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

Звіт
про виконання розрахунково-графічної роботи
з дисципліни «Методи оптимізції»
на тему «Дослідження методів оптимізації»

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

Київ – 2025

Варіант 20

1 Мета роботи: Набути практичних навичок застосування окремих методів оптимізації, включаючи правильне формулювання вхідних даних, побудову алгоритмів та їх реалізацію засобами алгоритмічних мов загального призначення. Провести експериментальне дослідження обраного методу оптимізації та здійснити оцінку його ефективності. Оформити результати відповідно до вимог щодо програмної документації.

2 Хід роботи:

Метод спряженого градієнту Флетчера-Рівса для функції багатьох змінних

Методи спряженого градієнту – це чисельні методи мінімізації функцій багатьох змінних, які використовують спряжені напрямки замість звичайного антиградієнта для ефективнішого наближення до мінімуму.

Спряжені напрямки – це такі напрямки, що не "заважають" один одному при пошуку мінімуму. Вони будуються так, щоб кожен новий напрямок враховував попередній. У математичному сенсі, два напрямки d_i та d_j називаються спряженими (відносно симетричної матриці A), якщо виконується умова:

$$d_i^T A d_j = 0$$

Це узагальнення поняття перпендикулярності, але з урахуванням властивостей функції, яку мінімізують. Спряжені напрямки дозволяють набагато швидше зближуватися до мінімуму функції, особливо якщо вона має витягнуту форму (наприклад, еліпси рівня).

Основна ідея:

Метод Флетчера–Рівса – це ітераційний метод мінімізації функцій багатьох змінних, який покращує звичайний градієнтний спуск.

Замість того щоб щоразу рухатись у напрямку антиградієнта, він рухається в спряжених напрямках, які враховують попередній рух, що дозволяє значно зменшити кількість ітерацій**.**

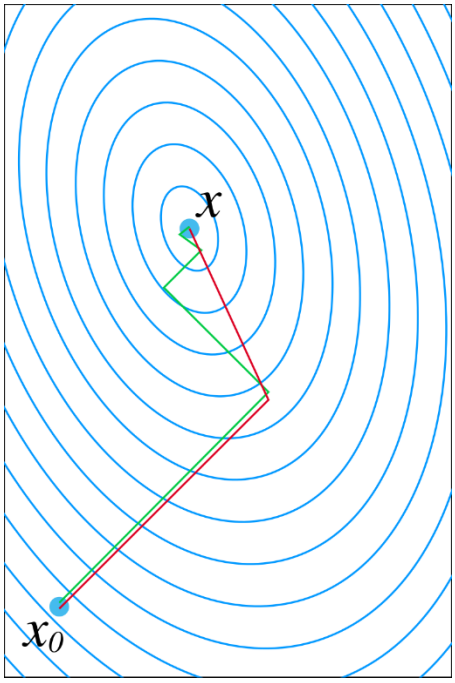


Рис 1. - Порівняння збіжності градієнтного спуску (зелений шлях) і методу спряжених градієнтів (червоний шлях) при мінімізації квадратичної функції.

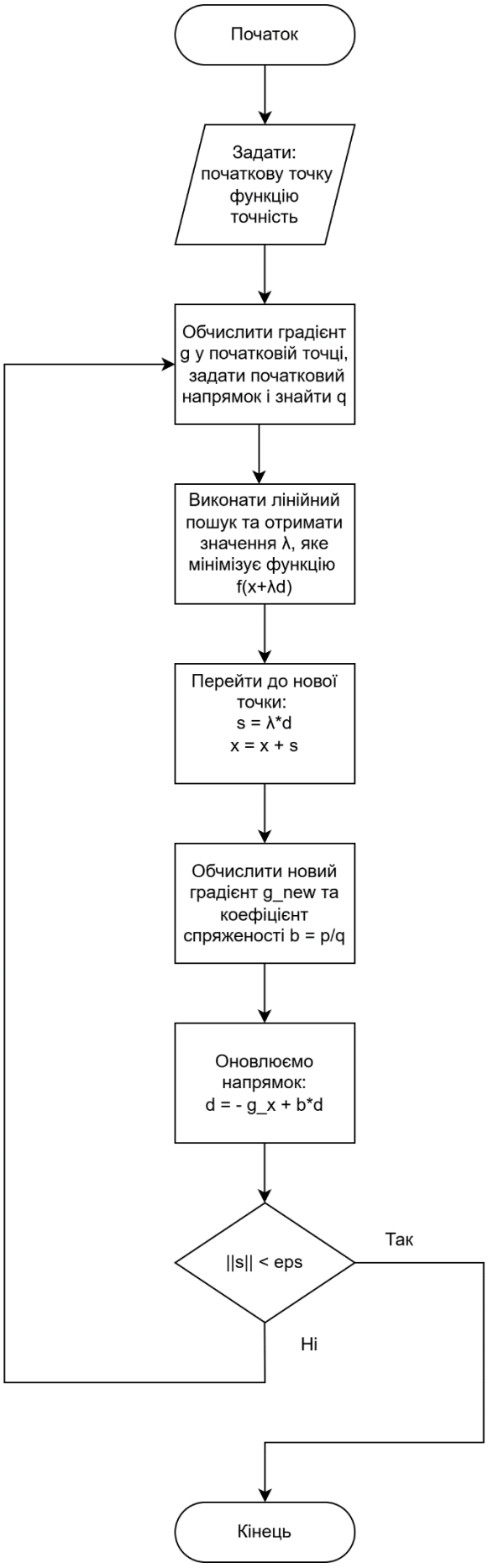
На рисунку показано перевагу методу спряжених градієнтів у задачах квадратичної мінімізації: він не "гойдається", а ефективно рухається до мінімуму**.**

Алгоритм:

1. Обчислити градієнт початкової точки x: $g_x = \nabla f(x)$, та напрямок спуску: $d = -g_x$
2. Виконати одновимірний пошук та знайти значення λ, яке мінімізує функцію уздовж заданого напрямку d, починаючи з точки x. (тут можна використати методи: дихотомії, золотого перетину або інші методи лінійного пошуку)
3. Перейти в нову точку: $\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{s}$, де $s = \lambda \times d$ ($s -$ вектор зсуву (в яку точку і наскільки треба рухатися)) та зберегти градієнт, знайдений у попередній точці: $g_y = g_x$.
4. Обчислити градієнт у щойно знайденій точці: $g_x = \nabla f(x)$ та обчислити коефіцієнт спряжності: $\beta = \frac{g_x^T \times g_x}{g_y^T \times g_y}$
5. Визначити новий напрямок руху: $d = -g_x + \beta \times d$
6. Якщо $||s|| < \varepsilon$, то перейти до кроку 2.
7. Завершити алгоритм

На блок-схемі замість $g_x^T \times g_x$ (скалярних добутків градієнтів з самим собою (тобто нормами в квадраті)) введемо допоміжні змінні q та p, які полегшують обчислення коефіцієнта β.

