Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра теоретической и прикладной информатики

Лабораторная работа №1
по дисциплине «Методы оптимизации»

Факультет: ПМИ

Группа: ПМ-92

Бригада: 7

Студенты: Иванов В., Кутузов И.

Преподаватель: Филиппова Е.В.

Новосибирск

1. Цель работы

Ознакомиться с методами одномерного поиска, используемыми в многомерных методах минимизации функций n переменных. Сравнить различные алгоритмы по эффективности на тестовых примерах.

2. Задание

1. Реализовать методы дихотомии, золотого сечения, исследовать их сходимость и провести сравнение по числу вычислений функции для достижения заданной точности. Построить график зависимости количества вычислений минимизируемой функции от десятичного логарифма задаваемой точности;
2. Реализовать алгоритм поиска интервала, содержащего минимум функции;
3. Реализовать метод Фибоначчи, сравнить его с методами дихотомии и золотого сечения.

$$f(x) = (x-7)^2,\ x \in [-2, 20]$$

3. Код программы

```
import time
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from IPython.display import clear_output
global a, b

def f(x):
    return np.power((x-7), 2)

a = -2
b = 20

def dichotomy(a, b, eps):
    a_prev, b_prev = a, b
    delta = eps / 2
    k, f_count, k_list, length_list = 0, 0, [], []
    df = pd.DataFrame(columns=('x1', 'x2', 'f(x1)', 'f(x2)', 'ai', 'bi', 'len', 'len_prev/len'))
    while abs(b-a) > eps:
        x1 = (a + b - delta) / 2
```

```
        x2 = (a + b + delta) / 2
        a_prev = a
        b_prev = b
        if f(x1) > f(x2):
            a = x1
        else:
            b = x2
        k += 1
        f_count += 2
        length_list.append(abs(b-a)), k_list.append(k)
        df.loc[len(df)] = np.array([x1, x2, f(x1), f(x2), a, b, b-a, (b_prev-a_prev)/(b-a)])
    df.index += 1
    print('Min:', (a+b)/2, '\nNumber of iterations:', k, '\nFunction calculations:', f_count)
    return df
    #plt.plot(k_list, length_list)

def golden_section(xl, xu, extremum_type, eps):
    xl_prev, xu_prev = xl, xu
    error, f_count  = 1, 0
    df = pd.DataFrame(columns=('x1', 'x2', 'f(x1)', 'f(x2)', 'ai', 'bi', 'len', 'len_prev/len'))
    while error >= eps:
        x1, x2 = update(xl, xu)
        flag = is_right(x1, x2)
        #clear_output(wait=True)
        #plot_func(xl, xu, x1, x2)
        #plt.show()
        #time.sleep(0.1)
        xl_prev, xu_prev = xl, xu
        xl, xu, x_opt = find_extremum(xl, xu, x1, x2, extremum_type, flag)
        r = (np.sqrt(5)-1)/2
        error = ((1-r)*(abs((xu-xl)/x_opt)))*100
        f_count += 1
        df.loc[len(df)] = np.array([x1, x2, f(x1), f(x2), xl, xu, xu-xl,
(xu_prev-xl_prev)/(xu-xl)])
    df.index += 1
    print('\nExtremum:', np.mean([x1, x2]), '\nError:', error, '\nFunction calculations:',
f_count)
    return df
    #plot_err(error_list, k_list)

def find_extremum(xl, xu, x1, x2, extremum_type, flag):
    if f(x2) > f(x1) and flag == 1:
        if extremum_type == 'max':
            xu = x1
            x1, x2 = update(xl, xu)
            x_opt = x2
        elif extremum_type == 'min':
            xl = x2
            x1, x2 = update(xl, xu)
            x_opt = x1
    else:
        if extremum_type == 'max':
            xl = x2
            x1, x2 = update(xl, xu)
```

```
            x_opt = x1
        elif extremum_type == 'min':
            xu = x1
