\*x - 2 \* x\*x + x - 1; // т.к ищем min

return f;

}

void Initialization(double &eps, double &l, double &A, double &B) {

cout << "Put in eps= ";

cin >> eps;

cout << "Put in l= ";

cin >> l;

cout << "Put in a= ";

cin >> A;

cout << "Put in b= ";

cin >> B;

}

void afterFunc(double aX, double bX, int Count, double (\*func)(double x)) {

cout << "Function calculated " << Count << " times\n";

cout << "Optimal x= " << (bX + aX) / 2 << "\n";

cout << "Optimal F(x)= " << func((bX + aX) / 2) << "\n";

}

void dichotomy(double eps, double l, double a, double b, double &aOpt, int &kCount, double &bOpt, double(\*func)(double)){

int k = 1; //счётчик итераций

double lam; //лямбда

double mu; //мю

shapka(); //вывод шапки таблицы

while (b - a >= l){ //пока интервал больше длины интервала неопределенности

cout << char(179) << setw(5) << k;

lam = (a + b) / 2 - eps;

mu = (a + b) / 2 + eps;

cout << char(179) << setw(10) << setprecision(4) << a

<< char(179) << setw(10) << setprecision(4) << b

<< char(179) << setw(10) << setprecision(4) << lam

<< char(179) << setw(10) << setprecision(4) << mu

<< char(179) << setw(10) << setprecision(4) << func(lam)

<< char(179) << setw(10) << setprecision(4) << func(mu)

<< char(179) << endl;

cout << char(179) << setw(6);

cout << char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << endl;

if (func(lam) < func(mu)) //если значение функции в точке лямбда меньше значения в точке мю

b = mu;

else

a = lam;

k++;

}//while

niz(); //закрываем таблицу

cout << endl;

aOpt = a;

bOpt = b;

kCount = (k - 1) \* 2 + 1;

}

void goldenRatio(double l, double a, double b, double &aOpt, double &bOpt, int &kCount, double(\*func)(double)){

const double al = 0.618; //значение альфа

int k = 1; //иттерация

double lam = a + (1 - al)\*(b - a); //вычисление значения лямбды

double mu = a + al \* (b - a); //вычисление значения мю

double fLam = func(lam); //вычисление значения функции лямбды

double fMu = func(mu); //вычисление значения функции мю

shapka();

while (b - a >= l){ //пока интервал больше длины интервала неопределенности

cout << char(179) << setw(5) << k;

cout << char(179) << setw(10) << setprecision(4) << a

<< char(179) << setw(10) << setprecision(4) << b

<< char(179) << setw(10) << setprecision(4) << lam

<< char(179) << setw(10) << setprecision(4) << mu

<< char(179) << setw(10) << setprecision(4) << func(lam)

<< char(179) << setw(10) << setprecision(4) << func(mu)

<< char(179) << endl;

cout << char(179) << setw(6);

cout << char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << endl;

if (fLam > fMu){

a = lam;

lam = mu;

fLam = fMu;

mu = a + al \* (b - a);

fMu = func(mu);

}//if

else{

b = mu;

mu = lam;

fMu = fLam;

lam = a + (1 - al)\*(b - a);

fLam = func(lam);

}//else

k++; //инкрементируем иттерационную переменную

if (b - a < l){ //если интервал меньше длины интервала неопределенности

cout << char(179) << setw(5) << k;

cout << char(179) << setw(10) << setprecision(4) << a

<< char(179) << setw(10) << setprecision(4) << b

<< char(179) << setw(10) << setprecision(4) << lam

<< char(179) << setw(10) << setprecision(4) << mu

<< char(179) << setw(10) << setprecision(4) << func(lam)

<< char(179) << setw(10) << setprecision(4) << func(mu)

<< char(179) << endl;

cout << char(179) << setw(6);

cout << char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << endl;

}//endif

}//while

niz();//закрываем таблицу

cout << endl;

aOpt = a;

bOpt = b;

kCount = k + 2;

}

double Fib(int i){

if (i < 0) return 0;

if (i == 0) return 0;

if (i == 1) return 1;

return Fib(i - 1) + Fib(i - 2);

}

void Fibonachi(double eps, double l, double a, double b, double &aOpt, int &kCount, double &bOpt, double(\*func)(double)){

int n = 1;

int j = 0;

while (Fib(n) < abs(b - a) / l) //условие для выбора числа иттераций

n++;

double lam = a + ((Fib(n - 2) / Fib(n)) \* abs(b - a)); //вычисление значения лямбда

double mu = a + ((Fib(n - 1) / Fib(n)) \* abs(b - a)); //вычисление значения мю

double fLam = func(lam); //вычисление значения функции в точке лямбда

double fMu = func(mu); //вычисление значения функции в точке мю

int k = 1;

double aPrev = a; //инициализация предыдущего значения а

double bPrev = b; //инициализация предыдущего значения b

double lamPrev = lam; //инициализация предыдущего значения лямбда

shapka(); //вывод шапки

while (k <= (n - 2))

{

cout << char(179) << setw(5) << k

<< char(179) << setw(10) << setprecision(4) << a

<< char(179) << setw(10) << setprecision(4) << b;

if (func(lam) > func(mu))

{

aPrev = a;

a = lam;

lamPrev = lam;

lam = mu;

mu = a + (Fib(n - k) / Fib(n - k + 1))\*(b - a);

if (k != (n - 2))

fMu = func(mu);

}

else

{

bPrev = b;

b = mu;

mu = lam;

lamPrev = lam;

lam = a + (Fib(n - k - 1) / Fib(n - k + 1))\*(b - a);

if (k != (n - 2))

fLam = func(lam);

}

if (k == (n - 2)){

lam = lamPrev;

mu = lam + eps;

if ((func(lam)) > (func(mu))){

a = lam;

b = bPrev;

