НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Курсова робота

із дисципліни «Дослідження операцій»

на тему: Дослідження методу Бройдена-Флетчера-Шенно

Студент: Шумель С. О. групи КМ-62

Керівник: ст.вик. Ладогубець Т. С.

Кількість балів:_______

Оцінка:_____________

Київ — 2019

ЗМІСТ

ОСНОВНА ЧАСТИНА 2

- Постановка задачі 2
- Теоретична частина 3
- Практична частина 6

ВИСНОВОК 12

Література 14

Додаток А Таблиця “Порівняльні характеристики дослідження” 15

Додаток Б. Графік пошуку мінімума (ДСК) 16

Додаток В. Графік пошуку мінімума (ЗП) 17

Додаток Г. Лістінг програми 18

ОСНОВНА ЧАСТИНА

1. Постановка задачі

Дослідити збіжність методу Бройдена-Флетчера-Шенно для функції Розенброка

$$f_1(x) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2$$

з початкової точки x0 = [-1.2, 0] в залежності від:

1. виду методу одновимірного пошуку для обчислення величини кроку
2. точності методу одновимірного пошуку для обчислення величини кроку
3. виду критерія закінчення пошуку
4. виду різницевої схеми обчислень похідної функції
5. величини прирощення при чисельному обчисленні похідної функції
6. кількості рестартів

Теоретично дослідити збіжність даного методу при вирішенні задачі умовної оптимізації методом штрафних функцій.

2. Теоретична частина

Задача безумовної оптимізації має наступний вигляд:

$$minf(x)$$

де x [x 1 ,..., x n] - вектор варіюємих параметрів, f(x) – цільова функція (в даному випадку - функція Розенброка).

В даній роботі досліджуються методи змінної метрики або квазін'ютоновські методи.

Квазін'ютоновські методи є ітераційними процедурами мінімізації та основані на властивостях квадратичних функцій. В усіх методах зазначеного класу алгоритми мінімізації базуються на використанні ітераційної формули:

$$x^{k+1} = x^k + \lambda^k \widehat{S}^k \qquad k = 0,1,\dots,$$

де S^k - напрямок пошуку, λ^k - довжина кроку у вибраному напрямку пошуку.

Довжина кроку обчислюється за допомогою методів одновимірного пошуку: методу "золотого претину", методу ДСК-Пауелла. Метод золотого перетину відноситься до методів виключення інтревалів, а ДСК-Пауелла використовує квадратичну апроксимацію. Перед використанням зазначених методів необхідно знайти інтервал невизначеності за допомогою рекурентної формули Свенна:

$$\lambda_{k+1} = \lambda_k \pm \Delta\lambda \cdot 2^k, \quad k = 0,1,\dots.$$

Для обчислення довжини кроку в алгоритмі Свенна використовується формула:

$$\Delta\lambda = \alpha\left(\frac{\|x^{(k)}\|}{\|s^{(k)}\|}\right),$$

Ефективність пошуку залежить від довжини крому, тому необхідно коригувати значення шляхом зміни параметра α.

Квазін'ютоновскі методи характерні знаходженням напрямку зменшення функції, який визначається за формулою:

$$s\,(x^{(k)}) = -\,\mathbf{A}^{(k)}\nabla f\,(x^{(k)}),$$

В даній формулі матриця А - це метрика, яка є апроксимацією зворотної матриці Гессе. Методи пошуку вздовж напрямків, визначених цією формулою, називаються методами змінної метрики, оскільки матриця А змінюється на кожній ітерації.

Необхідно побудувати матрицю А таким чином, щоб послідовність A_1 , A_2, … A_k була наближенням до матриці Гессе. Вибір А визначає конкретний метод змінної метрики. Для забезпечення збіжності матриця A_{k+1} повинна бути позитивно визначена.

Метод Бройдена – Флетчера – Шенно реалізується за допомогою рекурентної формули при обчисленні метрики:

$$A^{(k)} = A^{(k-1)} + \frac{\Delta x^{(k-1)}\Delta x^{(k-1)^{\mathrm{T}}}}{\Delta x^{(k-1)^{\mathrm{T}}}\Delta g^{(k-1)}} - \frac{A^{(k-1)}\Delta g^{(k-1)}\Delta g^{(k-1)^{\mathrm{T}}}A^{(k-1)}}{\Delta g^{(k-1)^{\mathrm{T}}}A^{(k-1)}\Delta g^{(k-1)}}\,.$$

Головною перевагою даного методу є слабка залежність від точності одномірного пошуку, а також обов’язкова необхідність повернення до початкової ітерації алгоритму. Було встановлена властивість квадратичної збіжності цих методів при відстутності одновимірного пошуку. Даний метод в умовах квадратичної збіжності та відсутності трудомістких операцій одновимірного пошуку є стійким та швидкодійним при розв’язані задач з функціями загального вигляду.

