НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Курсова робота на тему
«*Дослідження методу Бройдена-Флетчера-Гольдфарба-Шанно*»
з дисципліни
«Методи оптимізації»

Виконав:
студент групи КМ-03
Гармаш О. Є.

Керівник:
старший викладач
Ладогубець Т. С.

Київ – 2023

ЗМІСТ

1 ВСТУП

Чисельні методи оптимізації є дуже сильними інструментами для розв’язання задач в багатьох галузях науки. Методи оптимізації дозволяють наближено знаходити екстремуми функції в задачах різної складності. Одним з таких методів є BFGS (метод Бройдена-Флетчера-Гольдфарба-Шанно), який, ймовірно є одним із найбільш широко використовуваних алгоритмів першого порядку для чисельної оптимізації, через свою ефективність та стійкість.

У даній курсовій роботі досліджується збіжність методу на визначеному класі задач, вивчаються його переваги та недоліки. За мету курсової роботи взято модифікація даному методу, тобто підбір оптимальних параметрів, для визначеного класу задач. Результати цього дослідження можуть бути корисними для тих, хто займається оптимізацією функцій Розенброка та вирішенням інших складних задач у різних областях.

2 ОСНОВНА ЧАСТИНА

2.1 ПОСТАНОВКА ЗАДАЧІ

Основною метою роботи є дослідження методу BFGS при вирішення задачі мінімізації функції Розенброка. Критерієм якості виступає кількість обрахувань значень цільової функції, необхідних для досягнення точки мінімуму заданої функції із заданою точністю. Необхідно розробити програму, яка на кожному кроці пошуку оптимуму використовувала якомога більше наявної інформації про значення функції, похідних, щоб зменшити кількість обчислень значень цільової функції.

Використовуючи евристичні підходи, слід досягнути найкращих результатів в залежності від:

1. Величини кроку h при обчисленні похідних.
2. Вигляду різницевої схеми обчислень похідних.
3. Виду методу одновимірного пошуку (ДСК-Пауелла або Золотого перетину)
4. Точності методу одновимірного пошуку.
5. Значення параметрів в алгоритмі Свена.
6. Вигляду критерію закінчення.
7. Наявності рестартів.

2.2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.2.1 Квазіньютонівські методи

Дані методи засновані на властивостях квадратичних функцій, та мають деякі позитивні риси методу Ньютона, проте використовують лише похідні першого порядку. В методах даного класу вектори напрямків пошуку мають вигляд:

$$s\left(x^{(k)}\right) = -A^{(k)}\nabla f(x^{(k)})$$

де $A^{(k)}\Delta$ - матриця порядку $N\,x\,N$, яка має назву *метрика*. Методи, які використовують дані вектори напрямків називаються *методами змінної метрики*, оскільки матриця A змінюється на кожній ітерації.

Для апроксимації матриці, що обернена матриці Гессе, використовується наступне рекуррентне співвідношення:

$$A^{(k+1)} = A^{(k)} + {A_c}^{(k)}$$

де ${A_c}^{(k)}$ – коректуюча матриця.

Для побудови апроксимації необхідно визначити Δx^k та $\Delta g^{(k)}$:

$$\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$$

$$\Delta g^{(k)} = g(x^{(k+1)}) - g(x^{(k)})$$

2.2.2 Метод Бройдена-Флетчера-Гольдфарба-Шанно

Метод BFGS реалізується згідно з наступною рекурентною формулою:

$$A^{(k+1)} = \left[I - \frac{\Delta x^{(k)} \Delta g^{(k)^T}}{\Delta x^{(k)^T} \Delta g^{(k)}}\right] A^{(k)} \left[I - \frac{\Delta x^{(k)} \Delta g^{(k)^T}}{\Delta x^{(k)^T} \Delta g^{(k)}}\right] + \frac{\Delta x^{(k)} \Delta x^{(k)^T}}{\Delta x^{(k)^T} \Delta g^{(k)}}$$

До числа головних переваг цього метода відносять завжди обов'язкову необхідність повернення до початкової ітерації алгоритму і відносно слабку залежність від точності обчислень одновимірного пошуку.

Нехай задана деяка функція $f(x_1, x_2)$ та стоїть задача оптимізації $minf(x_1, x_2)$, де $f(x_1, x_2)$ не є випуклою функцією та має неперервні перші похідні.

Тоді алгоритм методу BFGS виглядає наступним чином:

1. Ініціалізуємо початкову точку x_0, початкову величину похибки ε та $A_0 = I$ (лише на першій ітерації)
2. Знаходимо напрямок $s_k = -A_k \cdot \nabla f_k$
3. Визначаємо оптимальну довжину кроку λ_k за допомогою МОП
4. Якщо довжина кроку не задовільняє критерії $\lambda_k \leq \varepsilon$, тоді відбувається рестарт $A_k = I$ та повертаємося до кроку 1.
5. Обчислюємо $x_{k+1} = x_k + \lambda_k \cdot s_k$
6. Перевірка виконання критерію закінчення

$$\begin{cases} \frac{\|x_{k+1} - x_k\|}{\|x_k\|} \leq \varepsilon \\ |f(x_{k+1}) - x_k| \leq \varepsilon \end{cases} \text{ або } \|\nabla f(x_k)\| \leq \varepsilon$$

7. Визначаємо $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ та $\Delta g^{(k)} = g(x^{(k+1)}) - g(x^{(k)})$
8. Оновлюємо гессіан функції, за допомогою рекуретної формули BFGS

$$A^{(k+1)} = \left[I - \frac{\Delta x^{(k)} \Delta g^{(k)^T}}{\Delta x^{(k)^T} \Delta g^{(k)}}\right] A^{(k)} \left[I - \frac{\Delta x^{(k)} \Delta g^{(k)^T}}{\Delta x^{(k)^T} \Delta g^{(k)}}\right] + \frac{\Delta x^{(k)} \Delta x^{(k)^T}}{\Delta x^{(k)^T} \Delta g^{(k)}}$$

9. Повертаємося до кроку 1.

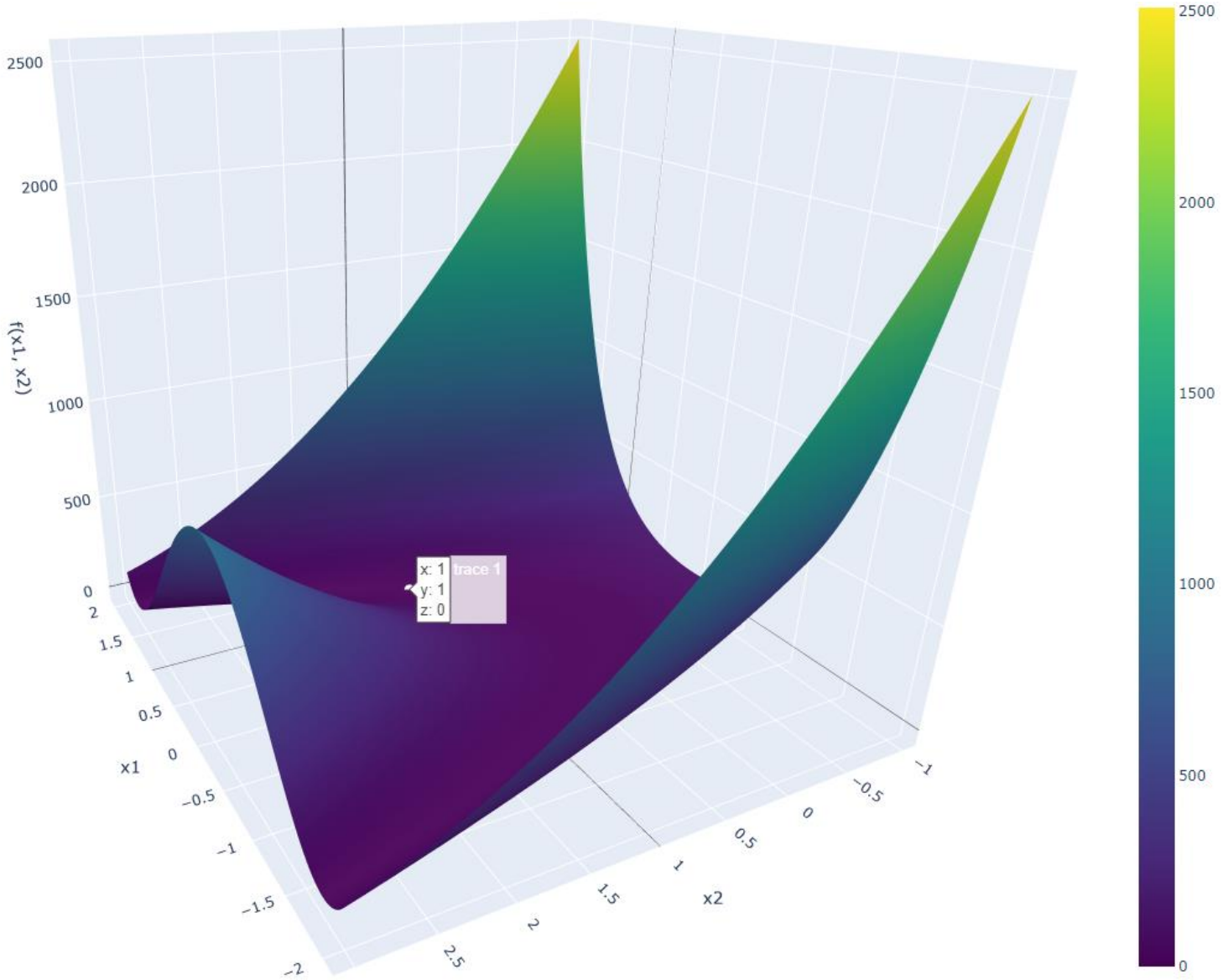
2.3 ПРАКТИЧНА ЧАСТИНА

Для поставленої задачі дослідження збіжності методу BFGS при мінімізації функції Розенброка в залежності від заданих параметрів використана еврістична методика, яка полягає в тому, що ми фіксуємо довільні значення параметрів і досліджуємо вплив значення кожного параметра на кількість обчислень цільової функції та похибку обчислення, при зафіксованих інших параметрах.

