МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»
Кафедра КМАД

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №8

Виконав студент:
Омельніцький Андрій Миколайович
Група: КН-120А

Перевірила:
Ольга Василівна Костюк

Харків, 2023 г.

Зміст

Глава 1

Мета роботи

Вивчення методу золотого перетину 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 золотого перетину 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) віконати її мінімізацію методом золотого перетину та змоделювати процес мінімізації.

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

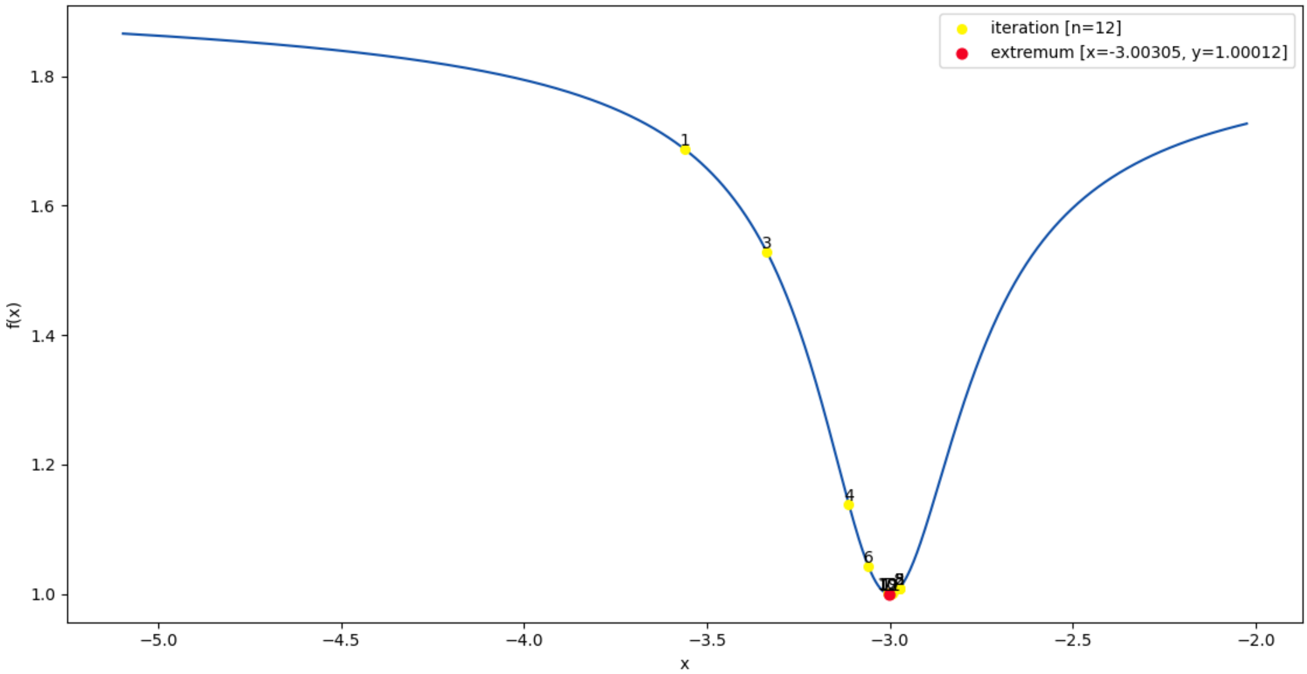

Рис. 2.1. Мінімізація функції

За результатом роботи програми отримаємо 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, знаяення x та значення f(x).

```
Sven method interval with step=0.001: (-5.096, -2.024)
Optimization method: Golden section method
Initial parameters:
        f(x)
        l = -5.09600, r = -2.02400
iteration 0:    L = 3.07200     x = -3.56000    f(x) = 1.68686
iteration 1:    L = 1.89860     x = -2.97330    f(x) = 1.00904
iteration 2:    L = 1.17340     x = -3.33590    f(x) = 1.52830
iteration 3:    L = 0.72520     x = -3.11180    f(x) = 1.13877
iteration 4:    L = 0.44820     x = -2.97330    f(x) = 1.00904
iteration 5:    L = 0.27700     x = -3.05890    f(x) = 1.04352
iteration 6:    L = 0.17120     x = -3.00600    f(x) = 1.00047
iteration 7:    L = 0.10581     x = -2.97330    f(x) = 1.00904
iteration 8:    L = 0.06539     x = -2.99351    f(x) = 1.00055
iteration 9:    L = 0.04041     x = -3.00600    f(x) = 1.00047
iteration 10:   L = 0.02498     x = -2.99828    f(x) = 1.00004
iteration 11:   L = 0.01544     x = -3.00305    f(x) = 1.00012
Result:
        x = -3.00010
        f(x) = 1.00000
Function calls:24
Number of iterations:12
```