Для даних методів властиві рестарти, тобто повернення на початкову ітерацію. Необхідність рестартів можна пояснити шляхом аналізу обумовленості зворотніх матриць. Але потрібно розуміти, що при збільшені стікості алгоритмів змінної метрики рестарти уповільнюють процес наближення до точки оптимума.

3. Практична частина

В даній роботі проведено дослідження збіжності методу Бройдена – Флетчера – Шенно на основі функції Розенброка.

$$f_1(x) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2$$

Нижче на Графіку 3.1 наведена дана функція в тривимірному просторі.

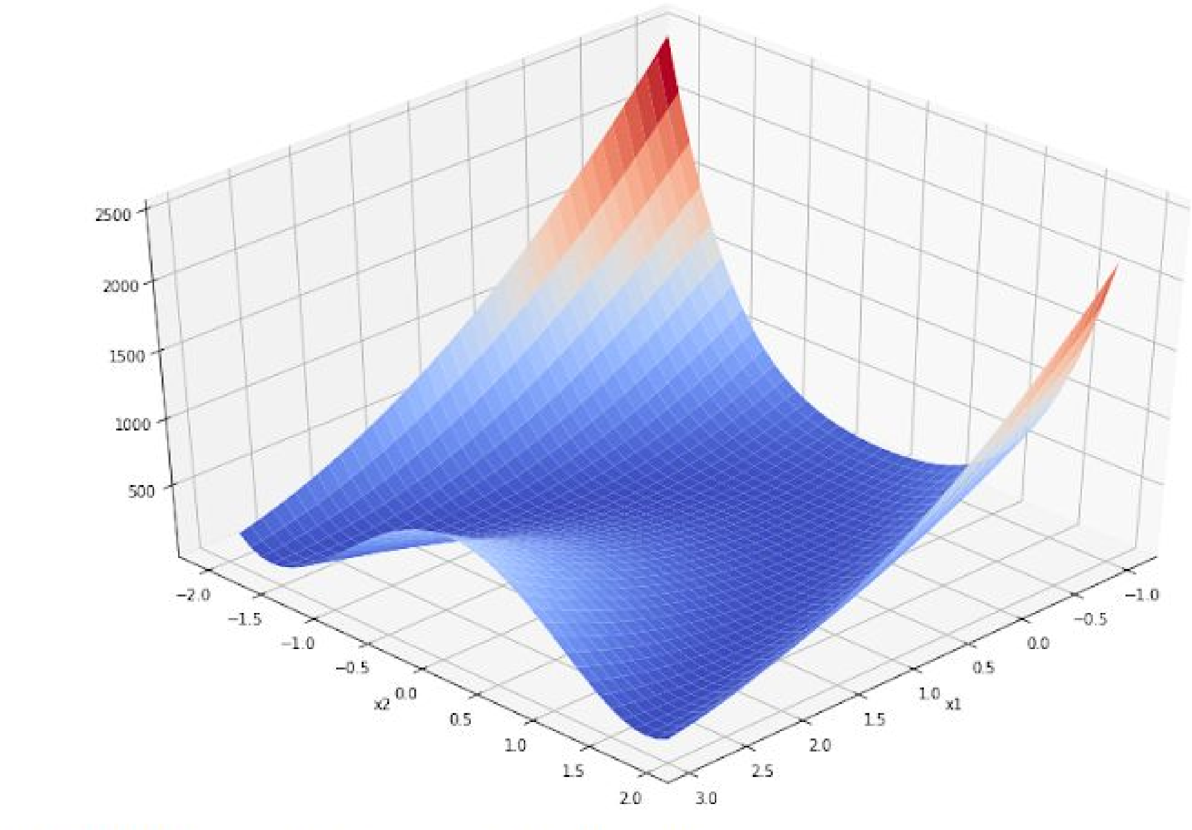
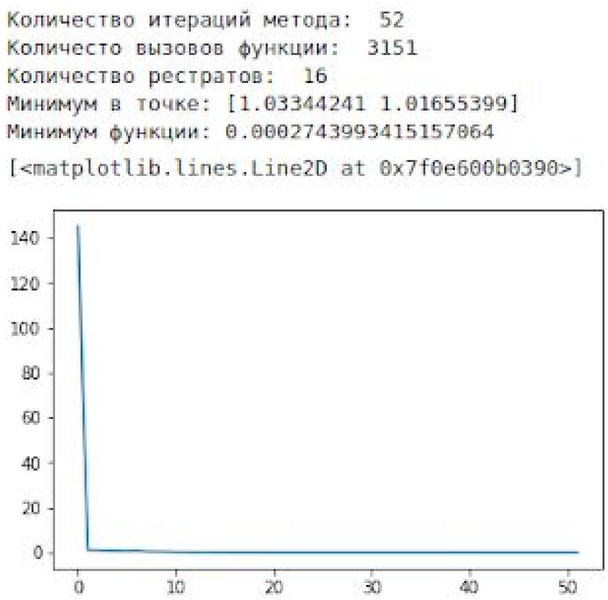
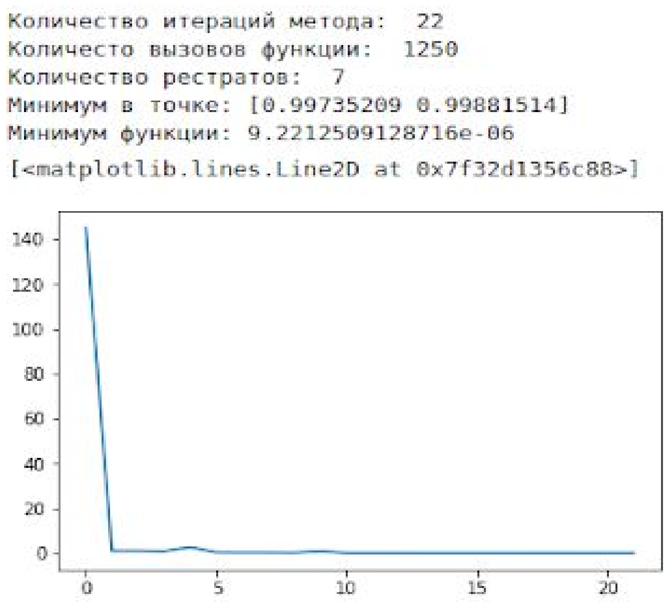


