Nguyễn Khắc Sơn – 21085691
Bài tập về nhà lần 3 (Xấp xỉ hàm – P2)

Bài 1: Dựa trên những kiến thức đã học trên lớp và đọc thêm tài liệu ở nhà, em hãy lập một bảng để tổng kết và so sánh 03 kiểu thuật toán học gradient descent: **batch gradient descent (BGD), stochastic gradient descent (SGD)** và **mini-batch gradient descent (MBGD).**

Ví dụ tham khảo: một bảng đơn giản do một bạn sinh viên lập để tóm tắt những kiến thức đã học gồm có các cột như sau:

Bảng 1: Ưu điểm, nhược điểm

Thuật toán	**Ưu điểm**	**Nhược điểm**
BGD	**Ổn định:** sử dụng toàn bộ dữ liệu để tính gradient, giúp giảm thiểu nhiễu **Tối ưu toàn cục:** Do tính toàn bộ dữ liệu, BGD thường hội tụ về một điểm tối ưu toàn cục.	**Tốn tài nguyên:** chậm và tốn bộ nhớ khi xử lý với các bộ dữ liệu lớn. **Khó mở rộng:** Không phù hợp khi dữ liệu không thể chứa vừa trong bộ nhớ.
SGD	**Nhanh và hiệu quả:** cập nhật trọng số sau mỗi điểm dữ liệu, giảm thời gian tính toán, làm việc tốt với dữ liệu lớn. **Khả năng thoát khỏi "local minima":** SGD có xu hướng thoát khỏi các điểm tối ưu cục bộ nhờ vào tính ngẫu nhiên.	**Nhiễu cao:** thường có nhiễu, dẫn đến quá trình hội tụ không ổn định, tốn thời gian hơn để đạt được tối ưu toàn cục. **Điều chỉnh học suất:** Cần phải cẩn thận trong việc điều chỉnh tốc độ học
MBGD	**Cân bằng giữa BGD và SGD:** là sự kết hợp ưu điểm của cả hai phương pháp, giúp giảm nhiễu và cải thiện tốc độ học. **Hiệu suất tính toán tốt**	**Vẫn có nhiễu:** vẫn có thể bị nhiễu khi kích thước mini-batch quá nhỏ. **Yêu cầu tinh chỉnh:** Cần điều chỉnh kích thước mini-batch sao cho phù hợp với yêu cầu của bài toán cũng như cấu hình của hệ thống.

Local Minima là điểm trong không gian tham số mà hàm mục tiêu (loss function) có giá trị thấp hơn so với các điểm lân cận, nhưng không phải là giá trị thấp nhất

Global Minimum là điểm trong không gian tham số mà hàm mục tiêu có giá trị thấp nhất (điểm tối ưu nhất) trên toàn bộ không gian

Bảng 2: Ứng dụng & Chú ý

Thuật toán	**Ứng dụng**	**Chú ý**
BGD	Thường được sử dụng trong các bài toán với tập dữ liệu nhỏ, nơi tính toán với toàn bộ dữ liệu là có thể	Đảm bảo bộ nhớ đủ lớn để chứa toàn bộ dữ liệu. Không phù hợp cho các bài toán có dữ liệu lớn hoặc yêu cầu xử lý nhanh.
SGD	Phù hợp với các bài toán học máy trên dữ liệu lớn hoặc streaming data. Được sử dụng rộng rãi trong các mô hình deep learning.	Nên sử dụng kèm với các kỹ thuật như giảm tốc độ học hoặc kỹ thuật tối ưu hóa khác (như Momentum, Adam) để cải thiện quá trình hội tụ. Kết quả có thể bị ảnh hưởng bởi sự sắp xếp dữ liệu, nên cần shuffle dữ liệu trước khi huấn luyện.
MBGD	Phổ biến trong deep learning, đặc biệt với các mạng nơ-ron sâu khi dữ liệu lớn, cần cân bằng giữa tốc độ và chính xác.	Phải chọn kích thước mini-batch hợp lý để đạt được hiệu suất tốt nhất, thường là 32, 64, hoặc 128. Nếu kích thước mini-batch quá lớn sẽ mất tính cân bằng. Ngược lại, quá nhỏ có thể làm giảm tính ổn định.