Рис. 2.2. Результат роботи

2.2. Пункт 5

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

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

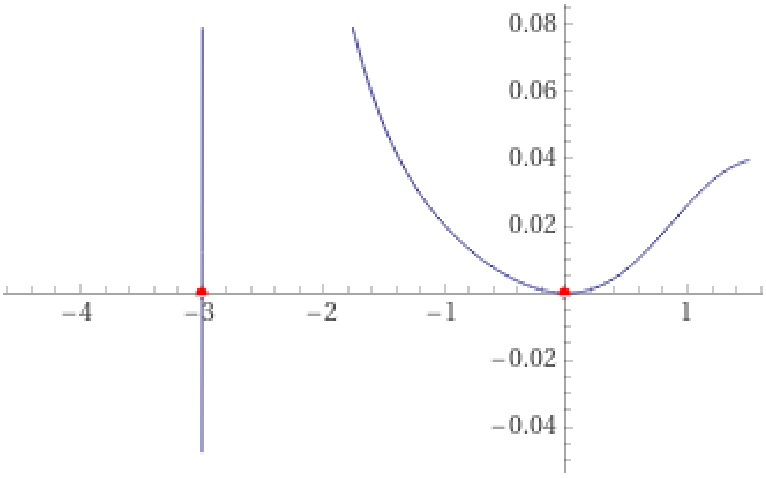

Рис. 2.3. Графік похідної

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 матоду треба її інвертувати тому, що метод шукає мінімум. Для кращого розуміння зобразимо початкову $G(\tau)$, а не інвертовану. Моделі (1-21) візмемо з лаб. 5. для них й проведемо оптимізацію.

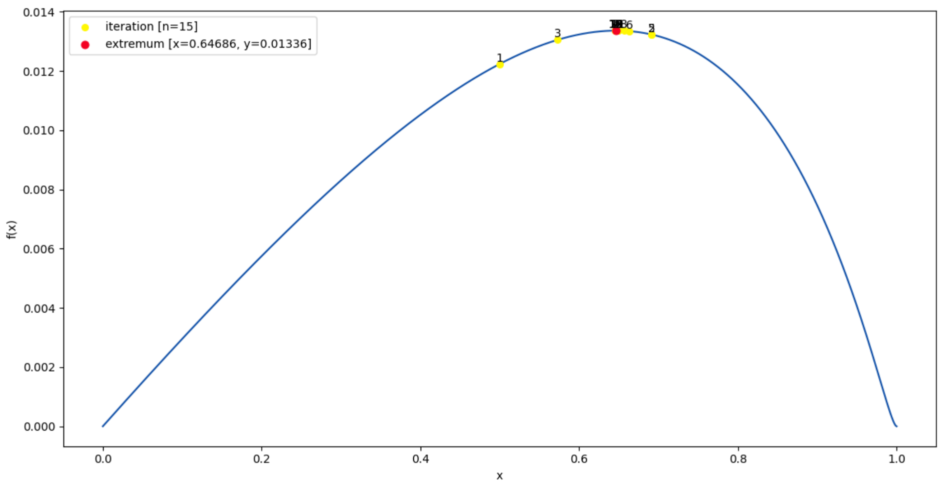

Рис. 2.4. Мінімізація функції

```
Optimization method: Golden section method
Initial parameters:
        f(x)
        l = 0.00000, r = 1.00000
iteration 0:    L = 1.00000     x = 0.50000     f(x) = -0.01224
iteration 1:    L = 0.61803     x = 0.69098     f(x) = -0.01324
iteration 2:    L = 0.38197     x = 0.57295     f(x) = -0.01305
iteration 3:    L = 0.23607     x = 0.64590     f(x) = -0.01336
iteration 4:    L = 0.14590     x = 0.69098     f(x) = -0.01324
iteration 5:    L = 0.09017     x = 0.66312     f(x) = -0.01335
iteration 6:    L = 0.05573     x = 0.64590     f(x) = -0.01336
iteration 7:    L = 0.03444     x = 0.65654     f(x) = -0.01336
iteration 8:    L = 0.02129     x = 0.64996     f(x) = -0.01336
iteration 9:    L = 0.01316     x = 0.64590     f(x) = -0.01336
iteration 10:   L = 0.00813     x = 0.64841     f(x) = -0.01336
iteration 11:   L = 0.00502     x = 0.64686     f(x) = -0.01336
iteration 12:   L = 0.00311     x = 0.64590     f(x) = -0.01336
iteration 13:   L = 0.00192     x = 0.64649     f(x) = -0.01336
iteration 14:   L = 0.00119     x = 0.64686     f(x) = -0.01336
Result:
        x = 0.64663
        f(x) = 0.01336
Function calls:30
Number of iterations:15
```

