MẠNG NEURON VÀ ỨNG DỤNG TRONG XỬ LÝ TÍN HIỆU

TS. TRẦN MẠNH CƯỜNG
TS. NGUYỄN THÚY BÌNH
BỘ MÔN KỸ THUẬT ĐIỆN TỬ
Email: thuybinh_ktdt@utc.edu.vn

Thuật toán Gradient Descent

1. Giới thiệu
2. Gradient descent cho với hàm 1 biến
3. Gradient descent cho hàm nhiều biến

1. Giới thiệu

Quá trình học của một mạng neuron có thể được thực hiện bởi

1. Tạo/xóa bỏ ra một khớp nối
2. Thay đổi giá trị trọng số của các khớp
3. Thay đổi hàm kích thích của các neuron trong mạng
4. Thay đổi giá trị bias
5. Thêm/xóa bỏ các tế bào neuron

Mục tiêu

1. Giảm thiểu số lần học
2. Tối thiểu hoá hàm mất mát

Giải pháp

1. Thay đổi phù hợp **w**
2. Xác định hệ số học phù hợp

1. Giới thiệu

- Xác định ma trận w thế nào khi số đầu vào lớn ?

$$w(n+1) = w(n) + \eta\left[d(n) - y(n)\right]x(n)$$

- Cần bao nhiêu bước tính ?
- Tối ưu các bước tính

Xây dựng hàm chi phí (cost function): ξ(w)

SSE $\boldsymbol{SSE = \sum_{i=1}^{n}(e^2)}$

Sum Squared Error

MSE $\boldsymbol{MSE = \frac{SSE}{n}}$

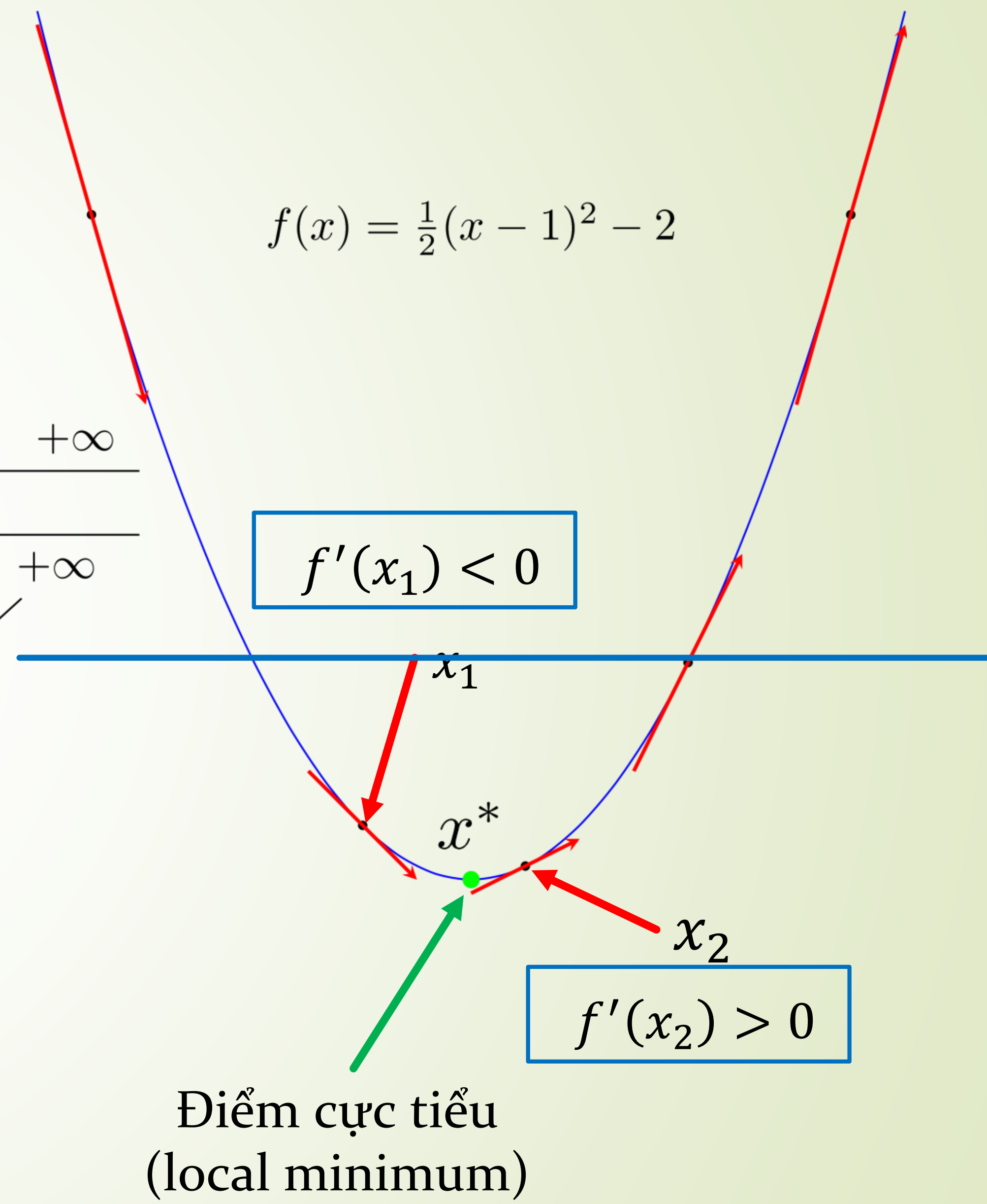
Mean Squared Error

Tối thiểu hoá hàm chi phí ξ(w)

1. Giới thiệu

- Điểm local minimum x^*là điểm có đạo hàm $f'(x) = 0$
- Đường tiếp tuyến với đồ thị hàm số đó tại 1 điểm bất kỳ có hệ số góc = đạo hàm của hàm số tại điểm đó

x	$-\infty$		1		$+\infty$
$f'(x)$		$-$	0	$+$	
$f(x)$	$+\infty$	$\searrow$	-2	$\nearrow$	$+\infty$



- **Bài toán Tối ưu** → đi tìm giá trị nhỏ nhất(lớn nhất) của một hàm số → phức tạp, đôi khi bất khả thi.

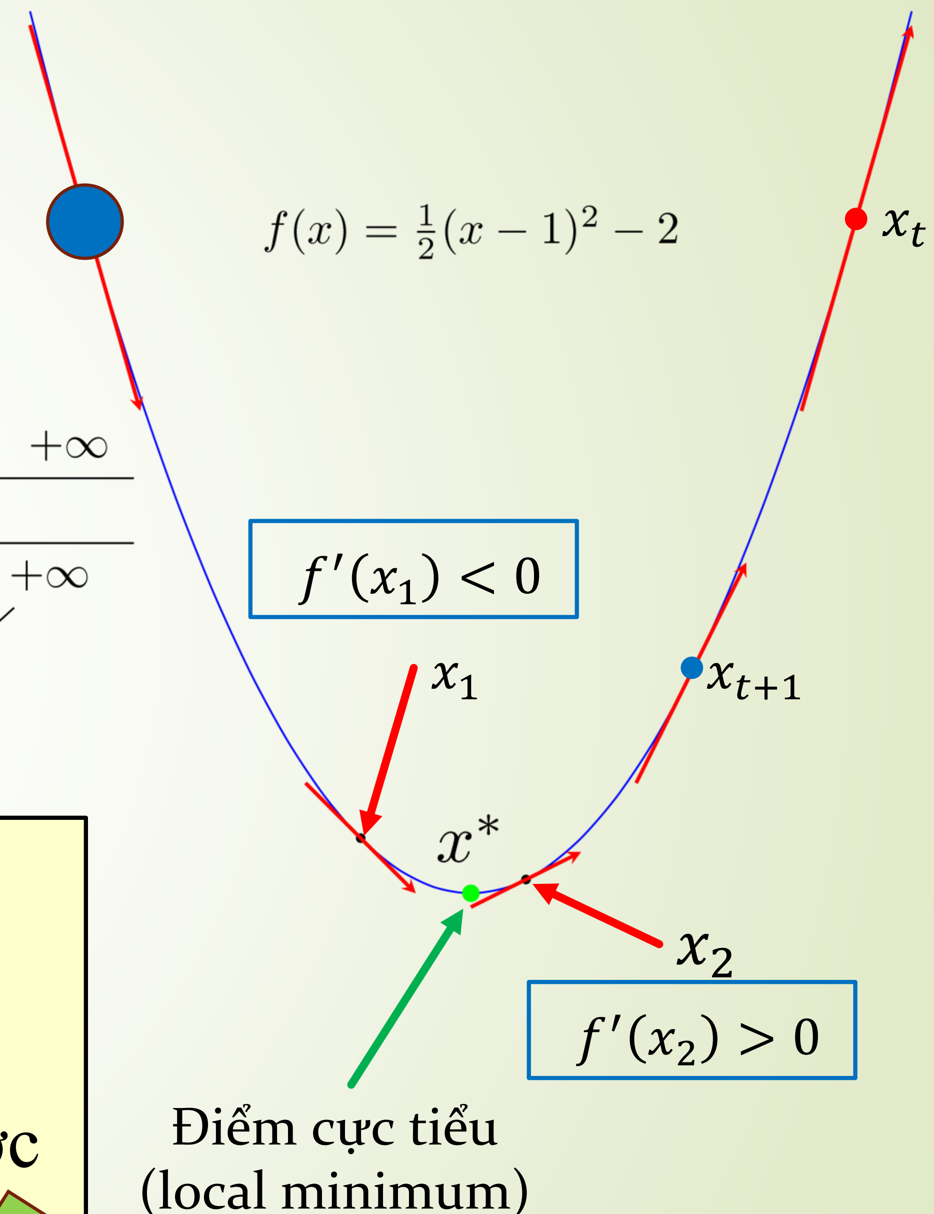
Tìm điểm cực tiểu (local minimum)

2. GD cho hàm một biến

x_t: điểm tìm được sau t vòng lặp

Đưa x_t về càng gần x^* càng tốt

x	$-\infty$		1		$+\infty$
$f'(x)$		$-$	0	$+$	
$f(x)$	$+\infty$	$\searrow$	-2	$\nearrow$	$+\infty$



- $f'(x_t) > 0$: x_t nằm bên phải của x^* (và ngược lại)$\longrightarrow$ cần di chuyển x_t về phía bên trái:

$$x_{t+1} = x_t + \Delta$$

- x_t càng xa x^* thì $f'(x_t)$ càng lớn hơn 0 (và ngược lại):

$$\Delta = -\eta f'(x_t)$$

$$\boxed{x_{t+1} = x_t - \eta f'(x_t)}$$

Learning rate (>0)

2. GD cho hàm một biến

$f(x) = x^2 + 5\sin(x) \Rightarrow f'(x) = 2x + 5\cos(x)$

$$\boxed{x_{t+1} = x_t - \eta(2x + 5\cos(x))}$$

Chương trình Python

```python
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import math
import numpy as np
import matplotlib.pyplot as plt
```

!pip install latex
!apt install texlive-fonts-recommended texlive-fonts-extra cm-super dvipng

