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 №2

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 метода, основанных на производных (метод Ньютона-Рафсона, метод средней точки)

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.

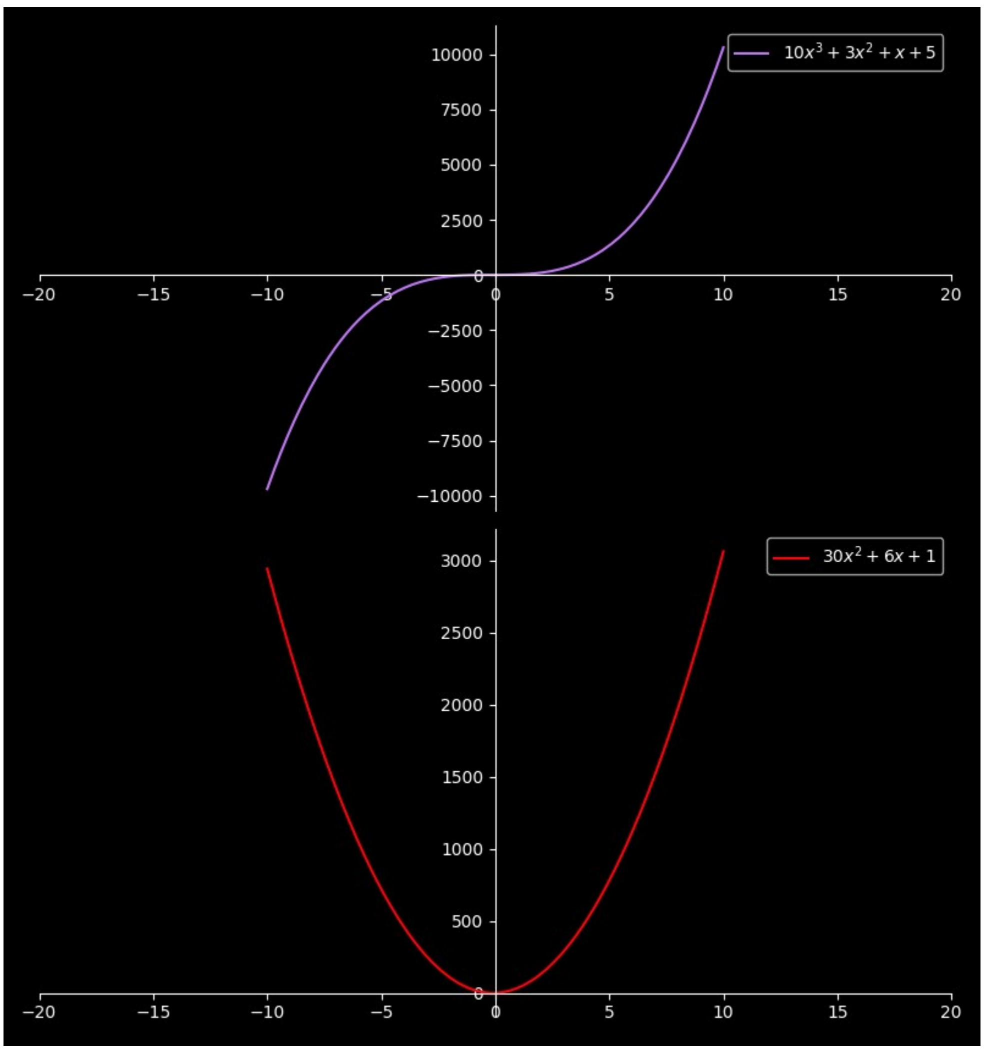


Рисунок 1.1 — исходная функция и первая производная по варианту

1 Теория

1.1 Метод Ньютона

Классическая поисковая схема метода Ньтона-Рафсона (МНР) разработана Ньютоном и позднее уточнена Рафсоном.

Пусть имеется текущее приближение точки экстремума x^k . Тогда следующее приближение x^{k+1} будем искать в виде $x^{k+1}=x^k+h_k$.

Критерий выбора начальной точки:

$$f'(x^0)*f'''(x^0)>0 \tag{1.1}$$

Критерий останова итерационного процесса выглядит следующим образом:

$$|x^{k+1}-x^k|\leq\varepsilon_x, |f(x^k)|\leq\varepsilon_y, \tag{1.2}$$

где $\varepsilon_x u \varepsilon_y$ - требуемая точность решения по аргументу и по функции соответственно. В зависимости от определения функции h_k различают несколько методов:

$$x^{k+1}=x^k-\frac{f'(x^k)}{f''(x^k)}$$ метод Ньютона-Рафсона (1.3)

$$x^{k+1}=x^k-\frac{f(x^k)}{f''(x^0)}$$ упрощённый метод Ньютона (1.4)

$$x^{k+1}=x^k-\frac{(x^k-x^{k-1})*f'(x^k)}{f'(x^k)-f'(x^{k-1})}$$ метод секущих (1.5)

$$x^{k+1}=x^k-\frac{f'(x^k)^2}{f'(x^k+f'(x^k))-f'(x^k)}$$ метод Стефенсена (1.6)

$$x^{k+1}=x^k-\frac{f'(x^k)}{f''(x^k)-\frac{f'(x^k)*f'''(x^k)}{2f''(x^k)}}$$ метод Уолла (1.7)

Алгоритм

Шаг 1. Выполняем присваивание k=0, а также выбираем на отрезке [a, b] начальную точку x^0 , используя критерий (1.1);

Шаг 2. Вычисляем следующее приближение точки экстремума, используя одну из формул (1.3 — 1.7);

Шаг 3. Вычисляем значения ЦФ и ее производных в точке x^{k+1} ;

Шаг 4. Если выполняется критерий останова итерационного процесса (1.2), то полагаем $\tilde{x} \approx x^{k+1}$ и переходим на шаг 5. Иначе выполняем присваивание k=k+1 и переходим на шаг 2.

