ФЕДЕРАЛЬНОЕ АГЕНТСВО ПО ОБРАЗОВАНИЮ РОССИЙСКОЙ ФЕДЕРАЦИИ

УФИМСКИЙ ГОСУДАРСТВЕННЫЙ АВИАЦИОННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Общенаучный факультет

Кафедра ВВТиС

Отчет по лабораторным работам №1

**По дисциплине «Методы оптимизации»**

Группа ПМИ-440

Студент \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ Сайфутдинов Р.Ф.

(дата) (подпись) (Фамилия И.О.)

Проверил \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ Касаткин А. А.

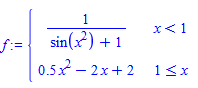
(дата) (подпись) (Фамилия И.О.)

Уфа 2016

**Цель работы:** Изучить численные методы поиска безусловного экстремума скалярной действительной функции.

**Постановка задачи**

Рассмотрим задачу оптимизации функции

**

***Необходимое условие экстремума:*** Если точка x_{0} – точка экстремума функции f(x), то она критическая.

***Первое достаточное условие экстремума в терминах первой производной:*** Пусть функция f(x) определена и [дифференцируема](http://ib.mazurok.com/2013/05/differenciable-and-differencial/) в некоторой окрестности точки x_{0}, кроме, быть может, самой точки x_{0} и непрерывна в этой точке. Тогда:

1. Если производная {f}' меняет знак с “-” на “+” при переходе через точку x_{0}: \forall x\in  (x_{0}-\delta ;x_{0}) {f}'(x)< 0 и \forall x\in  (x_{0}; x_{0}+\delta) {f}'(x)>  0, то x_{0} – точка строго минимума функции f(x).
2. Если производная {f}' меняет знак с “+” на “-” при переходе через точку x_{0}: \forall x\in  (x_{0}-\delta;x_{0} ){f}'(x)> 0 и  \forall x\in  (x_{0}; x_{0}+\delta) {f}'(x)<  0, то x_{0} – точка строго максимума функции f(x).

***Второе достаточное условие строгого экстремума в терминах второй производной:***

Пусть дана функция f(x), она определена в некоторой окрестности точки

x_{0} ,  ее первая [производная](http://ib.mazurok.com/2013/05/derivative-definition/) {f}'(x_{0})=0 и пусть \exists {f}''(x_{0}), тогда:

1. Если {f}''(x_{0})>0, то точка x_{0} – точка строгого минимума;
2. Если {f}''(x_{0})<0, то точка x_{0} – точка строгого максимума.

**Алгоритм численного метода решения поставленной задачи:**

1. Находим интервал поиска минимального значения функции методом Свенна:
2. Задать произвольно точку , величину шага ,
3. Вычислить значения
4. Если , то функция не является унимодальной, и интервал поиска решения не может быть найден,
5. Если , то интервал поиска решения найден
6. Если , то
7. Если , то ,
8. определить величину

* если , то ,
* если , то ,

1. Ищется следующая точка
2. Если и , то ,

если и , то ,

, перейти к шагу 8.

1. Если , интервал поиска решения найден. Если , то , если , то .
2. Находим приближенное решение задачи безусловной минимизации численным методом поиска безусловного экстремума – методом Дихотомии (деления отрезка пополам):
3. Зададим следующие значения: требуемая точность решения. Вычислим .
4. Определим и вычислим . Если , то . Если , тои вычислим . Если , то . Если , то . Переходим к пункту 3.
5. Если , то поиск решения завершен. В качестве решения можно взять . Если , то увеличим номер шага и переходим к пункту 2.
6. Находим приближенное решение задачи безусловной минимизации численным методом поиска безусловного экстремума – методом Золотого сечения.
7. Находим приближенное решение задачи безусловной минимизации численным методом поиска безусловного экстремума – методом Фибоначчи.

