

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
КАФЕДРА КМАД

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №7

Виконав студент:

Омельніцький Андрій Миколайович
Група: КН-120А

Перевірила:

Ольга Василівна Костюк

Харків, 2023 г.

Зміст

1. Мета роботи	2
2. Основна частина	3
2.1. Пункти 1-4	3
2.2. Пункт 5	5
2.3. Пункт 6	6
3. Висновок	7
4. Код програми	8
4.1. main.py	8
4.2. model.py	9
4.3. opt.py	12

Глава 1

Мета роботи

Вивчення методу дихотомії для зменшення інтервалу невизначеності унімодальної цільової функції, дослідження ефективності методу.

Порядок виконання:

1. Для унімодальної цільової функції однієї змінної з початковою точкою пошуку мінімуму за номером варіанта виконати постановку задачі мінімізації цільової функції.
2. Реалізувати програмно метод дихотомії для зменшення інтервалу невизначеності заданої цільової функції.
3. Здійснити зменшення інтервалу невизначеності заданої цільової функції.
4. Відобразити графічно процес мінімізації цільової функції однієї змінної.
5. Аналітично знайти точку $x^* \in R$ мінімуму заданої функції $f(x)$ і обчислити мінімальне значення функції $f^*(x) = f(x^*)$. Порівняти зі значеннями, які знайдено аналітично, зробити висновки.
6. Використовуючи програму методу, виконати постановку оптимізаційної задачі (взяти функцію однієї змінної $G(\tau)$, врахувати її економічний сенс, розглядаючи модель (1-21) в стаціонарному режимі), здійснити розв'язання оптимізаційної задачі, повторюючи дії пп. 2-5. За результатами розв'язання зробити висновки щодо оптимального значення норми оподаткування.
7. Код програми, усі результати, отримані в ході виконання роботи, занести до звіту. Зробити висновки.

Глава 2

Основна частина

2.1. Пункти 1-4

Для функції та початкових умов (1) виконати її мінімізацію методом дихотомії та змодлювати процес мінімізації.

$$f(x) = 2 - \frac{1}{\log_2(x^4 + 4x^3 + 29)}, x_0 = -1, \varepsilon = 0.01 \quad (1)$$