Shuffle dữ liệu : xáo trộn hoặc trộn ngẫu nhiên dữ liệu trong tập dữ liệu, nhằm tránh Overfitting, cải thiện hiệu suất

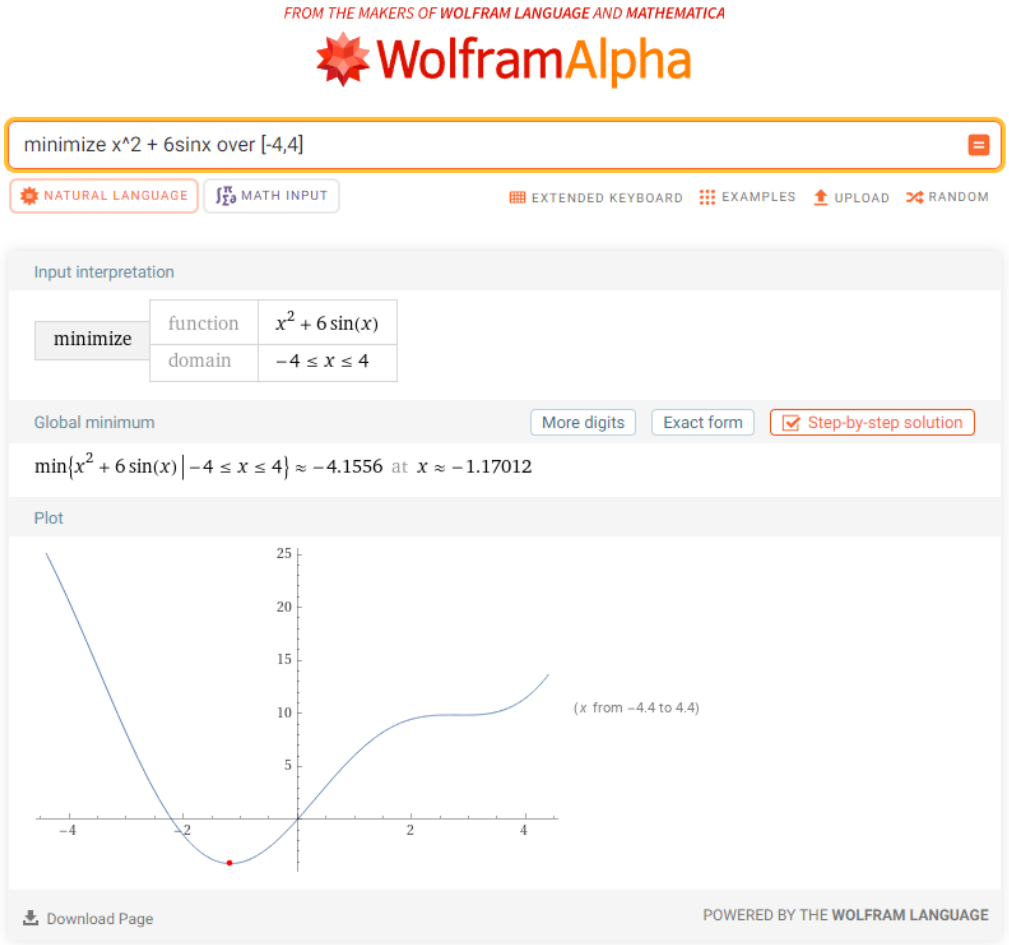
streaming data : loại dữ liệu được sinh ra và xử lý liên tục theo thời gian thực