Блок-схема:


Початок
Задати:
початкову точку
функцію
точність
Обчислити градієнт
g у початковій точці,
задати початковий
напрямок і знайти q
Виконати лінійний
пошук та отримати
значення λ, яке
мінімізує функцію
f(x+λd)
Перейти до нової
точки:
s = λ*d
x = x + s
Обчислити новий
градієнт g_new та
коефіцієнт
спряженості b = p/q
Оновлюємо
напрямок:
d = - g_x + b*d
||s|| < eps
Так
Ні
Кінець

Формулювання задач:

Оскільки метод передбачає обчислення градієнта цільової функції, на задачі, які можна ним розв’язати, накладаються певні обмеження.

Цільова функія повинна:

- бути неперервною та диференційовною у всіх точках області визначення;
- мати обчислюваний градієнт (аналітично або чисельно);

Отже найкращим варіантом задач для розв’язання методом Флетчера-Рівса є класичні математичні функції.

Визначимо декілька функцій, мінімуми яких знайдемо в подальшому:

1. Styblinski-Tang function

   Функція Стюблінські-Танга (англ. Styblinski-Tang function) - це відома Тестова функція для перевірки методів оптимізації.

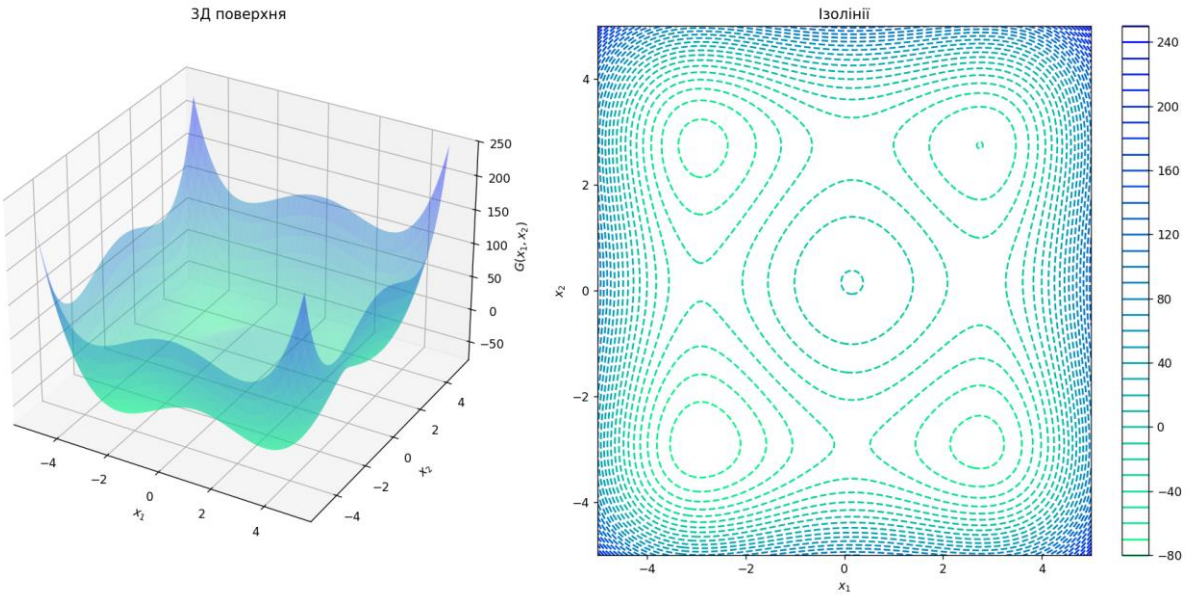
   Формула:

$$f(x) = \frac{1}{2}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i)$$

   Область визначення: $-5 < x_i < 5, i = 1, 2$

   Глобальний мінімум: $f(-2.903534, -2.903534) = -78.33234$

   Візуалізація:



2. Goldstein-Price function

Формула:

$$f(x,y) = [1 + (x + y + 1)^2 \times (19 - 14x + 3x^2 - 14y + 6xy + 3y^2)] \times$$
$$\times [30 + (2x - 3y)^2 \times (18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]$$

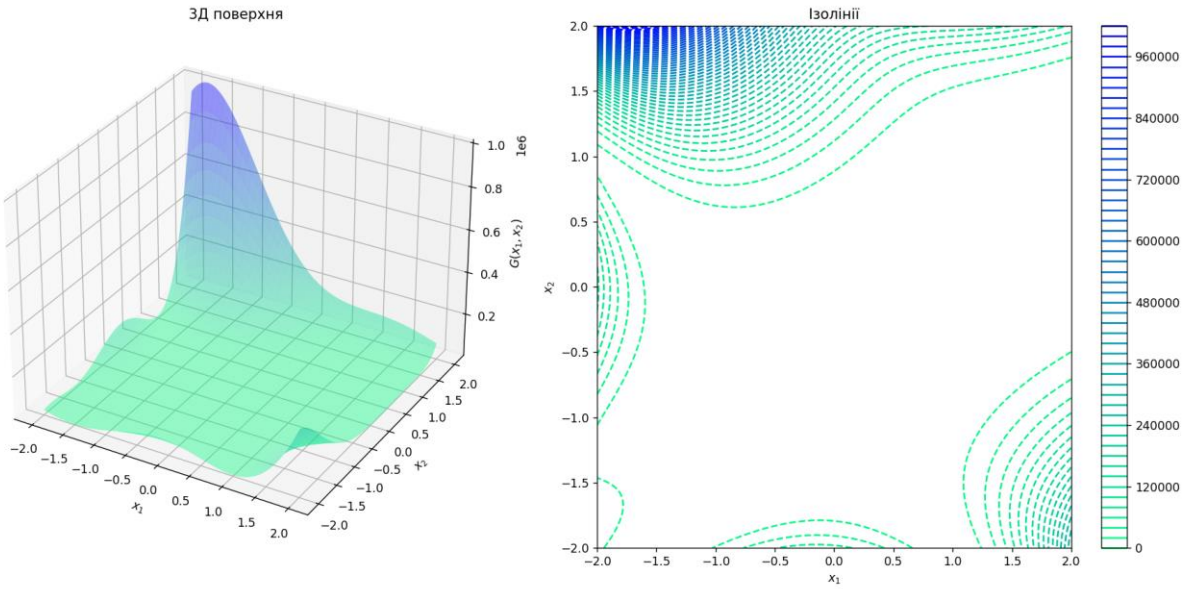
Область визначення:

$$-2 \leq x, y \leq 2$$

Глобальний мінімум:

$$f(0,-1) = 3$$

Візуалізація:



Ці функції мають складний рельєф (багато вигинів, різких підйомів та впадин). Через це звичайний градієнтний спуск може довго блукати, оскільки він орієнтується на антиградієнт, який може змінюватися дуже швидко. Натомість метод Флетчера-Рівса має враховувати попередній напрямок спуску і будувати більш оптимальний маршрут. Тому, теоретично, ці функції добре покажуть відмість цих методів та переваги методу Флетчера-Рівса.

