**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 № 2 на тему:   
**«Анализ алгоритмов и программ.»**

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-312, Серафимович Г. П.

«10» октября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ Серафимович Г. П./

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

«10» октября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ И.А. Моисеев /

**Основная часть**

**Цель работы:**

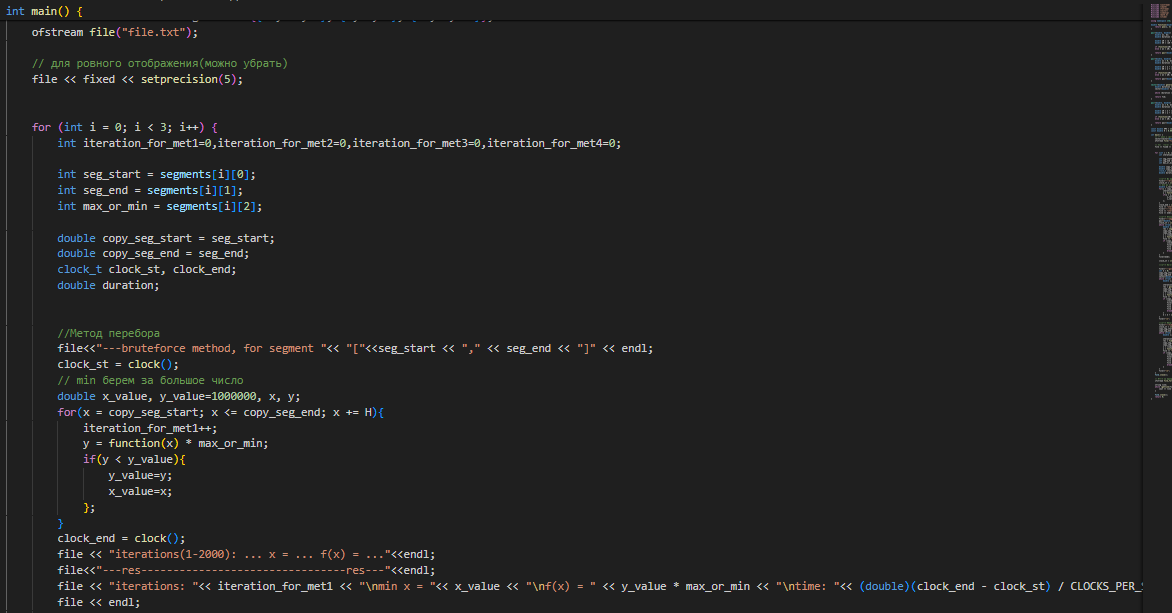
Цель данной работы — разработка и реализация алгоритмов для нахождения минимума(максимума) заданной функции f(x) = x^3 - x + e^{-x} на заданных интервалах. Программа использует методы перебора, деления отрезка пополам, метод золотого сечения и метод Фибоначчи для поиска экстремумов функции.

**Результаты выполнения работы:**

В данной работе были реализованы следующие методы:

1. **Метод пассивного поиска (перебором)** заключается в вычислении функции на равномерно распределённых точках интервала с шагом “Step”. На основе вычисленных значений определяется экстремум функции.
2. **Метод деления отрезка пополам,** в этом методе интервал поиска делится пополам и выбираются новые подотрезки для продолжения поиска. Этот процесс продолжается до тех пор, пока длина интервала не станет меньше заданной точности “Accurancy”.
3. **Метод золотого сечения,** в нём интервал поиска уменьшается по отношению к золотому сечению. В процессе поиска, значения функции в определённых точках сравниваются, и интервал поиска сокращается в зависимости от этих значений.
4. **Метод Фибоначчи**, этот метод основан на использовании чисел Фибоначчи для определения новых точек и уменьшения интервала поиска. Этот метод требует предварительного расчёта чисел Фибоначчи до заданного количества итераций.

Были реализованы все четыре метода, реализация и вывод программы представлен ниже.



**Рисунок 1.1 Метод пассивного поиска**

---bruteforce method, for segment [-5,-3]

iterations(1-2000): ... x = ... f(x) = ...