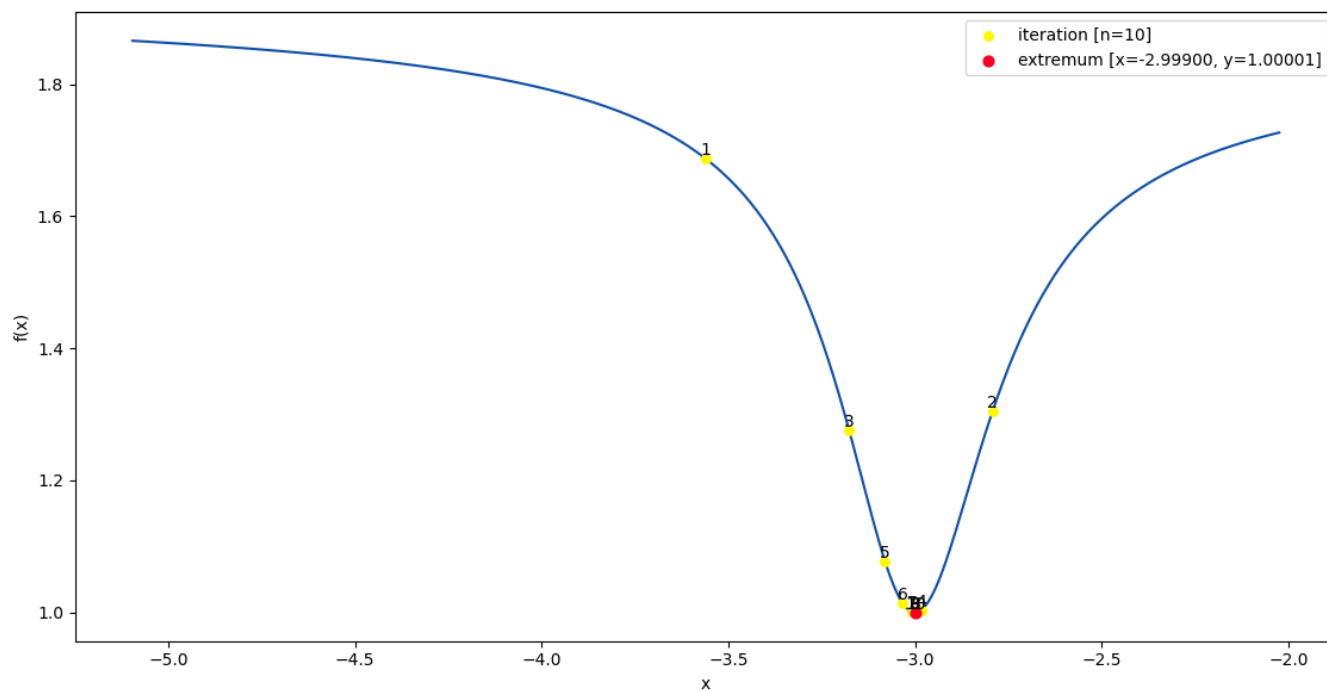


Рис. 2.1. Мінімізація функції

За результатом роботи програми отримуємо такий результат. Де можна побачити інтервал невизначеності який було отримано методом свена. Також можна побачити назву метода, вхідні параметри методу, та ітерації методу. В виведенні ітерації є номер ітерації, L - довжина інтервала, значення x та значення $f(x)$.

```
Sven method interval with step=0.001: (-5.096, -2.024)
Optimization method: Dichotomy method
Initial parameters:
    f(x) = 2 - (1 / (np.log2(x**4 + 4*(x**3) + 29)))
    l = -5.09600, r = -2.02400
iteration 0:    L = 1.53600    x = -3.56000    f(x) = 1.68686
iteration 1:    L = 0.76800    x = -2.79200    f(x) = 1.30438
iteration 2:    L = 0.38400    x = -3.17600    f(x) = 1.27520
iteration 3:    L = 0.19200    x = -2.98400    f(x) = 1.00329
iteration 4:    L = 0.09600    x = -3.08000    f(x) = 1.07716
iteration 5:    L = 0.04800    x = -3.03200    f(x) = 1.01325
iteration 6:    L = 0.02400    x = -3.00800    f(x) = 1.00083
iteration 7:    L = 0.01200    x = -2.99600    f(x) = 1.00021
iteration 8:    L = 0.00600    x = -3.00200    f(x) = 1.00005
iteration 9:    L = 0.00600    x = -2.99900    f(x) = 1.00001
Result:
    x = -2.99900
    f(x) = 1.00001
Function calls:18
Number of iterations:10
```

Рис. 2.2. Результат роботи

2.2. Пункт 5

Знайдемо мінімум функції (1) аналітично. Для цього знайдемо похідну функції.

$$\frac{d}{dx}(f(x) = 2 - \frac{1}{\log_2(x^4 + 4x^3 + 29)}) = \frac{4x^2(x + 3) \log(2)}{(x^4 + 4x^3 + 29) \log^2(x^4 + 4x^3 + 29)}$$

