НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Курсова робота

із дисципліни «Дослідження операцій»

на тему: “Безумовна оптимізація методом Нелдера-Міда і умовна оптимізація методом ковзного допуску”

Студента групи КМ-73

Тищенко Б.Ю.

Керівник:

Ладогубець Т.С.

Кількість балів:_____

Оцінка:_____

Київ — 2020

ЗМІСТ

ПОСТАНОВКА ЗАДАЧІ

Дослідити збіжність метода Нелдера-Міда при мінімізації функції Розенброка в залежності від:

1. Розміру початкового симплексу.
2. Значень параметрів деформації та редукції багатогранника.
3. Модифікацій метода.

Використати метод ковзного допуску для умовної оптимізації в залежності від:

1. Розташування локального мінімума (всередині/поза допустимою областю).
2. Виду допустимої області (випукла/невипукла).
3. Різновидів обмежень.

МЕТОД НЕЛДЕРА-МІДА

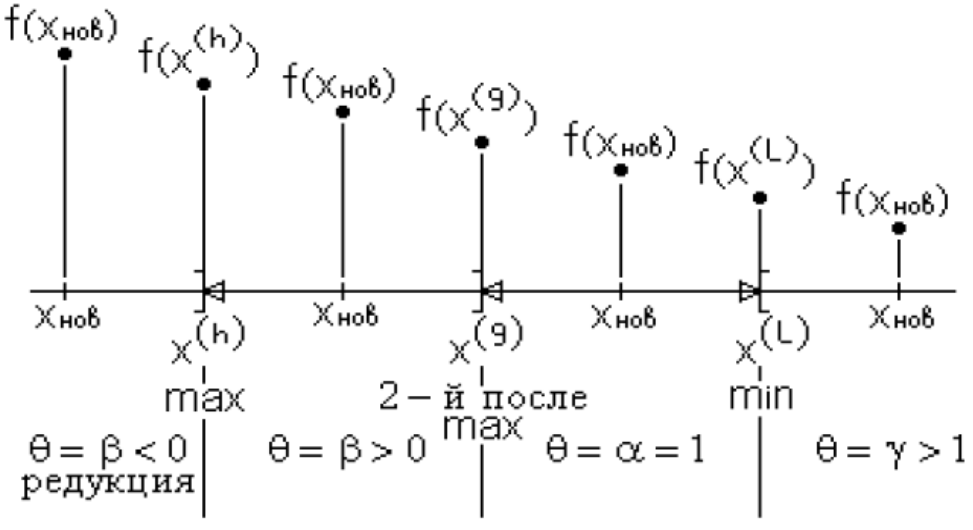
Теоретичні відомості

Метод полягає в тому що у допустимій області ми маємо певний многогранник що у процесі пошуку рухається до мінімуму змінюючи свої розміри. Многогранник рухається шляхом відображення вершини з найбільшим значенням функції ($x^{(h)}$) відносно його центру. На певному етапі многогранник стає завеликим для продовження пошуку і для досягнення бажаної точності необхідно його зменшити. Те що під час пошуку вершини починають повторюватися є індикатором того що він завеликий. З іншого боку, якщо многогранник буде дуже маленьким, пошук сповільниться. Для цього алгоритм методу допускає збільшення многогранника.

Зміну розміру многогранника контролює θ, параметр, що на кожній ітерації пошуку приймає значення початкових параметрів α, γ і β в залежності від результату пробного симетричного відображення $x^{(h)}$. Після визначення θ, відбувається крок алгоритму за такою формулою

$$x = x^{(h)} + (1 + \theta)\left(x_c - x^{(h)}\right)$$

Варто зазначити що для визначення параметру θ на кожній ітерації для всіх вершин многогранника визначається величина значення цільової функції і вершинам присвоюються відмітки. Так $x^{(h)}$вершина з найбільшим значенням, далі $x^{(g)}$ і $x^{(l)}$ відповідно. Так зміну θ і вибір початкових α, γ і β можна проілюструвати таким чином



Фіг.1

Критерієм закінчення алгоритму є умова, формула якої у наступному розділі.

Реалізація алгоритму

Після визначення початкових параметрів алгоритму та вершин початкового многогранника починається основний цикл. На кожній його ітерації вираховується «пробна» вершина шляхом симетричного відбиття xh відносно центру многокутника. Далі в залежності від значення функції у пробній вершині вибирається параметр стиснення тета і знаходиться наступна точка симплексу. На випадок зациклення попередні значення функції зберігаються у змінну previousF. Якщо на якомусь етапі пошуку значення починають повторюватись, відбувається редукція.

За деяких початкових параметрах алгоритм не сходиться до мінімума, це трапляється рідко. На такий випадок у нашому алгоритмі, якщо кількість ітерацій перевищує 5000, він зупиняється повідомляючи про безуспішність пошуку.

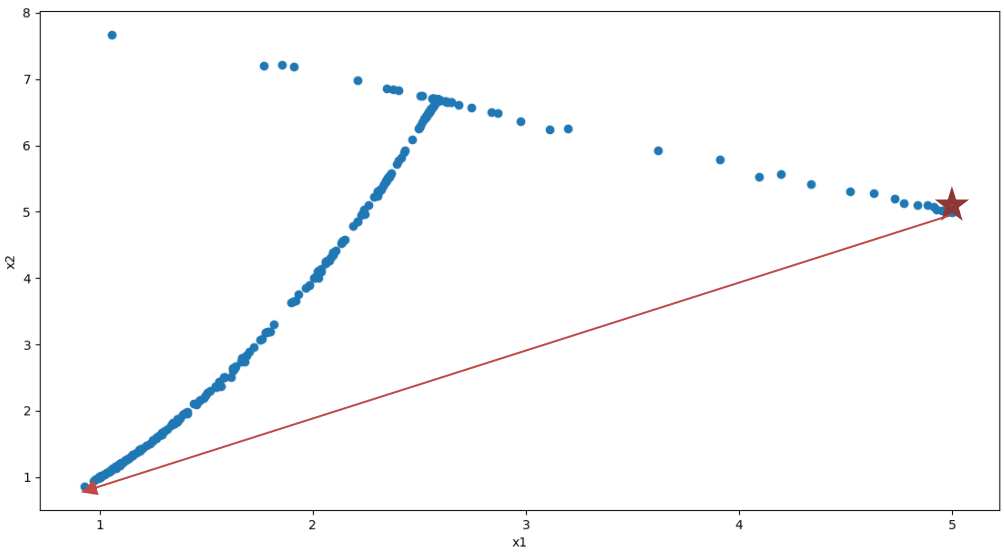
Критерій закінчення. Цикл закінчується тоді коли виконується умова закінчення

$$\sqrt{\left\{\frac{1}{n+1}\sum_{i=1}^{n+1}\left[f\left(x_i^{(k)}\right)-f\left(x_c^{(k)}\right)\right]^2\right\}} \le \xi$$

Дослідження збіжності

У цій частині роботи мінімізуватимемо функцію Розенброка зі стандартними параметрами яка має вигляд

$$f(x_1, x_2) = 100(x_1^2 - y)^2 + (1 - x_1)^2$$



Фіг.3 Загальний графік мінімізації методом Нелдера-Міда

1. Розмір початкового симплексу.

За показник розміру початкового симплексу візьмемо початкову довжину сторони.

Початковий симлекс будуватимемо так: беремо точку допустимої області, візьмемо (5, 5) і відкладаючи від неї фіксовані відстані вправо, вгору і вниз отримуємо відповідно три вершини многогранника.