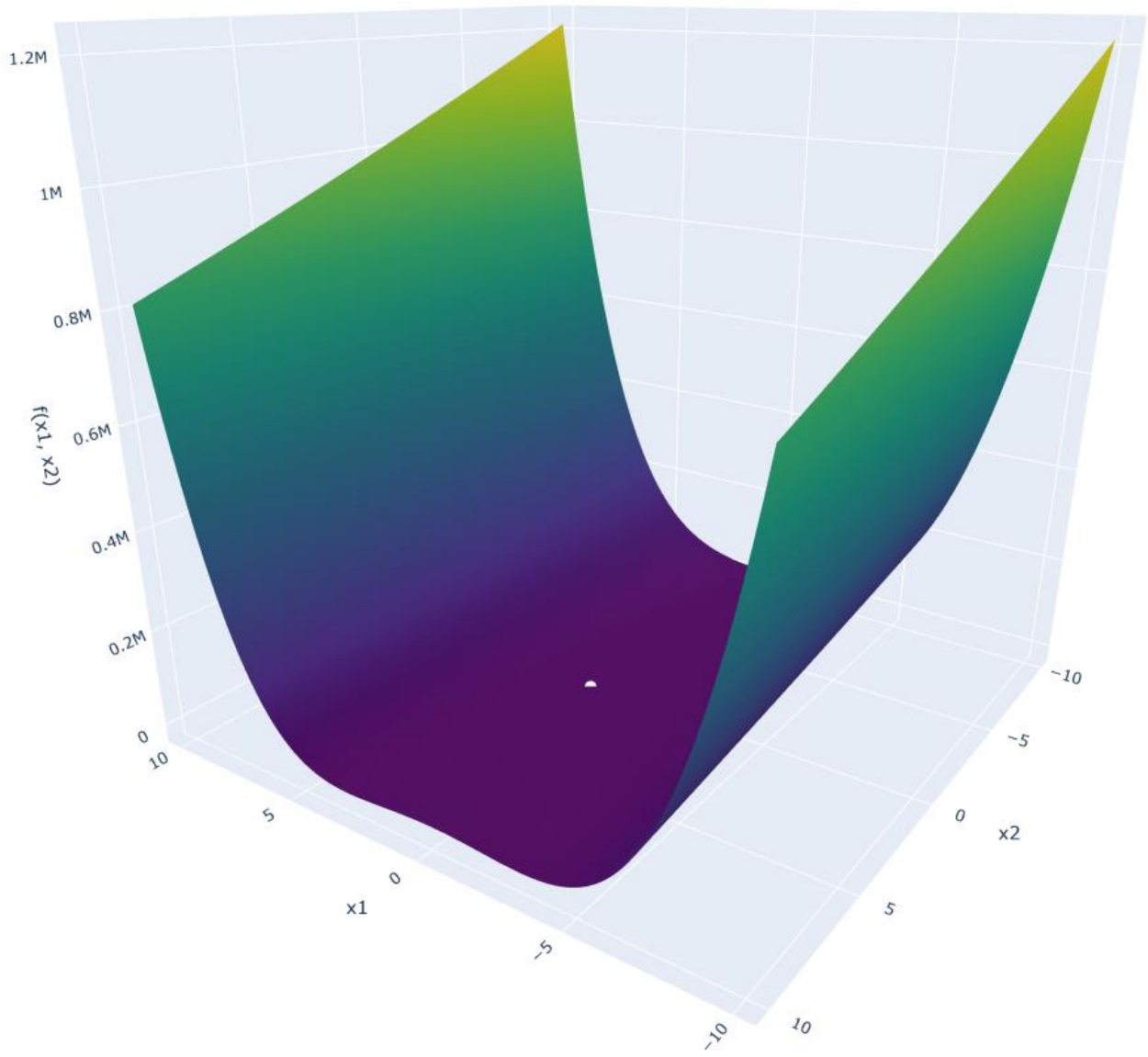
Даний підхід не дозволить підібрати оптимальні параметри, бо він залежить від порядку підбору параметрів, проте за мету дослідження взято модифікація методу під заданий клас задач, тому нам буде достатньо вибрати довільний порядок підбору параметрів.

Також було розроблено програмне забезпечення мовою *Python* з використанням допоміжних бібліотек (які лише пришвидшують роботу з масивами, фактично використовують векторні операції, замість ітераційних), це *numpy, scipy* та *matplotlib, seaborn* для візуалізації результатів.

Задана функція Розенброка $f(x_1, x_2) = 100 \cdot (x_1^2 - x_2)^2 + (x_1 - 1)^2$. Початкова точка $x^{(0)} = (-1.2, 0)$. Візуалізуємо функцію на деякому проміжку та покажемо мінімум:



Графік 1. Задана функція Розенброка на проміжку $x_1 \in [-2; 2], x_2 \in [-1; 3]$



Графік 2. Задана функція Розенброка на проміжку $x_1 \in [-10; 10], x_2 \in [-10; 10]$

В даному дослідженні параметри ініціалізуються наступними значеннями:

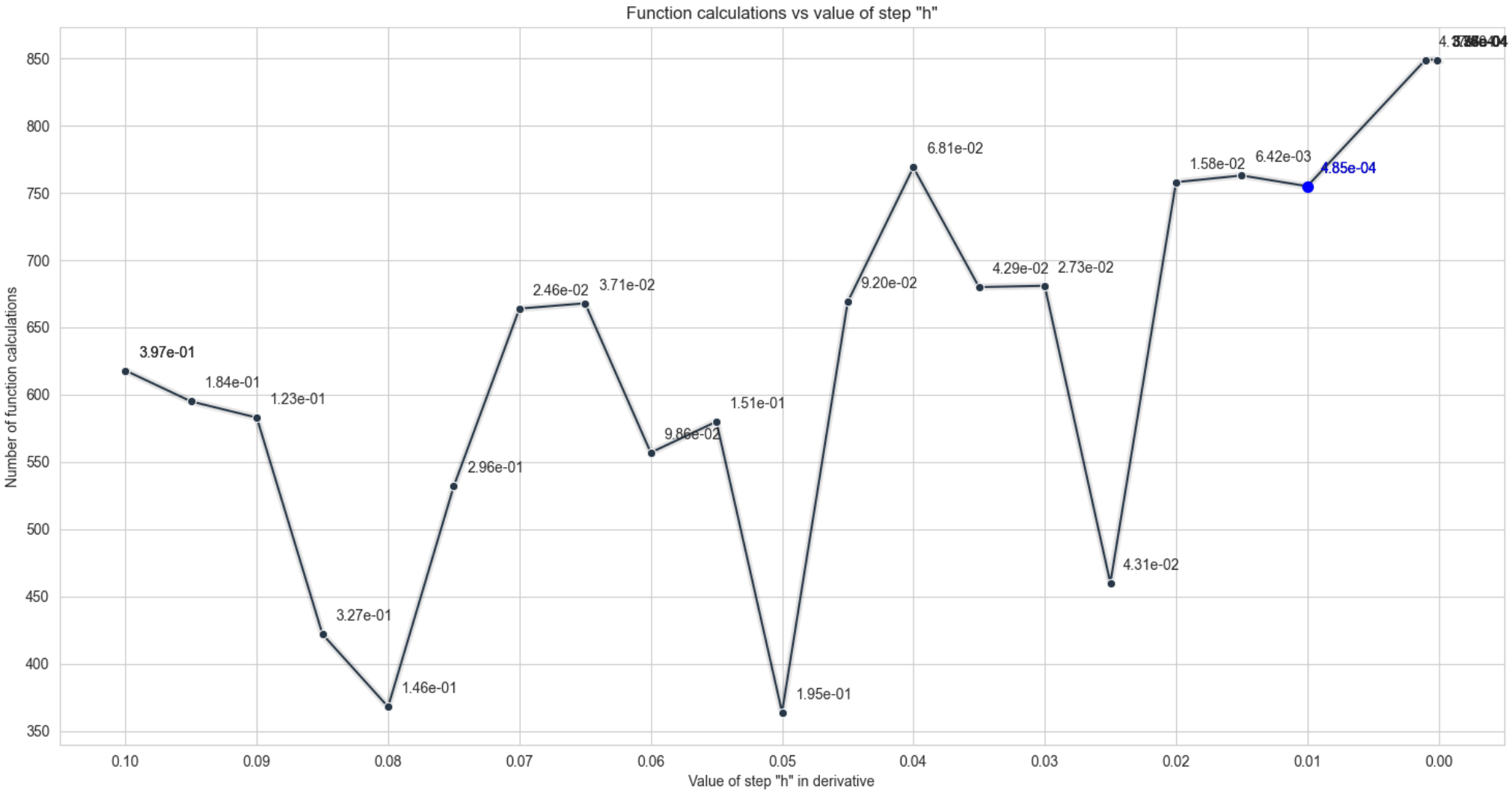
$\boldsymbol{\varepsilon_{BFGS} = 10^{-3}}$	(*величина похибки методу BFGS*)
$\boldsymbol{h = 0.01}$	(*величина кроку при обчисленні похідних)*
$\boldsymbol{schema = 'central'}$	(*вид різницевої схеми обчислення похідних: 'central, 'right', 'left'*)
$\boldsymbol{criterion = 'delta'}$	(**вид критерію закінчення методу BFGS: 'delta'* [1]*, 'norm'* [2])
method = 'golden'	(*МОП для визначення довжини кроку: 'golden_section', 'dsk_powell'*)
$\boldsymbol{\varepsilon_{method} = 10^{-2}}$	(*точність одновимірного методу*)
$\boldsymbol{\lambda_0 = 0}$	(*початкова довжина кроку в алгоритмі Свенна*)
$\boldsymbol{delta_{cf} = 0.01}$	(*значення параметру в алгоритмі Свенна* $\pm\Delta\lambda$)

Таблиця 1. Початкові значення параметрів

[1]*вид критерію "delta" означає* $\begin{cases} \frac{\|x_{k+1}-x_k\|}{\|x_k\|} \le \varepsilon \\ |f(x_{k+1}) - x_k| \le \varepsilon \end{cases}$

[2]*вид критерію "norm" означає* $\|\nabla f(x_k)\| \le \varepsilon$

Почнемо підбір параметрів **з величини кроку при обчисленні похідних**



Графік 3. Графік залежності величину кроку при обчисленні похідних від кількості обчислень цільової функції

З попереднього графіка відносно невелика кількість обчислень функції спостерігається при значенні кроку 0.01, причому значення функції в точці найменше з усіх $f(x_1, x_2) = 4.85E - 4$.

Значення кроків $h = 10^{-3}, 10^{-4} \dots 10^{-10}$, складно відобразити на одному графіку, тому виведемо наступну таблицю:

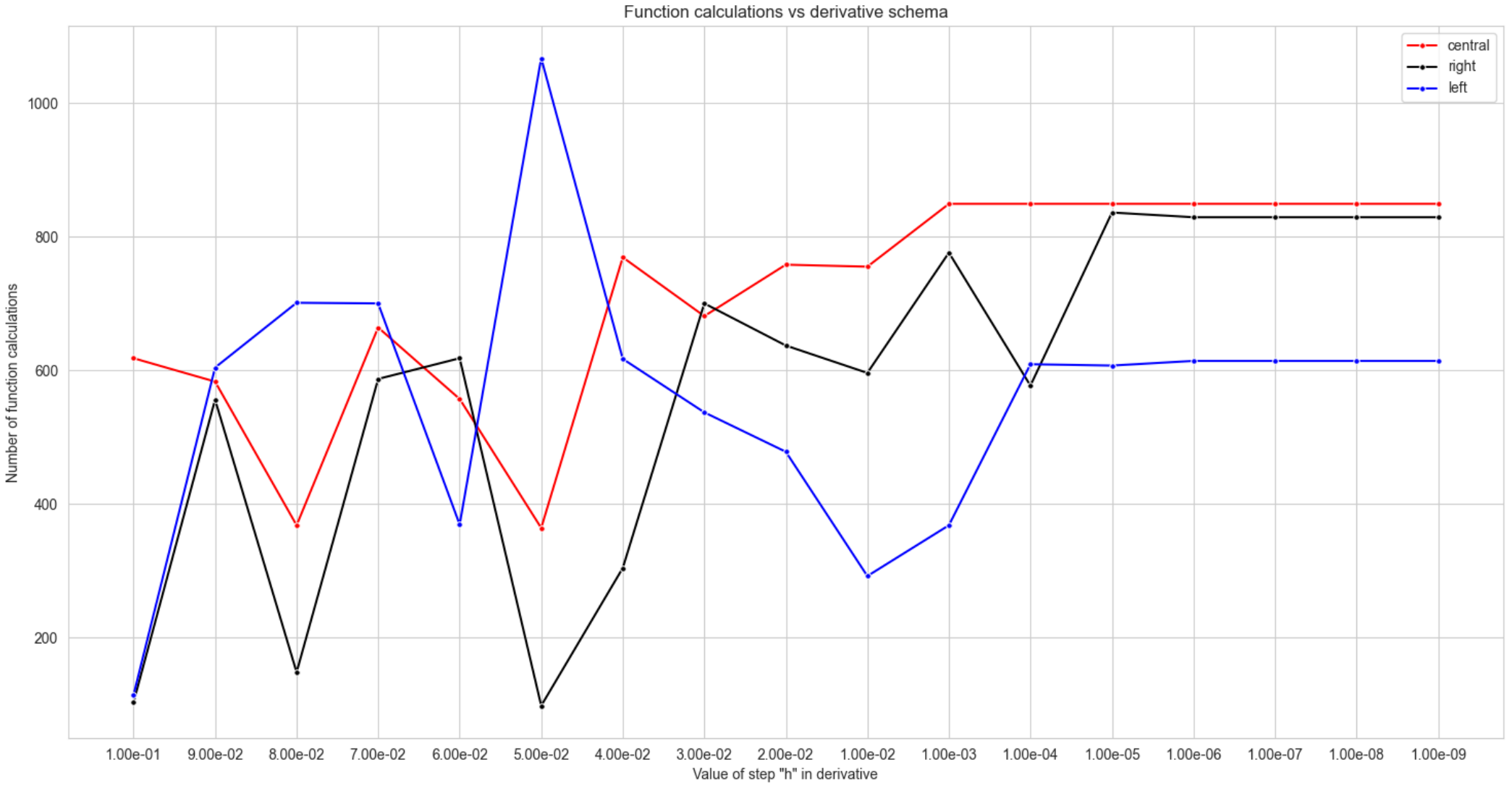
$\boldsymbol{h}$	$\boldsymbol{h}$	$[x_1, x_2]$	$f(x_1, x_2)$	$\boldsymbol{restarts}$
0.1	[0.38582893 0.13489511]	0.396718992	0	618
0.05	[0.56742502 0.31314318]	0.194914423	0	364
0.01	[0.98451901 0.96771058]	0.000485242	0	755
0.001	[0.99182143 0.98183755]	0.000417403	0	849
0.0001	[0.99229169 0.9828636]	0.000375971	0	849
0.00001	[0.99230354 0.9828828]	0.000377329	0	849
0.000001	[0.99287113 0.98410901]	0.000334429	0	849
0.0000001	[0.99285244 0.98407009]	0.000335304	0	849

Хоч і спостереження з графіка справдилися, візьмемо величину кроку $h = 0.0001$, бо вона має хоч і більшу кількість обчислень функції ніж $h = 0.01$ проте значення функції в точці мінімума менше. Також варто сказати, що чим менший крок, тим менше спостерігається зміна значень цільової функції.

Отже маємо наступні параметри (кольором виділені найкращі значення):

$\boldsymbol{\varepsilon_{BFGS}}$	$1.00E - 03$
$\boldsymbol{h}$	0.0001
$\boldsymbol{schema}$	$central$
$\boldsymbol{criterion}$	$delta$
$\boldsymbol{method}$	$golden$
$\boldsymbol{\varepsilon_{method}}$	0.001
$\boldsymbol{\lambda_0}$	0
$\boldsymbol{delta_{cf}}$	0.01

Підбір **різницевої схеми при обчисленні похідних**



Графік 4. Графік залежності величину кроку при обчисленні похідних від кількості обчислень цільової функції для кожної різницевої схеми

На графіку спостерігається падіння кількості обчислень функції при $h = 10^{-2}$, також вибиремо з графіка точки з найменшою кількістю обчислень в залежності від схем та подивимося яке значення цільової функції отримуємо в цих точках:

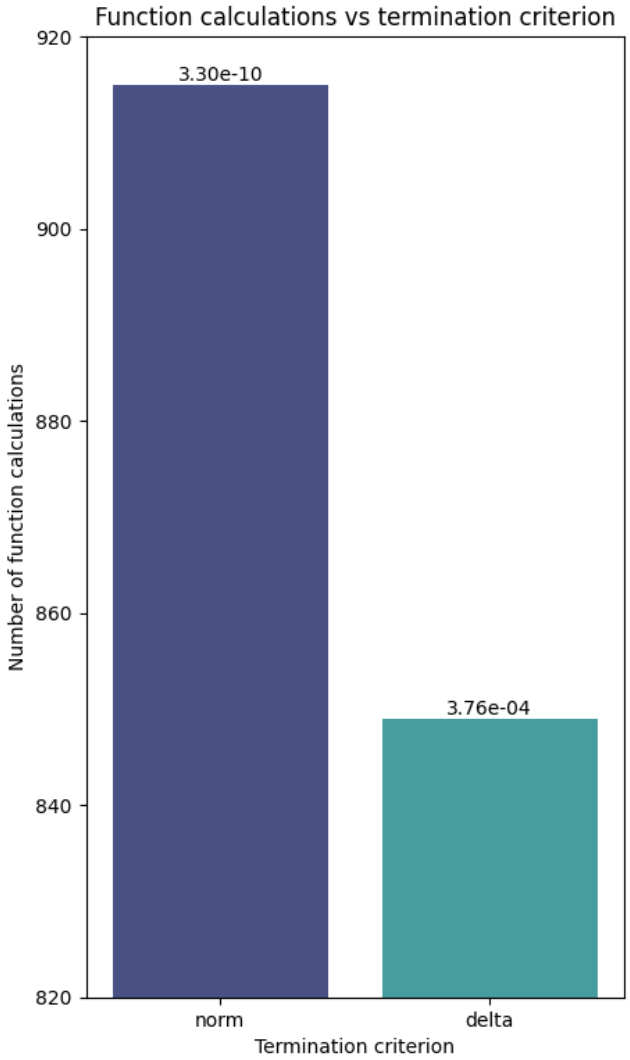
scheme	h	$[x_1, x_2]$	$f(x_1, x_2)$	restarts	f_iter
right	0.08	[-0.50654309 0.24888454]	2.275603152	0	148
right	0.05	[-0.74987094 0.58690377]	3.122551234	0	98
central	0.01	[0.98451901 0.96771058]	0.000485242	0	755
right	0.01	[0.92015515 0.83853235]	0.013022577	0	577
left	0.01	[0.74267011 0.54538952]	0.07002478	0	292
right	0.0001	[0.92015515 0.83853235]	0.013022577	0	577
central	0.0001	[0.99229169 0.9828636]	0.000375971	0	849

Очевидно, що надаємо перевагу центральній різницевій схемі, з параметром $h = 0.0001$, хоча вона і має найбільшу кількість обчислень, проте вона більш точна.

Отже маємо наступні параметри (кольором виділені найкращі значення):

$\boldsymbol{\varepsilon_{BFGS}}$	$1.00E-03$
$\boldsymbol{h}$	0.0001
$\boldsymbol{schema}$	*central*
$\boldsymbol{criterion}$	*delta*
$\boldsymbol{method}$	*golden*
$\boldsymbol{\varepsilon_{method}}$	0.001
$\boldsymbol{\lambda_0}$	0
$\boldsymbol{delta_{cf}}$	0.01

Підбір **критерію закінчення**



Графік 5. Вплив критерію закінчення

В даному дослідженні використовувалися два критерії закінчення:

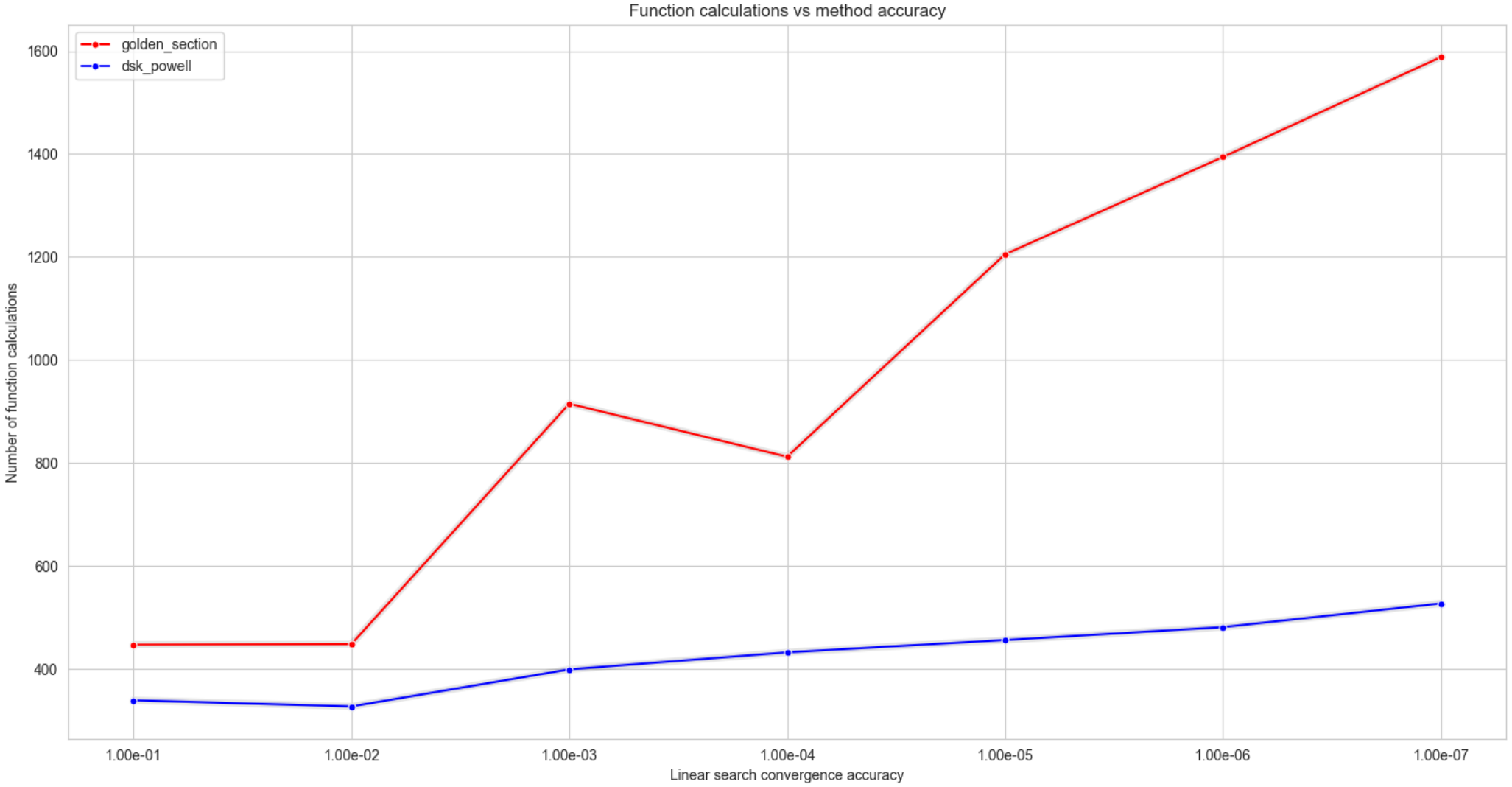
norm: $\|\nabla f(x_k)\| \leq \varepsilon$, та ***delta***: $\begin{matrix} \frac{\|x_{k+1}-x_k\|}{\|x_k\|} \leq \varepsilon \\ |f(x_{k+1}) - x_k| \leq \varepsilon \end{matrix}$.