График 3.1. Графік функції Розенброка

Відомо, що мінімум функції дорівнює 0 та знаходиться в точці x = [1, 1]. Результати повного дослідження у вигляді таблиці наведено в Додатку 1.

Початковим завдання дослідження збіжності було порівняти метод при різних способах одновимірного пошуку. Порівняння методів ЗП та ДСК було проведено на основі центральної різницевої схеми для обчислення похідних з кроком h = 0.0000001 та критерію закінчення по модулю градієнта. Для знаходження оптимального λ спочатку було визначено інтервал невизначеності. Як було викладено теоретично, ефективність МОП залежить від визначення довжини кроку $\Delta\lambda$ в алгоритмі Свена. При обчисленні довжини кроку $\Delta\lambda$ методом підбору було встановлено, що при коефіцієнті $\alpha = 0.1$ для ДСК-Пауелла пошук відбувається ефективніше. Нижче на Рисунку 3.1 та Рисунку 3.2 наведено результати програми при $\alpha = 0.1$ та $\alpha = 0.5$. Зауважу, що на цих рисунках також наведені графіки залежності величини функції від номера ітерації.





З цих міркувань робимо висновок, що ефективність способу сильно залежить від довжини кроку в алгоритмі Свенна. Варто зауважити, що для ЗП $\alpha = 0.5$ буде ефективніше.

Нижче в таблиці наведені порівняльні характеристики обчислень при ЗП та ДСК. Також в Додатку 2 та Додатку 3 продемонстровано графіки пошуку мінімуму функції.

МОП	Точність МОП	Вид різницевої схеми похідної	Крок h	Точність методу	Критерій закінчення	Кіл-ть рестрартів	Кіл-сть ітерацій	Кіл-сть викликів функції	Точка мінімума	Значення функції
ЗП	0.001	центральна	0.0000001	0.001	градієнт	4	17	1305	[1.00000001 0.99999996]	1.0383086136980672e-12
ДСК	0.001	центральна	0.0000001	0.001	градієнт	7	22	1250	[0.99735209 0.99881514]	9.2212509128716e-06

На основі цих досліджень можна побачити, кількість ітерацій методу ЗП менша, проте кількість викликів функцій більша. Ці два параметри компенсують одне одного, тому однозначно визначити який методом більш ефективний неможливо.

При проведені досліджень впливу точності МОП було встановлено, що вона не впливає на точність методу. Це можна побачити в таблиці, яку наведно нижче. Дослідження проводились на основі ДСК методу при центральній різницевій схемі. Таким чином, було доведено теоретичні викладки слабкої залежності від кроку в квазіньютоновських методах.

МОП	Точність МОП	Вид різницевої схеми похідної	Крок h	Точність методу	Критерій закінчення	Кіл-сть рестартів	Кіл-сть ітерацій	Кіл-сть викликів функції	Точка мінімума	Значення функції
ДСК	0.001	центральна	0.0000001	0.001	градієнт	7	22	1250	[0.99735209 0.99881514]	9.2212509128716e-06
ДСК	0.01	центральна	0.0000001	0.001	градієнт	3	10	523	[0.8448256 0.98037498]	1.353174863850376e-3
ДСК	0.0001	центральна	0.0000001	0.001	градієнт	10	37	2244	[1.00035077 1.00017501]	3.067988908724827e-08

Наступні порівняння проводились для аналізу критерія закінчення. Було проаналізовано для двох методів МОП при критерія закінчення: модуль градієнту та абсолютна похибка функції. В таблиці наведено результати досліджень. З цих порівнянь видно, що більш ефективним як критерій закінчення пошуку є абсолютна похибка функції. Критерій закінчення не впливає на точність методу, тому доречніше обрати другий.