Bảng 3: Tổng kết

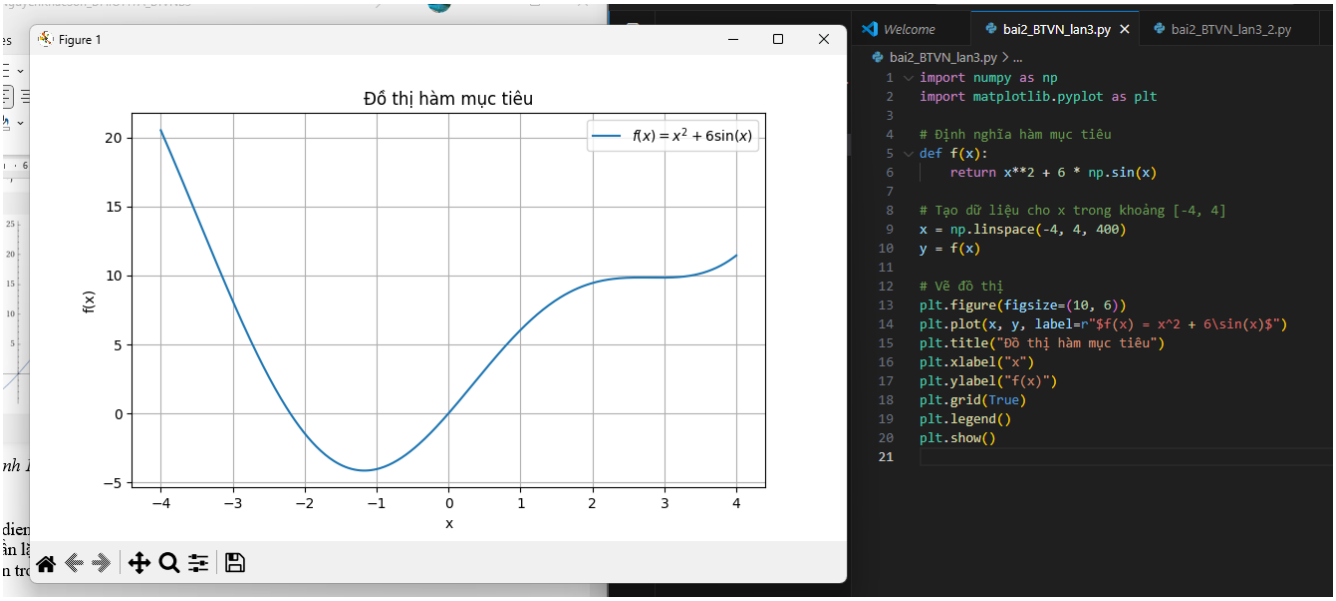
BGD:	SGD:	MBGD:
Tối ưu cho các bài toán nhỏ với yêu cầu chính xác cao	Hiệu quả với dữ liệu lớn và ứng dụng thực tế yêu cầu xử lý nhanh	Cân bằng giữa tốc độ và sự ổn định, phù hợp với đa số bài toán machine learning hiện đại.

Bài 2[1]: (Giải bài toán bằng bút và máy tính cầm tay)
Giả sử đã xây dựng được một hàm mục tiêu có dạng $\boldsymbol{f(x) = x^2 + 6\,sin\,x}$, giới hạn xét $\boldsymbol{x \in [-4, 4]}$.

a) Sử dụng một cách trực quan hóa dữ liệu để ước lượng điểm cực tiểu của hàm mục tiêu.
(Gợi ý: sử dụng Python hoặc công cụ vẽ hàm số trực tuyến của Wolfram tại https://www.wolframalpha.com/).

