

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра системного проєктування**

**Звіт
про виконання лабораторної роботи №2
з дисципліни «методи оптимізації»
“Дослідження методів багатовимірного пошуку на основі прямого
пошуку та з використанням градієнту”**

Виконав:
студент 2 курсу, групи КН-32
Сафонов Дмитро
Володимирович

Київ – 2025

Варіант 20

Мета роботи: Ознайомитись і дослідити методи пошуку мінімуму цільової функції двох змінних. Набути навичок чисельного оцінювання градієнта функції та його використання для дослідження функцій, реалізувати метод найшвидшого градієнтного спуску та метод прямого пошуку, навчитися працювати з готовими бібліотечними / пакетними рішеннями для візуалізації функцій у різному форматі.

Завдання:

N	$G(x_1, x_2)$
10	$G(x_1, x_2) = 0.5 \sum_{i=1}^n x_i^4 - 16x_i^2 + 5x_i; n = 2.$

Хід роботи:

1. Отримати індивідуальну функцію $G(x1,x2)$ відповідно до свого варіанту з розділу 5 (для варіантів >10 використати N%10).

- Побудувати графік функції у вигляді 3D поверхні та у вигляді ліній рівня (ізоліній) в приблизних межах $(-4,4)$ дляожної змінної. За необхідності змінити інтервал відповідно до функції по варіанту.
- Для візуалізації використати бібліотечне / пакетне рішення, що має готові інструменти для дослідження функцій та для 3D рендерингу функцій (python-matplotlib / Matlab / Mathematica / GNU Octave).

Для візуалізації функції використаємо бібліотеку python-matplotlib:

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

def G(x1, x2):
    return 0.5 * ((np.pow(x1, 4) - 16 * np.pow(x1, 2) + 5 * x1) + (np.pow(x2, 4) - 16 * np.pow(x2, 2) + 5 * x2))

x1_vals = np.linspace(-4, 4, 100)
x2_vals = np.linspace(-4, 4, 100)
X1, X2 = np.meshgrid(x1_vals, x2_vals)
```

```

Z = G(X1, X2)

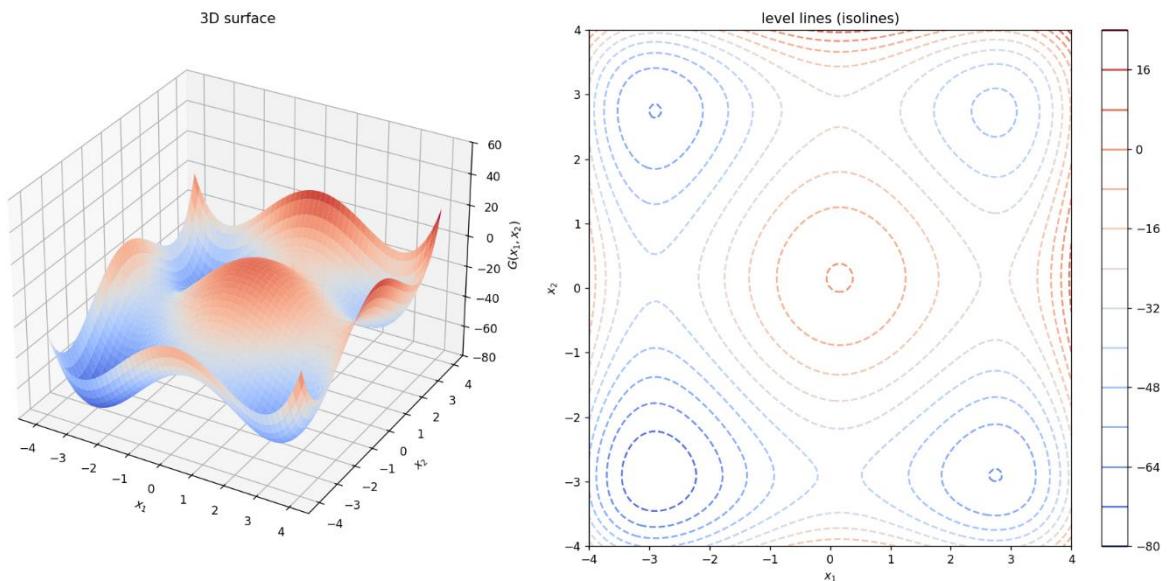
fig = plt.figure(figsize=(12, 5))

# 3Д графік
ax1 = fig.add_subplot(1, 2, 1, projection='3d')
ax1.plot_surface(X1, X2, Z, cmap=cm.coolwarm)
ax1.set_zlim(-80, 60)
ax1.set_xlabel('$x_1$')
ax1.set_ylabel('$x_2$')
ax1.set_zlabel('$G(x_1, x_2)$')
ax1.set_title('3D surface')

# контурний графік
ax2 = fig.add_subplot(1, 2, 2)
contour = ax2.contour(X1, X2, Z, cmap=cm.coolwarm, levels=12,
linestyles='dashed')
plt.colorbar(contour, ax=ax2)
ax2.set_xlabel('$x_1$')
ax2.set_ylabel('$x_2$')
ax2.set_title('level lines (isolines)')

plt.tight_layout()
plt.show()

```



2. Обчислити складові градієнта функції.

- 2.1. Аналітично обчислити складові градієнта функції (часткові похідні).
- 2.2. Реалізувати процедуру обчислення числового градієнта за формулою центральних різниць.
- 2.3. Довільно обрати 3-4 точки в області дослідження, визначити в них градієнт аналітично та за формулою центральних різниць з кроками 0.001, 0.01, 0.1, 1. Порівняти результати та визначити оптимальний крок

Обчислимо часткові похідні та їх значення у 4-х точках:

$$G(x_1, x_2) = 0,5 \cdot ((x_1^4 - 16x_1^2 + 5x_1) + (x_2^4 - 16x_2^2 + 5x_2))$$

Обчислимо часткові похідні:

$$\frac{dG}{dx_1} = 0,5 \left(\frac{\partial L}{\partial x_1} (x_1^4 - 16x_1^2 + 5x_1) + 0 \right) =$$

$$= 0,5(4x_1^3 - 32x_1 + 5) = 2x_1^3 - 16x_1 + 2,5$$

$$\frac{dG}{dx_2} = 0,5 \left(\frac{\partial L}{\partial x_2} (x_2^4 - 16x_2^2 + 5x_2) + 0 \right) = 2x_2^3 - 16x_2 + 2,5$$

Обчислимо похідні в точці $(-3, 1)$:

$$\frac{dG}{dx_1} = -54 + 48 + 2,5 = -6 + 2,5 = -3,5$$

$$\frac{dG}{dx_2} = 2. -16 + 2,5 = -11,5$$

В точці $(4; -2)$:

$$\frac{dG}{dx_1} = 128 - 64 + 2,5 = 66,5$$

$$\frac{dG}{dx_2} = -16 + 32 + 2,5 = 18,5$$

В точці $(0, 0)$:

$$\frac{dG}{dx_1} = 2,5$$

$$\frac{dG}{dx_2} = 2,5$$

В точці $(2, 0)$:

$$\frac{dG}{dx_1} = 16 - 32 + 2,5 = -13,5$$

$$\frac{dG}{dx_2} = 2,5$$

Мовою Python реалізуємо процедуру для обчислення складових градієнта за допомогою формули середніх різниць:

```

import numpy as np

def G(x1, x2):
    return 0.5 * ((np.pow(x1, 4) - 16 * np.pow(x1, 2) + 5 * x1) + (np.pow(x2, 4) - 16 * np.pow(x2, 2) + 5 * x2))

def central_differences(x1, x2, delta):
    derivative_x1 = (G(x1 + delta, x2) - G(x1 - delta, x2)) / (2 * delta)
    derivative_x2 = (G(x1, x2 + delta) - G(x1, x2 - delta)) / (2 * delta)
    return derivative_x1, derivative_x2

points = [(-3, 1), (4, -2), (0, 0), (2, 0)]
deltas = [1, 0.1, 0.01, 0.0001]

for delta in deltas:
    print(f"\ndelta = {delta}")
    for x1, x2 in points:
        der_x1, der_x2 = central_differences(x1, x2, delta)
        print(f"Похідна G за x1 в точці ({x1}, {x2}): {der_x1}")
        print(f"Похідна G за x2 в точці ({x1}, {x2}): {der_x2}")

```

Порівняння результатів, обчислених аналітично за допомогою часткових похідних, та результатів, обчислених програмно, наведено у таблиці 1:

Точка	Похідні $\left(\frac{\partial G}{\partial x_1}, \frac{\partial G}{\partial x_2}\right)$					
	Аналітично	$\Delta x = 1$	$\Delta x = 0.1$	$\Delta x = 0.01$	$\Delta x = 0.001$	$\Delta x = 0.0001$
(-3,1)	$\frac{\partial G}{\partial x_1} = -3.5$	-9.5	-3.55999999	-3.50059999	-3.50000600	-3.50000006
	$\frac{\partial G}{\partial x_2} = -11.5$	-9.5	-11.47999999	-11.49979999	-11.49997999	-11.49999998
(4,-2)	$\frac{\partial G}{\partial x_1} = 66.5$	74.5	66.57999999	66.50079999	66.50000800	66.50000008
	$\frac{\partial G}{\partial x_2} = 18.5$	14.5	18.46	18.49959999	18.49999599	18.49999996
(0,0)	$\frac{\partial G}{\partial x_1} = 2.5$	2.5	2.5	2.5	2.5	2.5
	$\frac{\partial G}{\partial x_2} = 2.5$	2.5	2.5	2.5	2.5	2.5
(2,0)	$\frac{\partial G}{\partial x_1} = -13.5$	-9.5	-13.46	-13.49959999	-13.49999599	-13.49999996
	$\frac{\partial G}{\partial x_2} = 2.5$	2.5	2.5	2.50000000	2.50000000	2.50000000

З результатів видно, що при великому значенні Δx похибка є найбільшою. При зменшенні значення Δx чисельні значення сильніше наближаються до аналітичних. При значенні $\Delta x = 0.01$ чисельно обчислене значення вже дуже точне, а при зменшенні Δx точність тільки збільшується.

3. Обчислити мінімум функції декількома методами.

- 3.1. Реалізувати метод найшвидшого спуску (метод Коші). Для обчислення градієнту використати процедуру чисельного диференціювання на основі формули центральних різниць.
- 3.2. Реалізувати один з методів прямого пошуку: симплекс метод, метод Хука-Дживса, метод спряжених напрямків Пауела.
- 3.3. Провести тестування реалізованих методів, використавши різні (мінімум по 3) значення початкових точок, точності пошуку та коефіцієнту навчання.
- 3.4. Візуалізувати шлях збіжності кожного методу поверх графіку ізоліній функції.
- 3.5. Заміряти час роботи, кількість ітерацій та КОЦФ кожного методу. Занести отримані результати у зведену таблицю. Порівняти метод найшвидшого градієнтного спуску з методами прямого пошуку.

Реалізація методу найшвидшого спуску (методу Коші):

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import time

function_calls = 0

def G(x1, x2):
    global function_calls
    function_calls += 1
    return 0.5 * ((np.pow(x1, 4) - 16 * np.pow(x1, 2) + 5 * x1) + (np.pow(x2,
4) - 16 * np.pow(x2, 2) + 5 * x2))

def central_differences(x1, x2, delta):
    derivative_x1 = (G(x1 + delta, x2) - G(x1 - delta, x2)) / (2 * delta)
    derivative_x2 = (G(x1, x2 + delta) - G(x1, x2 - delta)) / (2 * delta)
    return derivative_x1, derivative_x2

def gradient_descent(x1, x2, alpha, delta, accuracy, max_iter):
    path = [(x1, x2)]

    for i in range(max_iter):
        grad_x1, grad_x2 = central_differences(x1, x2, delta)

        x1_new = x1 - alpha * grad_x1
        x2_new = x2 - alpha * grad_x2

        path.append((x1_new, x2_new))
```

```

if np.sqrt((x1_new - x1) ** 2 + (x2_new - x2) ** 2) < accuracy:
    print(f'Метод зійшовся за {i + 1} ітерацій')
    break

x1, x2 = x1_new, x2_new

return x1, x2, G(x1, x2), path

```

Початкові значення

```

x1 = 0
x2 = 0
delta = 0.0001
alpha = 0.001
accuracy = 0.001
max_iter = 1000

start = time.perf_counter_ns()
x1_fin, x2_fin, G_fin, path = gradient_descent(x1, x2, alpha, delta,
accuracy, max_iter)
end = time.perf_counter_ns()
elapsed = end - start

print(f'Точка мінімуму: ({x1_fin},{x2_fin})\nЗначення функції: {G_fin}')
print(f'КОЦФ: {function_calls}')
print(f"Час виконання: {elapsed / 1_000_000_000:.6f} секунд")

```

Тепер протестуємо роботу метода. Оберемо різні початкові точки, різну величину коефіцієнта alpha та задамо різну точність обчислень.

Початкова точка (0, 0), alpha = 0.001, accuracy = 0.001:

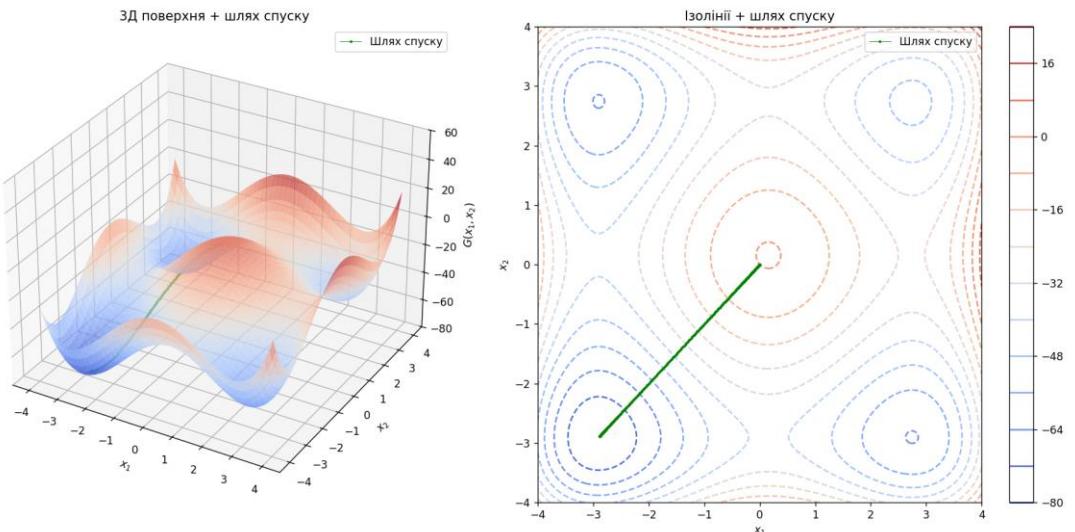
Метод зійшовся за 306 ітерацій

Точка мінімуму: (-2.882959879891338,-2.882959879891338)

Значення функції: -78.31779351983846

КОЦФ: 1225

Час виконання: 0.007220 секунд



Початкова точка (2, 0), alpha = 0.001, accuracy = 0.001:

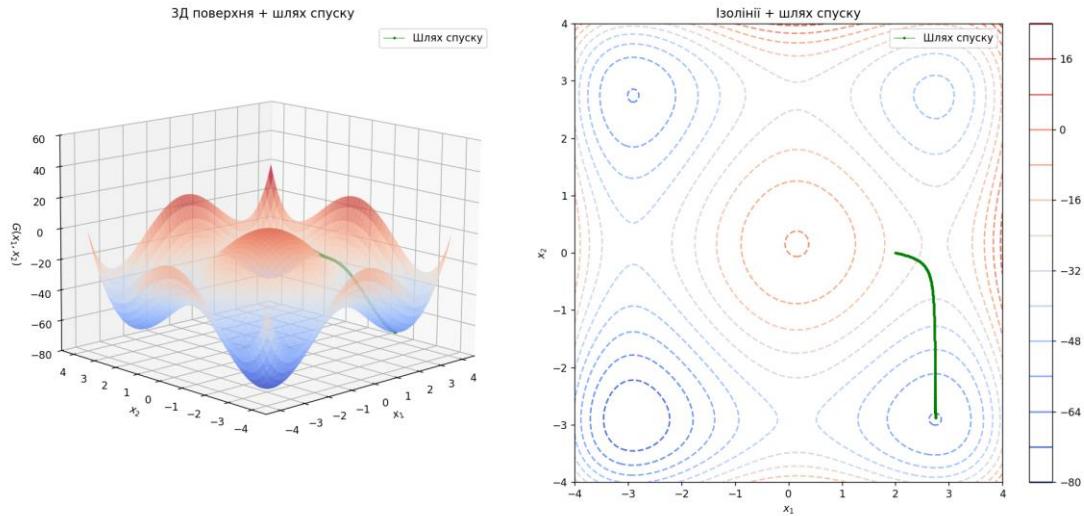
Метод зійшовся за 296 ітерацій

Точка мінімуму: (2.7466058165096356, -2.8744123138710975)

Значення функції: -64.18109035067505

КОЦФ: 1185

Час виконання: 0.006751 секунд



Початкова точка (0.156731, 0.156731), alpha = 0.001, accuracy = 0.001:

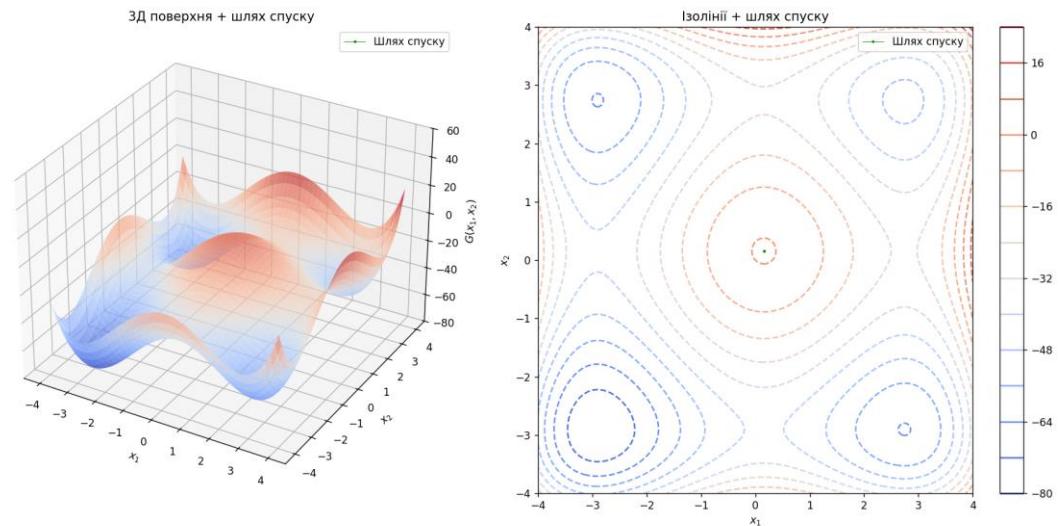
Метод зійшовся за 1 ітерацій

Точка мінімуму: (0.156731, 0.156731)

Значення функції: 0.3912247181096708

КОЦФ: 5

Час виконання: 0.000097 секунд



Початкова точка (0.156731, 0.156731), alpha = 0.01, accuracy = 0.00000001:

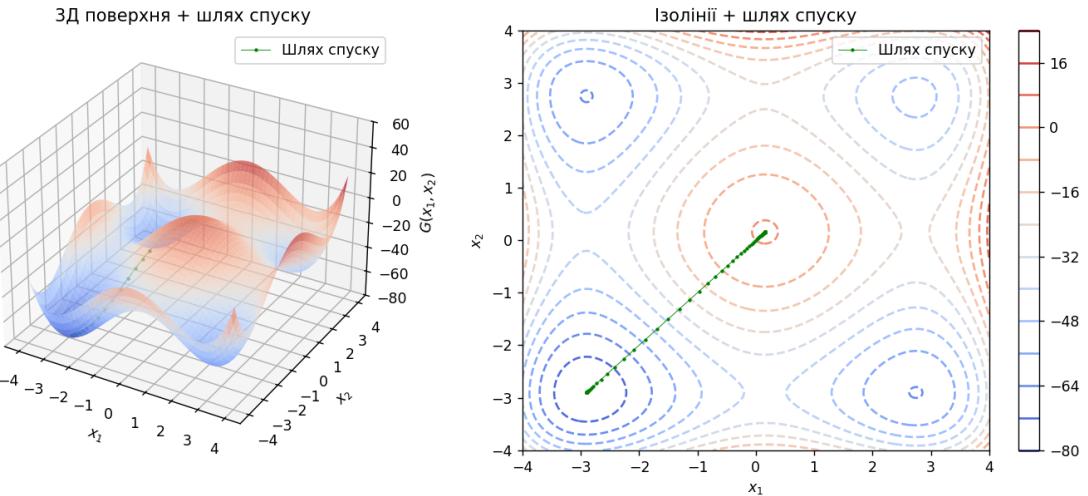
Метод зійшовся за 154 ітерацій

Точка мінімуму: (-2.903534006069587, -2.903534006069587)

Значення функції: -78.33233140754282

КОЦФ: 617

Час виконання: 0.003400 секунд



Початкова точка (0.156731, 0), alpha = 0.001, accuracy = 0.001:

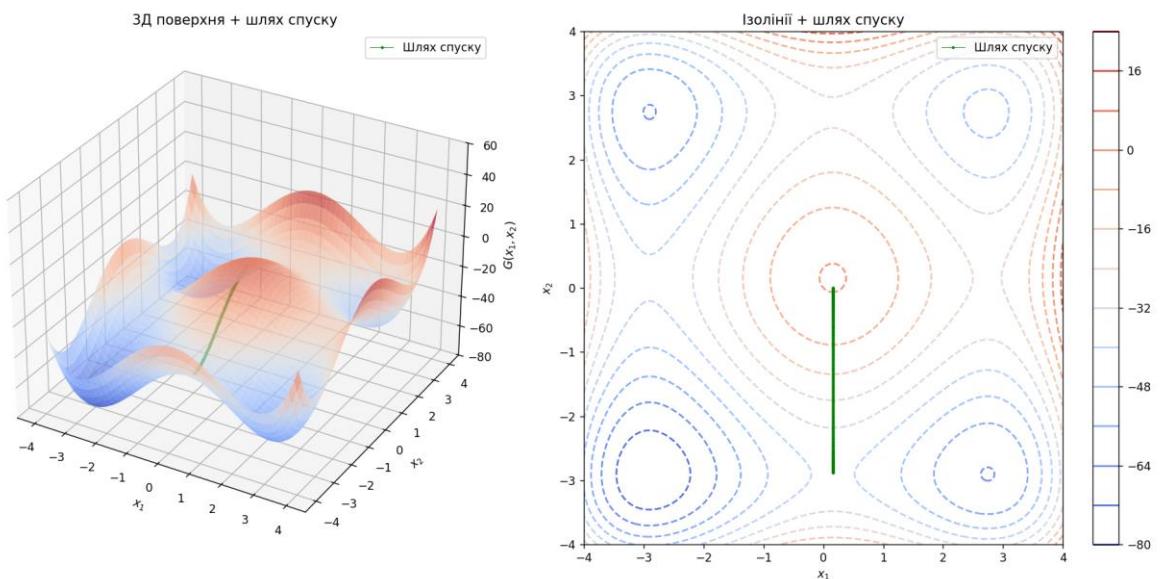
Метод зійшовся за 296 ітерацій

Точка мінімуму: (0.1567046525636128, -2.874412313872008)

Значення функції: -38.95603190960342

КОЦФ: 1185

Час виконання: 0.006457 секунд



Тепер збільшимо коефіцієнт alpha та запустимо алгоритм:

Початкова точка $(0, 0)$, $\text{alpha} = 0.01$, $\text{accuracy} = 0.001$:

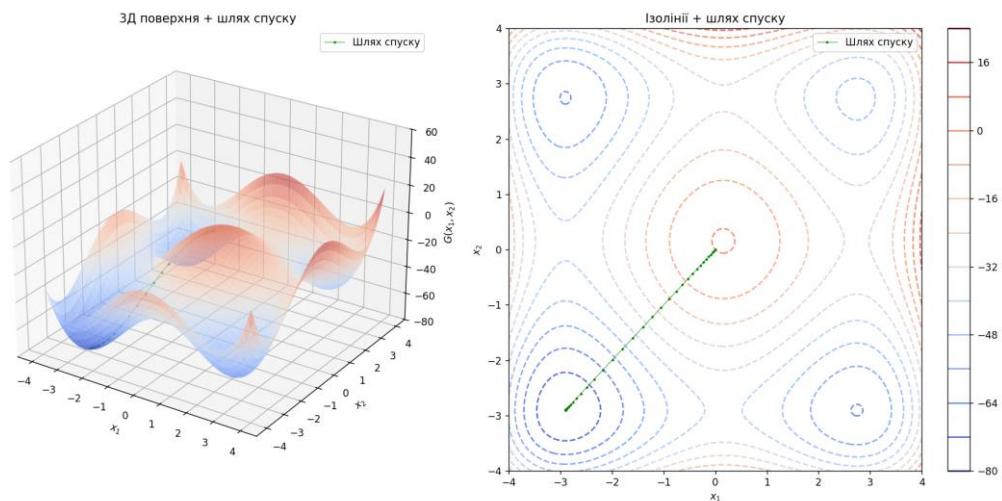
Метод зійшовся за 37 ітерацій

Точка мінімуму: $(-2.901990440299071, -2.901990440299071)$

Значення функції: -78.33224905050157

КОЦФ: 149

Час виконання: 0.000982 секунд



Початкова точка $(2, 0)$, $\text{alpha} = 0.01$, $\text{accuracy} = 0.001$:

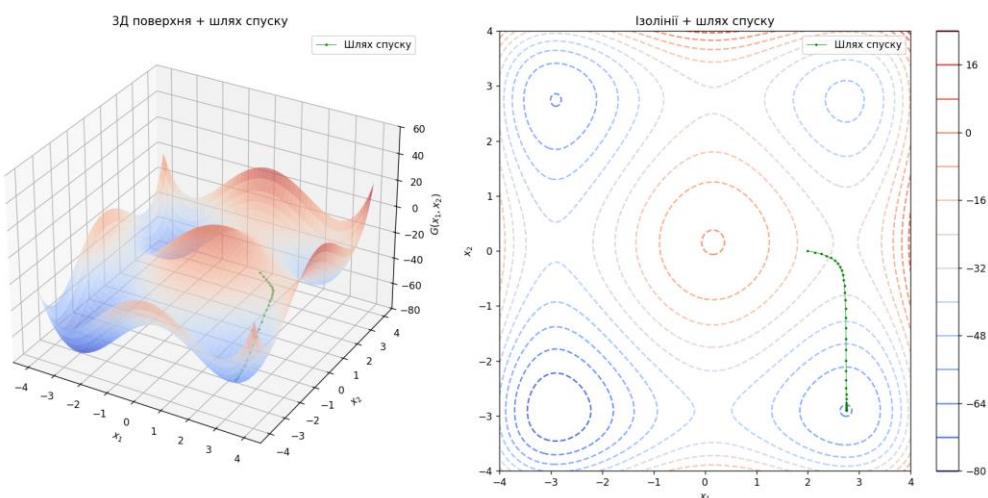
Метод зійшовся за 36 ітерацій

Точка мінімуму: $(2.7467945474434856, -2.901175894882435)$

Значення функції: -64.19551627962169

КОЦФ: 145

Час виконання: 0.000862 секунд



Початкова точка $(2, 0)$, alpha = 0.04, accuracy = 0.001:

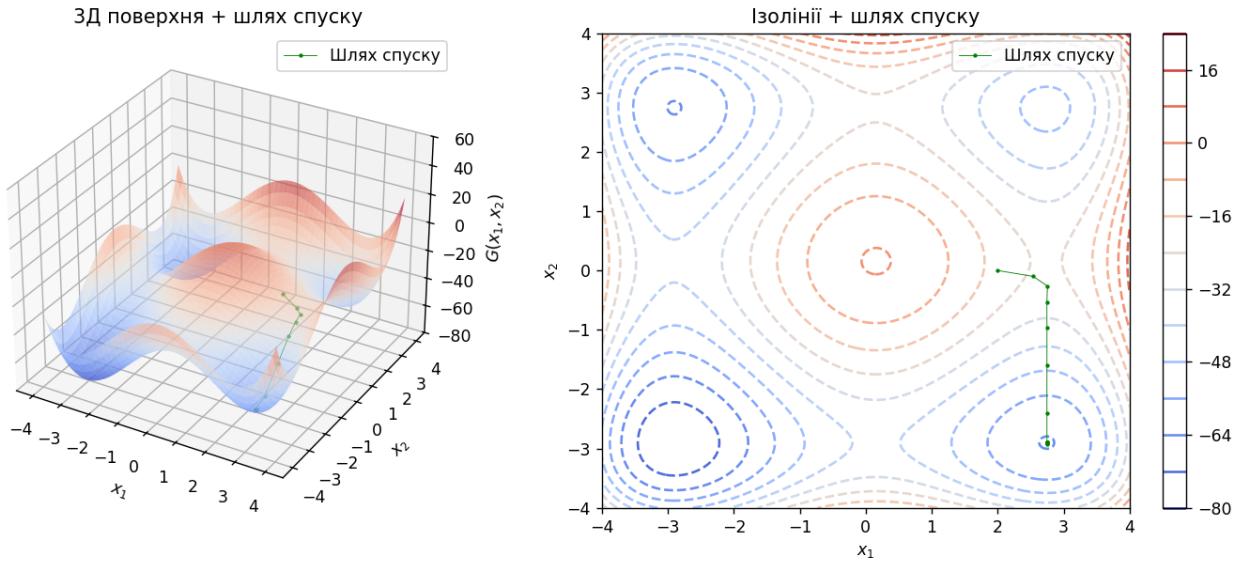
Метод зійшовся за 22 ітерацій

Точка мінімуму: $(2.7468027691153623, -2.9035340663391196)$

Значення функції: -64.19561235905533

КОЦФ: 89

Час виконання: 0.000617 секунд



Початкова точка $(0.156731, 0)$, alpha = 0.01, accuracy = 0.001:

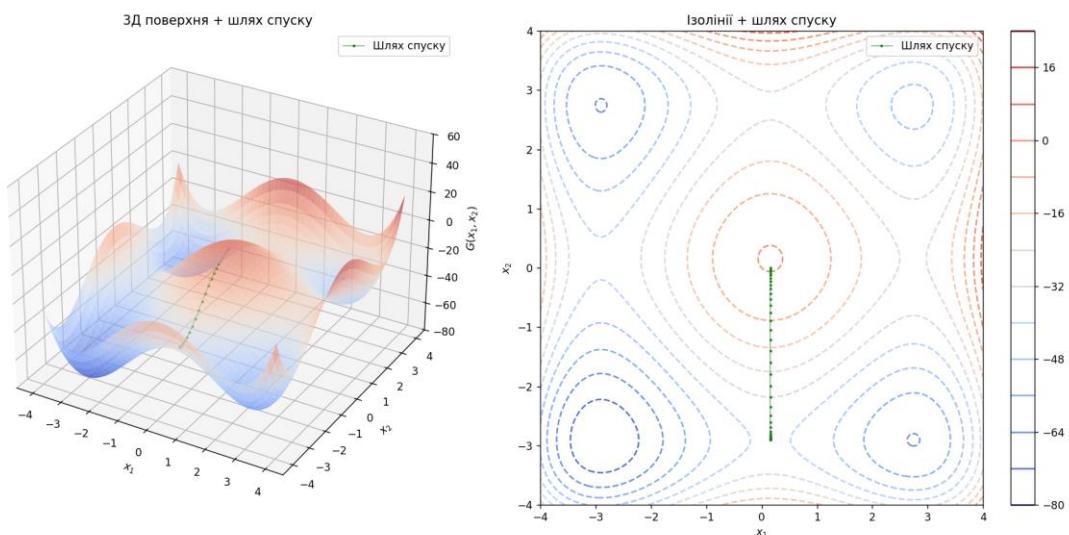
Метод зійшовся за 36 ітерацій

Точка мінімуму: $(0.15668693684214702, -2.9011758948829067)$

Значення функції: -38.97045728184145

КОЦФ: 145

Час виконання: 0.000867 секунд



Початкова точка (0, 0), alpha = 0.05, accuracy = 0.001:

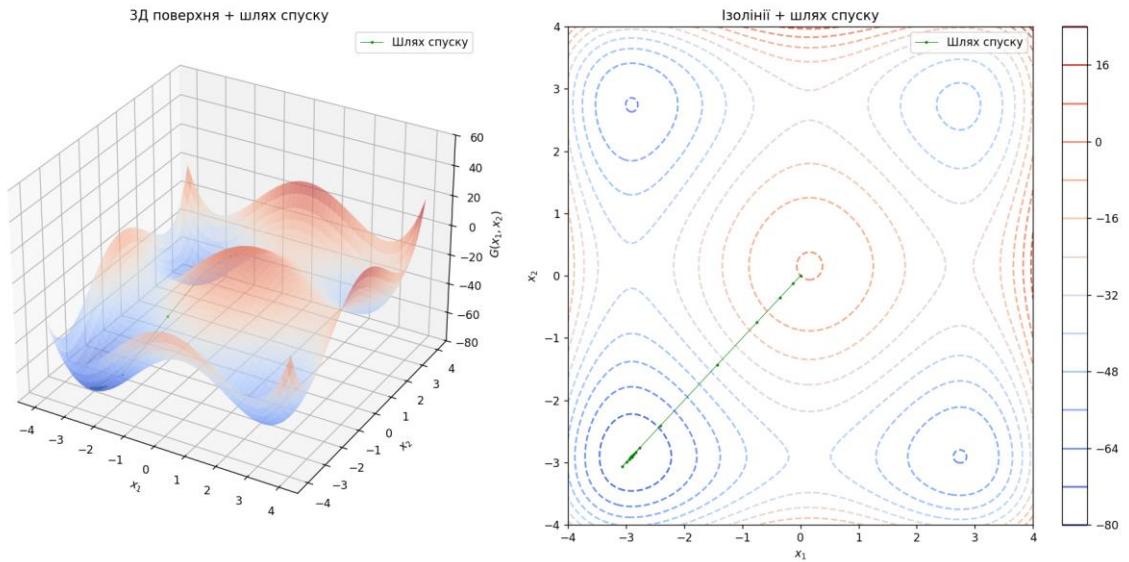
Метод зійшовся за 27 ітерацій

Точка мінімуму: (-2.903833497799671, -2.903833497799671)

Значення функції: -78.33232830574269

КОЦФ: 109

Час виконання: 0.000696 секунд



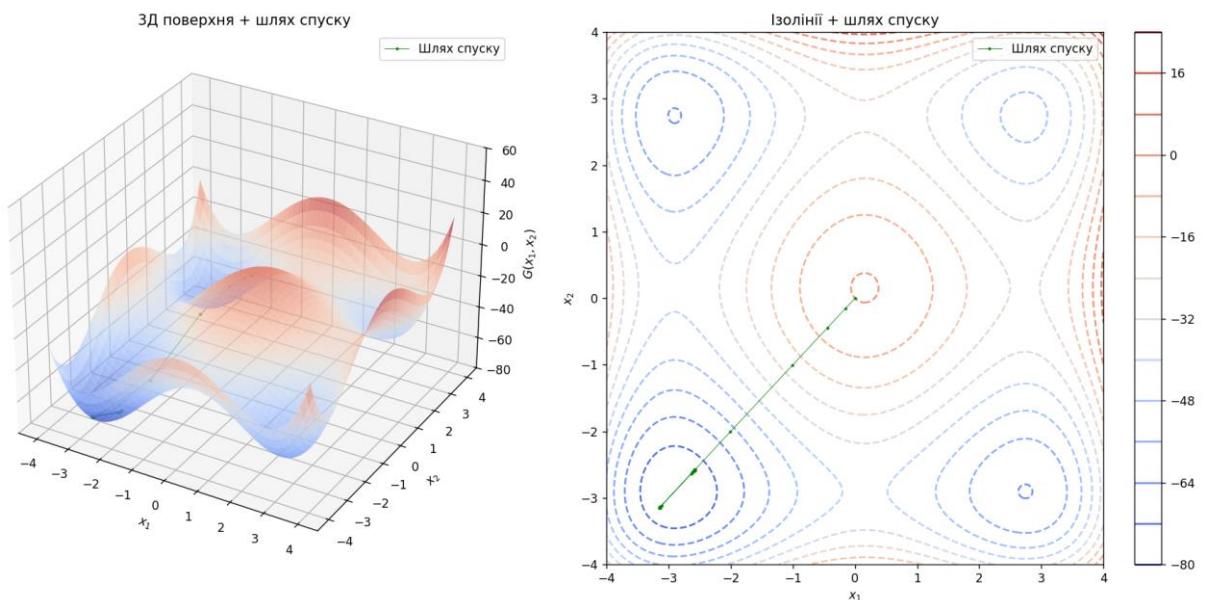
Початкова точка (0, 0), alpha = 0.06, accuracy = 0.001:

Точка мінімуму: (-2.57920309787511, -2.57920309787511)

Значення функції: -75.07968952583276

КОЦФ: 4001

Час виконання: 0.021978 секунд



Початкова точка (0.156731, 0), alpha = 0.05, accuracy = 0.001:

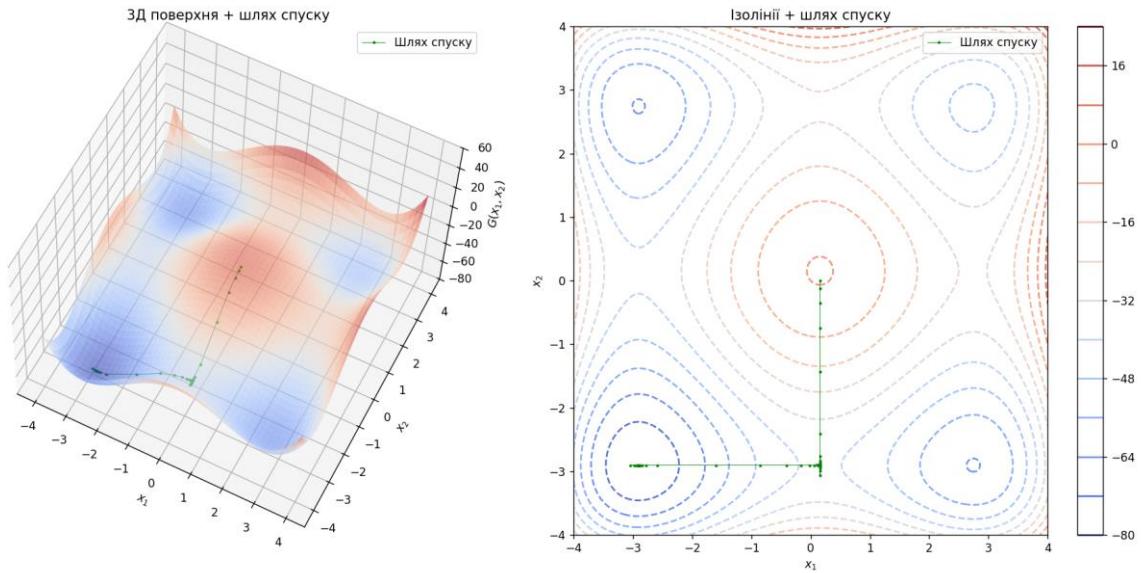
Метод зійшовся за 48 ітерацій

Точка мінімуму: (-2.9040484110389526, -2.9035336319650358)

Значення функції: -78.33232683158141

КОЦФ: 193

Час виконання: 0.001139 секунд



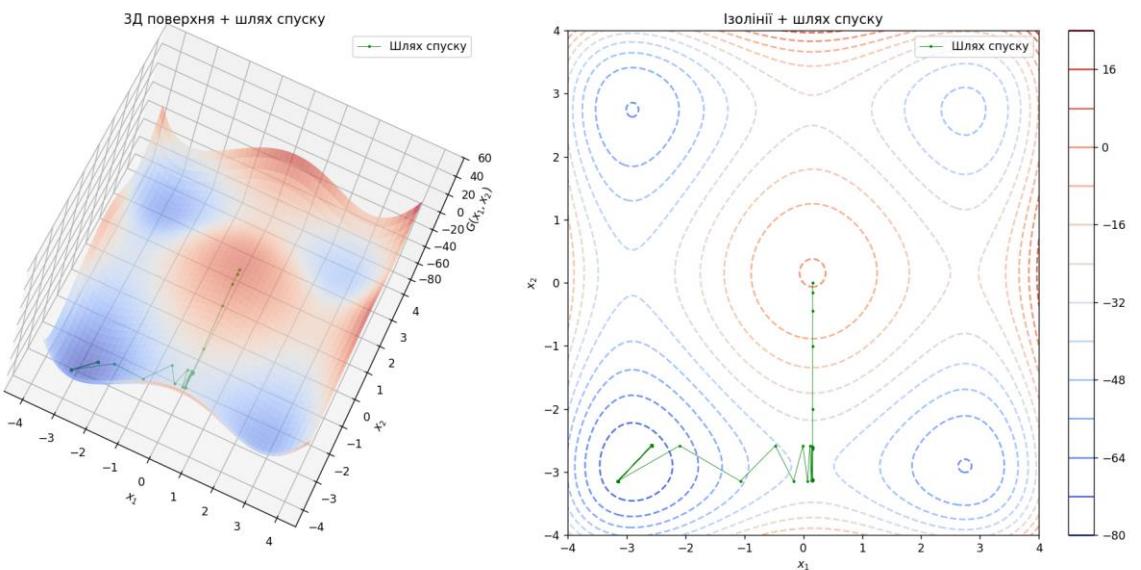
Початкова точка (0.156731, 0), alpha = 0.06, accuracy = 0.001:

Точка мінімуму: (-2.5792030978471203, -2.5792030978761424)

Значення функції: -75.07968952557795

КОЦФ: 4001

Час виконання: 0.021004 секунд



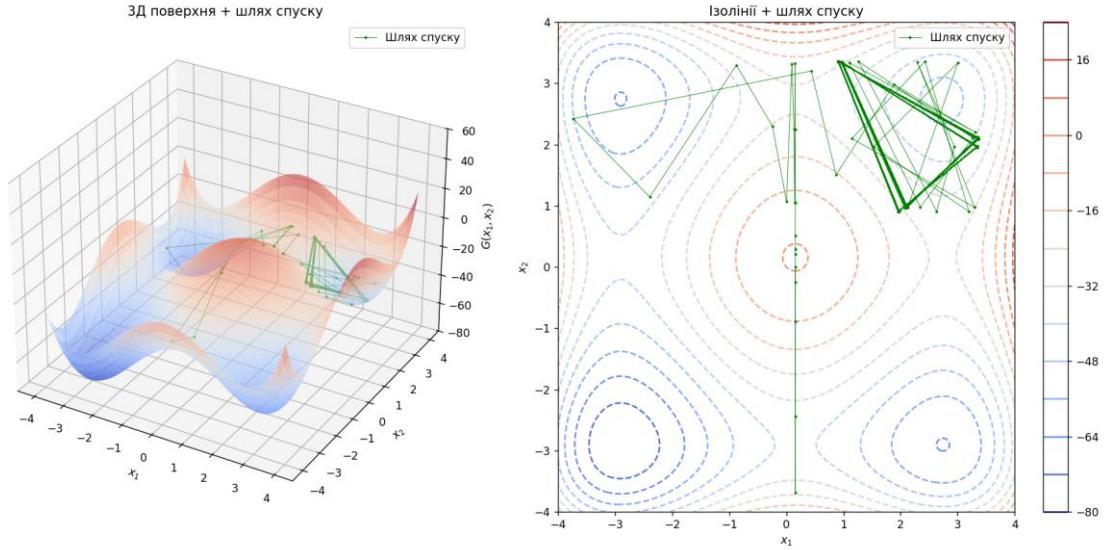
Початкова точка $(0.156731, 0)$, alpha = 0.1, accuracy = 0.001:

Точка мінімуму: $(3.3433830541931897, 1.9716984784737548)$

Значення функції: -37.20584808674199

КОЦФ: 4001

Час виконання: 0.022005 секунд



Початкова точка $(0.156731, 0)$, alpha = 0.35, accuracy = 0.001:

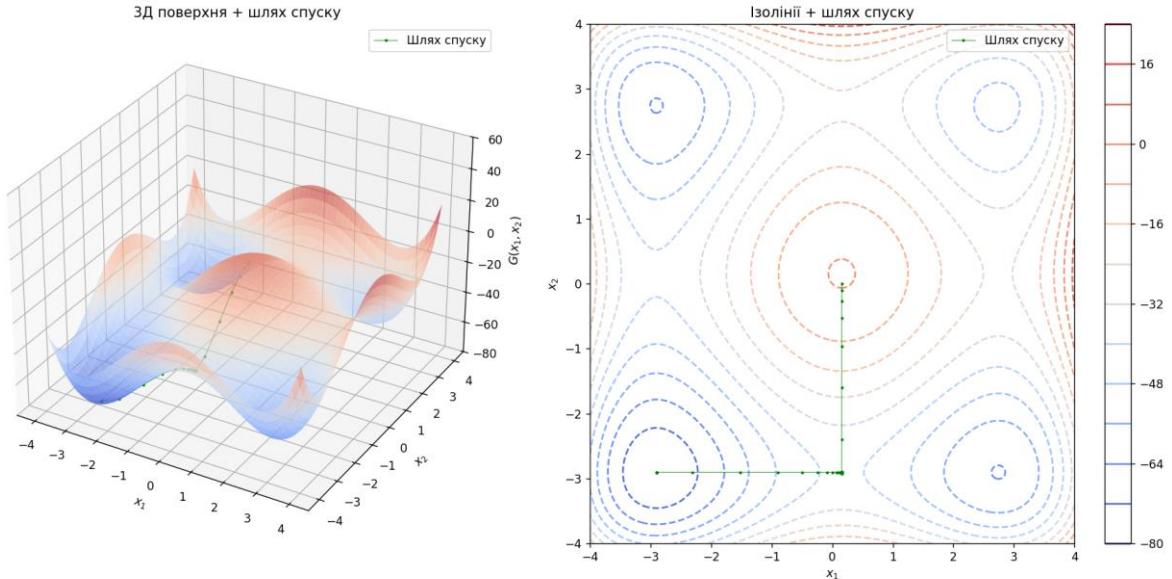
Метод зійшовся за 41 ітерацій

Точка мінімуму: $(-2.9035291379195183, -2.9035340260917586)$

Значення функції: -78.33233140712937

КОЦФ: 165

Час виконання: 0.000971 секунд



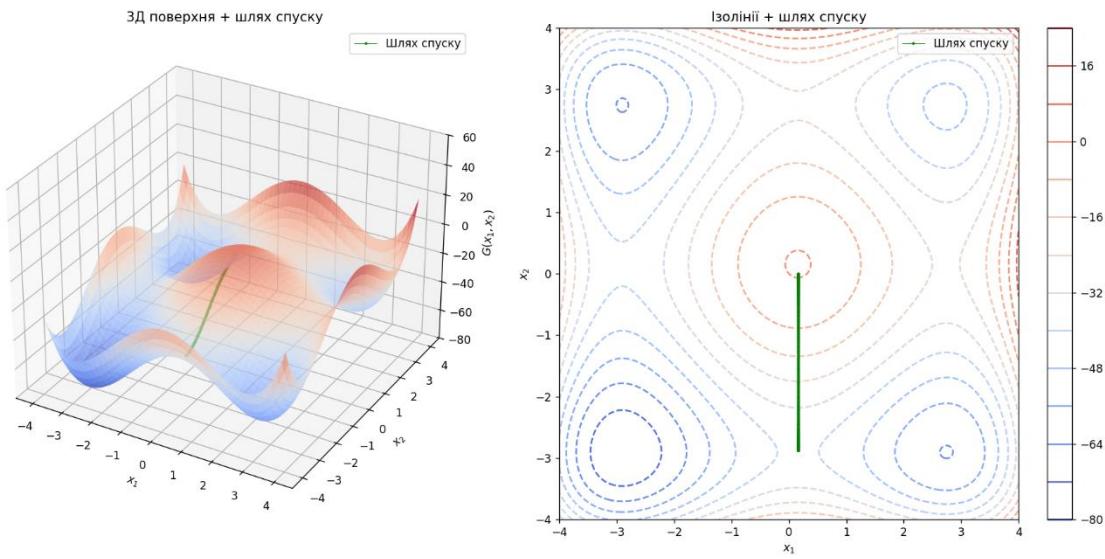
Аналіз результатів:

Точка	Alpha	Результати
(0, 0)	0.001	Метод зійшовся за 306 ітерацій Точка мінімуму: (-2.882959879891338,-2.882959879891338) Значення функції: -78.31779351983846 КОЦФ: 1225 Час виконання: 0.007220 секунд
(2, 0)	0.001	Метод зійшовся за 296 ітерацій Точка мінімуму: (2.7466058165096356,-2.8744123138710975) Значення функції: -64.18109035067505 КОЦФ: 1185 Час виконання: 0.006751 секунд
(0.156731,0)	0.001	Метод зійшовся за 36 ітерацій Точка мінімуму: (0.15668693684214702,-2.9011758948829067) Значення функції: -38.97045728184145 КОЦФ: 145 Час виконання: 0.000867 секунд
(0, 0)	0.01	Метод зійшовся за 37 ітерацій Точка мінімуму: (-2.901990440299071,-2.901990440299071) Значення функції: -78.33224905050157 КОЦФ: 149 Час виконання: 0.000982 секунд
(2, 0)	0.01	Метод зійшовся за 36 ітерацій Точка мінімуму: (2.7467945474434856,-2.901175894882435) Значення функції: -64.19551627962169 КОЦФ: 145 Час виконання: 0.000862 секунд
(0.156731,0)	0.01	Метод зійшовся за 36 ітерацій Точка мінімуму: (0.15668693684214702,-2.9011758948829067) Значення функції: -38.97045728184145 КОЦФ: 145 Час виконання: 0.000867 секунд
(0, 0)	0.05	Метод зійшовся за 27 ітерацій Точка мінімуму: (-2.903833497799671,-2.903833497799671) Значення функції: -78.33232830574269 КОЦФ: 109 Час виконання: 0.000696 секунд

(2, 0)	0.05	Метод зійшовся за 25 ітерацій Точка мінімуму: (2.74680277046113,-2.904097196268829) Значення функції: -64.19560687385808 КОЦФ: 101 Час виконання: 0.000656 секунд
(0.156731,0)	0.05	Метод зійшовся за 48 ітерацій Точка мінімуму: (-2.9040484110389526,-2.9035336319650358) Значення функції: -78.33232683158141 КОЦФ: 193 Час виконання: 0.001139 секунд
(0, 0)	0.04	Метод зійшовся за 15 ітерацій Точка мінімуму: (-2.9035009319289817,-2.9035009319289817) Значення функції: -78.33233136966322 КОЦФ: 61 Час виконання: 0.000418 секунд
(2, 0)	0.04	Метод зійшовся за 22 ітерацій Точка мінімуму: (2.7468027691153623,-2.9035340663391196) Значення функції: -64.19561235905533 КОЦФ: 89 Час виконання: 0.000617 секунд
(0.156731,0)	0.04	Метод зійшовся за 41 ітерацій Точка мінімуму: (-2.9035291379195183,-2.9035340260917586) Значення функції: -78.33233140712937 КОЦФ: 165 Час виконання: 0.000971 секунд

Проаналізувавши результати, можемо зробити висновок, що коефіцієнт alpha у формулі метода найшвидшого спуску найбільше впливає на результати обчислень.

Якщо обрати замалий крок, то час обчислень буде в декілька разів перевищувати час роботи цього ж методу для оптимального кроку. Також є небезпека того, що мінумум буде визначено неправильно, як наприклад у цьому випадку:



Це відбулося тому, що через дуже маленький крок зміни в градієнті стали меншими за вказану точність, і спрацювала умова завершення алгоритму.

Також через замале значення коефіцієнту alpha КОЦФ значно зростає, наприклад подивимося на результати роботи алгоритму з однієї і тієї самої точки, але з різним кроком:

Точка	(0,0)	
Alpha	0.001	0.01
Результати	Метод зійшовся за 306 ітерацій Точка мінімуму: (-2.882959879891338, -2.882959879891338) Значення функції: -78.31779351983846 КОЦФ: 1225 Час виконання: 0.007220 секунд	Метод зійшовся за 37 ітерацій Точка мінімуму: (-2.901990440299071, -2.901990440299071) Значення функції: -78.33224905050157 КОЦФ: 149 Час виконання: 0.000982 секунд

Оптимальним значенням кроку для заданої функції є значення 0.04:

Точка	(2,0)	
Alpha	0.05	0.04
Результати	Метод зійшовся за 25 ітерацій Точка мінімуму: (2.74680277046113, -2.904097196268829) Значення функції: -64.19560687385808 КОЦФ: 101 Час виконання: 0.000656 секунд	Метод зійшовся за 22 ітерацій Точка мінімуму: (2.746802769115323, -2.9035340663391196) Значення функції: -64.19561235905533 КОЦФ: 89 Час виконання: 0.000617 секунд

Але якщо ще трохи збільшити крок до 0.6, то метод почне розбігатися.

Тому для методу найшвидшого спуску дуже важливо правильно обирати значення коефіцієнту α . Це не добре, оскільки для кожної функції значення цього коефіцієнту має бути своїм, а для того щоб обрати його оптимально доведеться експерементувати з різними значеннями кроку.

Також слід зауважити, що від початкової точки залежить знайдено ми глобальний чи локальний мінімуми.

Реалізація методу Хука-Дживса:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import time

function_calls = 0

def G(x1, x2):
    global function_calls
    function_calls += 1
    return 0.5 * ((np.pow(x1, 4) - 16 * np.pow(x1, 2) + 5 * x1) + (np.pow(x2, 4) - 16 * np.pow(x2, 2) + 5 * x2))

def new_point(x, h, index):
    x = x.copy()

    val_start = G(x[0], x[1])

    x_right = x.copy()
    x_right[index] += h
    val_right = G(x_right[0], x_right[1])

    x_left = x.copy()
    x_left[index] -= h
    val_left = G(x_left[0], x_left[1])

    if val_right < val_start and val_right < val_left:
        return x_right
    elif val_left < val_start and val_left < val_right:
        return x_left
    else:
        return x

def exploratory_search(x, h, accuracy):
    x_new = x.copy()
    h_new = h
    G_start = G(x[0], x[1])

    # Досліджуємо відповж x1
    x1_new = new_point(x_new, h, 0)
    G_x1_new = G(x1_new[0], x1_new[1])

    if G_x1_new < G_start:
        x_new = x1_new

    # Досліджуємо відповж x2
    x2_new = new_point(x_new, h, 1)
    G_x2_new = G(x2_new[0], x2_new[1])

    if G_x2_new < G_start:
        x_new = x2_new

    if h_new < accuracy:
        return x_new, h_new

    # Не знайшов кращу точку - повторюємо з меншим кроком
    if x_new == x:
        h_new = h / 2
```

```

        return exploratory_search(x_new, h_new, accuracy)

    return x_new, h_new

def pattern_move(x, h, accuracy):
    x_prev = x
    x_current, h_current = exploratory_search(x, h, accuracy)
    x_next = [x_current[i] + (x_current[i] - x_prev[i]) for i in range(len(x))]

    G2 = G(x_current[0], x_current[1])
    G3 = G(x_next[0], x_next[1])

    if G3 < G2:
        return exploratory_search(x_next, h, accuracy)
    else:
        return exploratory_search(x_current, h, accuracy)

def hooke_jeeves(x_start, h_start, accuracy, max_iter):
    x = x_start.copy()
    h = h_start
    path = [x.copy()]
    iterations = 0

    while h > accuracy and iterations < max_iter:
        x_new, h = pattern_move(x, h, accuracy)
        path.append(x_new.copy())
        x = x_new
        iterations += 1

    return x, G(x[0], x[1]), path

x_start = [2, 0]
h_start = 0.1
accuracy = 0.0001
max_iter = 1000

start_time = time.time()
min_point, min_value, path = hooke_jeeves(x_start, h_start, accuracy,
max_iter)
end_time = time.time()
elapsed_time = end_time - start_time