---res--------------------------------res---

iterations: 2000

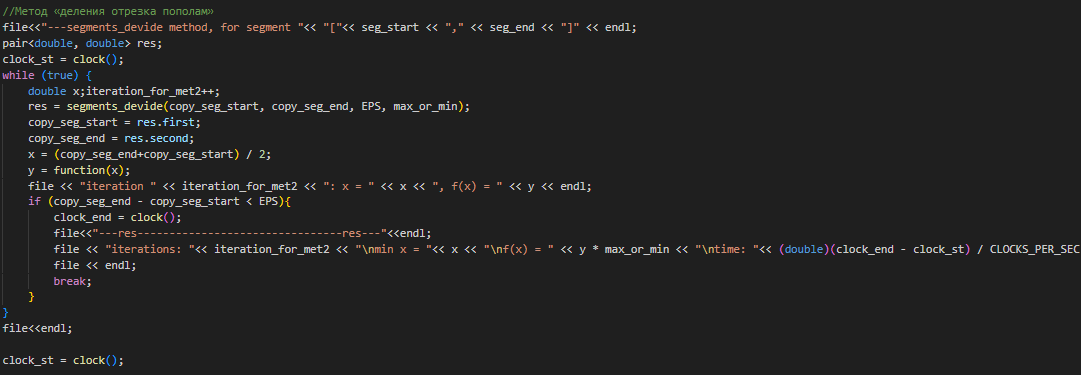
min x = -3.67900

f(x) = -6.50965

time: 0.00019

**Рисунок 1.2 Вывод для метода пассивного поиска**

Метод пассивного поиска реализует простой перебор значений функции на заданном интервале от a до b с шагом epsilon. Он сохраняет все вычисленные значения в векторе и выводит их на экран. В конце метода выводится значение x, при котором функция достигает минимума, а также само значение функции в этой точке.



**Рисунок 2.1 Метод деления отрезка пополам**

---segments\_devide method, for segment [-5,-3]

iteration 1: x = -3.60000, f(x) = -6.45777

iteration 2: x = -3.84000, f(x) = -6.25763

iteration 3: x = -3.69600, f(x) = -6.50705

iteration 4: x = -3.60960, f(x) = -6.46937

iteration 5: x = -3.66144, f(x) = -6.50701

iteration 6: x = -3.69254, f(x) = -6.50800

iteration 7: x = -3.67388, f(x) = -6.50943

iteration 8: x = -3.68508, f(x) = -6.50931

iteration 9: x = -3.67836, f(x) = -6.50965

iteration 10: x = -3.68239, f(x) = -6.50954

iteration 11: x = -3.67997, f(x) = -6.50963

iteration 12: x = -3.67852, f(x) = -6.50965

iteration 13: x = -3.67939, f(x) = -6.50965

iteration 14: x = -3.67887, f(x) = -6.50964

iteration 15: x = -3.67856, f(x) = -6.50964

iteration 16: x = -3.67837, f(x) = -6.50964

iteration 17: x = -3.67826, f(x) = -6.50964

iteration 18: x = -3.67819, f(x) = -6.50964

iteration 19: x = -3.67823, f(x) = -6.50964

iteration 20: x = -3.67820, f(x) = -6.50964

iteration 21: x = -3.67819, f(x) = -6.50964

iteration 22: x = -3.67818, f(x) = -6.50964

iteration 23: x = -3.67819, f(x) = -6.50964

iteration 24: x = -3.67819, f(x) = -6.50965

iteration 25: x = -3.67819, f(x) = -6.50964

iteration 26: x = -3.67819, f(x) = -6.50964

iteration 27: x = -3.67819, f(x) = -6.50964

iteration 28: x = -3.67819, f(x) = -6.50965

iteration 29: x = -3.67819, f(x) = -6.50965

---res--------------------------------res---

iterations: 29

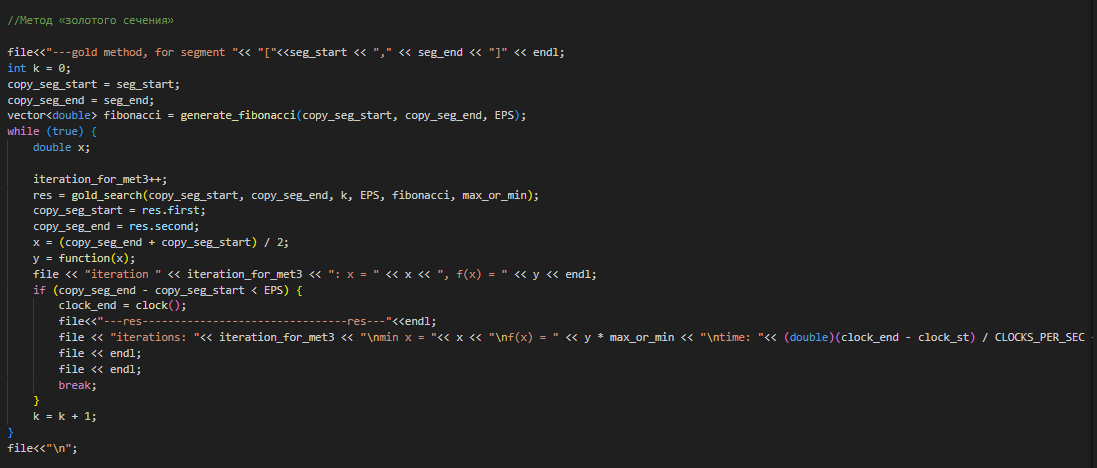
min x = -3.67819

f(x) = -6.50965

time: 0.00024

**Рисунок 2.2 Вывод метода деления отрезка пополам**

Метод деления отрезка пополам использует подход деления интервала на три части для нахождения минимума функции. Он последовательно сужает интервал, основываясь на значениях функции в средних точках. Метод записывает шаги в вектор и выводит их в конце выполнения, а также количество итераций, необходимых для достижения заданной точности.