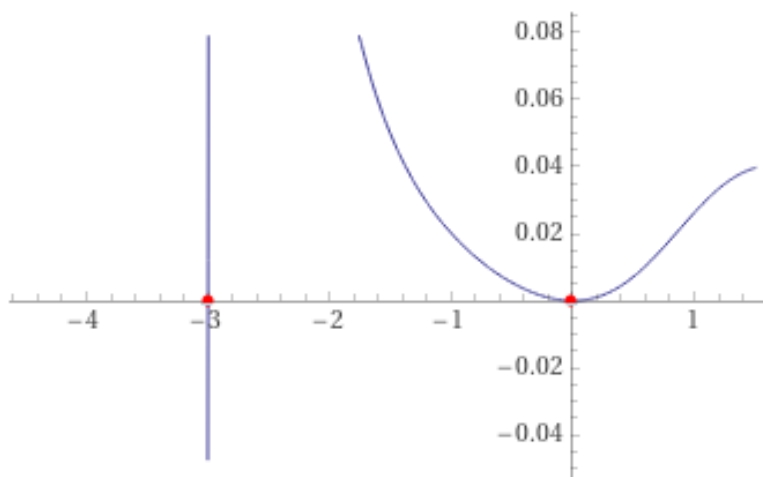


Рис. 2.3. Графік похідної

Маємо дві стаціонарні точки $x_1 = -3$ та $x_2 = 0$. Точка $x_1 = -3, f(x_1) = 1$ є точкою мінімуму яка й співпадає з точкою знайденою медодом розглянутим вище. А точка $x_2 = 0$ не є точкою екстремума.

2.3. Пункт 6

Враховуючи економічний сенс функції $G(\tau)$ з лаб. 5. яка є функцією прибутку держави яка залежить від норма оподаткування. Тому нам треба максимізувати $G(\tau)$. А для застосування методу треба її інвертувати тому, що метод шукає мінімум. Для кращого розуміння зобразимо початкову $G(\tau)$, а не інвертовану. Моделі (1-21) візьмо з лаб. 5. для них й проведемо оптимізацію.

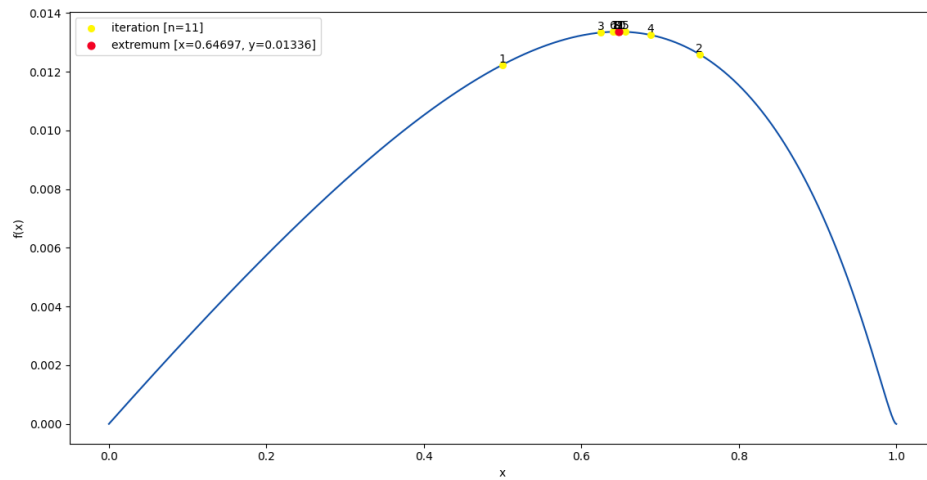


Рис. 2.4. Мінімізація функції

```
Optimization method: Dichotomy method
Initial parameters:
  f(x)
  l = 0.00000, r = 1.00000
iteration 0:    L = 0.50000    x = 0.50000    f(x) = -0.01224
iteration 1:    L = 0.25000    x = 0.75000    f(x) = -0.01260
iteration 2:    L = 0.12500    x = 0.62500    f(x) = -0.01334
iteration 3:    L = 0.06250    x = 0.68750    f(x) = -0.01326
iteration 4:    L = 0.03125    x = 0.65625    f(x) = -0.01336
iteration 5:    L = 0.01562    x = 0.64062    f(x) = -0.01336
iteration 6:    L = 0.00781    x = 0.64844    f(x) = -0.01336
iteration 7:    L = 0.00391    x = 0.64453    f(x) = -0.01336
iteration 8:    L = 0.00195    x = 0.64648    f(x) = -0.01336
iteration 9:    L = 0.00098    x = 0.64746    f(x) = -0.01336
iteration 10:   L = 0.00098    x = 0.64697    f(x) = -0.01336
Result:
  x = 0.64697
  f(x) = 0.01336
Function calls:20
Number of iterations:11
```