}

else{

a = aPrev;

b = mu;

}

}

k++;

cout << char(179) << setw(10) << setprecision(4) << lam

<< char(179) << setw(10) << setprecision(4) << mu

<< char(179) << setw(10) << setprecision(4) << func(lam)

<< char(179) << setw(10) << setprecision(4) << func(mu)

<< char(179) << endl;

cout << char(179) << setw(6);

cout << char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << setw(11) << setprecision(4)

<< char(179) << endl;

}//while

niz(); //закрываем таблицу

cout << endl;

aOpt = a;

bOpt = b;

kCount = k + 1;

}

void shapka() {

int j = 0;

//верхняя крышка таблицы

cout << char(218);

for (j = 0; j < 5; j++) cout << char(196); cout << char(194);

for (j = 0; j < 10; j++) cout << char(196); cout << char(194);

for (j = 0; j < 10; j++) cout << char(196); cout << char(194);

for (j = 0; j < 10; j++) cout << char(196); cout << char(194);

for (j = 0; j < 10; j++) cout << char(196); cout << char(194);

for (j = 0; j < 10; j++) cout << char(196); cout << char(194);

for (j = 0; j < 10; j++) cout << char(196); cout << char(191) << endl;

//шапка таблицы

cout << char(179) << setw(5) << "K"

<< char(179) << setw(10) << "a(k)"

<< char(179) << setw(10) << "b(k)"

<< char(179) << setw(10) << "lam(k)"

<< char(179) << setw(10) << "mu(k)"

<< char(179) << setw(10) << "F(lam(k))"

<< char(179) << setw(10) << "F(mu(k))" << char(179) << endl;

cout << char(195);

for (j = 0; j < 5; j++) cout << char(196); cout << char(197);

for (j = 0; j < 10; j++) cout << char(196); cout << char(197);

for (j = 0; j < 10; j++) cout << char(196); cout << char(197);

for (j = 0; j < 10; j++) cout << char(196); cout << char(197);

for (j = 0; j < 10; j++) cout << char(196); cout << char(197);

for (j = 0; j < 10; j++) cout << char(196); cout << char(197);

for (j = 0; j < 10; j++) cout << char(196); cout << char(180) << endl;

}

void niz() {

int j = 0;

//низ таблицы

cout << char(192);

for (j = 0; j < 5; j++) cout << char(196); cout << char(193);

for (j = 0; j < 10; j++) cout << char(196); cout << char(193);

for (j = 0; j < 10; j++) cout << char(196); cout << char(193);

for (j = 0; j < 10; j++) cout << char(196); cout << char(193);

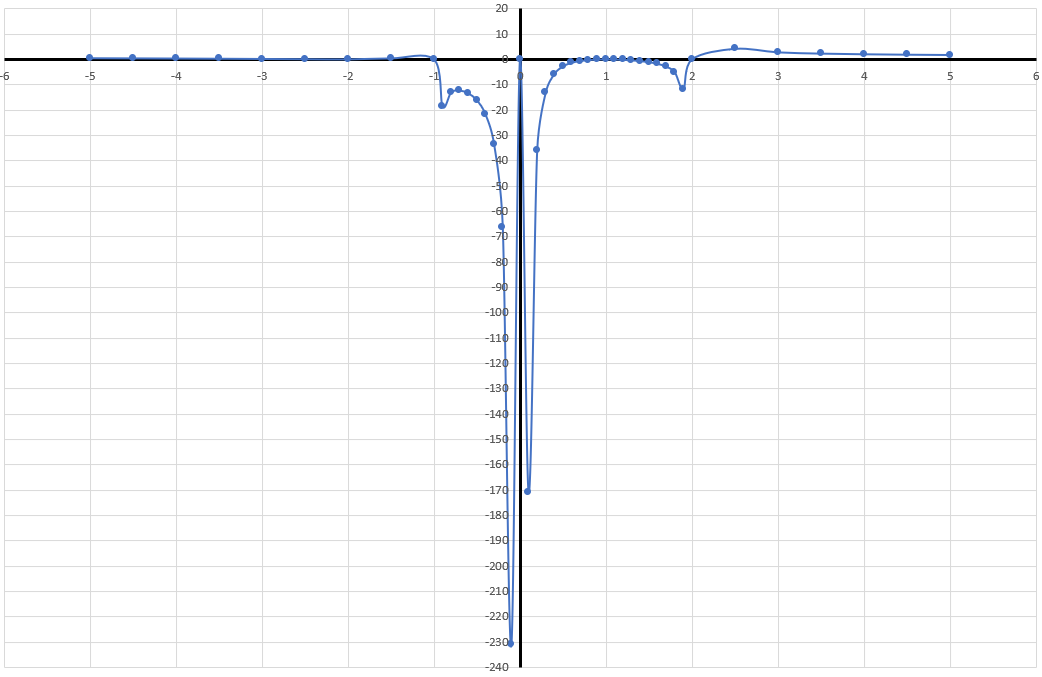
for (j = 0; j < 10; j++) cout << char(196); cout << char(193);