Шаг 5. Если точка x удовлетворяет критерию оптимальности, то принимаем ее за решение задачи: $\bar{x} = \tilde{x}$ В противном случае при заданных начальных условиях найти оптимум задачи невозможно.

1.2 Метод Больцано

Пусть задана функция $f(x), x \in [a, b]$ и задана точность по аргументу ε_x и функции ε_y .

Алгоритм

Шаг 1. Положить $k=0, a^0=a, b^0=b,$ при этом $f'(a)<0, f'(b)>0.$

Шаг 2. Вычислить $\bar{x} = \frac{a^k + b^k}{2}$ и значение $f'(\bar{x}^k).$

Шаг 3. Если заданная точность достигнута, т.е.

$$\frac{b^k - a^k}{2} \le \varepsilon_x \quad \text{и} \quad |f'(\bar{x}^k)| \le \varepsilon_y$$

то закончить поиск и принять $\tilde{x} \approx x^k$. Иначе:

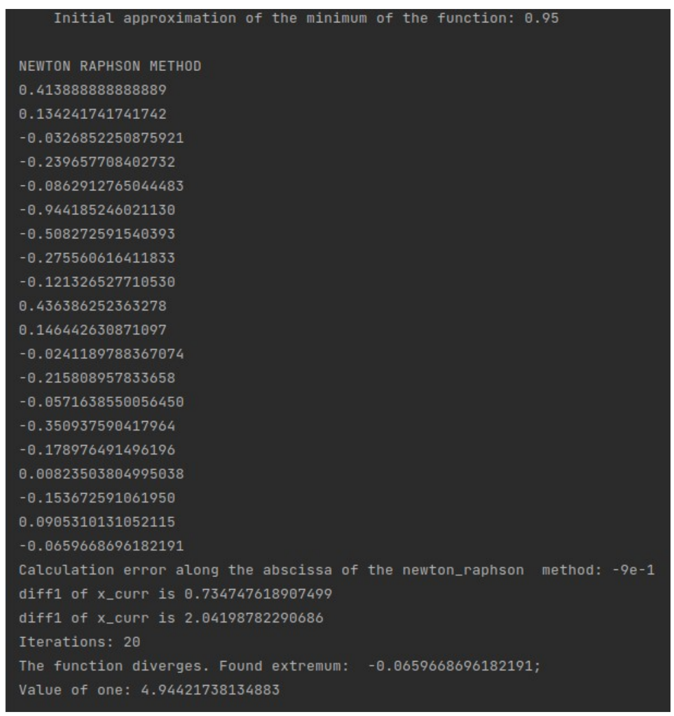
$$\left\{ \begin{matrix} \text{Если } f'(\bar{x}^k) > 0, \text{положить } a^{k+1} = a^k, b^{k+1} = \bar{x}^k, \\ \text{Иначе положить } a^{k+1} = \bar{x}^k, b^{k+1} = b^k, \end{matrix} \right\}$$

положить k=k+1 и перейти на шаг 2.

2 Результаты работы программы

В данной лабораторной работе были реализован как метод Ньютона-Рафсона, так и остальные перечисленные выше методы: упрощённый метод Ньютона, метод секущих, метод Стефенсена и метод Уолла; а также метод средней точки (метод Больцано). Реализация оных методов писалась как дополнение к классу, реализовывающей методы из первой лабораторной работы.

Эффективность методов Ньютона на примере функции по варианту представлены на рисунках 2.1-2.5. Графически, приближение минимума функции (или точек перегиба функции) представлены на рисунках 2.6 - 2.10 соответственно. Рассматриваемый интервал: [-1,1].



```
    Initial approximation of the minimum of the function: 0.95

NEWTON RAPHSON METHOD
0.413888888888889
0.134241741741742
-0.0326852250875921
-0.239657708402732
-0.0862912765044483
-0.944185246021130
-0.508272591540393
-0.275560616411833
-0.121326527710530
0.436386252363278
0.146442630871097
-0.0241189788367074
-0.215808957833658
-0.0571638550056450
-0.350937590417964
-0.178976491496196
0.00823503804995038
-0.153672591061950
0.0905310131052115
-0.0659668696182191
Calculation error along the abscissa of the newton_raphson  method: -9e-1
diff1 of x_curr is 0.734747618907499
diff1 of x_curr is 2.04198782290686
Iterations: 20
The function diverges. Found extremum:  -0.0659668696182191;
Value of one: 4.94421738134883
```

Рисунок 2.1 – минимизация функции по варианту методом Ньютона-Рафсона

```
    Initial approximation of the minimum of the function: 0.05

NEWTON SIMPLE METHOD
-0.102777777777778
-0.180581275720165
-0.280003526820240
-0.465785536823767
-0.989560177766382
-3.70506232177001
Calculation error along the abscissa of the newton_simple  method: 3e+0
diff1 of x_curr is 390.594230315374
diff1 of x_curr is 216.303739306201
Iterations: 6
The function diverges. Found extremum:  -3.70506232177001;
Value of one: -466.134543353731
```

Рисунок 2.2 – минимизация функции по варианту упрощённым методом Ньютона

```
    Initial approximation of the minimum of the function: 1000.5

NEWTON SECANT METHOD
-1.00083358340839
Calculation error along the abscissa of the newton_secant  method: 8e-4
diff1 of x_curr is 25.0450343498922
diff1 of x_curr is 54.0500150045036
Iterations: 1
The function diverges. Found extremum:  -1.00083358340839;
Value of one: -3.02085835225716
```