Рис. 2.5. Результат роботи

Як результат отримуємо, що норма оподаткування повинна дорівнювати $\tau = 0.647$, а $G(\tau) = 0.013$.

Глава 3

Висновок

У ході лабораторної роботи було вивчено метом дихотомії для зменшення інтервалу невизначеності унімодальної цільової функції та досліджено ефективність методу.

Глава 4

Код програми

4.1. main.py

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 from services.opt import *
5 from services.opt import GoldenSectionMethod
6 from services.model import EconomicModel, G_by_x
7
8
9 def task1():
10     f = lambda x: 2 - (1 / (np.log2(x**4 + 4*(x**3) + 29)
11         ))
12     function_text = "f(x) = 2 - (1 / (np.log2(x**4 + 4*(x
13         **3) + 29)))"
14     x0 = -1
15     eps = 0.01
16
17     dichotomy_method = DichotomyMethod()
18
19     step = 0.001
20     interval = dichotomy_method.get_interval(f, x0, step)
21     print(f'Sven method interval with {step}: {interval}',
22         )
23     dichotomy_method.is_plot = True
24     dichotomy_method(f, interval, eps, function_text=
25         function_text)
26
27
28 def task2():
29     model = EconomicModel()
30     profit = lambda x: G_by_x(model, x)
31
32     dichotomy_method = DichotomyMethod()
33     dichotomy_method.is_minimization = False
```

```

30     dichotomy_method.is_plot = True
31     dichotomy_method.is_log = True
32     dichotomy_method(profit, (0, 1), 0.001)
33
34
35 def main():
36     task1()
37     task2()
38
39
40 if __name__ == '__main__':
41     main()

```

4.2. model.py

```

1  import math
2  from scipy.optimize import fsolve
3  from collections.abc import Iterable
4
5
6  class EconomicModel(object):
7      def __init__(self):
8          self.set_default()
9
10     def set_default(self):
11         self.alpha = 0.5
12         self.beta = 1.5
13         self.gamma = 1.5
14         self.delta = 0.1
15         self.nu = 5
16         self.mu = 20
17         self.lambda_ = 20
18         self.rho = 10
19         self.A0 = 1
20         self.L0 = 1
21         self.D0 = 1
22         self.tau = 0.6
23         self.sigma = 0.5
24         self.theta = (1 + self.alpha * (self.beta - 1)) **
                        (-1)

```

```

25
26 def L1(self, x):
27     return x[3] * ((1 - self.alpha) * self.A0 * x[1] /
28         x[2]) ** (1 / self.alpha)
29
30 def Q1(self, x):
31     return self.A0 * x[3] ** self.alpha * self.L1(x)
32     ** (1 - self.alpha)
33
34 def D1(self, x):
35     return self.D0 * math.exp(-self.beta * x[1]) * x
36     [5] / (x[1] + x[5])
37
38 def S1(self, x):
39     return self.L0 * (1 - math.exp(-self.gamma * x[2])
40         ) * x[2] / (x[2] + x[6])
41
42 def I1(self, x):
43     return (1 - self.tau) * (1 - self.theta) * x[0]
44
45 def L2(self, x):
46     return x[7] * ((1 - self.alpha) * self.A0 * x[5] /
47         x[6]) ** (1 / self.alpha)
48
49 def Q2(self, x):
50     return self.A0 * x[7] ** self.alpha * self.L2(x)
51     ** (1 - self.alpha)
52
53 def D2(self, x):
54     return self.D0 * math.exp(-self.beta * x[5]) * x
55     [1] / (x[1] + x[5])
56
57 def S2(self, x):
58     return self.L0 * (1 - math.exp(-self.gamma * x[6])
59         ) * x[6] / (x[2] + x[6])
60
61 def I2(self, x):
62     return (1 - self.theta) * x[4]
63
64 def T(self, x):
65     return self.tau * x[0]

```