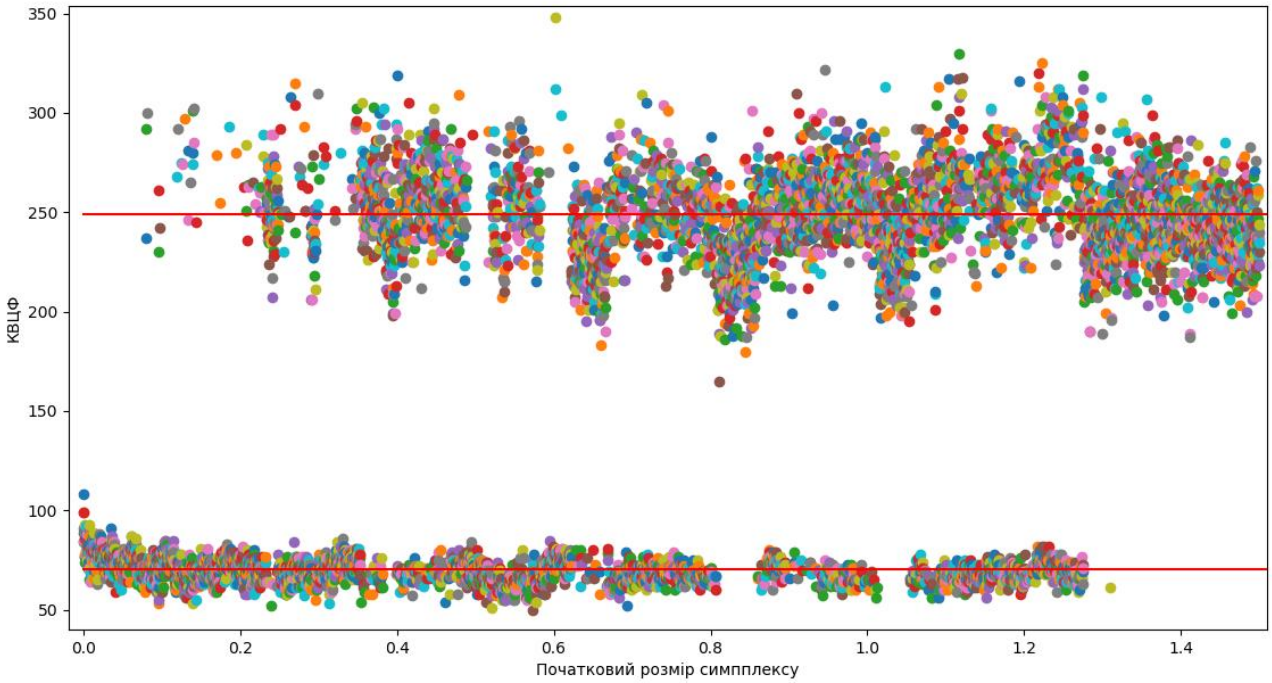
Використовуючи нашу програму отримуємо дані.

e = 10e-6

Розмір початкового симплексу	Ітерації	КВЦФ
10e-7	56	129
10e-6	46	108
10e-5	41	99
10e-4	43	99
10e-3	37	91
10e-2	32	81
10e-1	32	81
1	121	280
10	64	275
100	265	606

Виходячи з отриманих даних можемо зробити висновок, що є оптимальний розмір початкового симплексу за якого результат буде за мінімум КВЦФ.

Перевіримо результат зробивши 7500 ітерацій.



Фіг.2 Червоні лінії – середні значення для верхньої і нижньої частини відповідно

З графіку видно, що вибір розміру початкового симплексу дещо більш випадково впливає на КВЦФ.

Також на графіку видно як результати отримані після порівняно маленької КВЦФ - 50-100 - розділені великим пробілом з результатами отриманими після 190+ КВЦФ. Потім ми спробуємо скористатися цим для модифікації методу.

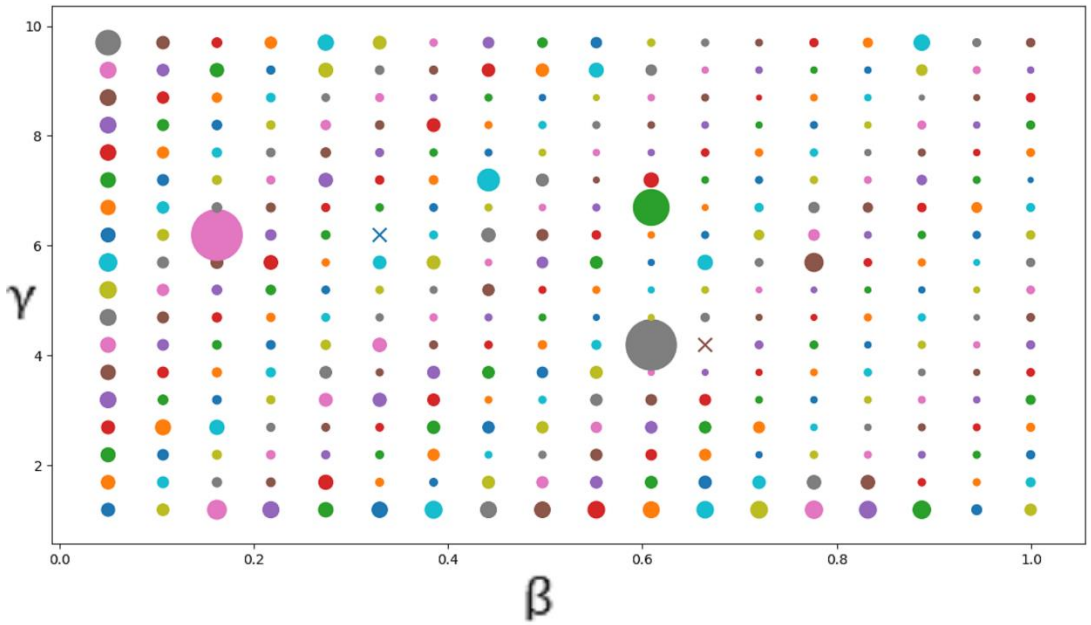
2. Значення параметрів деформації та редукції симплексу.

Стандартними параметрами деформації для цього методу є $\alpha = 1, \gamma = 2$ і $\beta = 0.5$, з них свобода вибору є для $\gamma(> 1)$ і $\beta(> 0)$.

Поекспериментуємо з вибором цих параметрів: перевіримо ефективність різних їх комбінацій для мінімізації функції Розенброка у діапазонах

$[1.2; 10]$ для γ,

$[0.05; 0.95]$ для β. Інші параметри такі самі як у минулому дослідженні.



Фіг.3 (Величина кругів відповідає кількості ітерацій. Хрестиками позначені пошуки що були розбіжними)

З цього графіку бачимо що є варіанти вибору параметрів для цієї задачі що працюватимуть краще страндартних $\gamma = 2$ і $\beta = 0.5$.

Найкраще спрацюють $\gamma = 8.9$ і $\beta = 0.7833$ давши результат за 12 кроків.

3. Модифікації метода. Рестарт

Дивлячись на дані з Фіг.2 можна задатися питанням: чи не буде ефективніше робити рестарт алгоритму після входження у пустку на середині графіка, адже вона буде лише гаяти час до моменту входження у верхню частину графіку, туди де алгоритм закінчиться.

Дослідимо модифікацію методу Нелдера-Міда, у якій кожний раз як алгоритм робитиме більше ніж 90 КВЦФ робитиметься рестарт, з новим початковим розміром многогранника.

Щоб дослідити ефективність цієї модифікації, знайдемо математичне сподівання класичного алгоритму і нового і порівняємо.

Програмно підрахувавши отримуємо такі дані:

Всього мінімізацій: 7500

```
Lower count 3333 Upper count 4167
Lower mean 70.18541854185419 Upper mean 248.81185505159587
```

Отже середнє значення для нижньої і верхньої частин графіку:

$LowerMean = 70.185$ КВЦФ в середньому для 3333 результатів (44%)

$UpperMean = 248.812$ КВЦФ в середньому для 4167 результатів (56%)

Маємо випадкову величину для класичного алгоритма:

```
(70.185, 0.44)
(248.812, 0.56)
```

Для обчислення мат. сподівання модифікованого алгоритму напишемо рекурсивну функцію (Додаток В). Результати:

(результат КВЦФ, ймовірність його реалізації)

```
(70.185, 0.44)
(160.185, 0.24640000000000004)
(250.185, 0.13798400000000002)
(340.185, 0.07727104000000003)
(430.185, 0.043271782400000014)
(520.185, 0.02423219814400001)
(610.185, 0.013570030960640007)
(700.185, 0.007599217337958405)
(790.185, 0.004255561709256707)
(880.185, 0.002383114557183756)
(970.185, 0.0013345441520229036)
(1060.185, 0.0007473447251328261)
(1150.185, 0.0004185130460743826 7)
(1240.185, 0.00023436730580165432)
(1330.185, 0.00013124569124892642)
(1420.185, 7.34975870993988e-05)
(1510.185, 4.1158648775663334e-05)
(1600.185, 2.304884331437147e-05)
(1690.185, 1.2907352256048024e-05)
сумма 0.9999835724607651
```

Тепер знайдемо мат. сподівання:

```
Класичний: M = 170.21612000000005
Модифікований: M = 184.69932878657286
```

Отже, класичний алгоритм виявився швидшим.