На графіку зліва можна побачити (див. масштабування), що кількість обчислень цільової функції не відрізняється суттєво, проте величина мінімуму функції чисельно обчислена даним методом при критерії *norm* дорівнює $3.3 * 10^{-10}$, на відміну від критерію *delta*, де мінімум функції дорівнює $3.76 * 10^{-4}$.

Очевидно, що в подальному будемо використовувати критерій *norm*. Отже маємо наступні параметри (кольором виділені найкращі значення):

$\boldsymbol{\varepsilon_{BFGS}}$	$1.00E-03$
$\boldsymbol{h}$	0.0001
schema	*central*
criterion	*norm*
method	*golden*
$\boldsymbol{\varepsilon_{method}}$	0.001
$\boldsymbol{\lambda_0}$	0
$\boldsymbol{delta_{cf}}$	0.01

Підбір **методу одновимірного пошуку та його точності**



Графік 6. Графік залежності точності МОП від кількості обчислень цільової функції

З побудованого графіку можна спостерігати перевагу методу ДСК-Пауелла над методом золотого перетину, оскільки зі збільшенням точності кожного методу, кількість ітерацій ДСК-Пауелла зростає «дуже повільно», в той час для методу золотого перетину, наприклад різниця кількості обчислень цільової функції для точності 10^{-2} та 10^{-3} становить приблизно 500 обчислень, в той час, як для метода ДСК-Пауелла приблизно 50.

Побудуємо наступну порівняльну таблицю:

ε_{golden}	$\varepsilon_{DSK-Powell}$	$[x_1, x_2]$	$[x_1, x_2]$	$f(x_1, x_2)$	$f(x_1, x_2)$	*f_iter*	*f_iter*
0.1	0.1	[0.99994527 0.99988983]	[0.99998533 0.99997892]	3.05E-09	7.04E-09	447	339
0.01	**0.01**	[0.99999788 0.99999679]	**[1.00000542 1.00001125]**	1.12E-10	**4.63E-11**	448	**327**
0.001	0.001	[0.99998249 0.99996547]	[1.00013711 1.00025877]	3.30E-10	4.27E-08	915	399
0.0001	0.0001	[0.99988146 0.99976924]	[0.99992688 0.99985955]	1.80E-08	8.69E-09	812	432
0.00001	0.00001	[1.00005017 1.00009539]	[0.99992458 0.99985509]	4.98E-09	9.21E-09	1205	456
0.000001	0.000001	[1.00015856 1.00029822]	[0.99992421 0.99985438]	6.10E-08	9.29E-09	1394	481

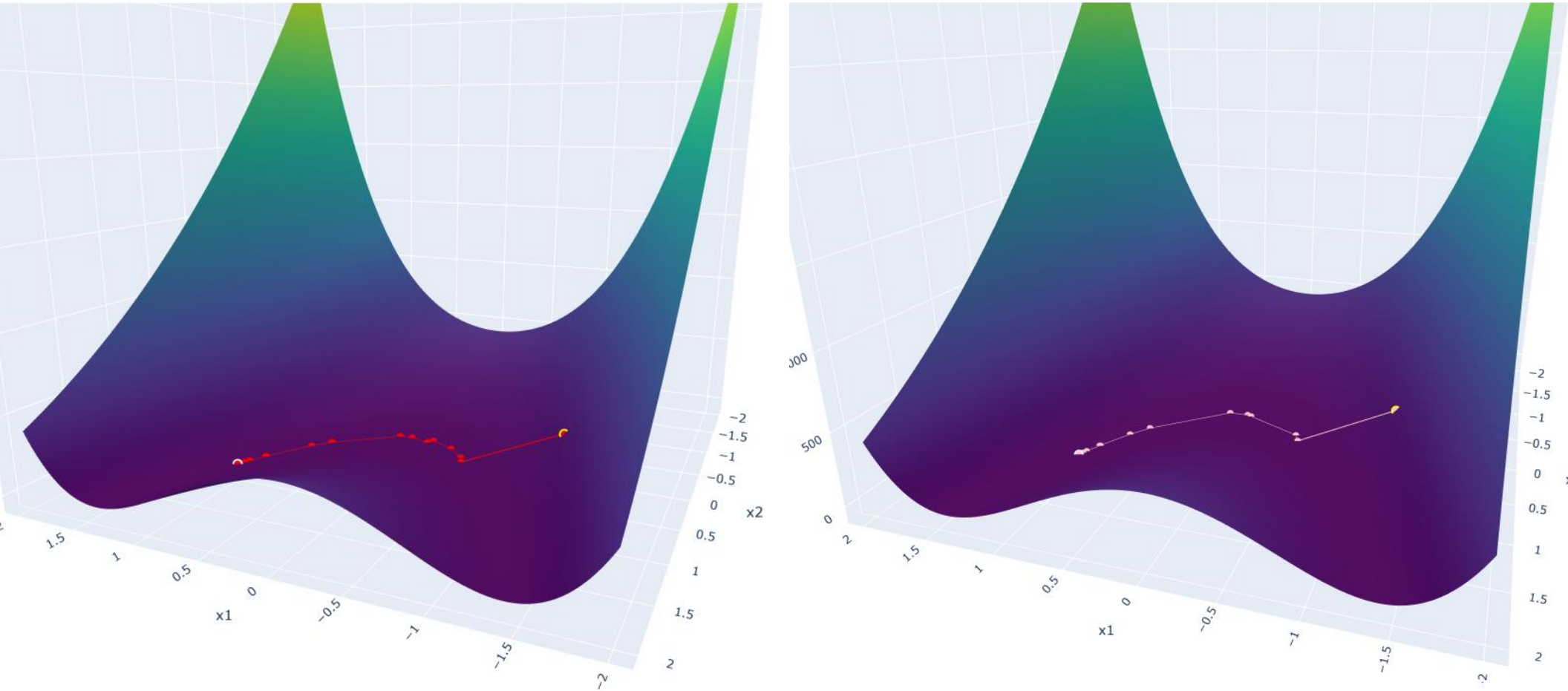
**червоним кольором зображені результати методу золотого перетину, жовтим – ДСК-Пауелла.*

З порівняльної таблиці бачимо, що краще себе показує метод ДСК-Пауелла з точністю 0.01. Всього лише за 327 обчислень цільової функції він досягає значення мінімуму $\mathbf{4.63e^{-11}}$.

Справді, якщо вдатися до теорії, то метод ДСК-Пауелла більш точний, але не завжди працює, бо використовує аналогічні різницеві схеми, як при використанні похідних і може відбуватися ділення на нуль.

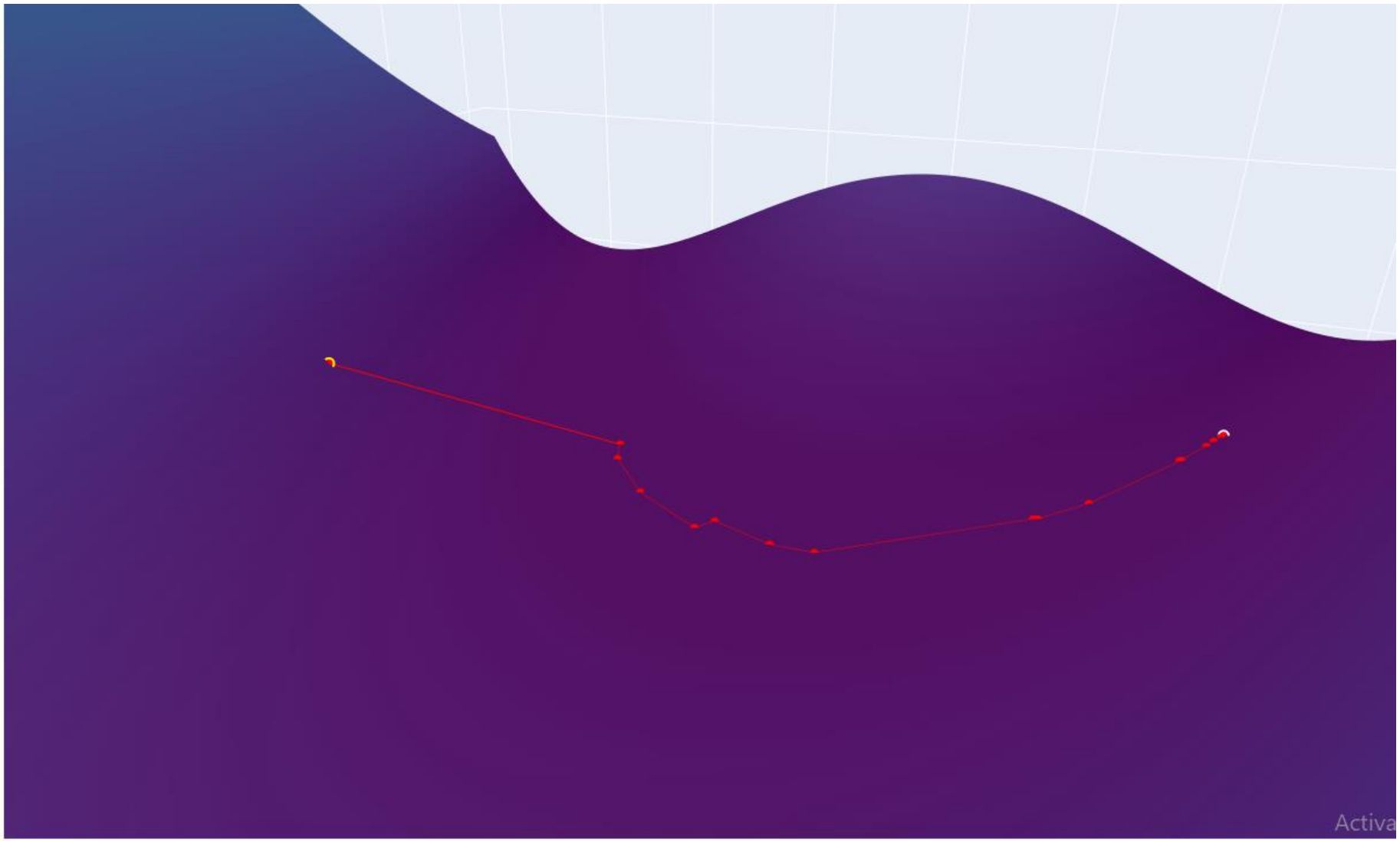
Порівняємо траекторію руху методу BFGS при використанні різних МОП