```

58
59 def G(self, x):
60     """ """
61     return (1 - self.sigma) * self.tau * x[0]
62
63 def G1(self, x):
64     """
65         """
66     return (1 - self.tau) * self.theta * x[0]
67
68 def G2(self, x):
69     """ """
70     return self.theta * x[4]
71
72 def count_model(self, x):
73     P1 = (x[1] * min(self.Q1(x), self.D1(x))\
74           - x[2] * min(self.L1(x), self.S1(x)) - x[0]) /
75           self.nu
76
77     p1 = (self.D1(x) - self.Q1(x)) / self.mu
78
79     w1 = (self.L1(x) - self.S1(x)) / self.lambda_
80
81     K1 = -self.delta * x[3] + self.I1(x)
82
83     P2 = (math.exp(-self.rho * self.sigma * self.T(x))
84           \
85           * x[5]\
86           * min(self.Q2(x), self.D2(x)) - x[6]\
87           * min(self.L2(x), self.S2(x)) - x[4]) / self.
88           nu
89
90     p2 = (self.D2(x) - self.Q2(x)) / self.mu
91
92     w2 = (self.L2(x) - self.S2(x)) / self.lambda_
93
94     K2 = -self.delta * x[7] + self.I2(x)
95
96     result = [P1, p1, w1, K1, P2, p2, w2, K2]
97     return result

```

```

95     def __call__(self, x, changed_params={}):
96         for param_name in changed_params:
97             if not hasattr(self, param_name):
98                 raise ValueError(param_name)
99             setattr(self, param_name, changed_params[
100                 param_name])
101
102         return self.count_model(x)
103
104 def G_by_x(model: EconomicModel, tau: float, init: list[
105     float] | None = None) -> float:
106     if isinstance(tau, Iterable):
107         return list(map(lambda x: G_by_x(model, x), list(
108             tau)))
109
110     prepared_model_call = lambda x, *args: model(x,
111         changed_params={"tau": args[0]})
112
113     #
114
115     model.set_default()
116
117     #
118     if init is None:
119         init = [0, 0.5, 0.25, 0.1, 0, 0.5, 0.25, 0.1]
120     new_x = fsolve(prepared_model_call, x0=init, args=(tau
121         ,))
122
123     model.tau = tau
124     return model.G(new_x)

```

4.3. opt.py

```

1 import copy
2 import numpy as np
3 import math
4 import matplotlib.pyplot as plt
5 from colorama import init as colorama_init
6 colorama_init()

```

```

7 from colorama import Fore, Back, Style
8
9
10 class ABSOptimizationMethod(object):
11     method_name = 'optimization_method'
12
13     def __init__(self):
14         self.function_text = 'f(x)'
15         self.is_plot = False
16         self.is_log = True
17         self.is_minimization = True
18         self.log_iteration_like_table = True
19
20     def __call__(self, func, bounds, *args, function_text=
21         None, **kwargs):
22         """
23                                     func
24                                     numpy"""
25         self._log_start(function_text or self.
26             function_text, *bounds)
27
28         # Decorate func to count number of calls
29         optimiation_func = copy.copy(func)
30         optimiation_func = self.
31             optimization_type_function_decorator(
32                 optimiation_func)
33         optimiation_func = self.function_calls_count(
34             optimiation_func)
35
36         result, iterations, *other = self.
37             optimization_method(optimiation_func, bounds, *
38                 args, **kwargs)
39
40         self._log_result(result, func(result))
41         self._log_number_of_iterations(len(iterations))
42
43         if self.is_plot:
44             self._plot_it(func, iterations, *bounds)
45
46         return result, iterations, *other

```