Можливості для покращення

У нашій модифікації рестарт робиться після 90ї ітерації, цифра що є *майже* верхнею межею нижнього рівня Фіг.2. Слід зазначити, що при зменшенні порогу рестарту, швидкість буде стрімко зростати ще до певного значення.

МЕТОД КОВЗНОГО ДОПУСКУ

Теоретичні відомості

Метод ковзного допуску доповнює будь-який інший метод мінімізації без обмежень, для того, щоб знаходити мінімум функцій саме з обмеженнями. Часто метод ковзного допуску використовують із методом Нелдера-Міда, що ми і спробуємо зробити.

Коли відбувається мінімізація методом ковзного допуску, вершини на кожній ітерації шукаються так само як під час звичайного пошуку, але робляться додаткові перевірки на те чи не порушені обмеження. Якщо нова точка порушує обмеження, починаючи із нової точки здійснюється мінімізація спеціальної функції порушення обмежень. Після мінімізації отримуємо точку що задовольняє обмеження і пошук продовжується. На графіку пошук, натрапивши на обмеження, починає ніби зісковзувати по графіку обмеження в сторону мінімуму.

Критерій допуску Ф

Вибір критерію допуску залежить від методу руху до мінімуму. Для методу Нелдера-Міда, із ходом мінімізації зменшується розмір многогранника, а з розміром многогранника змешується Ф.

$$\Phi^{(k)} = \min\left\{\Phi^{(k-1)}, \quad \frac{m+1}{r+1}\sum_{i=1}^{r+1} |\boldsymbol{x}_{\boldsymbol{i}}^{(\boldsymbol{k})} - \boldsymbol{x}_{\mathrm{c}}^{(\boldsymbol{k})}|\right\}$$

$$\Phi^{(0)} = 2(m+1)t,$$

де
$t -$ величина, що характеризує розмір початкового многогранника;
$m -$ число обмежень у вигляді рівностей;
$\boldsymbol{x}_{\boldsymbol{i}}^{(\boldsymbol{k})} -$ вектор, що задає положення i -ї вершини многогранника;
$r = (n - m) -$ число степенів свободи цільової функції f(x);
$\boldsymbol{x}_{\mathrm{c}}^{(\boldsymbol{k})} -$ вектор, центр тяжіння поточного многогранника.

Постійне зменшення значення Ф посилює вимоги до нових вершин многогранника змушуючи їх рухатися всередину області допустимості. Вимога що посилюється є нерівністю

$$\Phi^{(k)} - T(\boldsymbol{x}) \geq 0 \qquad (1)$$

Міра порушення обмежень T(x)

$$T(\mathbf{x}) = + \left[\sum_{l=1}^{m} h_i^2(\mathbf{x}) + \sum_{l=m+1}^{p} \mathcal{U}_i \, g_i^2(\mathbf{x}) \right]^{l/2}$$

що більше обмежень та що сильніше вони порушуються у точці x, то більшим буде значення порушення обмежень T.

Кажуть, що вектор $\boldsymbol{x}^{(k)}$ є:
1) допустимим, якщо $T\left(\boldsymbol{x}^{(k)}\right) = 0$
2) майже допустимим, якщо $0 \leq T\left(\boldsymbol{x}^{(k)}\right) \leq \Phi^{(k)}$;
3) недопустимим, якщо $T\left(\boldsymbol{x}^{(k)}\right) > \Phi^{(k)}$. Таким чином, область квазідопустимості визначається відношенням (1).
Мінімізація закінчується тоді коли $\Phi^{(k)} < \epsilon$, тобто, коли многогранник досяг певного розміру ϵ.

Реалізація алгоритму
Імпортуються бібліотеки numpy, matplotlib, numpy, імпортується написаний раніше метод Нелдера-Міда. Визначаються початкові параметри: цільова функція, функції обмежень, вершини початкового многогранника, точність. Також визначаються функції для $\Phi^{(k)}$ та $\Phi^{(0)}$, T.

Тепер опишемо основний алгоритм. Для вершин початкового многогранника знаходимо значення цільової функції, знаходимо $\Phi^{(0)}$ і переходимо до основного циклу. Цикл виконується доти доки Φ буде більше ϵ. На початку циклу вершини многогранника сортуються за значенням цільової функції, потім знаходиться значення T від найкращої вершини. Якщо ця вершина не допустима, виконується мінімізація T і знаходиться нова, допустима вершина і замінює найгіршу вершину. Далі робиться симетрична проекція вершини, за методом Нелдера-Міда. Якщо отримана вершина допустима, вона замінює найгіршу, якщо ж ні, виконується мінімізація T даючи в результаті нову допустиму вершину. Далі йде етап деформації многогранника що дає одну або дві нових вершини. Якщо відбулася редукція, переходимо до наступної ітерації, а якщо ні, перевіряємо отриману вершину на допустимість. Якщо не допустима, мінімізуємо T. Далі йде перевірка на зациклення, в разі якого відбувається редукція. В кінці циклу перераховується $\Phi^{(k)}$.

Дослідження збіжності

1. Розташування локального мінімума (всередині/поза допустимою областю).

Всередині області

Мінімізуємо $f(\boldsymbol{x}) = 4x_1 - x_2^2 - 12$

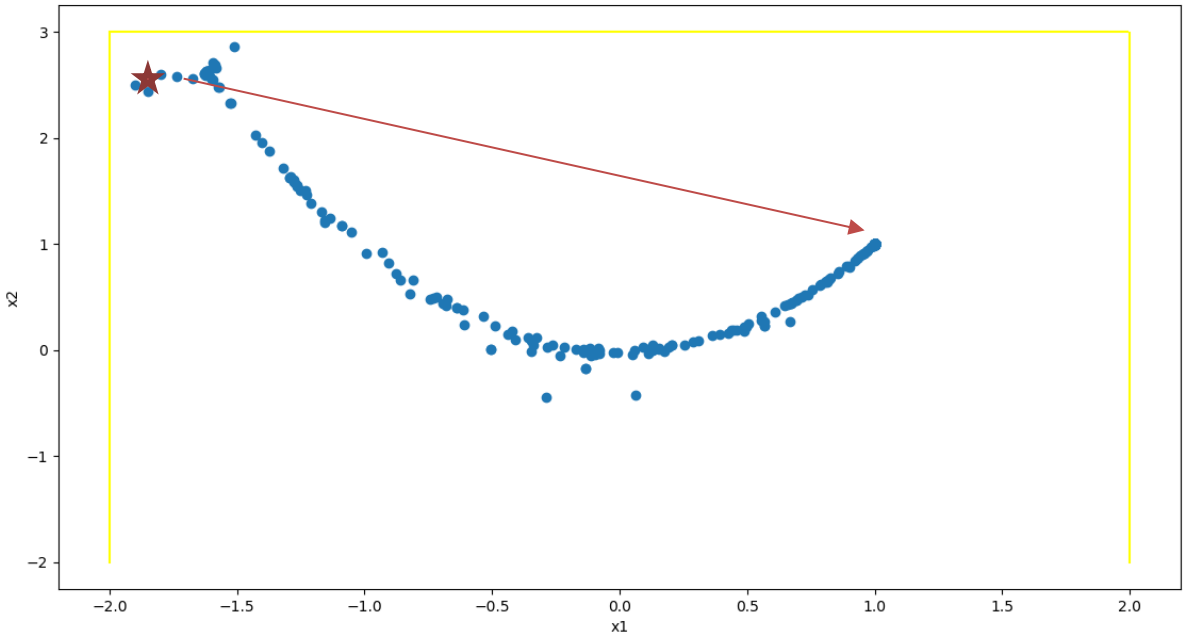
За обмежень $g_1(x) = -x_2 + 2 \geq 0$

$g_2(x) = x_1 + 2 \geq 0$

$g_3(x) = -x_1 + 2 \geq 0$

$g_4(x) = x_2 + 3 \geq 0$

Початковий многогранник [-1.9, 2.5], [-1.8, 2.6], [-1.85, 2.44]