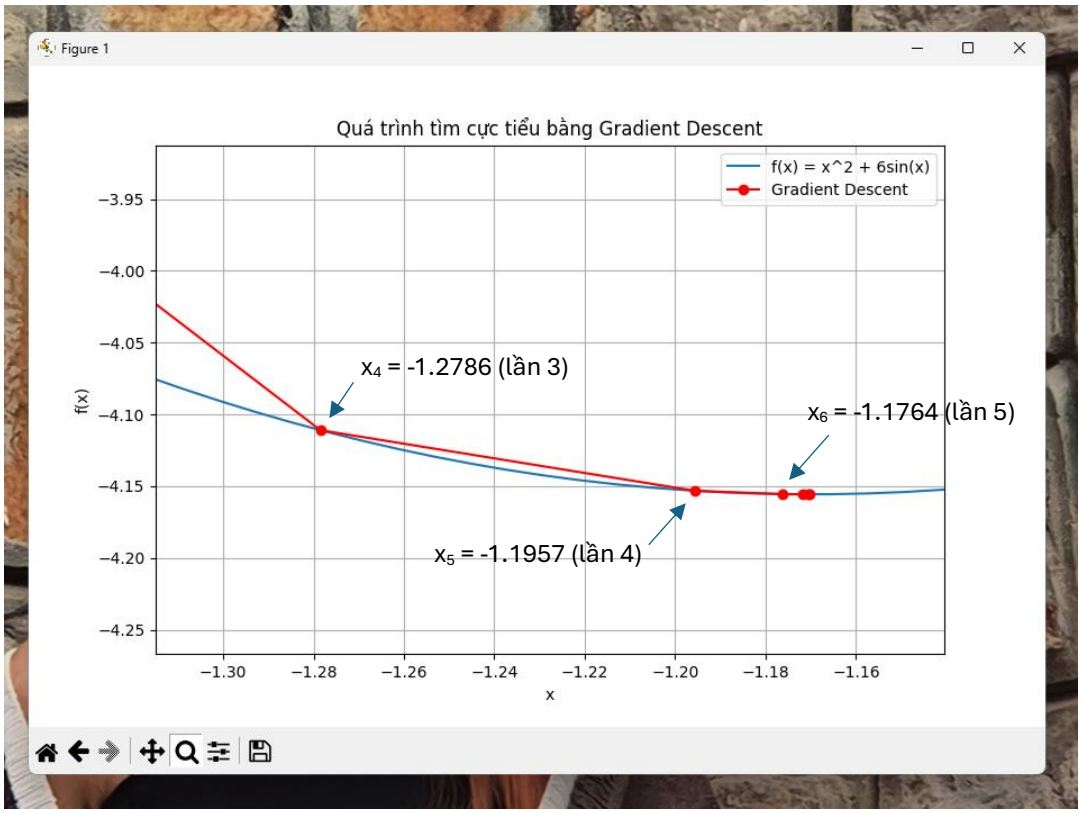


Hình 1: Sử dụng Wolfram Alpha để vẽ hàm mục tiêu



Hình 2: Sử dụng Python để vẽ hàm mục tiêu

b) Áp dụng thuật toán gradient descent để minh họa việc tìm cực tiểu của hàm mục tiêu $\boldsymbol{f(x)}$, thông qua việc thực hiện 10 lần lặp. Tốc độ học do sinh viên tự lựa chọn. Giá trị thiết lập ban đầu $\boldsymbol{x(0)}$ do sinh viên tự lựa chọn trong khoảng: $\boldsymbol{x \in [-4, -3]}$ hoặc $\boldsymbol{x \in [3, 4]}$.

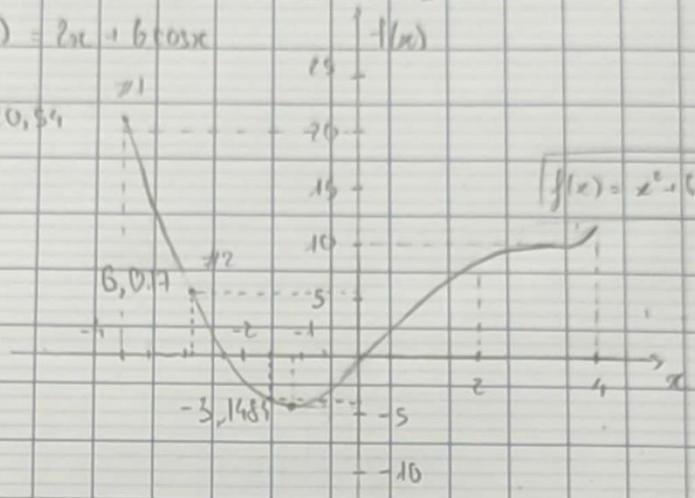


Ngày:

Bài 2:

a) $f(x) = x^2 + 6\sin x$ với $x \in [-4; 4]$

$f'(x) = 2x + 6\cos x$



b) Chọn $\begin{cases} \text{số lần} = \times 10 \\ x_0 = -4 \quad \text{(giá trị khởi tạo)} \\ \eta = 0,1 \quad \text{(tốc độ học)} \end{cases}$

"Rad MODE trong casio"

#1
$$\begin{cases} f(-4) = (-4)^2 + 6\sin(-4) = 20,5408 \\ f'(-4) = 2.(-4) + 6\cos(-4) = -11,9218 \\ x_2 = x_1 - \eta . f'(-4) = -2,8078 \end{cases}$$

#2
$$\begin{cases} f(-2,80..) = (-2,807)^2 + 6\sin(-2,807) = 6,0178 \\ f'(-2,80..) = 2.x_2 + 6\cos(x_2) = -11,284 \\ x_3 = x_2 - \eta . f'(x_2) = -1,6794 \end{cases}$$

#3
$$\begin{cases} f(-1,6794) = x_3^2 + 6\sin(x_3) = -3,1484 \\ f'(x_3) = 2x_3 + 6\cos(x_3) = -4,008 \\ x_4 = x_3 - \eta . f'(x_3) = -1,2786 \end{cases}$$

#4 $x_5 = -1,1957$ $\quad f(x_5) = -4,1111$

#5 $x_6 = -1,1764$ $\quad f(x_6) = -4,1531$

#6 $x_7 = -1,1717$ $\quad f(x_7) = -4,1555$

#7 $x_8 = -1,1705$ $\quad f(x_8) =$ "

#8 $x_9 = -1,1702$ $\quad f(x_9) =$

Bài 3: (Giải bài toán bằng cách lập trình)

