Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

ОПТИМИЗАЦИЯ ФУНКЦИИ ОДНОЙ ПЕРЕМЕННОЙ

Отчет по лабораторной работе №1

По дисциплине

«Методы оптимизации»

Выполнил:

Студент гр. 430-2

_____А.А. Лузинсан

«___»_____ 2022 г.

Проверил:

к.т.н., доцент каф. АСУ

_____А.А. Шелестов

«___»_____ 2022 г.

Томск 2022

Содержание

Введение

Задание: найти минимум функции одной переменной, используя два прямых метода из трех (метод равномерного поиска, метод деления отрезка пополам, метод золотого сечения).

Точность $\varepsilon = 10^{-4}$.

Вариант задания:

10) $f(x) = (10x^3 + 3x^2 + x + 5)$

Исходная функция и её производная на отрезке [-20,20] имеет вид, представленный на рисунке 1.1.

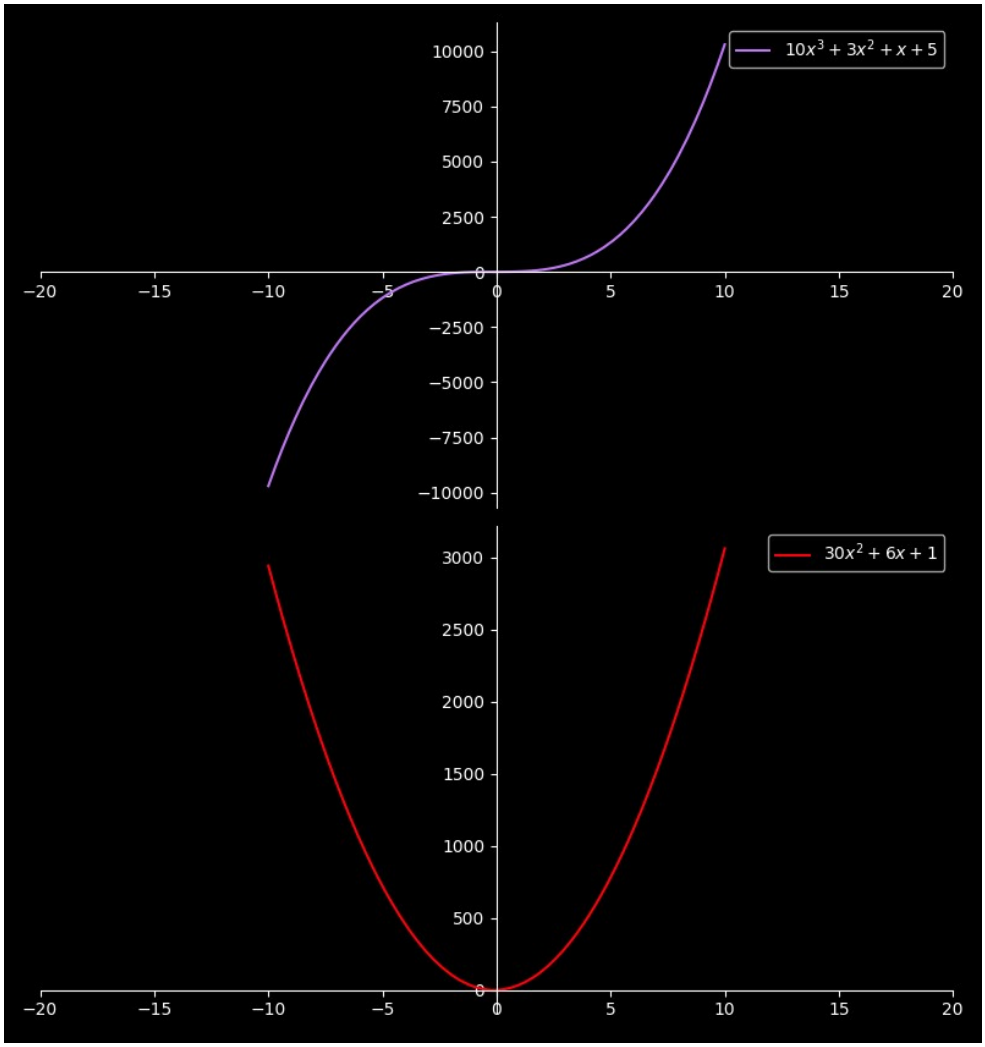

Рисунок 1.1 — исходная функция и первая производная по варианту

1 Теория

1.1 Метод дихотомии

Метод относится к последовательным стратегиям. Задается начальный интервал неопределенности и требуемая точность. Алгоритм опирается на анализ значений функции в двух точках. Для их нахождения текущий интервал неопределенности делится пополам и в обе стороны от середины откладывается по $\frac{\varepsilon}{2}$, где ε - малое положительное число. Условия окончания процесса поиска стандартные: поиск заканчивается, когда длина текущего интервала неопределенности оказывается меньше установленной величины.

Алгоритм

Шаг 1. Задать начальный интервал неопределенности $L_0=[a_0,b_0]$, $\varepsilon>0$ -малое число, $l>0$ – точность ($\varepsilon \in (0,2l)$).

Шаг 2. Положить $k=0$.

Шаг 3. Вычислить $y_k=\frac{a_k+b_k-\varepsilon}{2}, f(y_k)$, $z_k=\frac{a_k+b_k+\varepsilon}{2}, f(z_k)$.

Шаг 4. Сравнить $f(y_k)$ с $f(z_k)$:

а) если $f(y_k)\le f(z_k)$, положить $a_{k+1}=a_k, b_{k+1}=z_k$ и перейти к шагу 5;