*червона траекторія – DSK-Powell, рожева – Golden Section, початок руху – жовта точка (справа), мімум – біла точка

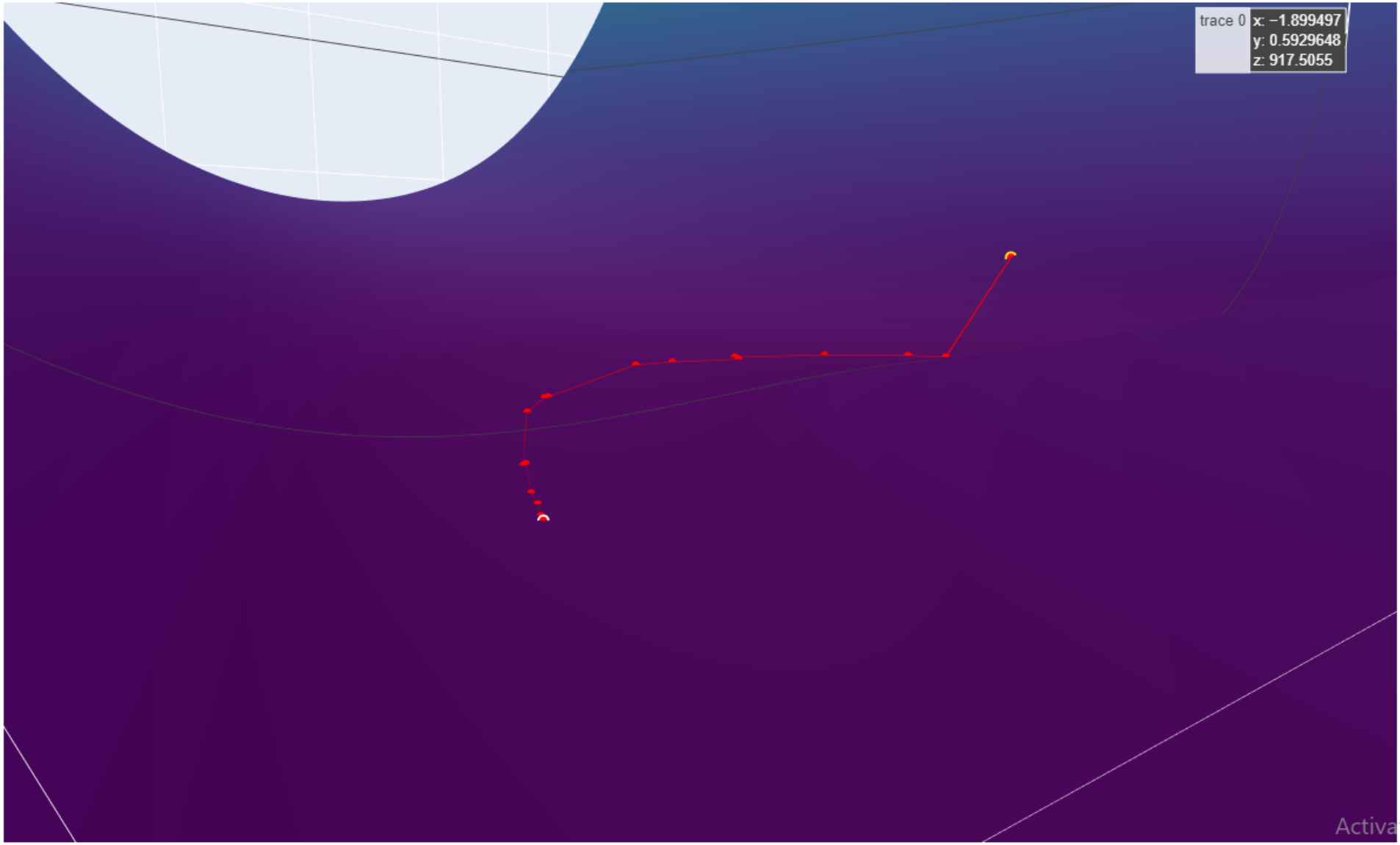


Графік 7.1 Графіки траекторії руху при використанні різних МОП

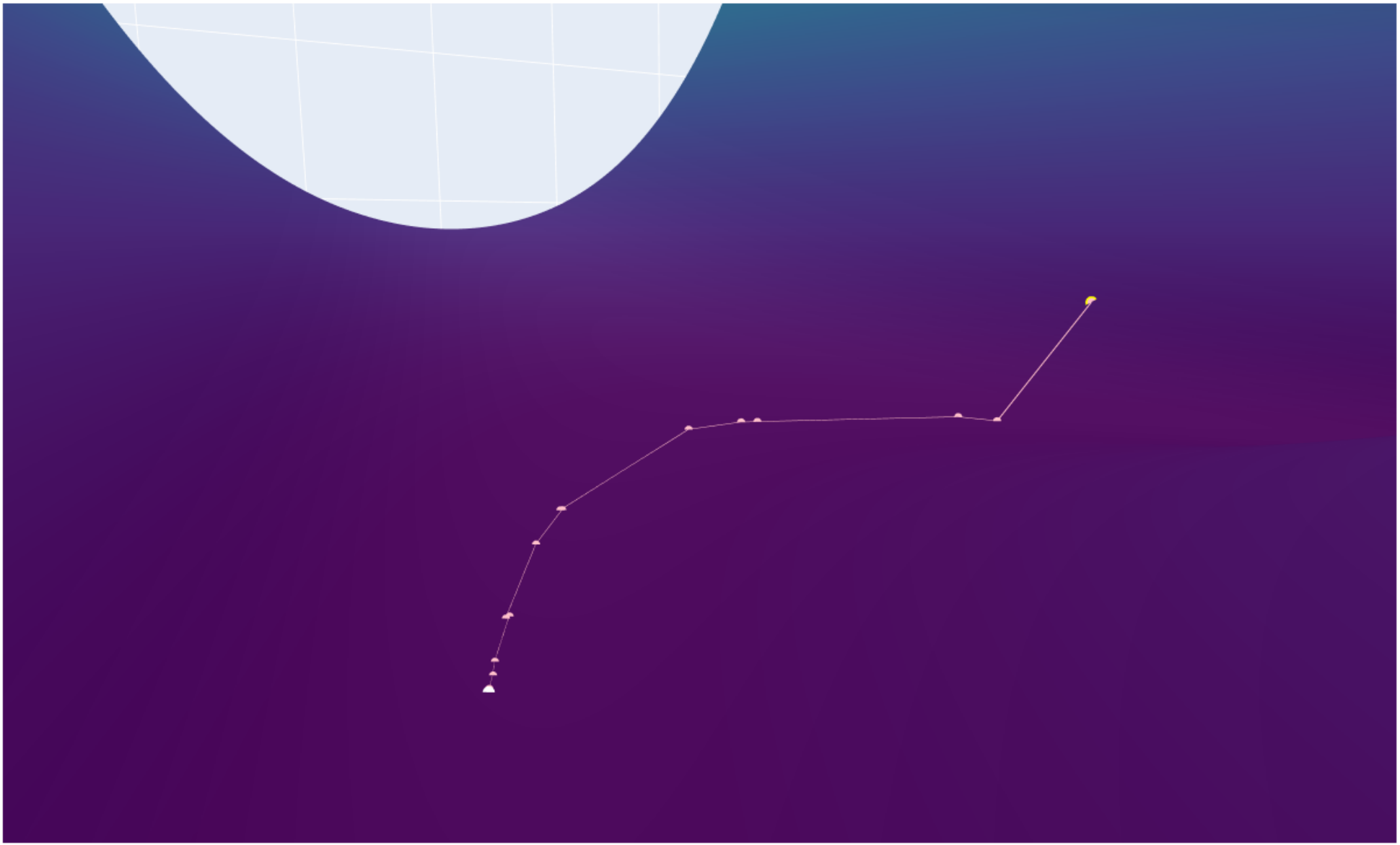
Можна наглядно побачити, як алгоритм шукає мінімум ярової функції (функції Розенброка).



Графік 7.2 Графік траекторії руху при використанні МОП - ДСК-Пауелла



Графік 7.3 Графік траєкторії руху при використанні МОП - ДСК-Пауелла



Графік 7.4 Графік траекторії руху при використанні МОП- золотий перетин

Отже маємо наступні параметри (кольором виділені найкращі значення):

$\boldsymbol{\varepsilon_{BFGS}}$	$1.00E-03$
$\boldsymbol{h}$	0.0001
$\boldsymbol{schema}$	$central$
$\boldsymbol{criterion}$	$norm$
$\boldsymbol{method}$	$dsk-powell$
$\boldsymbol{\varepsilon_{method}}$	0.01
$\boldsymbol{\lambda_0}$	0
$\boldsymbol{delta_{cf}}$	0.01

Підбір **значення параметрів в алгоритмі Свенна**

$\boldsymbol{\lambda_0}$	$[x_1, x_2]$	$f(x_1, x_2)$	$\boldsymbol{restarts}$	$\boldsymbol{f\_iter}$
0.01	[1.00000166 1.00000317]	4.63E-12	0	171
0.1	[1.0000473 1.00008905]	5.33E-09	0	296
0.12	[0.99998763 0.99997612]	2.27E-10	0	133
0.4	[0.99986878 0.99974352]	2.07E-08	0	197
1	[0.99994712 0.99989705]	3.58E-09	0	162

Бачимо, що при $\boldsymbol{\lambda_0 = 0.12}$ кількість обрахувань функції знижується майже на 200, порівняно з $\boldsymbol{\lambda_0 = 0,}$ причому значення мінімуму майже не змінилося, тому доцільно зафіксувати значення $\boldsymbol{\lambda_0 = 0.12.}$ Одразу почнемо підбір значення $\Delta\boldsymbol{\lambda}$**:**

$\boldsymbol{delta_{cf}}$	$[x_1, x_2]$	$f(x_1, x_2)$	$\boldsymbol{restarts}$	$\boldsymbol{f\_iter}$
0.01	[0.99998763 0.99997612]	2.27E-10	0	133
0.08	[0.99997626 0.99995321]	6.10E-10	0	128
0.16	[1.00000469 1.0000024]	4.88E-09	0	164
0.3	[0.99974019 0.99949947]	1.04E-07	0	974
0.7	[0.99895646 0.99793168]	1.12E-06	0	312

В даному випадку є два фаворити на роль кращого параметра $\Delta\boldsymbol{\lambda} = 0.01$ залишає кількість обчислень функції ту ж саму, проте має менше значення мінімуму, в той час як $\Delta\boldsymbol{\lambda = 0.08}$ має на 5 меншу кількість обчислень функції, проте має більше значення мінімуму.

Даний підхід еврістичний, тому в даному випадку немає переваг обрати конкретну $\Delta\boldsymbol{\lambda}$, тому візьмемо $\Delta\boldsymbol{\lambda} = \mathbf{0.08}$.