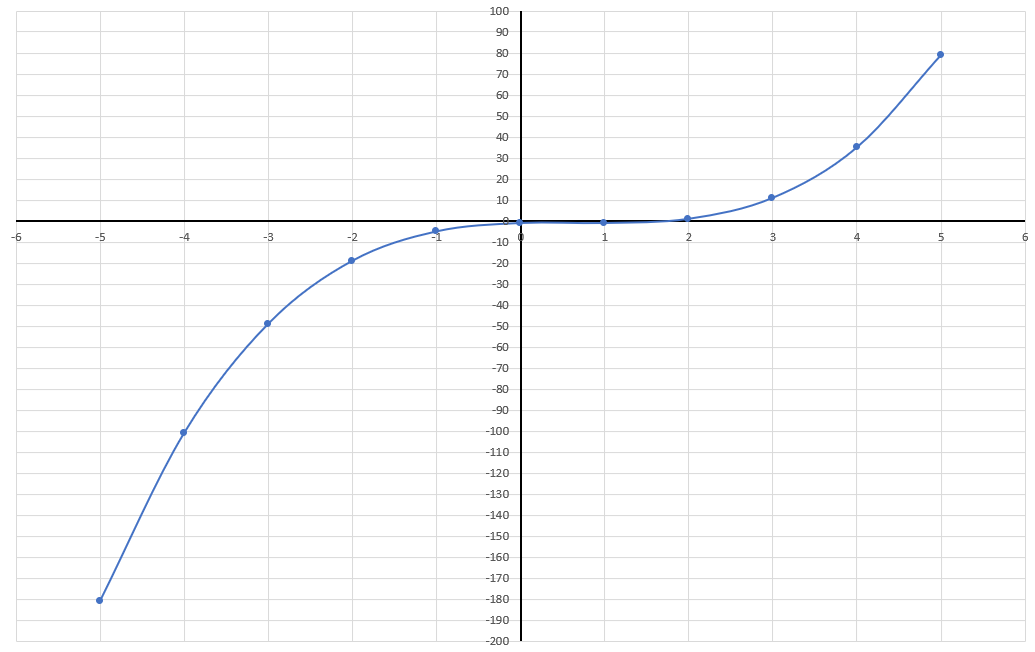
for (j = 0; j < 10; j++) cout << char(196); cout << char(193);

for (j = 0; j < 10; j++) cout << char(196); cout << char(217);