print(f'Точка мінімуму: ({min_point})\nЗначення функції: {min_value}')
print(f'КОЦФ: {function_calls}')
print(f'Час виконання: {elapsed_time:.6f} мікросекунд')

```

Тепер протестуємо роботу метода:

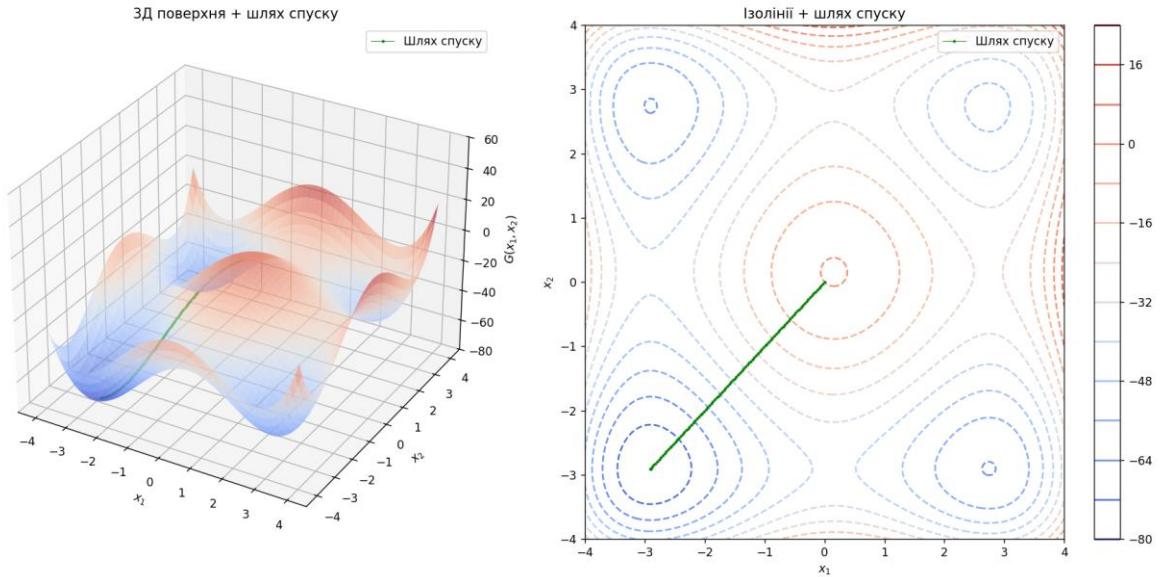
Початкова точка $(0, 0)$, $h = 0.01$, accuracy = 0.001:

Точка мінімуму: $([-2.9035156249999825, -2.9035156249999825])$

Значення функції: -78.33233139583095

КОЦФ: 2100

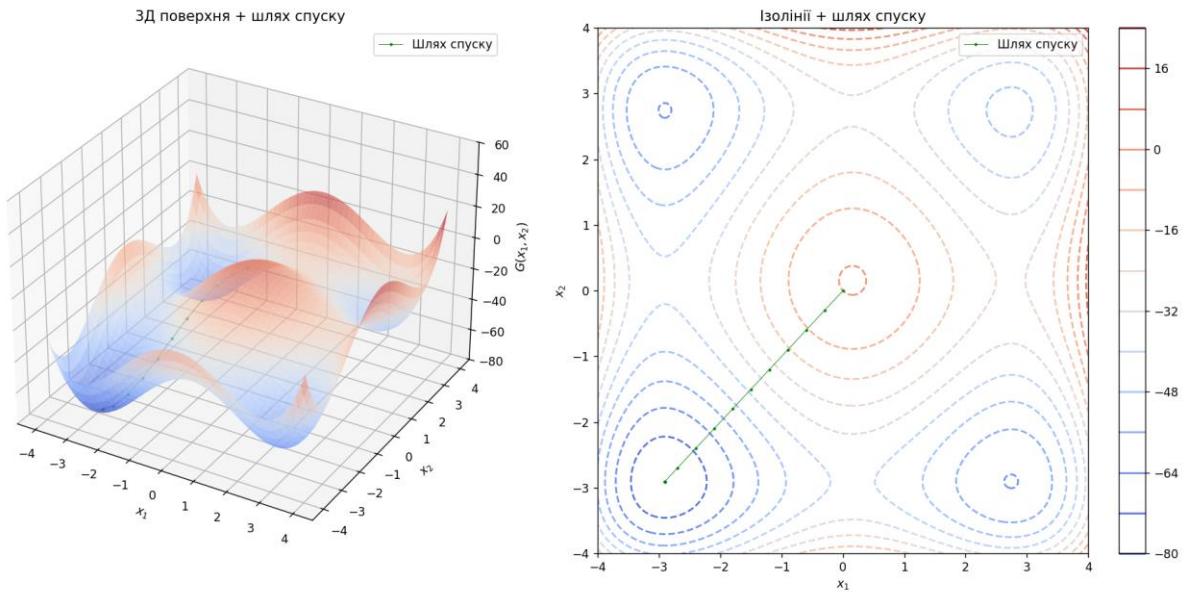
Час виконання: 0.011535 секунд



Збільшимо крок:

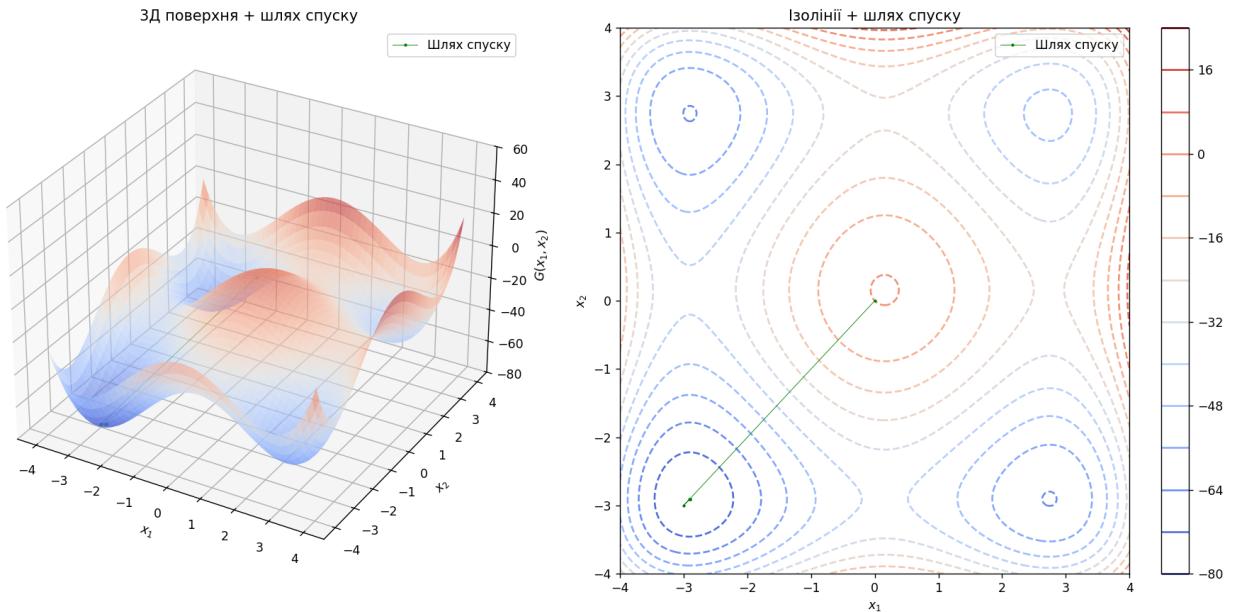
Початкова точка $(0, 0)$, $h = 0.1$, accuracy = 0.001:

- Точка мінімуму: $([-2.903515625000002, -2.903515625000002])$
- Значення функції: -78.33233139583095
- КОЦФ: 349
- Час виконання: 0.001909 секунд



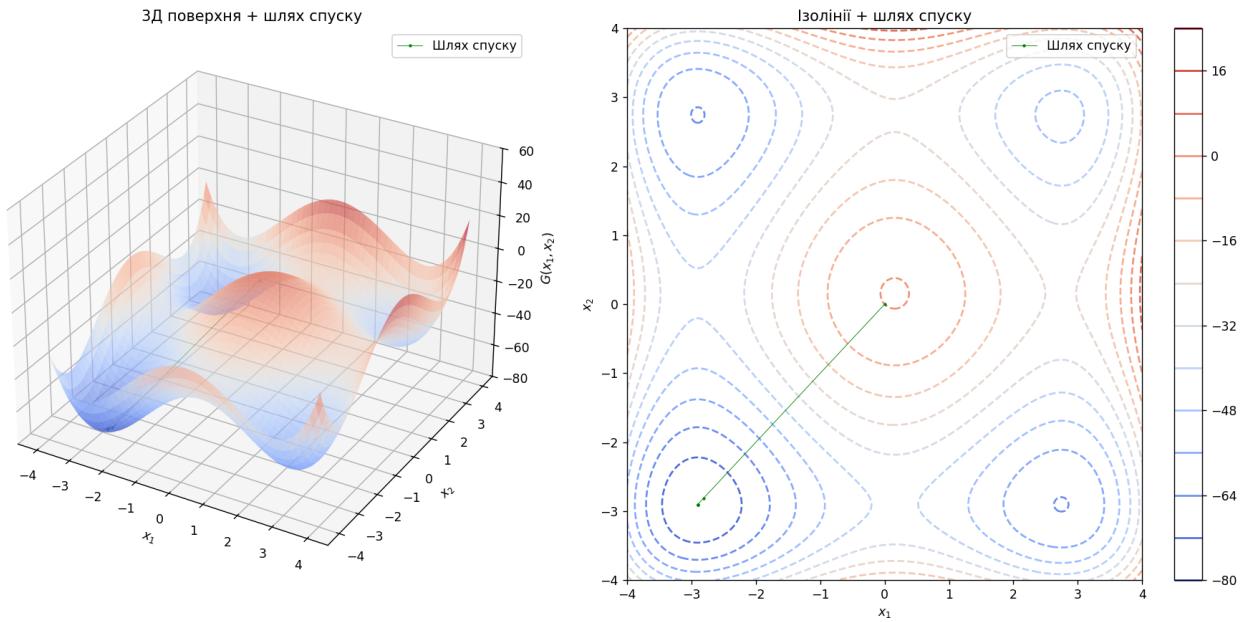
Початкова точка $(0, 0)$, $h = 1$, accuracy = 0.001:

- Точка мінімуму: $([-2.903564453125, -2.903564453125])$
- Значення функції: -78.3323313755289
- КОЦФ: 308
- Час виконання: 0.001880 секунд



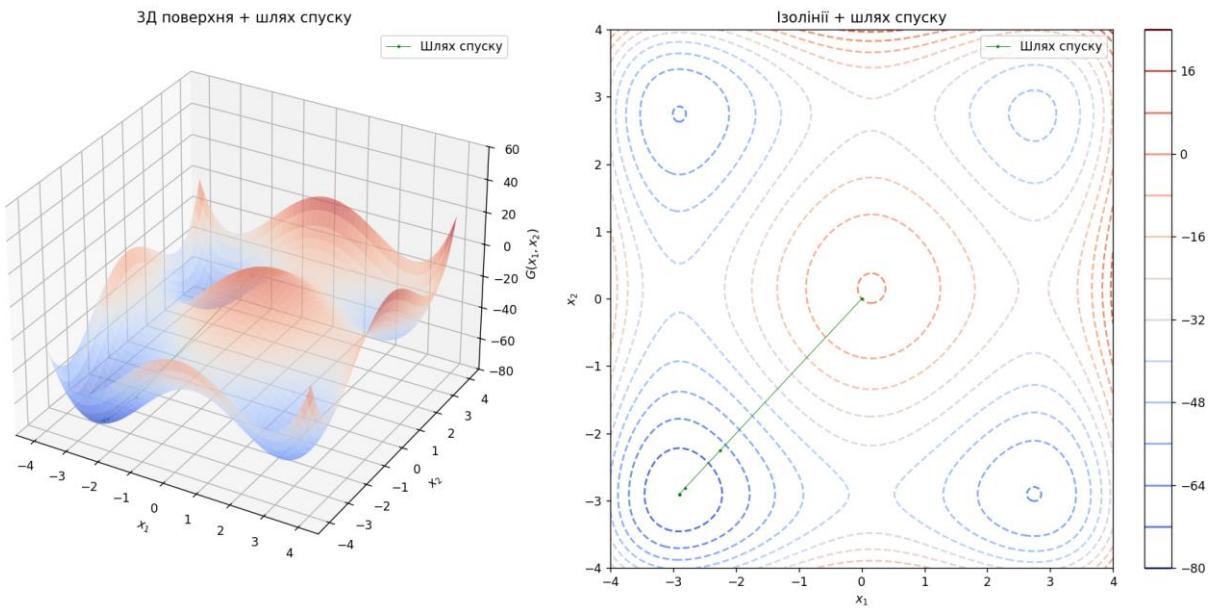
Початкова точка $(0, 0)$, $h = 3$, accuracy = 0.001:

- Точка мінімуму: $([-2.90350341796875, -2.90350341796875])$
- Значення функції: -78.33233137514021
- КОЦФ: 261
- Час виконання: 0.001471 секунд



Початкова точка $(0, 0)$, $h = 0.75$, accuracy = 0.001:

- Точка мінімуму: $([-2.90350341796875, -2.90350341796875])$
- Значення функції: -78.33233137514021
- КОЦФ: 263
- Час виконання: 0.001510 секунд

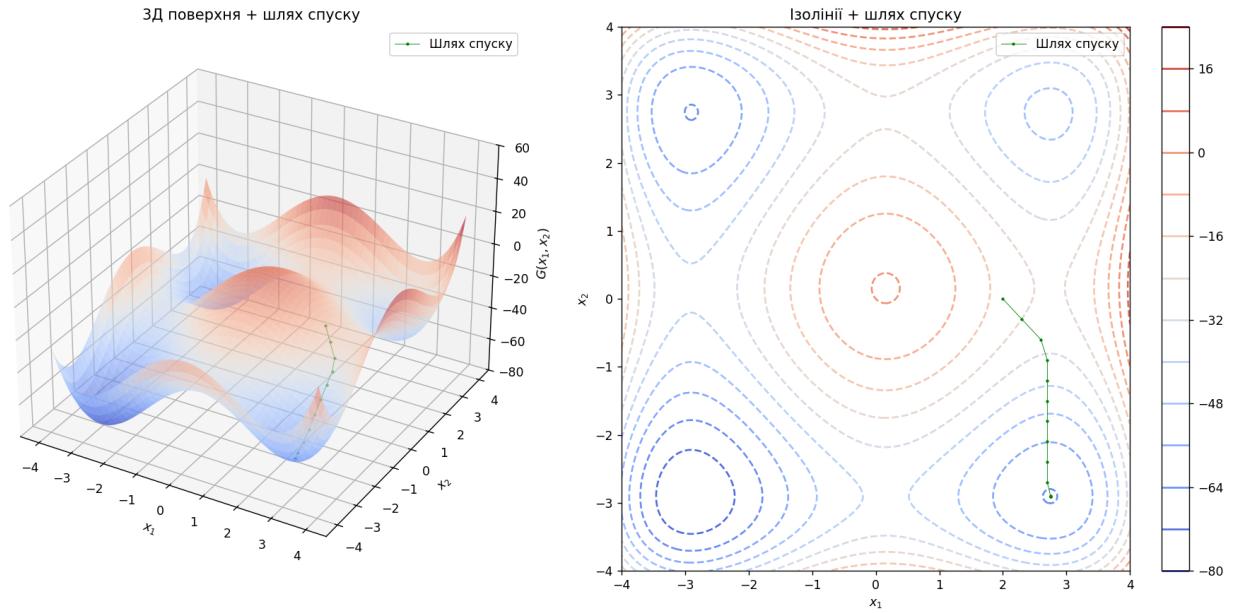


Візьмемо інші точки:

Початкова точка $(2, 0)$, $h = 0.1$, accuracy = 0.001:

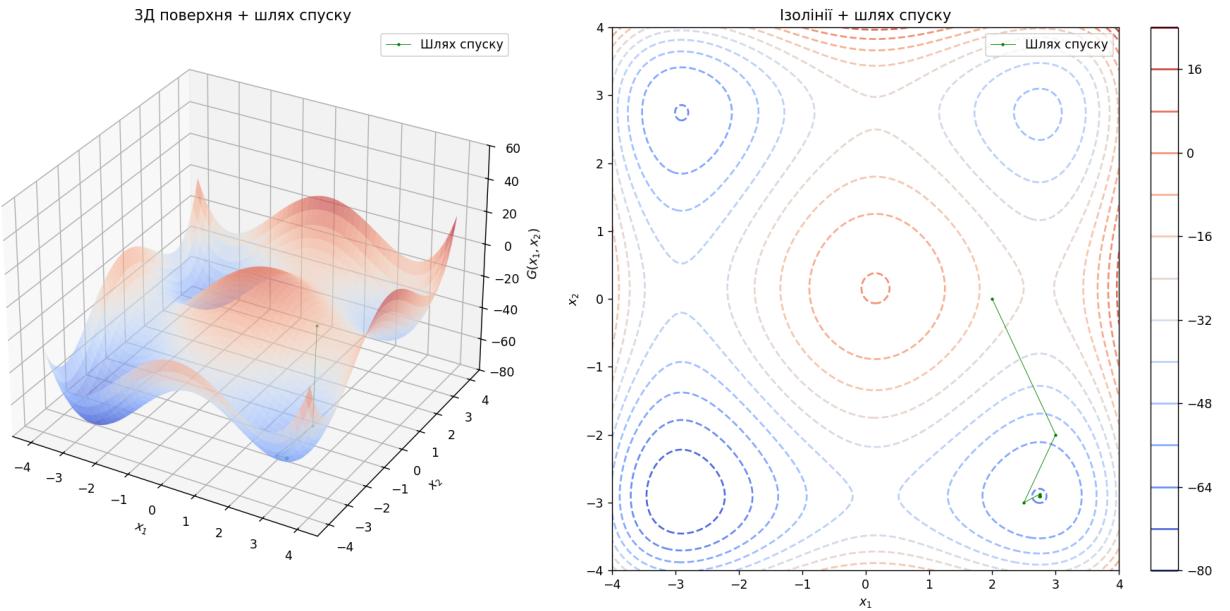
- Точка мінімуму: $([2.7467773437500003, -2.903515625000002])$
- Значення функції: -64.19561234373748
- КОЦФ: 414

- Час виконання: 0.002271 секунд



Початкова точка $(2, 0)$, $h = 1$, accuracy = 0.001:

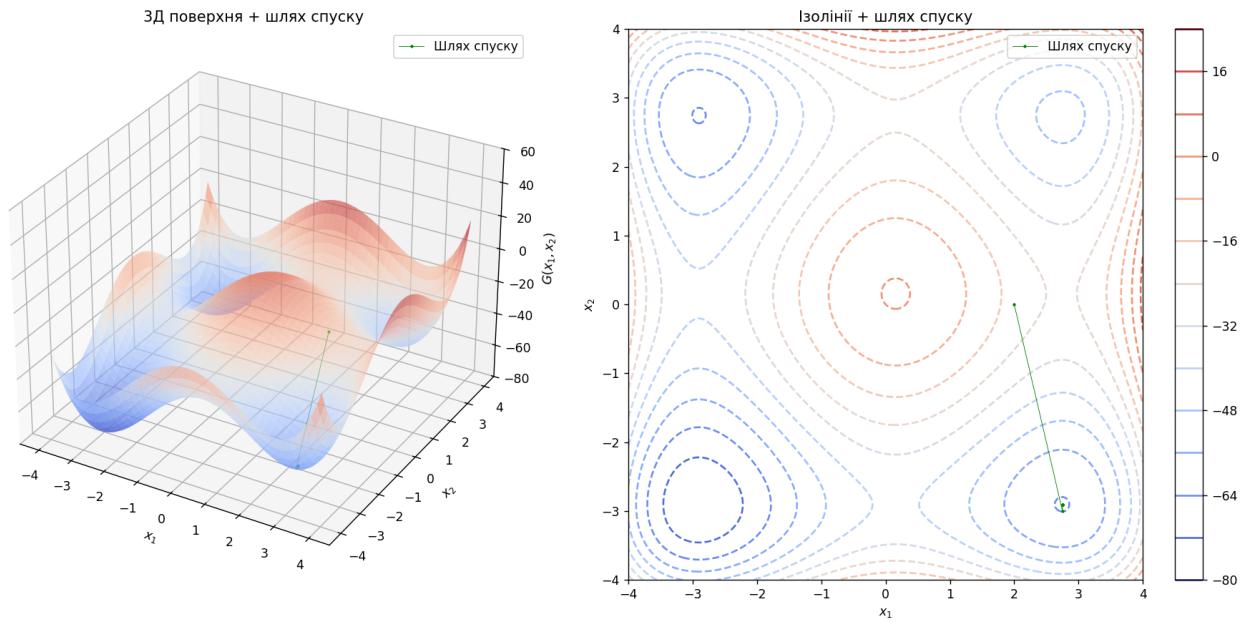
- Точка мінімуму: $([2.746826171875, -2.903564453125])$
- Значення функції: -64.1956123350343
- КОЦФ: 330
- Час виконання: 0.001880 секунд



Початкова точка $(2, 0)$, $h = 3$, accuracy = 0.001:

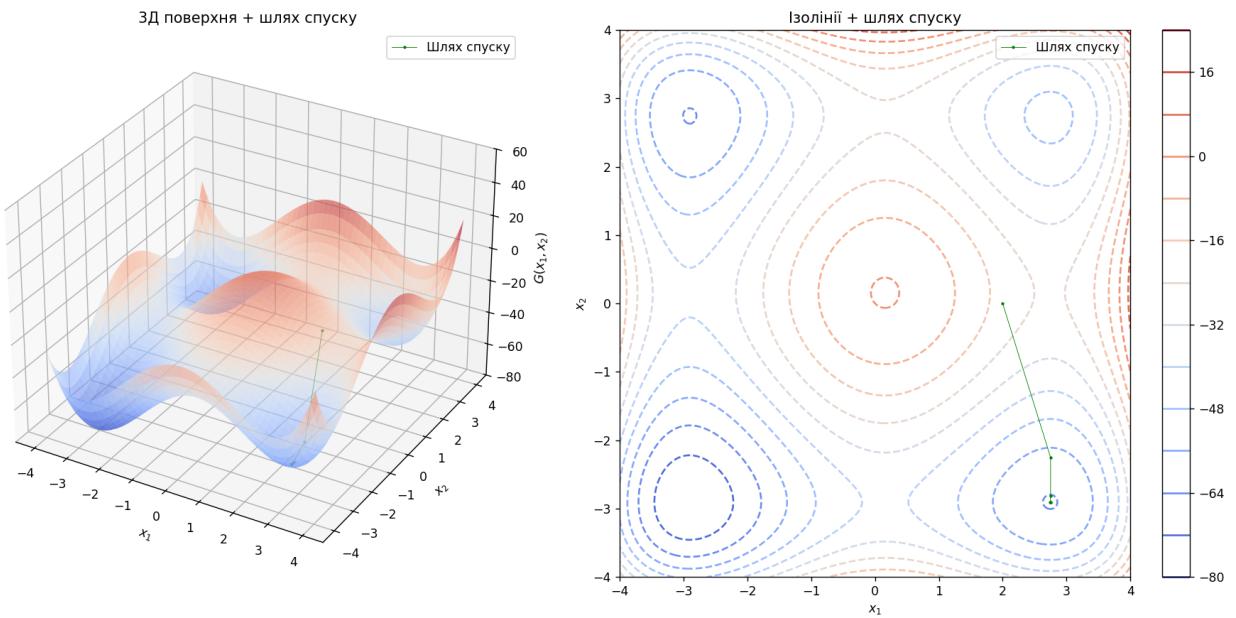
- Точка мінімуму: $([2.746795654296875, -2.90350341796875])$
- Значення функції: -64.19561234211284
- КОЦФ: 326

- Час виконання: 0.001879 секунд



Початкова точка $(2, 0)$, $h = 0.75$, accuracy = 0.001:

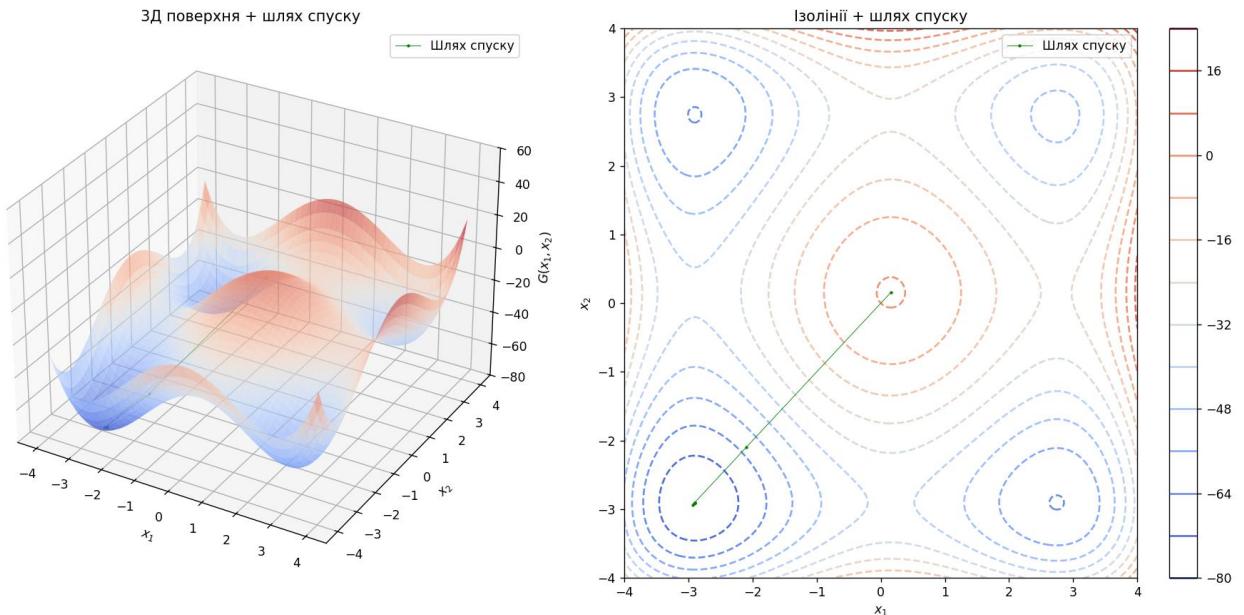
- Точка мінімуму: $([2.746795654296875, -2.90350341796875])$
- Значення функції: -64.19561234211284
- КОЦФ: 245
- Час виконання: 0.001558 секунд



Початкова точка $(0.156731, 0.156731)$, $h = 0.75$, accuracy = 0.001:

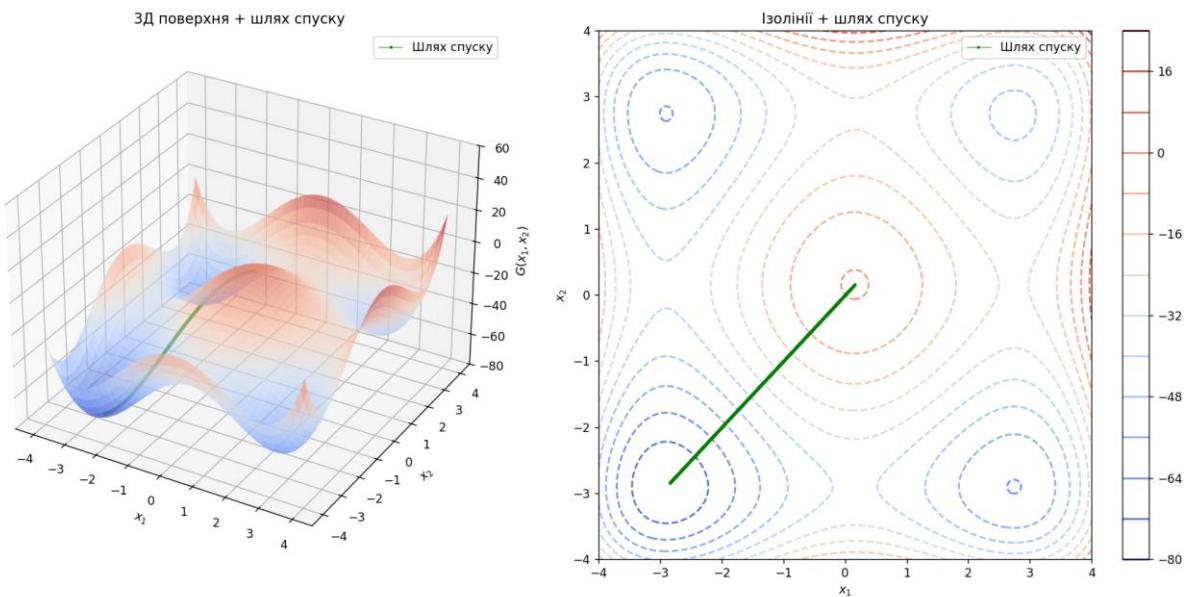
- Точка мінімуму: $([-2.9035106992187503, -2.9035106992187503])$
- Значення функції: -78.33233138872215

- КОЦФ: 274
- Час виконання: 0.001488 секунд



Початкова точка (0.156731, 0.156731), $h = 0.001$, accuracy = 0.001:

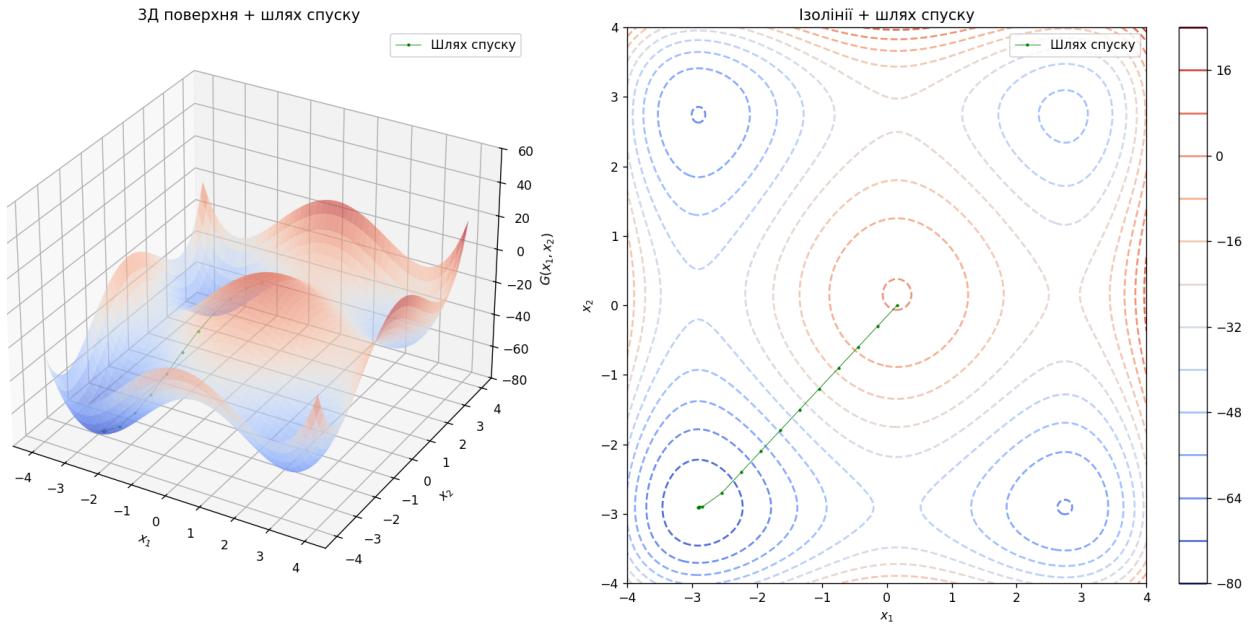
- Точка мінімуму: $([-2.843268999999798, -2.843268999999798])$
- Значення функції: -78.20925896223031
- КОЦФ: 20001
- Час виконання: 0.107245 секунд



Початкова точка (0.156731, 0), $h = 0.1$, accuracy = 0.001:

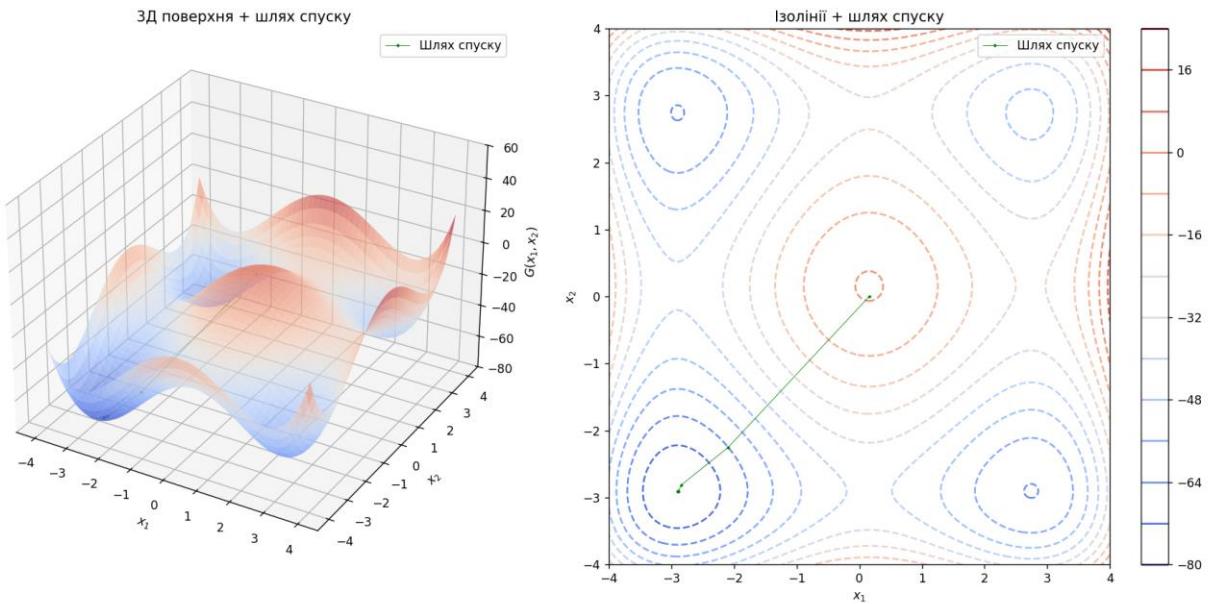
- Точка мінімуму: $([-2.903522906250002, -2.903515625000002])$
- Значення функції: -78.33233139954814
- КОЦФ: 436

- Час виконання: 0.003706 секунд



Початкова точка $(0.156731, 0)$, $h = 0.75$, accuracy = 0.001:

- Точка мінімуму: $([-2.9035106992187503, -2.90350341796875])$
- Значення функції: -78.33233138193118
- КОЦФ: 321
- Час виконання: 0.001757 секунд



Аналіз результатів:

Точка	h	Результати
(0, 0)	0.01	Точка мінімуму: $([-2.9035156249999825, -2.9035156249999825])$ Значення функції: -78.33233139583095 КОЦФ: 2100 Час виконання: 0.011535 секунд
(0, 0)	0.1	Точка мінімуму: $([-2.903515625000002, -2.903515625000002])$ Значення функції: -78.33233139583095 КОЦФ: 349 Час виконання: 0.001909 секунд
(2,0)	0.1	Точка мінімуму: $([2.7467773437500003, -2.903515625000002])$ Значення функції: -64.19561234373748 КОЦФ: 414 Час виконання: 0.002271 секунд
(0.156731, 0)	0.1	Точка мінімуму: $([-2.903522906250002, -2.903515625000002])$ Значення функції: -78.33233139954814 КОЦФ: 436 Час виконання: 0.003706 секунд
(0, 0)	1	Точка мінімуму: $([-2.903564453125, -2.903564453125])$ Значення функції: -78.3323313755289 КОЦФ: 308 Час виконання: 0.001880 секунд
(2, 0)	1	Точка мінімуму: $([2.746826171875, -2.903564453125])$ Значення функції: -64.1956123350343 КОЦФ: 330 Час виконання: 0.001880 секунд
(0.156731, 0)	1	Точка мінімуму: $([-2.9035106992187503, -2.903564453125])$ Значення функції: -78.33233138212552 КОЦФ: 319 Час виконання: 0.001697 секунд
(0, 0)	0.75	Точка мінімуму: $([-2.90350341796875, -2.90350341796875])$ Значення функції: -78.33233137514021 КОЦФ: 263 Час виконання: 0.001510 секунд
(2, 0)	0.75	Точка мінімуму: $([2.746795654296875, -2.90350341796875])$ Значення функції: -64.19561234211284 КОЦФ: 245 Час виконання: 0.001558 секунд
(0.156731,0)	0.75	Точка мінімуму: $([-2.9035106992187503, -2.90350341796875])$ Значення функції: -78.33233138193118 КОЦФ: 321 Час виконання: 0.001757 секунд

Проаналізувавши результати, робимо висновок, що результат роботи метода Хука-Дживса досить сильно залежить від обраного кроку, а також є значно більш стійкий за метод найшвидшого спуску.

Кількість обчислень в залежності від обраного кроку:

Візьмемо точку (0.156731, 0) та подивимося, які результати пошуку з цієї точки для різних кроків:

Крок	0.01	0.1	0.75
Результат	Значення функції: -78.33233138544007 КОЦФ: 2191 Час виконання: 0.012884 секунд	Значення функції: -78.33233139954814 КОЦФ: 436 Час виконання: 0.003706 секунд	Значення функції: -78.33233138193118 КОЦФ: 321 Час виконання: 0.001757 секунд

Бачимо, що правильно обраний крок значно знижує кількість обчислень цільової функції та скорочує час виконання алгоритму.

Окрім початкового кроку можна ще змінювати дільник у формулі для обчислення нового кроку. При правильному поєднанні кроку та дільника можна отримати куди кращі результати роботи метода Хука-Дживса:

```
h_new = h / 40
```

Крок h_new	0.75 h_new = h / 2	0.75 h_new = h / 40
Результат	Значення функції: -78.33233138193 КОЦФ: 321 Час виконання: 0.001757 секунд	Значення функції: -78.332026818413 КОЦФ: 99 Час виконання: 0.000573 секунд

Отже, для метода Хука-Дживса дуже важливо підібрати правильно не тільки крок, а й дільник для обчислення нового кроку.

Реалізація методу пошуку за симплексом (Спенделі-Хекста-Химсвортса)

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import time
import math

function_calls = 0

def G(x1, x2):
    global function_calls
    function_calls += 1
    return 0.5 * ((np.pow(x1, 4) - 16 * np.pow(x1, 2) + 5 * x1) + (np.pow(x2, 4) - 16 * np.pow(x2, 2) + 5 * x2))

def build_regular_simplex(x0, alpha):
    x0 = np.array(x0, dtype=float)
    simplex = [x0.tolist()]

    sigma1 = ((np.sqrt(N + 1) + N - 1) / (N * np.sqrt(2))) * alpha
    sigma2 = ((np.sqrt(N + 1) - 1) / (N * np.sqrt(2))) * alpha

    for i in range(N):
        xi = np.array(x0)
        for j in range(N):
            if i != j:
                xi[j] = xi[j] + sigma1
            else:
                xi[j] = xi[j] + sigma2
        simplex.append(xi.tolist())

    return simplex

def build_regular_simplex_around_point(x0, alpha):
    x0 = np.array(x0, dtype=float)
    simplex = []

    for i in range(3):
        angle = 2 * math.pi * i / 3
        dx = alpha * math.cos(angle)
        dy = alpha * math.sin(angle)
        point = x0 + np.array([dx, dy])
        simplex.append(point.tolist())

    return simplex

def find_centre(simplex, remove_index):
    sum_coords = []
    for j in range(N + 1):
        if j != remove_index:
            sum_coords.append(simplex[j])

    centre = [0, 0]
    for i in range(N):
        for j in range(N):
            centre[i] += sum_coords[j][i]
        centre[i] /= N
    return centre

def reflect_point(simplex, index):
```

```

xc = find_centre(simplex, index)
xprev = simplex[index]

xnew = [0, 0]
for i in range(N):
    xnew[i] = xc[i] + (xc[i] - xprev[i])

return xnew

def myFunc(e):
    return e[0]

def function_range(evaluated):
    values = [val for val, _ in evaluated]
    return max(values) - min(values)

def shh(x0, a, accuracy, max_iter):
    simplex_path = []
    simplex = build_regular_simplex_around_point(x0, a)
    simplex_path.append([p.copy() for p in simplex])

    repeat_count = [0] * (N + 1)
    M = round(1.65 * N + 0.05 * N ** 2)

    iter = 0

    while iter < max_iter:
        # Обчислюємо значення у кожній вершині
        evaluated = []

        for point in simplex:
            value = G(point[0], point[1])
            evaluated.append((value, point))

        evaluated.sort(key=myFunc)

        if function_range(evaluated) < accuracy:
            break

        # відобразити вершину з найгіршим значенням
        worst_index = simplex.index(evaluated[2][1])
        xnew = reflect_point(simplex, worst_index)

        new_value = G(xnew[0], xnew[1])
        evaluated[2] = (new_value, xnew)

        evaluated.sort(key=myFunc)

        # 1 правило
        if xnew == evaluated[2][1]: # нова точка виявилася найгіршою
            worst_index = simplex.index(evaluated[1][1]) # потрібно
            виключити другу за значенням точку
            xnew = reflect_point(simplex, worst_index)
            simplex[worst_index] = xnew
        else:
            simplex[worst_index] = xnew

        # 2 правило
        for i in range(N+1):
            if i == worst_index:
                repeat_count[i] = 0

```

```

    else:
        repeat_count[i] += 1

    if max(repeat_count) >= M:
        best_point = evaluated[0][1]
        a = a / 2 # редукція
        simplex = build_regular_simplex_around_point(best_point, a)
        repeat_count = [0] * (N + 1)

    simplex_path.append([p.copy() for p in simplex])
    iter += 1

# Знаходимо найкращу точку після завершення
best_value = float('inf')
xmin = evaluated[0][1]
gmin = evaluated[0][0]

return xmin, gmin, simplex_path

x_start = [0.156731, 0.156731]
a = 0.5
accuracy = 0.001
max_iter = 1000
N = 2

start = time.perf_counter_ns()
xmin, gmin, simplex_path = shh(x_start, a, accuracy, max_iter)
end = time.perf_counter_ns()
elapsed = end - start

print(f'Точка мінімуму: ({xmin})\nЗначення функції: {gmin}')
print(f'КОЦФ: {function_calls}')
print(f"Час виконання: {elapsed / 1_000_000_000:.6f} секунд")

# Візуалізація

x1_vals = np.linspace(-4, 4, 200)
x2_vals = np.linspace(-4, 4, 200)
X1, X2 = np.meshgrid(x1_vals, x2_vals)
Z = G(X1, X2)

fig = plt.figure(figsize=(12, 5))

# 3D графік
ax1 = fig.add_subplot(1, 2, 1, projection='3d')
ax1.plot_surface(X1, X2, Z, cmap=cm.viridis, alpha=0.5)

for triangle in simplex_path:
    triangle_closed = triangle + [triangle[0]]
    xs = [p[0] for p in triangle_closed]
    ys = [p[1] for p in triangle_closed]
    zs = [G(p[0], p[1]) for p in triangle_closed]
    ax1.plot(xs, ys, zs, color='blue', lw=0.8)

ax1.set_zlim(-80, 60)
ax1.set_xlabel('$x_1$')
ax1.set_ylabel('$x_2$')
ax1.set_zlabel('$G(x_1, x_2)$')
ax1.set_title('3D поверхня + симплекс')

# Ізолінії
ax2 = fig.add_subplot(1, 2, 2)
contour = ax2.contour(X1, X2, Z, cmap=cm.viridis, levels=12,

```

```

linestyles='dashed')

for triangle in simplex_path:
    triangle_closed = triangle + [triangle[0]]
    xs = [p[0] for p in triangle_closed]
    ys = [p[1] for p in triangle_closed]
    ax2.plot(xs, ys, color='blue', linewidth=0.8, label='_simplex')

plt.colorbar(contour, ax=ax2)
ax2.set_xlabel('$x_1$')
ax2.set_ylabel('$x_2$')
ax2.set_title('Ізолінії + симплекс')

plt.tight_layout()
plt.show()

```

У даному алгоритмі реалізовано два альтернативні методи для побудови симплексу:

```

def build_regular_simplex(x0, alpha):
    x0 = np.array(x0, dtype=float)
    simplex = [x0.tolist()]

    sigma1 = ((np.sqrt(N + 1) + N - 1) / (N * np.sqrt(2))) * alpha
    sigma2 = ((np.sqrt(N + 1) - 1) / (N * np.sqrt(2))) * alpha

    for i in range(N):
        xi = np.array(x0)
        for j in range(N):
            if i != j:
                xi[j] = xi[j] + sigma1
            else:
                xi[j] = xi[j] + sigma2
        simplex.append(xi.tolist())

    return simplex

```

Перший буде симплекс за даними формулами:

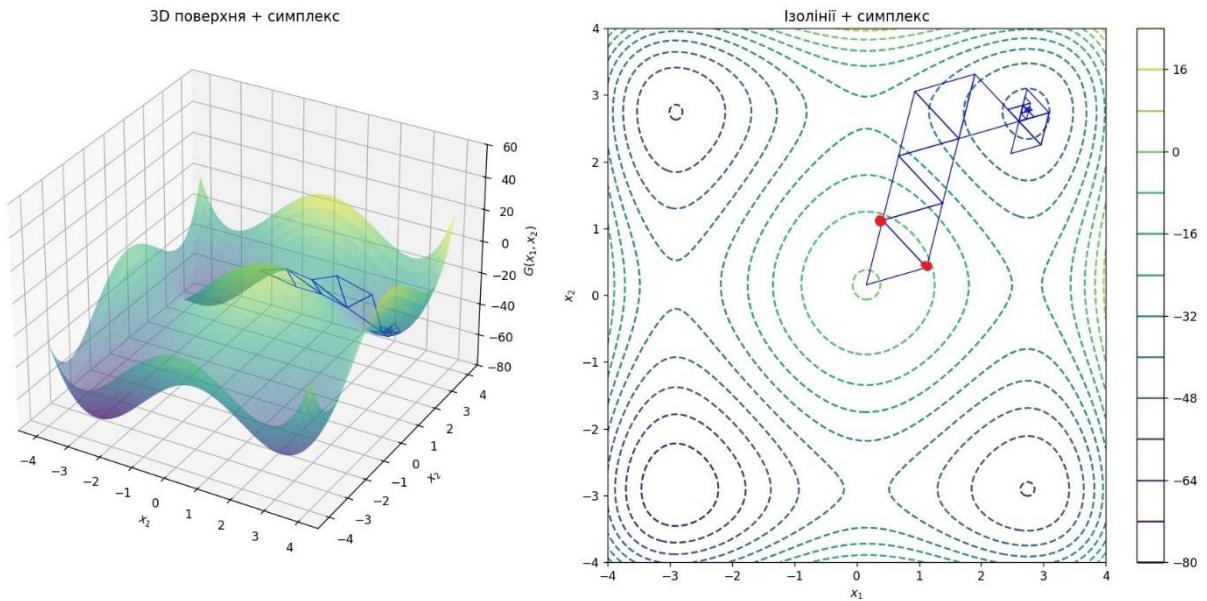
$$x_j^{(i)} = \begin{cases} x^{(0)} + \delta_1, & \text{якщо } j \neq i \\ x^{(0)} + \delta_2, & \text{якщо } j = i \end{cases}$$

для $i, j = 1, 2, 3, \dots, N$. Величини δ_1, δ_2 залежать від розмірності простору та обраного масштабного множника:

$$\delta_1 = \left[\frac{(N+1)^{\frac{1}{2}} + N - 1}{N\sqrt{2}} \right] \alpha$$

$$\delta_2 = \left[\frac{(N+1)^{\frac{1}{2}} - 1}{N\sqrt{2}} \right] \alpha$$

У результаті отримуємо дві нові точки, які разом із початковою утворюють симплекс. Такий підхід призводить до того, що в деяких точках алгоритм може визначати локальний мінімум замість глобального. Це пов'язано з тим, що координатний зсув у формулах не однаковий для всіх осей одночасно. У такому випадку відбувається зміщення симплексу в якийсь один бік. Для наглядності наведемо приклад:



Тут ми запустили алгоритм з точки максимуму функції та прийшли у локальний мінімум. На рисунку червоними точками позначені точки, знайдені за допомогою формул.

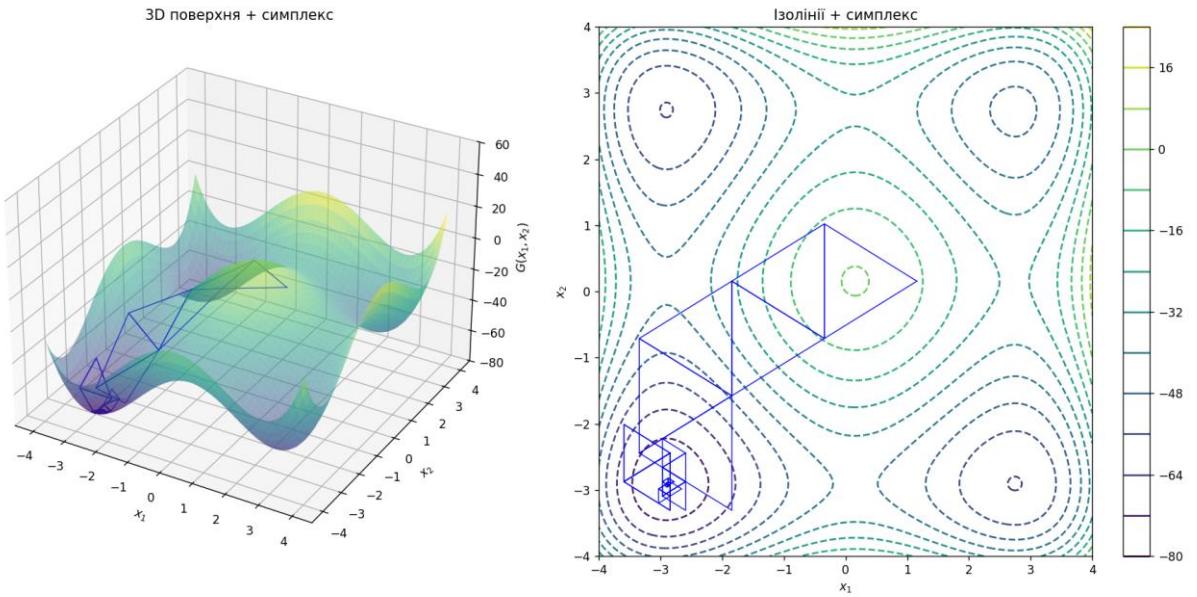
Для того, щоб уникнути нахилу потрібно будувати симплекс навколо заданої точки, для цього реалізовано інший метод:

```
def build_regular_simplex_around_point(x0, alpha):
    x0 = np.array(x0, dtype=float)
    simplex = []

    for i in range(3):
        angle = 2 * math.pi * i / 3
        dx = alpha * math.cos(angle)
        dy = alpha * math.sin(angle)
        point = x0 + np.array([dx, dy])
        simplex.append(point.tolist())

    return simplex
```

Нижче на рисунку показано його роботу:



Тут ми дослідили вже весь окіл точки максимуму, а не якусь частину, та спустилися у глобальний мінімум. Тому такий підхід є більш оптимальним, і надалі ми будемо використовувати саме його.

Тестування методу:

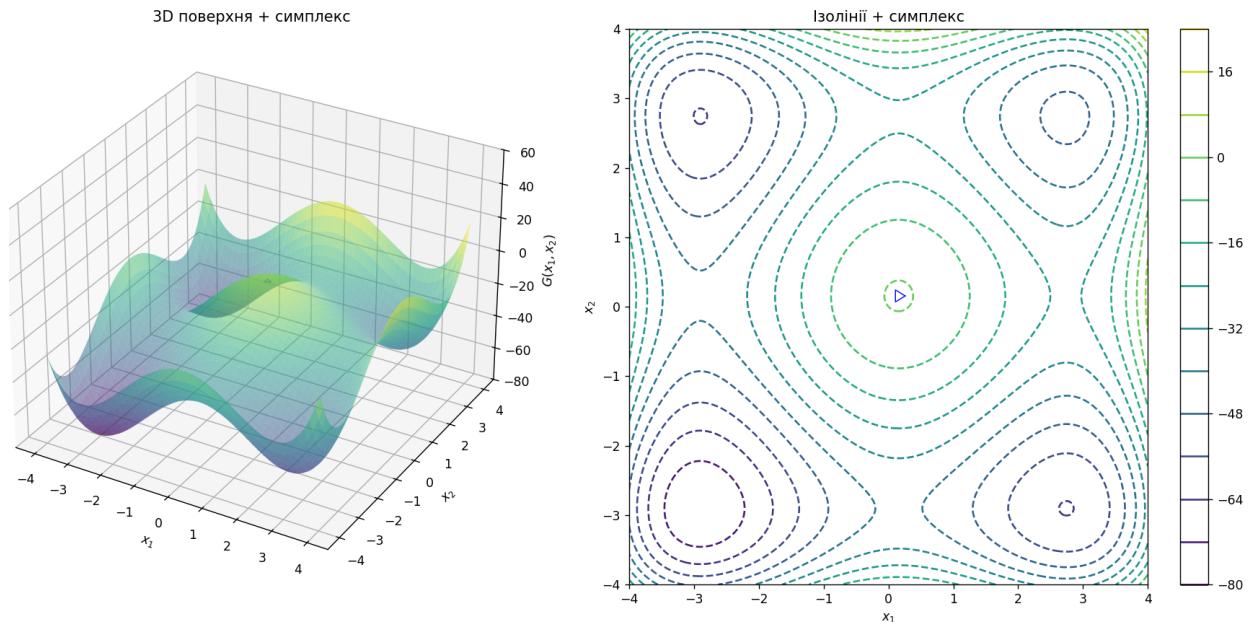
Початкова точка $(0.156731, 0.156731)$, $a = 0.1$, accuracy = 0.001:

Точка мінімуму: $([0.1067309999999996, 0.07012845962155616])$

Значення функції: 0.3117495679495137

КОЦФ: 3

Час виконання: 0.000083 секунд



Значення мінімуму визначено неправильно, оскільки розмір симплексу став меншим за точність. Збільшимо точність:

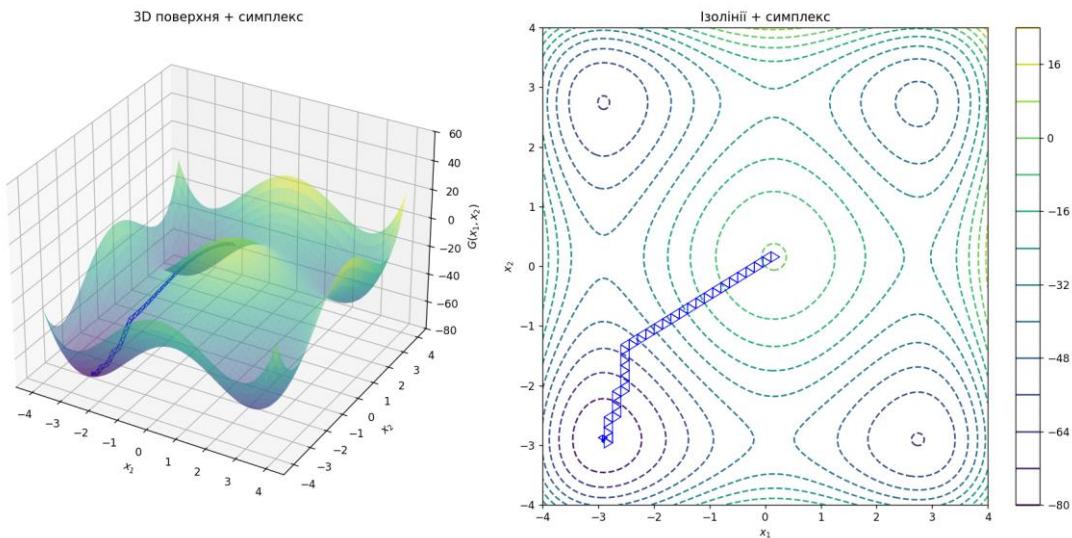
Початкова точка $(0.156731, 0.156731)$, $a = 0.1$, accuracy = 0.0001:

Точка мінімуму: $([-2.9026440000000053, -2.904127536500624])$

Значення функції: -78.3323116219608

КОЦФ: 315

Час виконання: 0.002055 секунд



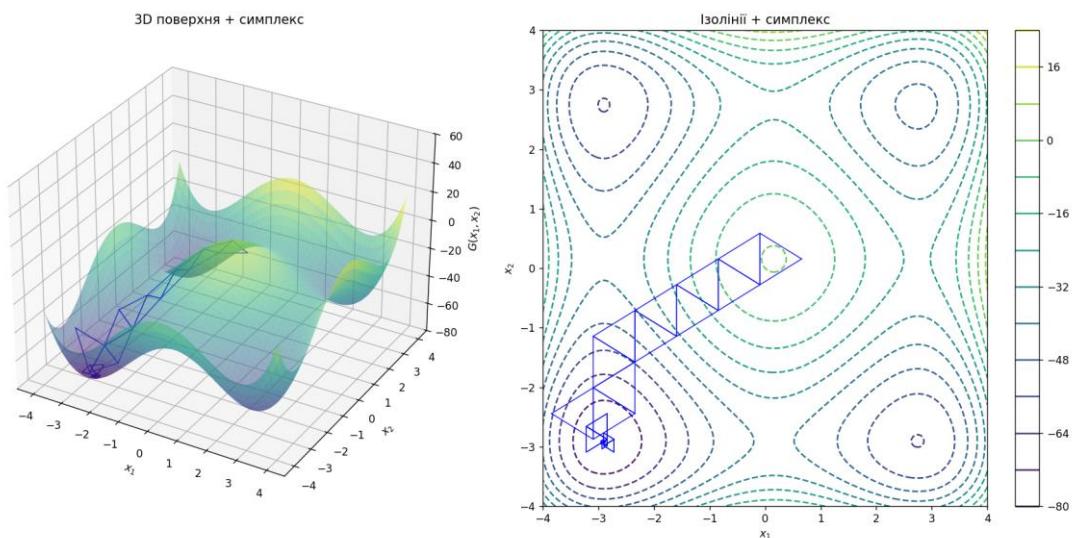
Початкова точка $(0.156731, 0.156731)$, $a = 0.5$, accuracy = 0.001:

Точка мінімуму: $([-2.9018627500000003, -2.9014212071137977])$

Значення функції: -78.33220600165616

КОЦФ: 147

Час виконання: 0.001063 секунд



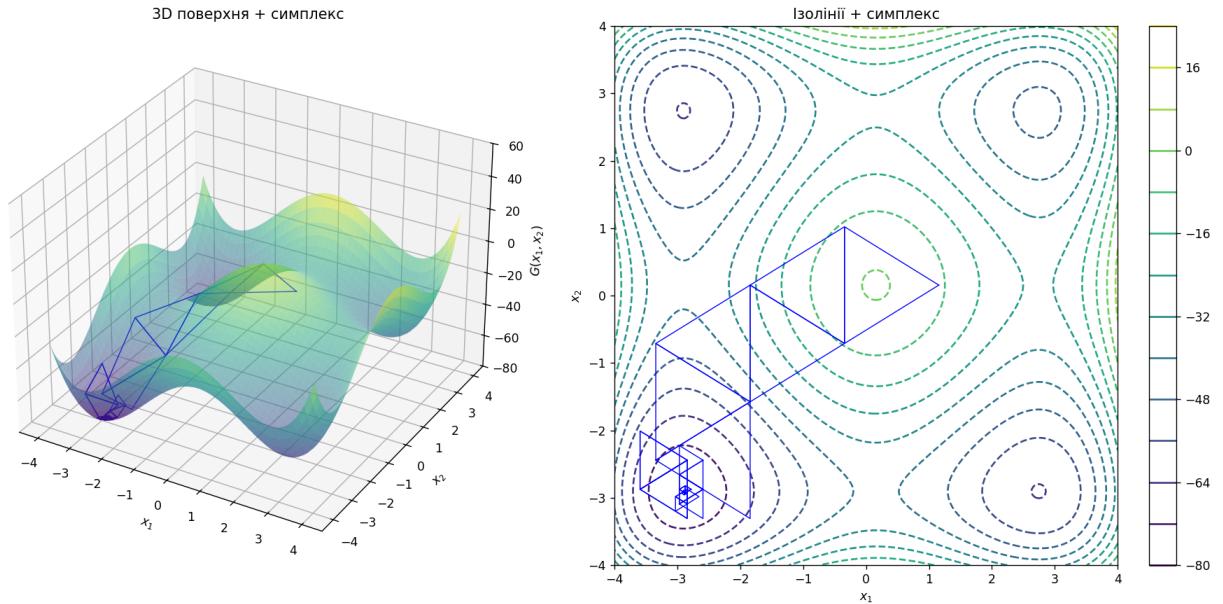
Початкова точка (0.156731, 0.156731), a = 1, accuracy = 0.001:

Точка мінімуму: [-2.9038158750000003, -2.9048041188473337])

Значення функції: -78.33230212840348

КОЦФ: 147

Час виконання: 0.001074 секунд



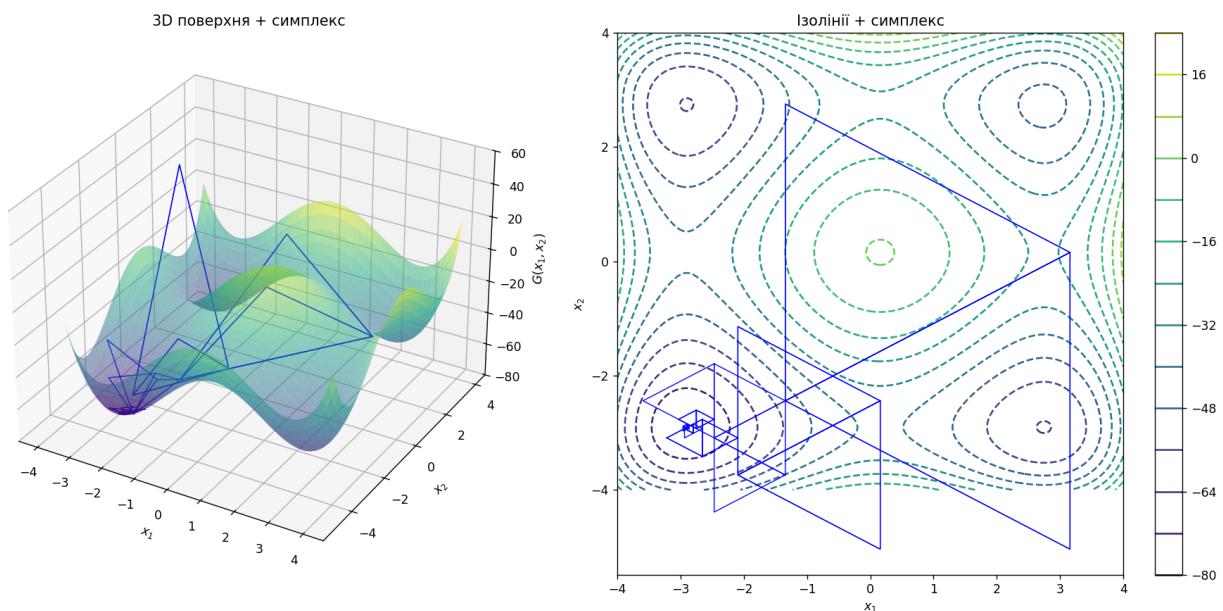
Початкова точка (0.156731, 0.156731), a = 3, accuracy = 0.001:

Точка мінімуму: [-2.901862750000002, -2.9081870305808644])