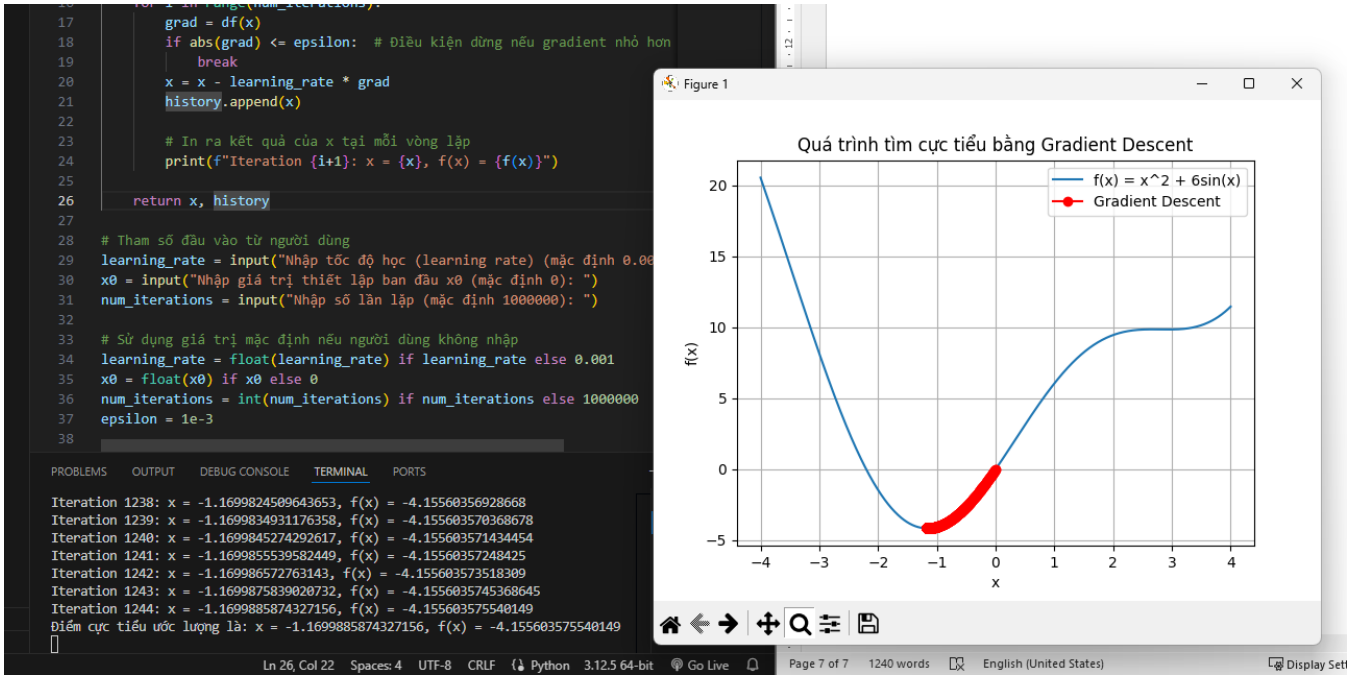
Sử dụng Python, lập một chương trình để giải quyết bài tập 2.

Tốc độ học, giá trị thiết lập ban đầu $x(0)$, và số lần lặp có thể do người dùng thay đổi. Chương trình có thể in ra kết quả của x tại mỗi vòng lặp hoặc vẽ kết quả. Mặc định của chương trình: tốc độ học $\eta = 0{,}001$; giá trị thiết lập ban đầu $x(0) = 0$; và số lần lặp là 1000000, có thể dừng lặp ngay khi $|f'(x)| \leq \varepsilon = 10^{-3}$.

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

# Hàm mục tiêu
def f(x):
    return x**2 + 6 * np.sin(x)

# Gradient của hàm mục tiêu
def df(x):
    return 2*x + 6*np.cos(x)

# Hàm Gradient Descent
def gradient_descent(f, df, x0, learning_rate, num_iterations, epsilon):
    x = x0
    history = [x0]
    for i in range(num_iterations):
        grad = df(x)
        if abs(grad) <= epsilon:  # Điều kiện dừng nếu gradient nhỏ hơn
ngưỡng epsilon
            break
        x = x - learning_rate * grad
        history.append(x)