Рисунок 2.3 – минимизация функции по варианту методом секущих

```
    Initial approximation of the minimum of the function: 0.1

NEWTON STEPHENSEN METHOD
0.0724637681159420
0.0450654681766585
0.0176965663127114
-0.00982875367856577
-0.0378151271472013
-0.0667399609255665
-0.0973006087849581
-0.130378903880199
-0.166748934818914
-0.206437813837230
-0.248351437894176
-0.290983980471879
-0.333332720963714
-0.374999398565545
-0.415912415346329
-0.456123439063119
-0.495716956687476
-0.534777445555541
-0.573379478976652
-0.611586066243774
Calculation error along the abscissa of the newton_stephensen  method: -4e-1
diff1 of x_curr is 8.55160909524336
diff1 of x_curr is 30.6951639746264
Iterations: 20
The function diverges. Found extremum:  -0.611586066243774;
Value of one: 3.22296515005623
```

Рисунок 2.4 – минимизация функции по варианту методом Стефенсена

```
    Initial approximation of the minimum of the function: -10.5

NEWTON WALL METHOD
-3.56467222125194
-1.24890050908416
-0.464807146053582
-0.161219465857845
-0.435481257353811
-0.145413640408806
-0.279941042759792
-0.00827471160099541
-3.06187188686474
-1.08028183641037
-0.405430029820535
-0.127726681548291
-0.191289763594053
3.27473258712011
1.01876077066125
0.254265301252975
-0.0443252803751237
0.165396506072463
-0.0993853968497903
-0.0981561109483497
Calculation error along the abscissa of the newton_wall  method: -9e-1
diff1 of x_curr is 0.700101997805044
diff1 of x_curr is 0.110633343099020
Iterations: 20
The function diverges. Found extremum:  -0.0981561109483497;
Value of one: 4.92129078502703
```

Рисунок 2.5 – минимизация функции по варианту методом Уолла

В результате в можем заключить, что методы, основанные на методе Ньютона зачастую подходят для нахождения стационарных точек (минимума, максимума, точек перегиба). Функция по варианту является кубической, в связи с чем методы Ньютона находили точку перегиба, либо вовсе расходились.

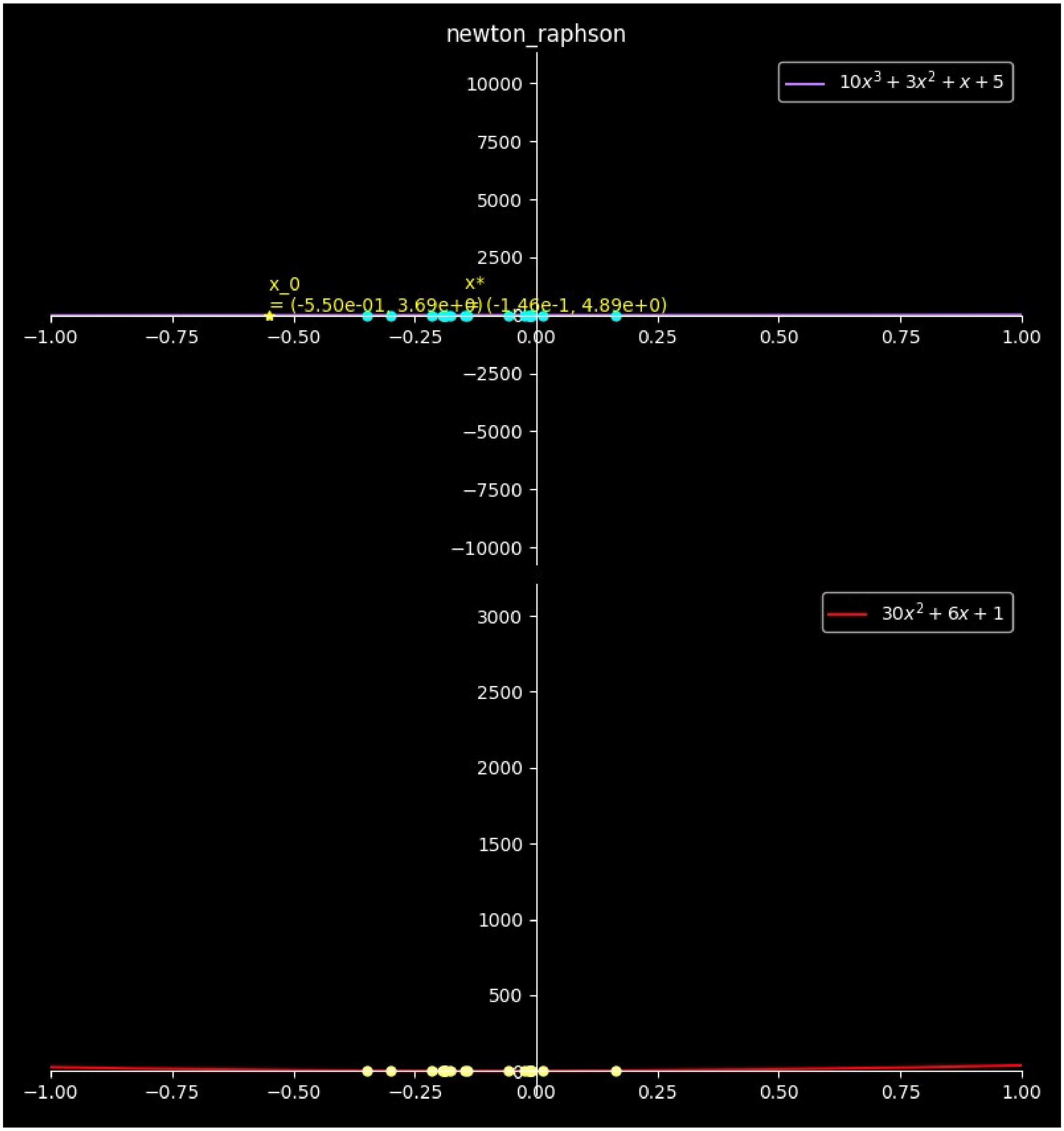


Рисунок 2.6 – график функции процесса минимизации по варианту методом Ньютона-Рафсона