**Рисунок 3.1 Метод для золотого сечения**

---gold method, for segment [-5,-3]

iteration 1: x = -3.61803, f(x) = -6.47842

iteration 2: x = -3.85410, f(x) = -6.20890

iteration 3: x = -3.70820, f(x) = -6.50198

iteration 4: x = -3.61803, f(x) = -6.47842

iteration 5: x = -3.67376, f(x) = -6.50942

iteration 6: x = -3.70820, f(x) = -6.50198

iteration 7: x = -3.68692, f(x) = -6.50908

iteration 8: x = -3.67376, f(x) = -6.50942

iteration 9: x = -3.68189, f(x) = -6.50957

iteration 10: x = -3.67687, f(x) = -6.50961

iteration 11: x = -3.67997, f(x) = -6.50963

iteration 12: x = -3.67805, f(x) = -6.50964

iteration 13: x = -3.67924, f(x) = -6.50965

iteration 14: x = -3.67851, f(x) = -6.50965

iteration 15: x = -3.67896, f(x) = -6.50965

iteration 16: x = -3.67868, f(x) = -6.50965

iteration 17: x = -3.67885, f(x) = -6.50965

iteration 18: x = -3.67875, f(x) = -6.50965

iteration 19: x = -3.67881, f(x) = -6.50965

iteration 20: x = -3.67877, f(x) = -6.50965

iteration 21: x = -3.67880, f(x) = -6.50965

iteration 22: x = -3.67878, f(x) = -6.50965

iteration 23: x = -3.67879, f(x) = -6.50965

iteration 24: x = -3.67878, f(x) = -6.50965

iteration 25: x = -3.67879, f(x) = -6.50965

iteration 26: x = -3.67879, f(x) = -6.50965

iteration 27: x = -3.67879, f(x) = -6.50965

iteration 28: x = -3.67879, f(x) = -6.50965

iteration 29: x = -3.67879, f(x) = -6.50965

iteration 30: x = -3.67879, f(x) = -6.50965

iteration 31: x = -3.67879, f(x) = -6.50965

---res--------------------------------res---

iterations: 31

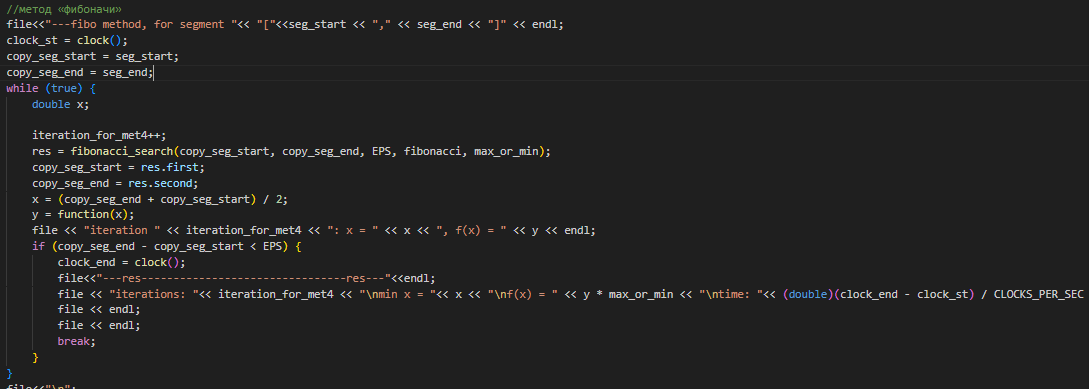
min x = -3.67879

f(x) = -6.50965

time: 0.00026

**Рисунок 3.2 Вывод метода для золотого сечения**

Метод золотого сечения использует свойство золотого сечения для нахождения минимума функции. Он последовательно обновляет границы интервала на основе значений функции в двух точках, которые определяются пропорцией золотого сечения. Также сохраняются шаги и выводится количество итераций.