МОП	Точність МОП	Вид різницевої схеми похідної	Крок h	Точність методу	Критерій закінчення	Кіл-сть рестартів	Кіл-сть ітерацій	Кіл-сть викликів функції	Точка мінімума	Значення функції
ЗП	0.001	централ	0.0000001	0.001	градієнт	4	17	1305	[1.00000001 0.99999996]	1.0383086136980672e-12
ЗП	0.001	централ	0.0000001	0.001	похибка	4	16	1113	[1.00003411 1.00001687]	2.977174177038963e-10

ДСК	0.001	централ	0.0000001	0.001	градієнт	7	22	1250	[0.99735209 0.99881514]	9.2212509128716e-06
ДСК	0.001	централ	0.0000001	0.001	похибка	6	15	755	[1.05232362 1.02592203]	0.000675653160839506 3

В дослідженні було розгянуто вплив різницевої схеми для обчислення похідної. В Додатку 1 продемонстровано, що вид схеми не впливає на збіжність та ефективність методу, проте крок приросту функції має велике значення. Дослідження проводились на основі ЗП та правої різницевої схеми. Видно, що довжина кроку впливає на кількість ітерацій та викликів функцій, а також на точність методу. Тому необхідно знаходити компроміс. Величина кроку 0.0000001 є найоптимальнішою.

МОП	Точність МОП	Вид різницевої схеми похідної	Крок h	Точність методу	Критерій закінчення	Кіл-сть рестрартів	Кіл-сть ітерацій	Кіл-сть викликів функції	Точка мінімума	Значення функції
ЗС	0.001	правая	0.0001	0.001	похибка	4	18	1247	[0.99984638 0.99992213]	6.515134861484372e-09
ЗС	0.001	правая	0.01	0.001	похибка	3	8	543	[0.79398358 0.89107489]	0.011864773981891638
ЗС	0.001	правая	0.0000001	0.001	другой	4	16	1113	[0.99983463 0.99991802]	6.919197255640028e-0

Теоретично було зазначено, що для даних методів необхідні рестрарти. В даній реалізації, якщо крок стає від’ємним, відбувається повернення до початкової ітерації̈ї, тобто початкової одиничної матриці. В таблицях можно побачити кількість рестартів при різних комбінаціях методів обчислень.

ВИСНОВОК

В даній курсовій роботі було досліджено метод Бройдена-Флетчера-Шенно.

Даний метод є квазін'ютоновскьм методом або методом змінної метрики. Цей метод є найефективнішим при розв'язку задач, де функція задано в загальному вигляді та має сверхлінійну швидкість збіжності.

В роботі було порівняно кількість ітерацій методу та кількість обчислення функцій в залежності від:

1. виду одновимірного пошуку:

Золотий перетин

- Кількість обчислення функції: 1305
- Кількість ітерацій: 17

ДСК-Пауелла

- Кількість обчислення функції: 1205
- Кількість ітерацій: 22

2. точності методу одновимірного пошуку для обчислення величини кроку:

ε = 0.01

- Кількість обчислення функції: 523
- Кількість ітерацій: 10

ε = 0.001

- Кількість обчислення функції: 1250
- Кількість ітерацій: 22

ε = 0.0001

- Кількість обчислення функції: 2244
- Кількість ітерацій: 37

3. виду критерія закінчення пошуку

Норма градієнту

- Кількість обчислення функції: 1205
- Кількість ітерацій: 22

Абсолютна похибка функцій

- Кількість обчислення функції: 755
- Кількість ітерацій: 15

4. виду різницевої схеми обчислень похідної функції

Права схема

- Кількість обчислення функції: 755
- Кількість ітерацій: 15

Ліва схема

- Кількість обчислення функції: 755
- Кількість ітерацій: 15

Центральна схема

- Кількість обчислення функції: 755
- Кількість ітерацій: 15

величини прирощення при чисельному обчисленні похідної функції

h = 0.0001:

- Кількість обчислення функції: 1247
- Кількість ітерацій: 18

h = 0.01

- Кількість обчислення функції: 543
- Кількість ітерацій: 8

Література

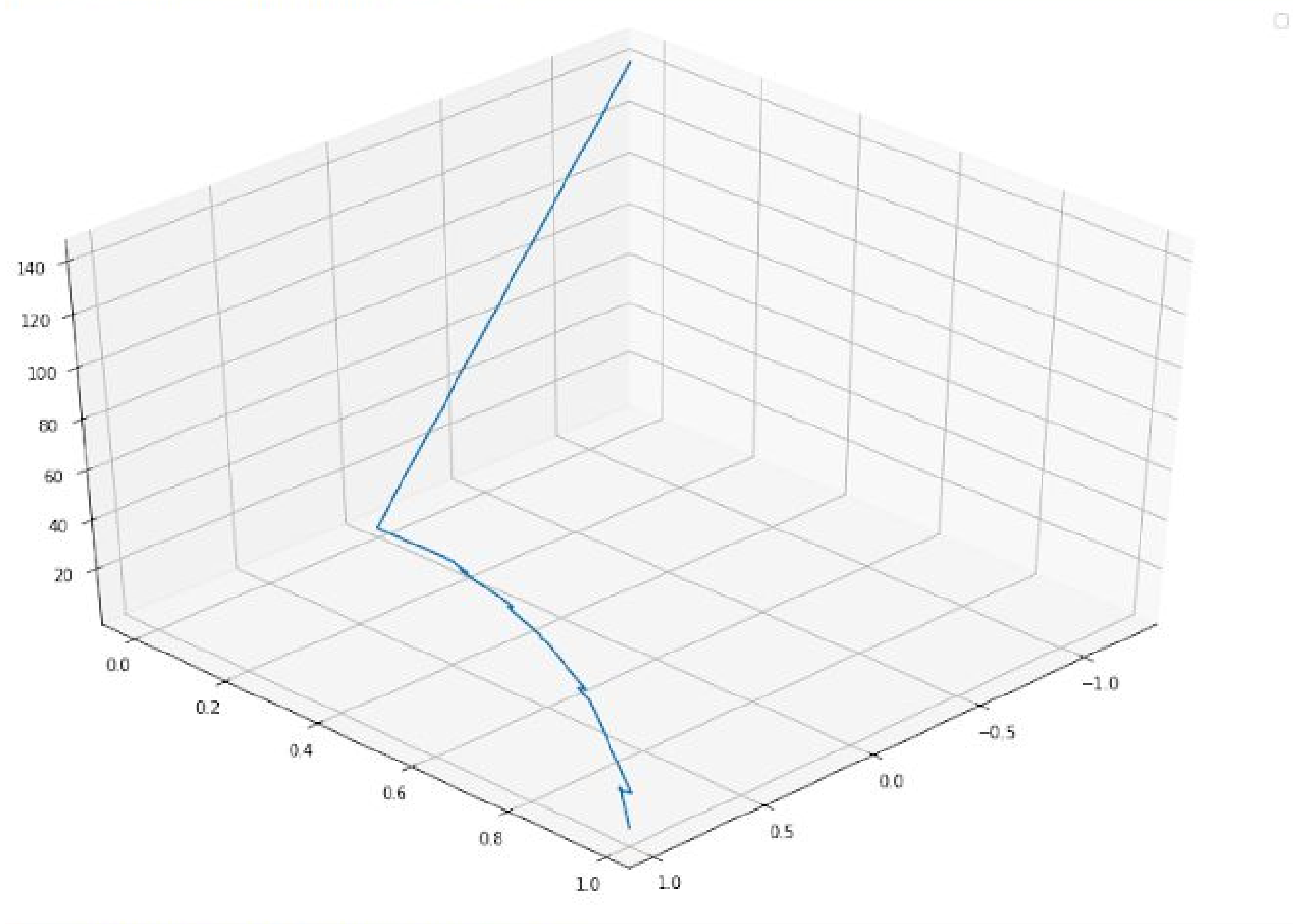
1. Реклейтис Г., Рейвиндран А., Рэгсдел К. Оптимизация в технике. В 2 томах. Том 1 - М.: Мир, 1986. – 348 с.
2. Вагнер Г. Основы исследования операций. Том 1 - Пер. с англ. Б.Т. Вавилова. - М.: Мир, 1972. - 337 с.

Додаток А Таблиця “Порівняльні характеристики дослідження”