Отже маємо **фінальні** параметри:

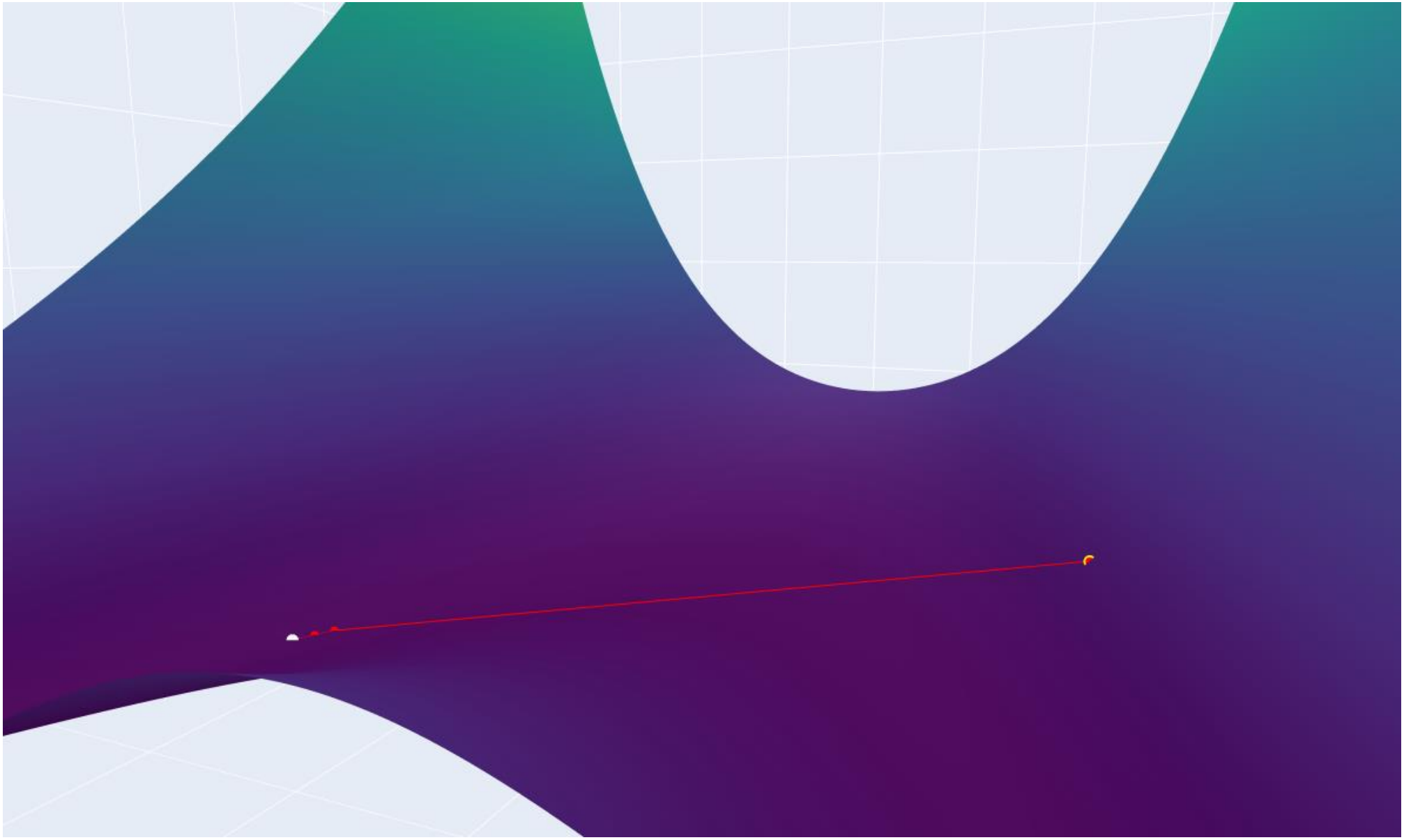
$\boldsymbol{h}$	0.0001
$\boldsymbol{schema}$	$central$
$\boldsymbol{criterion}$	$norm$
$\boldsymbol{method}$	$dsk - powell$
$\boldsymbol{\varepsilon_{method}}$	0.01
$\boldsymbol{\lambda_0}$	0.12
$\boldsymbol{delta_{cf}}$	0.08

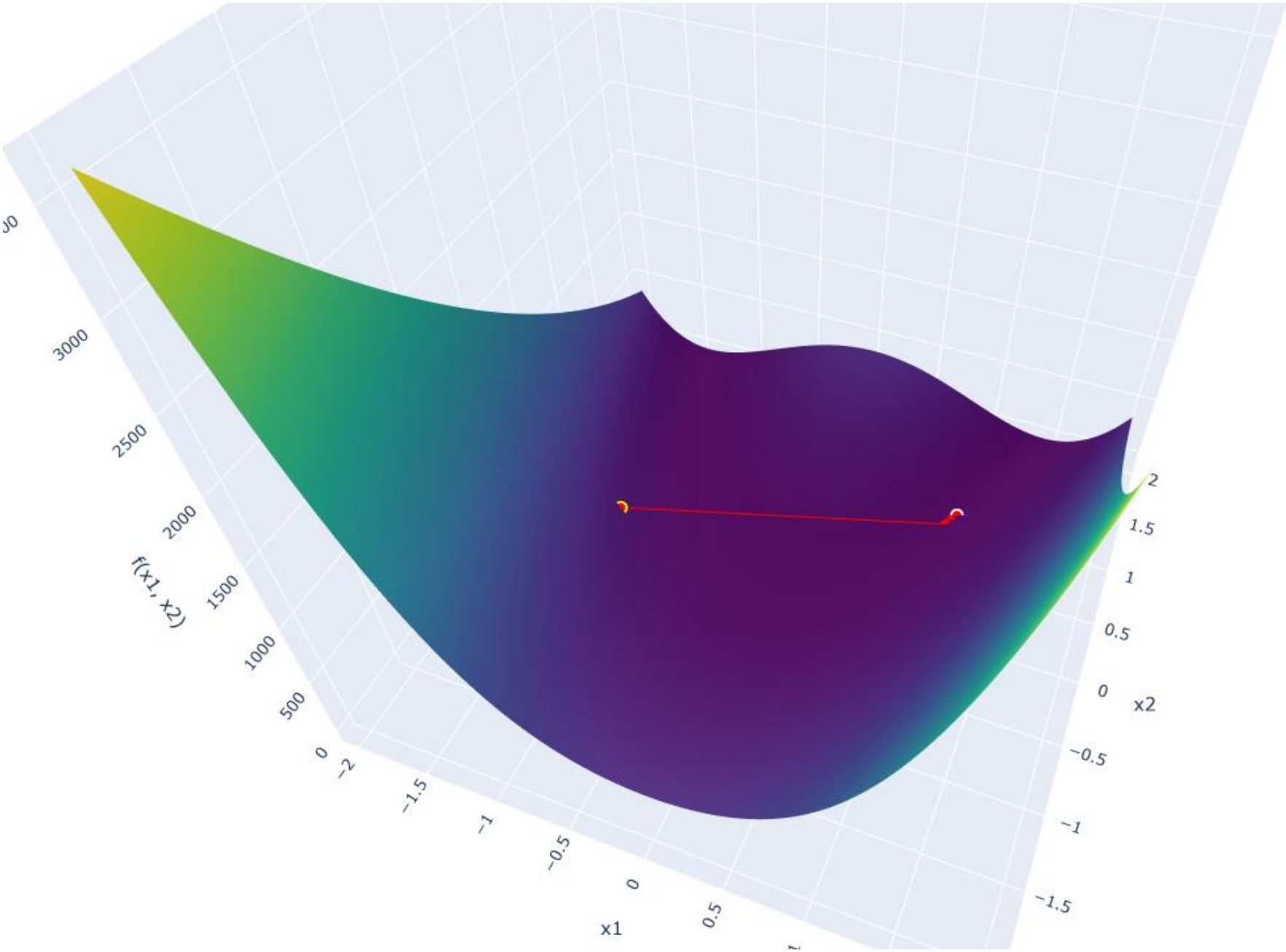
Дослідимо метод з заданими параметрами на **наявність рестартів**:

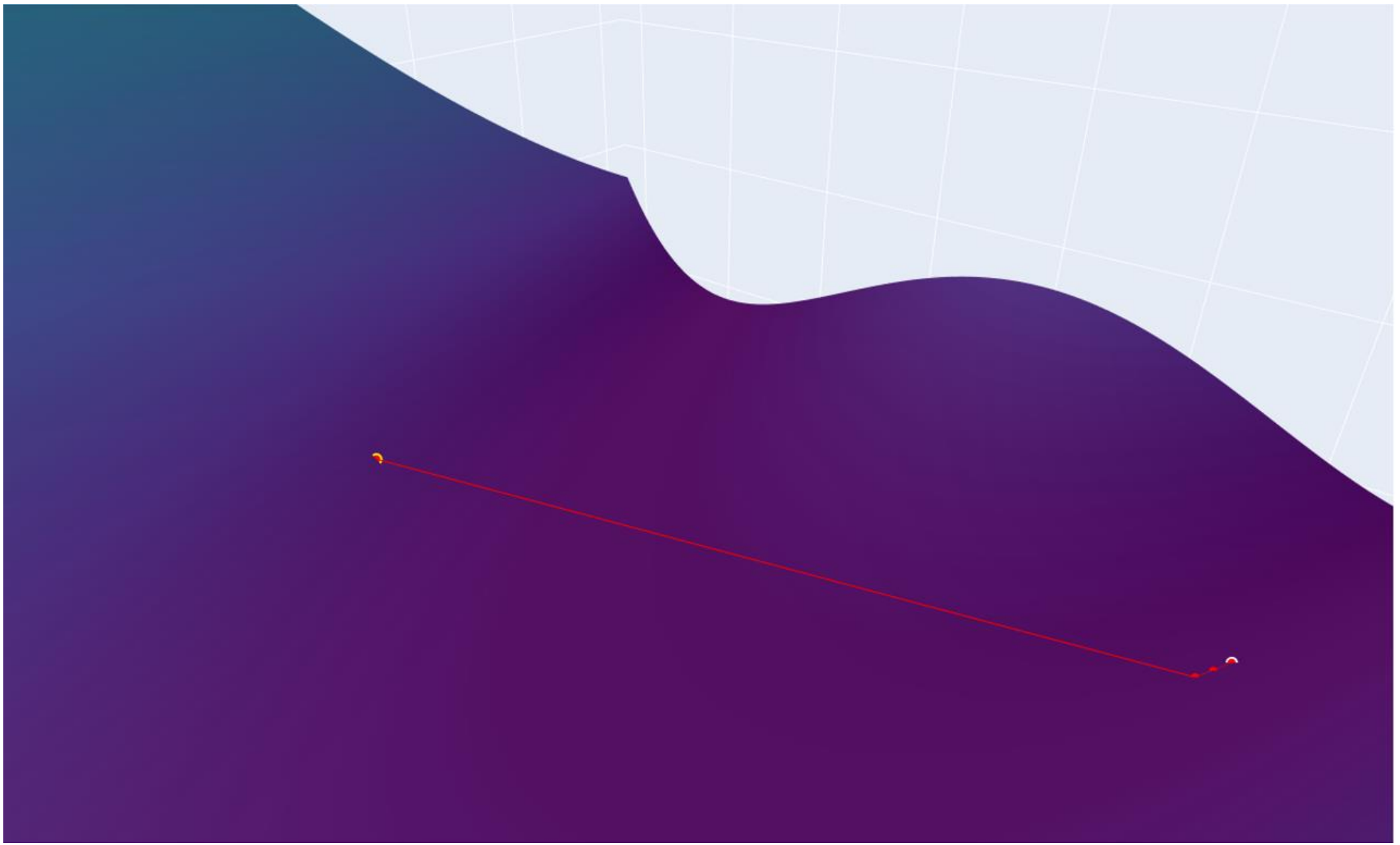
```
lambda_k:  0.003078409210692669
lambda_k:  0.0018208026358042528
lambda_k:  216.6021613448116
lambda_k:  2.6835789536095715
lambda_k:  0.4964068544582564
lambda_k:  0.12
Point:  [0.99997626 0.99995321]
f(x) =  6.102143615817427e-10
r_count:  0
f_iter:  128
```