б) если $f(y_k)>f(z_k)$, положить $a_{k+1}=y_k, b_{k+1}=b_k$.

Шаг 5. Вычислить $L_k=|b_{k+1}-a_{k+1}|$ и проверить условие окончания:

а) если $L_k\le l$, процесс поиска завершается и в качестве приближенного решения можно взять середину последнего интервала:

$$x^{¿}=\frac{a_{k+1}+b_{k+1}}{2};$$

б) если $L_k > l$, положить $k = k+1$ и перейти к шагу 3.

1.2 Метод золотого сечения

Задается начальный интервал неопределенности и требуемая точность. Алгоритм уменьшения интервала опирается на анализ значений функции в двух точках. В качестве точек вычисления функции выбираются точки золотого сечения. Тогда с учетом свойств золотого сечения на каждой итерации, кроме первой, требуется только одно новое вычисление функции. Условия окончания процесса поиска стандартные: поиск заканчивается, когда длина текущего интервала неопределенности оказывается меньше установленной величины.

Алгоритм

Шаг 1. Задать начальный интервал неопределенности $L_0 = [a_0, b_0]$, точность $l > 0$.

Шаг 2. Положить $k = 0$.

Шаг 3. Вычислить $y_0 = a_0 + \frac{3-\sqrt{5}}{2}(b_0 - a_0); z_0 = a_0 + b_0 - y_0$.

Шаг 4. Вычислить $f(y_k)$, $f(z_k)$.

Шаг 5. Сравнить $f(y_k)$ с $f(z_k)$:

а) если $f(y_k) \le f(z_k)$, то положить $a_{k+1} = a_k, b_{k+1} = z_k$ и $y_{k+1} = a_{k+1} + b_{k+1} - y_k, z_{k+1} = y_k$. Перейти к шагу 6;

б) если $f(y_k) > f(z_k)$, положить $a_{k+1} = y_k, b_{k+1} = b_k$ и $y_{k+1} = z_k, z_{k+1} = a_{k+1} + b_{k+1} - z_k$.

Шаг 6. Вычислить $L_k = |b_{k+1} - a_{k+1}|$ и проверить условие окончания:

а) если $L_k \le l$, процесс поиска завершается и в качестве приближенного решения можно взять середину последнего интервала:

$$x^{¿} = \frac{a_{k+1} + b_{k+1}}{2} \quad ;$$

б) если $L_k > l$, положить $k = k + 1$ и перейти к шагу 4.