Фіг. Жовті лінії – обмеження нерівності

Поза областю

Мінімізуємо $f(\boldsymbol{x}) = 4x_1 - x_2^2 - 12$

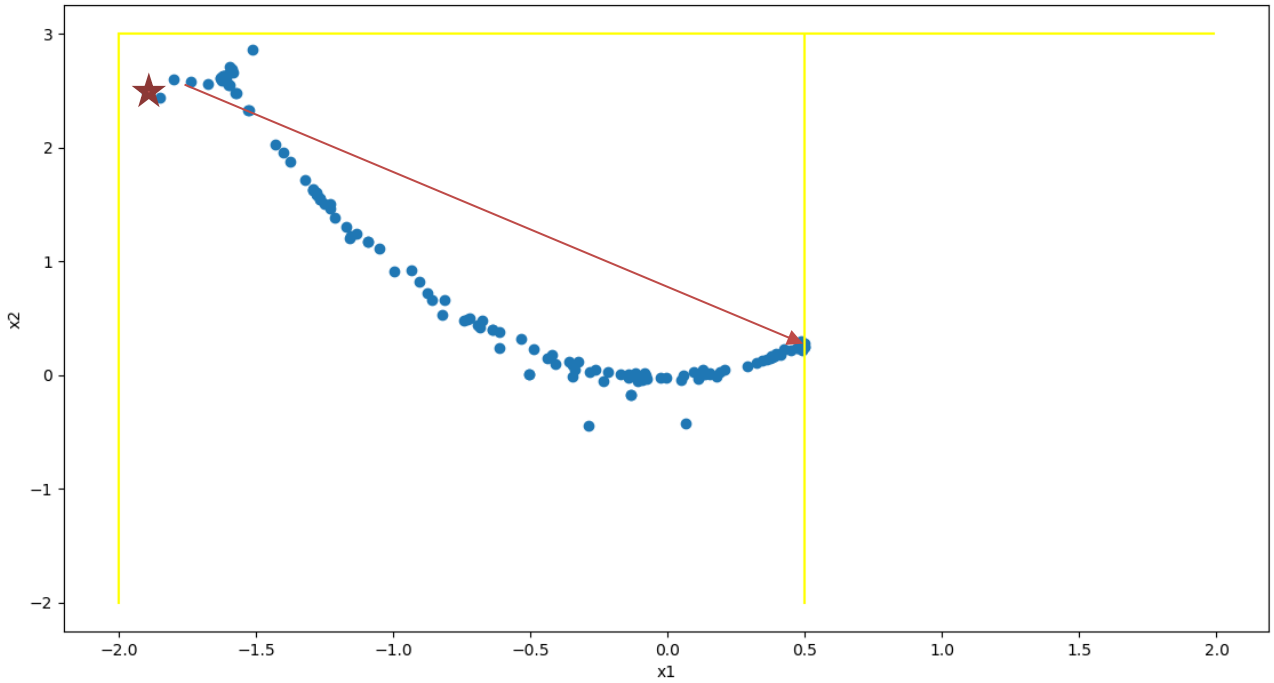
За обмежень $g_1(x) = -x_2 + 2 \geq 0$

$g_2(x) = x_1 + 2 \geq 0$

$g_3(x) = -x_1 + 0.5 \geq 0$

$g_4(x) = x_2 + 3 \geq 0$

Початковий многогранник [-1.9, 2.5], [-1.8, 2.6], [-1.85, 2.44]



Фіг. Жовті лінії – обмеження нерівності

Максимальна точність ($\Phi_{\text{останнє}}$) досягнута після 600 КВЦФ		
Обмеження типу	Ітерацій	Точність
Всередині	233	10e-32
Поза	162	10e-8

2. Вид допустимої області (випукла/невипукла).

Випукла

Мінімізуємо $f(\boldsymbol{x}) = 4x_1 - x_2^2 - 12$

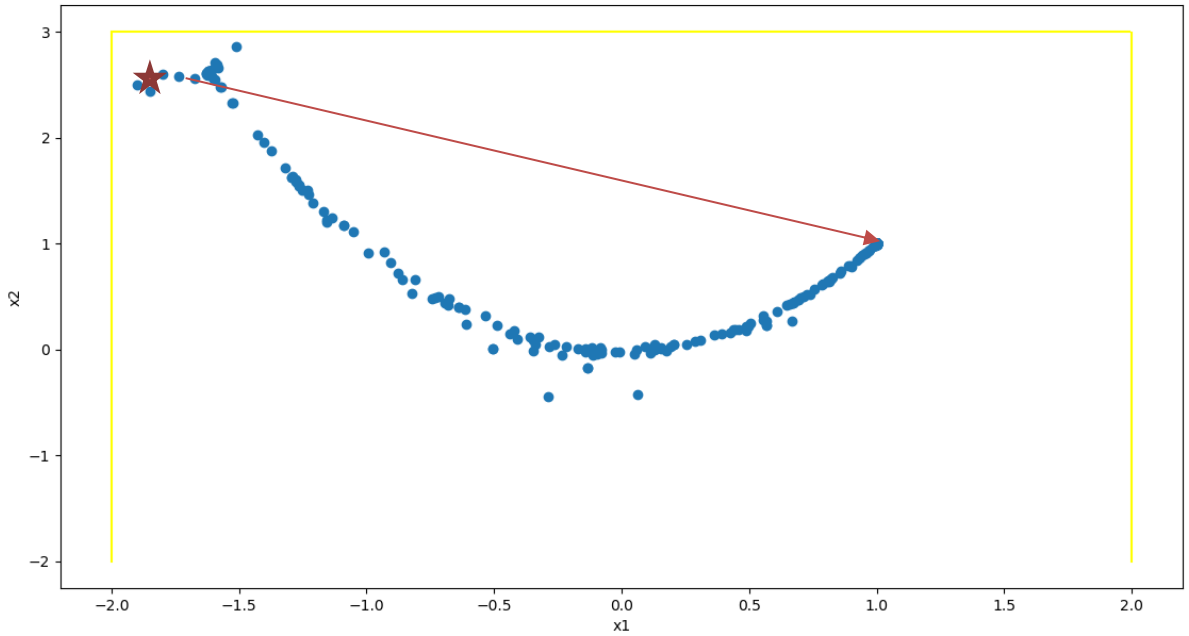
За обмежень $g_1(x) = -x_2 + 2 \geq 0$

$g_2(x) = x_1 + 2 \geq 0$

$g_3(x) = -x_1 + 2 \geq 0$

$g_4(x) = x_2 + 3 \geq 0$

Початковий многогранник [-1.9, 2.5], [-1.8, 2.6], [-1.85, 2.44]