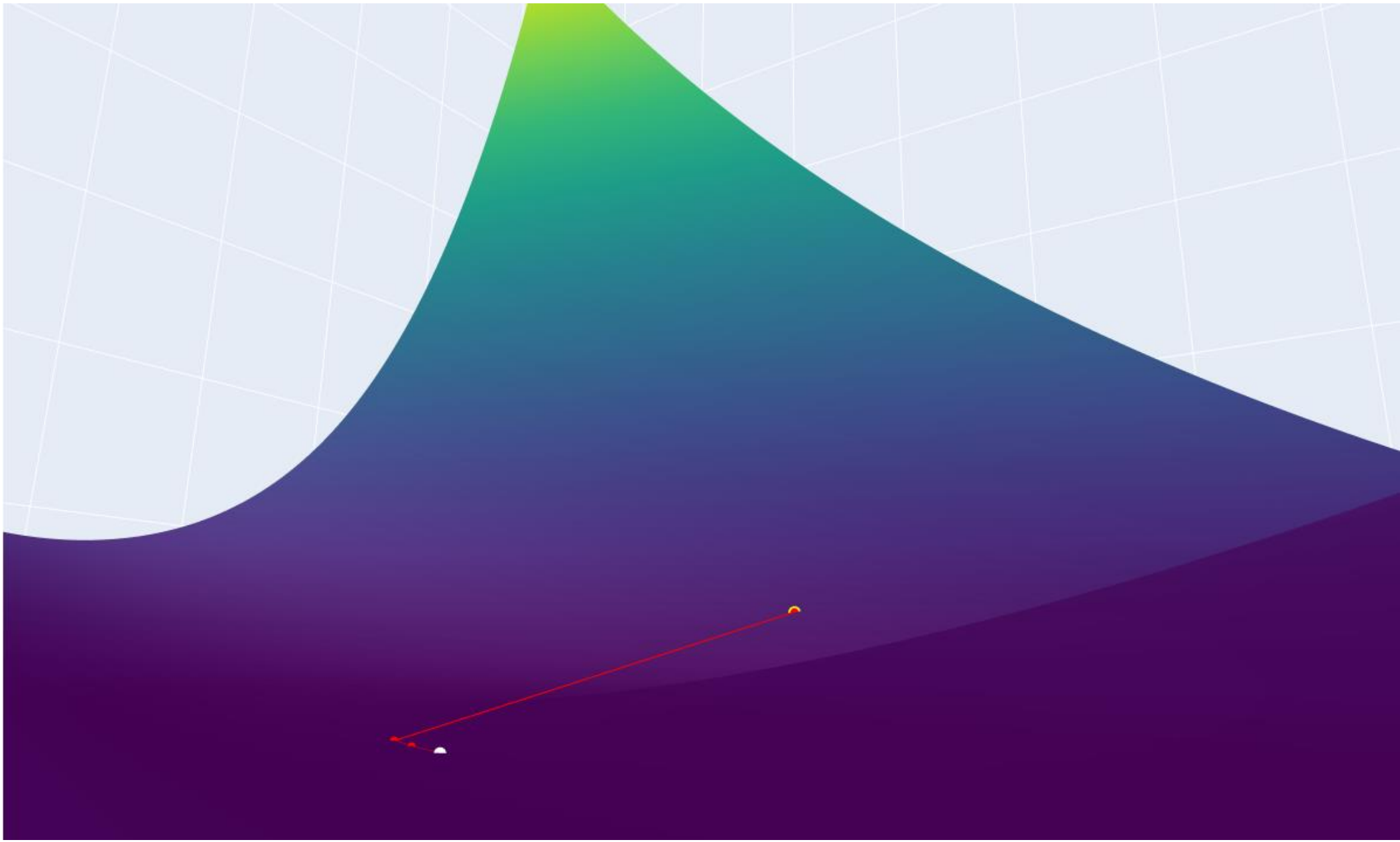
Як бачимо, параметри, які ми визначили в процесі підбору виявилися такими, що довжина кроку $\boldsymbol{\lambda}$ ніколи не приймає нульове або наближене до нуля значення, а тому метод не застосовує рестарти ($r_{count} = 0$).

Траєкторія руху з **фінальними параметрами**

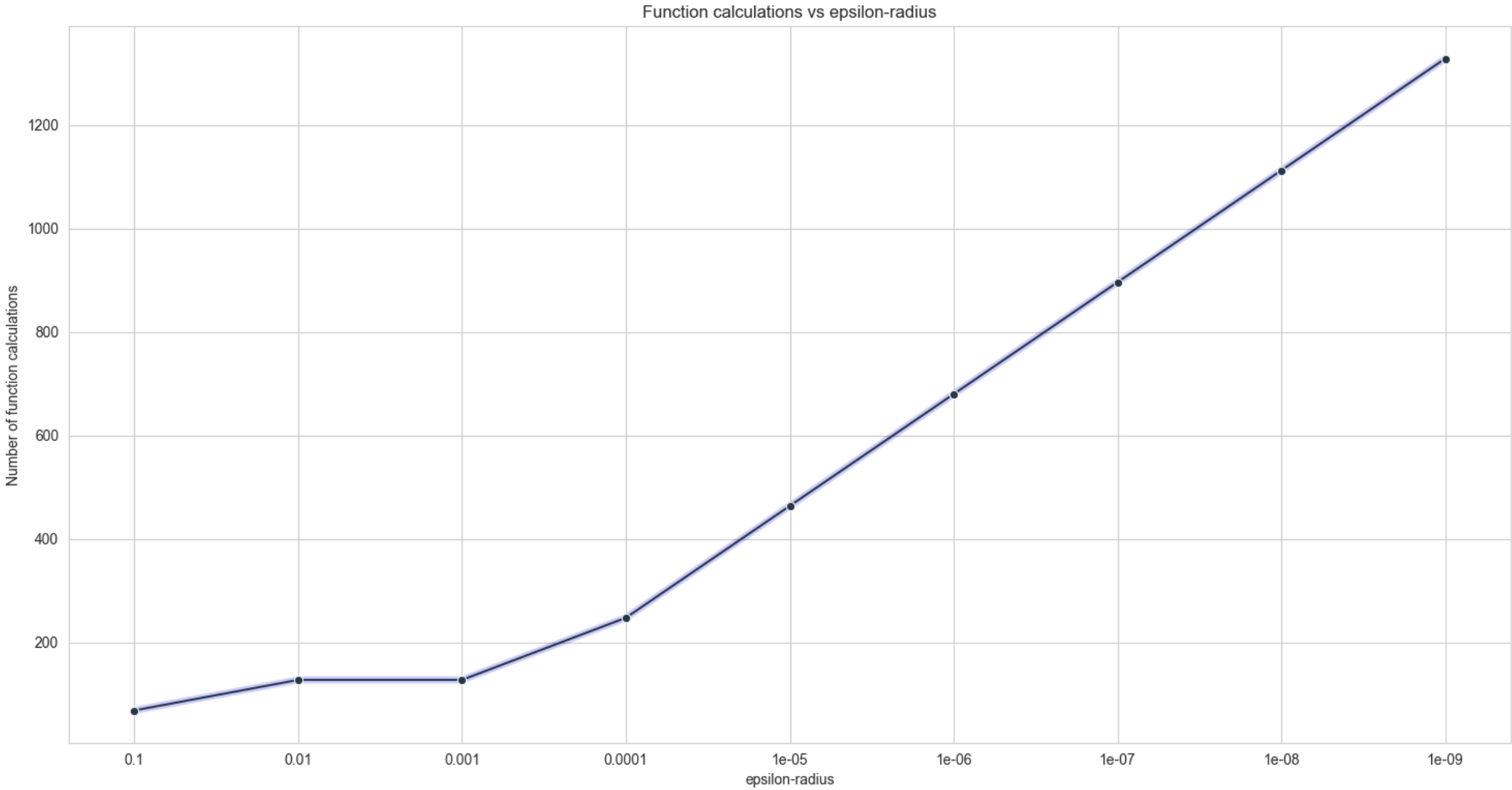



3000
2500
2000
1500
1000
500
0
f(x1, x2)
−2
−1.5
−1
−0.5
0
0.5
x1
2
1.5
1
0.5
0
−0.5
−1
−1.5
x2





Результуюча залежність КОЗЦФ від точності пошуку



Графік 8. Графік залежності кількості обчислень значень цільової функції метода BFGS від епсілон околу в фінальними параметрами

ВИСНОВКИ

В результаті виконання курсової роботи досліджено збіжність методу BFGS на визначеному класі задач (безумовна оптимізація функції Розенброка), досліджено його переваги та недоліки. В результаті аналізу та використання різних евристичних підходів метод було модифіковано для заданого класу задач.

Початкові параметри

$\boldsymbol{h}$	0.01
$\boldsymbol{schema}$	*central*
$\boldsymbol{criterion}$	*delta*
$\boldsymbol{method}$	*golden*
$\boldsymbol{\varepsilon_{method}}$	0.01
$\boldsymbol{\lambda_0}$	0
$\boldsymbol{delta_{cf}}$	0.01

Результуючі параметри

$\boldsymbol{h}$	0.0001
$\boldsymbol{schema}$	*central*
$\boldsymbol{criterion}$	*norm*
$\boldsymbol{method}$	*dsk − powell*
$\boldsymbol{\varepsilon_{method}}$	0.01
$\boldsymbol{\lambda_0}$	0.12
$\boldsymbol{delta_{cf}}$	0.08

Програма модифікує метод та наближає точку мінімума із точністю 10^{-3} за **128** обчислень цільової функції, в той час, як з початковими параметрами метод доходить до окола 10^{-3} точки мінімума за **755** обчислень.

В ході модифікації метода було використано всі його сильні сторони (наприклад рестарти) та підтверджено теоретичні дані щодо стійкості методу. Було визначено, що найбільший приріст зменшення КОЗЦФ залежав від вибору методу одновимірного пошуку, а також значення $\Delta\lambda$ в алгоритмі Свенна.

В даному дослідженні використовувалися два методи обчислення оптимальної довжини кроку: метод золотого перерізу та ДСК-Пауелла. Метод ДСК-Пауелла (*графік 6*) потребує набагато менше обчислень значень цільової функції зі збільшенням епсілон околу. Щодо вибору різницевої схеми

обчислення похідних, за результатами дослідження, центральна різницева схема виявилася набагато точнішою за інші схеми, причому різниця в кількості обчислень нівелюється різницею в точності наближення до точки мінімуму (як відомо центральна схема вимагає більше КОЗЦФ). При використанні градієнтного критерію закінчення досягалася точність приблизно на 5 порядків вища, аніж при використанні комбінованого критерію закінчення, причому різиця в кількості обчислень значень цільової функції сягала лише 60.