2 Результаты работы программы

Эффективность метода дихотомии и золотого сечения на примере функции по варианту представлены на рисунке 2.1 и 2.2 соответственно. Графически, приближение минимума функции представлены на рисунке 2.3 и 2.4 соответственно.

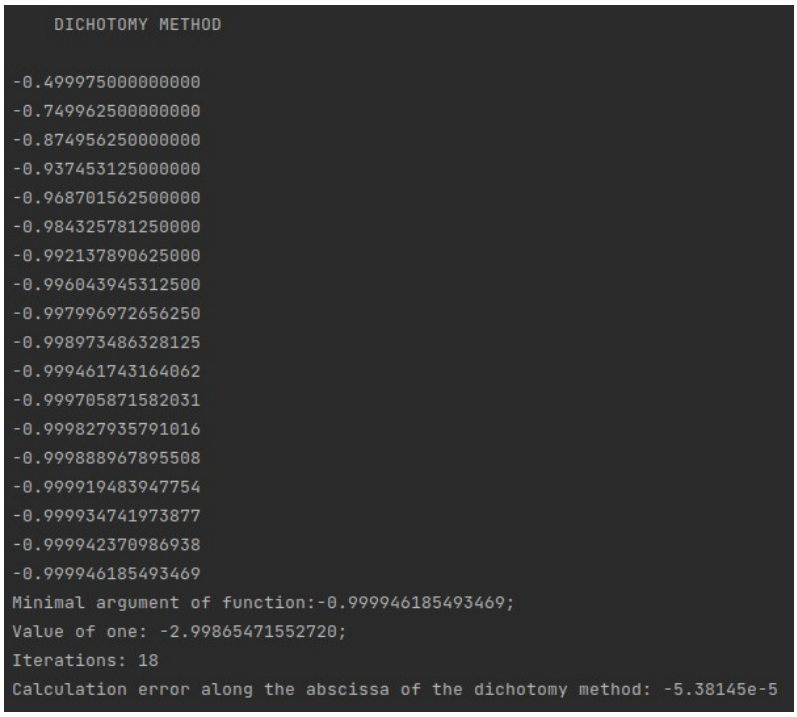

Рисунок 2.1 – минимизация функции по варианту методом дихотомии

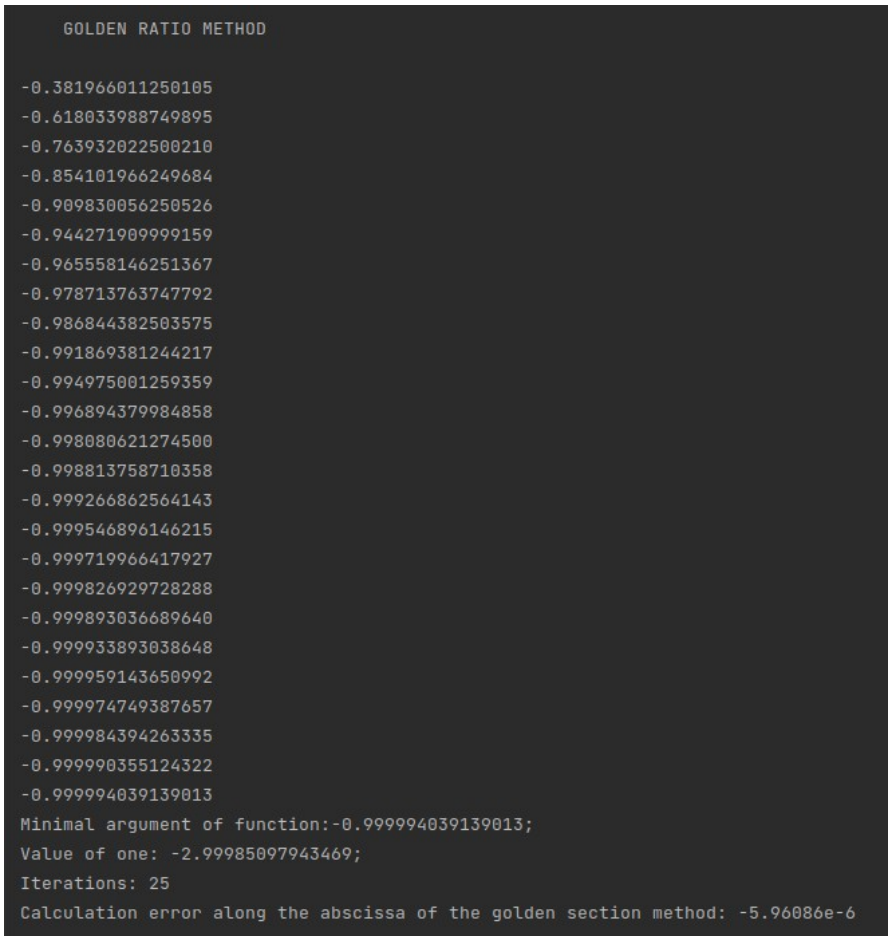

Рисунок 2.2 – минимизация функции по варианту методом золотого сечения

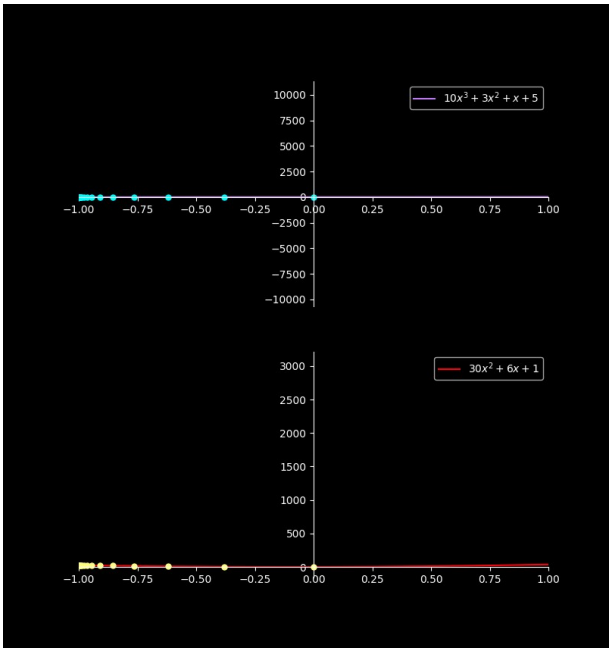