Фіг. Жовті лінії – обмеження нерівності

Невипукла

Мінімізуємо $f(\boldsymbol{x}) = 4x_1 - x_2^2 - 12$

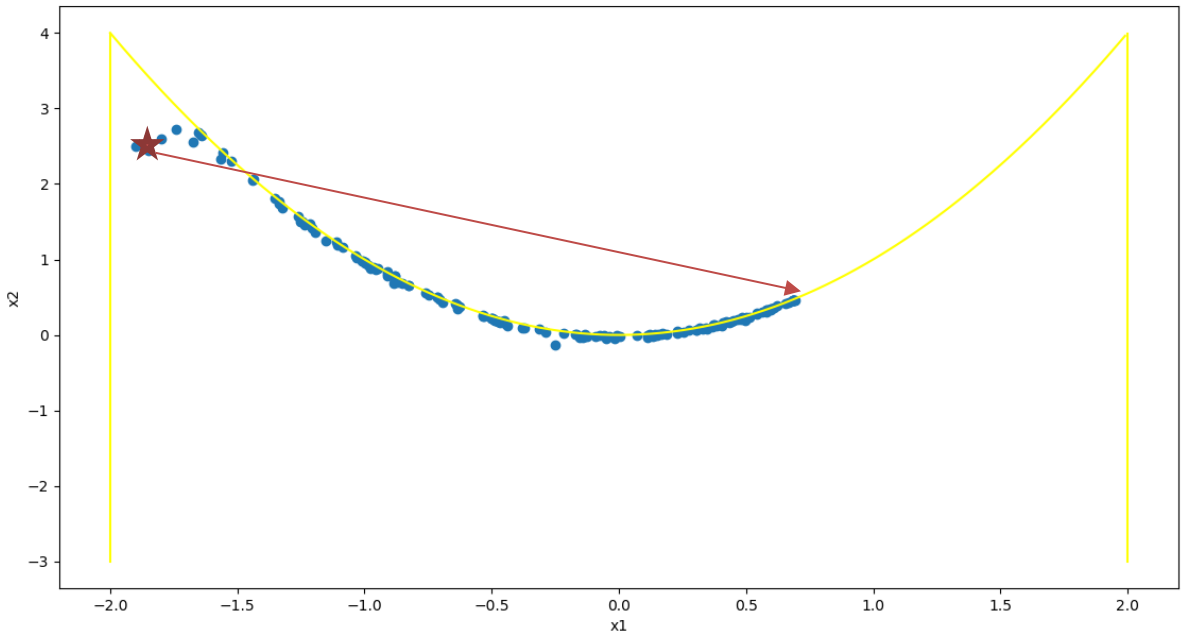
За обмежень $g_1(x) = x_1^2 - x_2 \geq 0$

$g_2(x) = x_1 + 2 \geq 0$

$g_3(x) = -x_1 + 2 \geq 0$

$g_4(x) = x_2 + 3 \geq 0$

Початковий многогранник [-1.9, 2.5], [-1.8, 2.6], [-1. 85, 2.44]



Фіг. Жовті лінії – обмеження нерівності

Максимальна точність ($\Phi_{останнє}$) досягнута після 600 КВЦФ		
	Ітерацій	Точність
Випукла область	233	10e-32
Невипукла область	162	10e-6

3. Різновиди обмежень.

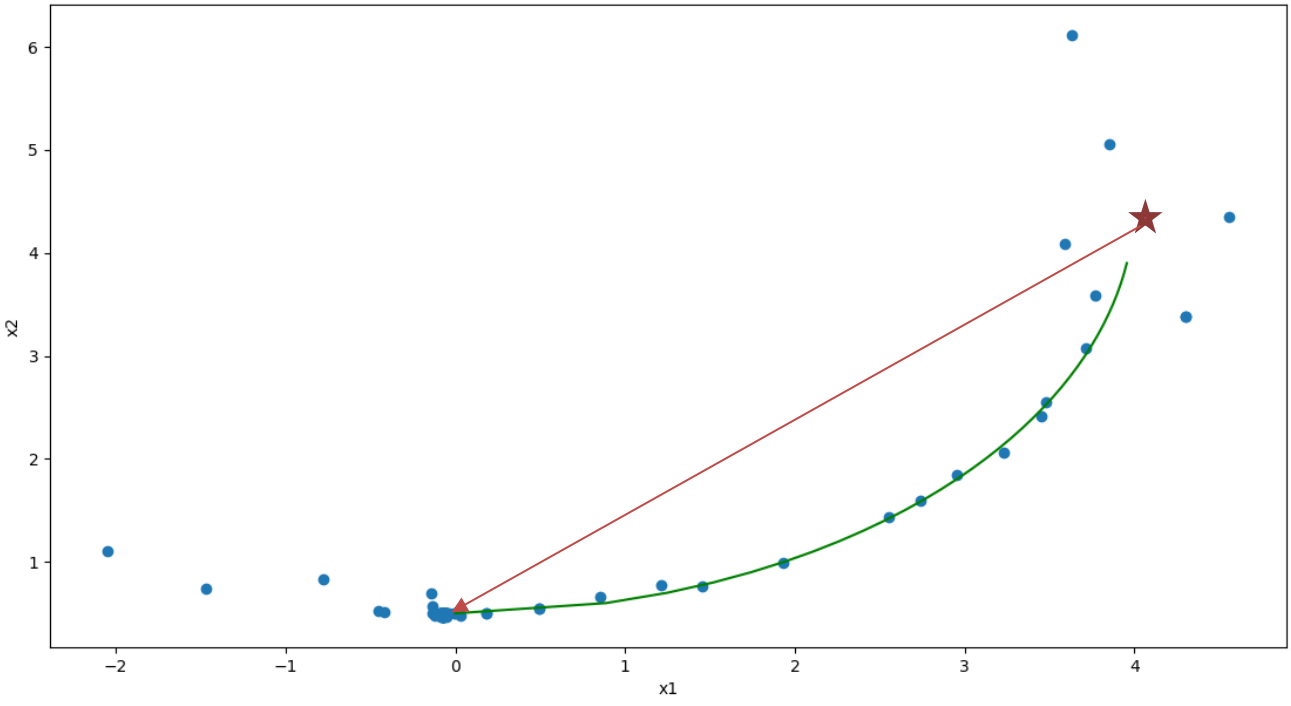
Рівності

Мінімізуємо $f(\boldsymbol{x}) = x_1^2 + x_2^2$

За обмеження $h_1(\boldsymbol{x}) = x_1^2 + x_2^2 - 9x_1^2 + 4.25 = 0$

Початковий многогранник $[3.592, 4.092]$, $[4.558, 4.351]$, $[3.85, 5.06]$;

точність $\epsilon = 10^{-6}$



Фіг. Зелена лінія – частина обмеження рівності (кола).

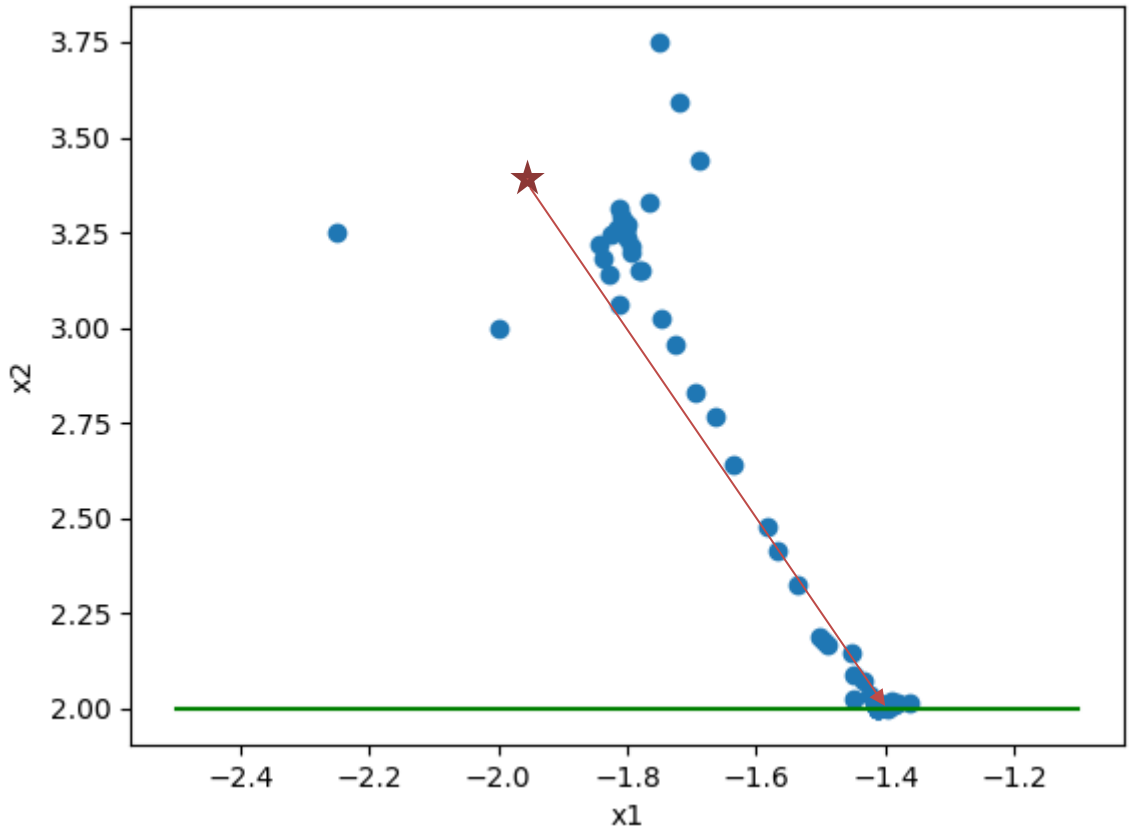
Нерівності

Мінімізуємо $f(\boldsymbol{x}) = 100(x_1^2 - y)^2 + (1 - x_1)^2$

За обмежень $h_1(\boldsymbol{x}) = 25 - x_1^2 - x_2^2 = 0$

$$g_2(x) = 10x_1 - x_1^2 + 10x_2 - x_2^2 - 34 \geq 0$$

Початковий многогранник [-2, 3], [-2.25, 3.25], [-1.75, 3.75]



Фіг. Зелена лінія – обмеження нерівності.