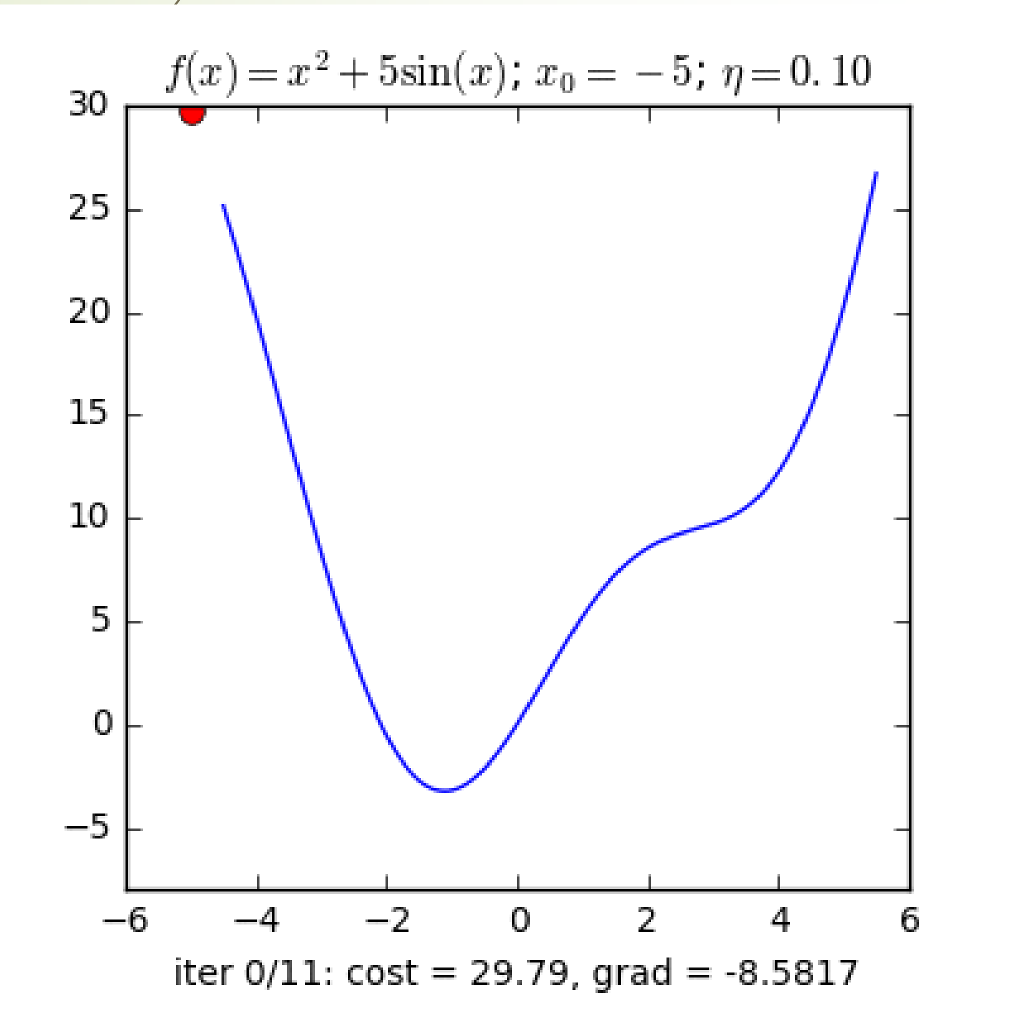
2. GD cho hàm một biến

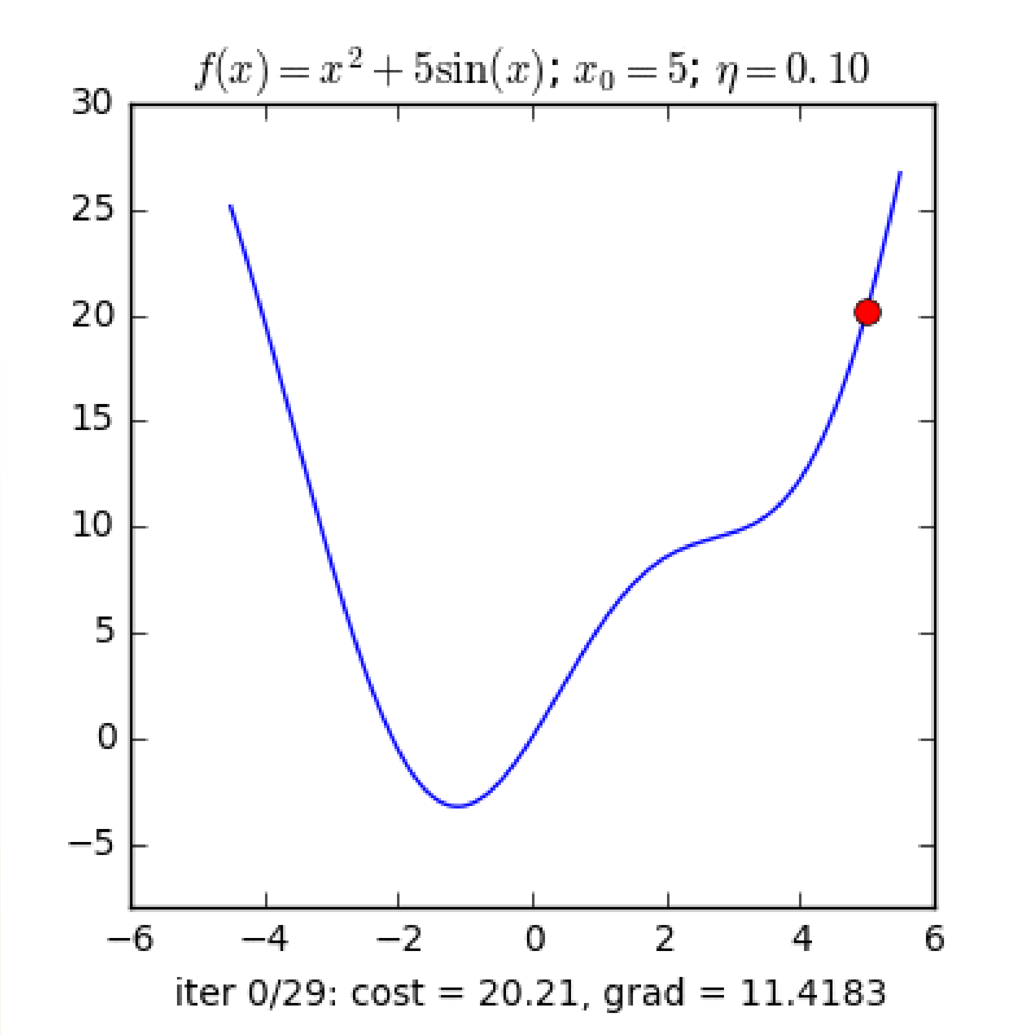
```python
def grad(x):
    return 2*x+ 5*np.cos(x)

def cost(x):
    return x**2 + 5*np.sin(x)

def myGD1(eta, x0):
    x = [x0]
    for it in range(100):
        x_new = x[-1] - eta*grad(x[-1])
        if abs(grad(x_new)) < 1e-3:
            break
        x.append(x_new)
    return (x, it)
```

2. GD cho hàm một biến

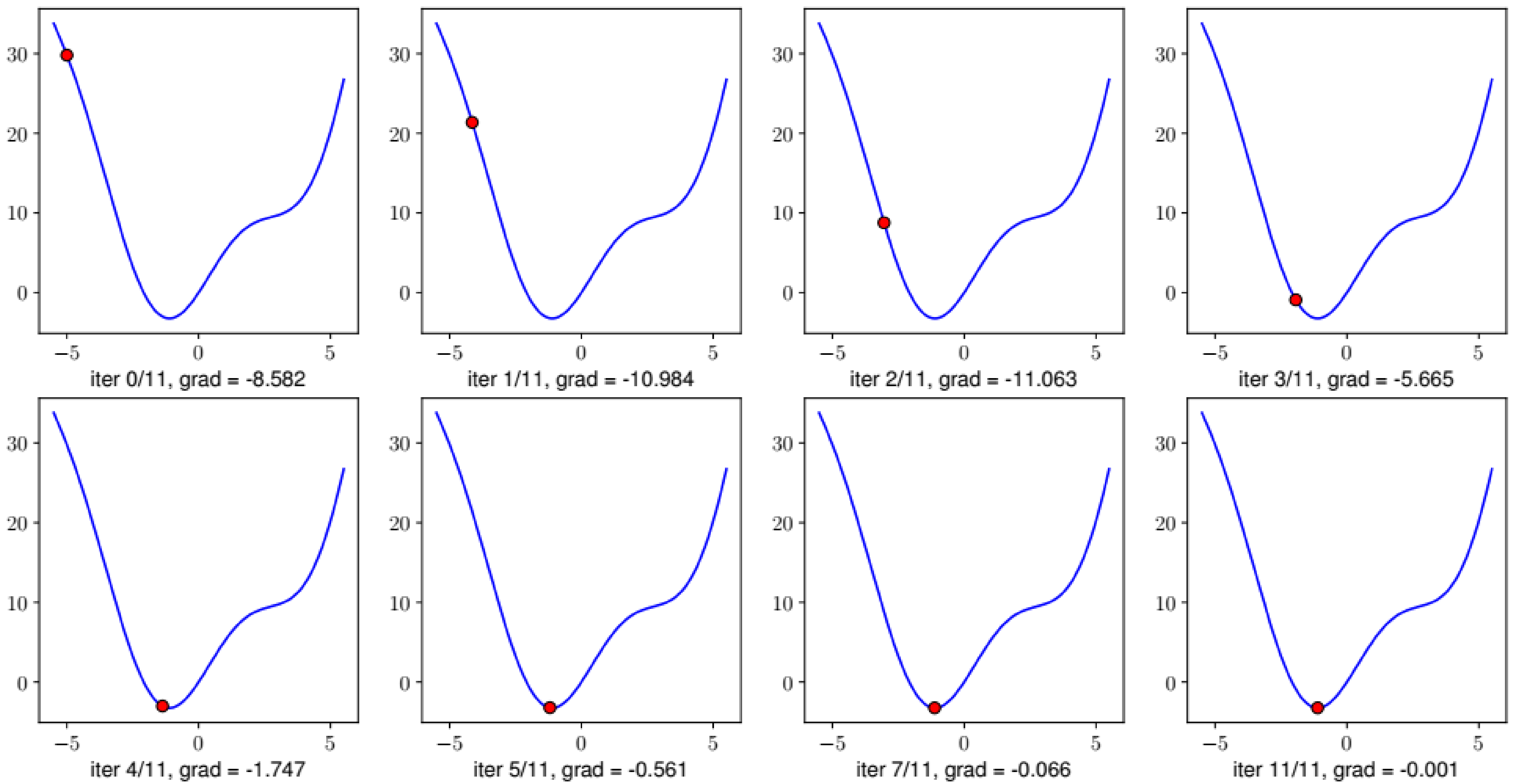
```
(x1, it1) = myGD1(.1, -5)
(x2, it2) = myGD1(.1, 5)
print('Solution x1 = %f, cost = %f, obtained after %d iterations'%(x1[-1], cost(x1[-1]), it1))
print('Solution x2 = %f, cost = %f, obtained after %d iterations'%(x2[-1], cost(x2[-1]), it2))
```

```
Solution x1 = -1.110667, cost = -3.246394, obtained after 11 iterations
Solution x2 = -1.110341, cost = -3.246394, obtained after 29 iterations
```