            x1, x2 = update(xl, xu)
            x_opt = x2
    return xl, xu, x_opt

def update(xl, xu):
    d = ((np.sqrt(5)-1)/2)*(xu-xl)
    return xl + d, xu - d

def is_right(x1, x2):
    if x1 > x2:
        return 1

def plot_func(xl, xu, x1, x2):
    clear_output(wait=True)
    fig, ax = plt.subplots(figsize=(20, 6))
    ax.plot(np.linspace(a, b, 50), f(np.linspace(a, b, 50)))
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    ax.spines['left'].set_visible(False)
    ax.set_xticks(list(range(a, b+1, 1)))
    ax.set_yticks([])
    ax.tick_params(left=False, bottom=False)
    ax.plot([-2, 20], [0, 0], 'grey')
    ax.plot([xl, xl], [0, f(max([a, b]))], linestyle='dashed', color='grey') # xl line
    ax.plot([xu, xu], [0, f(max([a, b]))], linestyle='dashed', color='grey') # xu line
    ax.plot([x1, x1], [0, f(max([a, b]))], 'k') # x1 line
    ax.plot([x2, x2], [0, f(max([a, b]))], 'k') # x2 line
    ax.plot(x1, f(x1), 'bo', label='x1', zorder=3) # x1 point
    ax.plot([x1, x1], [0, f(x1)], 'k')
    ax.plot(x2, f(x2), 'bo', label='x2', zorder=3) # x2 point
    ax.plot([x2, x2], [0, f(x2)], 'k')

def plot_err(error_list, k_list):
    fig, ax = plt.subplots(figsize=(20, 6))
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.plot(k_list, error_list, 'r')
    ax.scatter(k_list, error_list, zorder=2)
    ax.set_xlabel('iteration')
    ax.set_ylabel('error')

def select_step(x, delta):
    if f(x) > f(x + delta):
        return delta
    return -delta

def interval_search(x_0, delta):
    df = pd.DataFrame(columns=('xi', 'f(xi)'))
    step = select_step(x_0, delta)
    x_k = x_0 + step
```

```
    step = step * 2
    while f(x_k) > f(x_k + step):
        x_k = x_k + step
        step = step * 2
        df.loc[len(df)] = np.array([x_k, f(x_k)])
    if step < 0.0:
        print('Interval: (', x_k + step/2, x_k - step/2, ')')
    else:
        print('Interval: (', x_k - step/2, x_k + step/2, ')')
    df.index += 1
    return df

def fibonacci_number(n):
    return np.power((1 + np.sqrt(5))/2, n) / np.sqrt(5)

def find_min(segment_length, eps):
    n = 0
    while segment_length / eps > fibonacci_number(n):
        n += 1
    return n

def fibonacci(a_0, b_0, eps):
    df = pd.DataFrame(columns=('x1', 'x2', 'f(x1)', 'f(x2)', 'ai', 'bi', 'len', 'len_prev/len'))
    n = find_min(b_0 - a_0, eps)
    f_count, k = 0, 0
    a_k = a_0
    b_k = b_0
    a_kprev, b_kprev = a_k, b_k
    x_k = a_0 + (fibonacci_number(n - 2) / fibonacci_number(n)) * (b_k - a_k)
    y_k = a_0 + (fibonacci_number(n - 1) / fibonacci_number(n)) * (b_k - a_k)
    while (k != n - 3):
        a_kprev, b_kprev = a_k, b_k
        if f(x_k) <= f(y_k):
            b_k = y_k
            y_k = x_k
            x_k = a_k + (fibonacci_number(n - k - 3) / fibonacci_number(n - k - 1)) * (b_k - a_k)
        else:
            a_k = x_k
            x_k = y_k
            y_k = a_k + (fibonacci_number(n - k - 2) / fibonacci_number(n - k - 1)) * (b_k - a_k)
        k += 1
        f_count += 1
        df.loc[len(df)] = np.array([x_k, y_k, f(x_k), f(y_k), a_k, b_k, b_k-a_k,
(b_kprev-a_kprev)/(b_k-a_k)])
    y_k = x_k + eps
    if f(x_k) <= f(y_k):
        b_k = y_k
    else:
        a_k = x_k
    f_count += 1
    print('Min:', (a_k + b_k) / 2, '\nFunction calculations:', f_count)
    df.index += 1
    return df
```