**Рисунок 4.1 Метод для фибоначи**

---fibo method, for segment [-5,-3]

iteration 1: x = -3.61803, f(x) = -6.47842

iteration 2: x = -3.85410, f(x) = -6.20890

iteration 3: x = -3.70820, f(x) = -6.50198

iteration 4: x = -3.61803, f(x) = -6.47842

iteration 5: x = -3.67376, f(x) = -6.50942

iteration 6: x = -3.70820, f(x) = -6.50198

iteration 7: x = -3.68692, f(x) = -6.50908

iteration 8: x = -3.67376, f(x) = -6.50942

iteration 9: x = -3.68189, f(x) = -6.50957

iteration 10: x = -3.67687, f(x) = -6.50961

iteration 11: x = -3.67997, f(x) = -6.50963

iteration 12: x = -3.67805, f(x) = -6.50964

iteration 13: x = -3.67924, f(x) = -6.50965

iteration 14: x = -3.67851, f(x) = -6.50965

iteration 15: x = -3.67896, f(x) = -6.50965

iteration 16: x = -3.67868, f(x) = -6.50965

iteration 17: x = -3.67885, f(x) = -6.50965

iteration 18: x = -3.67875, f(x) = -6.50965

iteration 19: x = -3.67881, f(x) = -6.50965

iteration 20: x = -3.67877, f(x) = -6.50965

iteration 21: x = -3.67880, f(x) = -6.50964

iteration 22: x = -3.67878, f(x) = -6.50965

iteration 23: x = -3.67877, f(x) = -6.50965

iteration 24: x = -3.67878, f(x) = -6.50965

iteration 25: x = -3.67878, f(x) = -6.50965

iteration 26: x = -3.67878, f(x) = -6.50965

iteration 27: x = -3.67878, f(x) = -6.50964

iteration 28: x = -3.67878, f(x) = -6.50965

iteration 29: x = -3.67878, f(x) = -6.50965

iteration 30: x = -3.67878, f(x) = -6.50965

iteration 31: x = -3.67878, f(x) = -6.50965

---res--------------------------------res---

iterations: 31

min x = -3.67878

f(x) = -6.50965

time: 0.00025

**Рисунок 4.2 Вывод метода для фибоначи**

Метод Фибоначчи использует последовательность Фибоначчи для нахождения минимума функции. Он вычисляет точки на интервале на основе соотношений Фибоначчи и постепенно сужает интервал до тех пор, пока не будет достигнута заданная точность. Как и в предыдущих методах, шаги сохраняются и выводятся.

Также прикрепляю вывод для других сегментов([0,3],[-3,0])

---bruteforce method, for segment [0,3]

iterations(1-2000): ... x = ... f(x) = ...

---res--------------------------------res---

iterations: 3001

min x = 0.70600

f(x) = 0.13951

time: 0.00022

---segments\_devide method, for segment [0,3]

iteration 1: x = 0.90000, f(x) = 0.23557

iteration 2: x = 0.54000, f(x) = 0.20021

iteration 3: x = 0.75600, f(x) = 0.14562

iteration 4: x = 0.62640, f(x) = 0.15390

iteration 5: x = 0.70416, f(x) = 0.13952

iteration 6: x = 0.75082, f(x) = 0.14442

iteration 7: x = 0.72282, f(x) = 0.14021

iteration 8: x = 0.70603, f(x) = 0.13951

iteration 9: x = 0.69595, f(x) = 0.13973

iteration 10: x = 0.70200, f(x) = 0.13954

iteration 11: x = 0.70562, f(x) = 0.13951

iteration 12: x = 0.70780, f(x) = 0.13952

iteration 13: x = 0.70649, f(x) = 0.13951

iteration 14: x = 0.70571, f(x) = 0.13951

iteration 15: x = 0.70524, f(x) = 0.13951

iteration 16: x = 0.70552, f(x) = 0.13951

iteration 17: x = 0.70569, f(x) = 0.13951

iteration 18: x = 0.70559, f(x) = 0.13951

iteration 19: x = 0.70553, f(x) = 0.13951

iteration 20: x = 0.70557, f(x) = 0.13951

iteration 21: x = 0.70554, f(x) = 0.13951

iteration 22: x = 0.70556, f(x) = 0.13951

iteration 23: x = 0.70556, f(x) = 0.13951

iteration 24: x = 0.70556, f(x) = 0.13951

iteration 25: x = 0.70556, f(x) = 0.13951

iteration 26: x = 0.70556, f(x) = 0.13951

iteration 27: x = 0.70557, f(x) = 0.13951

iteration 28: x = 0.70557, f(x) = 0.13951

iteration 29: x = 0.70557, f(x) = 0.13951

iteration 30: x = 0.70557, f(x) = 0.13951

---res--------------------------------res---

iterations: 30

min x = 0.70557

f(x) = 0.13951

time: 0.00024

---gold method, for segment [0,3]