**Блок-схема (Метод золотого сечения)**

Ввод

a,b,epsilon

a\_k = a, b\_k = b, a1, b1, y1, z1;

y = a + (3 -) / 2 \* (b - a);

z = a + ( - 1) / 2 \* (b - a);

да

(fabs(a1 - b1) > epsilon)

k = k + 1;

if (func(y) ≤ func(z))

нет

нет

x = (a1 + b1) / 2;

да

a1 = a\_k;

b1 = z;

y1 = a1 + (3 - ) / 2 \* (b1 - a1);

z1 = y;

Вывод

x

конец

a1 = a\_k;

b1 = z;

y1 = a1 + (3 a1 = y;

b1 = b\_k;

y1 = z;

z1 = a1 + (sqrt(5) - 1) / 2 \* (b1 - a1);- sqrt(5)) / 2 \* (b1 - a1);

z1 = y;

y = y1;z = z1;a\_k = a1;b\_k = b1;

**Решение задачи в пакете Maple**

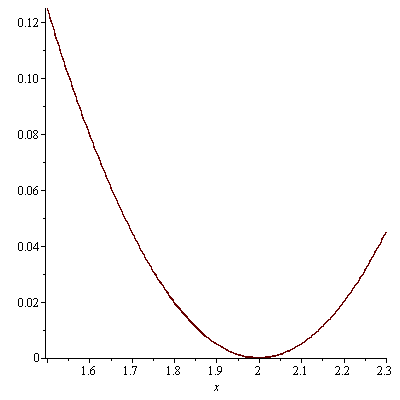
**> **



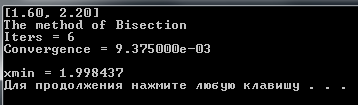
**> **

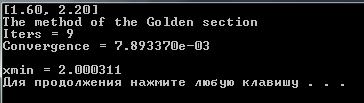


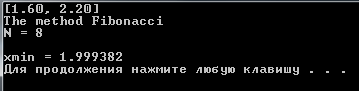
**> **



**Пример выполнения программы**



****

****

**Вывод:** были изучены и выполнены методы дихотомии, золотого сечения и Фибоначчи. Полученные результаты были проверены с результатом, который был получен в Maple.**Листинг**

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <inttypes.h>

#include <math.h>

struct segment {

double a;

double b;

};

typedef struct segment segment\_t;

double fun(double x);

segment\_t methodSven(double x0, double t);

double methodBisection(segment\_t seg, double epsilon);

double methodGoldenSection(segment\_t seg, double epsilon);

uint32\_t findNforFibonacci(segment\_t seg, double l);

double methodFibonacci(segment\_t seg, double l, double epsilon);

uint32\_t main() {

const double x0 = 1.9;

const double epsilon = 1.0e-2;

const double t = 0.3;

const double l = epsilon; // for methodFibonacci

const segment\_t seg = methodSven(x0, t);

printf("[%.2f, %.2f]\n", seg.a, seg.b);

const double xmin = methodFibonacci(seg, l, epsilon);

printf("\nxmin = %f\n", xmin);

system("pause");

return 0;

}

inline double fun(const double x) {

return x < 1.0 ? 1.0 / (sin(x\*x) + 1.0) : 0.5 \* x\*x - 2.0 \* x + 2.0;

}

segment\_t methodSven(const double x0, const double t) {

segment\_t seg = { 0 };

double a = x0 - t;

double xk = x0;

double b = x0 + t;

const double f\_a = fun(a);

double f\_xk = fun(xk);

const double f\_b = fun(b);

if (f\_a <= f\_xk && f\_b <= f\_xk) {

printf("Error! No unimodal\n");

return seg;

} else if (f\_a >= f\_xk && f\_b >= f\_xk) {

seg.a = a;

seg.b = b;

return seg;

}

double delta, xk1;

double \*bound\_xk, \*bound\_xk1;

if (f\_a >= f\_xk && f\_xk >= f\_b) {

xk1 = b;

delta = t;

bound\_xk = &a;

bound\_xk1 = &b;

}

else {

xk1 = a;

delta = -t;

bound\_xk1 = &a;

bound\_xk = &b;

}

double f\_xk1 = fun(xk1);

uint16\_t k = 0;

do {

k++;

xk = xk1;

f\_xk = f\_xk1;

xk1 += (1 << k) \* delta;

f\_xk1 = fun(xk1);