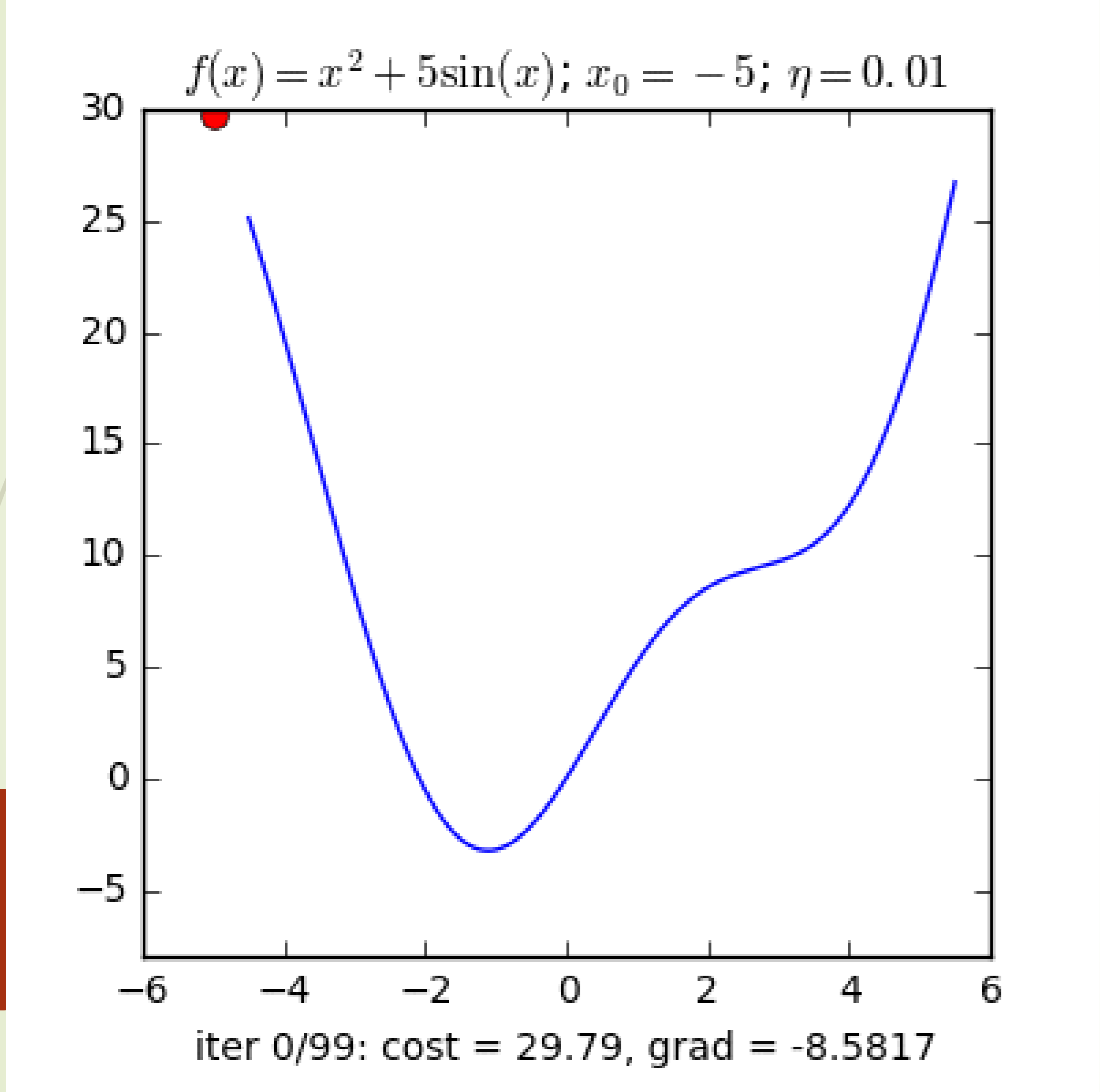
2. GD cho hàm một biến

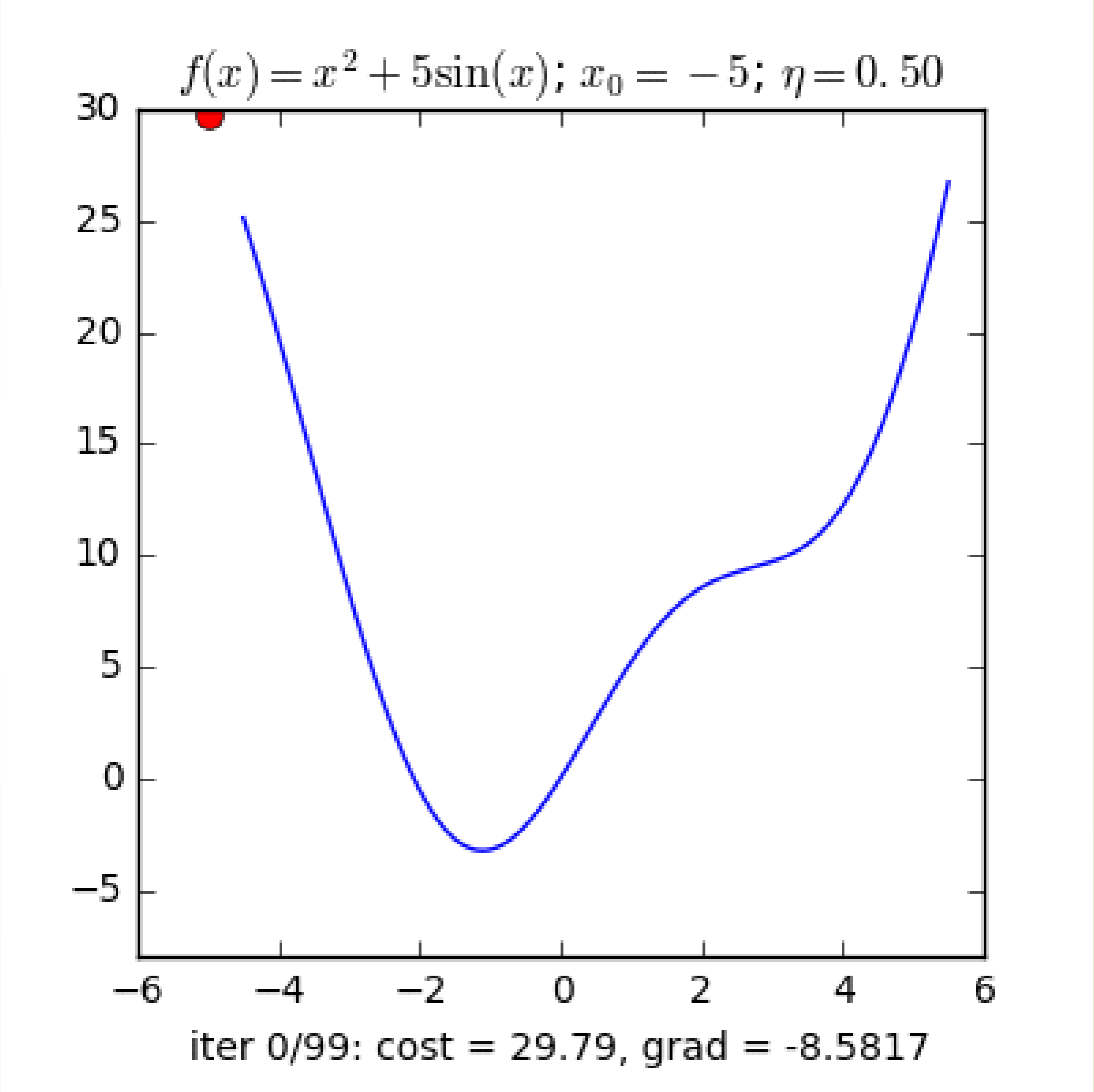


Hình 12.2: Nghiệm tìm được qua các vòng lặp với $x_0 = -5, \eta = 0.1$

2. GD cho hàm một biến

Learning rate khác nhau





Learning rate nhỏ⟶ tốc độ hội tụ chậm. Nếu tính toán phức tạp ⟶ ảnh hưởng tới tốc độ của thuật toán→ không đến được đích.

Learning rate lớn ⟶ tiến đến gần đích nhanh. Tuy nhiên, thuật toán không hội tụ được do bước nhảy lớn⟶ quanh quẩn ở đích

3. GD cho hàm nhiều biến

- **Mục tiêu:**
  - Tìm giá trị nhỏ nhất (global minimum) của hàm $f(\theta)$
  - θ là một vector, tập hợp các tham số của mô hình cần tối ưu
  - Cập nhật trọng số của mô hình:

$$\boxed{\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta)}$$

luôn luôn đi ngược hướng với đạo hàm

3. GD cho hàm nhiều biến

Bài toán Hồi quy tuyến tính (Linear Regression):

$$y_i = f(\mathrm{x}_i) = \sum_{j=0}^{m} w_j x_{i,j} = \mathrm{w}^T \overline{\mathrm{x}_i} = \overline{\mathrm{x}_i}^T \mathrm{w}$$

$$\overline{\mathrm{x}_i} = \begin{bmatrix} 1 \\ x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,m} \end{bmatrix} \quad \mathrm{w} = \begin{bmatrix} w_o \\ w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \quad \mathrm{y} = \begin{bmatrix} y_o \\ y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{bmatrix} \quad \mathrm{d} = \begin{bmatrix} d_o \\ d_1 \\ d_2 \\ \vdots \\ d_{N-1} \end{bmatrix}$$

Đầu vào **Vector trọng số** **Đầu ra** **Đầu ra mong muốn**

- Hàm mất mát:

$$\mathcal{L}(\mathrm{w}) = \frac{1}{2N} \|\mathrm{d} - \mathrm{y}\|_2^2 = \frac{1}{2N} [(d_1 - y_1)^2 + (d_2 - y_2)^2 + \cdots]$$

3. GD cho hàm nhiều biến

Phương pháp Đại số

Bài toán Hồi quy tuyến tính (Linear Regression):

- Hàm mất mát:

$$\mathcal{L}(\mathrm{w}) = \frac{1}{2N}\|\mathrm{d} - \mathrm{y}\|_2^2 = \frac{1}{2N}\|\mathrm{d} - \bar{X}^T\mathrm{w}\|_2^2$$

Đạo hàm

$$\boxed{\nabla_{\mathrm{w}}\mathcal{L}(\mathrm{w}) = -\frac{1}{N}\bar{X}(d - \bar{X}^T\mathrm{w}) = \frac{1}{N}\bar{X}(\bar{X}^T\mathrm{w} - \mathrm{d})}$$

Mục tiêu

$$\mathrm{w}^* = \mathbf{arg}\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

$\boxed{\nabla_{\mathrm{w}}\mathcal{L}(\mathrm{w}) = 0 \Longrightarrow \bar{X}\bar{X}^T\mathrm{w} = \bar{X}\mathrm{d}}$ → $\boxed{\mathbf{A}.\mathrm{w} = \mathbf{b}}$ → $\boxed{\mathrm{w} = \boldsymbol{A}^{-1}\mathbf{b}}$

3. GD cho hàm nhiều biến

Ví dụ Python

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
np.random.seed(2)
```

Tạo chuỗi ngẫu nhiên xác định

```
X = np.random.rand(1000, 1)
y = 4 + 3 * X + .2*np.random.randn(1000, 1) # noise added
```

Tạo 1000 điểm dữ liệu gần với đường thẳng: $y = 4 + 3x$

```
one = np.ones((X.shape[0],1))
Xbar = np.concatenate((one, X), axis = 1)
```

A

B

$$\underbrace{\begin{bmatrix} 1 & X^T \end{bmatrix}}_{\bar{X}^T}$$

3. GD cho hàm nhiều biến

Ví dụ Python

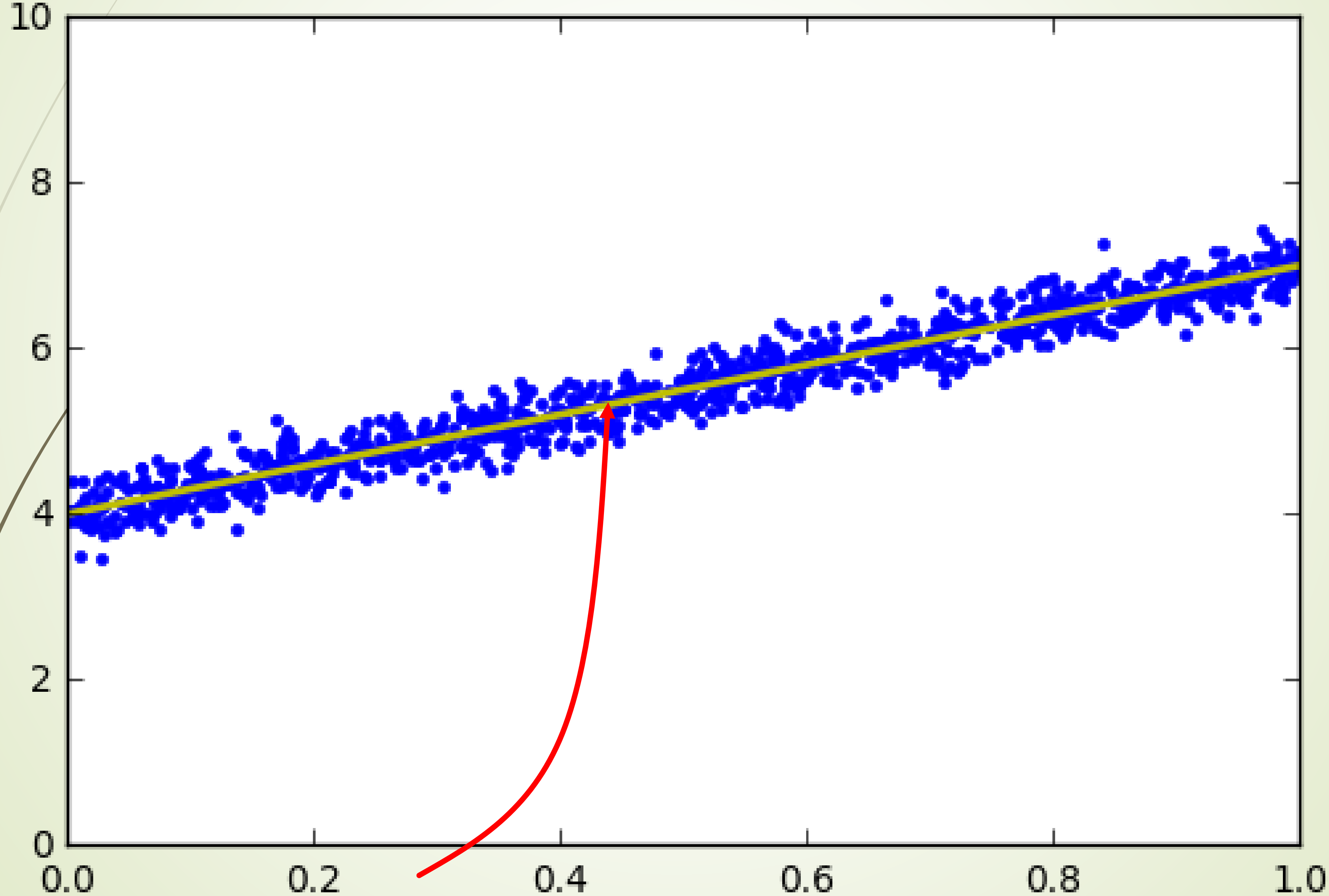
```
A = np.dot(Xbar.T, Xbar)
b = np.dot(Xbar.T, y)
w_lr = np.dot(np.linalg.pinv(A), b)
print('Solution found by formula: w = ',w_lr.T)