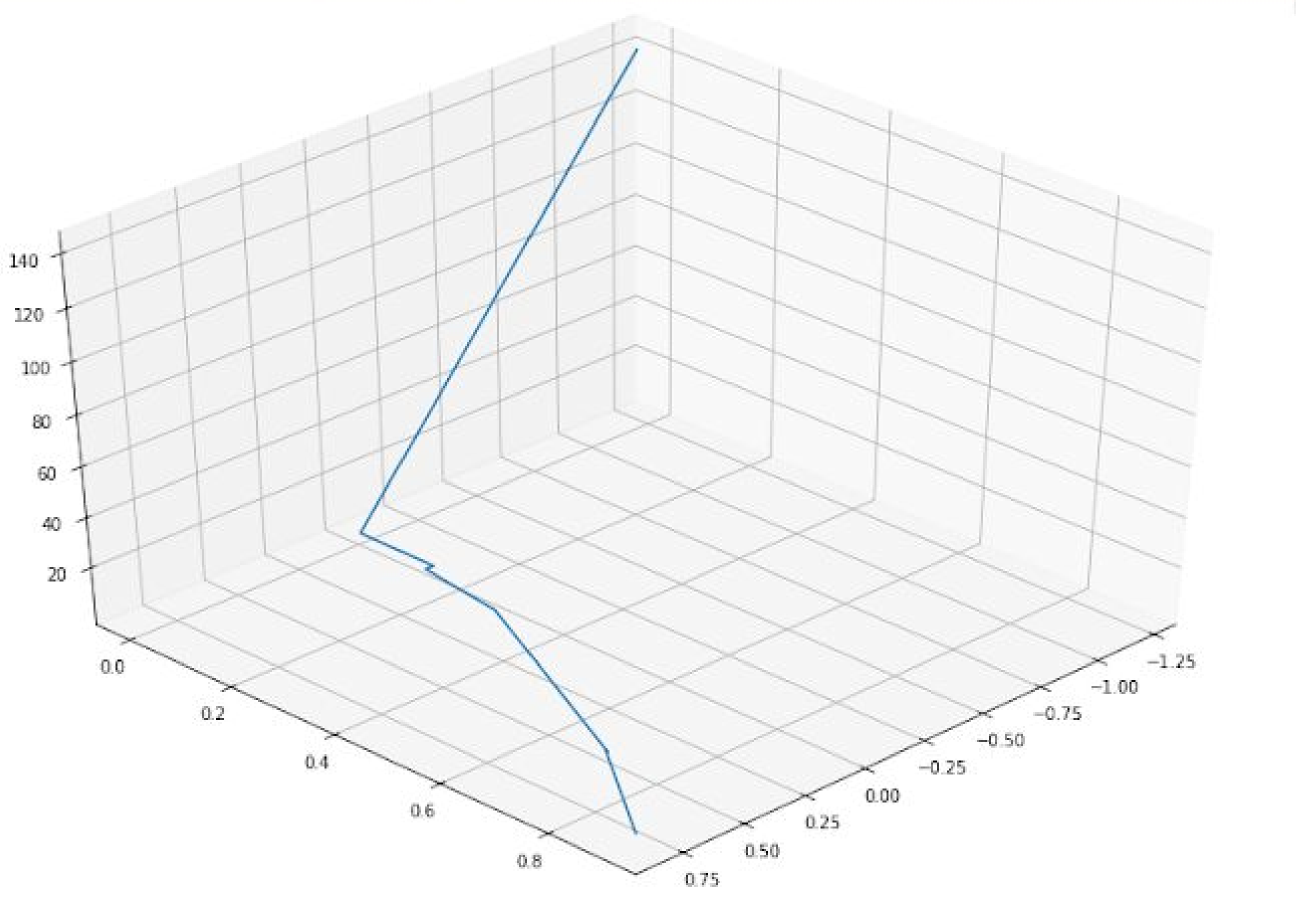
МОП	ТочнІсть МОП	Вид різницевої схеми похідної	Крок h	Точність методу	Критерій закінчення	К-сть рестартів	К-сть ітерацій	К-сть викликів функції	Точка мінімума	Значення функції
ЗП	0.001	центральна	0.0000001	0.001	градієнт	4	17	1305	[1.00000001 0.99999996]	1.03830861369806 72e-12
ДСК	0.001	центральна	0.0000001	0.001	градієнт	7	22	1250	[0.99735209 0.99881514]	9.2212509128716e -06
ДСК	0.01	центральна	0.0000001	0.001	градієнт	3	10	523	[0.8448256 0.98037498]	1.353174863850376e-3
ДСК	0.0001	центральна	0.0000001	0.001	градієнт	10	37	2244	[1.00035077 1.00017501]	3.067988908724827e-08
ЗП	0.001	центральна	0.0000001	0.001	похибка	4	16	1113	[1.00003411 1.00001687]	2.977174177038963e-10
ДСК	0.001	центральна	0.0000001	0.001	похибка	6	15	755	[1.05232362 1.02592203]	0.0006756531608395063
ЗП	0.001	права	0.0000001	0.001	похибка	4	16	1113	[0.99983463 0.99991802]	6.919197255640028e-0
ЗП	0.001	ліва	0.0000001	0.001	похибка	4	16	1113	[0.99999573 0.99999789]	4.647127492875372e-12