4. Исследование

Метод дихотомии

Min: 7.000000006408101
Function calculations: 58

i	x1	x2	f(x1)	f(x2)	ai	bi	len	len_prev/len
1	9.000000	9.000000	4.000000e+00	4.000000e+00	-2.000000	9.000000	1.100000e+01	2.000000
2	3.500000	3.500000	1.225000e+01	1.225000e+01	3.500000	9.000000	5.500000e+00	2.000000
3	6.250000	6.250000	5.625000e-01	5.625000e-01	6.250000	9.000000	2.750000e+00	2.000000
4	7.625000	7.625000	3.906250e-01	3.906250e-01	6.250000	7.625000	1.375000e+00	2.000000
5	6.937500	6.937500	3.906253e-03	3.906246e-03	6.937500	7.625000	6.875000e-01	2.000000
6	7.281250	7.281250	7.910155e-02	7.910158e-02	6.937500	7.281250	3.437500e-01	2.000000
7	7.109375	7.109375	1.196289e-02	1.196290e-02	6.937500	7.109375	1.718750e-01	2.000000
8	7.023437	7.023438	5.493154e-04	5.493178e-04	6.937500	7.023438	8.593755e-02	1.999999
9	6.980469	6.980469	3.814705e-04	3.814686e-04	6.980469	7.023438	4.296880e-02	1.999999
10	7.001953	7.001953	3.814617e-06	3.814813e-06	6.980469	7.001953	2.148442e-02	1.999998
11	6.991211	6.991211	7.724798e-05	7.724710e-05	6.991211	7.001953	1.074224e-02	1.999995
12	6.996582	6.996582	1.168265e-05	1.168231e-05	6.996582	7.001953	5.371144e-03	1.999991
13	6.999268	6.999268	5.364718e-07	5.363985e-07	6.999268	7.001953	2.685597e-03	1.999981
14	7.000610	7.000610	3.725041e-07	3.725651e-07	6.999268	7.000610	1.342823e-03	1.999963
15	6.999939	6.999939	3.727788e-09	3.721685e-09	6.999939	7.000610	6.714367e-04	1.999926
16	7.000275	7.000275	7.542589e-08	7.545336e-08	6.999939	7.000275	3.357434e-04	1.999851
17	7.000107	7.000107	1.140433e-08	1.141501e-08	6.999939	7.000107	1.678967e-04	1.999702
18	7.000023	7.000023	5.229330e-10	5.252223e-10	6.999939	7.000023	8.397334e-05	1.999405
19	6.999981	6.999981	3.645786e-10	3.626717e-10	6.999981	7.000023	4.201167e-05	1.998810
20	7.000002	7.000002	3.560369e-12	3.751559e-12	6.999981	7.000002	2.103083e-05	1.997623
21	6.999991	6.999991	7.402061e-11	7.316276e-11	6.999991	7.000002	1.054042e-05	1.995256
22	6.999997	6.999997	1.127828e-11	1.094495e-11	6.999997	7.000002	5.295209e-06	1.990558
23	6.999999	6.999999	5.412696e-13	4.701986e-13	6.999999	7.000002	2.672604e-06	1.981292
24	7.000001	7.000001	3.313060e-13	3.913652e-13	6.999999	7.000001	1.361302e-06	1.963270
25	7.000000	7.000000	6.409474e-15	9.035548e-16	7.000000	7.000001	7.056511e-07	1.929143
26	7.000000	7.000000	6.138817e-14	8.866480e-14	7.000000	7.000000	3.778255e-07	1.867664
27	7.000000	7.000000	7.031423e-15	1.791678e-14	7.000000	7.000000	2.139128e-07	1.766260