# Display result
w = w_lr
w_0 = w[0][0]
w_1 = w[1][0]
x0 = np.linspace(0, 1, 2, endpoint=True)
y0 = w_0 + w_1*x0

# Draw the fitting line
plt.plot(X.T, y.T, 'b.')      # data
plt.plot(x0, y0, 'y', linewidth = 2)   # the fitting line
plt.axis([0, 1, 0, 10])
plt.show()
```

$$\begin{bmatrix} w\_0 \\ w\_1 \end{bmatrix}$$

```
Solution found by formula: w =  [[ 4.00305242  2.99862665]]
```

3. GD cho hàm nhiều biến

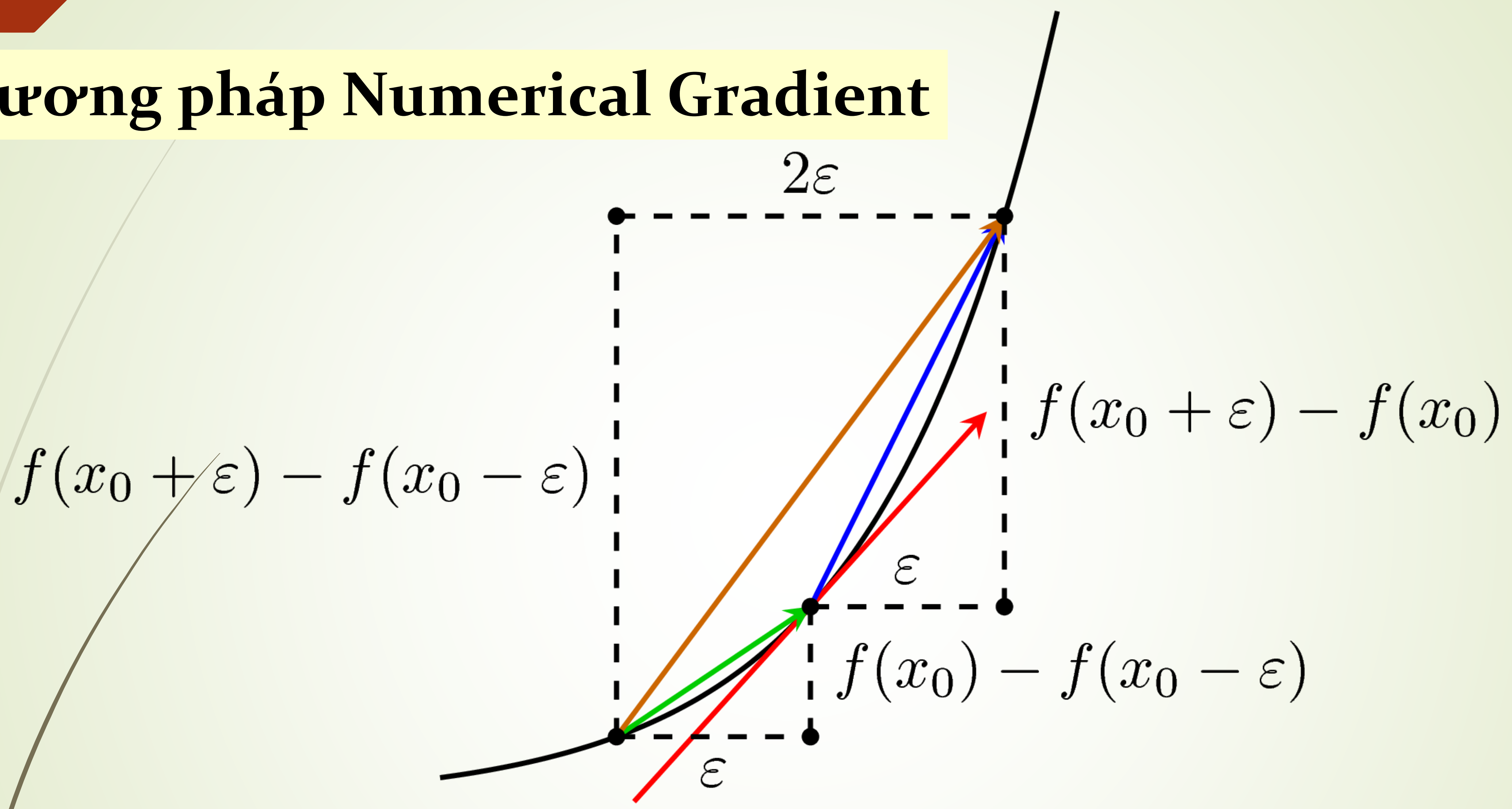
Hiển thị dữ liệu



$$y = 4 + 2.998x$$

3. GD cho hàm nhiều biến

Phương pháp Numerical Gradient



$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon)-f(x)}{\varepsilon}$$

$$f'(x) \approx \frac{f(x+\varepsilon)-f(x-\varepsilon)}{2\varepsilon}$$

3. GD cho hàm nhiều biến

Khai triển Taylor

$$f(x+\varepsilon) \approx f(x) + f'(x)\varepsilon + \frac{f''(x)}{2}\varepsilon^2 + \dots$$

$$f(x-\varepsilon) \approx f(x) - f'(x)\varepsilon + \frac{f''(x)}{2}\varepsilon^2 - \dots$$

$$f'(x) \approx \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon}$$

3. GD cho hàm nhiều biến

Định nghĩa hàm chi phí (cost function) và gradient

```
def grad(w):
    N = Xbar.shape[0]
    return 1/N * Xbar.T.dot(Xbar.dot(w) - y)

def cost(w):
    N = Xbar.shape[0]
    return .5/N*np.linalg.norm(y - Xbar.dot(w), 2)**2;
```

$$\nabla_{\mathrm{w}}\mathcal{L}(\mathrm{w}) = -\frac{1}{N}\bar{X}^T(d - \bar{X}\mathrm{w}) = \frac{1}{N}\bar{X}^T(\bar{X}\mathrm{w}-\mathrm{d})$$

Số lượng mẫu = số hàng của $\bar{X}$

Chuẩn L2

$$y = w_o x_o + w_1 x_1 + \cdots . + w_m x_m$$

$$y = \mathrm{w}^T\mathrm{x} = \mathrm{x}^T\mathrm{w}$$

$$\mathcal{L}(\mathrm{w}) = \frac{1}{2N}\|\mathrm{d} - \mathrm{y}\|_2^2 = \frac{1}{2N}\|\mathrm{d} - \bar{X}^T\mathrm{w}\|_2^2$$

3. GD cho hàm nhiều biến

```
def numerical_grad(w, cost):
    eps = 1e-4
    g = np.zeros_like(w)
    for i in range(len(w)):
        w_p = w.copy()
        w_n = w.copy()
        w_p[i] += eps
        w_n[i] -= eps
        g[i] = (cost(w_p) - cost(w_n))/(2*eps)
    return g

def check_grad(w, cost, grad):
    w = np.random.rand(w.shape[0], w.shape[1])
    grad1 = grad(w)
    grad2 = numerical_grad(w, cost)
    return True if np.linalg.norm(grad1 - grad2) < 1e-6 else False

print( 'Checking gradient...', check_grad(np.random.rand(2, 1), cost, grad))
```

Tạo mảng gồm các phần tử bằng 0 có kích thước bằng với mảng w

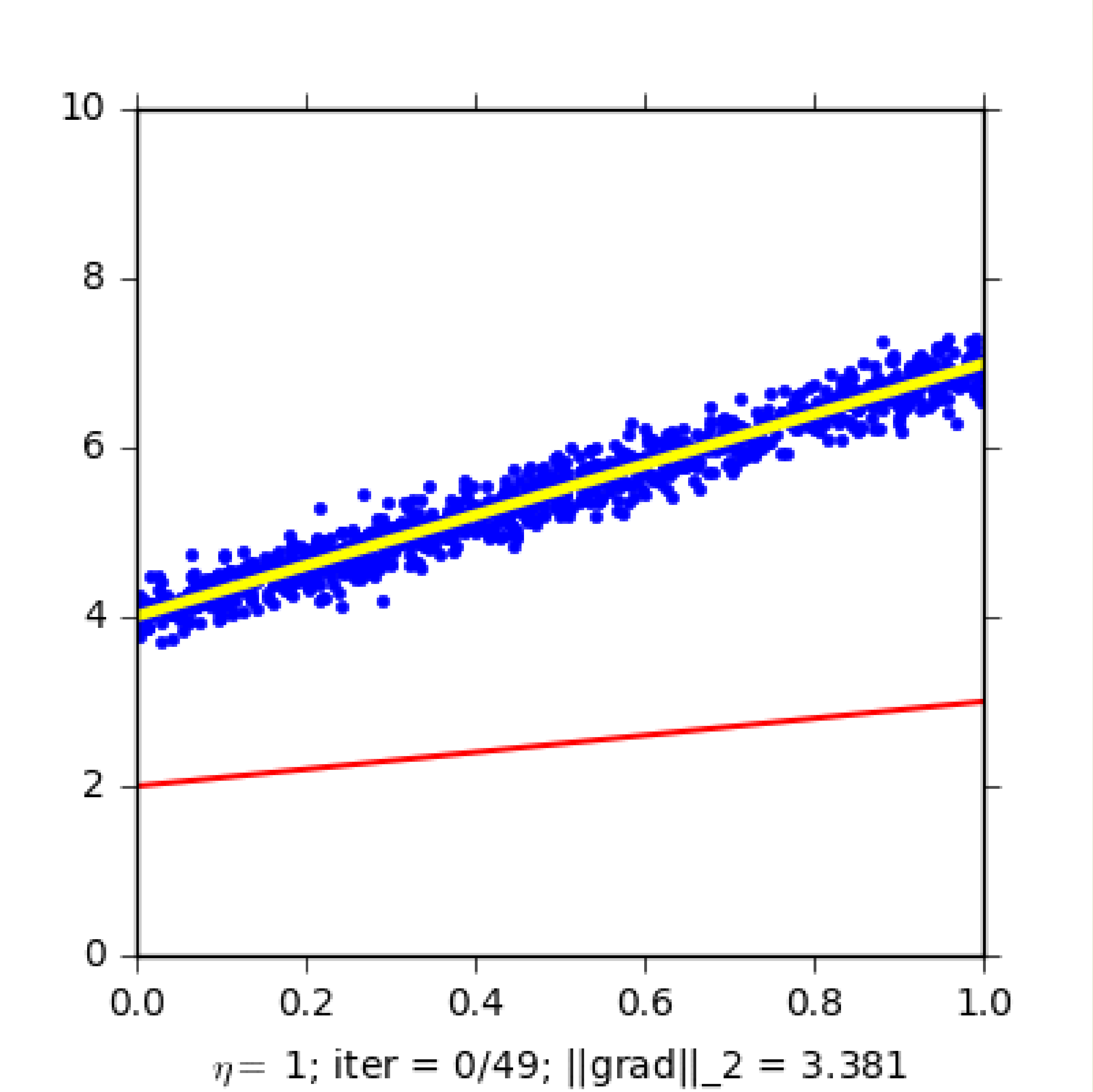
```
Checking gradient... True
```

3. GD cho hàm nhiều biến