Рисунок 2.3 – график функции процесса минимизация функции по варианту методом дихотомии

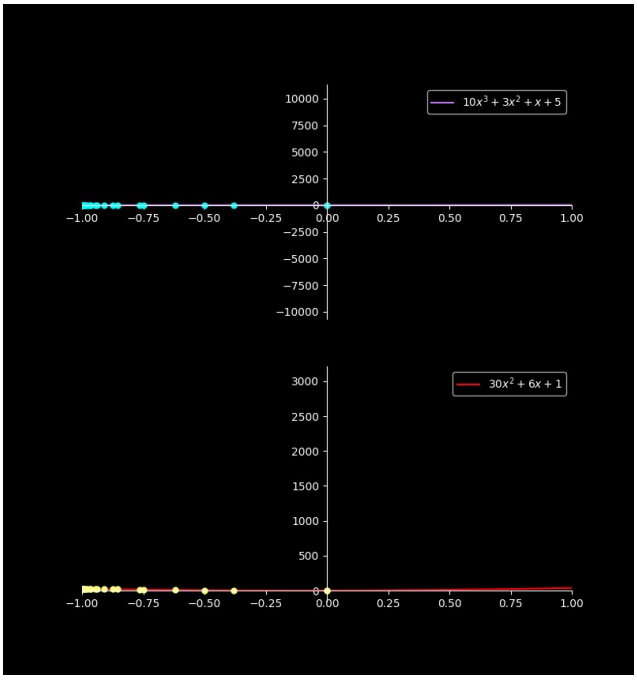

Рисунок 2.4 – график функции процесса минимизация функции по варианту методом золотого сечения

Заключение

В ходе выполнения данной лабораторной работы я изучила теоретические сведения, связанные с методами минимизации функций одной переменной и нашла минимум функции одной переменной, используя два прямых метода: метод дихотомии, метод золотого сечения; а также сравнила данные методы на одной функции по варианту.