```
39 def get_interval(self, func, x0, step=0.5):
40     """Sven method"""
41
42     f_l = func(x0 - step)
43     f_r = func(x0 + step)
44     f = func(x0)
45
46     if f_l >= f and f < f_r:
47         return (x0 - step, x0 + step)
48
49     if f_l < f < f_r:
50         step = -step
51
52     p = 1
53     while True:
54         f_new = func(x0 + 2**p * step)
55
56         if f_new >= f:
57             a = x0 + 2**(p - 2) * step
58             b = x0 + 2**p * step
59
60             return tuple(sorted([a, b]))
61
62         f = f_new
63         p += 1
64
65     def function_calls_count(self, func):
66         self.function_calls = 0
67
68         def wrap(*args, ignore_call=False, **kwargs):
69             if not ignore_call:
70                 self.function_calls += 1
71             return func(*args, **kwargs)
72
73         return wrap
74
75     def optimization_type_function_decorator(self, func):
76         def wrap(*args, ignore_call=False, **kwargs):
77             result = func(*args, **kwargs)
78             if not self.is_minimization:
79                 return -result
```

```

80         return result
81
82
83     return wrap
84
85 @staticmethod
86 def _log_decorator(func):
87     def wrap(self, *args, **kwargs):
88         if not self.is_log:
89             return
90
91         func(self, *args, **kwargs)
92
93     return wrap
94
95 @_log_decorator
96 def _log_iteration(self, idx, x, y, L):
97     if self.log_iteration_like_table:
98         print(Fore.GREEN + f'iteration_{idx}:' + Style
99               .RESET_ALL, end='')
100         print(f'\tL={L:.5f}', end='')
101         print(f'\tx={x:.5f}', end='')
102         print(f'\tf(x)={y:.5f}', end='')
103         print()
104         return
105
106     print(Fore.GREEN + f'iteration_{idx}:' + Style.
107           RESET_ALL)
108     print(f'\tL={L:.5f}')
109     print(f'\tx={x:.5f}')
110     print(f'\tf(x)={y:.5f}')
111
112 @_log_decorator
113 def _log_start(self, func_text, l, r):
114     print(Fore.RED + 'Optimization_method:' + self.
115           method_name + Style.RESET_ALL)
116     print(Fore.GREEN + 'Initial_parameters:' + Style.
117           RESET_ALL)
118     print(f'\t{func_text}')
119     print(f'\tl={l:.5f}, r={r:.5f}')

```



```

117 @_log_decorator
118 def _log_result(self, x, y):
119     print(Fore.GREEN + 'Result:' + Style.RESET_ALL)
120     print(f'\tx={x:.5f}')
121     print(f'\tf(x)={y:.5f}')
122     print(Fore.GREEN + 'Function_calls:' + Style.
123           RESET_ALL + str(self.function_calls))
124
125 @_log_decorator
126 def _log_number_of_iterations(self, iters_num):
127     print(Fore.GREEN + 'Number_of_iterations:' + Style
128           .RESET_ALL + str(iters_num))
129
130 def _plot_it(self, func, iterations, l, r, steps =
131             1000):
132     color_function = '#1050A8'
133     color_iteration = '#FFF702'
134     color_extremum = '#F30223'
135
136     x = np.linspace(l, r, steps)
137     # y = list(map(func, x))
138     y = func(x)
139     iterations_y = list(map(func, iterations))
140
141     plt.xlabel('x')
142     plt.ylabel('f(x)')
143
144     #
145     plt.plot(x, y, color=color_function, zorder=0)
146
147     #
148     plt.scatter(
149         iterations, iterations_y,
150         label=f'iteration_{n={len(iterations)}},
151         color=color_iteration,
152         zorder=1,
153         s=30,
154     )
155
156     #
157     plt.scatter(

```