```
def myGD(w_init, grad, eta):
    w = [w_init]
    for it in range(100):
        w_new = w[-1] - eta*grad(w[-1])
        if np.linalg.norm(grad(w_new))/len(w_new) < 1e-3:
            break
        w.append(w_new)
    return (w, it)

w_init = np.array([[2], [1]])
(w1, it1) = myGD(w_init, grad, 1)
print('Solution found by GD: w = ', w1[-1].T, ',\nafter %d iterations.' %(it1+1))
```

```
Solution found by GD: w =  [[ 4.01780793  2.97133693]] ,
after 49 iterations.
```

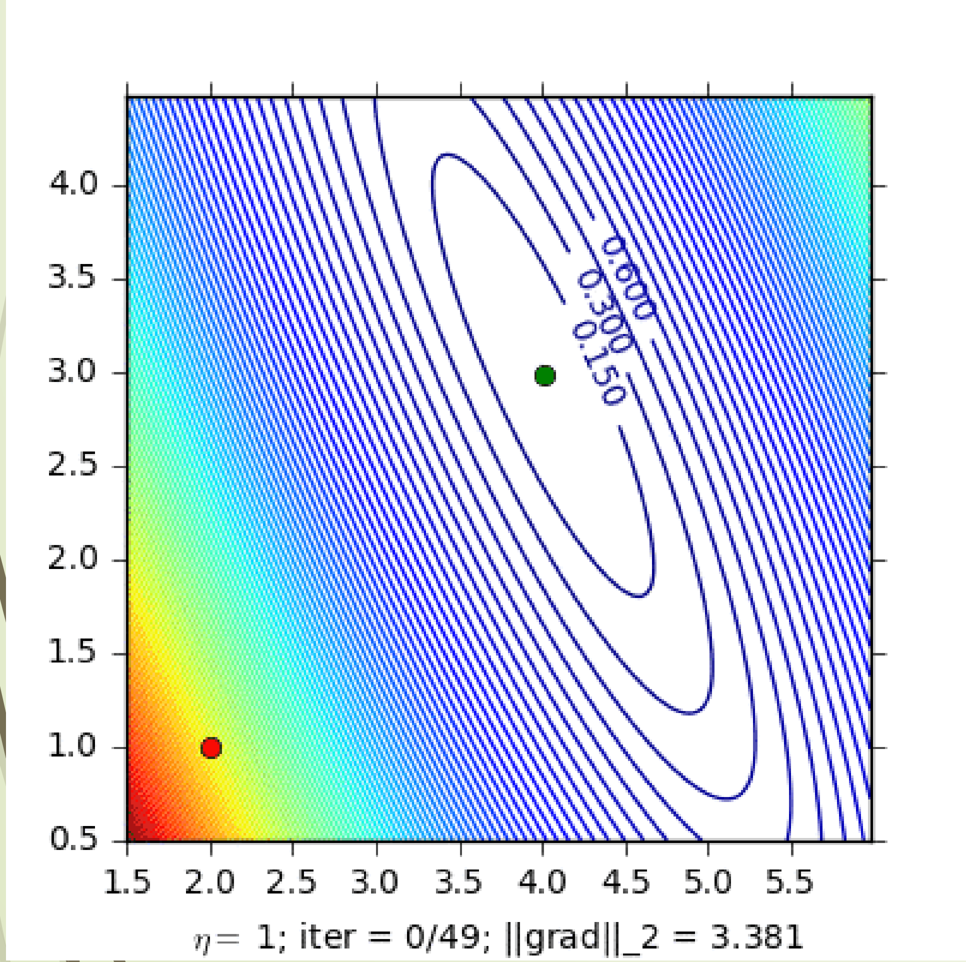
3. GD cho hàm nhiều biến

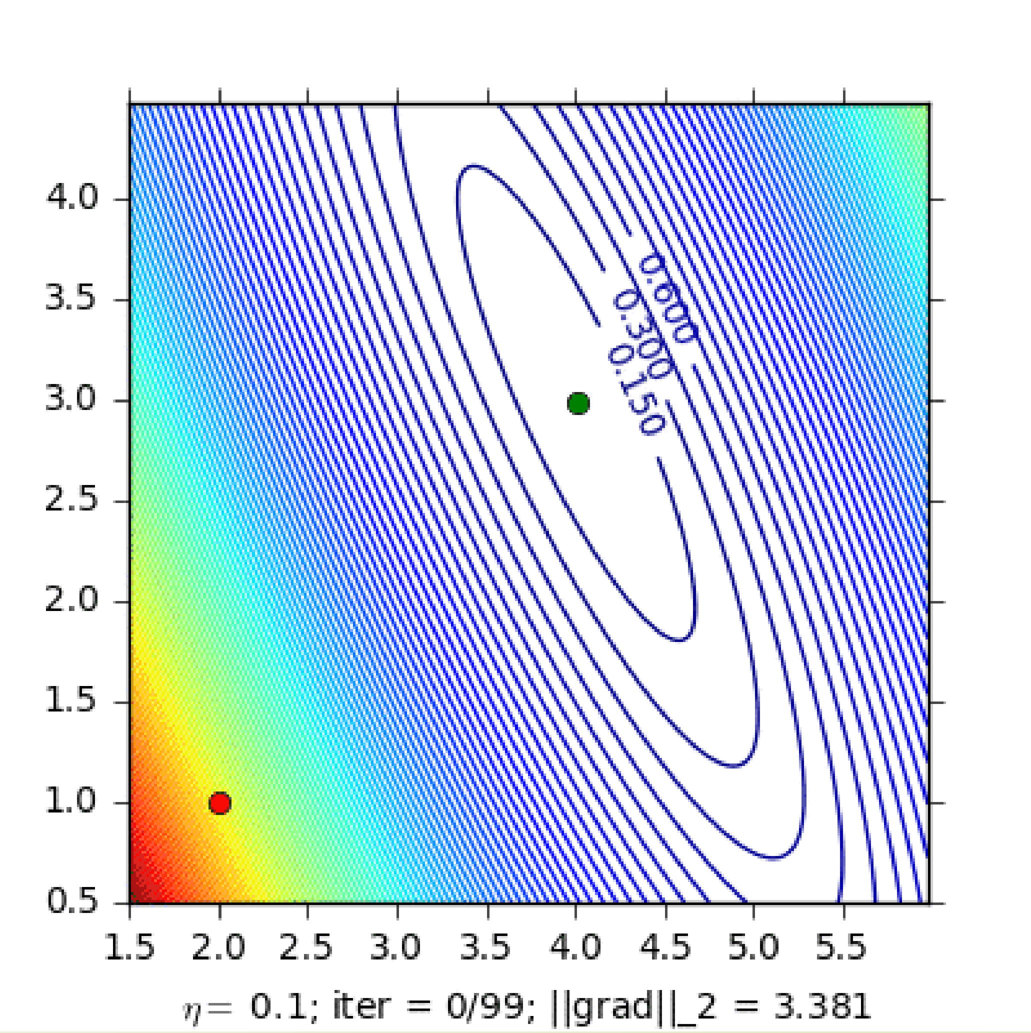


Sau 49 vòng lặp, thuật toán đã hội tụ với một nghiệm khá gần với nghiệm tìm được theo công thức

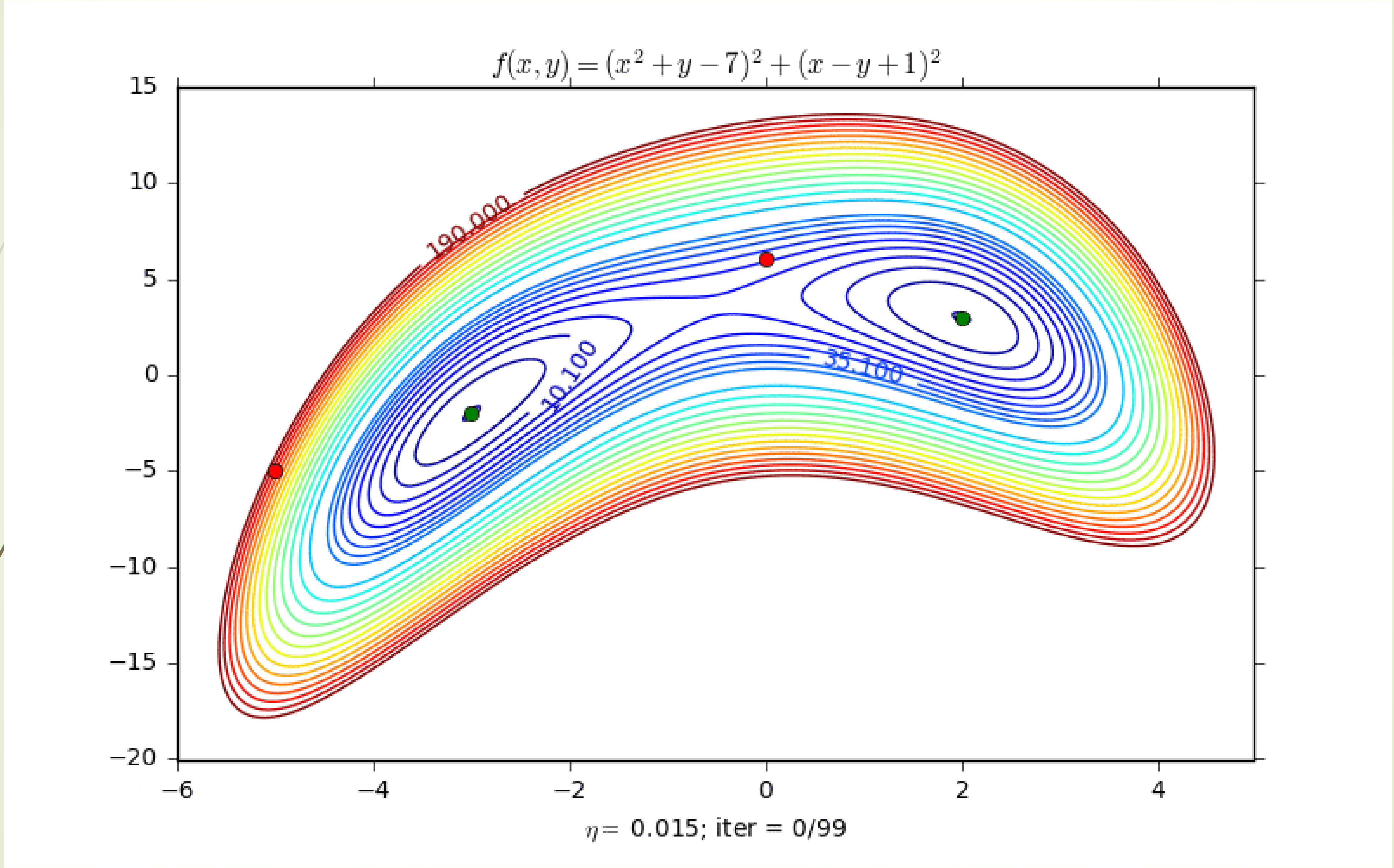
3. GD cho hàm nhiều biến

- **Đường đồng mức (level sets)**
  - Các điểm trên cùng một vòng, hàm mất mát có giá trị như nhau
  - Các vòng màu xanh có giá trị thấp
  - Các vòng tròn màu đỏ phía ngoài có giá trị cao hơn



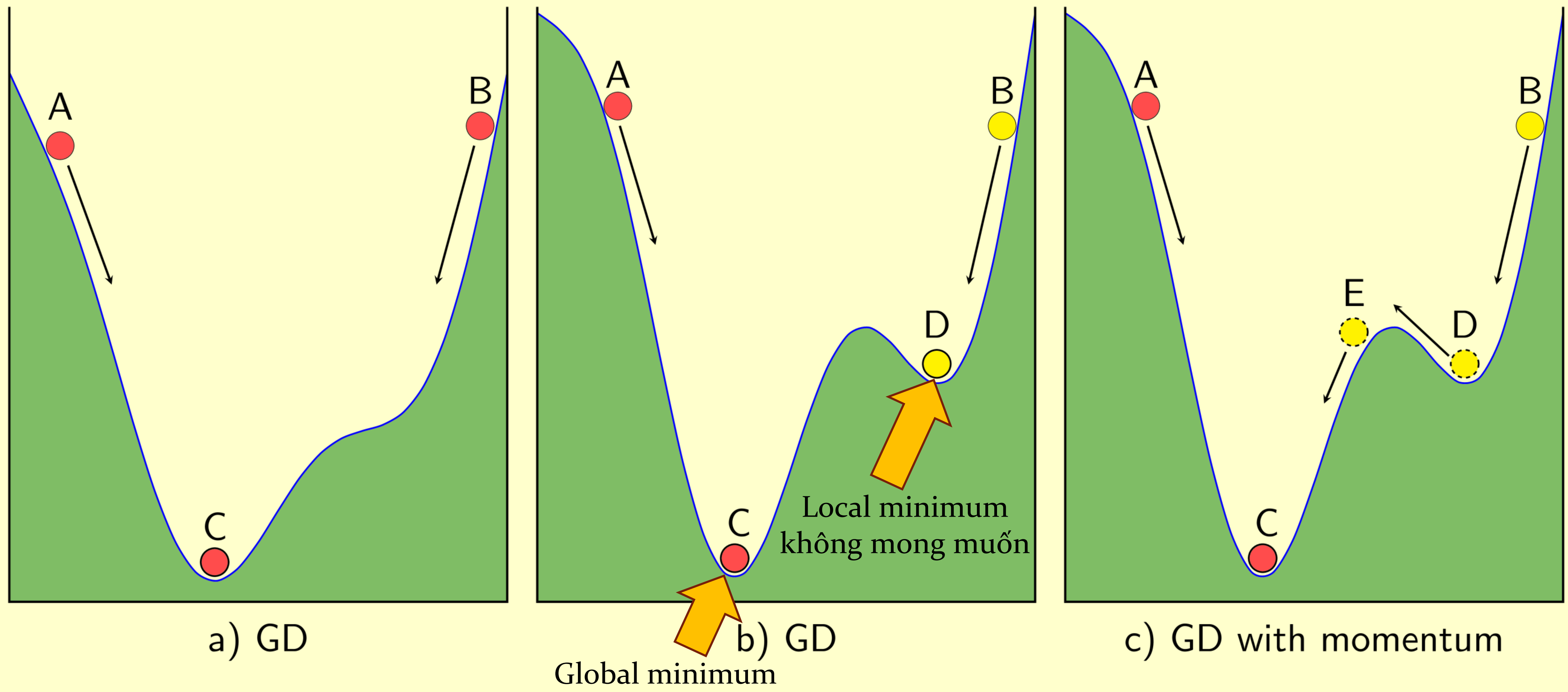