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 вивченно метом золотого перетину для зменшення інтервалу невизначеності унімодальної цільової функції та дослідженно ефективность методу.

Глава 4

Код програми

4.1. main.py

```
import matplotlib.pyplot as plt
import numpy as np

from services.opt import *
from services.opt import GoldenSectionMethod
from services.model import EconomicModel, G_by_x


def task1():
    f = lambda x:  2 - (1 / (np.log2(x**4 + 4*(x**3) + 29)
      ))
    x0 = -1
    eps = 0.01

    golden_section_method = GoldenSectionMethod()

    step = 0.001
    interval = golden_section_method.get_interval(f, x0,
      step)
    print(f'Sven method interval with {step=}: {interval}'
      )
    golden_section_method.is_plot = True
    golden_section_method(f, interval, eps)


def task2():
    model = EconomicModel()
    profit = lambda x: G_by_x(model, x)

    golden_section_method = GoldenSectionMethod()
    golden_section_method.is_minimization = False
    golden_section_method.is_plot = True
    golden_section_method.is_log = True
```

```
    golden_section_method(profit, (0, 1), 0.001)


def main():
    task1()
    task2()


if __name__ == '__main__':
    main()
```

4.2. model.py

```
import math
from scipy.optimize import fsolve
from collections.abc import Iterable


class EconomicModel(object):
    def __init__(self):
        self.set_default()

    def set_default(self):
        self.alpha = 0.5
        self.beta = 1.5
        self.gamma = 1.5
        self.delta = 0.1
        self.nu = 5
        self.mu = 20
        self.lambda_ = 20
        self.rho = 10
        self.A0 = 1
        self.L0 = 1
        self.D0 = 1
        self.tau = 0.6
        self.sigma = 0.5
        self.theta = (1 + self.alpha * (self.beta - 1)) ** (-1)

    def L1(self, x):
```

```
        return x[3] * ((1 - self.alpha) * self.A0 * x[1] / x[2]) ** (1 / self.alpha)

    def Q1(self, x):
        return self.A0 * x[3] ** self.alpha * self.L1(x) ** (1 - self.alpha)

    def D1(self, x):
        return self.D0 * math.exp(-self.beta * x[1]) * x[5] / (x[1] + x[5])

    def S1(self, x):
        return self.L0 * (1 - math.exp(-self.gamma * x[2])) * x[2] / (x[2] + x[6])

    def I1(self, x):
        return (1 - self.tau) * (1 - self.theta) * x[0]

    def L2(self, x):
        return x[7] * ((1 - self.alpha) * self.A0 * x[5] / x[6]) ** (1 / self.alpha)

    def Q2(self, x):
        return self.A0 * x[7] ** self.alpha * self.L2(x) ** (1 - self.alpha)

    def D2(self, x):
        return self.D0 * math.exp(-self.beta * x[5]) * x[1] / (x[1] + x[5])

    def S2(self, x):
        return self.L0 * (1 - math.exp(-self.gamma * x[6])) * x[6] / (x[2] + x[6])

    def I2(self, x):
        return (1 - self.theta) * x[4]

    def T(self, x):
        return self.tau * x[0]

    def G(self, x):
```

```
        """                                  """
        return (1 - self.sigma) * self.tau * x[0]

    def G1(self, x):
        """
          """
        return (1 - self.tau) * self.theta * x[0]

    def G2(self, x):
        """                                           """
        return self.theta * x[4]

    def count_model(self, x):
        P1 = (x[1] * min(self.Q1(x), self.D1(x))\
            - x[2] * min(self.L1(x), self.S1(x)) - x[0]) / self.nu

        p1 = (self.D1(x) - self.Q1(x)) / self.mu

        w1 = (self.L1(x) - self.S1(x)) / self.lambda_

        K1 = -self.delta * x[3] + self.I1(x)

        P2 = (math.exp(-self.rho * self.sigma * self.T(x)) \
            * x[5]\
            * min(self.Q2(x), self.D2(x)) - x[6]\
            * min(self.L2(x), self.S2(x)) - x[4]) / self.nu

        p2 = (self.D2(x) - self.Q2(x)) / self.mu

        w2 = (self.L2(x) - self.S2(x)) / self.lambda_

        K2 = -self.delta * x[7] + self.I2(x)

        result = [P1, p1, w1, K1, P2, p2, w2, K2]
        return result

    def __call__(self, x, changed_params={}):
        for param_name in changed_params:
```

```
                if not hasattr(self, param_name):
                    raise ValueError(param_name)
                setattr(self, param_name, changed_params[
                  param_name])

        return self.count_model(x)


def G_by_x(model: EconomicModel, tau: float, init: list[
    float] | None = None) -> float:
    if isinstance(tau, Iterable):
        return list(map(lambda x: G_by_x(model, x), list(
          tau)))

    prepared_model_call = lambda x, *args: model(x,
      changed_params={"tau": args[0]})

    #

    model.set_default()

    #
    if init is None:
        init = [0, 0.5, 0.25, 0.1, 0, 0.5, 0.25, 0.1]
    new_x = fsolve(prepared_model_call, x0=init, args=(tau
      ,))

    model.tau = tau
    return model.G(new_x)
```