Список использованных источников

1. Грибанова, Е. Б. Исследование операций и методы оптимизации: Учебное пособие [Электронный ресурс] / Грибанова Е. Б., Мицель А. А. — Томск: ТУСУР, 2017. — 185 с. — Режим доступа: https://edu.tusur.ru/publications/7127 (дата обращения: 05.09.2022)
2. Мицель, А. А. Методы оптимизации: Учебное пособие [Электронный ресурс] / Мицель А. А., Шелестов А. А., Романенко В. В. — Томск: ТУСУР, 2017. — 198 с. — Режим доступа: https://edu.tusur.ru/publications/7045 (дата обращения: 05.09.2022)
3. Грибанова, Е. Б. Исследование операций и методы оптимизации: Методические указания к лабораторным работам [Электронный ресурс] / Грибанова Е. Б. — Томск: ТУСУР, 2017. — 110 с. — Режим доступа: https://edu.tusur.ru/publications/7128 (дата обращения: 06.09.2022)

Листинг программы

```
from sympy import *
import numpy as np
from math import sqrt
from sympy.parsing.sympy_parser import parse_expr
from sympy.parsing.sympy_parser import standard_transformations, implicit_multiplication_application
from matplotlib import style

x = symbols('x')

class Expression:
    def __init__(self, filename='', **kwargs) -> None:
        if filename != '':
            with open(filename, "rt") as file:
                expression = file.readline()
                    self.__start, self.__end, self.__eps = [parse_expr(num).evalf() for num in file.readline().split()]
                                        transformations = (standard_transformations + (implicit_multiplication_application,))
                                        self.__function: Expr = parse_expr(expression, transformations=transformations)
            for name, value in kwargs.items():
                setattr(self, name, value)
            self.__diff1: Expr = diff(self.__function, x, 1)
            self.__diff2: Expr = diff(self.__function, x, 2)
            self.__diff3: Expr = diff(self.__function, x, 3)
```

```
        self.__x_min = 0.0
        self.__y_min = 0.0
        self.__x_approx_min: np.array = None
        self.__y_val_approx_min: np.array = None
        self.__y_diff_val_approx_min: np.array = None

                    self.__plot_0: plotting = plot(self.__function, show=False, xlim=(self.__start, self.__end), markers=[],
                                                   line_color='xkcd:light purple', legend=True, xlabel=None, ylabel=None)

            self.__plot_1: plotting = plot(self.diff1, show=False, xlim=(self.__start, self.__end), markers=[],
                                                    line_color='xkcd:bright red', legend=True, xlabel=None, ylabel=None)
            self.__plot_grid = plotting.PlotGrid(2, 1, self.__plot_0, self.__plot_1, show=False, size=(8., 4.5))
        style.use('dark_background')
        self.show_plot()

    # region Property
    @property
    def min(self) -> {float: float}:
        return {self.__x_min: self.__y_min}

    @property
    def func(self) -> Expr:
        return self.__function
```

```
@property
def diff1(self) -> Expr:
    return self.__diff1

@property
def diff2(self) -> Expr:
    return self.__diff2

@property
def diff3(self) -> Expr:
    return self.__diff3

def show_plot(self) -> None:
    self.__plot_grid.show()

# endregion

# region StaticMethods
@staticmethod
def border_shift(function, start, end, x_left, x_right):
    f_k0 = function.subs(x, x_left).evalf()
    f_k1 = function.subs(x, x_right).evalf()
    if f_k0 < f_k1:
        end = x_right
    else:
        start = x_left
    return start, end
```

```
        @staticmethod
        def initial_approximation(diff1, diff3, left, right):
            """Начальное приближение минимума функции"""
            # критерий сходимости: f'(x_k) * f'"(x_k) > 0
                if diff1.subs(x, left) * diff3.subs(x, left) > 0:  # левая граница подходит?