Програмна реалізація методу Флетчера-Рівса:

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

function_calls = 0


# Styblinski-Tang function
#
# Область визначення
# x1_vals = np.linspace(-5, 5, 400)
# x2_vals = np.linspace(-5, 5, 400)
#
def G(x1, x2):
    global function_calls
    function_calls += 1
    return 0.5 * ((x1 ** 4 - 16 * x1 ** 2 + 5 * x1) + (x2 ** 4 - 16 * x2 ** 2
+ 5 * x2))


# Goldstein-Price function
#
# Область визначення
# x1_vals = np.linspace(-2, 2, 400)
# x2_vals = np.linspace(-2, 2, 400)
# def G(x, y):
#     global function_calls
#     function_calls += 1
#     term1 = 1 + (x + y + 1)**2 * (19 - 14*x + 3*x**2 - 14*y + 6*x*y +
3*y**2)
#     term2 = 30 + (2*x - 3*y)**2 * (18 - 32*x + 12*x**2 + 48*y - 36*x*y +
27*y**2)
#     return term1 * term2

def golden_ratio(f, a, b, eps):
    l = b - a
    x1 = a + 0.382 * l
    x2 = a + 0.618 * l
    g1 = f(x1)
    g2 = f(x2)

    while b - a > eps:
        if g1 > g2:
            a = x1
            l = b - a
            x1 = x2
            g1 = g2
            x2 = a + 0.618 * l
            g2 = f(x2)
        elif g1 < g2:
            b = x2
            l = b - a
```

```
            x2 = x1
            g2 = g1
            x1 = a + 0.382 * l
            g1 = f(x1)
        else:
            a = x1
            b = x2
            l = b - a
            x1 = a + 0.382 * l
            x2 = a + 0.618 * l
            g1 = f(x1)
            g2 = f(x2)

    return (a + b) / 2

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


def fletcher_reeves(x1, x2, accuracy, delta, max_iter):
    path = [(x1, x2)]
    # Обчислюємо градієнт та задаємо початковий напрямок

    grad_x1, grad_x2 = central_differences(x1, x2, delta)
    d_x1, d_x2 = -grad_x1, -grad_x2

    for i in range(max_iter):

        # одновимірна функція, яка описує значення початкової функції G
        вздовж напрямку d = (d_x1, d_x2)
        def phi(lam):
            return G(x1 + lam * d_x1, x2 + lam * d_x2)

        lam = golden_ratio(phi, 0, 0.1, accuracy)
```

```
        # переходимо в нову точку
        x1_new = x1 + lam * d_x1
        x2_new = x2 + lam * d_x2

        old_grad_x1 = grad_x1
        old_grad_x2 = grad_x2
        q = old_grad_x1 ** 2 + old_grad_x2 ** 2

        grad_x1, grad_x2 = central_differences(x1_new, x2_new, delta)

        p = grad_x1 ** 2 + grad_x2 ** 2

        if q == 0:
            print(f'Ділення на 0')
            break

        b = p/q

        d_x1 = -grad_x1 + b * d_x1
        d_x2 = -grad_x2 + b * d_x2

        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
        path.append((x1, x2))

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


# Початкові значення
x_start = [-1, 1]
delta = 0.0001
alpha = 0.001
accuracy = 0.0001
max_iter = 1000

print(f"Метод найщвидшого спсуску:")
start1 = time.perf_counter_ns()
x1_fin, x2_fin, G_fin, path_gd = gradient_descent(x_start[0], x_start[1],
alpha, delta, accuracy, max_iter)
end1 = time.perf_counter_ns()
elapsed1 = end1 - start1

print(f'Точка мінімуму: ({x1_fin},{x2_fin})\nЗначення функції: {G_fin}')
print(f'КОЦФ: {function_calls}')
print(f"Час виконання: {elapsed1 / 1_000_000_000:.6f} секунд")

print(f"Метод Флетчера-Рівса:")
function_calls = 0
start = time.perf_counter_ns()
x1_fin, x2_fin, G_fin, path_fr = fletcher_reeves(x_start[0], x_start[1],
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")

# Візуалізація

x1_vals = np.linspace(-5, 5, 400)
x2_vals = np.linspace(-5, 5, 400)
X1, X2 = np.meshgrid(x1_vals, x2_vals)
Z = G(X1, X2)

# Для найшвидшого спуску
path_x1_gd = [p[0] for p in path_gd]
path_x2_gd = [p[1] for p in path_gd]
path_G_gd = [G(p[0], p[1]) for p in path_gd]

# Для Флетчера-Рівса
path_x1_fr = [p[0] for p in path_fr]
path_x2_fr = [p[1] for p in path_fr]
path_G_fr = [G(p[0], p[1]) for p in path_fr]


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.winter_r, alpha=0.4)
ax1.plot(path_x1_gd, path_x2_gd, path_G_gd, color='black', label='Найшвидший
спуск', marker='.')
ax1.plot(path_x1_fr, path_x2_fr, path_G_fr, color='red', label='Флетчера-
Рівса', marker='.')
ax1.set_zlim(np.min(Z), np.max(Z))
ax1.set_xlabel('$x_1$')
ax1.set_ylabel('$x_2$')
ax1.set_zlabel('$G(x_1, x_2)$')
ax1.set_title('3Д поверхня')
ax1.legend()

# Контурний графік
ax2 = fig.add_subplot(1, 2, 2)
contour = ax2.contour(X1, X2, Z, cmap=cm.winter_r, levels=40,
linestyles='dashed')
ax2.plot(path_x1_gd, path_x2_gd, color='black', label='Найшвидший спуск',
marker='.')
ax2.plot(path_x1_fr, path_x2_fr, color='red', label='Флетчера-Рівса',
marker='.')
plt.colorbar(contour, ax=ax2)
ax2.set_xlabel('$x_1$')
ax2.set_ylabel('$x_2$')
ax2.set_title('Ізолінії')
ax2.legend()