В результаті підбору фінальних параметрів було визначено залежність КОЗЦФ від точності пошуку і виявлено, що починаючи з точності $\varepsilon = 10^{-4}$ залежність приймає лінійний вигляд.

Також було розроблено програмне забезпечення мовою *Python* з використанням допоміжних бібліотек (які лише пришвидшують роботу з масивами, фактично використовують векторні операції, замість ітераційних), це *numpy, scipy* та *matplotlib, seaborn* для візуалізації результатів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «*Методи одновимірної оптимізації: Практикут з дисципліни Дослідження операцій*» - Ладогубець Т.С., Фіногенов Олексій Дмитрович, Факультет прикладної математики, «Київський Політехнічний Інститут», Україна, 2020
2. «Engineering Optimization. Methods and Applications» - G.V. Reklaitis, A. Ravindran, Chemical Engineering Purdue University, Industrial Engineering University of Oklahoma, 1983.
3. «*Broyden-Fletcher-Goldfarb-Shanno algorithm*» - Wikipedia, URL: https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno_algorithm
4. «Обчислювальна математика» - Прокопенко Ю. В., Татарчук Д. Д., «Київський Політехнічний Інститут», Україна, 2013.

ДОДАТОК 1

Лістинг *settings.py* (параметри методів)

```
# BFGS method parameters
e_BFGS = 10e-3          # convergence accuracy (epsilon-neighbourhood)
method = 'dsk_powell'    # 'dsk_powell' or 'golden_section'
criterion = 'norm'       # stopping criterion: 'delta' or 'norm'
h = 0.0001               # value of step in derivative formula
schema = 'central'       # derivative schema: 'central', 'right', 'left'

# Svenn's algorithm parameters
init_lambda = 0.12          # initial value of lambda
delta_cf = 0.08         # value of coefficient in delta lambda

# Golden section method
e_GS = 0.01             # golden section convergence accuracy

# DSK-Powell method
e_DSK = 0.01
```

Лістинг *bfgs.py*

```
import numpy as np
import numpy.linalg as ln
from svenn_method import svenn
from golden_section_method import golden_section
from dsk_powell_method import dsk_powell
from settings import max_BFGS, schema, h, criterion, method, e_BFGS


f_iter = 0


# given function
def f(x):
    return (100 * (x[0] ** 2 - x[1]) ** 2) + (x[0] - 1) ** 2


# gradient of the initial function
def f_gradient(x):
    if schema == 'central':
        return np.array([(f([x[0] + h, x[1]]) - f([x[0] - h, x[1]])) / (2 *
h),
                         (f([x[0], x[1] + h]) - f([x[0], x[1] - h])) / (2 *
h)]), 4
    if schema == 'right':
        grad_x_y = f(x)
        return np.array([(f([x[0] + h, x[1]]) - grad_x_y) / h,
                         (f([x[0], x[1] + h]) - grad_x_y) / h]), 3
    if schema == 'left':
        grad_x_y = f(x)
        return np.array([(grad_x_y - f([x[0] - h, x[1]])) / (2 * h),
                         (grad_x_y - f([x[0], x[1] - h])) / (2 * h)]), 3
```

```
def one_dimensional_search(a, b, x_k, pk, name):
    if name == 'dsk_powell':
        return dsk_powell(f, a, b, x_k, pk)
    else:
        return golden_section(f, a, b, x_k, pk, 0)


def stop_criterion(grad_fk, x_k, x_k1):
    global f_iter
    if criterion == 'norm':
        return ln.norm(grad_fk) < e_BFGS
    f_xk = f([x_k[0], x_k[1]])
    f_xk1 = f([x_k1[0], x_k1[1]])
    f_iter += 2
    return ln.norm(x_k1 - x_k) / ln.norm(x_k) < e_BFGS and
ln.norm(np.abs(f_xk1 - f_xk)) < e_BFGS


def BFGS(f, gradient, x0):
    global f_iter
    # define initial method parameters
    # n_iter = 0     # define var of number of iterations
    grad_fk, g_iter = gradient(x0)     # find the gradient in given point
    f_iter += g_iter
    I = np.eye(len(x0), dtype=int)      # set the identity matrix I
    A_k = I     # set the initial value of A_k of metric matrix equals
identity matrix X
    x_k = x0    # set the initial value of x_k equals x_0
    restart_count = 0

    # check the method for the termination criterion
    while True:
        pk = -np.dot(A_k, grad_fk)      # find direction of search
        a, b, f_svenn = svenn(f, x_k, pk)
        f_iter += f_svenn
        lambda_k, f_mop = one_dimensional_search(a, b, x_k, pk, method)
        f_iter += f_mop
        if lambda_k == 0:
            A_k = I
            grad_fk, g_iter = gradient(x0)
            f_iter += g_iter
            restart_count += 1
            continue
        x_k1 = x_k + lambda_k * pk    # find the next point x_k+1 by
recurrent formula

        delta_xk = x_k1 - x_k     # find the value of algorithm step
(delta_x_k)
        if stop_criterion(grad_fk, x_k, x_k1):
            return x_k, restart_count

        x_k = x_k1     # update point value
        grad_fk1, g_iter = gradient(x_k1)    # update gradient of function in
new point x_k+1
        f_iter += g_iter
```

```
        delta_grad_fk = grad_fk1 - grad_fk    # find delta gradient
        grad_fk = grad_fk1
        # n_iter += 1     # update number of iterations

        # correcting matrix has formula: A_k+1 = [I - term1] * A_k * [I -
term2] + term3
        term1 = I - (delta_xk[:, np.newaxis] * delta_grad_fk[np.newaxis, :])
/ np.dot(delta_xk, delta_grad_fk)
        term2 = I - (delta_xk[np.newaxis, :]) * delta_grad_fk[:, np.newaxis]
/ np.dot(delta_xk, delta_grad_fk)
        term3 = (delta_xk[:, np.newaxis] * delta_xk[np.newaxis, :]) /
np.dot(delta_xk, delta_grad_fk)
        A_k = np.dot(term1, np.dot(A_k, term2)) + term3


result, r_count = BFGS(f, f_gradient, np.array([-1.2, 0]))
print('Point: ', result)
print('f(x) = ', f(result))
print('r_count: ', r_count)
print('f_iter: ', f_iter)
```

Лістинг *svenn_method.py*

```
from numpy.linalg import norm
from settings import init_lambda, delta_cf


def svenn(f, x, pk):
    f_iter = 0
    lambdas = [init_lambda]
    f_lambdas = [f(x)]
    delta_lambda = delta_cf * norm(x) / norm(pk)

    f_plus_delta = f(x + (lambdas[0] + delta_lambda) * pk)
    f_minus_delta = f(x + (lambdas[0] - delta_lambda) * pk)

    if f_plus_delta > f_minus_delta:
        f_lambdas.append(f_minus_delta)
        delta_lambda = -1 * delta_lambda
    else:
        f_lambdas.append(f_plus_delta)

    lambdas.append(lambdas[0] + delta_lambda)
    f_iter += 3

    while True:
        f_iter += 1
        delta_lambda *= 2
        lambdas.append(lambdas[-1] + delta_lambda)
        f_lambdas.append(f(x + lambdas[-1] * pk))
        if f_lambdas[-1] > f_lambdas[-2]:
            lambdas.append((lambdas[-1] + lambdas[-2]) / 2)
            f_lambdas.append(f(x + lambdas[-1] * pk))
            return lambdas[-4], lambdas[-1], f_iter
```

Лістинг *golden_section_method.py*

```python
from settings import e_GS


def golden_section(f, a, b, x, pk, f_iter):
    if b - a <= e_GS:
        return (a + b) / 2, f_iter
    x1 = a + 0.382 * (b - a)
    x2 = a + 0.618 * (b - a)
    f_x1 = f([x[0] + x1 * pk[0], x[1] + x1 * pk[1]])
    f_x2 = f([x[0] + x2 * pk[0], x[1] + x2 * pk[1]])
    if f_x1 < f_x2:
        return golden_section(f, a, x2, x, pk, f_iter + 2)
    else:
        return golden_section(f, x1, b, x, pk, f_iter + 2)
```

Лістинг *dsk_powell_method.py*

```python
import numpy as np
from settings import e_DSK


def dsk_powell_iteration(interval):
    values = np.array(list(interval.values()))
    center = list(interval.keys())[np.where(values[:, 1] == min(values[:,
1]))[0][0]]
    s_interval = sorted(interval.items(), key=lambda x: x[1][0])
    s_interval = {s_interval[i][0]: s_interval[i][1] for i in
range(len(s_interval))}

    center_index = list(s_interval.keys()).index(center)
    if center_index == len(s_interval) - 1:
        return s_interval[center][0]
    x1, x3 = list(s_interval.keys())[center_index - 1],
list(s_interval.keys())[center_index + 1]
    new_interval = {'x1': s_interval[x1], 'x2': s_interval[center], 'x3':
s_interval[x3]}
    return new_interval


def dsk_powell(f, x1, x3, x0, pk):
    f_iter = 0
    x2 = (x1 + x3) / 2
    delta_x = x2 - x1
    f_x1, f_x2, f_x3 = [f([x0[0] + i * pk[0], x0[1] + i * pk[1]]) for i in
[x1, x2, x3]]
    x = x2 + (delta_x * (f_x1 - f_x3)) / (2 * (f_x1 - 2 * f_x2 + f_x3))
    f_x = f([x0[0] + x * pk[0], x0[1] + x * pk[1]])
    f_iter += 4

    if np.abs(f_x2 - f_x) <= e_DSK and np.abs(x2 - x) <= e_DSK:
        return x, f_iter
```

```
    while True:
        interval = dsk_powell_iteration({'x1': (x1, f_x1), 'x2': (x2, f_x2),
'x3': (x3, f_x3), 'x*': (x, f_x)})
        if not isinstance(interval, dict):
            return interval, f_iter
        a1 = (interval['x2'][1] - interval['x1'][1]) / (interval['x2'][0] -
interval['x1'][0])
        a2 = (((interval['x3'][1] - interval['x1'][1]) / (interval['x3'][0] -
interval['x1'][0])) -
              ((interval['x2'][1] - interval['x1'][1]) / (interval['x2'][0] -
interval['x1'][0]))) / (interval['x3'][0] - interval['x2'][0])
        x = (interval['x1'][0] + interval['x2'][0]) / 2 - a1 / (2 * a2)
        f_x = f([x0[0] + x * pk[0], x0[1] + x * pk[1]])
        f_iter += 1
        if np.abs(f_x2 - f_x) <= e_DSK and np.abs(x2 - x) <= e_DSK:
            return x, f_iter
        x1, x2, x3 = [interval[i][0] for i in ['x1', 'x2', 'x3']]
        f_x1, f_x2, f_x3 = [interval[i][1] for i in ['x1', 'x2', 'x3']]
```