3. GD cho hàm nhiều biến



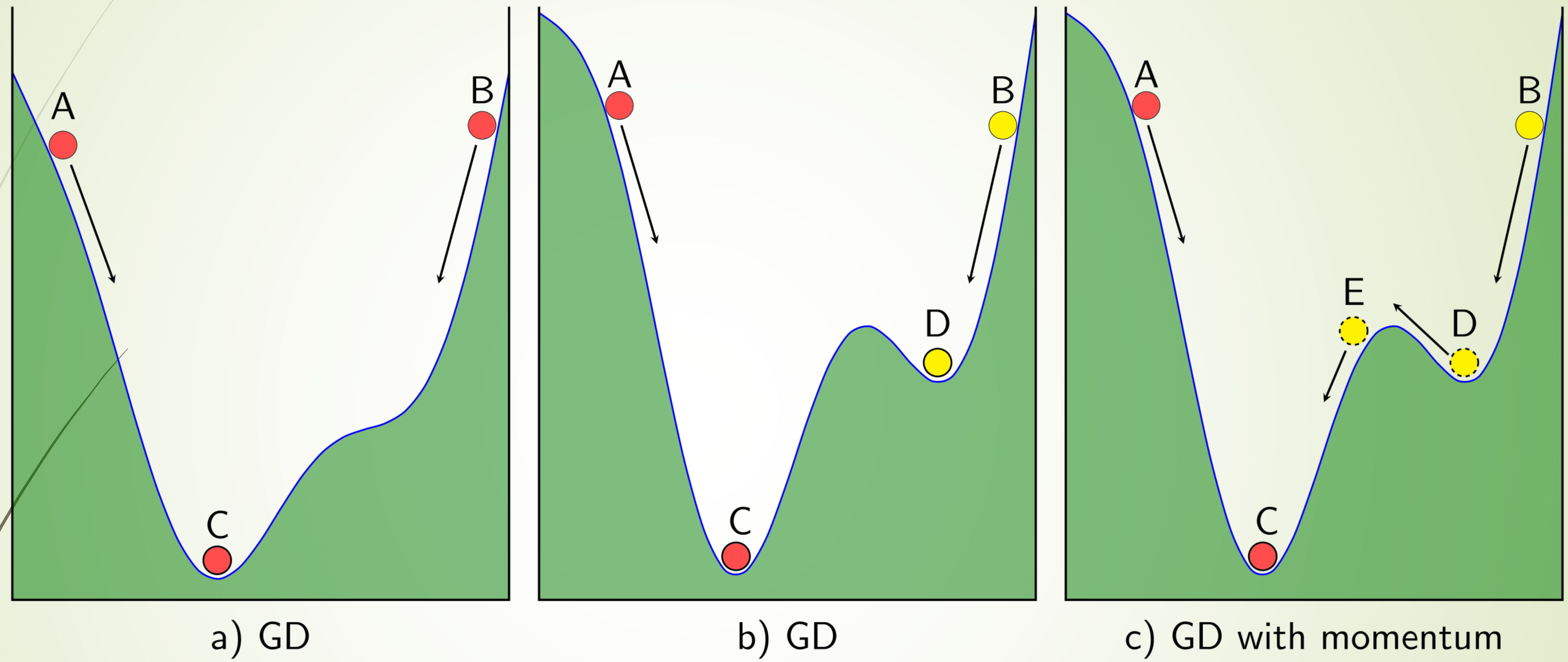
Nghiệm cuối cùng của Gradient Descent phụ thuộc rất nhiều vào điểm khởi tạo và learning rate

3. Momentum



- Nếu **vận tốc ban đầu** của bi khi ở điểm B **đủ lớn** → bi **lăn đến điểm D**, theo *đà* → bi có thể tiếp tục **di chuyển lên dốc phía bên trái của D**.
- Nếu giả sử **vận tốc ban đầu lớn hơn nữa**, bi có thể **vượt dốc tới điểm E** rồi **lăn xuống C** (hình c)

3. Momentum



- Vị trí mới của viên bi: $\theta_{t+1} = \theta_t - v_t$
- v_t vừa mang thông tin về độ dốc (đạo hàm) vừa mang thông tin về vận tốc:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta_{t+1} = \theta_t - \gamma v_{t-1} - \eta \nabla_\theta J(\theta)$$

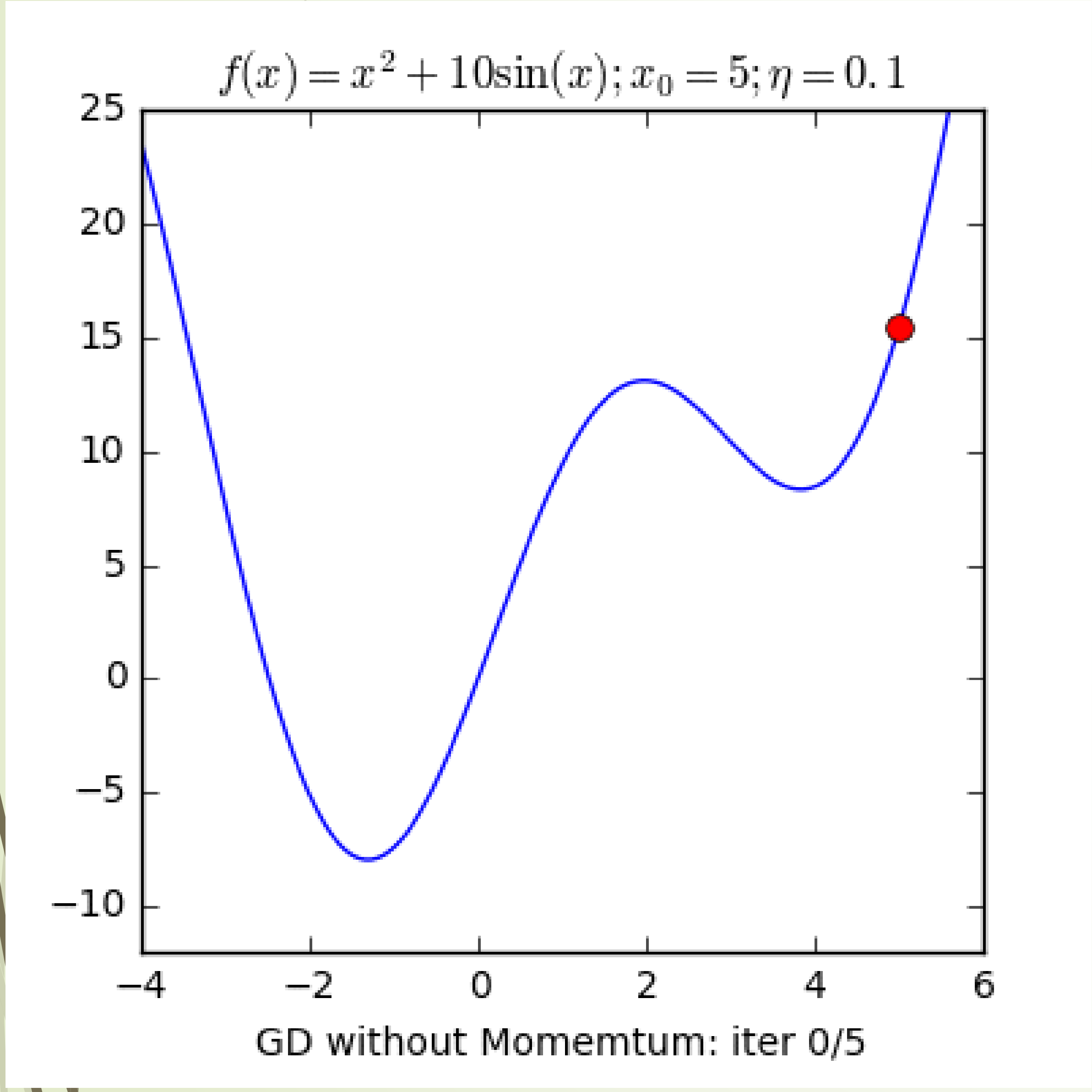
3. Momentum

- **Ví dụ:** Tìm giá trị nhỏ nhất của hàm số $f(x) = x^2 + 10\sin(x)$
  - $f'(x) = 2x + 10\cos(x)$

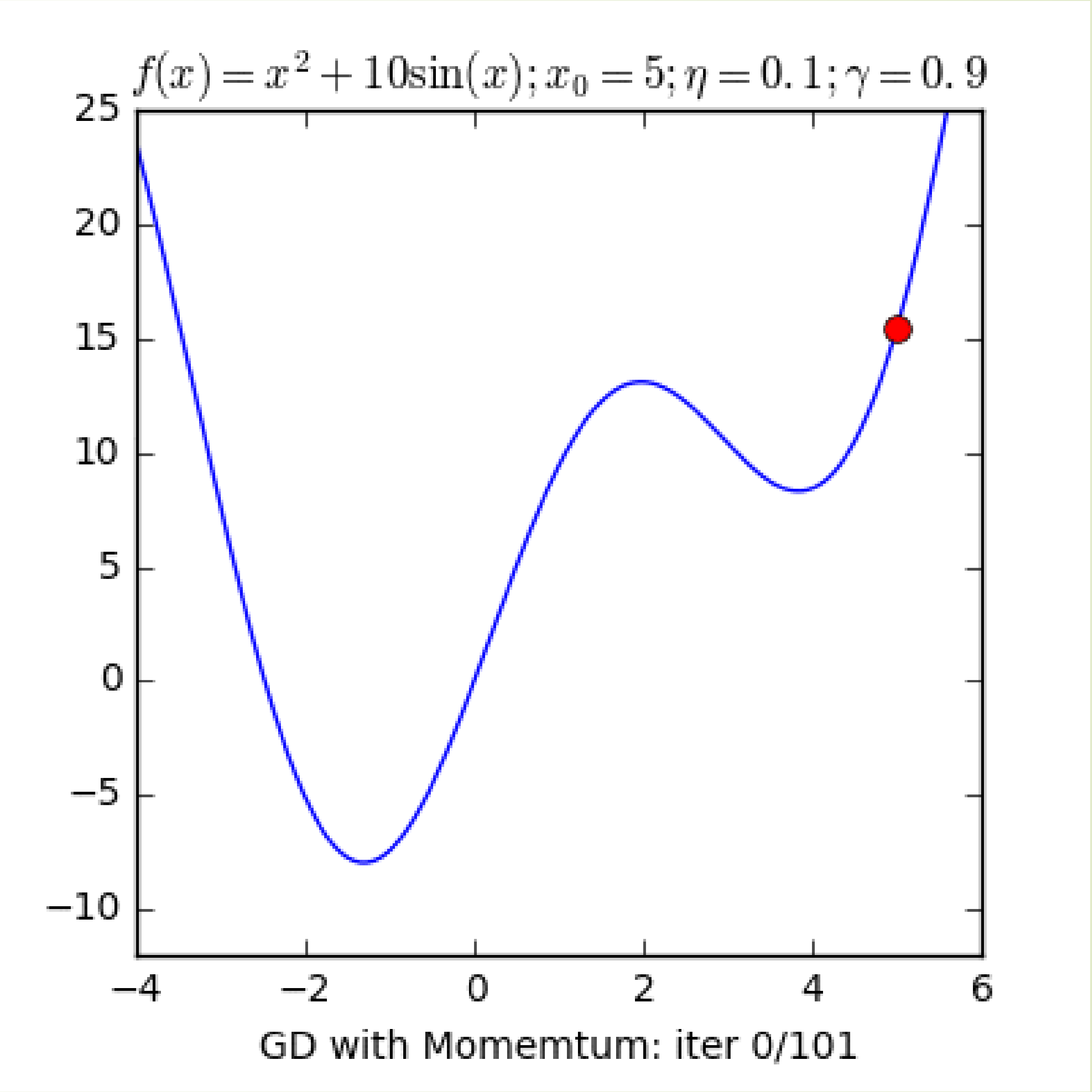
```python
def grad(x):
    return 2*x+ 10*np.cos(x)