}

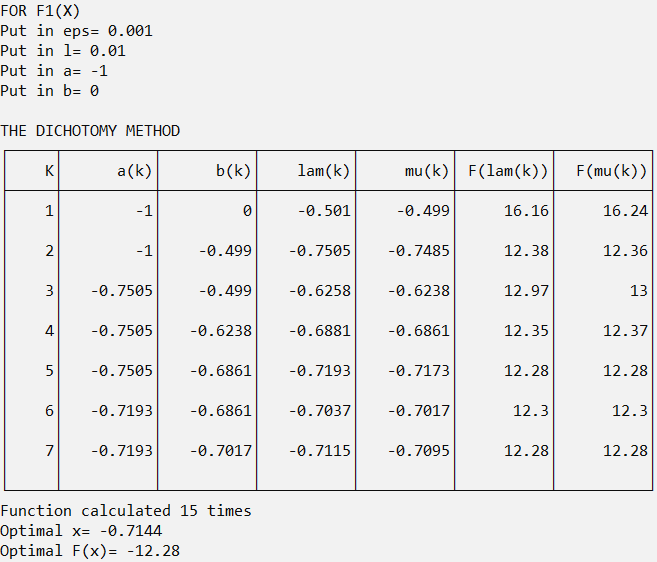
Графики исходных функций

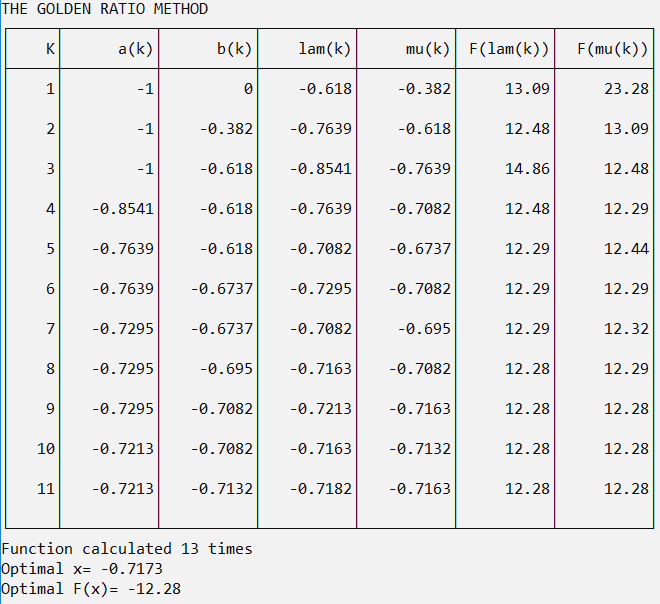
1)F1(x)= 

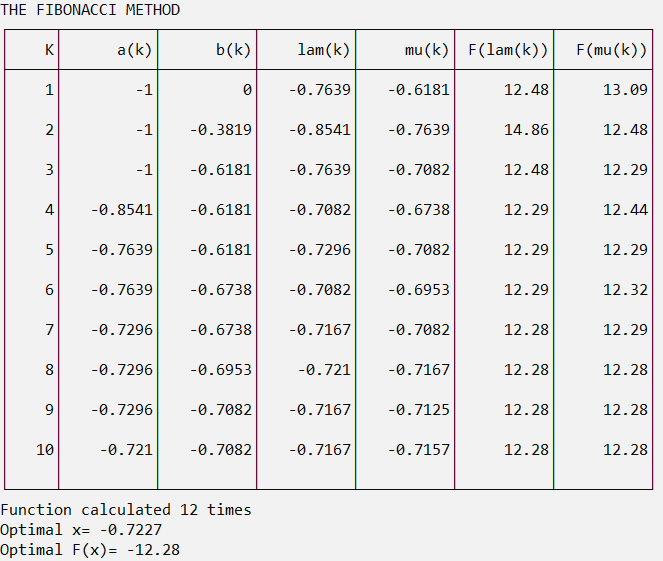
2)F2(x)=

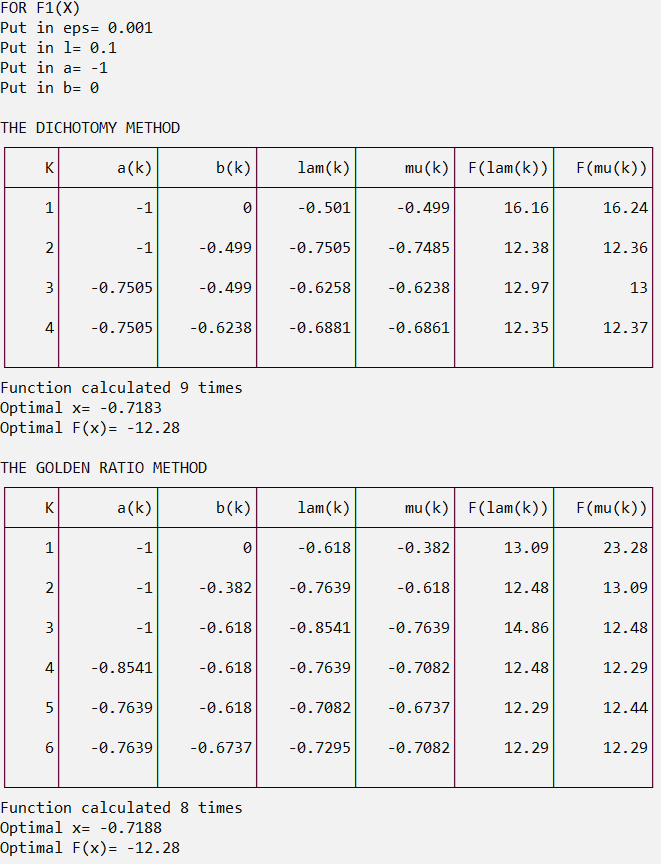
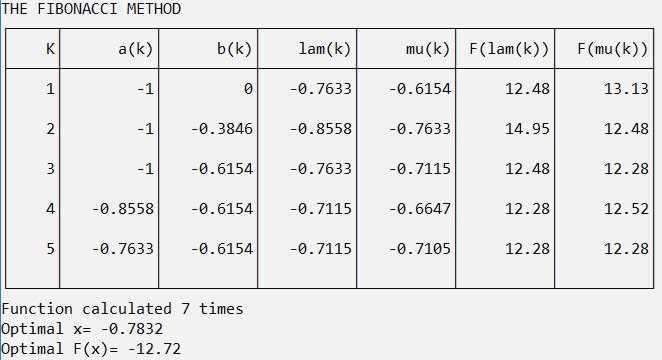
Тесты

1)Для первой функции максимум на интервале неопределенности (-1;0)