iteration 1: x = 0.92705, f(x) = 0.26540

iteration 2: x = 0.57295, f(x) = 0.17899

iteration 3: x = 0.79180, f(x) = 0.15764

iteration 4: x = 0.65654, f(x) = 0.14510

iteration 5: x = 0.74013, f(x) = 0.14236

iteration 6: x = 0.68847, f(x) = 0.14020

iteration 7: x = 0.72040, f(x) = 0.14003

iteration 8: x = 0.70067, f(x) = 0.13957

iteration 9: x = 0.71286, f(x) = 0.13963

iteration 10: x = 0.70532, f(x) = 0.13951

iteration 11: x = 0.70998, f(x) = 0.13955

iteration 12: x = 0.70710, f(x) = 0.13952

iteration 13: x = 0.70532, f(x) = 0.13951

iteration 14: x = 0.70642, f(x) = 0.13951

iteration 15: x = 0.70574, f(x) = 0.13951

iteration 16: x = 0.70532, f(x) = 0.13951

iteration 17: x = 0.70558, f(x) = 0.13951

iteration 18: x = 0.70574, f(x) = 0.13951

iteration 19: x = 0.70565, f(x) = 0.13951

iteration 20: x = 0.70558, f(x) = 0.13951

iteration 21: x = 0.70562, f(x) = 0.13951

iteration 22: x = 0.70560, f(x) = 0.13951

iteration 23: x = 0.70561, f(x) = 0.13951

iteration 24: x = 0.70562, f(x) = 0.13951

iteration 25: x = 0.70562, f(x) = 0.13951

iteration 26: x = 0.70562, f(x) = 0.13951

iteration 27: x = 0.70562, f(x) = 0.13951

iteration 28: x = 0.70562, f(x) = 0.13951

iteration 29: x = 0.70562, f(x) = 0.13951

iteration 30: x = 0.70562, f(x) = 0.13951

iteration 31: x = 0.70562, f(x) = 0.13951

iteration 32: x = 0.70562, f(x) = 0.13951

---res--------------------------------res---

iterations: 32

min x = 0.70562

f(x) = 0.13951

time: 0.00027

---fibo method, for segment [0,3]

iteration 1: x = 0.92705, f(x) = 0.26540

iteration 2: x = 0.57295, f(x) = 0.17899

iteration 3: x = 0.79180, f(x) = 0.15764

iteration 4: x = 0.65654, f(x) = 0.14510

iteration 5: x = 0.74013, f(x) = 0.14236

iteration 6: x = 0.68847, f(x) = 0.14020

iteration 7: x = 0.72040, f(x) = 0.14003

iteration 8: x = 0.70067, f(x) = 0.13957

iteration 9: x = 0.71286, f(x) = 0.13963

iteration 10: x = 0.70532, f(x) = 0.13951

iteration 11: x = 0.70998, f(x) = 0.13955

iteration 12: x = 0.70710, f(x) = 0.13952

iteration 13: x = 0.70532, f(x) = 0.13951

iteration 14: x = 0.70642, f(x) = 0.13951

iteration 15: x = 0.70574, f(x) = 0.13951

iteration 16: x = 0.70532, f(x) = 0.13951

iteration 17: x = 0.70558, f(x) = 0.13951

iteration 18: x = 0.70574, f(x) = 0.13951

iteration 19: x = 0.70565, f(x) = 0.13951

iteration 20: x = 0.70558, f(x) = 0.13951

iteration 21: x = 0.70562, f(x) = 0.13951

iteration 22: x = 0.70560, f(x) = 0.13951

iteration 23: x = 0.70558, f(x) = 0.13951

iteration 24: x = 0.70559, f(x) = 0.13951

iteration 25: x = 0.70559, f(x) = 0.13951

iteration 26: x = 0.70559, f(x) = 0.13951

iteration 27: x = 0.70559, f(x) = 0.13951

iteration 28: x = 0.70559, f(x) = 0.13951

iteration 29: x = 0.70559, f(x) = 0.13951

iteration 30: x = 0.70559, f(x) = 0.13951

iteration 31: x = 0.70559, f(x) = 0.13951

---res--------------------------------res---

iterations: 31

min x = 0.70559

f(x) = 0.13951

time: 0.00025

---bruteforce method, for segment [-3,0]

iterations(1-2000): ... x = ... f(x) = ...