28	7.000000	7.000000	3.599357e-18	2.693319e-15	7.000000	7.000000	1.319564e-07	1.621087
29	7.000000	7.000000	1.527324e-15	1.192247e-16	7.000000	7.000000	9.097819e-08	1.450418

Метод золотого сечения

Extremum: 7.0000000017267325
Error: 7.654083740496931e-08
Function calculations: 44

i	x1	x2	f(x1)	f(x2)	ai	bi	len	len_prev/len
1	11.596748	6.403252	2.113009e+01	3.561079e-01	-2.000000	11.596748	1.359675e+01	1.618034
2	6.403252	3.193496	3.561079e-01	1.448948e+01	3.193496	11.596748	8.403252e+00	1.618034
3	8.386991	6.403252	1.923744e+00	3.561079e-01	3.193496	8.386991	5.193496e+00	1.618034
4	6.403252	5.177234	3.561079e-01	3.322475e+00	5.177234	8.386991	3.209757e+00	1.618034
5	7.160973	6.403252	2.591232e-02	3.561079e-01	6.403252	8.386991	1.983739e+00	1.618034
6	7.629270	7.160973	3.959810e-01	2.591232e-02	6.403252	7.629270	1.226018e+00	1.618034
7	7.160973	6.871549	2.591232e-02	1.649955e-02	6.403252	7.160973	7.577208e-01	1.618034
8	6.871549	6.692676	1.649955e-02	9.444814e-02	6.692676	7.160973	4.682972e-01	1.618034
9	6.982099	6.871549	3.204309e-04	1.649955e-02	6.871549	7.160973	2.894236e-01	1.618034
10	7.050423	6.982099	2.542485e-03	3.204309e-04	6.871549	7.050423	1.788736e-01	1.618034
11	6.982099	6.939873	3.204309e-04	3.615246e-03	6.939873	7.050423	1.105500e-01	1.618034
12	7.008197	6.982099	6.718631e-05	3.204309e-04	6.982099	7.050423	6.832364e-02	1.618034
13	7.024326	7.008197	5.917421e-04	6.718631e-05	6.982099	7.024326	4.222633e-02	1.618034
14	7.008197	6.998228	6.718631e-05	3.138422e-06	6.982099	7.008197	2.609731e-02	1.618034
15	6.998228	6.992068	3.138422e-06	6.292135e-05	6.992068	7.008197	1.612902e-02	1.618034
16	7.002036	6.998228	4.145243e-06	3.138422e-06	6.992068	7.002036	9.968285e-03	1.618034
17	6.998228	6.995875	3.138422e-06	1.701358e-05	6.995875	7.002036	6.160739e-03	1.618034
18	6.999683	6.998228	1.006196e-07	3.138422e-06	6.998228	7.002036	3.807546e-03	1.618034
19	7.000582	6.999683	3.382978e-07	1.006196e-07	6.998228	7.000582	2.353193e-03	1.618034
20	6.999683	6.999127	1.006196e-07	7.616391e-07	6.999127	7.000582	1.454353e-03	1.618034
21	7.000026	6.999683	6.822696e-10	1.006196e-07	6.999683	7.000582	8.988397e-04	1.618034
22	7.000238	7.000026	5.679049e-08	6.822696e-10	6.999683	7.000238	5.555135e-04	1.618034
23	7.000026	6.999895	6.822696e-10	1.102892e-08	6.999895	7.000238	3.433262e-04	1.618034
24	7.000107	7.000026	1.148511e-08	6.822696e-10	6.999895	7.000107	2.121873e-04	1.618034
25	7.000026	6.999976	6.822696e-10	5.745766e-10	6.999895	7.000026	1.311389e-04	1.618034