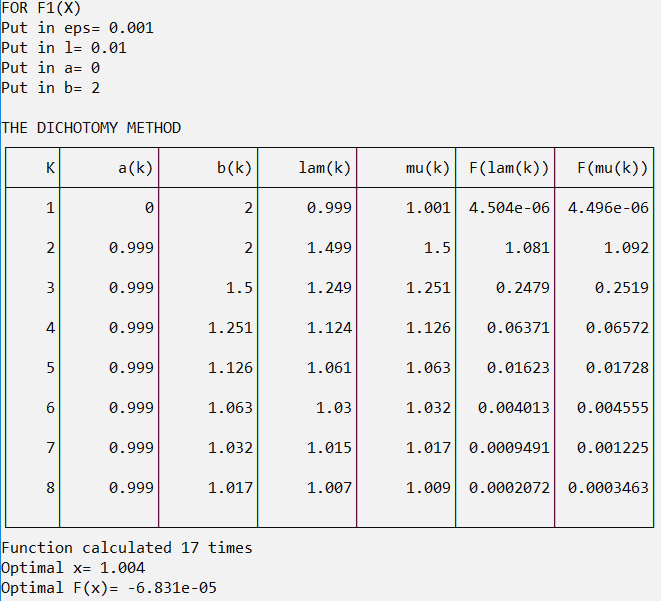


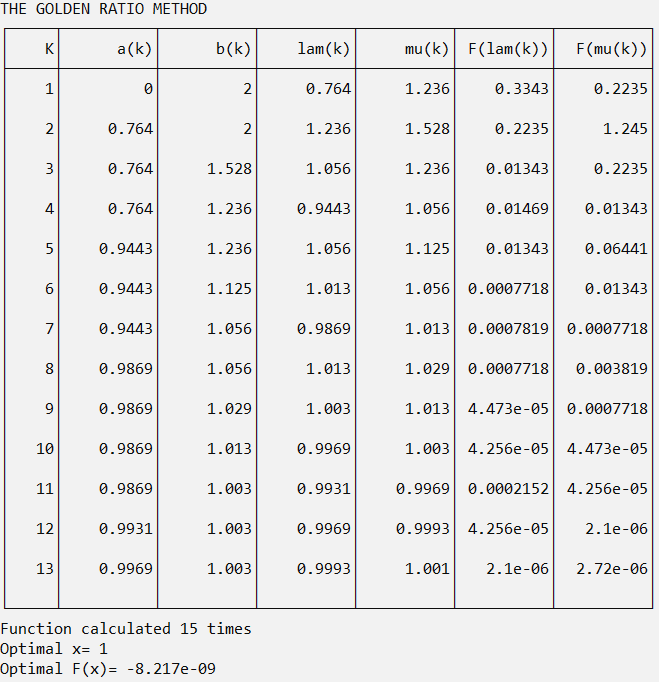


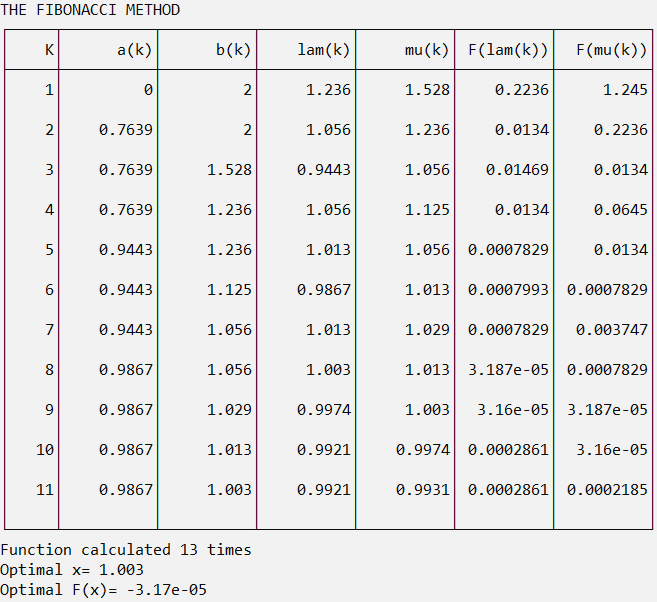


Константу различимости оставим прежней, а допустимую конечную длину инт-ла неопр. Увеличим (если константа различимости больше или равна интервалу неопределенности, то программа зацикливается и процес вычисления оптимальных точки и значения становится бесконечным):

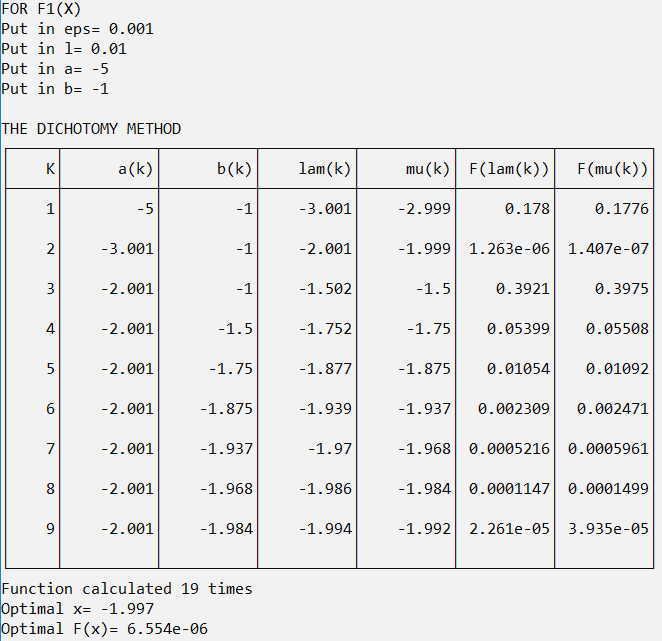
2)Для первой функции максимум на интервале неопределенности (0;2)

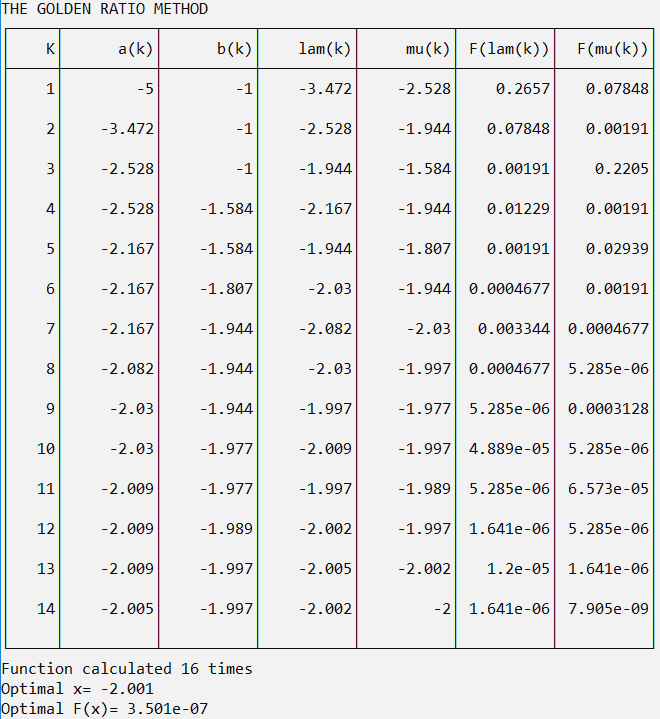
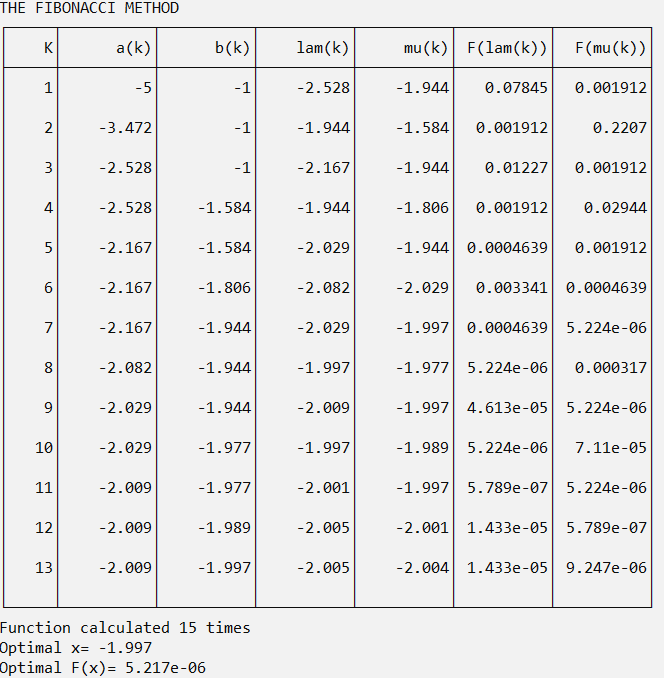