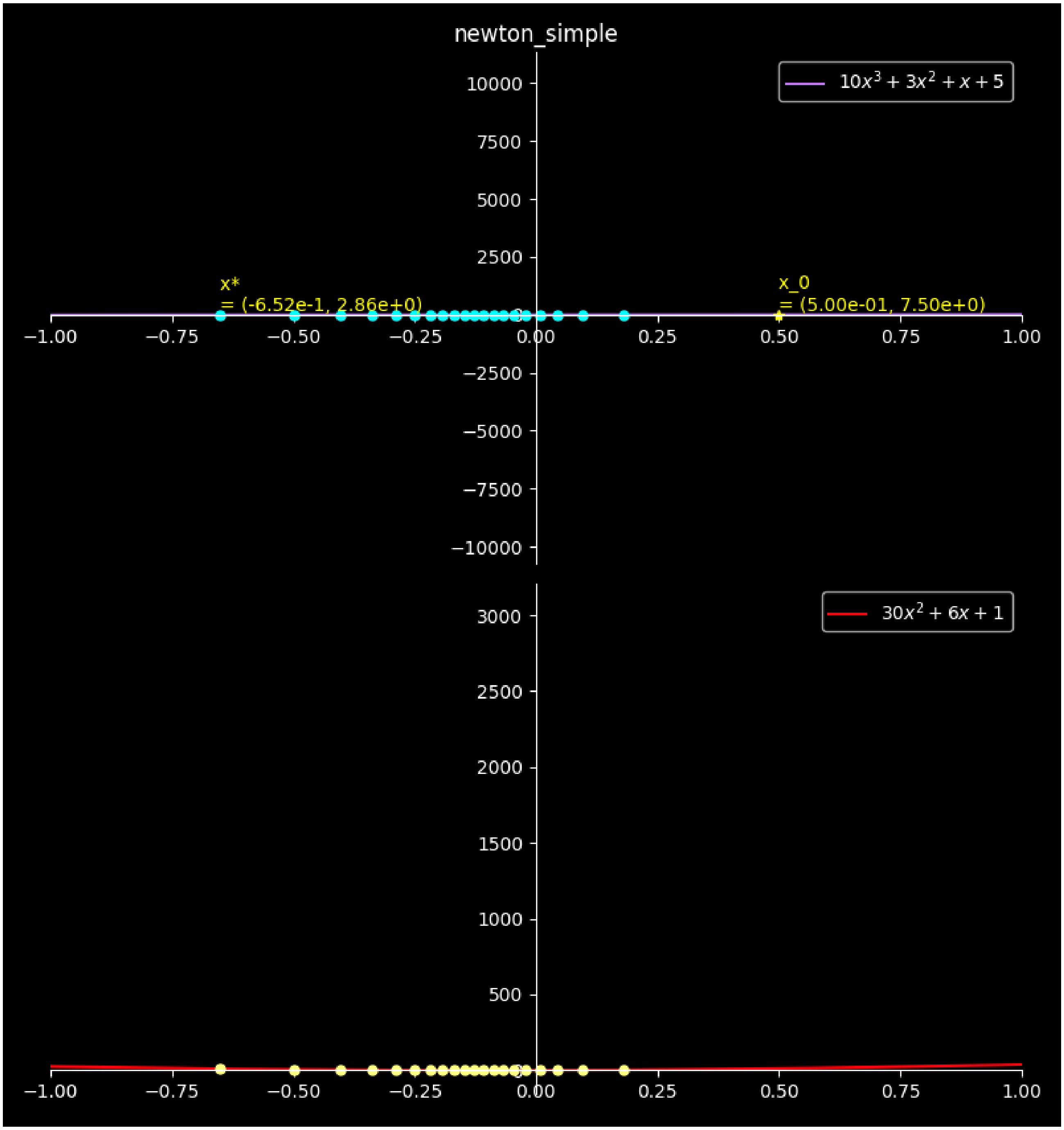


Рисунок 2.7 – график функции процесса минимизация по варианту упрощённым методом Ньютона