Рівності й нерівності

Мінімізуємо $f(\boldsymbol{x}) = 4x_1 - x_2^2 - 12$

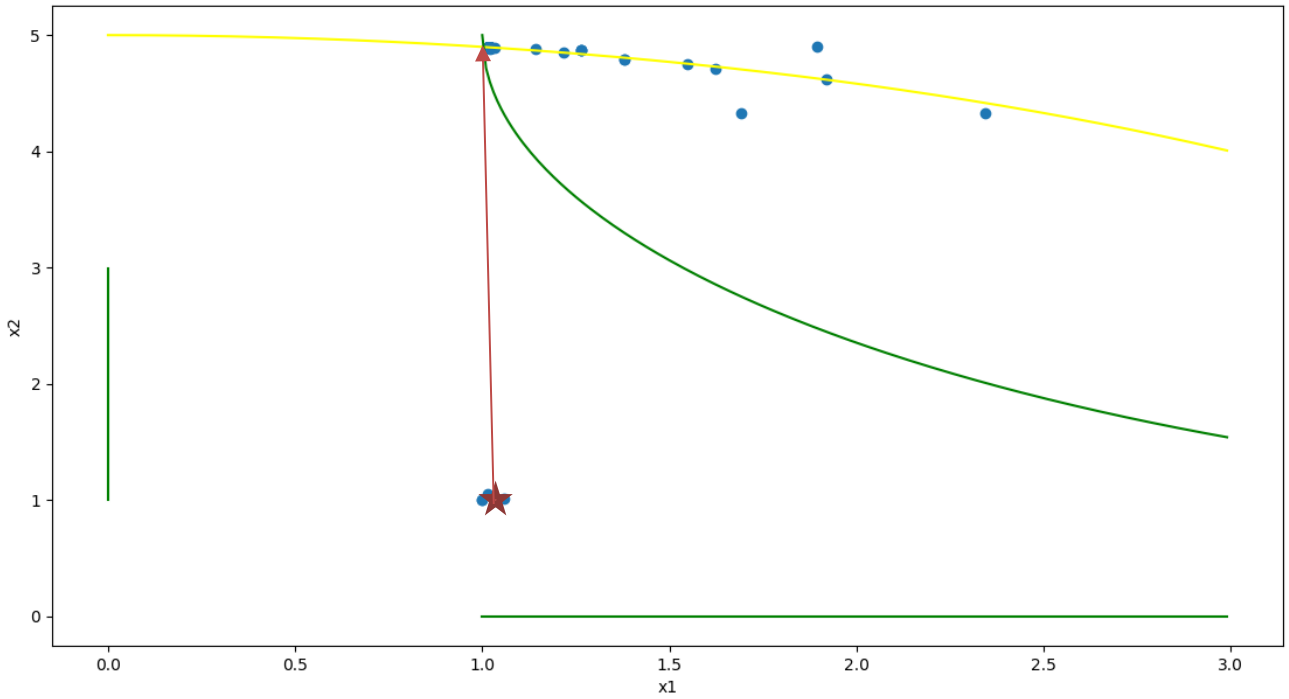
За обмежень $h_1(\boldsymbol{x}) = 25 - x_1^2 - x_2^2 = 0$

$$g_2(x) = 10x_1 - x_1^2 + 10x_2 - x_2^2 - 34 \geq 0$$

$$g_3(x) = x_1 \geq 0$$

$$g_4(x) = x_2 \geq 0$$

Початковий многогранник [1, 1], [1.057, 1.015], [1.015, 1.057]



Фіг. Зелена лінія – обмеження нерівності,

жовта – обмеження рівності

Максимальна точність ($Ф_{останнє}$) досягнута після 600 КВЦФ		
Обмеження типу	Ітерацій	Точність
Рівності	19	10e-5
Нерівності	123	10e-9
Рівності й нерівності	11	10e-5

Ітерації під час мінімізацій T(x) не включено у Ітерації.

Висновки

Розмір початкового симплексу достатньо випадково пливає на швидкість збіжності. Знайдені параметри деформації що найшвидше дають результат. Досліджена ефективність модифікації методу Нелдера-Міда, що полягає у рестарті алгоритму після проходження певного порогу КВЦФ. Вона виявилася на 14 ВЦФ в середньому менш ефективною, але її можливо покращити.

Після дослідження метода ковзного допуску ми з’ясували, що чим частіше алгоритм натрапляє на обмеження, тим більша КВЦФ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Химмельблау Д. Прикладное нелинейное программирование /

Химмельблау Д. — М. : Мир, 1975. — 535 с.