plt.tight_layout()
plt.show()
```

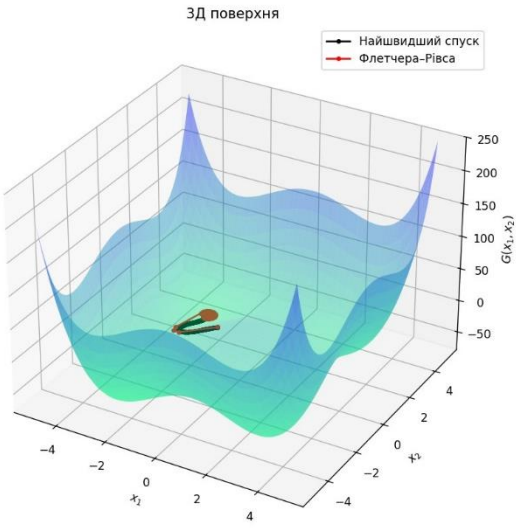
Тестування методу Флетчера-Рівса:

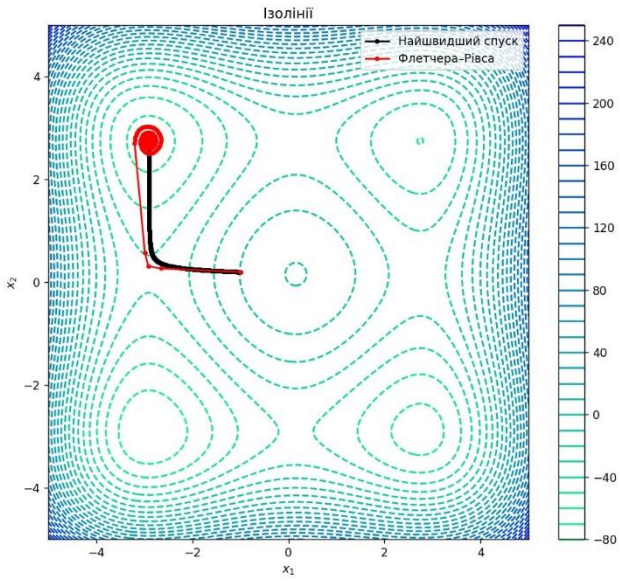
Під час тестування ми будемо порівнювати метод Флетчера-Рівса із методом найшвидшого спуску, щоб побачити чи вдається йому покращити результати.

Серед параметрів, які можна змінювати у методі Флетчера-Рівса є точність, розмір проміжку, який ми досліджуємо одновимірним пошуком (розмір кроку).

Початкова точка $\mathbf{x}[-\mathbf{1}, \mathbf{0}.\mathbf{2}]$, розмір проміжку: 0.1

Метод Флетчера-Рівса	Метод найшвидшого спуску
Метод зійшовся за 437 ітерацій Точка мінімуму: (-2.903238, 2.746242) Значення функції: -64.195606 **КОЦФ: 9182** Час виконання: 0.008303 секунд	Метод зійшовся за 467 ітерацій Точка мінімуму: (-2.903533, 2.743413) Значення функції: -64.195444 **КОЦФ: 1869** Час виконання: 0.001958 секунд





Розмір проміжку 0.05 дає такі результати:

Метод зійшовся за 119 ітерацій

Точка мінімуму: (-2.903799, 2.746802)

Значення функції: -64.19561114003821

КОЦФ: 2861, Час виконання: 0.002423 секунд

Якщо обрати крок 0.55, то вдасться ще сильніше зменшити час виконання та КОЦФ:

Метод зійшовся за 95 ітерацій

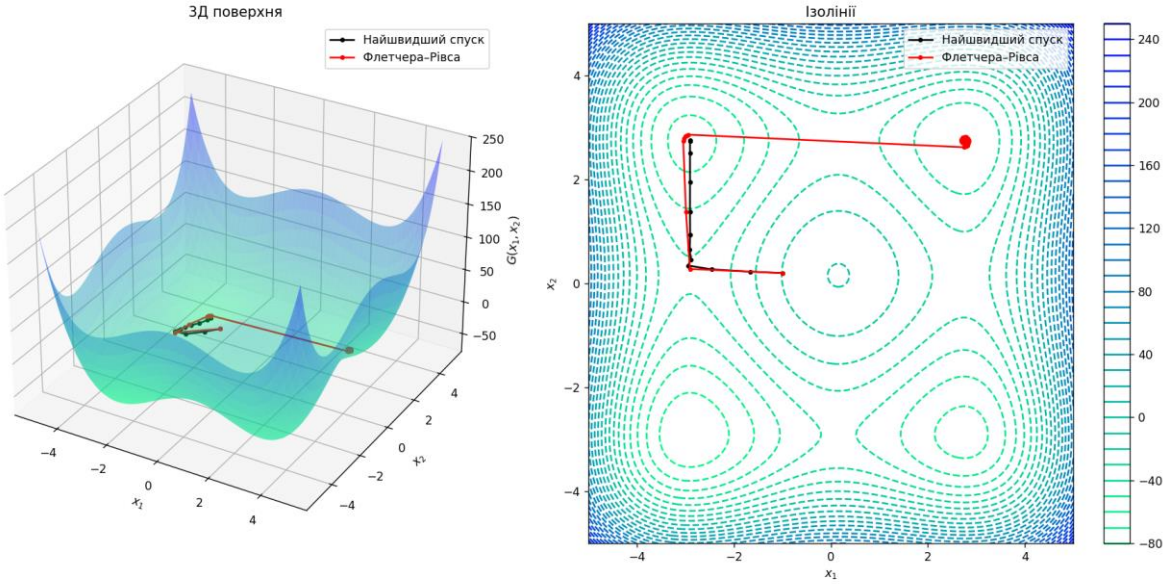
Точка мінімуму: (-2.9035221050919073,2.7471124990015254)

Значення функції: -64.19561095249895

КОЦФ: 2285

Час виконання: 0.001978 секунд

Якщо обрати завеликий крок, то метод вибухає і або зовсім не знаходить рішення або вистрибує з мінімуму, до якого наближався:

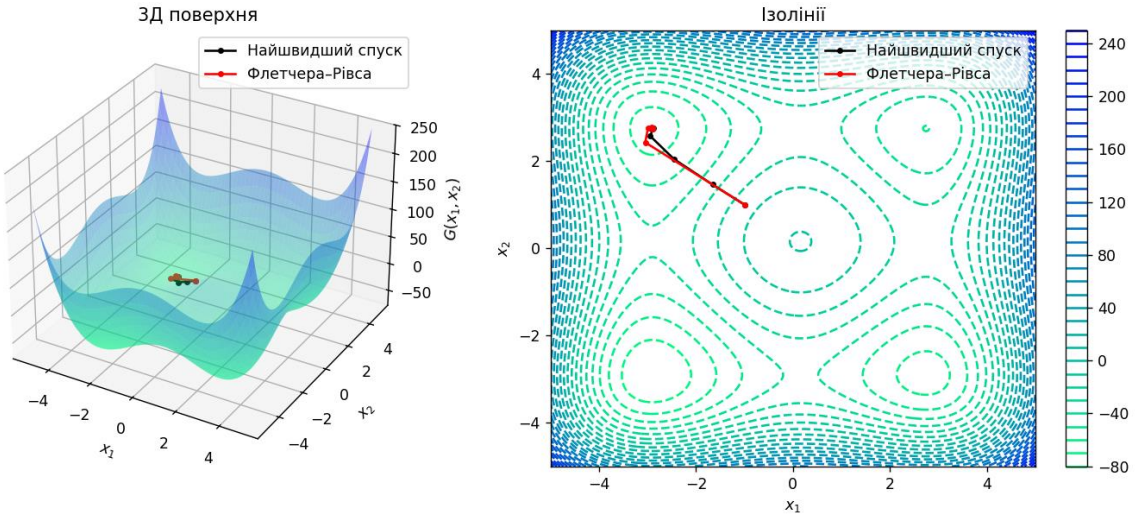


Початкова точка x$[-1, 1]$:

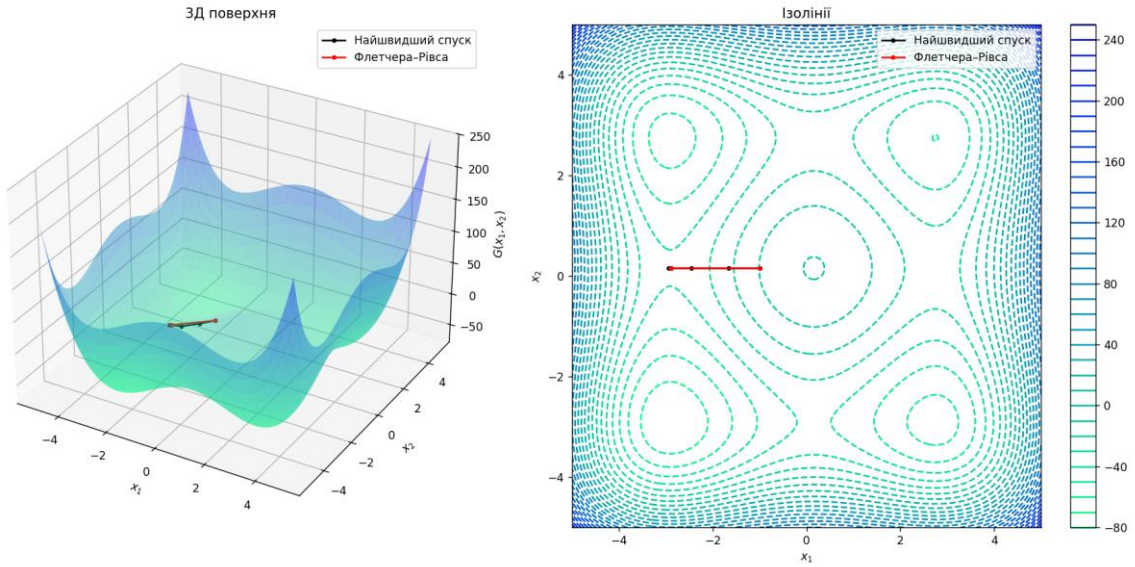
Розмір проміжку, досліджуваного одновимірним пошуком: 0.05

Результати:

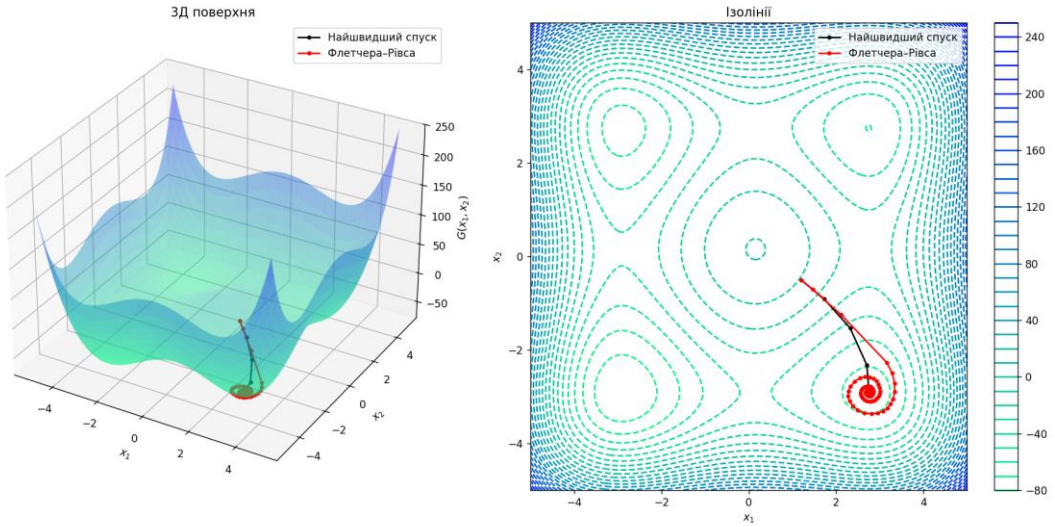
Метод Флетчера-Рівса	Метод найшвидшого спуску