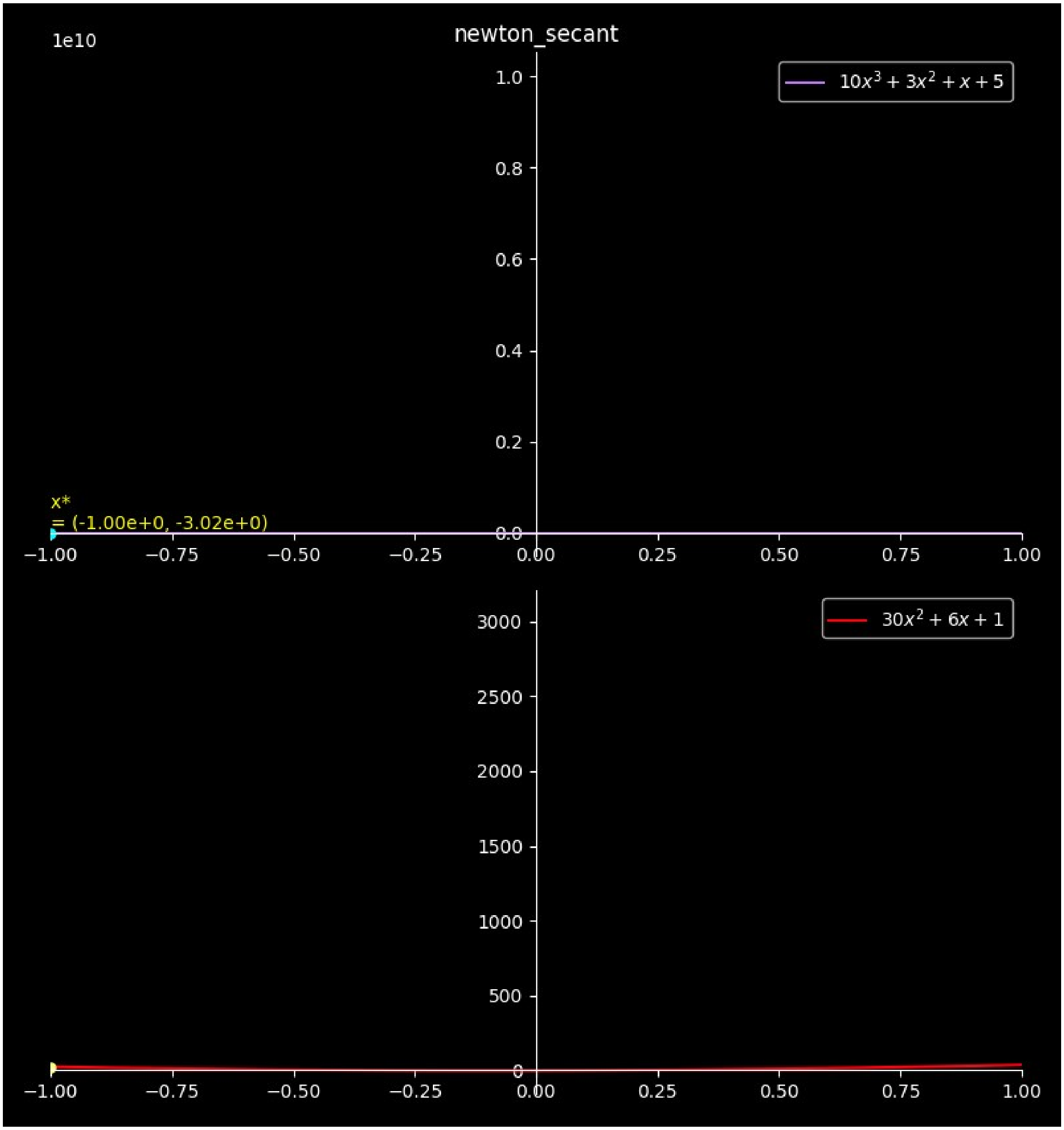


Рисунок 2.8 – график функции процесса минимизации по варианту методом секущих

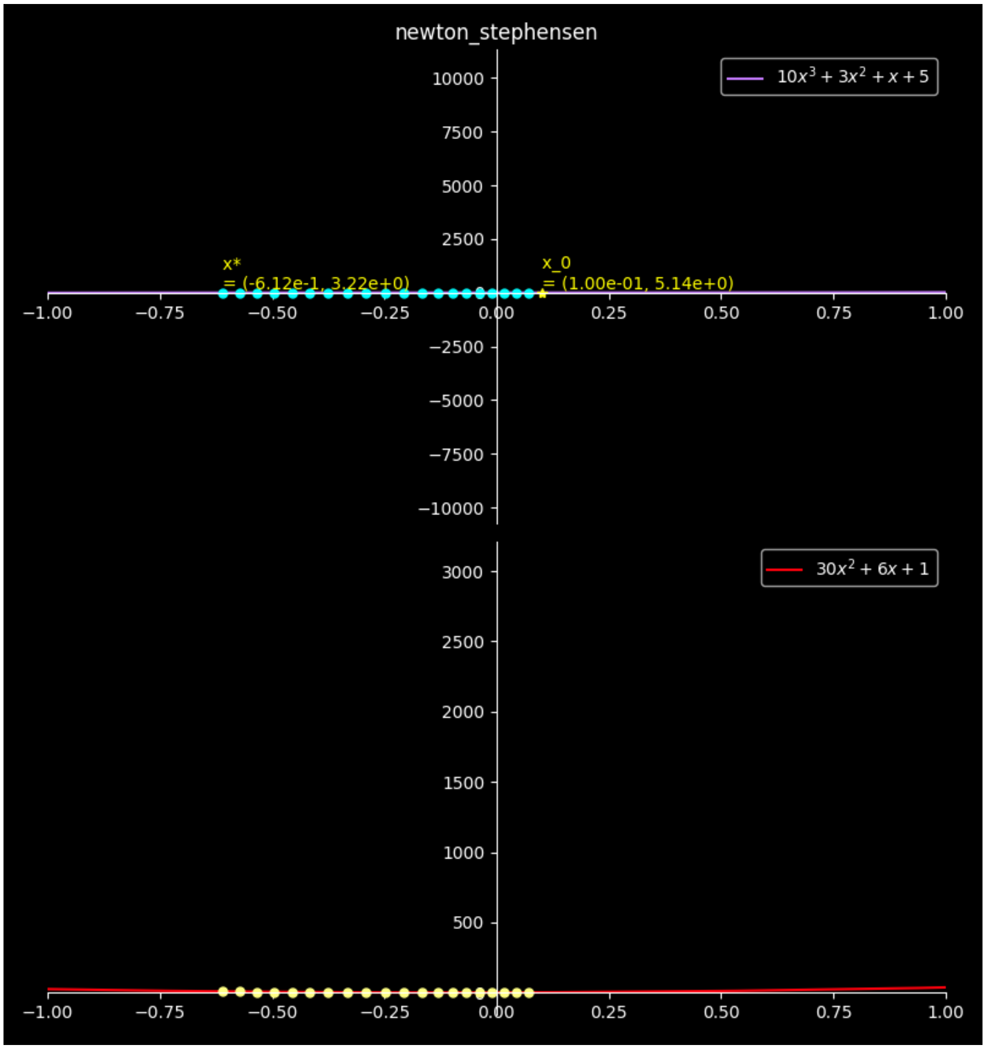


Рисунок 2.9 – график функции процесса минимизации по варианту методом Стефенсена

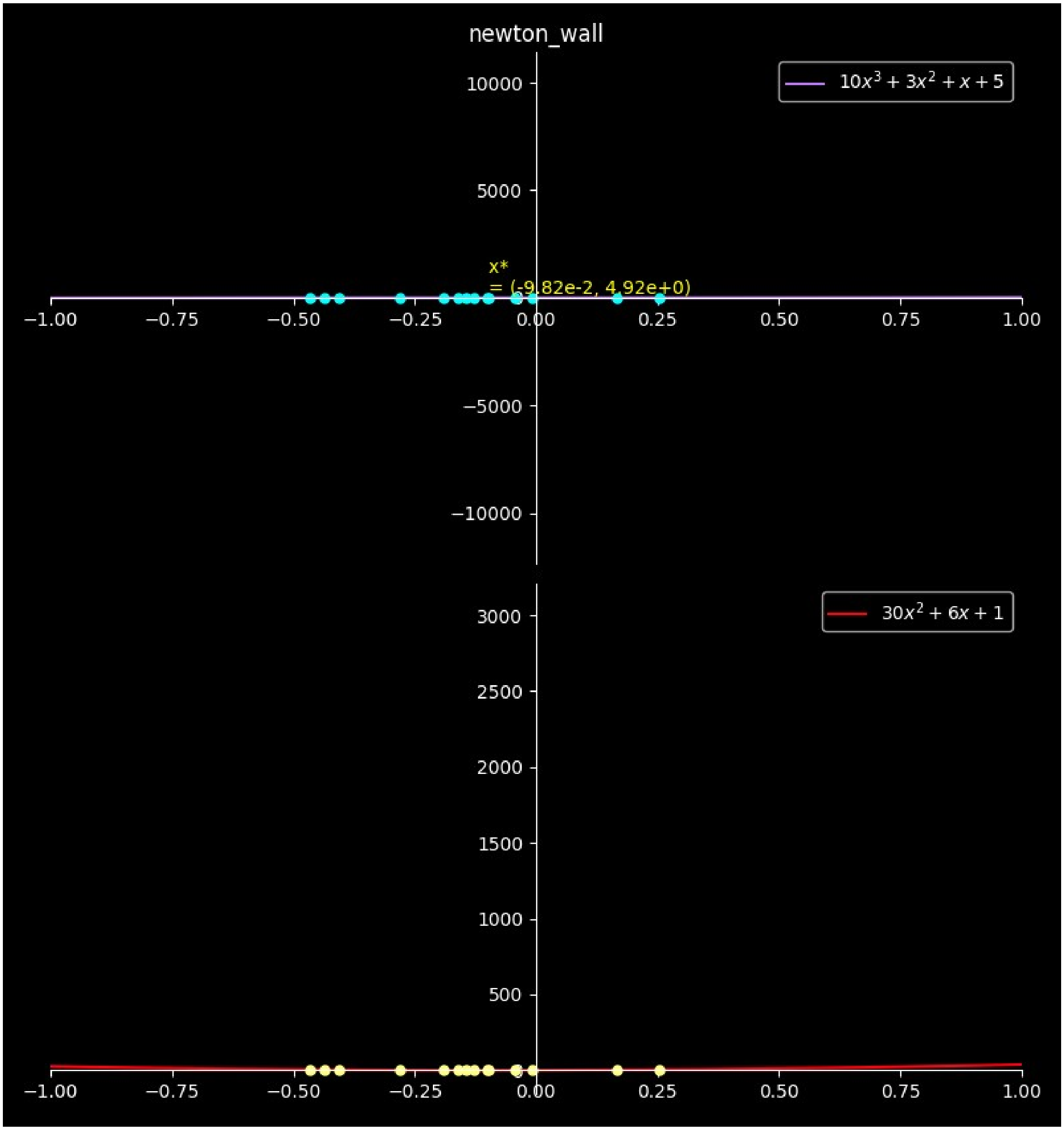


Рисунок 2.10 – график функции процесса минимизации по варианту методом Уолла

Далее был реализован метод Больцано (метод средней точки), принцип которой очень схож с методом дихотомии. Тестирование метода выполнялось на функции по варианту на промежутке [-1; 1]. Результаты работы метода представлены на рисунках 2.11 и 2.12.

```
METHOD BOLZANO
Start approximation: 0.100000000000000
0.100000000000000
-0.450000000000000
-0.725000000000000
-0.862500000000000
-0.793750000000000
-0.828125000000000
-0.845312500000000
-0.853906250000000
-0.858203125000000
-0.860351562500000
-0.859277343750000
-0.859814453125000
-0.860083007812500
-0.859948730468750
-0.859881591796875
-0.859915161132813
-0.859898376464844
-0.859906768798828
-0.859902572631836
-0.859900474548340
-0.859901523590088
-0.859902048110962
-0.859902310371399
-0.859902441501617
-0.859902375936508
-0.859902343153954
Iterations: 26
Minimal argument of function: -0.859902343153954;
Value of one: 3.40189439285155E-7
```