ДОДАТОК А (код для методу Нелдера-Міда+дослідження)

```
import matplotlib.pyplot as plt
from numpy import array
import random
import numpy as np

def tupleAdd(x, y):
        return (x[0]+y[0],x[1]+y[1])

#f = lambda x: 4*(x[0]-3)**2 + (x[1]-2)**2
# Функция Розенброка
a, b = 1, 100
f = lambda x: (a-x[0])**2 + b*(x[1]-x[0]**2)**2

# Початкові параметри
simpSizeCoef = 0.01
e = 0.00001
alpha, beta, gamma = 1, (-0.5, 0.5), 2
n = 3
# Функція основного методу
def nelderMid(beta, gamma):
        # Начальные точки
        xStart = array([5, 5])
        x_f = ( xStart-array([0, simpSizeCoef]), None),(
xStart+array([simpSizeCoef, 0]), None),( xStart+array([0, simpSizeCoef]) , None)
        x_f = [(p[0], f(p[0])) for p in x_f]
        xc = (x_f[0][0]+x_f[1][0])/2
        startSize = (sum([(x_f[i][1]-f(xc))**2 for i in range(n)])/(n+1))**0.5
        # print("Начальные точки: "+" ".join(["x"+str(i+1)+" = "+str(x_f[i][0])
for i in range(3)]))

        # print(" ".join(["f"+str(i+1)+" = "+str(x_f[i][1]) for i in range(n)]))

        project = lambda xh, xc, teta: xh + (1+teta)*(xc-xh)

        previousF = ['*','*','*', '*', '*']
        k=0
        i=0
        # превычисление (для первой итерации)
        x_f.sort(key=lambda p:p[1])
        while (sum([(x_f[i][1]-f(xc))**2 for i in range(n)])/(n+1))**(0.5) > e:
                # print(str(i+1)+")")
                x_f.sort(key=lambda p:p[1])
                xc = (x_f[0][0]+x_f[1][0])/2
                # print("xl =", x_f[0][0], "xg =", x_f[1][0],"xh =", x_f[2][0])
                # print("f(xl) =", x_f[0][1], "f(xg) =", x_f[1][1],"f(xh) =",
x_f[2][1])
                # print("Центр тяжести: xc =", xc)

                teta = 1
                xNew = project(x_f[2][0], xc, teta)
                fNew = f(xNew)
                # print("пробное xNew =", xNew)
                # print("пробное f(xNew) =", fNew)
                #compare
                if fNew<x_f[0][1]:
                        teta = gamma
                elif fNew<x_f[1][1]:
                        teta = alpha
                elif fNew<x_f[2][1]:
                        teta = beta[1]
```

```
                else:
                        teta = beta[0]
                        x1 = project(x_f[0][0], x_f[1][0], teta)
                        x2 = project(x_f[0][0], x_f[2][0], teta)
                        x_f = [(x1,f(x1)), (x2, f(x2)), x_f[0]]
                        # print("Редукция")
                        i+=1
                        k+=1
                        continue

                xNew = project(x_f[2][0], xc, teta)
                fNew = f(xNew)
                # Проверка на зацикливание
                # print("prev", previousF[-2], "new", fNew)
                if fNew == previousF[-2] or fNew == previousF[-3] or fNew ==
previousF[-4]:
                        # print("in")
                        teta=beta[0]
                        xNew = project(x_f[2][0], xc, teta)
                        fNew = f(xNew)
                # print("teta =", teta)
                # print("xNew =", xNew)
                # print("f(xNew) =", fNew)
                x_f[2] = (xNew, fNew)
                i+=1
                if i >5000:
                        return 1000
                previousF.append(fNew)



        xc = (x_f[0][0]+x_f[1][0])/2
        # Критерий окончания
        # print("Критерий окончания [1/(n+1)] * sum(f(x)-f(xc))**2 =
",sum([(x_f[i][1]-f(xc))**2 for i in range(n)])/(n+1), "<e")
        # print("колво редукций:", k)
        x_f.sort(key=lambda p:p[1])
        # print("xl =", x_f[0][0], "xg =", x_f[1][0],"xh =", x_f[2][0])
        # print("f(xl) =", x_f[0][1], "f(xg) =", x_f[1][1],"f(xh) =", x_f[2][1])
        # print("*xmin =", x_f[0][0])
        # print("*f(xmin) =", x_f[0][1])
        # print("Начальный размер симплекса", simpSizeCoef)
        return i

# Функція модифікації методу з випадковими коефіцієнтами
def nelderMidRand(betaRange, gammaRange, simpSizeCoef=0.01):
# Начальные точки
        xStart = array([5, 5])
        x_f = ( xStart-array([0, simpSizeCoef]), None),(
xStart+array([simpSizeCoef, 0]), None),( xStart+array([0, simpSizeCoef]) , None)
        x_f = [(p[0], f(p[0])) for p in x_f]
        xc = (x_f[0][0]+x_f[1][0])/2
        startSize = (sum([(x_f[i][1]-f(xc))**2 for i in range(n)])/(n+1))**0.5
        # print("Начальные точки: "+" ".join(["x"+str(i+1)+" = "+str(x_f[i][0])
for i in range(3)]))

        # print(" ".join(["f"+str(i+1)+" = "+str(x_f[i][1]) for i in range(n)]))

        project = lambda xh, xc, teta: xh + (1+teta)*(xc-xh)

        previousF = ['*','*','*', '*', '*']
        k=0
```

```
        i=0
        # превычисление (для первой итерации)
        x_f.sort(key=lambda p:p[1])
        while (sum([(x_f[i][1]-f(xc))**2 for i in range(n)])/(n+1))**(0.5) > e:
                # print(str(i+1)+")")
                x_f.sort(key=lambda p:p[1])
                xc = (x_f[0][0]+x_f[1][0])/2
                # print("xl =", x_f[0][0], "xg =", x_f[1][0],"xh =", x_f[2][0])
                # print("f(xl) =", x_f[0][1], "f(xg) =", x_f[1][1],"f(xh) =",
x_f[2][1])
                # print("Центр тяжести: xc =", xc)

                teta = 1
                xNew = project(x_f[2][0], xc, teta)
                fNew = f(xNew)
                # print("пробное xNew =", xNew)
                # print("пробное f(xNew) =", fNew)
                #compare
                if fNew<x_f[0][1]:
                        teta = random.choice(gammaRange)
                elif fNew<x_f[1][1]:
                        teta = alpha
                elif fNew<x_f[2][1]:
                        teta = random.choice(betaRange)
                else:
                        teta = -random.choice(betaRange)
                        x1 = project(x_f[0][0], x_f[1][0], teta)
                        x2 = project(x_f[0][0], x_f[2][0], teta)
                        x_f = [(x1,f(x1)), (x2, f(x2)), x_f[0]]
                        # print("Редукция")
                        i+=1
                        k+=1
                        continue

                xNew = project(x_f[2][0], xc, teta)
                fNew = f(xNew)
                # Проверка на зацикливание
                # print("prev", previousF[-2], "new", fNew)
                if fNew == previousF[-2] or fNew == previousF[-3] or fNew ==
previousF[-4]:
                        # print("in")
                        teta=beta[0]
                        xNew = project(x_f[2][0], xc, teta)
                        fNew = f(xNew)
                # print("teta =", teta)
                # print("xNew =", xNew)
                # print("f(xNew) =", fNew)
                x_f[2] = (xNew, fNew)
                i+=1
                if i >5000:
                        return 1000
                previousF.append(fNew)



        xc = (x_f[0][0]+x_f[1][0])/2
        # Критерий окончания
        # print("Критерий окончания [1/(n+1)] * sum(f(x)-f(xc))**2 =
",sum([(x_f[i][1]-f(xc))**2 for i in range(n)])/(n+1), "<e")
        # print("колво редукций:", k)
        x_f.sort(key=lambda p:p[1])
        # print("xl =", x_f[0][0], "xg =", x_f[1][0],"xh =", x_f[2][0])
```

```
        # print("f(xl) =", x_f[0][1], "f(xg) =", x_f[1][1],"f(xh) =", x_f[2][1])
        # print("*xmin =", x_f[0][0])
        # print("*f(xmin) =", x_f[0][1])
        # print("Начальный размер симплекса", simpSizeCoef)
        return i


# random mod
gammaRange = np.arange(2, 5, 0.5)
betaRange = np.linspace(0.7, 0.99, len(gammaRange))
# su = 0
# for i in range(100):
        # res = nelderMidRand(betaRange, gammaRange)
        # print(res)
        # su += res

# print('Classic result: ', nelderMid(beta, gamma))
# print('Modified result:', su/100)



# Дослідження коефіцієнтів
gamma = np.arange(1.1, 10, 0.2)
beta = np.linspace(0.05, 0.999, len(gamma))
alpha = 1
minres, ming, minb = 1999, 10000, 9
for b in beta:
        for g in gamma:
                res = nelderMid((-b, b),g)
                if res< minres:
                        minres = res
                        ming, minb = g, b
                if res==1000:
                        plt.scatter(b,g, 100, marker='x')
                plt.scatter(b,g,res)
print(minres, 'if gamma =',ming, ', beta =,', minb)
plt.show()



# start simplex size
# coefs = np.arange(0.0000001,1, 0.0001)
# r45, r70, r45s, r70s  = 0, 0, 0, 0
# min = 1000
# for c in coefs:
        # xStart = array([5, 5])
        # x_f = ( xStart-array([0, c]), None),( xStart+array([c, 0]), None),(
xStart+array([0, c]) , None)
        # x_f = [(p[0], f(p[0])) for p in x_f]
        # xc = (x_f[0][0]+x_f[1][0])/2
        # startSize = (sum([(x_f[i][1]-f(xc))**2 for i in range(n)])/(n+1))**0.5
        # res = nelderMid(c)
        # if res<45:
                # r45 += 1
                # r45s += res
        # elif res>70:
                # r70 +=1
                # r70s +=res
        # if res<min:
                # min = res
        # plt.scatter(startSize, res)
# print('45-',r45, '70+', r70)
```

```
# print('s45-',r45s/r45, 's70+', r70s/r70)
# plt.show()

# Дослідження модифікації з випадковими коефіцієнтами
# for c in coefs:
        # su1 = 0
        # size calc
        # xStart = array([5, 5])
        # x_f = ( xStart-array([0, c]), None),( xStart+array([c, 0]), None),(
xStart+array([0, c]) , None)
        # x_f = [(p[0], f(p[0])) for p in x_f]
        # xc = (x_f[0][0]+x_f[1][0])/2
        # startSize = (sum([(x_f[i][1]-f(xc))**2 for i in range(n)])/(n+1))**0.5
        ##### nelderMidRand calc
        # for i in range(5):
                # iterations = nelderMidRand(betaRange, gammaRange, c)
                # su1 += iterations
        # plt.scatter(startSize,su1/5, marker='x')
        # plt.scatter(startSize, nelderMid(c), marker='o')
# plt.legend()
# plt.show()
```