---res--------------------------------res---

iterations: 2000

max x = -1.20000

f(x) = 2.79212

time: 0.00025

---segments\_devide method, for segment [-3,0]

iteration 1: x = -0.90000, f(x) = 2.63060

iteration 2: x = -1.26000, f(x) = 2.78505

iteration 3: x = -1.04400, f(x) = 2.74666

iteration 4: x = -1.17360, f(x) = 2.79077

iteration 5: x = -1.25136, f(x) = 2.78695

iteration 6: x = -1.20470, f(x) = 2.79207

iteration 7: x = -1.17671, f(x) = 2.79107

iteration 8: x = -1.19351, f(x) = 2.79203

iteration 9: x = -1.20358, f(x) = 2.79209

iteration 10: x = -1.19754, f(x) = 2.79210

iteration 11: x = -1.20117, f(x) = 2.79211

iteration 12: x = -1.19899, f(x) = 2.79211

iteration 13: x = -1.20029, f(x) = 2.79212

iteration 14: x = -1.19951, f(x) = 2.79212

iteration 15: x = -1.19998, f(x) = 2.79212

iteration 16: x = -1.19970, f(x) = 2.79212

iteration 17: x = -1.19987, f(x) = 2.79212

iteration 18: x = -1.19997, f(x) = 2.79212

iteration 19: x = -1.20003, f(x) = 2.79212

iteration 20: x = -1.20007, f(x) = 2.79212

iteration 21: x = -1.20005, f(x) = 2.79212

iteration 22: x = -1.20003, f(x) = 2.79212

iteration 23: x = -1.20002, f(x) = 2.79212

iteration 24: x = -1.20002, f(x) = 2.79212

iteration 25: x = -1.20002, f(x) = 2.79212

iteration 26: x = -1.20002, f(x) = 2.79212

iteration 27: x = -1.20002, f(x) = 2.79212

iteration 28: x = -1.20002, f(x) = 2.79212

iteration 29: x = -1.20002, f(x) = 2.79212

iteration 30: x = -1.20002, f(x) = 2.79212

---res--------------------------------res---

iterations: 30

max x = -1.20002

f(x) = 2.79212

time: 0.00022

---gold method, for segment [-3,0]

iteration 1: x = -0.92705, f(x) = 2.65737

iteration 2: x = -1.28115, f(x) = 2.77912

iteration 3: x = -1.06231, f(x) = 2.75653

iteration 4: x = -1.19756, f(x) = 2.79211

iteration 5: x = -1.28115, f(x) = 2.77912

iteration 6: x = -1.22949, f(x) = 2.79042

iteration 7: x = -1.19756, f(x) = 2.79211

iteration 8: x = -1.21729, f(x) = 2.79154

iteration 9: x = -1.20510, f(x) = 2.79207

iteration 10: x = -1.19756, f(x) = 2.79211

iteration 11: x = -1.20222, f(x) = 2.79211

iteration 12: x = -1.19934, f(x) = 2.79212

iteration 13: x = -1.20112, f(x) = 2.79211

iteration 14: x = -1.20002, f(x) = 2.79212

iteration 15: x = -1.20070, f(x) = 2.79212

iteration 16: x = -1.20028, f(x) = 2.79212

iteration 17: x = -1.20002, f(x) = 2.79212

iteration 18: x = -1.20018, f(x) = 2.79212

iteration 19: x = -1.20008, f(x) = 2.79212

iteration 20: x = -1.20014, f(x) = 2.79212

iteration 21: x = -1.20018, f(x) = 2.79212

iteration 22: x = -1.20016, f(x) = 2.79212

iteration 23: x = -1.20017, f(x) = 2.79212

iteration 24: x = -1.20016, f(x) = 2.79212

iteration 25: x = -1.20017, f(x) = 2.79212

iteration 26: x = -1.20016, f(x) = 2.79212

iteration 27: x = -1.20017, f(x) = 2.79212

iteration 28: x = -1.20017, f(x) = 2.79212

iteration 29: x = -1.20017, f(x) = 2.79212

iteration 30: x = -1.20017, f(x) = 2.79212

iteration 31: x = -1.20017, f(x) = 2.79212

iteration 32: x = -1.20017, f(x) = 2.79212

---res--------------------------------res---

iterations: 32

max x = -1.20017

f(x) = 2.79212

time: 0.00027

---fibo method, for segment [-3,0]