26	6.999976	6.999945	5.745766e-10	3.017089e-09	6.999945	7.000026	8.104832e-05	1.618034
27	6.999995	6.999976	2.340058e-11	5.745766e-10	6.999976	7.000026	5.009062e-05	1.618034
28	7.000007	6.999995	4.882343e-11	2.340058e-11	6.999976	7.000007	3.095770e-05	1.618034
29	6.999995	6.999988	2.340058e-11	1.475141e-10	6.999988	7.000007	1.913291e-05	1.618034
30	7.000000	6.999995	1.028782e-13	2.340058e-11	6.999995	7.000007	1.182479e-05	1.618034
31	7.000002	7.000000	6.104399e-12	1.028782e-13	6.999995	7.000002	7.308123e-06	1.618034
32	7.000000	6.999998	1.028782e-13	4.185952e-12	6.999998	7.000002	4.516668e-06	1.618034
33	7.000001	7.000000	5.557620e-13	1.028782e-13	6.999998	7.000001	2.791454e-06	1.618034
34	7.000000	6.999999	1.028782e-13	9.598498e-13	6.999999	7.000001	1.725214e-06	1.618034
35	7.000000	7.000000	7.485966e-15	1.028782e-13	7.000000	7.000001	1.066241e-06	1.618034
36	7.000000	7.000000	1.143973e-13	7.485965e-15	7.000000	7.000000	6.589730e-07	1.618034
37	7.000000	7.000000	7.485966e-15	4.766653e-15	7.000000	7.000000	4.072677e-07	1.618034
38	7.000000	7.000000	4.766653e-15	2.728569e-14	7.000000	7.000000	2.517053e-07	1.618034
39	7.000000	7.000000	9.257121e-17	4.766653e-15	7.000000	7.000000	1.555624e-07	1.618034
40	7.000000	7.000000	7.345138e-16	9.257120e-17	7.000000	7.000000	9.614287e-08	1.618034
41	7.000000	7.000000	9.257120e-17	1.044430e-15	7.000000	7.000000	5.941956e-08	1.618034
42	7.000000	7.000000	1.940986e-17	9.257121e-17	7.000000	7.000000	3.672331e-08	1.618034
43	7.000000	7.000000	1.709519e-16	1.940985e-17	7.000000	7.000000	2.269625e-08	1.618034
44	7.000000	7.000000	1.940986e-17	9.066783e-19	7.000000	7.000000	1.402706e-08	1.618034