Значення функції: -78.33190818210139

КОЦФ: 151

Час виконання: 0.001119 секунд



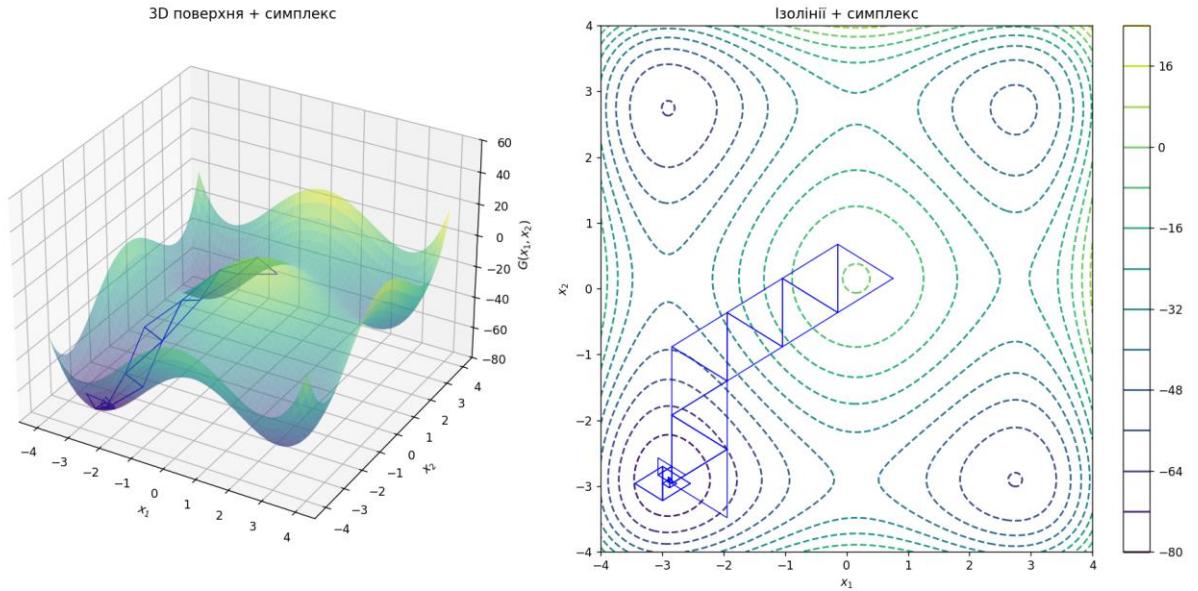
Початкова точка $(0.156731, 0.156731)$, $a = 0.6$, accuracy = 0.001:

Точка мінімуму: $([-2.899519, -2.8960085483401445])$

Значення функції: -78.33107624093535

КОЦФ: 131

Час виконання: 0.000968 секунд



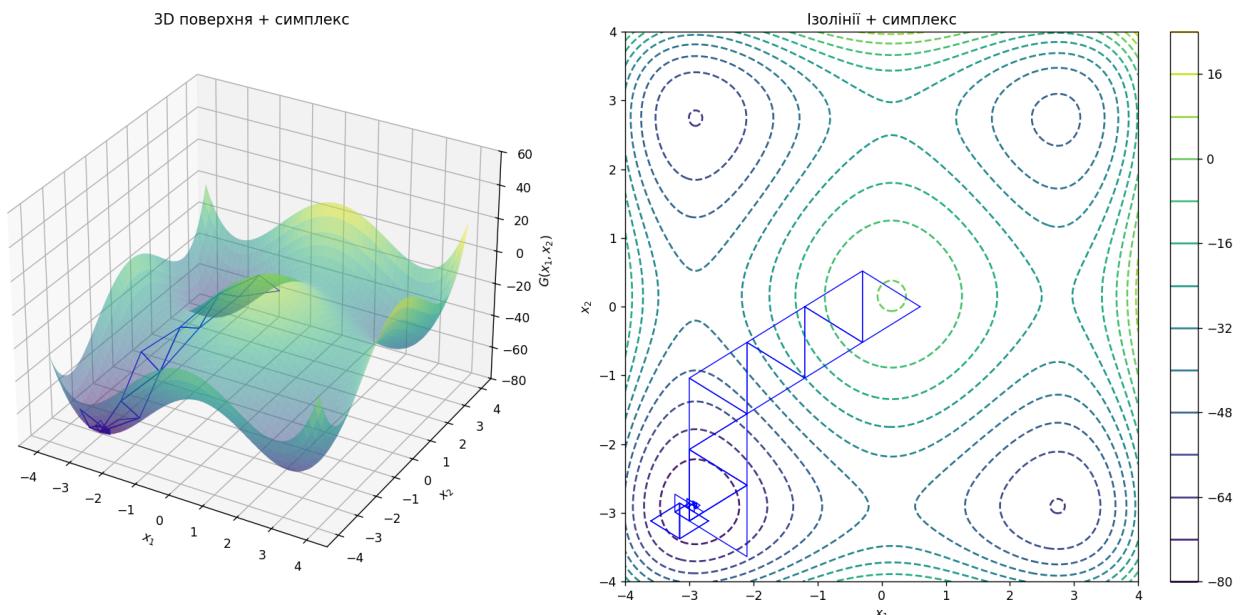
Початкова точка $(0, 0)$, $a = 0.6$, accuracy = 0.001:

Точка мінімуму: $([-2.903906250000001, -2.9025382673712814])$

Значення функції: -78.33231187202352

КОЦФ: 151

Час виконання: 0.001083 секунд



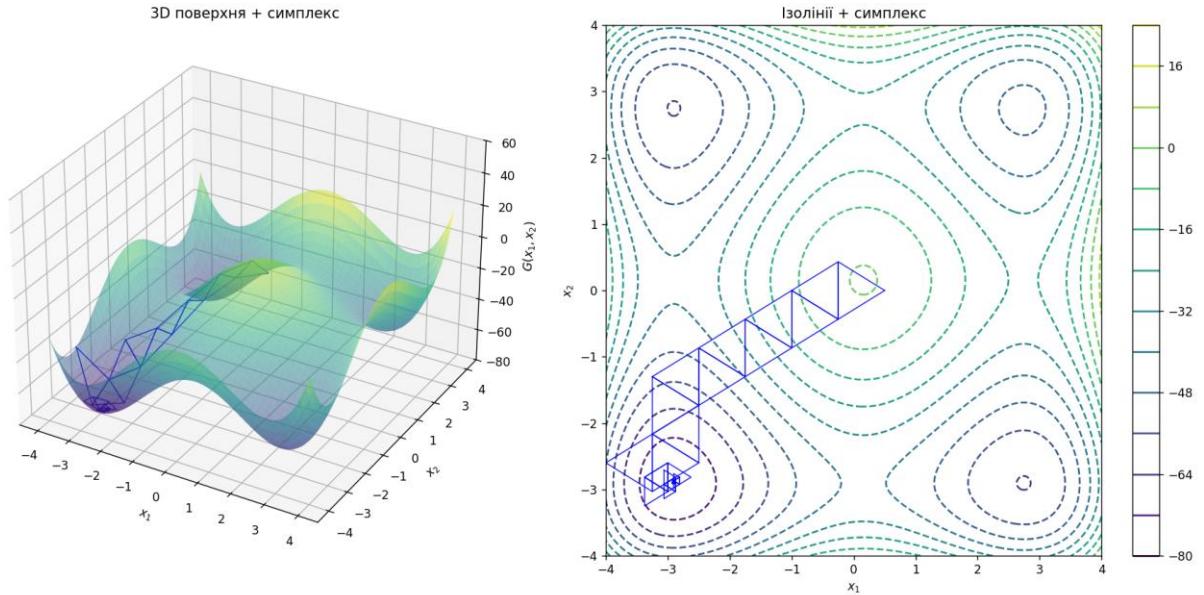
Початкова точка $(0, 0)$, $a = 0.5$, accuracy = 0.001:

Точка мінімуму: $([-2.9101562500000036, -2.9025382673712827])$

Значення функції: -78.33155428091305

КОЦФ: 135

Час виконання: 0.001012 секунд



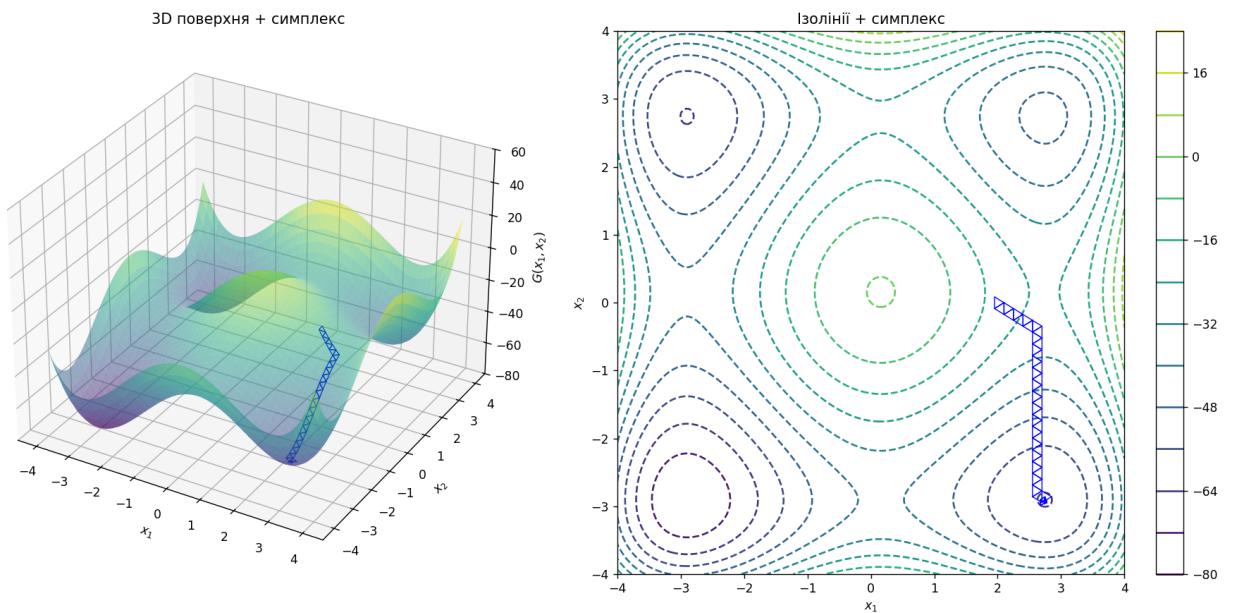
Початкова точка $(2, 0)$, $a = 0.1$, accuracy = 0.001:

Точка мінімуму: $([2.7453124999999874, -2.9038914320646936])$

Значення функції: -64.19557766570131

КОЦФ: 223

Час виконання: 0.001509 секунд



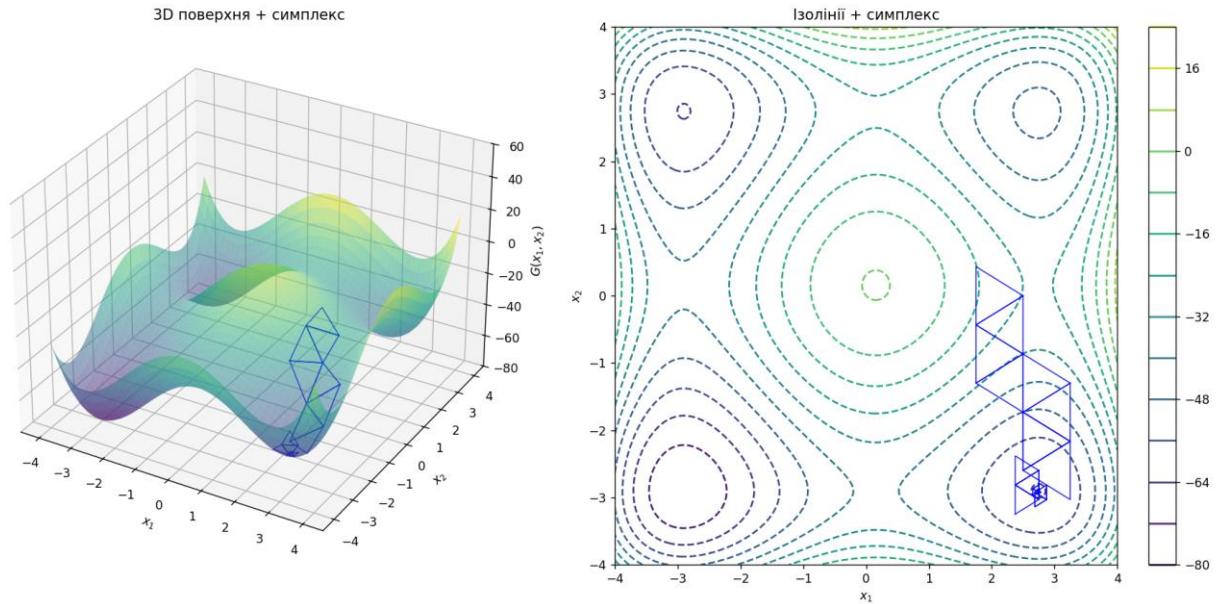
Початкова точка $(2, 0)$, $a = 0.5$, accuracy = 0.001:

Точка мінімуму: $([2.75, -2.9025382673712836])$

Значення функції: -64.19544543928242

КОЦФ: 135

Час виконання: 0.001015 секунд



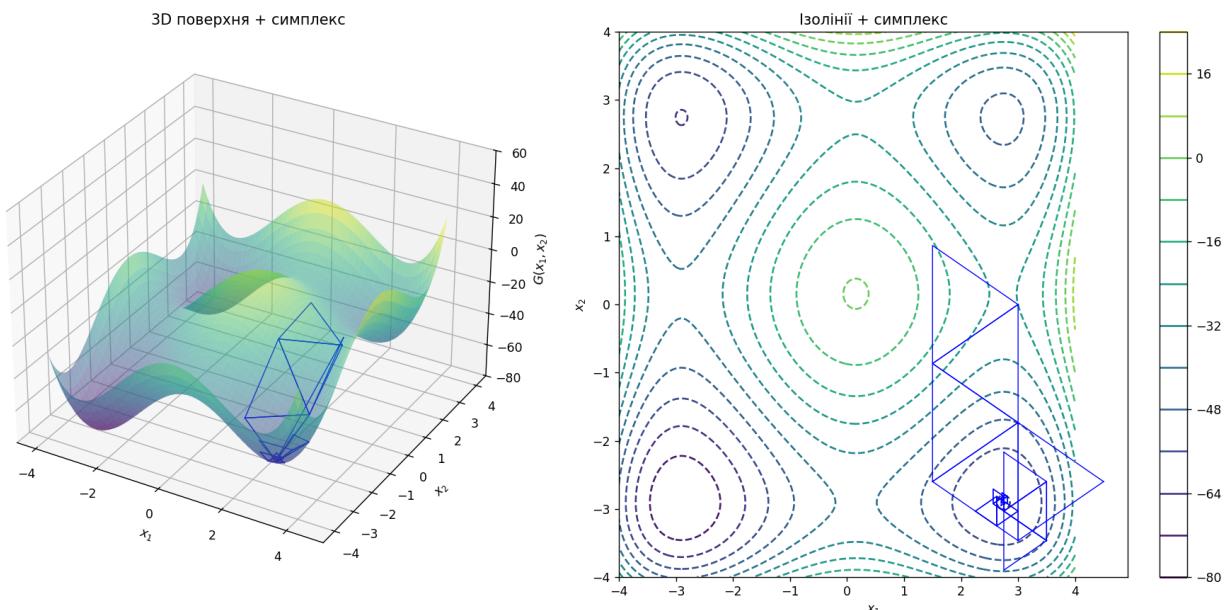
Початкова точка $(2, 0)$, $a = 1$, accuracy = 0.001:

Точка мінімуму: $([2.7480468750000004, -2.9059211791048156])$

Значення функції: -64.19549108220028

КОЦФ: 143

Час виконання: 0.001007 секунд



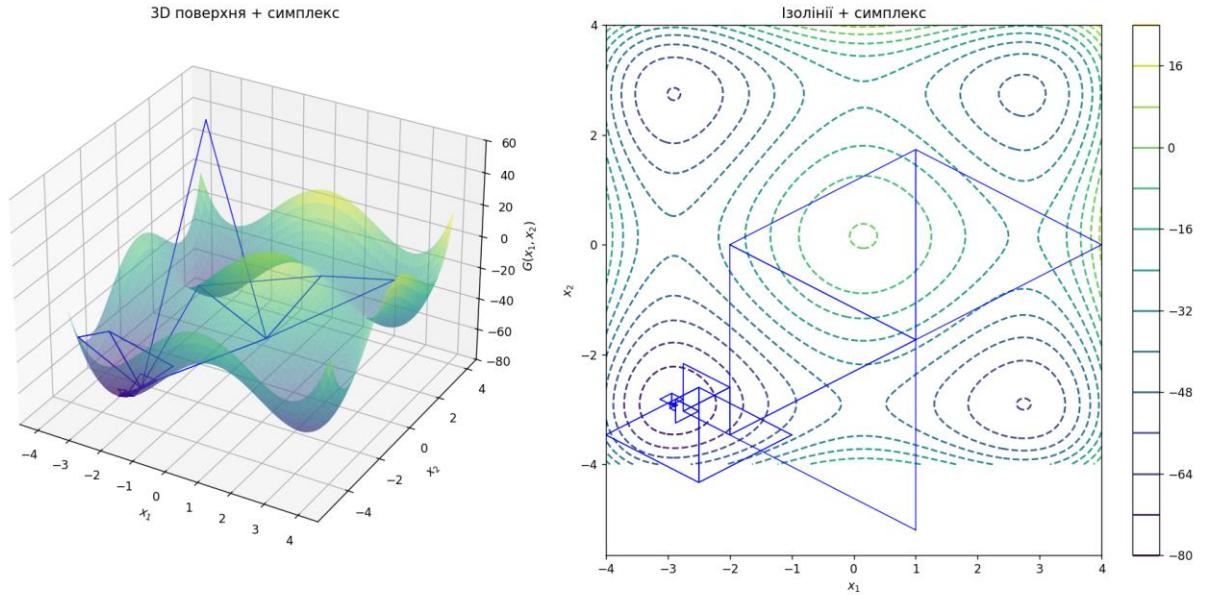
Початкова точка $(2, 0)$, $a = 2$, accuracy = 0.001:

Точка мінімуму: $([-2.904296875000001, -2.9059211791048147])$

Значення функції: -78.3322227277583

КОЦФ: 147

Час виконання: 0.001178 секунд



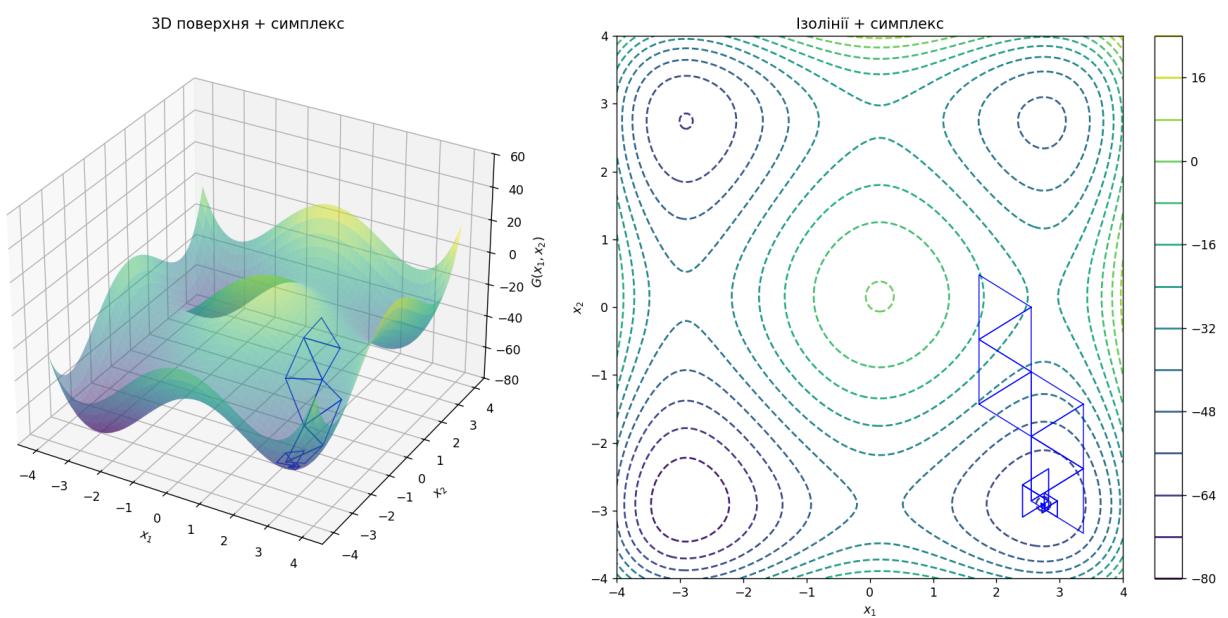
Початкова точка $(2, 0)$, $a = 0.55$, accuracy = 0.001:

Точка мінімуму: $([2.7433593749999976, -2.9099806731850553])$

Значення функції: -64.19471887963176

КОЦФ: 119

Час виконання: 0.001024 секунд



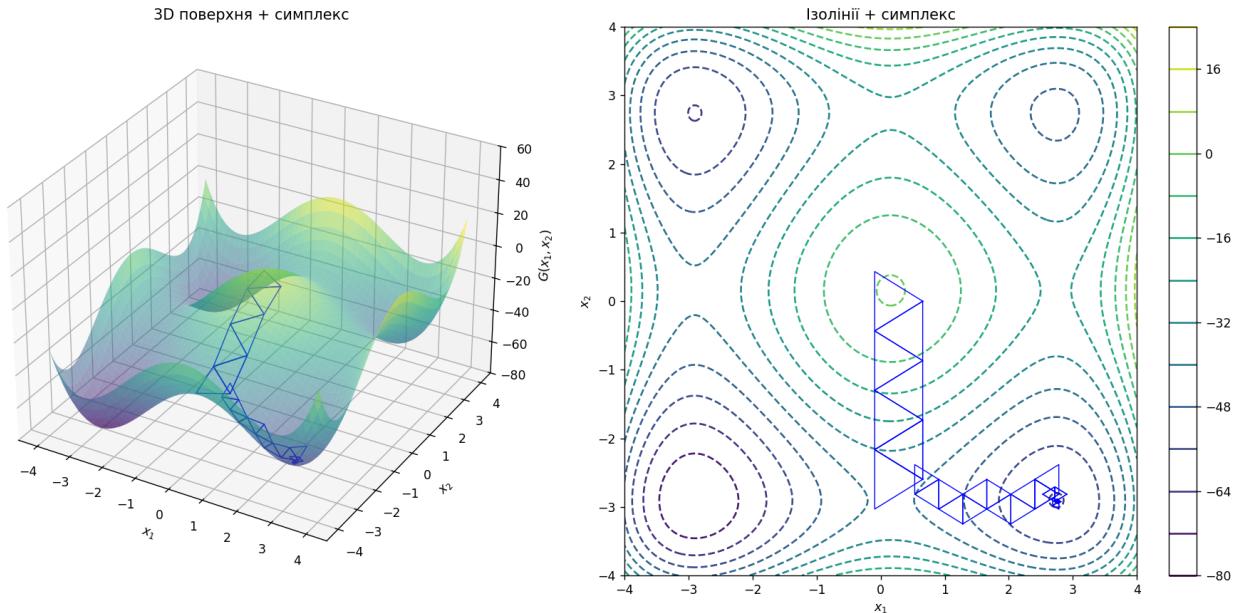
Початкова точка (0.156731, 0), a = 0.5, accuracy = 0.001:

Точка мінімуму: ([2.74657474999999, -2.9025382673712854])

Значення функції: -64.19559445871721

КОЦФ: 171

Час виконання: 0.001206 секунд



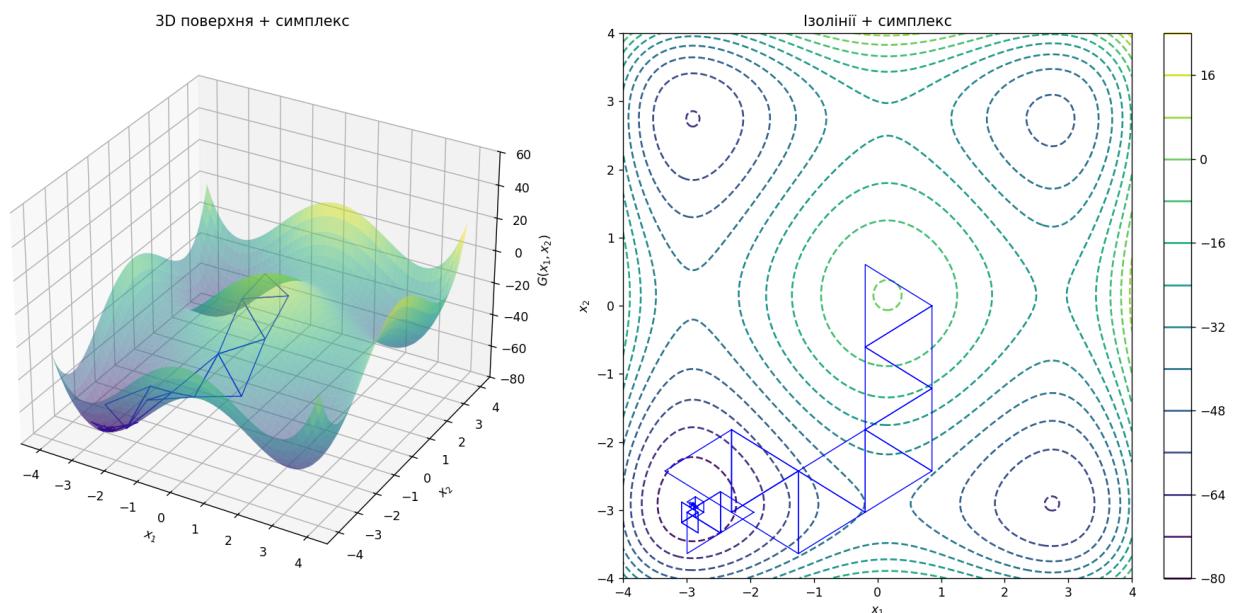
Початкова точка (0.156731, 0), a = 0.7, accuracy = 0.001:

Точка мінімуму:([-2.904401812500002, -2.905582887931462])

Значення функції: -78.33224574553539

КОЦФ: 159

Час виконання: 0.001170 секунд



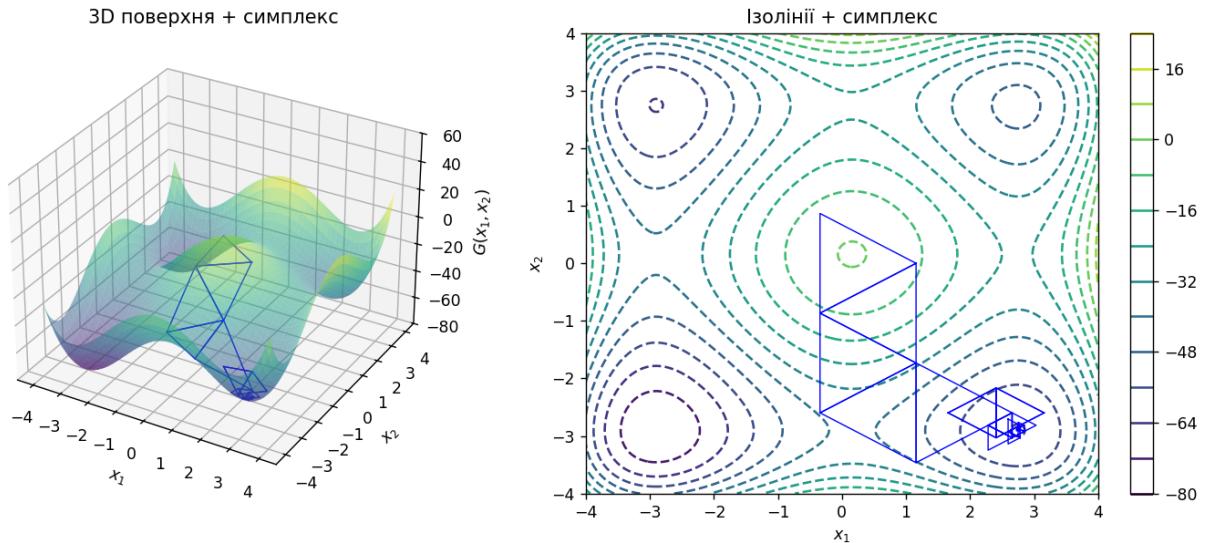
Початкова точка (0.156731, 0), a = 1, accuracy = 0.001:

Точка мінімуму: ([2.744621624999999, -2.9059211791048165])

Значення функції: -64.19544417795069

КОЦФ: 135

Час виконання: 0.001246 секунд



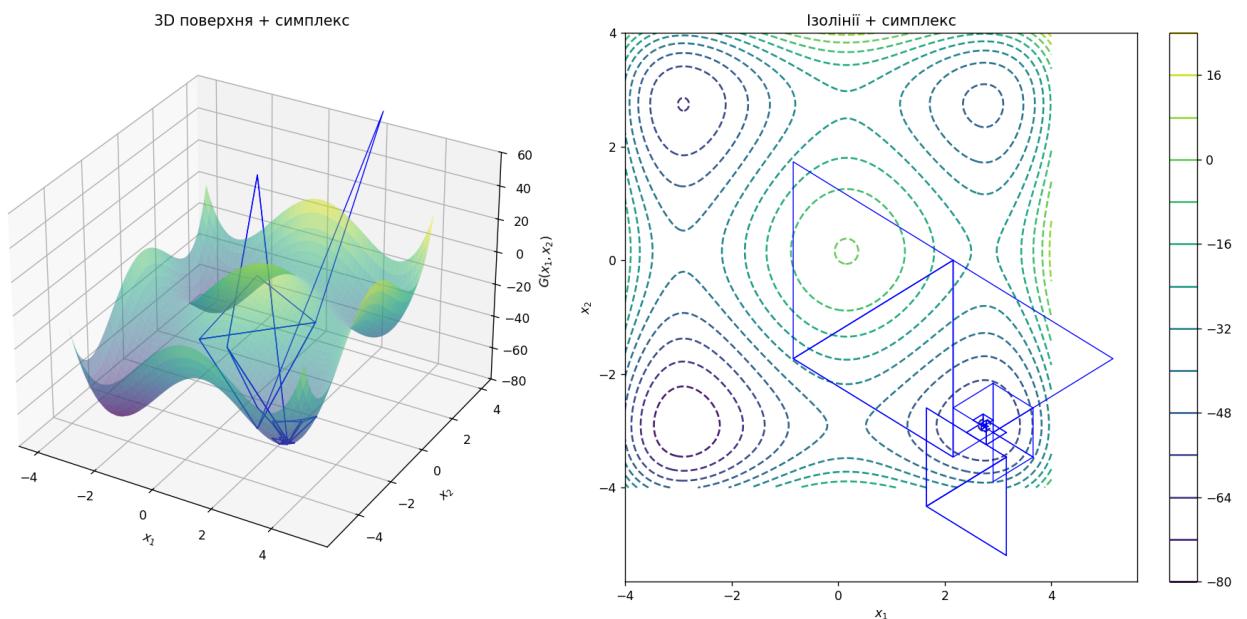
Початкова точка (0.156731, 0), a = 2, accuracy = 0.001:

Точка мінімуму: ([2.746574749999999, -2.902538267371283])

Значення функції: -64.19559445871721

КОЦФ: 151

Час виконання: 0.001128 секунд



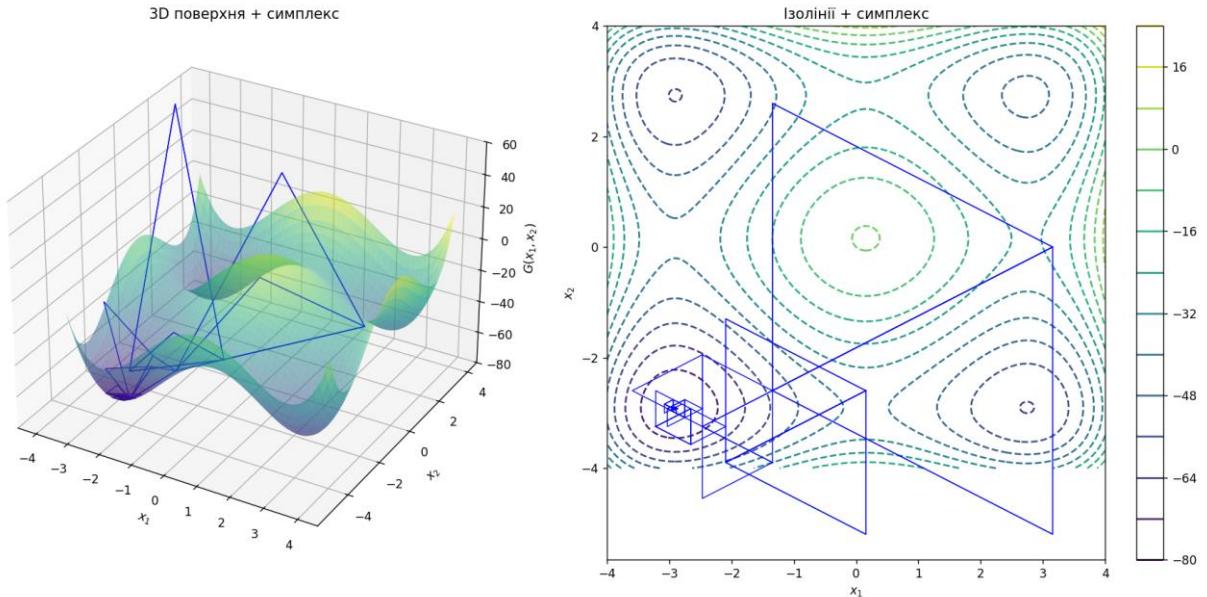
Початкова точка (0.156731, 0), a = 3, accuracy = 0.001:

Точка мінімуму: [-2.903327593750002, -2.9050754511714314])

Значення функції: -78.33228956501083

КОЦФ: 163

Час виконання: 0.001217 секунд



Основні результати роботи методу зібрані у таблиці:

Точка	a	Результати
(0, 0)	0.6	Точка мінімуму: [-2.903906250000001, -2.9025382673712814] Значення функції: -78.33231187202352 КОЦФ: 151 Час виконання: 0.001083 секунд
(0, 0)	0.5	Точка мінімуму: [-2.9101562500000036, -2.902538267371282] Значення функції: -78.33155428091305 КОЦФ: 135 Час виконання: 0.001012 секунд
(2, 0)	0.1	Точка мінімуму: [2.745312499999874, -2.903891432064693] Значення функції: -64.19557766570131 КОЦФ: 223 Час виконання: 0.001509 секунд
(2, 0)	0.5	Точка мінімуму: [2.75, -2.9025382673712836] Значення функції: -64.19544543928242 КОЦФ: 135 Час виконання: 0.001015 секунд
(2, 0)	0.55	Точка мінімуму: [2.7433593749999976, -2.909980673185055] Значення функції: -64.19471887963176 КОЦФ: 119 Час виконання: 0.001024 секунд

(2, 0)	1	Точка мінімуму: ([2.748046875000004, -2.905921179104815]) Значення функції: -64.19549108220028 КОЦФ: 143 Час виконання: 0.001007 секунд
(2, 0)	2	Точка мінімуму: ([-2.904296875000001, -2.9059211791048147]) Значення функції: -78.3322227277583 КОЦФ: 147 Час виконання: 0.001178 секунд
(0.156731, 0.156731)	0.1	Точка мінімуму: ([-2.9026440000000053, -2.904127536500624]) Значення функції: -78.3323116219608 КОЦФ: 315 Час виконання: 0.002055 секунд
(0.156731, 0.156731)	0.5	Точка мінімуму: ([-2.9018627500000003, -2.901421207113797]) Значення функції: -78.33220600165616 КОЦФ: 147 Час виконання: 0.001063 секунд
(0.156731, 0.156731)	0.6	Точка мінімуму: ([-2.899519, -2.8960085483401445]) Значення функції: -78.33107624093535 КОЦФ: 131 Час виконання: 0.000968 секунд
(0.156731, 0.156731)	1	Точка мінімуму: ([-2.9038158750000003, -2.904804118847333]) Значення функції: -78.33230212840348 КОЦФ: 147 Час виконання: 0.001074 секунд
(0.156731, 0.156731)	3	Точка мінімуму: ([-2.901862750000002, -2.9081870305808644]) Значення функції: -78.33190818210139 КОЦФ: 151 Час виконання: 0.001119 секунд
(0.156731, 0)	0.5	Точка мінімуму: ([2.746574749999999, -2.9025382673712854]) Значення функції: -64.19559445871721 КОЦФ: 171 Час виконання: 0.001206 секунд
(0.156731, 0)	0.7	Точка мінімуму: ([-2.904401812500002, -2.905582887931462]) Значення функції: -78.33224574553539 КОЦФ: 159 Час виконання: 0.001170 секунд
(0.156731, 0)	1	Точка мінімуму: ([2.744621624999999, -2.9059211791048165]) Значення функції: -64.19544417795069 КОЦФ: 135 Час виконання: 0.001246 секунд
(0.156731, 0)	2	Точка мінімуму: ([2.746574749999999, -2.902538267371283]) Значення функції: -64.19559445871721 КОЦФ: 151 Час виконання: 0.001128 секунд
(0.156731, 0)	3	Точка мінімуму: ([-2.903327593750002, -2.9050754511714314]) Значення функції: -78.33228956501083 КОЦФ: 163 Час виконання: 0.001217 секунд

Аналіз результатів:

Замале значення коефіцієнту a , яке визначає розмір симплексу, призводить до збільшення кількості обчислень, а також в одному випадку призвело до некоректного визначення точки мінімуму, через обмеження точності.

Також у результаті проведеного тестування, робимо висновок, що підібрати оптимальне значення коефіцієнта a дуже важко. Тому, що:

- Навіть невеликі зміни у коефіцієнті a призводять до значного збільшення кількості обчислень цільової функції.
- Для кожної точки оптимальне значення множника a відрізняється.
- Для однієї і тієї ж самої точки знайдений мінімум може відрізнятися в залежності від обраного кроку.

У таблиці наведені оптимальні значення a дляожної точки:

Точка	Оптимальне a
(0, 0)	0.5
(2, 0)	0.55
(0.156731, 0.156731)	0.6
(0.156731, 0)	0.7 (глобальний мінімум) 1 (локальний мінімум)

Для точки (0.156731, 0) виходить так, що від обраного коефіцієнту a залежить куди ми спустимося. Також такий ефект стався і з точкою (2, 0), яка розміщена близче до локального мінімуму, але через великий розмір симплексу метод спустився до глобального мінімуму.

Завелике значення коефіцієнту a не зможе зламати метод, але збільшить кількість обчислень цільової функції.

У підсумку маємо, що СХХ алгоритм є досить стійким та забезпечую хорошу збіжність з меншою КОЦФ, у порівнянні з методом Хука-Дживса, але

результат дуже сильно залежить від обраного коефіцієнту a , який обрати оптимально досить складно.

Реалізація модифікованого методу пошуку за симплексом (Недлера-Міда)

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import time
import math

function_calls = 0

def G(x1, x2):
    global function_calls
    function_calls += 1
    return 0.5 * ((np.pow(x1, 4) - 16 * np.pow(x1, 2) + 5 * x1) + (np.pow(x2, 4) - 16 * np.pow(x2, 2) + 5 * x2))

def build_regular_simplex_around_point(x0, alpha):
    x0 = np.array(x0, dtype=float)
    simplex = []

    for i in range(3):
        angle = 2 * math.pi * i / 3
        dx = alpha * math.cos(angle)
        dy = alpha * math.sin(angle)
        point = x0 + np.array([dx, dy])
        simplex.append(point.tolist())

    return simplex

def find_centre(simplex, remove_index):
    sum_coords = []
    for j in range(N + 1):
        if j != remove_index:
            sum_coords.append(simplex[j])

    centre = [0, 0]
    for i in range(N):
        for j in range(N):
            centre[i] += sum_coords[j][i]
        centre[i] /= N
    return centre

def reflect_point(simplex, index, coef):
    xc = find_centre(simplex, index)
    xprev = simplex[index]

    xnew = [0, 0]
    for i in range(N):
        xnew[i] = xc[i] + coef * (xc[i] - xprev[i])

    return xnew
```

```

def myFunc(e):
    return e[0]

def function_range(evaluated):
    values = [val for val, _ in evaluated]
    return max(values) - min(values)

def nedler_mead(x0, a, accuracy, max_iter):
    simplex_path = []
    simplex = build_regular_simplex_around_point(x0, a)
    simplex_path.append([p.copy() for p in simplex])

    iter = 0

    while iter < max_iter:
        evaluated = [(G(p[0]), p[1]), p] for p in simplex]
        evaluated.sort(key=myFunc)

        if function_range(evaluated) < accuracy:
            break

        best_point = evaluated[0][1]
        best_value = evaluated[0][0]
        worst_point = evaluated[2][1]
        worst_value = evaluated[2][0]
        worst_index = simplex.index(worst_point)

        # Три пробни точки
        new_points = [
            reflect_point(simplex, worst_index, 2),  # Розтягнення
            reflect_point(simplex, worst_index, 3),  # Сильне розтягнення
            reflect_point(simplex, worst_index, 1.5)  # Стиснення
        ]

        new_evaluated = [(G(p[0]), p[1]), p] for p in new_points]

        # Обираємо найкращу з нових точок
        best_new_value, best_new_point = min(new_evaluated, key=lambda x:
x[0])

        if best_new_value < worst_value:
            simplex[worst_index] = best_new_point
        else:
            for i in range(N + 1):
                if simplex[i] != best_point:
                    simplex[i] = [(simplex[i][j] + best_point[j]) / 2 for j
in range(N)]

        simplex_path.append([p.copy() for p in simplex])
        iter += 1

    evaluated = [(G(p[0]), p[1]), p] for p in simplex]
    evaluated.sort(key=myFunc)
    xmin = evaluated[0][1]
    gmin = evaluated[0][0]

    return xmin, gmin, simplex_path

```

```

x_start = [0.156731, 0.156731]
a = 1
accuracy = 0.001
max_iter = 100
N = 2

xmin = 0
gmin = 0

start = time.perf_counter_ns()
xmin, gmin, simplex_path = nedler_mead(x_start, a, accuracy, max_iter)
end = time.perf_counter_ns()
elapsed = end - start

print(f'Точка мінімуму: ({xmin})\nЗначення функції: {gmin}')
print(f'КОЦФ: {function_calls}')
print(f'Час виконання: {elapsed / 1_000_000_000:.6f} секунд')

# Візуалізація

x1_vals = np.linspace(-4, 4, 200)
x2_vals = np.linspace(-4, 4, 200)
X1, X2 = np.meshgrid(x1_vals, x2_vals)
Z = G(X1, X2)

fig = plt.figure(figsize=(12, 5))

# 3D графік
ax1 = fig.add_subplot(1, 2, 1, projection='3d')
ax1.plot_surface(X1, X2, Z, cmap=cm.viridis, alpha=0.5)

for triangle in simplex_path:
    triangle_closed = triangle + [triangle[0]]
    xs = [p[0] for p in triangle_closed]
    ys = [p[1] for p in triangle_closed]
    zs = [G(p[0], p[1]) for p in triangle_closed]
    ax1.plot(xs, ys, zs, color='blue', lw=0.8)

ax1.set_zlim(-80, 60)
ax1.set_xlabel('$x_1$')
ax1.set_ylabel('$x_2$')
ax1.set_zlabel('$G(x_1, x_2)$')
ax1.set_title('3D поверхня + симплекс')

# Ізолінії
ax2 = fig.add_subplot(1, 2, 2)
contour = ax2.contour(X1, X2, Z, cmap=cm.viridis, levels=12,
linestyles='dashed')

for triangle in simplex_path:
    triangle_closed = triangle + [triangle[0]]
    xs = [p[0] for p in triangle_closed]
    ys = [p[1] for p in triangle_closed]
    ax2.plot(xs, ys, color='blue', linewidth=0.8, label='_simplex')

plt.colorbar(contour, ax=ax2)
ax2.set_xlabel('$x_1$')
ax2.set_ylabel('$x_2$')
ax2.set_title('Ізолінії + симплекс')