                    return left
                elif diff1.subs(x, right) * diff3.subs(x, right) > 0:  # правая граница подходит?
                    return right
                elif diff1.subs(x, (left + right) / 2) * diff3.subs(x, (left + right) / 2) > 0:  # тогда может середина?
                    return (left + right) / 2
                else:
                    return None

        is_stop_criterion = lambda self, x_prev, x_curr: \
                (abs(x_curr - x_prev) <= self.__eps and abs(self.diff1.subs(x, x_curr).evalf()) <= self.__eps) \
            or (x_curr < self.__start or x_curr > self.__end)

        is_min_extremum = lambda diff1_func, diff2_func, argument, eps: \
            (abs(diff1_func.subs(x, argument).evalf()) <= eps) and diff2_func.subs(x, argument).evalf() >= 0

        def append_approximation(self, x_next):
            np.append(self.__x_approx_min, x_next)
```

```
                        np.append(self.__y_val_approx_min,  self.__function.subs(x,
x_next).evalf())
                    self.__plot_0.markers.append({'args': [x_next, self.__function.subs(x,
x_next).evalf(), 'o'],
                                                  'color': 'xkcd:cyan', 'ms': 5})
                    np.append(self.__y_diff_val_approx_min,  self.__diff1.subs(x,
x_next).evalf())
                    self.__plot_1.markers.append({'args': [x_next, self.__diff1.subs(x,
x_next).evalf(), 'o'],
                                                  'color': 'xkcd:pale yellow', 'ms': 5})
                return x_next

            # endregion

            # region Simple Methods
            def dichotomy(self):
                x_curr = (self.__start + self.__end) / 2
                start_curr, end_curr = self.__start, self.__end
                while true:
                    x_prev = x_curr
                    self.append_approximation(x_curr)
                    x_left = x_curr - (self.__eps / 2)
                    x_right = x_curr + (self.__eps / 2)
                        start_curr, end_curr = Expression.border_shift(self.__function,
start_curr, end_curr, x_left, x_right)
                    x_curr = (start_curr + end_curr) / 2
                    print(x_curr)
                    self.show_plot()
```

```
            if abs(end_curr - start_curr) / 2 < self.__eps \
                    and abs(self.__function.subs(x, x_curr).evalf()
                            - self.__function.subs(x, x_prev).evalf()) < self.__eps:
                break
        self.__x_min = x_curr
        self.__y_min = self.__function.subs(x, x_curr).evalf()

        # self.plot.show()
        return self.__x_min, self.__y_min

    def golden_ratio(self):
        harmonic_division = (1 + sqrt(5)) / 2
        x_curr = (self.__start + self.__end) / 2
        start_curr, end_curr = self.__start, self.__end
        while true:
            x_prev = x_curr
            self.append_approximation(x_curr)
            delta = end_curr - start_curr
            x_left = start_curr + (delta / (harmonic_division ** 2))
            x_right = start_curr + (delta / harmonic_division)
                    start_curr,  end_curr  =  Expression.border_shift(self.__function,
start_curr, end_curr, x_left, x_right)
            x_curr = (start_curr + end_curr) / 2
            print(x_curr)
            self.show_plot()
            if abs(end_curr - start_curr) / 2 < self.__eps \
                    and abs(self.__function.subs(x, x_curr).evalf()
                            - self.__function.subs(x, x_prev).evalf()) < self.__eps:
```

```
            break
        self.__x_min = x_curr
        self.__y_min = self.__function.subs(x, x_curr).evalf()
        return self.__x_min, self.__y_min

    # endregion
```