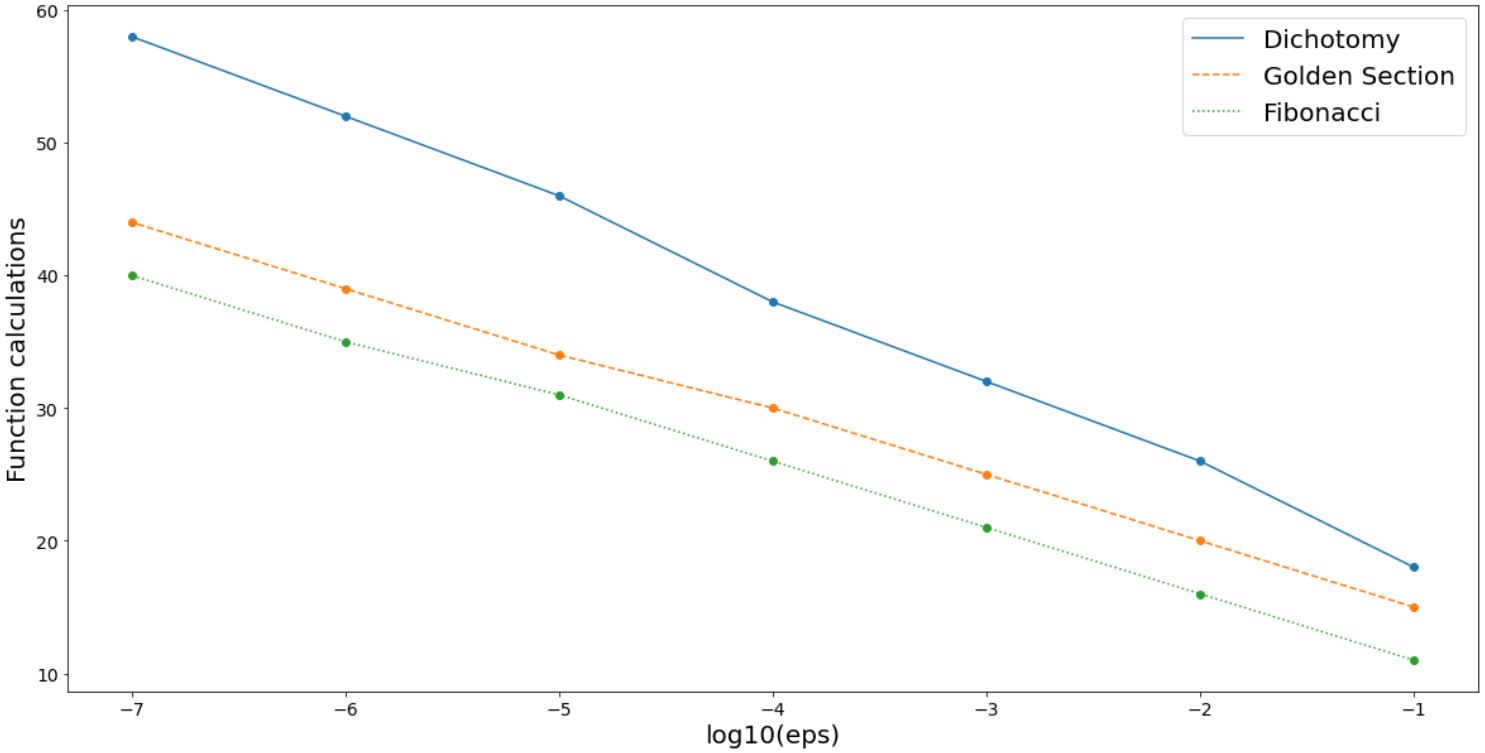
Метод Фибоначчи

Min: 7.000000010668826
Function calculations: 40

i	x1	x2	f(x1)	f(x2)	ai	bi	len	len_prev/len
1	3.193496	6.403252	1.448948e+01	3.561079e-01	-2.000000	11.596748	1.359675e+01	1.618034
2	6.403252	8.386991	3.561079e-01	1.923744e+00	3.193496	11.596748	8.403252e+00	1.618034
3	5.177234	6.403252	3.322475e+00	3.561079e-01	3.193496	8.386991	5.193496e+00	1.618034
4	6.403252	7.160973	3.561079e-01	2.591232e-02	5.177234	8.386991	3.209757e+00	1.618034
5	7.160973	7.629270	2.591232e-02	3.959810e-01	6.403252	8.386991	1.983739e+00	1.618034
6	6.871549	7.160973	1.649955e-02	2.591232e-02	6.403252	7.629270	1.226018e+00	1.618034
7	6.692676	6.871549	9.444814e-02	1.649955e-02	6.403252	7.160973	7.577208e-01	1.618034
8	6.871549	6.982099	1.649955e-02	3.204309e-04	6.692676	7.160973	4.682972e-01	1.618034
9	6.982099	7.050423	3.204309e-04	2.542485e-03	6.871549	7.160973	2.894236e-01	1.618034