ДОДАТОК Б (код для методу ковзного допуску)

```
import matplotlib.pyplot as plt
from numpy import array
import random
from math import sqrt
import numpy as np
from nelderMid import nelderMid, nelderMid1

def areEqual(arr1, arr2):
        n, m = len(arr1), len(arr2)
        if (n != m):
                return False;
        for i in range(0, n - 1):
                if (arr1[i] != arr2[i]):
                        return False;
        return True;

# Функция Розенброка
a, b = 1, 100
f = lambda x: (a-x[0])**2 + b*(x[1]-x[0]**2)**2

# f = lambda x: 4*x[0]-x[1]**2-12
g = [ lambda x: x[0]+2, lambda x: -x[0]+0.5, lambda x: x[1]+3]
h = []
f1 = lambda x: -2
f2 = lambda x: 2
f3 = lambda x: 0.5
# f3 = lambda x: x**2
# g = [lambda x: x[1]-2 ]
# h = [lambda x: 25-x[0]**2-x[1]**2]
# g = [lambda x: 10*x[0]-x[0]**2+10*x[1]-x[1]**2-34, lambda x: x[1], lambda x:
x[0]]
# f = lambda x: x[0]**2 + x[1]**2
# f1 = lambda x: sqrt(16-(x-4.5)**2)
# f1 = lambda x: 0
# f2 = lambda x: sqrt(25-x**2)
# f3 = lambda x: -sqrt(16-(x-5)**2)+5
# Начальные парараметры
# g=[]
# h = [lambda x: x[0]**2 + x[1]**2 - 9* x[1] + 4.25]
n = 2
t = 1
m = len(h)
r = n - m
alpha, beta, gamma = 1, (-0.5, 0.5), 2
e = 10e-8

Phi0 = lambda m, t:2*(m+1)*t
Phi = lambda phiPrev, m, r, x, xc: min(phiPrev, (m+1)/(r+1) * sum([sum((xi-
xc)**2) for xi in x]))
T = lambda x, h=h, g=g: abs( (sum([hi(x)**2 for hi in h]) + sum([gi(x)**2 if
gi(x) < 0 else 0 for gi in g]))**(1/2) )



xStart = (4, 4.5)
def slidingDecent():
        kvcf = 0
        # x_f = (array([3.592, 4.092]), None),(array([4.558, 4.351]),
None),(array([3.85, 5.06]) , None)
```

```
        # x_f = (array([-2, 3]), None),(array([-2.25, 3.25]), None),(array([-
1.75, 3.75]) , None)
        # x_f = (array([1, 1]), None),(array([1.057, 1.015]),
None),(array([1.015, 1.057]) , None) # = + >=
        x_f = (array([-1.9, 2.5]), None),(array([-1.8, 2.6]), None),(array([-
1.85, 2.44]) , None)
        x_f = [(p[0], f(p[0])) for p in x_f]
        kvcf += 3
        xTrack = [x_f[i][0] for i in range(len(x_f))]
        i = 0

        phi = (Phi0(m, t))
        # while phi>e and not np.allclose(array(x_f[1][0]), array(x_f[0][0]),
e):
        while kvcf<600:

                x_f.sort(key=lambda p:p[1])
                xc = (x_f[0][0] + x_f[1][0])/2
                tt = T(x_f[0][0], h, g)
                if phi-tt > 0:
                        pass
                else:
                        # min T(X) start from xNew
                        xNew = x_f[0][0]
                        xMinforT, incKvcf = nelderMid(phi, f=T, xStart=xNew)
                        x_f[2] = (xMinforT, f(xMinforT))
                        x_f.sort(key=lambda p:p[1])
                        xc = (x_f[0][0] + x_f[1][0])/2
                        xTrack = xTrack + [x_f[2][0]]
                        kvcf +=incKvcf + 1
                teta = 1
                project = lambda xh, xc, teta: xh + (1+teta)*(xc-xh)
                # x test
                xNew = project(x_f[2][0], xc, teta)
                fNew = f(xNew)
                kvcf += 1
                tt = T(xNew, h, g)
                if phi-tt > 0:
                        pass
                else:
                        # min T(X) start from xNew
                        xNew, incKvcf = nelderMid(phi, f=T, xStart=xNew)
                        fNew = f(xNew)
                        kvcf += incKvcf
                        xTrack.append(xNew)
                x_f.sort(key=lambda p:p[1])
                #compare
                if fNew<x_f[0][1]:
                        teta = gamma
                elif fNew<x_f[1][1]:
                        teta = alpha
                elif fNew<x_f[2][1]:
                        teta = beta[1]
                else:
                        teta = beta[0]
                if teta==beta[0]:
                        x1 = project(x_f[0][0], x_f[1][0], teta)
                        x2 = project(x_f[0][0], x_f[2][0], teta)
                        x_f = [(x1,f(x1)), (x2, f(x2)), x_f[0]]
                        kvcf += 2
                        x_f.sort(key=lambda p:p[1])
```

```
                        xNew = x_f[0][0]
                        xTrack.append(x1)
                        xTrack.append(x2)
                        thing = [x_f[i][0] for i in range(len(x_f))]
                        xc = sum(thing)/len(x_f)
                        phi = Phi(phi, m, r, thing, xc)
                        i+=1
                        continue
                else:
                        xc = (x_f[0][0] + x_f[1][0])/2
                        xNew = project(x_f[2][0], xc, teta)
                tt = T(xNew, h, g)
                if phi-tt > 0:
                        x_f[2] = (xNew, f(xNew))
                        xTrack.append(xNew)
                        kvcf += 1
                        x_f.sort(key=lambda p:p[1])
                else:
                        # min T(X) start from xNew
                        xMinforT, incKvcf = nelderMid(phi, f=T, xStart=xNew)
                        x_f[2] = (xMinforT, f(xMinforT))
                        kvcf += incKvcf + 1
                        xTrack.append(xMinforT)
                        x_f.sort(key=lambda p:p[1])
                if areEqual(xTrack[-1], xTrack[-2]) or areEqual(xTrack[-1],
xTrack[-3]):
                        teta=beta[1]
                        x1 = project(x_f[0][0], x_f[1][0], teta)
                        x2 = project(x_f[0][0], x_f[2][0], teta)
                        x_f = [(x1,f(x1)), (x2, f(x2)), x_f[0]]
                        kvcf += 2
                        x_f.sort(key=lambda p:p[1])

                        xNew = x_f[0][0]
                        xTrack.append(x1)
                        xTrack.append(x2)
                        thing = [x_f[i][0] for i in range(len(x_f))]
                        xc = sum(thing)/len(x_f)
                        phi = Phi(phi, m, r, thing, xc)
                        i+=1
                        continue
                i+=1
                thing = [x_f[i][0] for i in range(len(x_f))]
                if i > 1000:
                        break
                xc = sum(thing)/len(x_f)
                phi = Phi(phi, m, r, thing, xc)
                print(phi)


        x = [xTrack[i][0] for i in range(len(xTrack))]
        y = [xTrack[i][1] for i in range(len(xTrack))]
        plt.scatter(x,y)
        ar = np.arange(-2,3,0.01)
        # plt.plot(ar, [f1(ari) for ari in ar], c='yellow')
        plt.plot([f1(ari) for ari in ar], ar, c='yellow')
        ar = np.arange(-2,3,0.01)
        plt.plot([f3(ari) for ari in ar], ar, c='yellow')
        ar = np.arange(-2,2,0.01)
        plt.plot(ar, [f2(ari)+1 for ari in ar], c='yellow')
        print('i =', i)
        plt.xlabel('x1')
```

```
        plt.ylabel('x2')
        plt.show()

        return phi, kvcf



print(slidingDecent())
```

ДОДАТОК В (Ефективність модифікації м. Нелдера-Міда)

```
 def calc(x):
        restartVal = 40
        if x[-1][1]<0.00001:
                return []
        return x[:-1] + calc([(restartVal+x[0][0], x[-1][1]*0.37),
(restartVal+x[-1][0], 0.63*x[1][1])])


a = [(28,0.37), (117, 0.63)]
print(a[0])
print(a[1])
res = calc(a)
s, Mnew = 0, 0
for t in res:
        print(t)
        s+=t[1]
        Mnew += t[0]*t[1]
print('Класичний: M =', a[0][0]*a[0][1]+a[1][0]*a[1][1] )
print('Модифікований: M =',Mnew)
```