def cost(x):
    return x**2 + 10*np.sin(x)
```

```python
# check convergence
def has_converged(theta_new, grad):
    return np.linalg.norm(grad(theta_new))/
                            len(theta_new) < 1e-3

def GD_momentum(theta_init, grad, eta, gamma):
    # Suppose we want to store history of theta
    theta = [theta_init]
    v_old = np.zeros_like(theta_init)
    for it in range(100):
        v_new = gamma*v_old + eta*grad(theta[-1])
        theta_new = theta[-1] - v_new
        if has_converged(theta_new, grad):
            break
        theta.append(theta_new)
        v_old = v_new
    return theta
    # this variable includes all points in the path
    # if you just want the final answer,
    # use `return theta[-1]`
```

3. Momentum



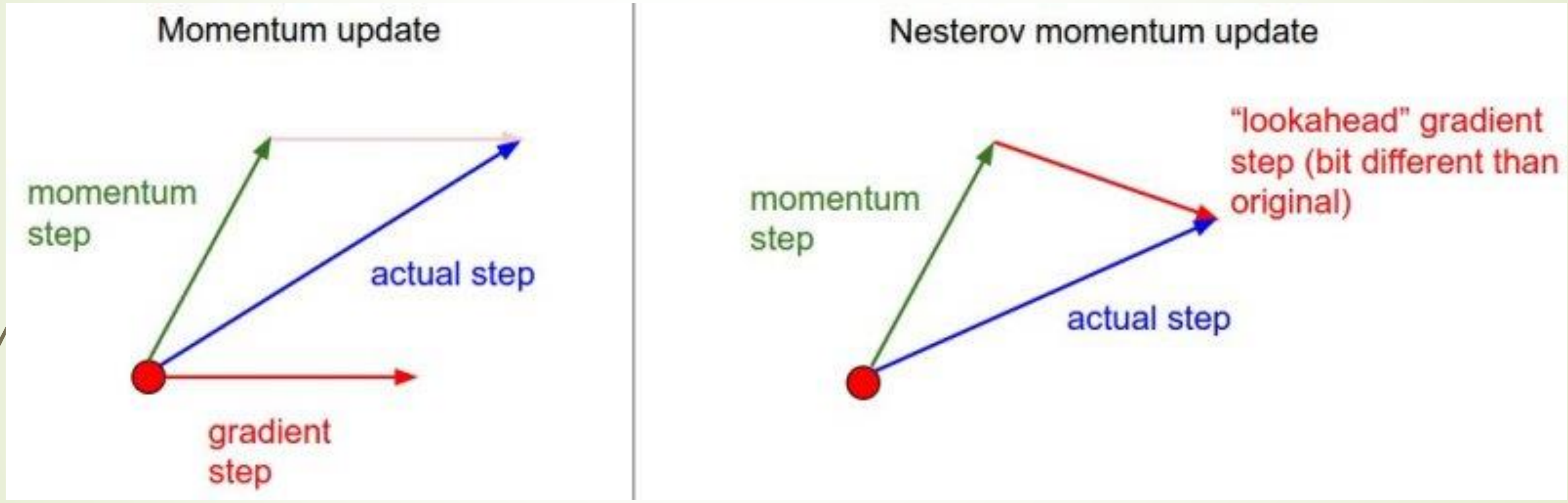
$f(x) = x^2 + 10\sin(x); x_0 = 5; \eta = 0.1$
25
20
15
10
5
0
−5
−10
−4
−2
0
2
4
6
GD without Momemtum: iter 0/5



$f(x) = x^2 + 10\sin(x); x_0 = 5; \eta = 0.1; \gamma = 0.9$
25
20
15
10
5
0
−5
−10
−4
−2
0
2
4
6
GD with Momemtum: iter 0/101

4. Nesterov accelerated gradient (NAG)

- Nhược điểm của Momentum: Khi tới gần đích, momentum mất khá nhiều thời gian trước khi dừng lại.
- Nesterov accelerated gradient (NAG) → giúp cho thuật toán hội tụ nhanh hơn



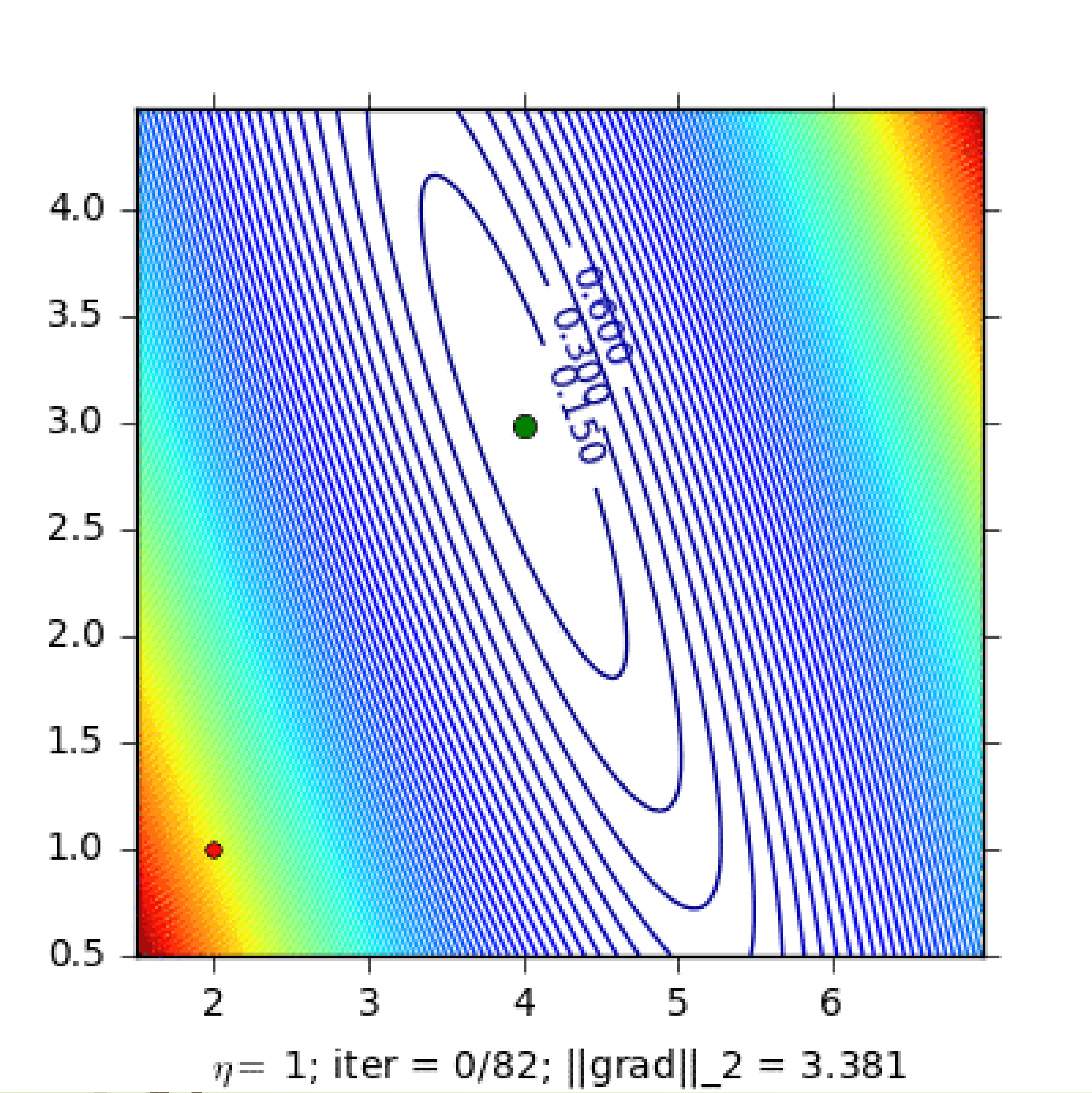
$$\boxed{\theta_{t+1} = \theta_t - v_t}$$

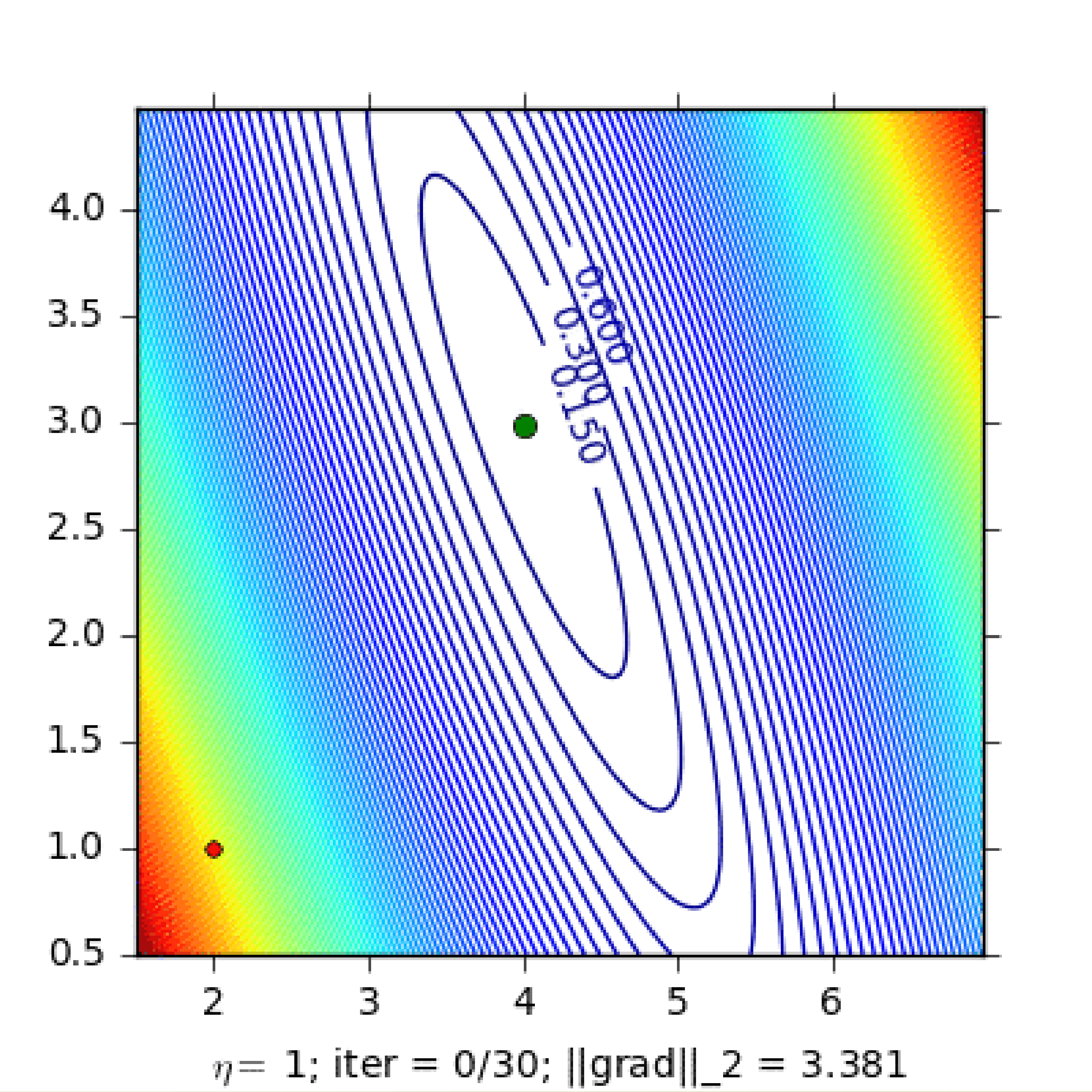
$$\boxed{v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})}$$

$$\Downarrow$$

$$\boxed{\theta_{t+1} = \theta_t - \gamma v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1})}$$

4. Nesterov accelerated gradient (NAG)





4. Nesterov accelerated gradient (NAG)