plt.tight_layout()
plt.show()

```

Тестування:

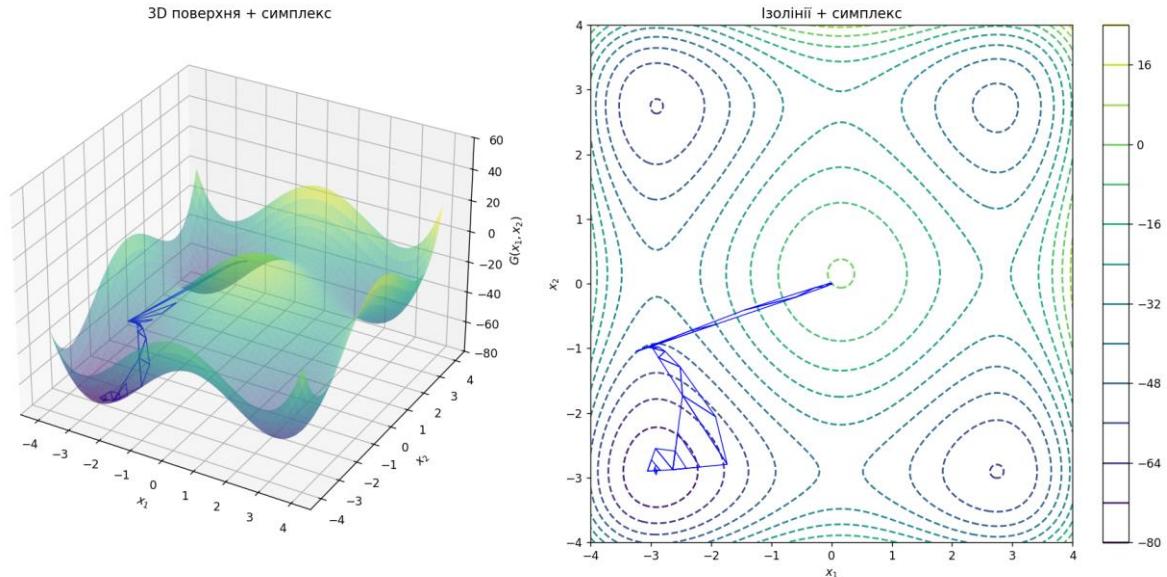
Початкова точка $(0, 0)$, $a = 0.01$, accuracy = 0.001:

Точка мінімуму: $([-2.9006476989408725, -2.9057928368970813])$

Значення функції: -78.33209920113737

КОЦФ: 228

Час виконання: 0.001479 секунд



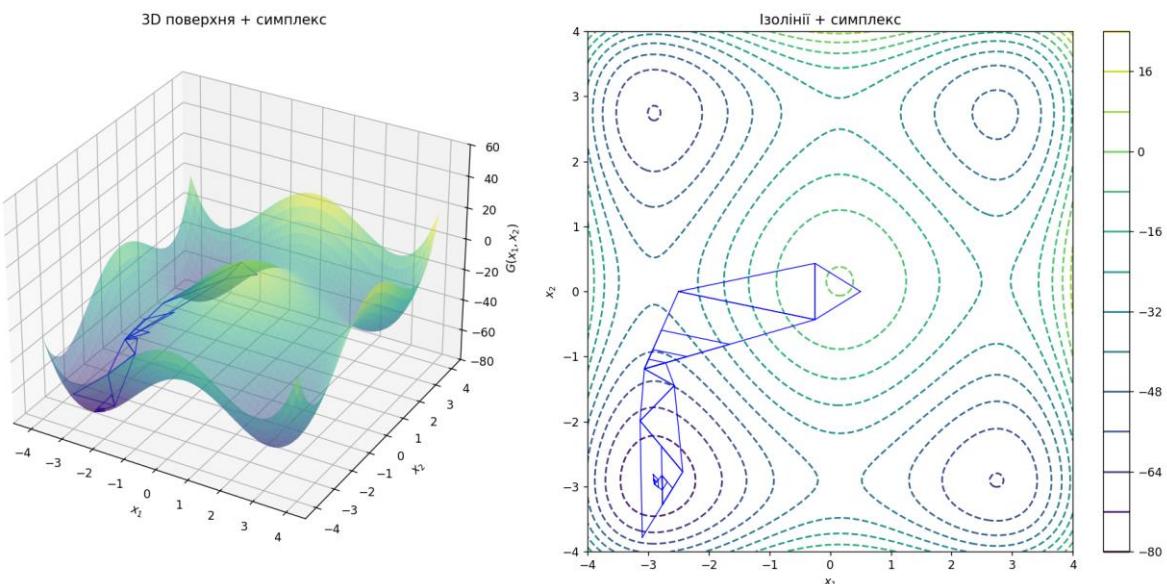
Початкова точка $(0, 0)$, $a = 0.5$, accuracy = 0.001:

Точка мінімуму: $([-2.9042861461639404, -2.9015525483395512])$

Значення функції: -78.33225377768022

КОЦФ: 126

Час виконання: 0.000901 секунд



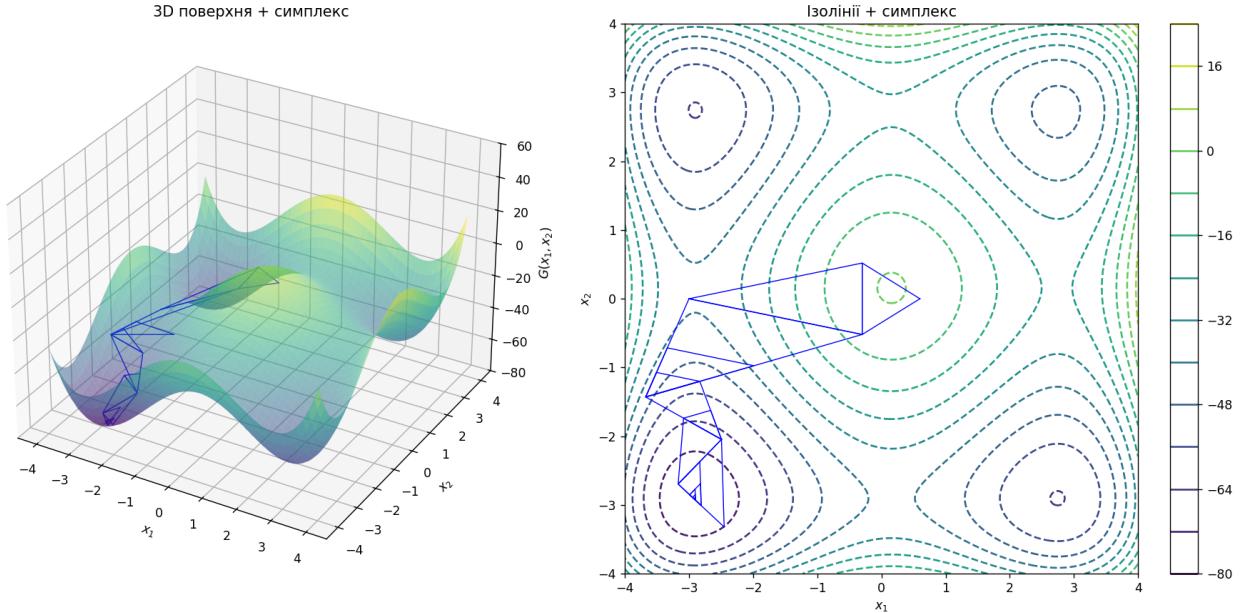
Початкова точка $(0, 0)$, $a = 0.6$, accuracy = 0.001:

Точка мінімуму: $([-2.9060668945312504, -2.905138880766436])$

Значення функції: -78.33217582158042

КОЦФ: 96

Час виконання: 0.000687 секунд



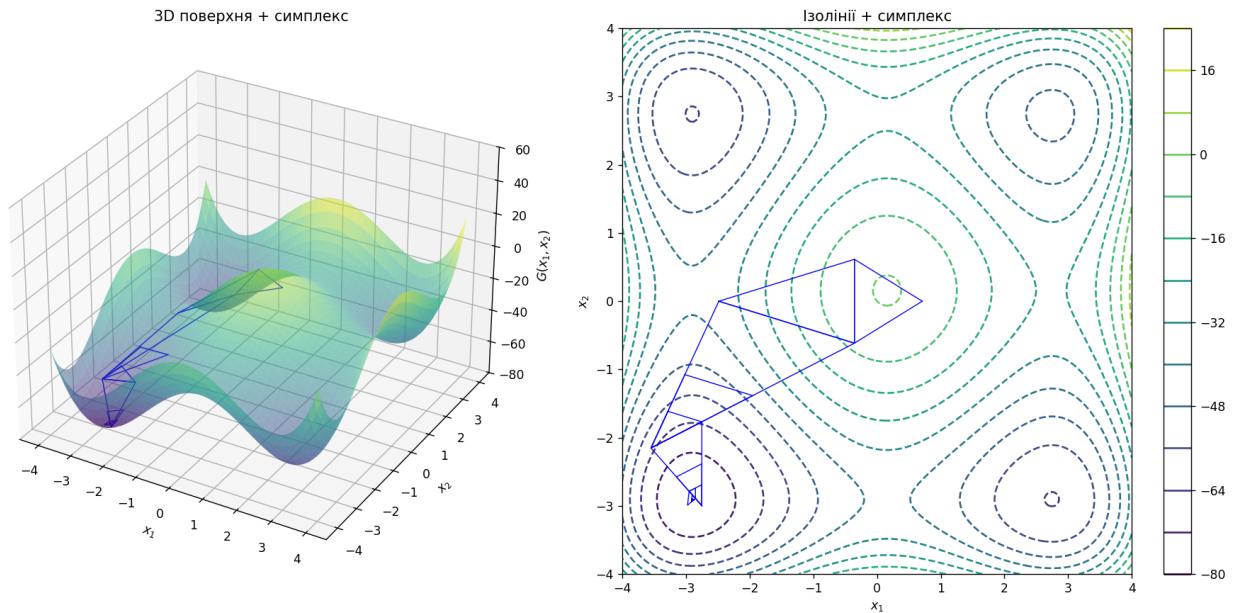
Початкова точка $(0, 0)$, $a = 0.71$, accuracy = 0.001:

Точка мінімуму: $([-2.902575683593751, -2.9094119211498537])$

Значення функції: -78.33171693585675

КОЦФ: 84

Час виконання: 0.000636 секунд



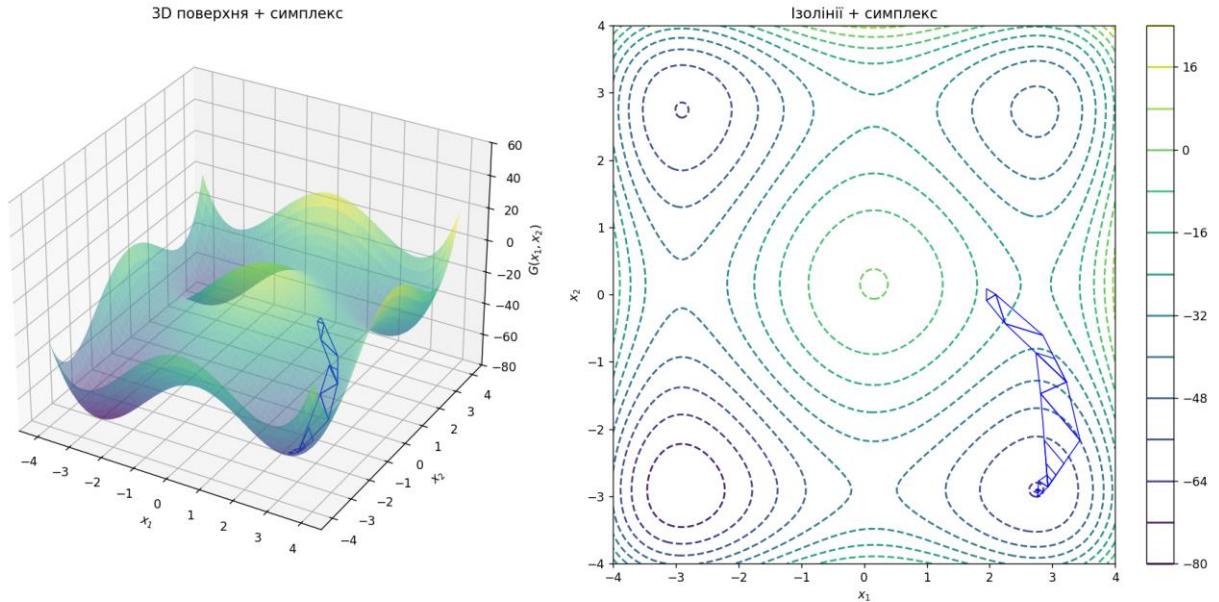
Початкова точка (2, 0), a = 0.1, accuracy = 0.001:

Точка мінімуму: ([2.744988799095152, -2.903785716073024])

Значення функції: -64.19556314073694

КОЦФ: 126

Час виконання: 0.000909 секунд



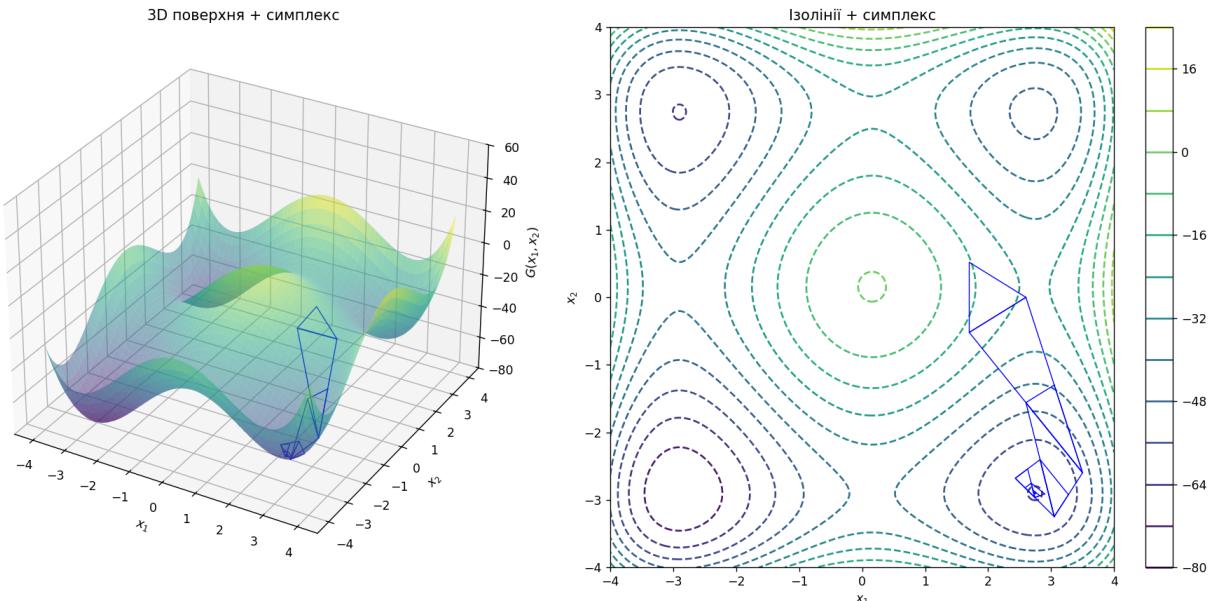
Початкова точка (2, 0), a = 0.6, accuracy = 0.001:

Точка мінімуму: ([2.7467773437499994, -2.9043142960313872])

Значення функції: -64.19560181943054

КОЦФ: 84

Час виконання: 0.000622 секунд



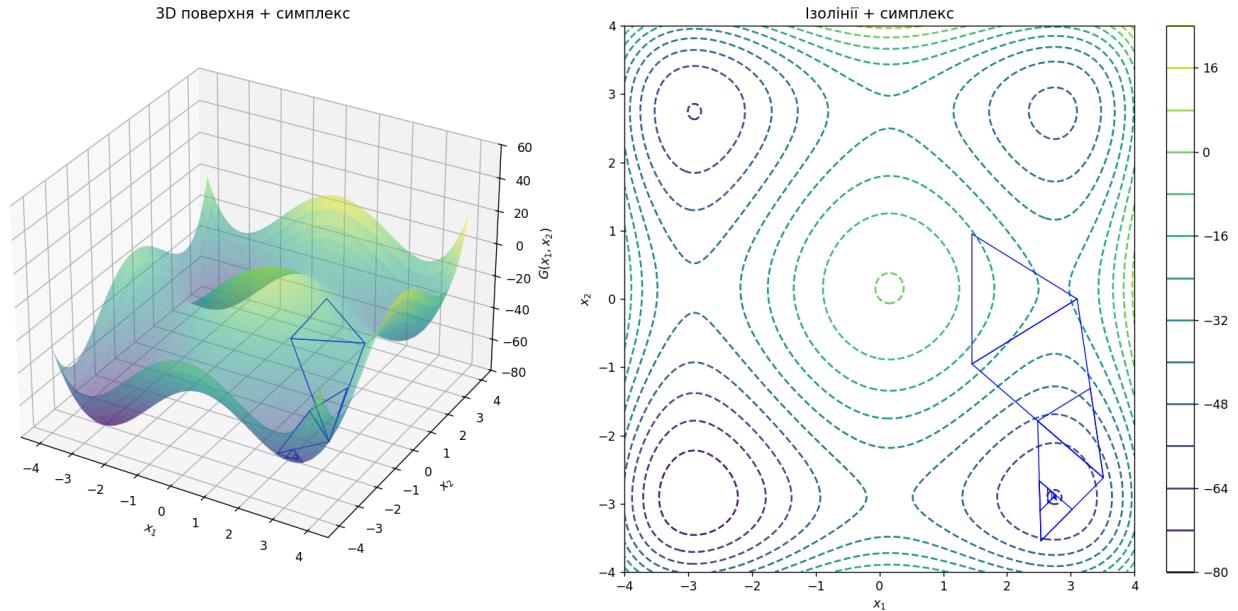
Початкова точка $(2, 0)$, $a = 1.1$, accuracy = 0.001:

Точка мінімуму: $([2.7444000244140616, -2.9131785819331606])$

Значення функції: -64.19391431841527

КОЦФ: 72

Час виконання: 0.000538 секунд



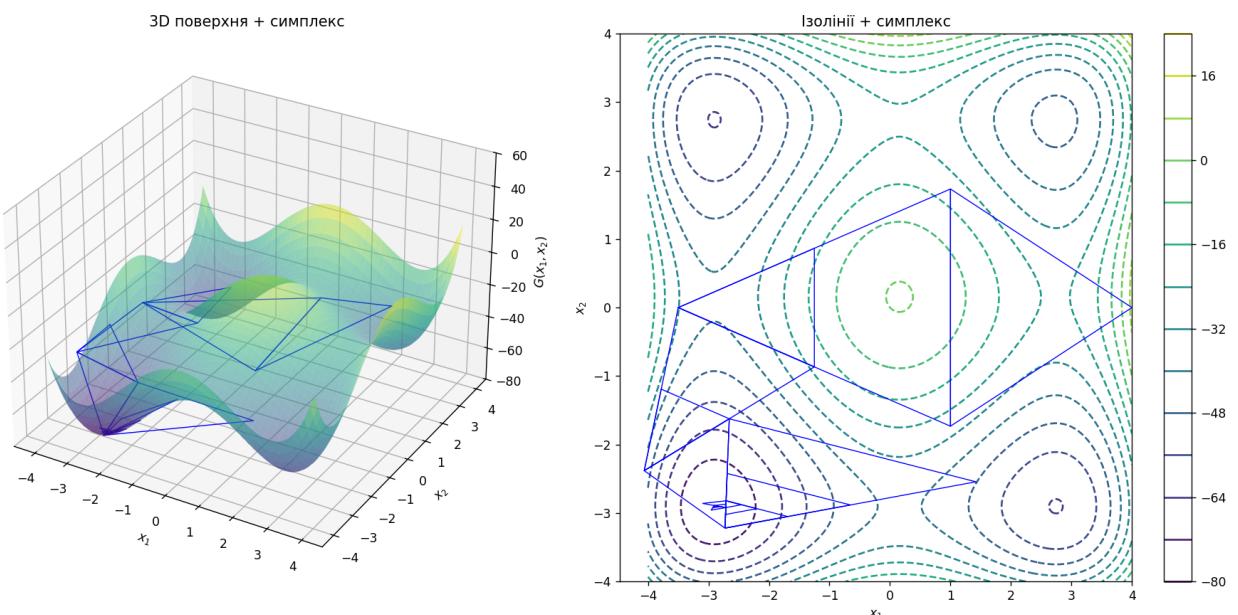
Початкова точка $(2, 0)$, $a = 2$, accuracy = 0.001:

Точка мінімуму: $([-2.9114933013916016, -2.8988844584090865])$

Значення функції: -78.3308598250895

КОЦФ: 108

Час виконання: 0.000769 секунд



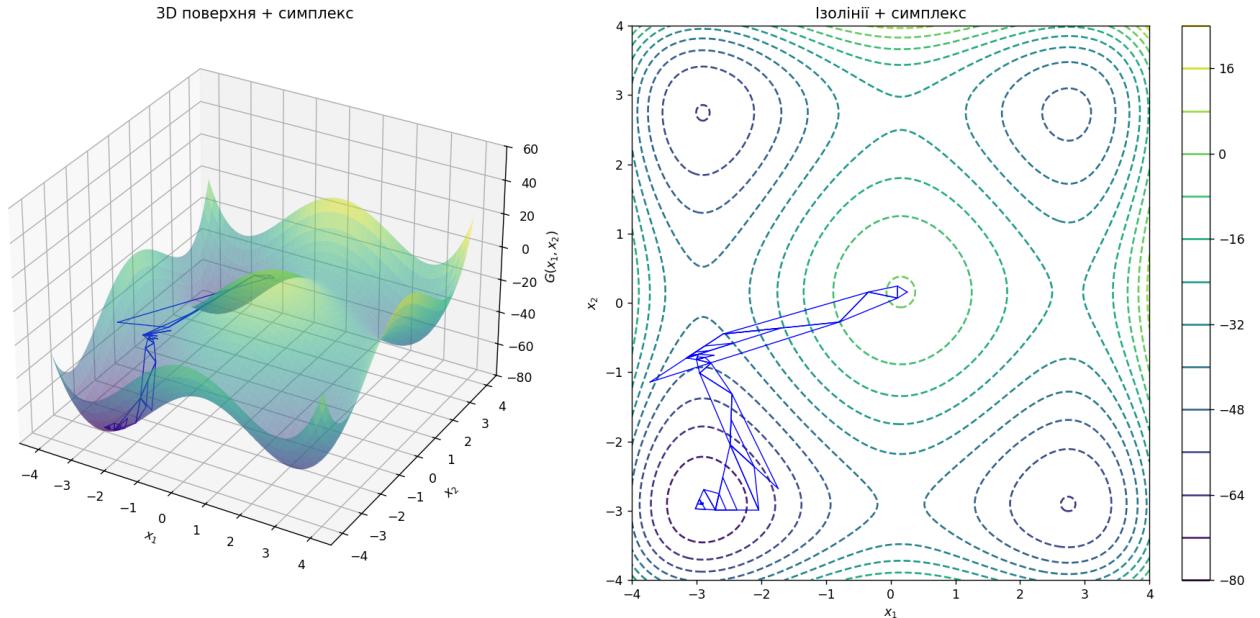
Початкова точка (0.156731, 0.156731), $a = 0.1$, accuracy = 0.0001:

Точка мінімуму: $([-2.9036904370251087, -2.902452722643485])$

Значення функції: -78.33231077422857

КОЦФ: 204

Час виконання: 0.001387 секунд



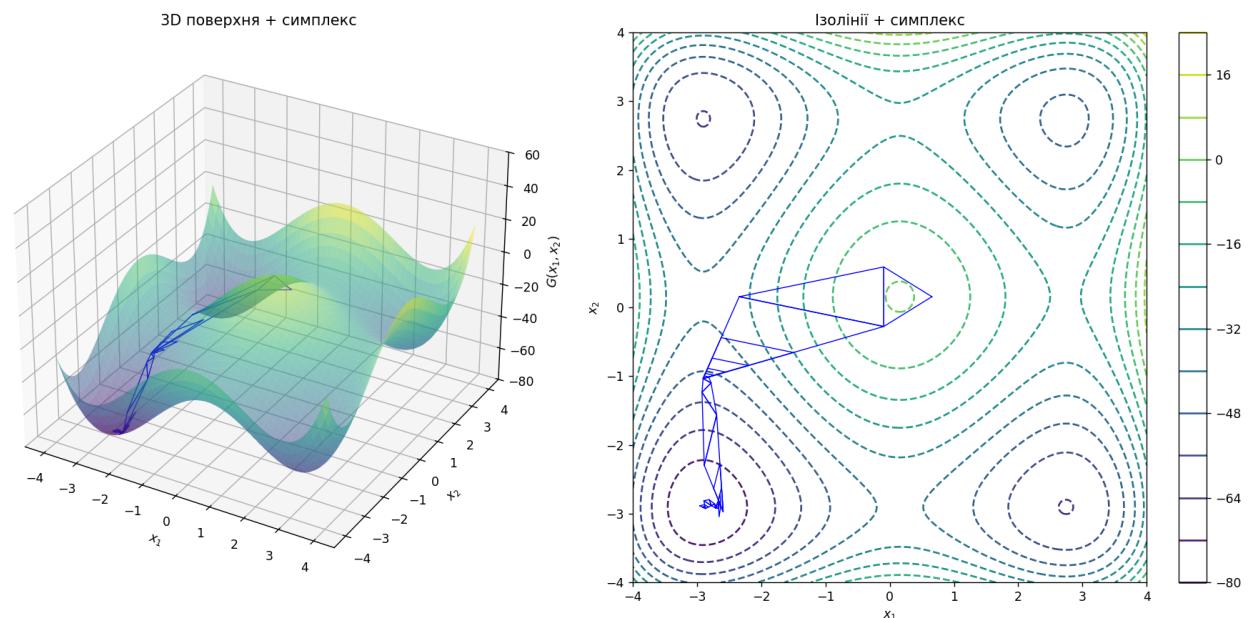
Початкова точка (0.156731, 0.156731), $a = 0.5$, accuracy = 0.001:

Точка мінімуму: $([-2.9045302091936183, -2.902897513612839])$

Значення функції: -78.33230723790747

КОЦФ: 198

Час виконання: 0.001358 секунд



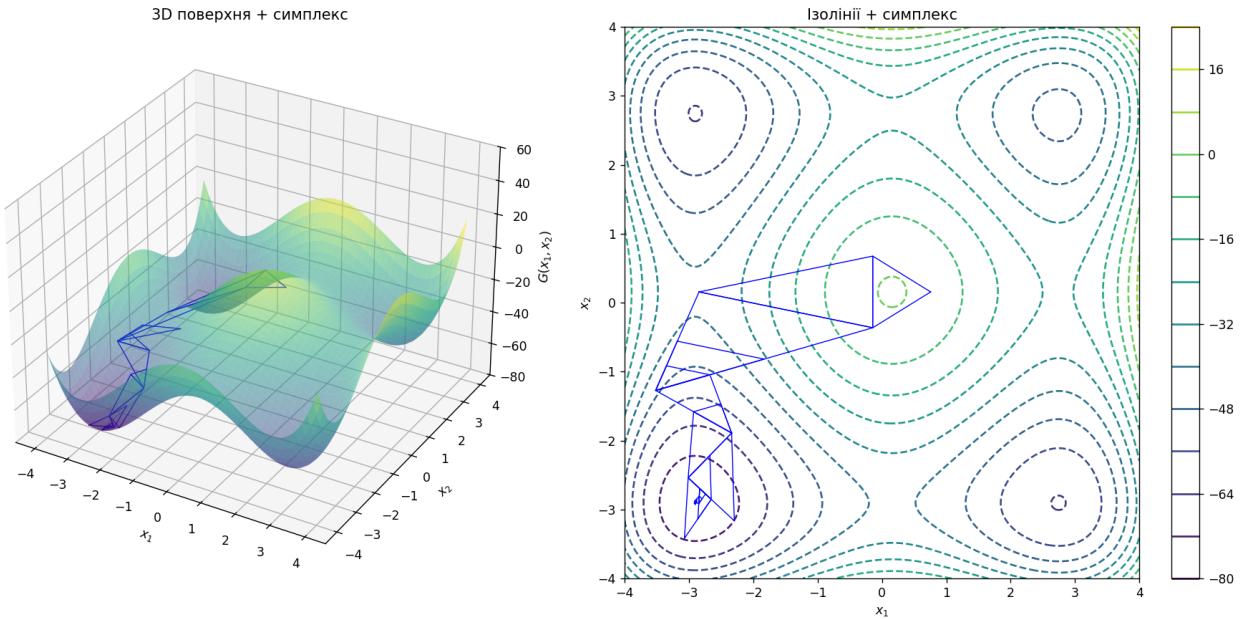
Початкова точка (0.156731, 0.156731), a = 0.6, accuracy = 0.001:

Точка мінімуму:([-2.9029374753894794, -2.9032805593841355])

Значення функції: -78.33232414433603

КОЦФ: 144

Час виконання: 0.000994 секунд



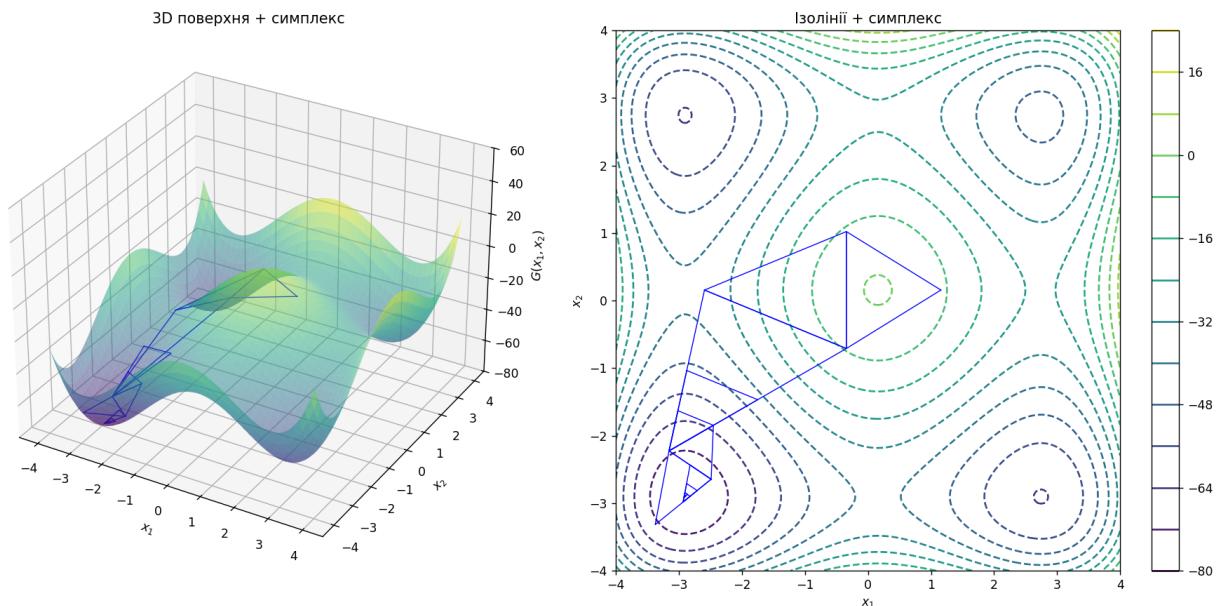
Початкова точка (0.156731, 0.156731), a = 1, accuracy = 0.001:

Точка мінімуму:([-2.9040447568359387, -2.902108361059673])

Значення функції: -78.33229176772348

КОЦФ: 102

Час виконання: 0.000717 секунд



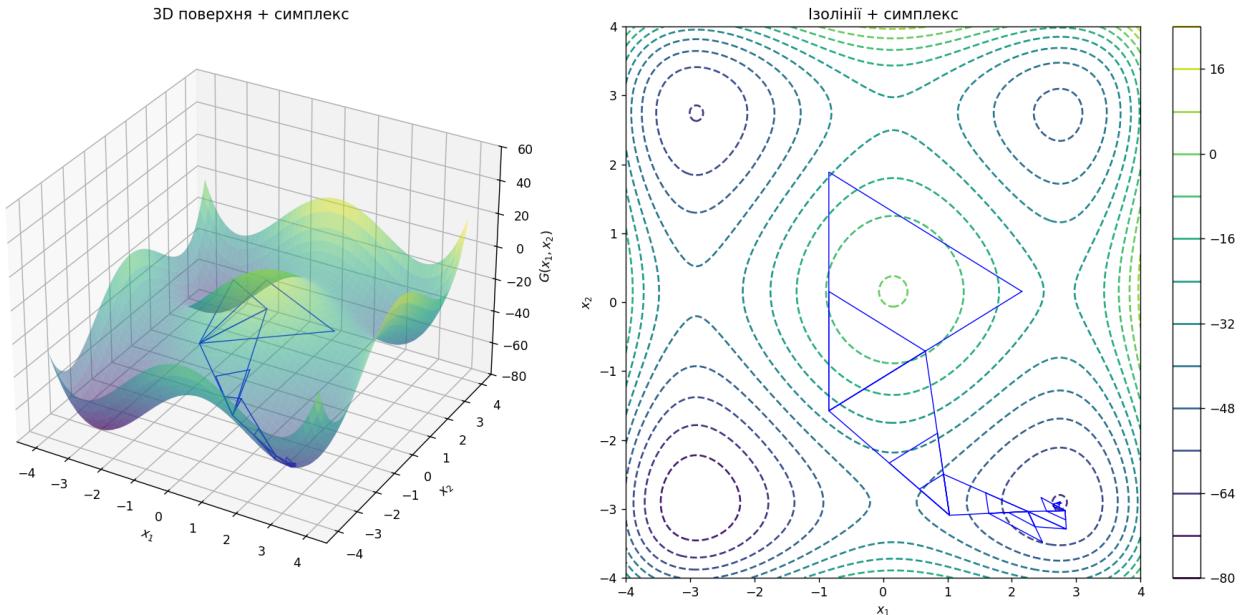
Початкова точка (0.156731, 0.156731), a = 2, accuracy = 0.001:

Точка мінімуму: ([2.747848829084395, -2.9053025016108087])

Значення функції: -64.19554222746787

КОЦФ: 138

Час виконання: 0.000971 секунд



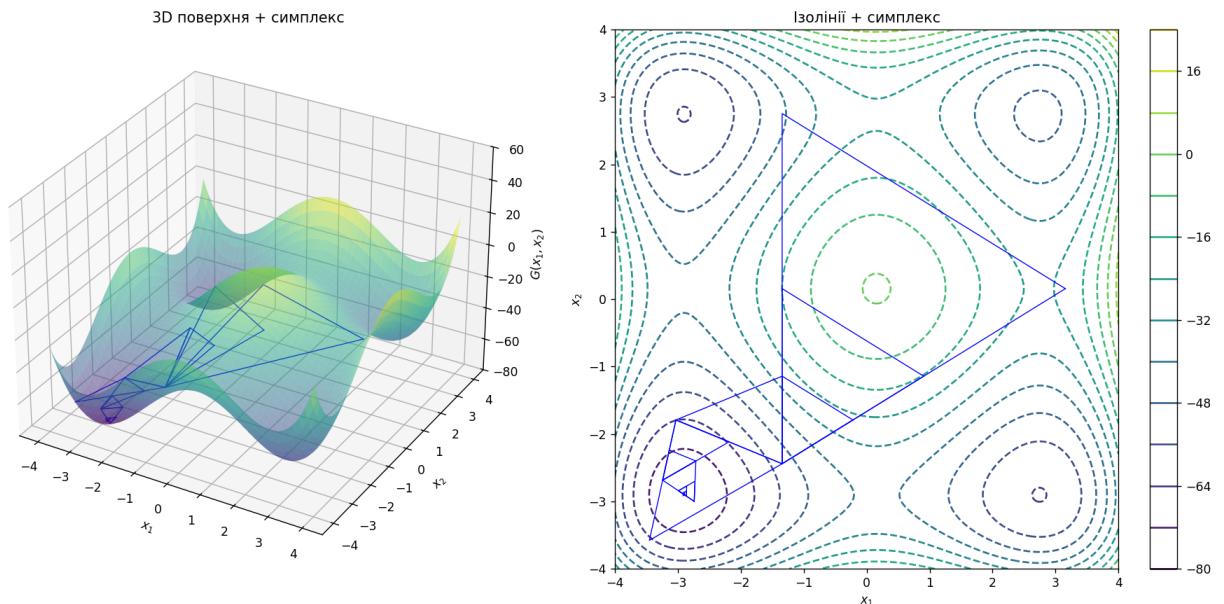
Початкова точка (0.156731, 0.156731), a = 3, accuracy = 0.001:

Точка мінімуму: ([-2.9024864530029317, -2.9051543030697475])

Значення функції: -78.33226701825076

КОЦФ: 108

Час виконання: 0.000772 секунд



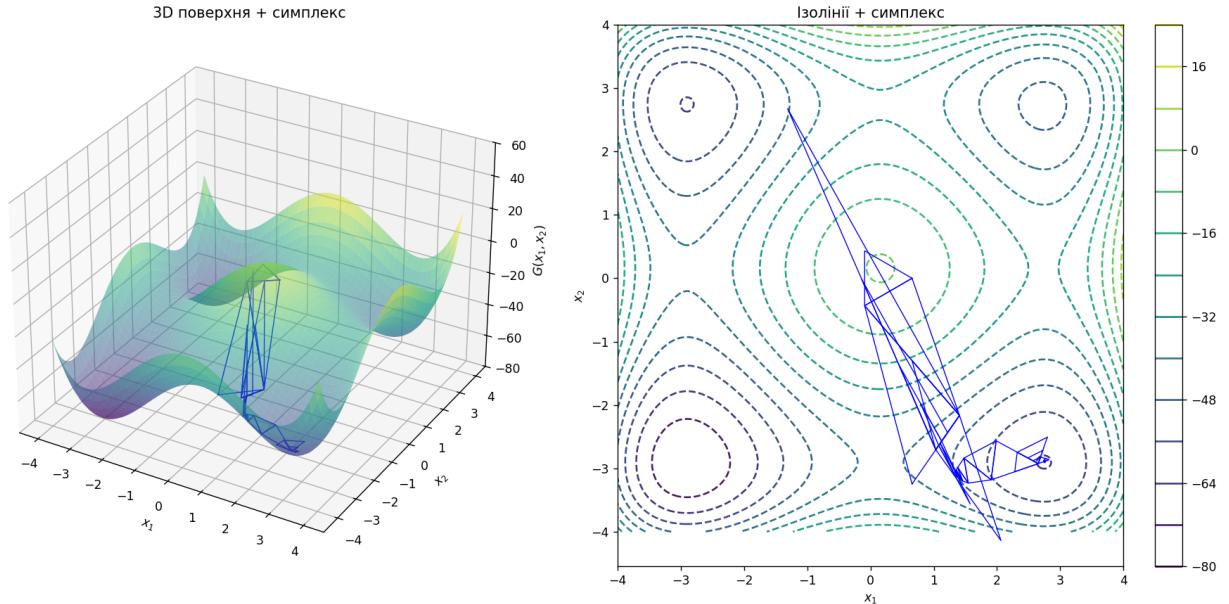
Початкова точка (0.156731, 0), a = 0.5, accuracy = 0.001:

Точка мінімуму: ([2.746427091486984, -2.9056126697643103])

Значення функції: -64.19553552928164

КОЦФ: 204

Час виконання: 0.001357 секунд



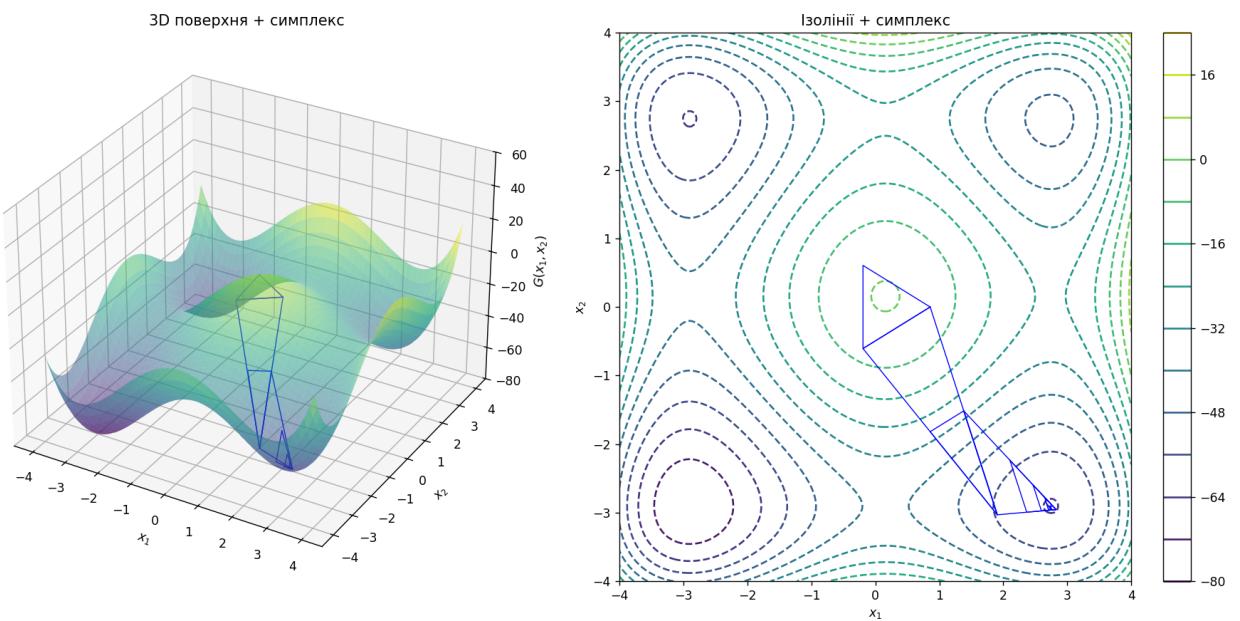
Початкова точка (0.156731, 0), a = 0.7, accuracy = 0.001:

Точка мінімуму: ([2.7485126162109372, -2.9076179207711648])

Значення функції: -64.19528075883915

КОЦФ: 84

Час виконання: 0.000593 секунд



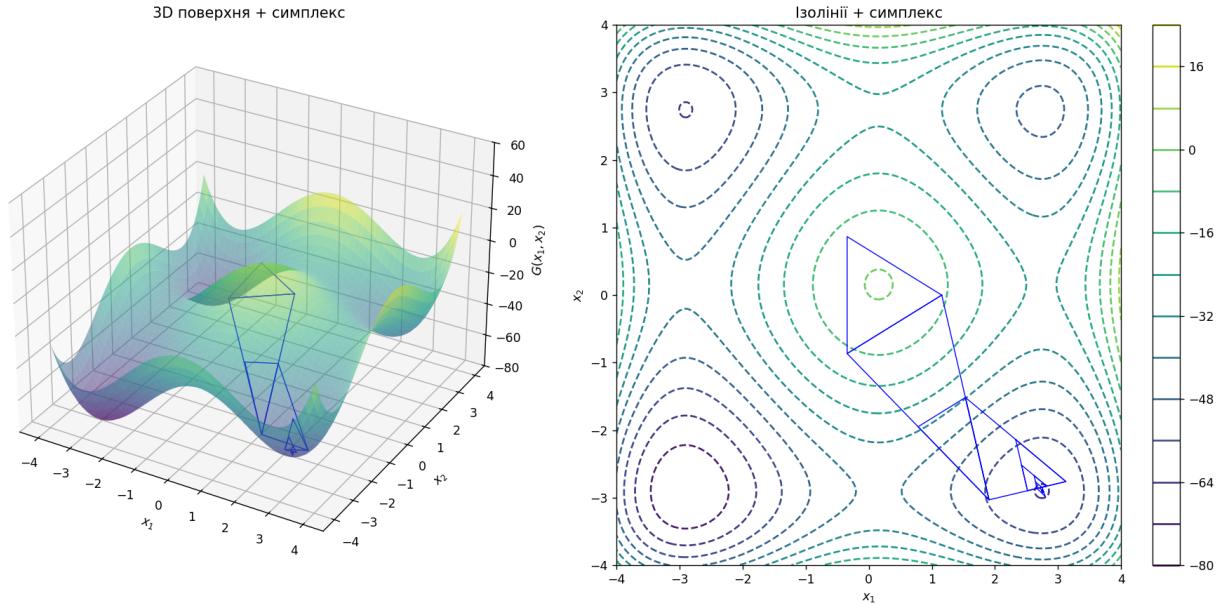
Початкова точка (0.156731, 0), a = 1, accuracy = 0.001:

Точка мінімуму: ([2.7477687502441395, -2.9040513275021023])

Значення функції: -64.19559407016023

КОДФ: 90

Час виконання: 0.000658 секунд



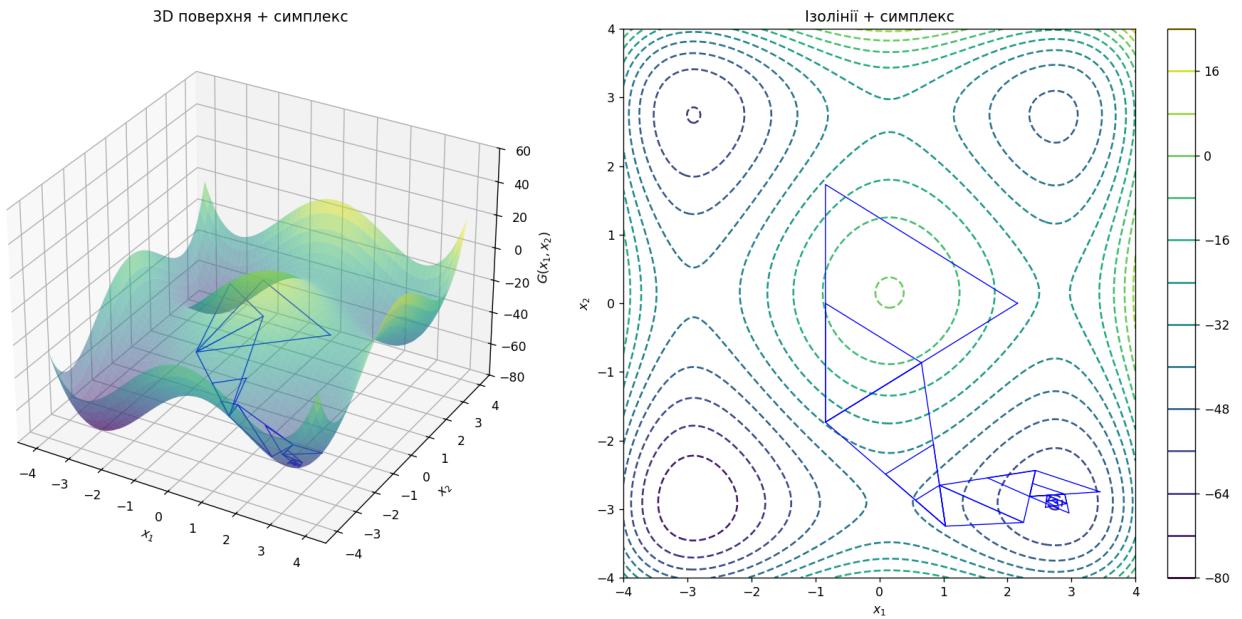
Початкова точка (0.156731, 0), a = 2, accuracy = 0.001:

Точка мінімуму: ([2.7495320940551746, -2.902556437307351])

Значення функції: -64.1954867100971

КОДФ: 120

Час виконання: 0.000851 секунд



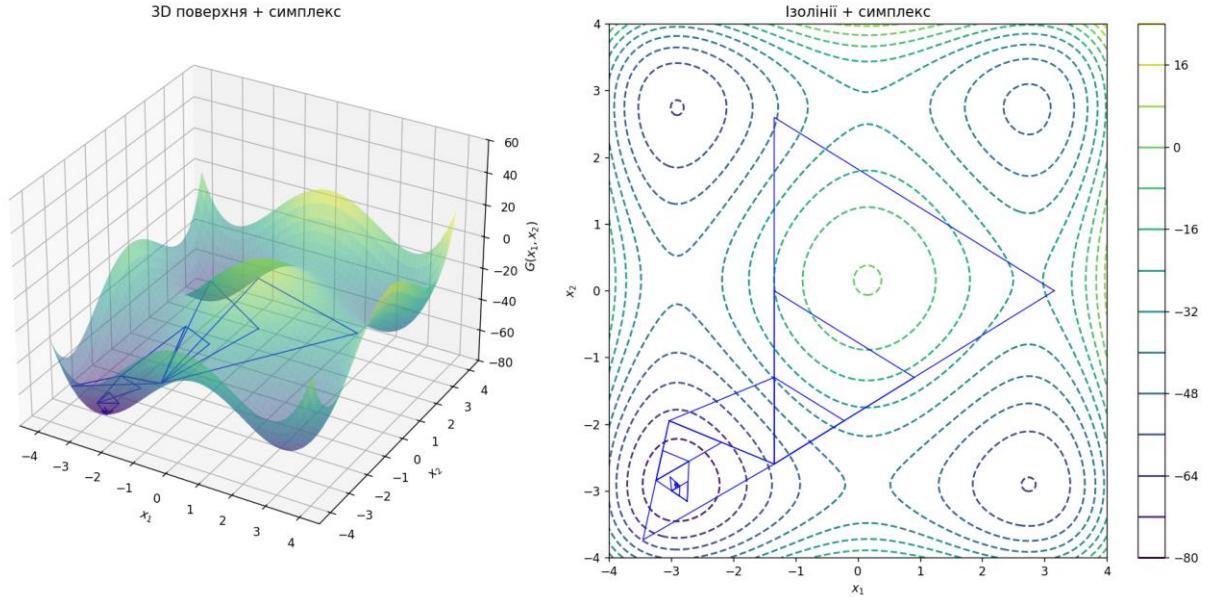
Початкова точка (0.156731, 0), a = 3, accuracy = 0.001:

Точка мінімуму:([-2.903748164123537, -2.902151743276728])

Значення функції: -78.33229759082322

КОЦФ: 102

Час виконання: 0.000737 секунд



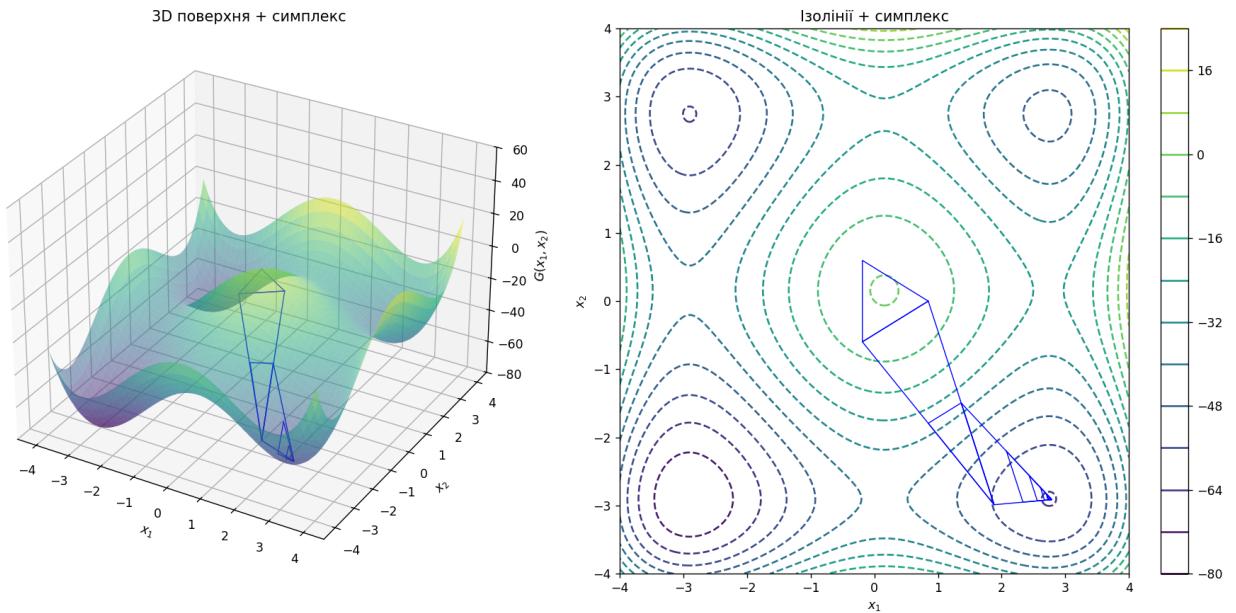
Початкова точка (0.156731, 0), a = 0.69, accuracy = 0.001:

Точка мінімуму: ([2.740861859374998, -2.9049232201434227])

Значення функції: -64.19506359851206

КОЦФ: 72

Час виконання: 0.000532 секунд



Аналіз результатів:

Точка	a	Результати
(0, 0)	0.5	Точка мінімуму: ([-2.9042861461639404, -2.901552548339551]) Значення функції: -78.33225377768022 КОЦФ: 126 Час виконання: 0.000901 секунд
(0, 0)	0.6	Точка мінімуму: ([-2.9060668945312504, -2.905138880766436]) Значення функції: -78.33217582158042 КОЦФ: 96 Час виконання: 0.000687 секунд
(0, 0)	0.71	Точка мінімуму: ([-2.902575683593751, -2.9094119211498537]) Значення функції: -78.33171693585675 КОЦФ: 84 Час виконання: 0.000636 секунд
(2, 0)	0.1	Точка мінімуму: ([2.744988799095152, -2.903785716073024]) Значення функції: -64.19556314073694 КОЦФ: 126 Час виконання: 0.000909 секунд
(2, 0)	0.5	Точка мінімуму: ([-2.9045302091936183, -2.902897513612839]) Значення функції: -78.33230723790747 КОЦФ: 198 Час виконання: 0.001358 секунд
(2, 0)	0.6	Точка мінімуму: ([2.7467773437499994, -2.904314296031387]) Значення функції: -64.19560181943054 КОЦФ: 84 Час виконання: 0.000622 секунд
(2, 0)	1.1	Точка мінімуму: ([2.7444000244140616, -2.91317858193316]) Значення функції: -64.19391431841527 КОЦФ: 72 Час виконання: 0.000538 секунд
(2, 0)	2	Точка мінімуму: ([-2.9114933013916016, -2.898884458409086]) Значення функції: -78.3308598250895 КОЦФ: 108 Час виконання: 0.000769 секунд
(0.156731, 0.156731)	0.1	Точка мінімуму: ([-2.9036904370251087, -2.902452722643485]) Значення функції: -78.33231077422857 КОЦФ: 204 Час виконання: 0.001387 секунд
(0.156731, 0.156731)	0.5	Точка мінімуму: ([-2.9045302091936183, -2.902897513612839]) Значення функції: -78.33230723790747 КОЦФ: 198 Час виконання: 0.001358 секунд
(0.156731, 0.156731)	0.6	Точка мінімуму: ([-2.9029374753894794, -2.903280559384135]) Значення функції: -78.33232414433603 КОЦФ: 144 Час виконання: 0.000994 секунд

(0.156731, 0.156731)	1	Точка мінімуму: ([-2.9040447568359387, -2.902108361059673]) Значення функції: -78.33229176772348 КОЦФ: 102 Час виконання: 0.000717 секунд
(0.156731, 0.156731)	2	Точка мінімуму: ([2.747848829084395, -2.9053025016108087]) Значення функції: -64.19554222746787 КОЦФ: 138 Час виконання: 0.000971 секунд
(0.156731, 0.156731)	3	Точка мінімуму: ([2.747848829084395, -2.9053025016108087]) Значення функції: -64.19554222746787 КОЦФ: 138 Час виконання: 0.000971 секунд
(0.156731, 0)	0.5	Точка мінімуму: ([2.746427091486984, -2.9056126697643103]) Значення функції: -64.19553552928164 КОЦФ: 204 Час виконання: 0.001357 секунд
(0.156731, 0)	0.69	Точка мінімуму: ([2.740861859374998, -2.9049232201434227]) Значення функції: -64.19506359851206 КОЦФ: 72 Час виконання: 0.000532 секунд
(0.156731, 0)	0.7	Точка мінімуму: ([2.7485126162109372, -2.907617920771164]) Значення функції: -64.19528075883915 КОЦФ: 84 Час виконання: 0.000593 секунд
(0.156731, 0)	1	Точка мінімуму: ([2.7477687502441395, -2.904051327502102]) Значення функції: -64.19559407016023 КОЦФ: 90 Час виконання: 0.000658 секунд
(0.156731, 0)	2	Точка мінімуму: ([2.7495320940551746, -2.902556437307351]) Значення функції: -64.1954867100971 КОЦФ: 120 Час виконання: 0.000851 секунд
(0.156731, 0)	3	Точка мінімуму: ([-2.903748164123537, -2.902151743276728]) Значення функції: -78.33229759082322 КОЦФ: 102 Час виконання: 0.000737 секунд

Аналіз результатів:

Результати методу пошуку за симплексом Недлера-Міда схожі на результати СХХ алгоритму. З однієї стартової точки можуть бути знайдені різні точки мінімуму, в залежності від обраного коефіцієнту a . При дуже малих та дуже великих значеннях a КОЦФ значно збільшується.

Загалом алгоритм має високу стійкість та досить швидко знаходить мінімум функції.

Також у коді можна змінити значення коефіцієнту λ у формулі:

$$x = x^{(j)} + \lambda(x_c - x^{(j)})$$

Для тестування було обрано такі значення λ : $\lambda_1 = 2, \lambda_2 = 3, \lambda_3 = 1.5$.
Але можна взяти інший набір та подивитися чи покращатиметься результат.
Візьмімо новий набір λ : $\lambda_1 = 1, \lambda_2 = 2, \lambda_3 = 0.5$. та подивимося чи покращиться збіжність для різних точок:

Точка	$\lambda_1 = 2, \lambda_2 = 3, \lambda_3 = 1.5$	$\lambda_1 = 1, \lambda_2 = 2, \lambda_3 = 0.5$
(0.156731, 0)	Точка мінімуму: ([2.7408618593749, -2.904923220143422]) Значення функції: -64.195063598512 КОЦФ: 72 Час виконання: 0.000532 секунд	Точка мінімуму: ([2.7436098146972, -2.9010754223365]) Значення функції: -64.195358900199 КОЦФ: 114 Час виконання: 0.000798 секунд
(0.156731, 0.156731)	Точка мінімуму: ([-2.9040447568359, -2.902108361059]) Значення функції: -78.332291767723 КОЦФ: 102 Час виконання: 0.000717 секунд	Точка мінімуму: ([-2.898953392526, -2.902820814834]) Значення функції: -78.331960357164 КОЦФ: 120 Час виконання: 0.000838 секунд

Попередній набір λ виявився кращим, спробуємо з ще одним набором:

$$\lambda_1 = 3, \lambda_2 = 0.25, \lambda_3 = -1.$$

Точка	$\lambda_1 = 2, \lambda_2 = 3, \lambda_3 = 1.5$	$\lambda_1 = 1, \lambda_2 = 1.5, \lambda_3 = 0.75$
(2, 0)	Точка мінімуму: ([2.7444000244140, -2.913178581933]) Значення функції: -64.193914318415 КОЦФ: 72 Час виконання: 0.000538 секунд	Точка мінімуму: ([-2.902206420898, -2.904916877183]) Значення функції: -78.332267862642 КОЦФ: 96 Час виконання: 0.000704 секунд
(0.156731, 0.156731)	Точка мінімуму: ([-2.904044756835, -2.902108361059]) Значення функції: -78.332291767723 КОЦФ: 102 Час виконання: 0.000717 секунд	Точка мінімуму: ([-2.903973221948, -2.8937356467763]) Значення функції: -78.33067340007 КОЦФ: 96 Час виконання: 0.000689 секунд

Тут можна побачити, що значення коефіцієнтів λ також впливають на те, який мінімум буде знайдено. А також для точки (0.156731, 0.156731) з новим набором значень мінімум знайшовся швидше.

Неправильно обраний набір λ може зробити так, що метод збігатися не буде.

Порівняння модифікованого методу пошуку за симплексом (Недлера-Міда) зі звичайним (Спендлі-Хекста-Химсвортса)

Точка/метод	Найкращий результат
(0, 0) CXX	Точка мінімуму:([-2.9101562500000036, -2.902538267371282]) Значення функції: -78.33155428091305 КОЦФ: 135 Час виконання: 0.001012 секунд
(0,0) HM	Точка мінімуму:([-2.902575683593751, -2.9094119211498537]) Значення функції: -78.33171693585675 КОЦФ: 84 Час виконання: 0.000636 секунд
(2,0) CXX	Точка мінімуму:([2.7433593749999976, -2.909980673185055]) Значення функції: -64.19471887963176 КОЦФ: 119 Час виконання: 0.001024 секунд
(2,0) HM	Точка мінімуму:([2.7444000244140616, -2.91317858193316]) Значення функції: -64.19391431841527 КОЦФ: 72 Час виконання: 0.000538 секунд
(0.156731, 0156731) CXX	Точка мінімуму:([-2.899519, -2.8960085483401445]) Значення функції: -78.33107624093535 КОЦФ: 131 Час виконання: 0.000968 секунд
(0.156731, 0156731) HM	Точка мінімуму:([-2.9040447568359387, -2.902108361059673]) Значення функції: -78.33229176772348 КОЦФ: 102 Час виконання: 0.000717 секунд
(0.156731, 0) CXX	Точка мінімуму:([2.744621624999999, -2.9059211791048165]) Значення функції: -64.19544417795069 КОЦФ: 135 Час виконання: 0.001246 секунд
(0.156731, 0) HM	Точка мінімуму:([2.740861859374998, -2.9049232201434227]) Значення функції: -64.19506359851206 КОЦФ: 72 Час виконання: 0.000532 секунд

У результаті бачимо, що алгоритм Недлера-Міда працює значно швидше за алгоритм Спендлі-Хекста-Химсвортса при найбільш оптимальних значеннях коефіцієнтів a та λ .

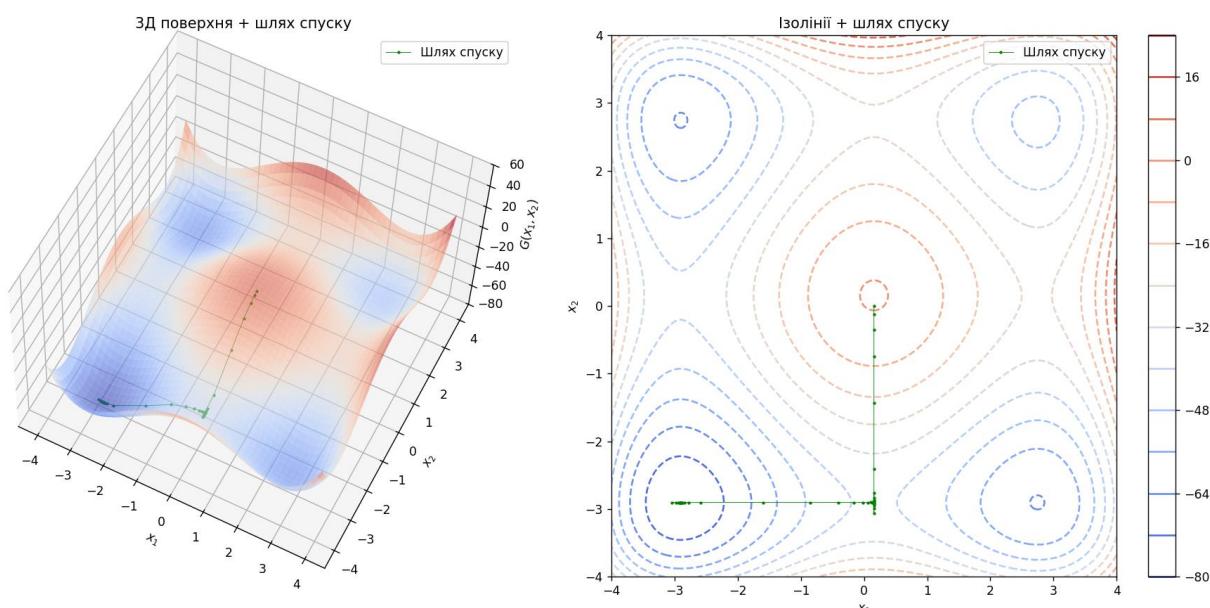
Порівняння всіх реалізованих методів:

Точка/метод	Найкращий результат
(0, 0) Найшвидший спуск	Точка мінімуму: (-2.9037337712209053,-2.9037337712209053) Значення функції: -78.33233002767454 КОЦФ: 45 Час виконання: 0.000340 секунд
(0,0) Хука-Дживса	Точка мінімуму: ([-2.903512499999997, -2.903512499999997]) Значення функції: -78.33233139151561 КОЦФ: 117 Час виконання: 0.000678 секунд
(0, 0) CXX	Точка мінімуму: ([-2.9101562500000036,- 2.902538267371282]) Значення функції: -78.33155428091305 КОЦФ: 135 Час виконання: 0.001012 секунд
(0,0) HM	Точка мінімуму: ([-2.902575683593751, -2.9094119211498537]) Значення функції: -78.33171693585675 КОЦФ: 84 Час виконання: 0.000636 секунд
(2, 0) Найшвидший спуск	Точка мінімуму: (2.746802769083388,-2.903192324734505) Значення функції: -64.19561034031136 КОЦФ: 45 Час виконання: 0.000314 секунд
(2,0) Хука-Дживса	Точка мінімуму: ([2.7497, -2.909699999999999]) Значення функції: -64.19483060964363 КОЦФ: 99 Час виконання: 0.000603 секунд
(2,0) CXX	Точка мінімуму: ([2.7433593749999976, -2.909980673185055]) Значення функції: -64.19471887963176 КОЦФ: 119 Час виконання: 0.001024 секунд
(2,0) HM	Точка мінімуму: ([2.7444000244140616, -2.91317858193316]) Значення функції: -64.19391431841527 КОЦФ: 72 Час виконання: 0.000538 секунд
(0.156731, 0156731) Найшвидший спуск	Точка мінімуму: (-2.9035340653393904,-2.9035340653393904) Значення функції: -78.33233140754277 КОЦФ: 193 Час виконання: 0.001111 секунд
(0.156731, 0156731) Хука-Дживса	Точка мінімуму: ([-2.909757888888886, -2.909757888888886]) Значення функції: -78.33098898114376 КОЦФ: 88 Час виконання: 0.000499 секунд
(0.156731, 0156731) CXX	Точка мінімуму: ([-2.899519, -2.8960085483401445]) Значення функції: -78.33107624093535 КОЦФ: 131 Час виконання: 0.000968 секунд

(0.156731, 0156731) HM	Точка мінімуму: [-2.9040447568359387, -2.902108361059673] Значення функції: -78.33229176772348 КОЦФ: 102 Час виконання: 0.000717 секунд
(0.156731, 0) Найшвидший спуск	Точка мінімуму: (-2.9035291379195183, -2.9035340260917586) Значення функції: -78.33233140712937 КОЦФ: 165 Час виконання: 0.000986 секунд
(0.156731, 0) Хука-Дживса	Точка мінімуму: [-2.91165025, -2.8963687499999997]) Значення функції: -78.3303036216384 КОЦФ: 88 Час виконання: 0.000536 секунд
(0.156731, 0) CXX	Точка мінімуму: [2.744621624999999, -2.9059211791048165]) Значення функції: -64.19544417795069 КОЦФ: 135 Час виконання: 0.001246 секунд
(0.156731, 0) HM	Точка мінімуму: [2.740861859374998, -2.9049232201434227]) Значення функції: -64.19506359851206 КОЦФ: 72 Час виконання: 0.000532 секунд

Аналіз кожного методу:

Метод швидкого спуску – це градієнтний метод. Він показав себе найшвидшим для точок, де є яскраво виражений градієнт $((0, 0), (2,0))$ та при правильно підібраних значення параметра a у формулі. Але там де зміни градієнту незначні метод спускається досить довго. На рисунку показано шлях методу через “плато”:



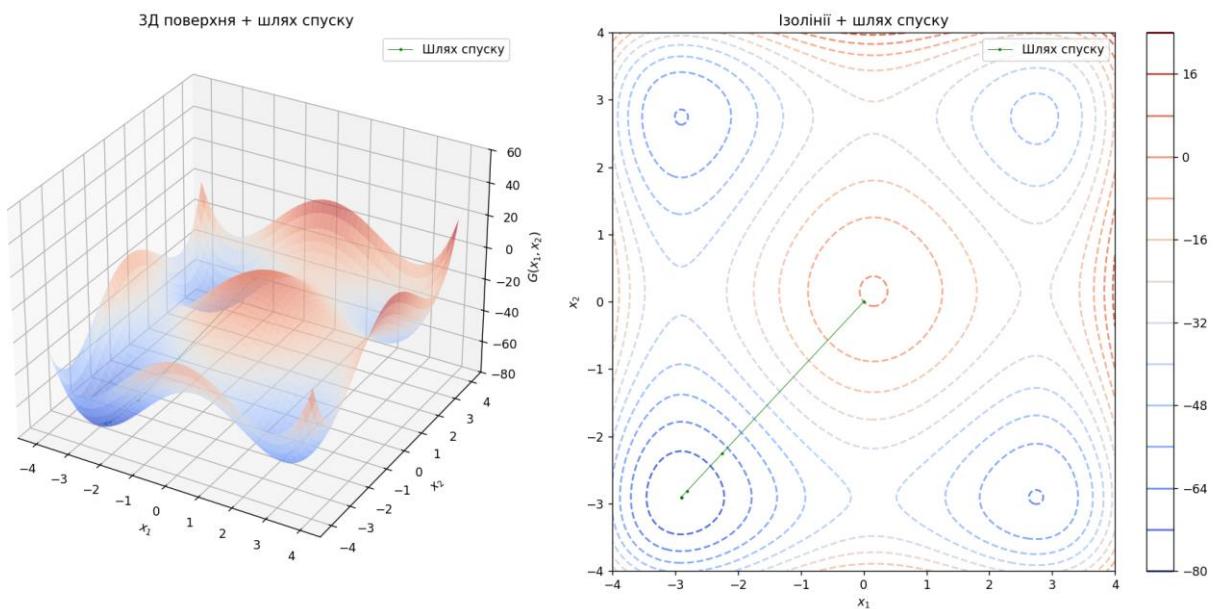
Метод є дуже нестійким. Оскільки неправильно підібрані значення параметрів легко призводять до того, що метод перестає збігатися або кількість обчислень цільової функції зростає в десятки або сотні разів.

Але якщо пощастило правильно підібрати точку та параметри, то метод показує найшвидший результат із найменшою КОЦФ.

Метод Хука-Дживса – це метод рівномірного пошуку. Він є досить простим та надійним. Неправильно підібране значення кроку може дуже сильно збільшити КОЦФ, а також у поєднанні з неоптимальним визначенням нового кроку може привести до втрати збіжності. У методі Хука-Дживса можна по різному розраховувати величину нового кроку.