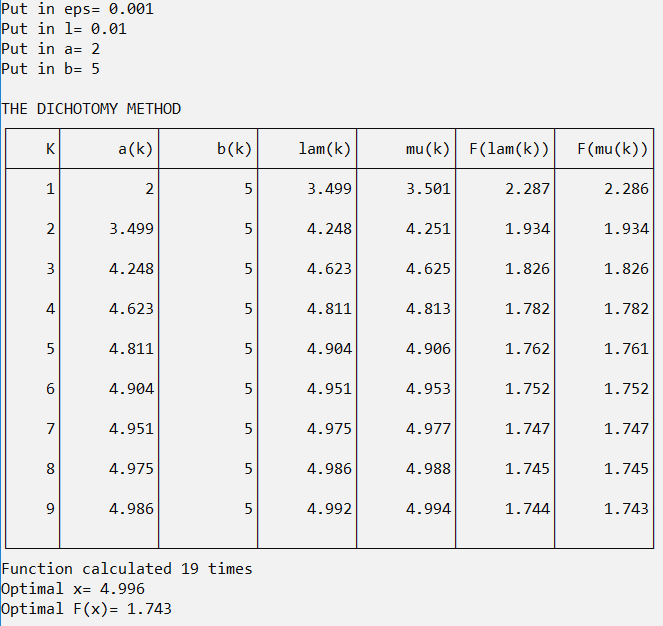


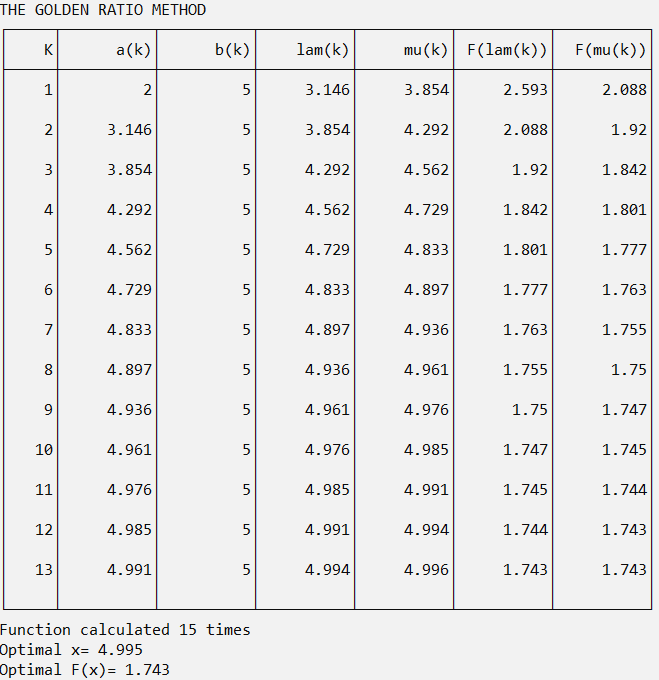
3)Для первой функции минимум на интервале неопределенности (-5;-1)

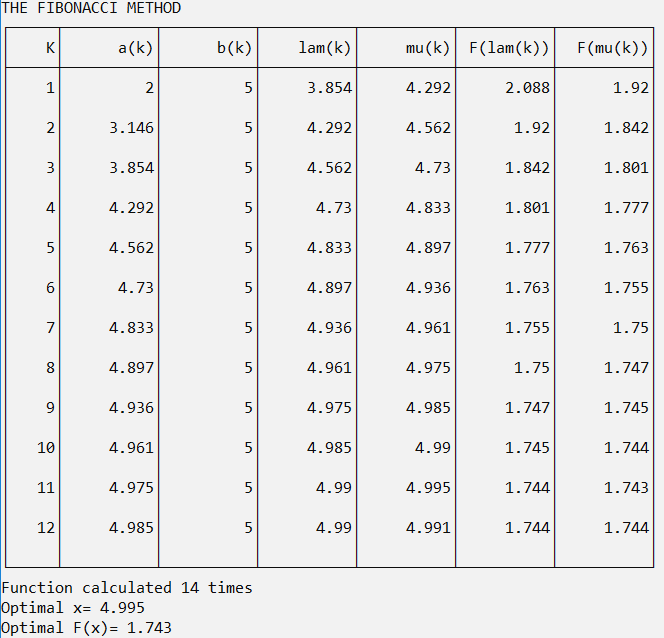




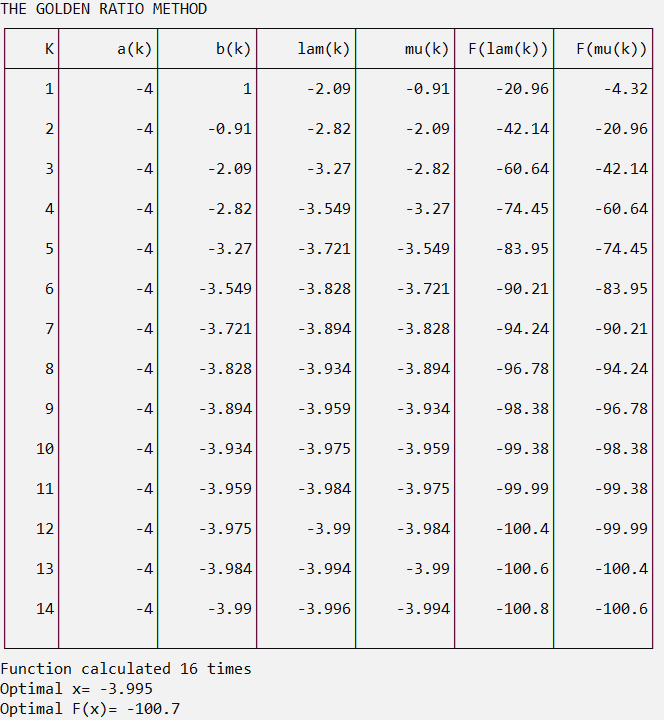
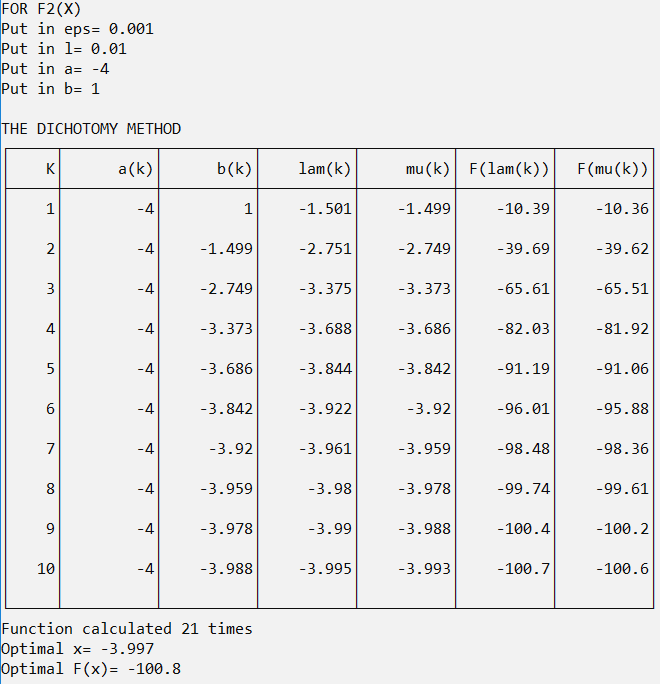
4)Для первой функции минимум на интервале неопределенности (2;5)