10	6.939873	6.982099	3.615246e-03	3.204309e-04	6.871549	7.050423	1.788736e-01	1.618034
11	6.982099	7.008197	3.204309e-04	6.718631e-05	6.939873	7.050423	1.105500e-01	1.618034
12	7.008197	7.024326	6.718631e-05	5.917421e-04	6.982099	7.050423	6.832364e-02	1.618034
13	6.998228	7.008197	3.138422e-06	6.718631e-05	6.982099	7.024326	4.222633e-02	1.618034
14	6.992068	6.998228	6.292135e-05	3.138422e-06	6.982099	7.008197	2.609731e-02	1.618034
15	6.998228	7.002036	3.138422e-06	4.145243e-06	6.992068	7.008197	1.612902e-02	1.618034
16	6.995875	6.998228	1.701358e-05	3.138422e-06	6.992068	7.002036	9.968285e-03	1.618034
17	6.998228	6.999683	3.138422e-06	1.006196e-07	6.995875	7.002036	6.160739e-03	1.618034
18	6.999683	7.000582	1.006196e-07	3.382978e-07	6.998228	7.002036	3.807546e-03	1.618034
19	6.999127	6.999683	7.616391e-07	1.006196e-07	6.998228	7.000582	2.353193e-03	1.618034
20	6.999683	7.000026	1.006196e-07	6.822696e-10	6.999127	7.000582	1.454353e-03	1.618034
21	7.000026	7.000238	6.822696e-10	5.679049e-08	6.999683	7.000582	8.988397e-04	1.618034
22	6.999895	7.000026	1.102892e-08	6.822696e-10	6.999683	7.000238	5.555135e-04	1.618034
23	7.000026	7.000107	6.822696e-10	1.148511e-08	6.999895	7.000238	3.433262e-04	1.618034
24	6.999976	7.000026	5.745766e-10	6.822696e-10	6.999895	7.000107	2.121873e-04	1.618034
25	6.999945	6.999976	3.017089e-09	5.745766e-10	6.999895	7.000026	1.311389e-04	1.618034
26	6.999976	6.999995	5.745766e-10	2.340058e-11	6.999945	7.000026	8.104832e-05	1.618034
27	6.999995	7.000007	2.340058e-11	4.882343e-11	6.999976	7.000026	5.009062e-05	1.618034
28	6.999988	6.999995	1.475141e-10	2.340058e-11	6.999976	7.000007	3.095770e-05	1.618034
29	6.999995	7.000000	2.340058e-11	1.028782e-13	6.999988	7.000007	1.913291e-05	1.618034
30	7.000000	7.000002	1.028782e-13	6.104399e-12	6.999995	7.000007	1.182479e-05	1.618034
31	6.999998	7.000000	4.185952e-12	1.028782e-13	6.999995	7.000002	7.308123e-06	1.618034
32	7.000000	7.000001	1.028782e-13	5.557620e-13	6.999998	7.000002	4.516668e-06	1.618034
33	6.999999	7.000000	9.598498e-13	1.028782e-13	6.999998	7.000001	2.791454e-06	1.618034
34	7.000000	7.000000	1.028782e-13	7.485965e-15	6.999999	7.000001	1.725214e-06	1.618034
35	7.000000	7.000000	7.485965e-15	1.143973e-13	7.000000	7.000001	1.066241e-06	1.618034
36	7.000000	7.000000	4.766653e-15	7.485965e-15	7.000000	7.000000	6.589730e-07	1.618034
37	7.000000	7.000000	2.728569e-14	4.766653e-15	7.000000	7.000000	4.072677e-07	1.618034
38	7.000000	7.000000	4.766653e-15	9.257123e-17	7.000000	7.000000	2.517053e-07	1.618034
39	7.000000	7.000000	9.257123e-17	7.345137e-16	7.000000	7.000000	1.555624e-07	1.618034

График



Алгоритм поиска интервала

```
interval_search(2, 0.001)
```

i	xi	f(xi)
1	2.003	24.970009
2	2.007	24.930049
3	2.015	24.850225
4	2.031	24.690961
5	2.063	24.373969
6	2.127	23.746129
7	2.255	22.515025
8	2.511	20.151121
9	3.023	15.816529
10	4.047	8.720209
11	**6.095**	0.819025
12	10.191	10.182481

Interval: (4.048, 10.191)

5. Выводы

Результаты исследования показали, что метод Фибоначчи является наиболее эффективным с точки зрения количества вычислений целевой функции.
При этом метод дихотомии лучше с точки зрения точности.