4.3. opt.py

```
import copy
import numpy as np
import math
import matplotlib.pyplot as plt
from colorama import init as colorama_init
colorama_init()
from colorama import Fore, Back, Style

```

```

class ABSOptimizationMethod(object):
    method_name = 'optimization method'

    def __init__(self):
        self.function_text = 'f(x)'
        self.is_plot = False
        self.is_log = True
        self.is_minimization = True
        self.log_iteration_like_table = True

    def __call__(self, func, bounds, *args, function_text=None, **kwargs):
        """                 func

                                        numpy"""
        self._log_start(function_text or self.function_text, *bounds)

        # Decorate func to count number of calls
        optimiation_func = copy.copy(func)
        optimiation_func = self.optimization_type_function_decorator(optimiation_func)
        optimiation_func = self.function_calls_count(optimiation_func)

        result, iterations, *other = self.optimization_method(optimiation_func, bounds, *args, **kwargs)

        self._log_result(result, func(result))
        self._log_number_of_iterations(len(iterations))

        if self.is_plot:
            self._plot_it(func, iterations, *bounds)

        return result, iterations, *other

    def get_interval(self, func, x0, step=0.5):
        """Sven method"""
```

```

        f_l = func(x0 - step)
        f_r = func(x0 + step)
        f = func(x0)

        if f_l >= f and f < f_r:
            return (x0 - step, x0 + step)

        if f_l < f < f_r:
            step = -step

        p = 1
        while True:
            f_new = func(x0 + 2**p * step)

            if f_new >= f:
                a = x0 + 2**(p - 2) * step
                b = x0 + 2**p * step

                return tuple(sorted([a, b]))

            f = f_new
            p += 1

    def function_calls_count(self, func):
        self.function_calls = 0

        def wrap(*args, ignore_call=False, **kwargs):
            if not ignore_call:
                self.function_calls += 1
            return func(*args, **kwargs)

        return wrap

    def optimization_type_function_decorator(self, func):
        def wrap(*args, ignore_call=False, **kwargs):
            result = func(*args, **kwargs)
            if not self.is_minimization:
                return -result

            return result
```

```

        return wrap

    @staticmethod
    def _log_decorator(func):
        def wrap(self, *args, **kwargs):
            if not self.is_log:
                return

            func(self, *args, **kwargs)

        return wrap

    @_log_decorator
    def _log_iteration(self, idx, x, y, L):
        if self.log_iteration_like_table:
            print(Fore.GREEN + f'iteration {idx}:' + Style.RESET_ALL, end='')
            print(f'\tL = {L:.5f}', end='')
            print(f'\tx = {x:.5f}', end='')
            print(f'\tf(x) = {y:.5f}', end='')
            print()
            return

        print(Fore.GREEN + f'iteration {idx}:' + Style.RESET_ALL)
        print(f'\tL = {L:.5f}')
        print(f'\tx = {x:.5f}')
        print(f'\tf(x) = {y:.5f}')

    @_log_decorator
    def _log_start(self, func_text, l, r):
        print(Fore.RED + 'Optimization method: ' + self.method_name + Style.RESET_ALL)
        print(Fore.GREEN + 'Initial parameters:' + Style.RESET_ALL)
        print(f'\t{func_text}')
        print(f'\tl = {l:.5f}, r = {r:.5f}')

    @_log_decorator
    def _log_result(self, x, y):
```

```
        print(Fore.GREEN + 'Result:' + Style.RESET_ALL)
        print(f'\tx = {x:.5f}')
        print(f'\tf(x) = {y:.5f}')
        print(Fore.GREEN + 'Function calls:' + Style.RESET_ALL + str(self.function_calls))

    @_log_decorator
    def _log_number_of_iterations(self, iters_num):
        print(Fore.GREEN + 'Number of iterations:' + Style.RESET_ALL + str(iters_num))

    def _plot_it(self, func, iterations, l, r, steps = 1000):
        color_function = '#1050A8'
        color_iteration = '#FFF702'
        color_extremum = '#F30223'

        x = np.linspace(l, r, steps)
        # y = list(map(func, x))
        y = func(x)
        iterations_y = list(map(func, iterations))

        plt.xlabel('x')
        plt.ylabel('f(x)')

        #
        plt.plot(x, y, color=color_function, zorder=0)

        #
        plt.scatter(
            iterations, iterations_y,
            label=f'iteration [n={len(iterations)}]',
            color=color_iteration,
            zorder=1,
            s=30,
        )

        #
        plt.scatter(
            [iterations[-1]], [iterations_y[-1]],
```

```
                label=f'extremum␣[x={iterations[-1]:.5f},␣y={
                    iterations_y[-1]:.5f}]',
                color=color_extremum,
                zorder=1,
                s=40,
            )

            #

            for i in range(len(iterations)):
                plt.annotate(str(i + 1), (iterations[i],
                    iterations_y[i]), ha='center', va='bottom')

            plt.legend()
            plt.show()

    def optimization_method(self):
        ...


class DichotomyMethod(ABSOptimizationMethod):
    method_name = 'Dichotomy␣method'

    def optimization_method(self, func, bounds, eps):
        l, r = bounds

        iterations = []
        idx = 0

        while r - l >= eps:
            temp_x = (l + r) / 2
            x1 = temp_x - eps / 2
            x2 = temp_x + eps / 2

            if func(x1) > func(x2):
                l = temp_x
            else:
                r = temp_x