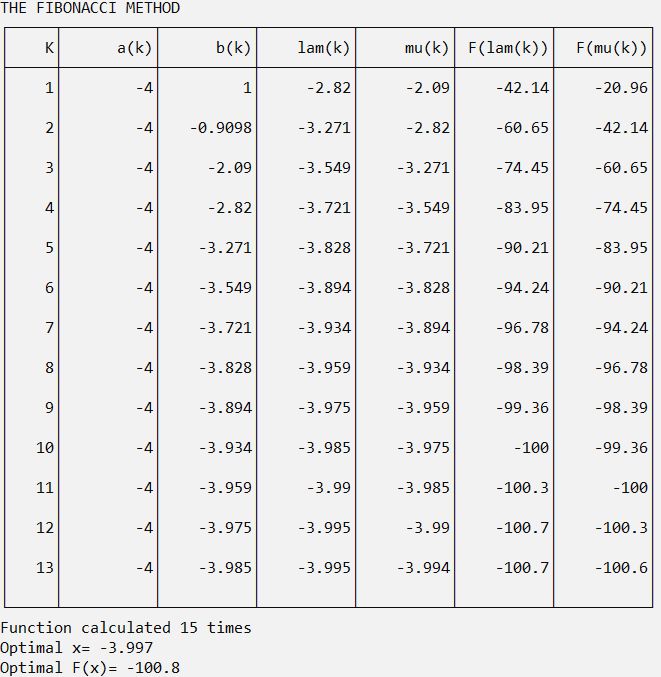




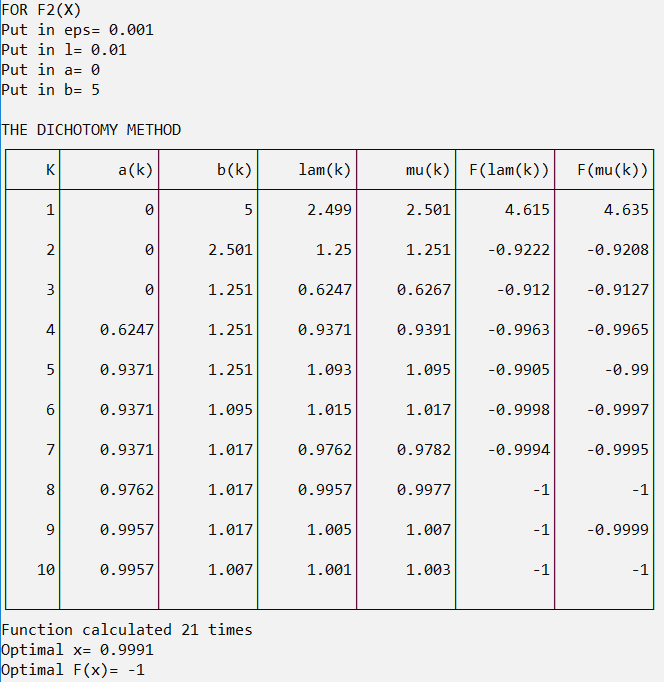


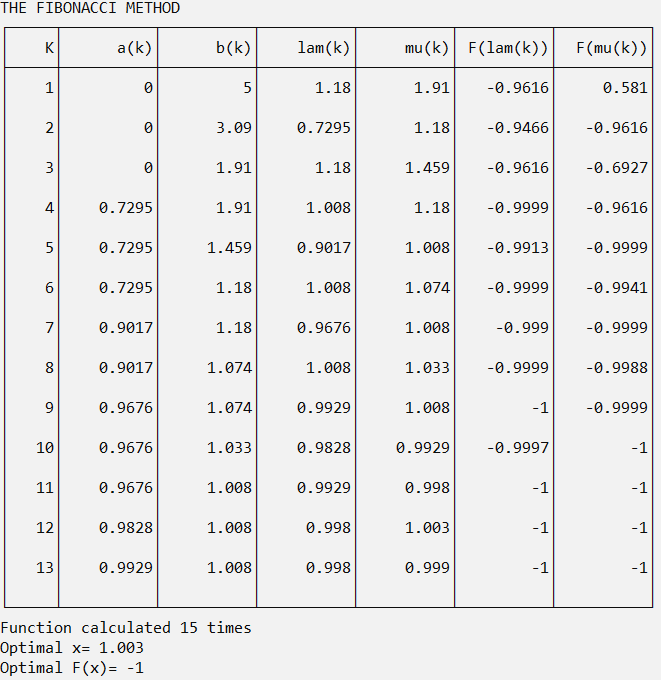
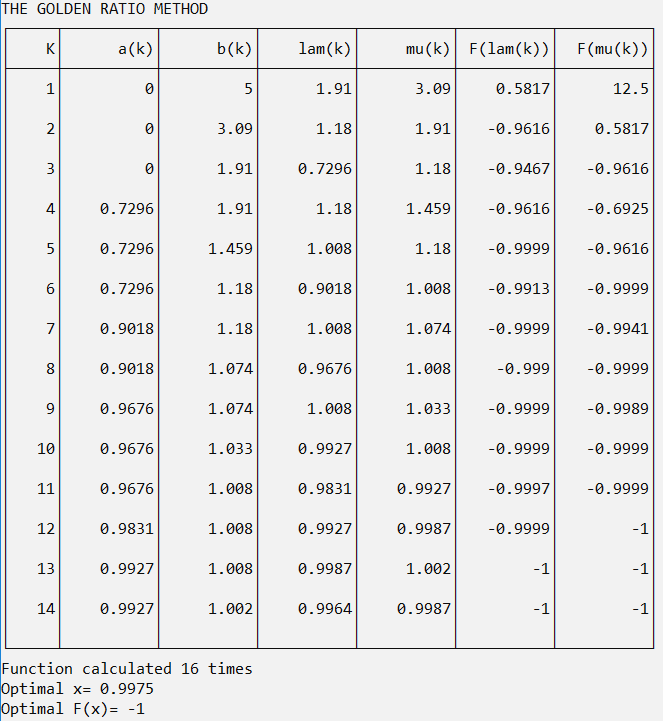
5)Для второй функции минимум на интервале неопределенности (-4;1)





6)Для второй функции минимум на интервале неопределенности (0;5)



  
Вывод

В первую очередь отметим, что для успешной работы программы следует выбирать l>2ε (для исключения зацикливания).

Сравним методы:

Результаты вычислений разных методов совпали между собой, однако использование МДП оказывается менее эффективным, в следствие необходимости вычисления функции по два раза на каждой итерации, в отличии от МЗС, где два раза функция вычисляется исключительно на первой итерации, а на последующих функция рассчитывается по одному разу, несмотря на сравнительно большее количество итераций в МЗС.

Из сравнения результатов работы программы для двух функций с их графиками на заданных промежутках поиска экстремума можно заметить, что методы Дихотомического поиска и Золотого сечения гораздо точнее работают с унимодальными функциями, тогда как метод Фибоначчи достаточно точен как для не унимодальных, так и для унимодальных функций.

При уменьшении константы различимости ε (а, следовательно, и уменьшении допустимой конечной длины интервала l) наблюдается увеличение количества итераций для всех методов.

Метод Дихотомического поиска вычислительно самый простой, но требует самого большого числа вычислений функции. Чувствителен к повышению точности вычислений (уменьшению ε и l – заметны изменения оптимальных значений аргумента и функции), а также плохо работает с не унимодальными функциями

Метод Золотого сечения схож с предыдущим, но сложнее вычислительно, однако требует меньшего числа вычислений функции. Почти не чувствителен к смене точности. Плох в работе с не унимодальными функциями.