```
h_new = h / 10
```

При аналізі траєкторії руху точки, видно, що найбільша кількість обчислень відбувається саме в околі точки:



Тут ми за один крок влучили в окіл, але при цьому: КОЦФ становить 274.

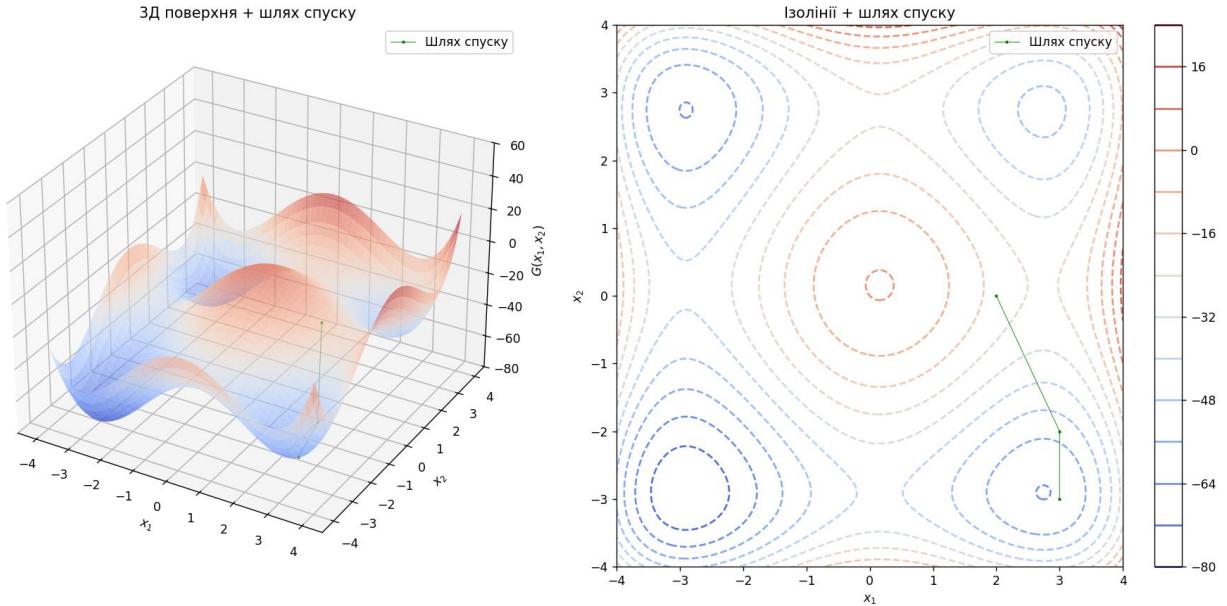
В цьому прикладі ми визначали величину нового кроку, так:

```
h_new = h / 2
```

Якщо збільшити дільник, з 2-х до 10-ти, то вдасться зменшити кількість обчислень до 126. Але якщо збільшити до 100, то кількість обчислень зросте до 178. Тобто збільшення дільника не гарантує зменшення КОЦФ

Більш того, якщо спробувати взяти завеликий дільник, то метод може не зйтися. Наприклад, запустимо алгоритм з точки $(2, 0)$ та будемо обраховувати новий крок так:

$$h_{\text{new}} = h / 1000$$



Тому на швидкість збіжності та збіжність методу впливає поєдання двох параметрів: величина початкового кроку та дільник при визначенні нового кроку. Для кожної точки таке поєдання, яке дасть найкращий результат, буде унікальним.

Пошук за симплексом (Спендлі-Хекста-Химсворт) – це алгоритм рівномірного пошуку. Він виявився найбільш повільним, але дуже стійким. Єдиним параметром, який впливає на збіжність є коефіцієнт a , який визначає розмір симплексу. Також у методів пошуку за симплексом є особливість, що в залежності від розміру симплексу можна прийти у різні точки мінімуму. У попередніх методах такого ефекту помічено не було.

Пошук за симплексом (Недлера-Міда) – алгоритм рівномірного пошуку, який є модифікацією СХХ-алгоритму. На мою думку, є найбільш оптимальним. Він збігається швидше за СХХ-алгоритм та алгоритм Хука-Дживса. Може поступатися у швидкості алгоритму швидкого спуску, але на відміну від методу швидкого спуску він не схильний до впливу градієнту, тому

у деяких випадках є швидшим за метод швидкого спуску. Тобто він є швидким та дуже стійким.

Метод	Коротка характеристика
Швидкого спуску	Дуже швидкий, але нестійкий
Хука-Дживса	Середня швидкість, показує хорошу швидкість лише при правильно підібраних параметрах. Стійкий
Спенделі-Хекста-Химсвортса	Середня швидкість. Сильно залежить від розміру симплексу. Стійкий.
Недлера-Міда	Швидкий та стійкий. Не так сильно залежить від початкового розміру, бо може розтягувати або стискати симплекс.

Висновок:

У процесі виконання лабораторної роботи було досліджено декілька методів багатовимірної оптимізації, зокрема метод найшвидшого градієнтного спуску, метод Хука-Дживса, метод пошуку за симплексом (СХХ) та модифікований алгоритм Нелдера-Міда. Було реалізовано аналітичне та чисельне обчислення градієнта функції, проведено тестування ефективності методів у різних стартових умовах, побудовано візуалізації шляхів збіжності, а також проаналізовано вплив параметрів (кроків, точності, коефіцієнтів) на результати.

Отримані результати показали, що:

- Градієнтний спуск демонструє швидку збіжність лише за правильного вибору коефіцієнта α , який важко визначити правильно з першого разу.
- Метод Хука-Дживса показав високу стабільність та гнучкість, але є чутливим до початкового кроку та дільника зменшення кроку.

- Метод СХХ виявився не найоптимальнішим, оскільки виявився найповільнішим, але обирати параметри для його оптимальної роботи виявилося легше ніж для попередніх методів. Також метод дуже стійкий. Хороший метод.
- Метод НМ є найоптимальнішим, оскільки із навищою стійкістю забезпечує найкращу збіжність. У реалізації виявився важчим за інші методи.

Реалізувавши ці методи, я глибше зрозумів їх суть та особливості використання кожного.