} while (f\_xk1 < f\_xk);

\*bound\_xk = xk;

\*bound\_xk1 = xk1;

seg.a = a;

seg.b = b;

printf("The method Sven\nIters = %" PRIu16 "\n", k);

return seg;

}

double methodBisection(const segment\_t seg, const double epsilon) {

const double inv2 = 0.5;

const double inv4 = 0.25;

double ak = seg.a;

double bk = seg.b;

double ck = (ak + bk) \* inv2;

double f\_ck = fun(ck);

double convergence = fabs(bk - ak);

uint16\_t k = 0;

while (convergence > epsilon) {

const double yk = ak + convergence \* inv4;

const double f\_yk = fun(yk);

if (f\_yk <= f\_ck) {

bk = ck;

ck = yk;

f\_ck = fun(ck);

} else {

const double zk = bk - convergence \* inv4;

const double f\_zk = fun(zk);

if (f\_ck <= f\_zk) {

ak = yk;

bk = zk;

}

else {

ak = ck;

ck = zk;

f\_ck = fun(ck);

}

}

convergence = fabs(bk - ak);

k++;

}

printf("The method of Bisection\nIters = %" PRIu16 "\nConvergence = %e\n", k, convergence);

return (ak + bk) \* inv2;

}

double methodGoldenSection(const segment\_t seg, const double epsilon) {

double ak = seg.a;

double bk = seg.b;

// (3 - sqrt(5))/2 ~ 0.3819660112501051

double yk = ak + 0.3819660112501051 \* (bk - ak);

double zk = ak + bk - yk;

double f\_yk = fun(yk);

double f\_zk = fun(zk);

double convergence;

uint16\_t k = 0;

do {

if (f\_yk <= f\_zk) {

bk = zk;

zk = yk;

f\_zk = f\_yk;

yk = ak + bk - yk;

f\_yk = fun(yk);

} else {

ak = yk;

yk = zk;

f\_yk = f\_zk;

zk = ak + bk - zk;

f\_zk = fun(zk);

}

convergence = fabs(ak - bk);

k++;

} while (convergence > epsilon);

printf("The method of the Golden section\nIters = %" PRIu16 "\nConvergence = %e\n", k, convergence);

return (ak + bk) \* 0.5;

}

uint32\_t findNforFibonacci(const segment\_t seg, const double l) {

uint32\_t N = 1;

uint64\_t F0 = 1, F1 = 2, F2 = 3;

const uint64\_t bound = (uint64\_t)(fabs(seg.b - seg.a) / l);

while (F2 < bound) {

F0 = F1;

F1 = F2;

F2 = F1 + F0;

N++;

}

return N;

}

double methodFibonacci(const segment\_t seg, const double l, const double epsilon) {

double ak = seg.a;

double bk = seg.b;

const uint32\_t N = findNforFibonacci(seg, l); // N >= 1

double \*F = (double \*)malloc((N + 3) \* sizeof(double));

F[0] = F[1] = 1.0; F[2] = 2.0; F[3] = 3.0;

double \*currentF = F + 1;

for (uint32\_t i = 4; i < N + 3; ++i) {

currentF++;

currentF[2] = currentF[1] + currentF[0];

}

double yk = ak + currentF[0] / currentF[2] \* (bk - ak);

double zk = ak + bk - yk;

double f\_yk = fun(yk);

double f\_zk = fun(zk);

uint32\_t k = 0;

while (k < N) {

k++;

currentF--;

if (f\_yk <= f\_zk) {

bk = zk;

zk = yk;

f\_zk = f\_yk;

yk = ak + currentF[0] / currentF[2] \* (bk - ak);

f\_yk = fun(yk);

} else {

ak = yk;

yk = zk;

f\_yk = f\_zk;

zk = ak + currentF[1] / currentF[2] \* (bk - ak);

f\_zk = fun(zk);

}

}

zk += epsilon;

f\_zk = fun(zk);

if (f\_yk <= f\_zk) {

bk = zk;

}

printf("The method Fibonacci\nN = %" PRIu32 "\n", N);

free(F);

return (ak + bk) \* 0.5;

}