```

155         [iterations[-1]], [iterations_y[-1]],
156         label=f'extremum_{x={iterations[-1]:.5f},_y={
157             iterations_y[-1]:.5f}}',
158         color=color_extremum,
159         zorder=1,
160         s=40,
161     )
162
163     #
164
165     for i in range(len(iterations)):
166         plt.annotate(str(i + 1), (iterations[i],
167             iterations_y[i]), ha='center', va='bottom')
168
169     plt.legend()
170     plt.show()
171
172     def optimization_method(self):
173         ...
174
175 class DichotomyMethod(ABSOptimizationMethod):
176     method_name = 'Dichotomy_method'
177
178     def optimization_method(self, func, bounds, eps):
179         l, r = bounds
180
181         iterations = []
182         idx = 0
183
184         while r - l >= eps:
185             temp_x = (l + r) / 2
186             x1 = temp_x - eps / 2
187             x2 = temp_x + eps / 2
188
189             if func(x1) > func(x2):
190                 l = temp_x
191             else:
192                 r = temp_x

```

```

192         self._log_iteration(idx, temp_x, func(temp_x,
193             ignore_call=True), (r - 1))
194
195         iterations.append(temp_x)
196         idx += 1
197
198         result = (1 + r) / 2
199         iterations.append(result)
200
201         #
202
203         self._log_iteration(idx, result, func(result,
204             ignore_call=True), (r - 1))
205
206         return result, iterations
207
208 phi = golden_ratio = (1 + math.sqrt(5)) / 2
209
210 class GoldenSectionMethod(ABSOptimizationMethod):
211     method_name = 'Golden_section_method'
212
213     def optimization_method(self, func, bounds, eps,
214         max_iter=100):
215         l, r = bounds
216         iterations = []
217         d = (r - l)
218         ind = 0
219         interval_changes = 0
220
221         while (r - l) >= eps:
222             d = d / phi
223             x1 = r - d
224             x2 = l + d
225
226             temp_x = (1 + r) / 2
227             self._log_iteration(ind, temp_x, func(temp_x,
228                 ignore_call=True), (r - 1))
229
230             if func(x1) <= func(x2):
231                 r = x2

```

```

228         else:
229             l = x1
230
231             iterations.append(temp_x)
232             interval_changes += 1
233             ind += 1
234
235         result = (l + r) / 2
236
237         return result, iterations, interval_changes, self.
            function_calls
238
239
240 class FibonacciMethod(ABSOptimizationMethod):
241     method_name = 'Fibonacci_method'
242
243     @staticmethod
244     def fib(n):
245         a = 0
246         b = 1
247
248         if n == 0:
249             return a
250
251         if n == 1:
252             return b
253
254         for i in range(1, n):
255             a, b = b, a + b
256
257         return b
258
259     def optimization_method(self, func, bounds, eps):
260         fib = self.fib
261         l, r = bounds
262         iterations = []
263         ind = 0
264         interval_changes = 0
265
266         n = 0
267         while fib(n) <= (r - l) / (eps):

```

```
268         n += 1
269
270     n = max(n, 3)
271
272     for k in range(n - 2):
273         p = (fib(n - k - 1) / fib(n - k))
274         x1 = 1 + (r - 1) * (1 - p)
275         x2 = 1 + (r - 1) * p
276
277         temp_x = (1 + r) / 2
278         self._log_iteration(ind, temp_x, func(temp_x,
279                                     ignore_call=True), (r - 1))
280
281         if func(x1) <= func(x2):
282             r = x2
283         else:
284             l = x1
285
286         iterations.append(temp_x)
287         interval_changes += 1
288         ind += 1
289
290     result = (1 + r) / 2
291
292     return result, iterations, interval_changes, self.
293         function_calls
```