Рисунок 2.11 – минимизация функции по варианту методом Больцано

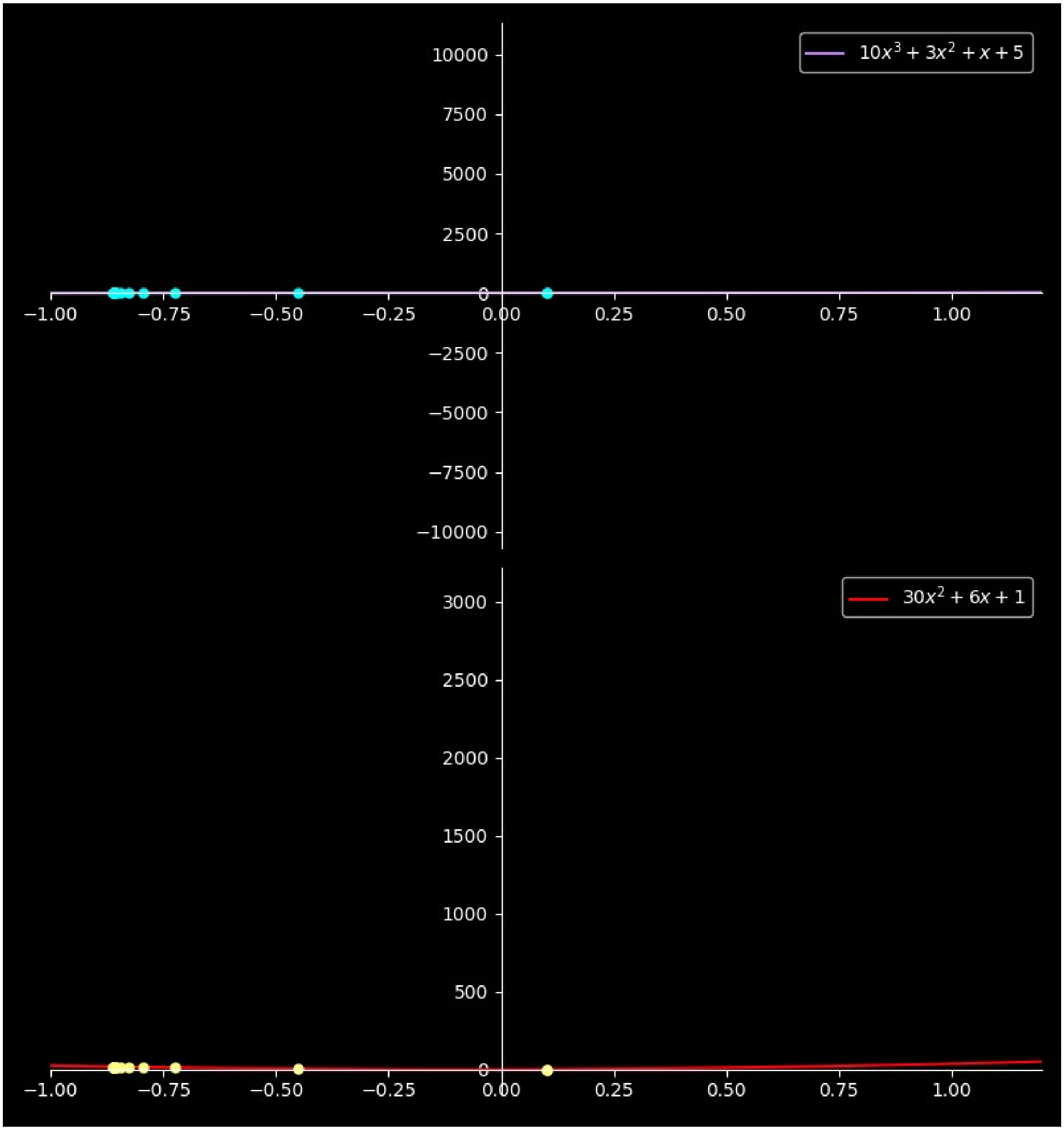


Рисунок 2.12 – график функции по варианту процесса минимизации методом Больцано

Заключение

В ходе выполнения данной лабораторной работы я изучила теоретические сведения, связанные с методами минимизации функций одной переменной через производные и протестировала каждый метод: метод Ньютона-Рафсона, упрощённый метод Ньютона; метод секущих, метод Стефенсена, метод Уолла; а также метод средней точки (метод Больцано) и 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: 12.09.2022)
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: 12.09.2022)
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: 12.09.2022)

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
import imageio

x = symbols('x')
ITERATIONS = 20



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_max = 0.0
        self.__y_max = 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., 8.5))
        style.use('dark_background')
        self.show_plot()

    is_stop_criterion = lambda self, x_prev, x_curr, counter: \
            (abs(x_curr - x_prev) <= self.__eps and abs(self.__diff1.subs(x,
x_curr).evalf()) <= (2 * self.__eps)) \
        or (counter >= ITERATIONS)
```

```
        #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

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

    is_inflection = lambda diff1_func, diff2_func, argument, eps: \
            (abs(diff1_func.subs(x, argument).evalf()) <= (2 * eps)) and
abs(diff2_func.subs(x, argument).evalf()) <= eps

    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
```

```
# region Newton
    def newton_get_method(self, type=0):
        if type == 0:
            print('NEWTON RAPHSON METHOD')
            return self.newton_raphson, 'newton_raphson'
        elif type == 1:
            print('NEWTON SIMPLE METHOD')
            return self.newton_simple, 'newton_simple'
        elif type == 2:
            print('NEWTON SECANT METHOD')
            return self.newton_secant, 'newton_secant'
        elif type == 3:
            print('NEWTON STEPHENSEN METHOD')
            return self.newton_stephensen, 'newton_stephensen'
        elif type == 4:
            print('NEWTON WALL METHOD')
            return self.newton_wall, 'newton_wall'
        else:
            return -1, ''

    def newton_init(self, type=0, x_curr=None):
        self.__x_approx_min: np.array = None
        self.__y_val_approx_min: np.array = None
        self.__y_diff_val_approx_min: np.array = None