Метод зійшовся за 11 ітерацій Точка мінімуму: (-2.903483, 2.746803) Значення функції: -64.19561231 КОЦФ: 45 Час виконання: 0.000092 секунд	Метод зійшовся за 10 ітерацій Точка мінімуму: (-2.903493, 2.746841) Значення функції: -64.195612 КОЦФ: 245 Час виконання: 0.000229 секунд



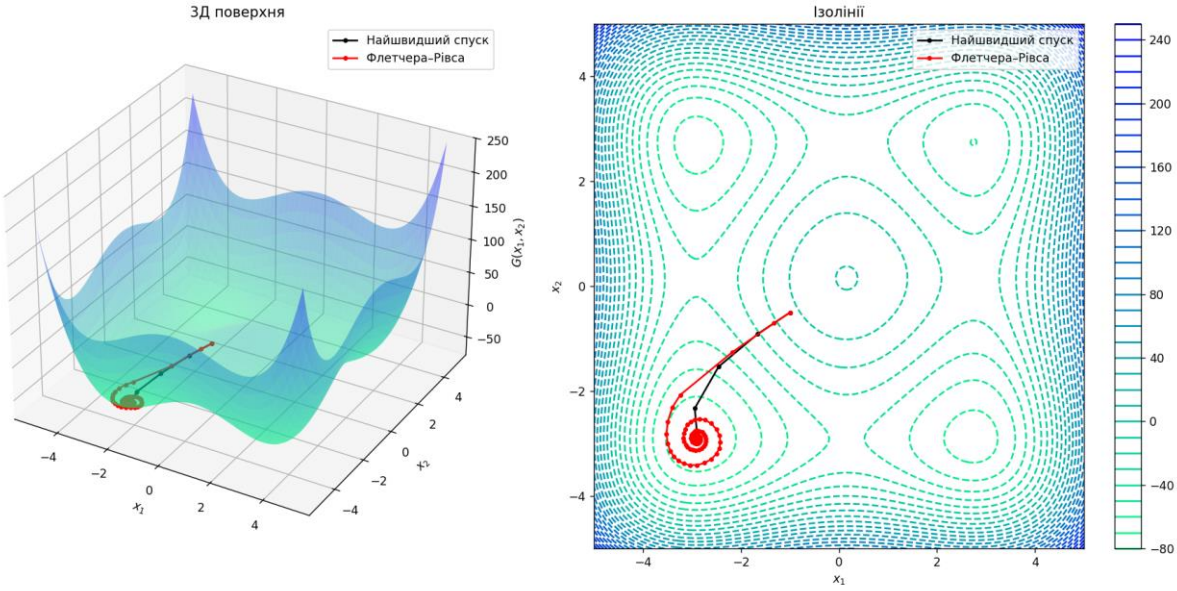
У точках, де градієнт дуже близький до 0, метод Флетчера-Рівса, так само як і метод найшвидшого спуску зупиняється.



Початкова точка $x[1.2, -0.5]$:



Початкова точка $x[-1,-0.5]$:



З проведених експериментів видно, що методу Флетчера-Рівса притамана та сама поведінка, що і методу найшвидшого спуску. Тобто всі недоліки градієнтних методів: якщо взяти початкову точку на схилі, то ймовірніше за все алгоритм просто спуститься до найближчого мінімума і не вийде за межі “ями”. У місцях, де зміни функції незначні (плато) алгоритм також дуже важко продвинутися, бо градієнт дуже малий і відповідно крок також, тому уже ймовірно, що алгоритм там і зупиниться. Якщо обрати неправильний крок, то метод вибухає.

Поки що метод Флетчера-Рівса не дав покращень у порівняні з методом найшвидшого спуску. Можливо це відбувається через неправильно підібрані параметри або спецфіку обраної функції. Але природньо, що КОЦФ буде більшою, оскільки всередині ми додатково виконуємо одновимірний пошук.

Спробуємо виконати пошук мінімуму наступної функції:

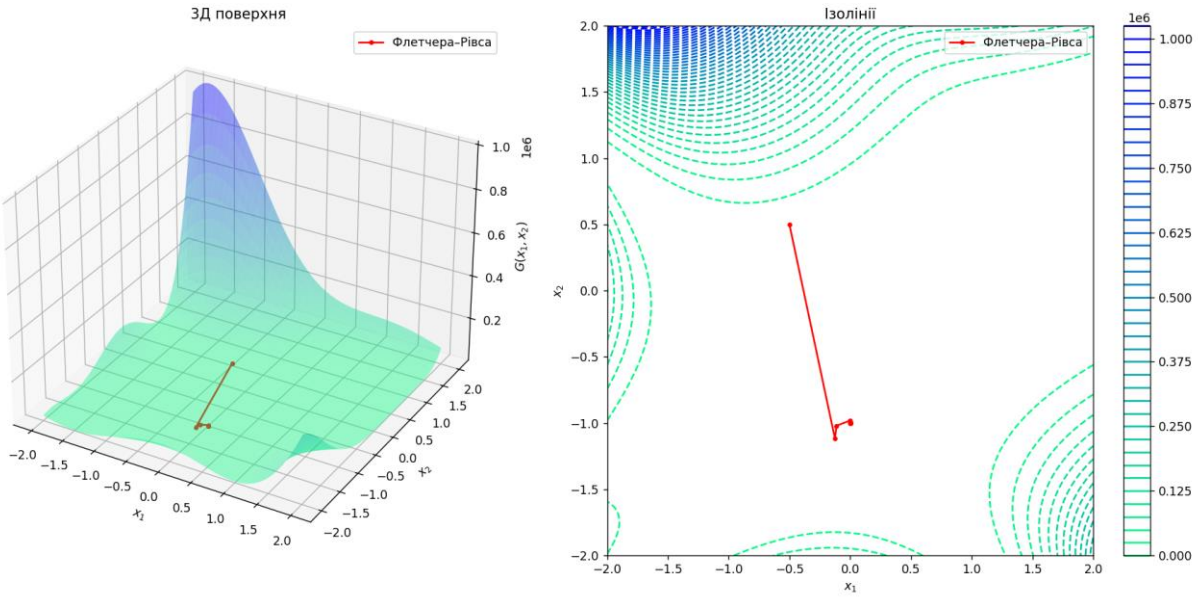
Початкова точка $x[-0.5, 0.5]$:

Результати:

Метод Флетчера-Рівса	Метод найшвидшого спуску
Метод зійшовся за 7 ітерацій Точка мінімуму: (-8.880499044135659e-05,-1.000086802240042) Значення функції: **3.0000035776204985** КОЦФ: 96 Час виконання: 0.000177 секунд	Метод зійшовся за 3 ітерацій Точка мінімуму: (2.5377017250357115e+25,7.60307003275 58835e+25) Значення функції: 9.413137825021491**e+209** КОЦФ: 13 Час виконання: 0.000070 секунд