        # In ra kết quả của x tại mỗi vòng lặp
        print(f"Iteration {i+1}: x = {x}, f(x) = {f(x)}")

    return x, history

# Tham số đầu vào từ người dùng
learning_rate = input("Nhập tốc độ học (learning rate) (mặc định 0.001): ")
x0 = input("Nhập giá trị thiết lập ban đầu x0 (mặc định 0): ")
num_iterations = input("Nhập số lần lặp (mặc định 1000000): ")

# Sử dụng giá trị mặc định nếu người dùng không nhập
learning_rate = float(learning_rate) if learning_rate else 0.001
x0 = float(x0) if x0 else 0
```

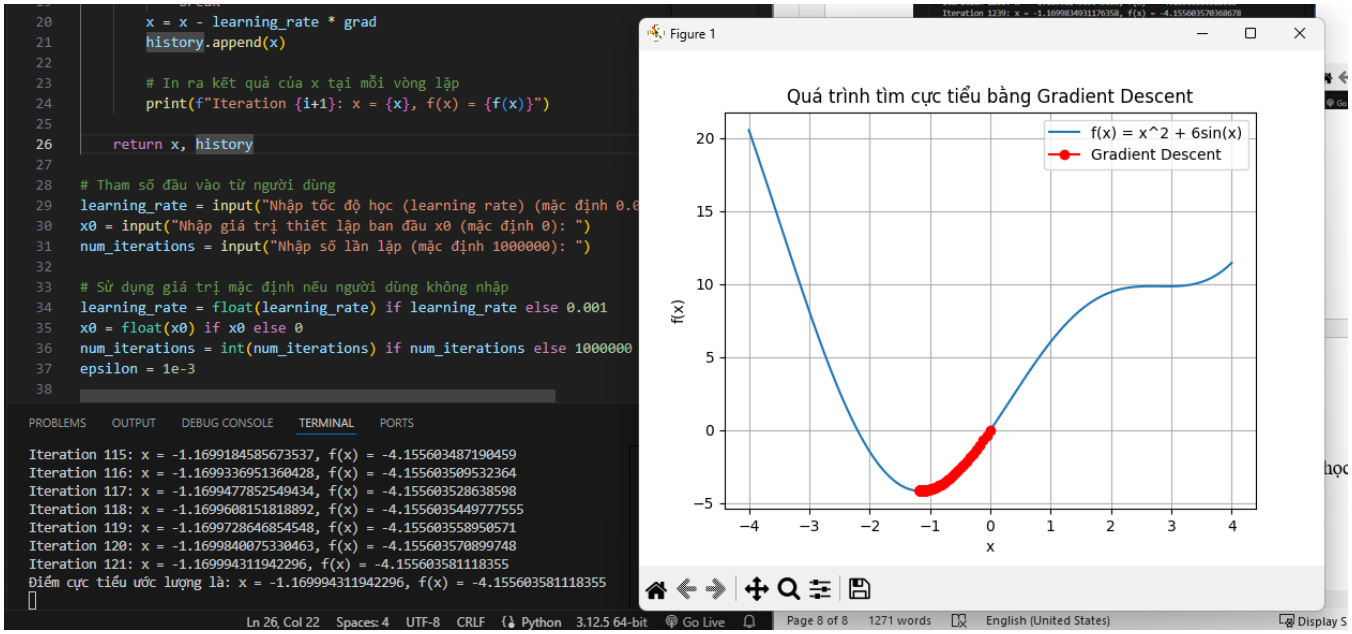
```python
num_iterations = int(num_iterations) if num_iterations else 1000000
epsilon = 1e-3

# Tìm điểm cực tiểu
xmin, history = gradient_descent(f, df, x0, learning_rate, num_iterations,
epsilon)

# In kết quả cuối cùng
print(f"Điểm cực tiểu ước lượng là: x = {xmin}, f(x) = {f(xmin)}")

# Trực quan hóa quá trình Gradient Descent
x = np.linspace(-4, 4, 1000)
y = f(x)
history = np.array(history)

plt.plot(x, y, label='f(x) = x^2 + 6sin(x)')
plt.plot(history, f(history), 'ro-', label='Gradient Descent')
plt.title('Quá trình tìm cực tiểu bằng Gradient Descent')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)
plt.legend()
plt.show()
```

Khi để giá trị mặc định, kết quả như sau (không nhập, để trống)



➔ ***1244 lần*** *tổng cộng để đến giá trị cực tiểu*

Khi giảm số lần lặp xuống còn 1000 và tăng tốc độ học lên 0.01



➔ *Khoảng 121 lần lặp cho đến cực tiểu của hàm*