ДСК	0.001	права	0.0000001	0.001	похибка	6	15	755	[1.0525517 1.02602077]	0.00067986689334 30548
ДСК	0.001	ліва	0.0000001	0.001	похибка	6	15	755	[1.05228645 1.02589545]	0.00067363812593
ЗП	0.001	права	0.0001	0.001	похибка	4	18	1247	[0.99984638 0.99992213]	6.515134861484372e-09
ЗП	0.001	права	0.01	0.001	похибка	3	8	543	[0.79398358 0.89107489]	0.011864773981891638

Додаток Б. Графік пошуку мінімума (ДСК)



Додаток В. Графік пошуку мінімума (ЗП)



Додаток Г. Лістінг програми

```
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd

count = 0
def RosenbrockFunction(x):
    """The Rosenbrock function"""
    global count
    count = count + 1
    return np.sum(100.0*(x[1:]**2.0-x[:-1])**2.0 + (x[1:]-1)**2.0, axis=0)

#правая производная
def dif(function, x):
    global type_dif
    type_dif = "right"

    global h
    h = 0.01
    H = np.array([h, h])
    e1 = np.array([1, 0])
    e2 = np.array([0,1])

    RD = []
    RD1 = (1/h)*(function(x + H*e1) - function(x))
    RD2 = (1/h)*(function(x + H*e2) - function(x))

    RD.append(RD1)
    RD.append(RD2)
    return np.array(RD)

#левая производная
def l_dif(function, x):
    global type_dif
    type_dif = "left"

    global h
    h = 0.0000001
    H = np.array([h, h])
    e1 = np.array([1, 0])
    e2 = np.array([0,1])

    LD = []
```

```
    LD1 = (1/h)*(function(x) - function(x - H*e1))
    LD2 = (1/h)*(function(x) - function(x - H*e2))

    LD.append(LD1)
    LD.append(LD2)
    return np.array(LD)

#центральная производн
def c_dif(function, x):
    global type_dif
    type_dif = "central"

    global h
    h = 0.0000001
    H = np.array([h, h])
    e1 = np.array([1, 0])
    e2 = np.array([0,1])

    CD = []
    CD1 = (1/(2*h))*(function(x + H*e1) - function(x - H*e1))
    CD2 = (1/(2*h))*(function(x + H*e2) - function(x - H*e2))

    CD.append(CD1)
    CD.append(CD2)
    return np.array(CD)

def delt_x(X1, X2):
    X1 = np.array(X1)
    X2 = np.array(X2)
    return np.array(X1-X2)

def delt_g(X1, X2):
    F1 = np.array(dif(RosenbrockFunction, X1))
    F2 = np.array(dif(RosenbrockFunction, X2))
    return np.array(F1 - F2)

def multiplication_matrix(x1, x2):
    a = []
    b = []
    c = []

    b.append(x1[0]*x2[0])
    b.append(x1[0]*x2[1])

    a.append(b)
```

```
    c.append(x1[1]*x2[0])
    c.append(x1[1]*x2[1])
    a.append(c)

    return a


def metrics(x1, x2, old_metrics):
    I = np.eye(2)
    m1 = I - (multiplication_matrix(delt_x(x1, x2), delt_g(x1, x2)) / np.dot(delt_x(x1, x2),
delt_g(x1, x2)))
    m2 = multiplication_matrix(delt_x(x1, x2), delt_x(x1, x2)) / np.dot(delt_x(x1, x2), delt_g(x1,
x2))
    m3 = np.dot(np.dot(m1, old_metrics), m1)
    return m3 + m2

def delta_lamda(x, S):
    a = 0.1
    return (a * np.linalg.norm(x)/np.linalg.norm(S))

def New_lamda(l, k, delta):
    return l + pow(2, k)*delta

def transform_x (X, lamda, S):
    X = np.array(X)
    S = np.array(S) * lamda
    return S + X
def svenmeth(S, function, x0):
    a = 0
    b = 0
    lambda_0 = 0

    delta = round(delta_lamda(x0, S), 4)

    F = function(x0)
    F_minus = function(transform_x(x0, lambda_0 - delta, S))
    F_plus = function(transform_x(x0, lambda_0 + delta, S))


    if (F <= F_minus) and (F <= F_plus):
        a = lambda_0 - delta
        b = lambda_0 + delta
    elif F_minus < F and F > F_plus:
        print ("The function is not unimodal!")
    else:
        if F_minus < F and F < F_plus:
```

```
            delta = - delta

        k = 0
        old_lamda = lambda_0
        now_lamda = lambda_0 + delta
        while True:
            k = k + 1
            new_lamda = New_lamda(now_lamda, k, delta)
            if function(transform_x(x0, now_lamda, S)) < function(transform_x(x0, new_lamda,
S)):
                half_lamda = (new_lamda + now_lamda)*(1/2)
                if half_lamda > old_lamda:
                    a = old_lamda