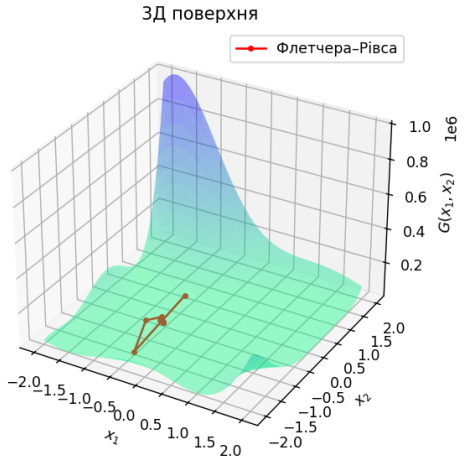
Отут вже ситуація дещо змінюється на користь методу Флетчера-Рівса. Функція Goldstein-Price function через свою структуру призводить до того, що метод найшвидшого спуску просто вибухає. Після числених спроб підібрати початкову точку та параметри, він не знаходить мінімум. Натомість алгоритм Флетчера-Рівса без проблем знаходить очікуваний мінімум.

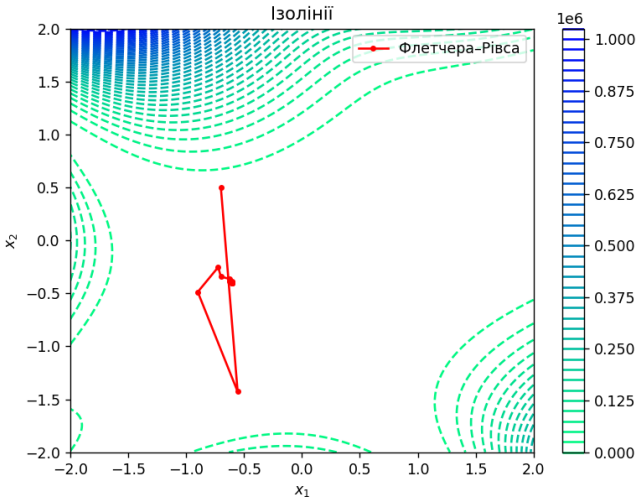
Так виглядає шлях для методу Флетчера-Рівса:



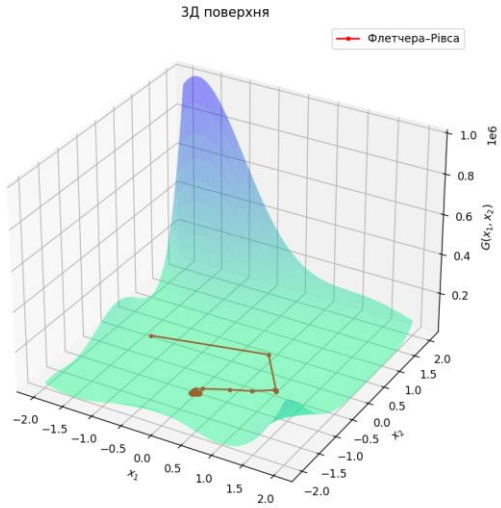
Запустимо цей алгоритм з різних точок:

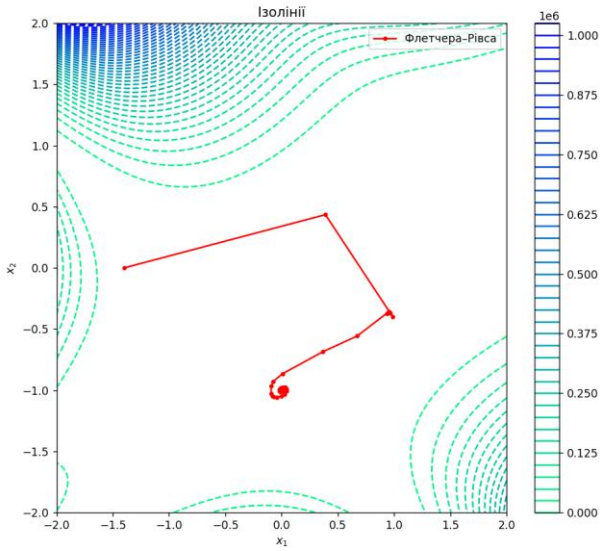
Початкова точка $x[-0.7, 0.5]$:



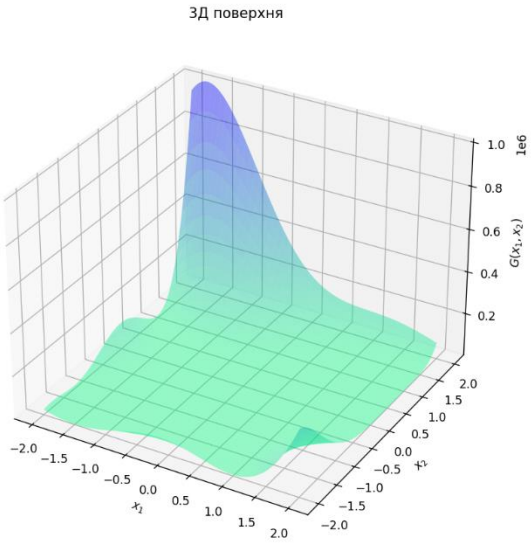


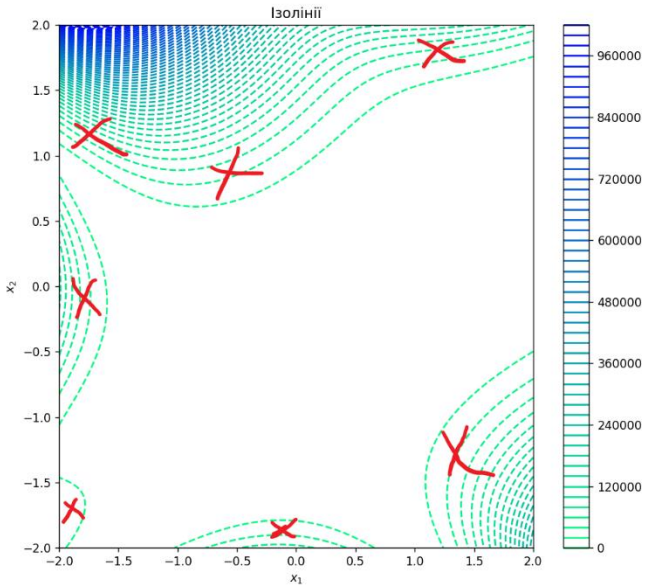
Початкова точка $x[-1.4, 0]$:





Якщо спробувати обрати початкову точку на “схилах”, то метод вибухає.





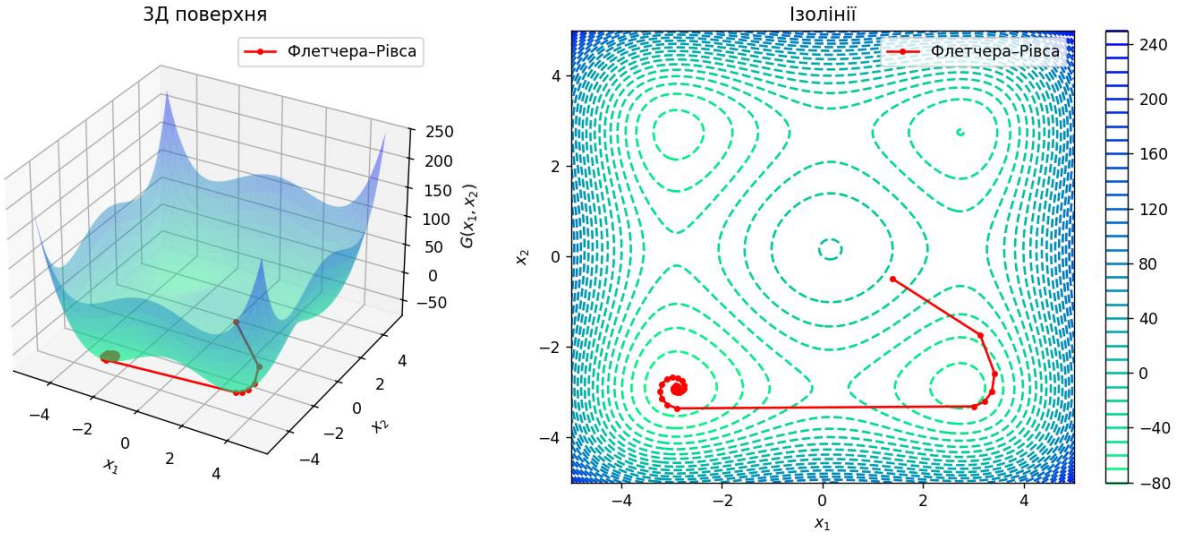
Це стається через великі значення градієнта у вибраній точці на “схилі” функції. У результаті метод обирає занадто великий крок і виходить за межі області визначення, де значення функції та її градієнт стають нестабільними. Через це обчислення втрачають точність, і алгоритм "вибухає", тобто породжує величезні або недійсні значення.

Тепер протестуємо метод для цих двох функцій та подивимося, як він буде себе поводити, стартуючи з однієї точки, але щ різними параметрами. Спочатку повернемося до функції Styblinski-Tang function:

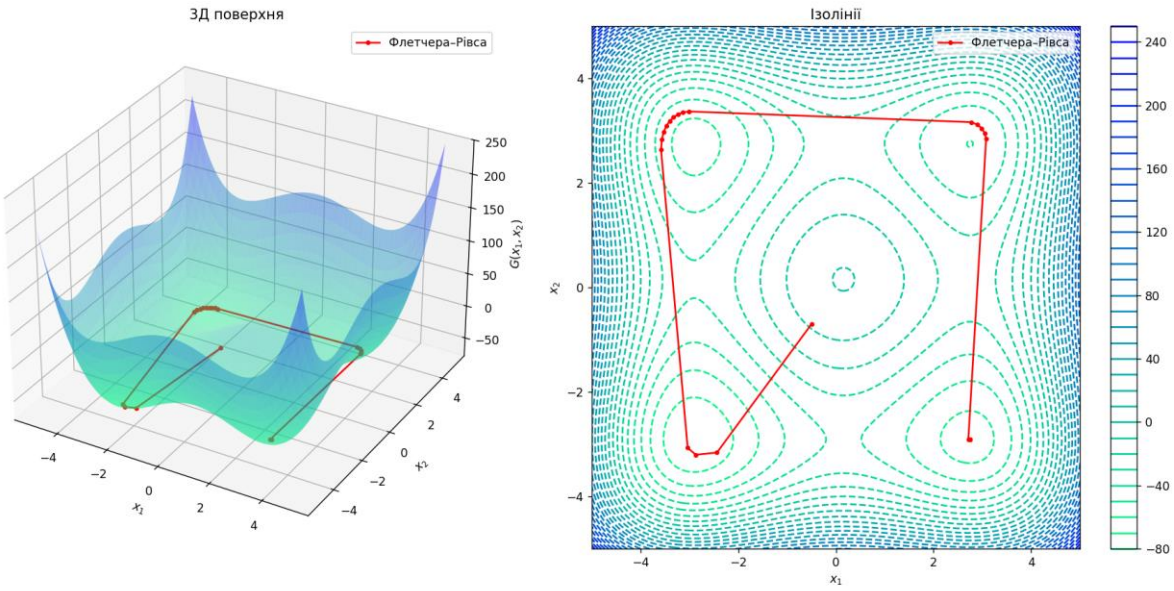
Точка	Розмір кроку	Результат
[1.4, -0.5]	0.001	Значення функції: -63.81074558030043 КОЦФ: 11005 Час виконання: 0.010081 секунд
	0.01	Значення функції: -64.19560766412133 КОЦФ: 7461 Час виконання: 0.006410 секунд
	0.1	Значення функції: -64.19561178020511 КОЦФ: 1538 Час виконання: 0.001355 секунд