```

```
            self._log_iteration(idx, temp_x, func(temp_x, ignore_call=True), (r - l))

            iterations.append(temp_x)
            idx += 1

        result = (l + r) / 2
        iterations.append(result)

        #
        self._log_iteration(idx, result, func(result, ignore_call=True), (r - l))

        return result, iterations


phi = golden_ratio = (1 + math.sqrt(5)) / 2

class GoldenSectionMethod(ABSOptimizationMethod):
    method_name = 'Golden section method'

    def optimization_method(self, func, bounds, eps, max_iter=100):
        l, r = bounds
        iterations = []
        d = (r - l)
        ind = 0
        interval_changes = 0

        while (r - l) >= eps:
            d = d / phi
            x1 = r - d
            x2 = l + d

            temp_x = (l + r) / 2
            self._log_iteration(ind, temp_x, func(temp_x, ignore_call=True), (r - l))

            if func(x1) <= func(x2):
                r = x2
```

```
            else:
                l = x1

            iterations.append(temp_x)
            interval_changes += 1
            ind += 1

        result = (l + r) / 2

        return result, iterations, interval_changes, self.
            function_calls


class FibonacciMethod(ABSOptimizationMethod):
    method_name = 'Fibonacci␣method'

    @staticmethod
    def fib(n):
        a = 0
        b = 1

        if n == 0:
            return a

        if n == 1:
            return b

        for i in range(1, n):
            a, b = b, a + b

        return b

    def optimization_method(self, func, bounds, eps):
        fib = self.fib
        l, r = bounds
        iterations = []
        ind = 0
        interval_changes = 0

        n = 0
        while fib(n) <= (r - l) / (eps):
```

```
            n += 1

        n = max(n, 3)

        for k in range(n - 2):
            p = (fib(n - k - 1) / fib(n - k))
            x1 = l + (r - l) * (1 - p)
            x2 = l + (r - l) * p

            temp_x = (l + r) / 2
            self._log_iteration(ind, temp_x, func(temp_x, ignore_call=True), (r - l))

            if func(x1) <= func(x2):
                r = x2
            else:
                l = x1

            iterations.append(temp_x)
            interval_changes += 1
            ind += 1

        result = (l + r) / 2

        return result, iterations, interval_changes, self.function_calls
```