                    b = half_lamda
                else:
                    a = half_lamda
                    b = old_lamda
                break
            old_lamda = now_lamda
            now_lamda = new_lamda
    return a, b
def calculation_func(lamda1, lamda2, lamda3, S, function, x0):
    func1 = function(transform_x(x0, lamda1, S))
    func2 = function(transform_x(x0, lamda2, S))
    func3 = function(transform_x(x0, lamda3, S))
    return func1, func2, func3

def dskmeth(S, function, x):

    a, b = svenmeth(S, function, x)

    la1 = a
    la2 = (a+b)/2
    la3 = b
    del_la = (b - a)/2

    f1, f2, f3 = calculation_func(la1, la2, la3, S, function, x)

    sp_la = la2 + (del_la * (f1 - f3))/(2* (f1 - 2*f2 + f3 ))
    f_sp_la = round(function(transform_x(x, sp_la, S)), 5)

    if sp_la < la2:
        la3 = la2
    elif sp_la > la2:
        la1 = la2
```

```
    la2 = sp_la
    f1, f2, f3 = calculation_func(la1, la2, la3, S, function, x)

    A1 = (f2 - f1)/(la2 - la1)
    A2 = (1/(la3-la2))*(((f3-f1)/(la3-la1))-((f2-f1)/(la2 - la1)))

    sp_la = ((la1 + la2)/2) - (A1/(2*A2))
    return sp_la
def goldrotatiometh(eps, S, function, x0):
    a, b = svenmeth(S, function, x0)
    k = 0
    L = b - a
    co = 0

    while math.fabs(L) > eps:
        k += 1
        l_1 = a + (0.382)*L
        l_2 = a + (0.618)*L

        if function(transform_x(x0, l_1, S)) <= function(transform_x(x0, l_2, S)):
            b = l_2
        else:
            a = l_1
        L = b - a
        co = co + 1
    la = (b + a)/2
    return la
MATRIX_steps = []
MATRIX_X = []
MATRIX_F = []
MATRIX_count = []

count = 0
count_restart = 0
i = 0

eps = 0.001
eps1 = 0.001
eps2 = 0.001

old_x = np.array([-1.2, 0])


METRICS = np.eye(2)
DIRECTION = np.array((-1)*dif(RosenbrockFunction, old_x))
#STEP = goldrotatiometh(eps, DIRECTION, RosenbrockFunction, old_x)
```

```
STEP = dskmeth(DIRECTION, RosenbrockFunction, old_x)
current_x = np.array(old_x + STEP*DIRECTION)

while True:

#    check_G = np.linalg.norm(delt_g(current_x, old_x))

    check_X = np.linalg.norm(delt_x(current_x, old_x))
    check_F = math.fabs((RosenbrockFunction(current_x) - RosenbrockFunction(old_x)))


#     if check_G < eps1:
    if check_X < eps1 and check_F < eps1:
        min_x = current_x
        min_f = RosenbrockFunction(current_x)
        break

    MATRIX_count.append(count)
    count = 0

    MATRIX_steps.append(STEP)
    MATRIX_X.append(old_x)
    MATRIX_F.append(RosenbrockFunction(old_x))

    METRICS = metrics(current_x, old_x, METRICS)
    DIRECTION = np.array((-1) * np.dot(METRICS, dif(RosenbrockFunction, current_x)))
    #STEP = goldrotatiometh(eps, DIRECTION, RosenbrockFunction, current_x)
    STEP = dskmeth(DIRECTION, RosenbrockFunction, old_x)

    if (STEP < 0):

        count_restart = count_restart + 1

        METRICS = np.eye(2)
        DIRECTION = np.array((-1) * np.dot(METRICS, dif(RosenbrockFunction, current_x)))
        STEP = goldrotatiometh(eps, DIRECTION, RosenbrockFunction, current_x)

    new_x = np.array(current_x + STEP*DIRECTION)

    old_x = current_x
    current_x = new_x

    i = i + 1

print("Количество итераций метода: ", i)
print("Количесто вызовов функции: ", sum(MATRIX_count))
```

```
print("Количество рестратов: ", count_restart)

print("Минимум в точке:", min_x)
print("Минимум функции:", min_f)

plt.plot(MATRIX_F)

dc = pd.DataFrame({"Steps": MATRIX_steps , "X": MATRIX_X, "F": MATRIX_F, "count
calculating function": MATRIX_count})
dc
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

mpl.rcParams['legend.fontsize'] = 10

fig = plt.figure(figsize=[15, 10])
ax = fig.gca(projection='3d')

ax.view_init(40, 45)

x1 = []
x2 = []
for i in range(len(MATRIX_X)):
   x1.append(MATRIX_X[i][0])
   x2.append(MATRIX_X[i][1])
z = MATRIX_F

ax.plot(x1, x2, z)
ax.legend()

plt.show()
```