iteration 1: x = -0.92705, f(x) = 2.65737

iteration 2: x = -1.28115, f(x) = 2.77912

iteration 3: x = -1.06231, f(x) = 2.75653

iteration 4: x = -1.19756, f(x) = 2.79211

iteration 5: x = -1.28115, f(x) = 2.77912

iteration 6: x = -1.22949, f(x) = 2.79042

iteration 7: x = -1.19756, f(x) = 2.79211

iteration 8: x = -1.21729, f(x) = 2.79154

iteration 9: x = -1.20510, f(x) = 2.79207

iteration 10: x = -1.19756, f(x) = 2.79211

iteration 11: x = -1.20222, f(x) = 2.79211

iteration 12: x = -1.19934, f(x) = 2.79212

iteration 13: x = -1.20112, f(x) = 2.79211

iteration 14: x = -1.20002, f(x) = 2.79212

iteration 15: x = -1.20070, f(x) = 2.79212

iteration 16: x = -1.20028, f(x) = 2.79212

iteration 17: x = -1.20002, f(x) = 2.79212

iteration 18: x = -1.20018, f(x) = 2.79212

iteration 19: x = -1.20008, f(x) = 2.79212

iteration 20: x = -1.20014, f(x) = 2.79212

iteration 21: x = -1.20018, f(x) = 2.79212

iteration 22: x = -1.20016, f(x) = 2.79212

iteration 23: x = -1.20014, f(x) = 2.79212

iteration 24: x = -1.20015, f(x) = 2.79212

iteration 25: x = -1.20015, f(x) = 2.79212

iteration 26: x = -1.20015, f(x) = 2.79212

iteration 27: x = -1.20015, f(x) = 2.79212

iteration 28: x = -1.20015, f(x) = 2.79212

iteration 29: x = -1.20015, f(x) = 2.79212

iteration 30: x = -1.20015, f(x) = 2.79212

iteration 31: x = -1.20015, f(x) = 2.79212

---res--------------------------------res---

iterations: 31

max x = -1.20015

f(x) = 2.79212

time: 0.00024

**Выводы:**

Я многое понял из реализации данной практической работы. В результате реализации различных методов поиска минимума(максимума) функции можно сделать следующие выводы: 1. Каждый из методов имеет свои особенности и может быть более или менее эффективным в зависимости от конкретной функции и заданного интервала. 2. Все методы позволяют достичь заданной точности, однако методы деления отрезка пополам и золотого сечения обычно работают быстрее благодаря более эффективному сужению интервала. 3. Пассивный поиск является самым простым в реализации, но неэффективен для больших интервалов или малых значений epsilon.**Приложение**

Листинг программы:

#include <iostream>

#include <cmath>

#include <fstream>

#include <vector>

#include <iomanip>

#include <math.h>

#include <ctime>

using namespace std;

double function(double x){

    return pow(x, 3) - x + expf(-x);

}

pair<double, double> segments\_devide(double a, double b, double eps, int max\_or\_min) {

    double a1, b1;

    double duration = (b - a) / 10;

    double a0 = (a + b) / 2 - duration;

    double b0 = (a0 + 2 \* duration);

    if (function(a0) \* max\_or\_min <= function(b0) \* max\_or\_min) { a1 = a; b1 = b0; }

    else { a1 = a0; b1 = b; }

    return pair<double, double>(a1, b1);

}

pair<double, double> gold\_search(double a, double b, int k, double eps, vector<double> fibo, int max\_or\_min) {

    double a1 = 0, b1 = 0;

    double duration = b - a;

    double a0 = a + (fibo[fibo.size() - k - 2 - 1]/fibo[fibo.size() - k - 1]) \* duration;

    double b0 = a + (fibo[fibo.size() - k - 1 - 1]/fibo[fibo.size() - k - 1]) \* duration;

    if (function(a0) \* max\_or\_min <= function(b0) \* max\_or\_min) { a1 = a; b1 = b0; }

    else { a1 = a0; b1 = b; }

    return pair<double, double>{a1, b1};

}

vector<double> generate\_fibonacci(double a,double b, double eps){

    double duration = b - a;

    vector<double> fib{1, 1};

    while (duration / (fib.back() + fib[fib.size() - 1 - 1]) > eps){ fib.insert(fib.end(), 1, fib.back() + fib[fib.size() - 2]); }

    return fib;

}

pair<double, double> fibonacci\_search(double a, double b, double eps, vector<double> fibo, int max\_or\_min) {

    double a1 = 0, b1 = 0;

    double duration = b - a;

    double a0 = a + (2 \* duration)/(3 + sqrt(5));

    double b0 = a + (2 \* duration)/(1 + sqrt(5));

    if (function(a0) \* max\_or\_min <= function(b0) \* max\_or\_min) { a1 = a; b1 = b0; }

    else { a1 = a0; b1 = b; }

    return pair<double, double>{a1, b1};

}

const double EPS = 1e-6;

const double H = 0.001;

int main() {

    // отрезки и макс(мин)

    vector<vector<double>> segments = {{-5, -3, 1}, {0, 3, 1}, {-3, 0, -1}};

    ofstream file("file.txt");

    // для ровного отображения(можно убрать)

    file << fixed << setprecision(5);

    for (int i = 0; i < 3; i++) {

        int iteration\_for\_met1=0,iteration\_for\_met2=0,iteration\_for\_met3=0,iteration\_for\_met4=0;

        int seg\_start = segments[i][0];

        int seg\_end = segments[i][1];

        int max\_or\_min = segments[i][2];

        double copy\_seg\_start = seg\_start;

        double copy\_seg\_end = seg\_end;

        clock\_t clock\_st, clock\_end;

        double duration;

        //Метод перебора

        file<<"---bruteforce method, for segment "<< "["<<seg\_start << "," << seg\_end << "]" << endl;

        clock\_st = clock();