	0.2	Значення функції: -78.33233076308588 КОЦФ: 1544 Час виконання: 0.001355 секунд
[-0.5, -0.7]	0.01	Значення функції: -78.33232975811787 КОЦФ: 2277 Час виконання: 0.002064 секунд
	0.1	Значення функції: -78.33233107363371 КОЦФ: 803 Час виконання: 0.000792 секунд

	0.2	Значення функції: -78.33233111980257 КОЦФ: 511 Час виконання: 0.000486 секунд
	0.65	Значення функції: -64.19561232308077 КОЦФ: 605 Час виконання: 0.000579 секунд

При старті з точки $x[-1.4, 0.5]$ і кроці 0.2 в алгоритму вже з'являються досить сильні коливання, які можуть призводити до великих скачків, і на рисунку показано, як алгоритм перестрибнув з одного локального мінімуму до іншого.



При старті з точки $x[-0.5, -0.7]$ та кроці 0.65:



Отже при кроках близьких до вибуху, алгоритм починає вести себе хаотично і може вистрибувати з "ям".

Тепер спробуємо поекспереметувати з функцією Goldstein-Price function:

Точка	Розмір кроку	Результат
[0, -1.6]	0.001	Значення функції: 3.0000026474004686 КОЦФ: 104 Час виконання: 0.000216 секунд
	0.01	Значення функції: 3.0000007659031835 КОЦФ: 213 Час виконання: 0.000388 секунд
	0.055	Значення функції: 3.0000158195531252 КОЦФ: 1605 Час виконання: 0.002467 секунд
	0.1	Вибух
[-1.4, 0]	0.001	Значення функції: 3.0000286877006817 КОЦФ: 654 Час виконання: 0.001054 секунд
	0.01	Вибух
	0.0005	Значення функції: **84.00084900181854** КОЦФ: 125 Час виконання: 0.000256 секунд
	0.0009	Значення функції: 3.0000011451103687 КОЦФ: 181 Час виконання: 0.000345 секунд

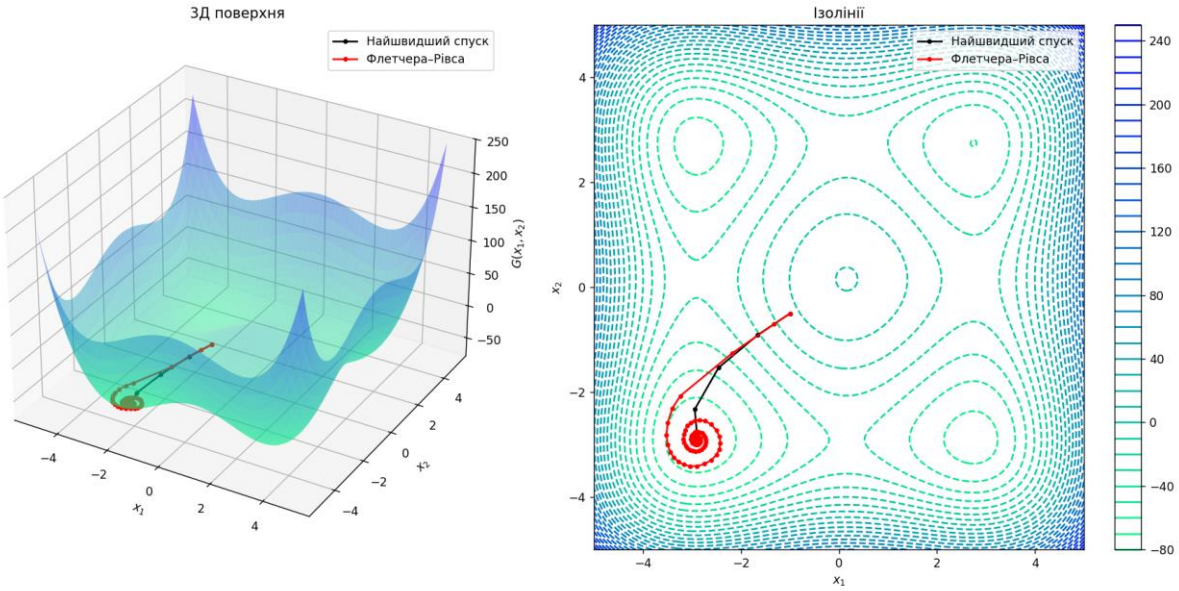
Замале або завелике значання кроку призводять до того, що метод перестає правильно знаходити мінімум.

Якщо для попередньої функції збільшення кроку сприяло зменшенню кількості обчислень цільової функції, то тепер збільшення кроку навпаки призводить до збільшення КОЦФ. Тобто для даної функції зменшення кроку