Метод Фибоначчи – самый вычислительно сложный из всех рассматриваемых, требует столько же вычислений функции, сколько и метод Золотого сечения, но в работе с НЕ унимодальной функцией показал себя лучше всех, найдя очень точное значение экстремума. К повышению точности почти не чувствителен.

Сравним численные значения:

Для большей точности (0.001 по сравнению с 0.1) наблюдаются более точные результаты ,а также наиболее верное количество итераций метода ЗС по сравнению с МДП

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| f | Метод | Интер-вал | Eps | L | Направление  оптимизации | Итерации | Расчет функции | x\* | f(x)\* |
| F1 | Дихотомии | (-1;0) | 0.001 | 0.01 | max | 7 | 15 | -0.714 | -12.28 |
| Золотое сечение | 11 | 13 | -0.713 | -12.28 |
| Фибоначи | 10 | 12 | -0.7227 | -12.28 |
| F1 | Дихотомии | (-1;0) | 0.001 | 0.1 | max | 4 | 9 | -0.718 | -12.28 |
| Золотое сечение | 6 | 8 | -0.718 | -12.28 |
| Фибоначи | 5 | 7 | -0.7832 | -12.72 |
| F1 | Дихотомии | (0;2) | 0.001 | 0.01 | max | 8 | 17 | 1.004 | -6.8e-05 |
| Золотое сечение | 13 | 15 | 1 | -8.2e-09 |
| Фибоначи | 11 | 13 | 1.003 | -3.2e-05 |
| F1 | Дихотомии | (-5;1) | 0.001 | 0.01 | min | 9 | 19 | -1.997 | 6.55e-06 |
| Золотое сечение | 14 | 16 | -2.001 | 3.5e-07 |
| Фибоначи | 13 | 15 | -1.997 | 5.22e-06 |
| F1 | Дихотомии | (2;5) | 0.001 | 0.01 | min | 9 | 19 | 4.996 | 1.743 |
| Золотое сечение | 13 | 15 | 4.995 | 1.743 |
| Фибоначи | 12 | 14 | 4.995 | 1.743 |
| F2 | Дихотомии | (-4;1) | 0.001 | 0.01 | min | 10 | 21 | -3.997 | -100.8 |
| Золотое сечение | 14 | 16 | -3.995 | -100.7 |
| Фибоначи | 13 | 15 | -3.997 | -100.8 |
| F2 | Дихотомии | (0;5) | 0.001 | 0.01 | min | 10 | 21 | 0.9991 | -1 |
| Золотое сечение | 14 | 16 | 0.9975 | -1 |
| Фибоначи | 13 | 15 | 1.003 | -1 |