        // min берем за большое число

        double x\_value, y\_value=1000000, x, y;

        for(x = copy\_seg\_start; x <= copy\_seg\_end; x += H){

            iteration\_for\_met1++;

            y = function(x) \* max\_or\_min;

            if(y < y\_value){

                y\_value=y;

                x\_value=x;

            };

        }

        clock\_end = clock();

        file << "iterations(1-2000): ... x = ... f(x) = ..."<<endl;

        file<<"---res--------------------------------res---"<<endl;

        file << "iterations: "<< iteration\_for\_met1 << "\nmin x = "<< x\_value << "\nf(x) = " << y\_value \* max\_or\_min << "\ntime: "<< (double)(clock\_end - clock\_st) / CLOCKS\_PER\_SEC << endl;

        file << endl;

        //Метод «деления отрезка пополам»

        file<<"---segments\_devide method, for segment "<< "["<< seg\_start << "," << seg\_end << "]" << endl;

        pair<double, double> res;

        clock\_st = clock();

        while (true) {

            double x;iteration\_for\_met2++;

            res = segments\_devide(copy\_seg\_start, copy\_seg\_end, EPS, max\_or\_min);

            copy\_seg\_start = res.first;

            copy\_seg\_end = res.second;

            x = (copy\_seg\_end+copy\_seg\_start) / 2;

            y = function(x);

            file << "iteration " << iteration\_for\_met2 << ": x = " << x << ", f(x) = " << y << endl;

            if (copy\_seg\_end - copy\_seg\_start < EPS){

                clock\_end = clock();

                file<<"---res--------------------------------res---"<<endl;

                file << "iterations: "<< iteration\_for\_met2 << "\nmin x = "<< x << "\nf(x) = " << y \* max\_or\_min << "\ntime: "<< (double)(clock\_end - clock\_st) / CLOCKS\_PER\_SEC << endl;

                file << endl;

                break;

            }

        }

        file<<endl;

        clock\_st = clock();

        //Метод «золотого сечения»

        file<<"---gold method, for segment "<< "["<<seg\_start << "," << seg\_end << "]" << endl;

        int k = 0;

        copy\_seg\_start = seg\_start;

        copy\_seg\_end = seg\_end;

        vector<double> fibonacci = generate\_fibonacci(copy\_seg\_start, copy\_seg\_end, EPS);

        while (true) {

            double x;

            iteration\_for\_met3++;

            res = gold\_search(copy\_seg\_start, copy\_seg\_end, k, EPS, fibonacci, max\_or\_min);

            copy\_seg\_start = res.first;

            copy\_seg\_end = res.second;

            x = (copy\_seg\_end + copy\_seg\_start) / 2;

            y = function(x);

            file << "iteration " << iteration\_for\_met3 << ": x = " << x << ", f(x) = " << y << endl;

            if (copy\_seg\_end - copy\_seg\_start < EPS) {

                clock\_end = clock();

                file<<"---res--------------------------------res---"<<endl;

                file << "iterations: "<< iteration\_for\_met3 << "\nmin x = "<< x << "\nf(x) = " << y \* max\_or\_min << "\ntime: "<< (double)(clock\_end - clock\_st) / CLOCKS\_PER\_SEC << endl;

                file << endl;

                file << endl;

                break;

            }

            k = k + 1;

        }

        file<<"\n";

        //метод «фибоначи»

        file<<"---fibo method, for segment "<< "["<<seg\_start << "," << seg\_end << "]" << endl;

        clock\_st = clock();

        copy\_seg\_start = seg\_start;

        copy\_seg\_end = seg\_end;

        while (true) {

            double x;

            iteration\_for\_met4++;

            res = fibonacci\_search(copy\_seg\_start, copy\_seg\_end, EPS, fibonacci, max\_or\_min);

            copy\_seg\_start = res.first;

            copy\_seg\_end = res.second;

            x = (copy\_seg\_end + copy\_seg\_start) / 2;

            y = function(x);

            file << "iteration " << iteration\_for\_met4 << ": x = " << x << ", f(x) = " << y << endl;

            if (copy\_seg\_end - copy\_seg\_start < EPS) {

                clock\_end = clock();

                file<<"---res--------------------------------res---"<<endl;

                file << "iterations: "<< iteration\_for\_met4 << "\nmin x = "<< x << "\nf(x) = " << y \* max\_or\_min << "\ntime: "<< (double)(clock\_end - clock\_st) / CLOCKS\_PER\_SEC << endl;

                file << endl;

                file << endl;

                break;

            }

        }

        file<<"\n";

    }

    file.close();

    // Вывод из файла

    ifstream file\_for\_out("file.txt");

    string line;

    while (getline(file\_for\_out, line)) {

        cout << line << endl;

    }

    file.close();

    return 0;

}