призводить до покращення результату, хоча для попередньої функції це навпаки призводило до збільшення кількості обчислень.

Висновок: у процесі виконання розрахунково-графічної роботи я детально розібрався з ідеєю та алгоритмом роботи методу спряженого градієнта Флетчера-Рівса, а також програмно його реалізував і провів порівняння із методом найшвидшого спуску, аби зрозуміти його переваги і недоліки.

Порівняння методів відбувалося на двох різних функціях: Styblinski-Tang та Goldstein-Price. При знаходженні мінімуму першої функції краще себе показав метод найшвидшого спуску. У методі Флетчера-Рівса ми враховуємо попередній напрямок градієнту та не можемо рухатися безпосередньо у напрямку градієнту в поточній точці. І це не завжди корисно, оскільки замість того, щоб спуститися напряму, ми почали рухатися по спіралі:



І якщо у випадку першої функції це тільки заважало, то для другої функції це працювало чудово, і у той час, як метод найшвидшого спуску вибухав і жодного разу не зміг знайти мінімум, метод Флетчера-Рівса добре себе показав та без особливих проблем (якщо не зважати на нестійкість при погано обраній початковій точці) знайшов правильний мінімум. Це підтверджує теорію та доводить те, що для яружних функцій, у яких мінімум може знаходитися в досить великій зоні підхід із сопряженими градієнтами чудово працює.

Під час тестування методів я залишав коментарі та пояснював чому отримано саме такі результати, але можна ще раз відмітити, що метод Флетчера-Рівса страждає тими самими проблемами, що і метод найшвидшого спуску (і я підозрюю всі інші градієнтні методи), оскільки обчислення градієнта приносить нестійкість. При неправильно обраних параметрах метод просто вибухає.