        # Проверка критерия сходимости - определение начального приближения
```

```
if not x_curr:
    x_curr = self.initial_approximation(self.__diff1, self.__diff3,
self.__start, self.__end)
method, type_str = self.newton_get_method(type)
self.__plot_0.title = type_str
self.__plot_0.markers.append({'args': [x_curr, self.__function.subs(x,
x_curr).evalf(), '*'],
                              'color': 'xkcd:yellow', 'ms': 6})
self.__plot_0.annotations = [{'xy': (x_curr, self.__function.subs(x,
x_curr).evalf()),
                              'text': 'x_0 \n= ({0:.2e}, {1:.2e})'.format(x_curr,
self.__function.subs(x, x_curr).evalf()),
                              'ha': 'left', 'va': 'bottom', 'color': 'yellow'}]
x_prev = None
if method == self.newton_secant:
    x_prev = self.initial_approximation(self.__diff1, self.__diff3,
self.__start, self.__end - self.__eps)
x_curr, counter, images = self.newton_get_approx_by_method(method,
type_str, x_curr, x_prev)
self.__plot_0.annotations.append({'xy': (x_curr, self.__function.subs(x,
x_curr).evalf()),
                              'text': 'x* \n= ({0:.2e}, {1:.2e})'.format(x_curr,
                                                              self.__function.subs(x,
x_curr).evalf()),
                              'ha': 'left', 'va': 'bottom', 'color': 'yellow'})
eps = str(self.__eps).rfind('1')
print('Calculation error along the abscissa of the {0}  method: {1:.
{2}e}'.format(
```

```
            type_str, -1 - x_curr, eps))
        imageio.mimsave(f'../minimization_of_a_unimodal_function/gifs/{type_
str}.gif', images, duration=0.5)
        return (self.classification(x_curr)), counter

    def newton_get_approx_by_method(self, method, method_str, x_curr,
x_prev=None):
        counter = 0
        images = []
        alpha = None
        if method == self.newton_simple:
            alpha = self.diff2.subs(x, x_curr) if self.diff2.subs(x, x_curr) != 0 else
None
        if x_prev is None:
            x_prev = x_curr
        temp_secant = 0.0
        while true:
            counter += 1
            if method == self.newton_secant:
                temp_secant = x_curr
                x_curr = method(x_curr, x_prev, alpha)
                x_prev = temp_secant
            else:
                x_prev = x_curr
                x_curr = method(x_curr, x_prev, alpha)
            print(x_curr)
            self.append_approximation(x_curr)
            self.__plot_grid.save(
```

```
                    f'../minimization_of_a_unimodal_function/Plots/newton/{method_st
r}/{method_str}{counter}.png')
                images.append(imageio.imread(
                    f'../minimization_of_a_unimodal_function/Plots/newton/{method_st
r}/{method_str}{counter}.png'))
                if self.is_stop_criterion(x_prev, x_curr, counter):
                    return (x_curr, counter, images)


        def classification(self, x_curr):
            #  Классификация найденной точки
            if  Expression.is_min_extremum(self.__diff1,  self.__diff2,  x_curr,
self.__eps):
                self.__x_min = x_curr
                self.__y_min = self.__function.subs(x, x_curr).evalf()
                return (self.__x_min, self.__y_min, 'min')
            elif  Expression.is_max_extremum(self.__diff1,  self.__diff2,  x_curr,
self.__eps):
                self.__x_max = x_curr
                self.__y_max = self.__function.subs(x, x_curr).evalf()
                return (self.__x_max, self.__y_max, 'max')
            elif   Expression.is_inflection(self.__diff1,   self.__diff2,   x_curr,
self.__eps):
                return (x_curr, self.__function.subs(x, x_curr).evalf(), 'inflection')
            else:
                print(f'diff1 of x_curr is {abs(self.__diff1.subs(x, x_curr).evalf())}')
                print(f'diff1 of x_curr is {abs(self.__diff2.subs(x, x_curr).evalf())}')
                return (x_curr, self.__function.subs(x, x_curr).evalf(), 'diverged')
```

```
# region Methods
newton_raphson = lambda self, x_curr, x_prev=0.0, alpha=0.0: \
    x_curr - self.__diff1.subs(x, x_curr) / self.__diff2.subs(x, x_curr)

newton_simple = lambda self, x_curr, x_prev, alpha: \
    x_curr - (self.__diff1.subs(x, x_curr) / alpha)

newton_secant = lambda self, x_curr, x_prev, alpha=0.0: \
    x_curr - (x_curr - x_prev) \
            / (self.__diff1.subs(x, x_curr) - self.__diff1.subs(x, x_prev)) * self.__diff1.subs(x, x_curr)

newton_stephensen = lambda self, x_curr, x_prev=0.0, alpha=0.0: \
    x_curr - ((self.__diff1.subs(x, x_curr) ** 2) \
            / (self.__diff1.subs(x, x_curr + self.__diff1.subs(x, x_curr))
              - self.__diff1.subs(x, x_curr)))

newton_wall = lambda self, x_curr, x_prev=0.0, alpha=0.0: \
    x_curr - self.__diff1.subs(x, x_curr) \
            / (self.__diff2.subs(x, x_curr) - ((self.__diff1.subs(x, x_curr) * self.__diff3.subs(x, x_curr))
                             / (2 * self.__diff2.subs(x, x_curr))))
# endregion
# endregion

# region Bolzano
def bolzano(self):
    print("METHOD BOLZANO")
```

```
x_curr = (self.__start + self.__end) / 2
print(f'Start approximation: {x_curr}')
self.__x_approx_min: np.array = None
self.__y_val_approx_min: np.array = None
self.__y_diff_val_approx_min: np.array = None
start_curr, end_curr = self.__start, self.__end
counter = 0
images = []
while true:
    x_prev = x_curr
    self.append_approximation(x_curr)
    x_curr = (start_curr + end_curr) / 2
            start_curr, end_curr = Expression.border_shift(self.__function,
start_curr, end_curr, x_center=x_curr)
    print(x_curr)
    self.__plot_grid.save(f'../minimization_of_a_unimodal_function/Plots/
bolzano/bolzano{counter}.png')
    images.append(
        imageio.imread(f'../minimization_of_a_unimodal_function/Plots/bol
zano/bolzano{counter}.png'))
    counter += 1
    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()
imageio.mimsave('../minimization_of_a_unimodal_function/gifs/bolzano
```

```
.gif', images, duration=0.5)
            return self.__x_min, self.__y_min, counter

        #endregion
```