```
def GD_NAG(w_init, grad, eta, gamma):
    w = [w_init]
    v = [np.zeros_like(w_init)]
    for it in range(100):
        v_new = gamma*v[-1] + eta*grad(w[-1] - gamma*v[-1])
        w_new = w[-1] - v_new
#           print(np.linalg.norm(grad(w_new))/len(w_new))
        if np.linalg.norm(grad(w_new))/len(w_new) < 1e-3:
            break
        w.append(w_new)
        v.append(v_new)
    return (w, it)
w_init = np.array([[2], [1]])
(w_mm, it_mm) = GD_NAG(w_init, grad, .5, 0.9)
```

5. Các biến thể của Gradient descent

- **Batch Gradient descent**
- **Stochastic Gradient descent**
- **Mini-batch Gradient descent**

- **Nhược điểm của Batch Gradient descent (GD thông thường)**
  - Cần tính toán lại đạo hàm của tất cả các điểm dữ liệu sau mỗi vòng lặp
  - Việc tính toán cồng kềnh và không hiệu quả với Online learning

Stochastic Gradient descent

Chỉ tính đạo hàm của hàm mất mát dựa trên *chỉ một* điểm dữ liệu và cập nhật vector trọng số dựa trên đạo hàm này

Stochastic Gradient Descent

- **Epoch:** Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu huấn luyện
- **Batch GD:** Mỗi Epoch ⇔ một lần cập nhật vector trọng số của mô hình
- **SGD:** Mỗi Epoch ⇔ N lần cập nhật vector trọng số của mô hình
- **SGD** chỉ yêu cầu một lượng epoch rất nhỏ, không phù hợp với các bài toán với cơ sở dữ liệu lớn
- Phương trình cập nhật trọng số của thuật toán SGD:

$$\boxed{\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta, x_i, y_i)}$$

Stochastic Gradient Descent

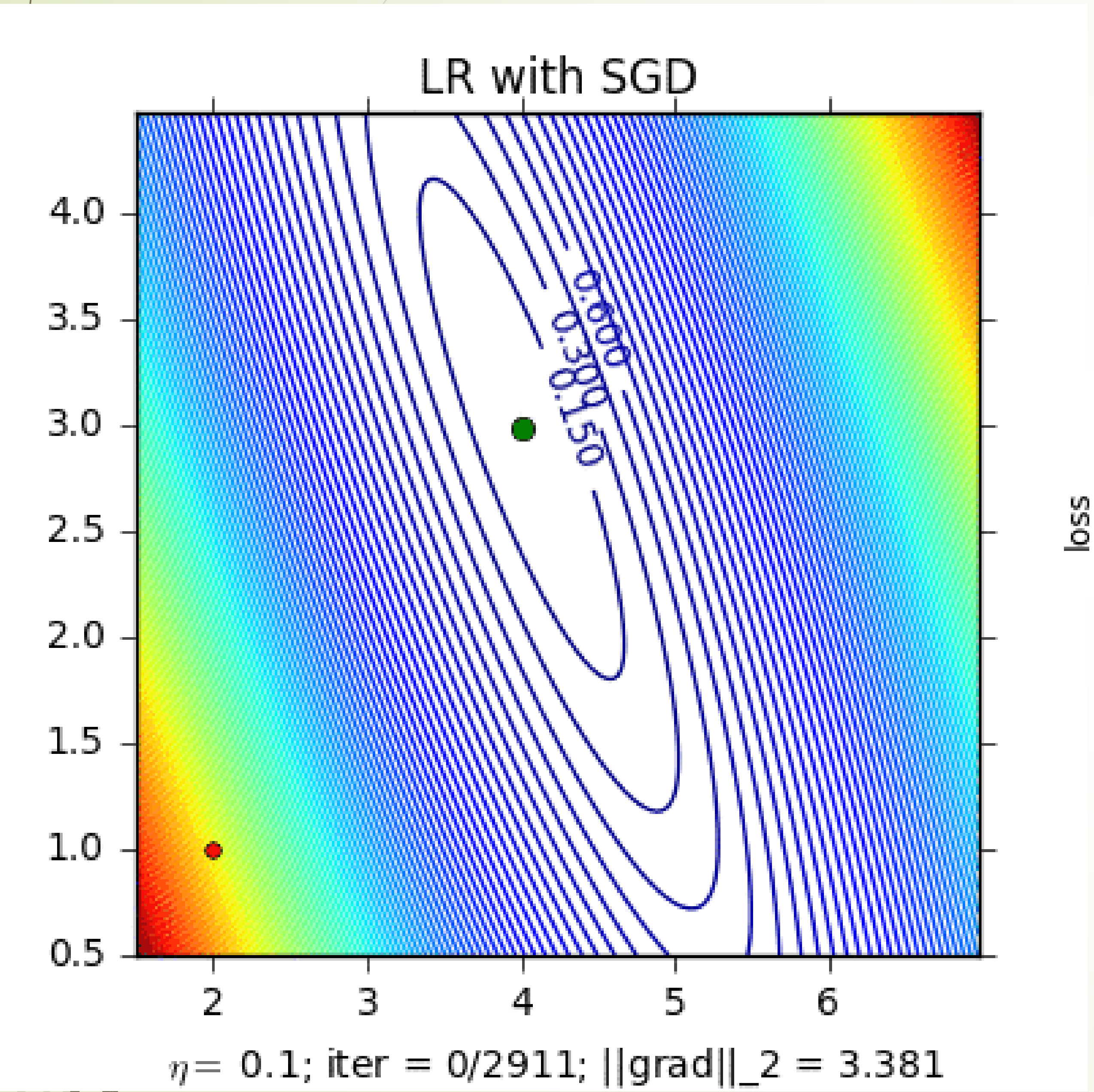
$$J(\mathrm{w}, x_i, y_i) = \frac{1}{2}(x_i\mathrm{w} - y_i)^2$$

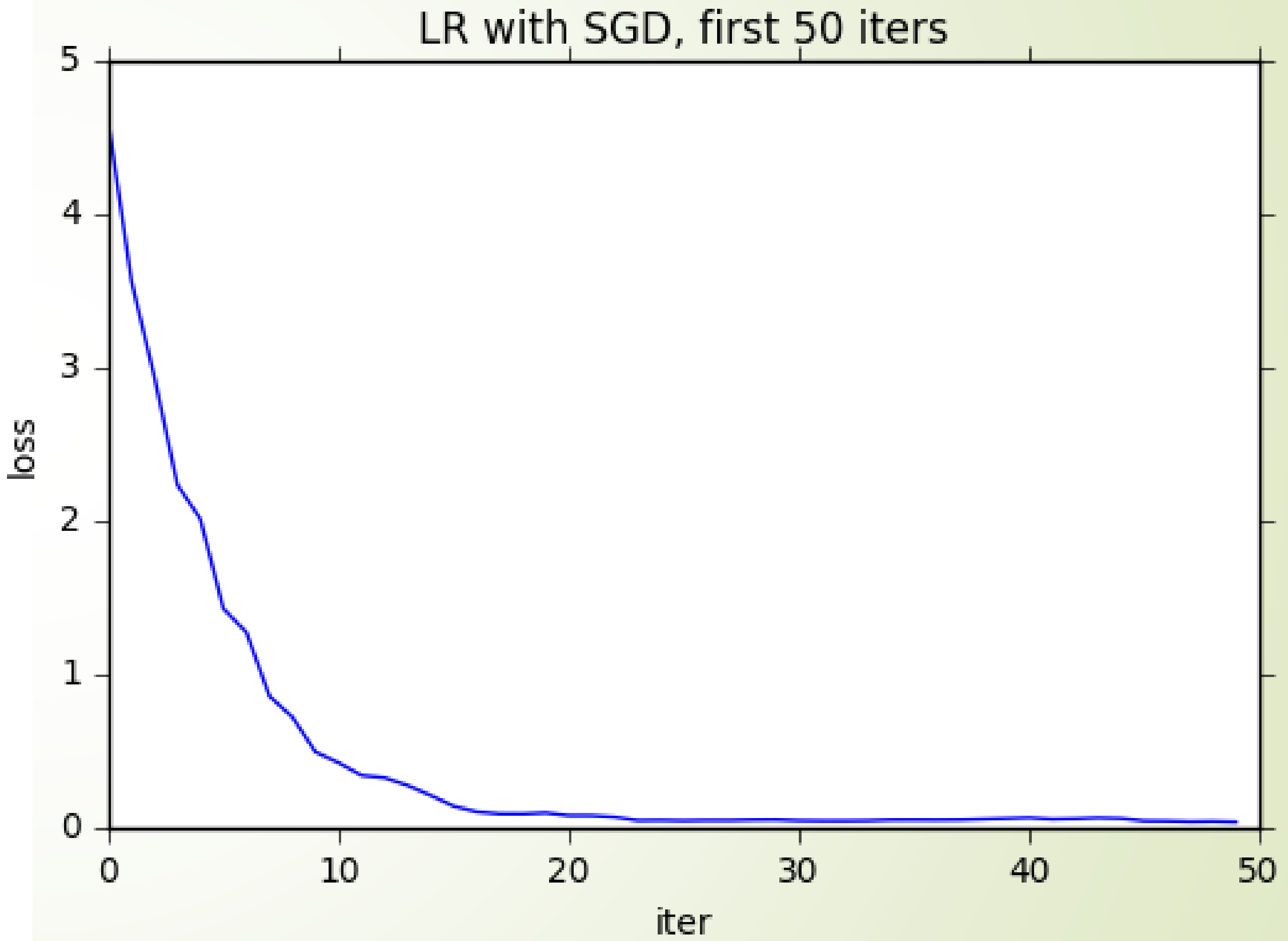
Gradient

$$\nabla_{\mathrm{w}} J(\mathrm{w}, x_i, y_i) = x_i^T(x_i\mathrm{w} - y_i)$$

```
# single point gradient
def sgrad(w, i, rd_id):
    true_i = rd_id[i]
    xi = Xbar[true_i, :]
    yi = y[true_i]
    a = np.dot(xi, w) - yi
    return (xi*a).reshape(2, 1)

def SGD(w_init, grad, eta):
    w = [w_init]
    w_last_check = w_init
    iter_check_w = 10
    N = X.shape[0]
    count = 0
    for it in range(10):
        # shuffle data
        rd_id = np.random.permutation(N)
        for i in range(N):
            count += 1
            g = sgrad(w[-1], i, rd_id)
            w_new = w[-1] - eta*g
            w.append(w_new)
            if count%iter_check_w == 0:
                w_this_check = w_new
                if np.linalg.norm(w_this_check - w_last_check)/len(w_init) < 1e-3:
                    return w
                w_last_check = w_this_check
    return w
```

Stochastic Gradient Descent





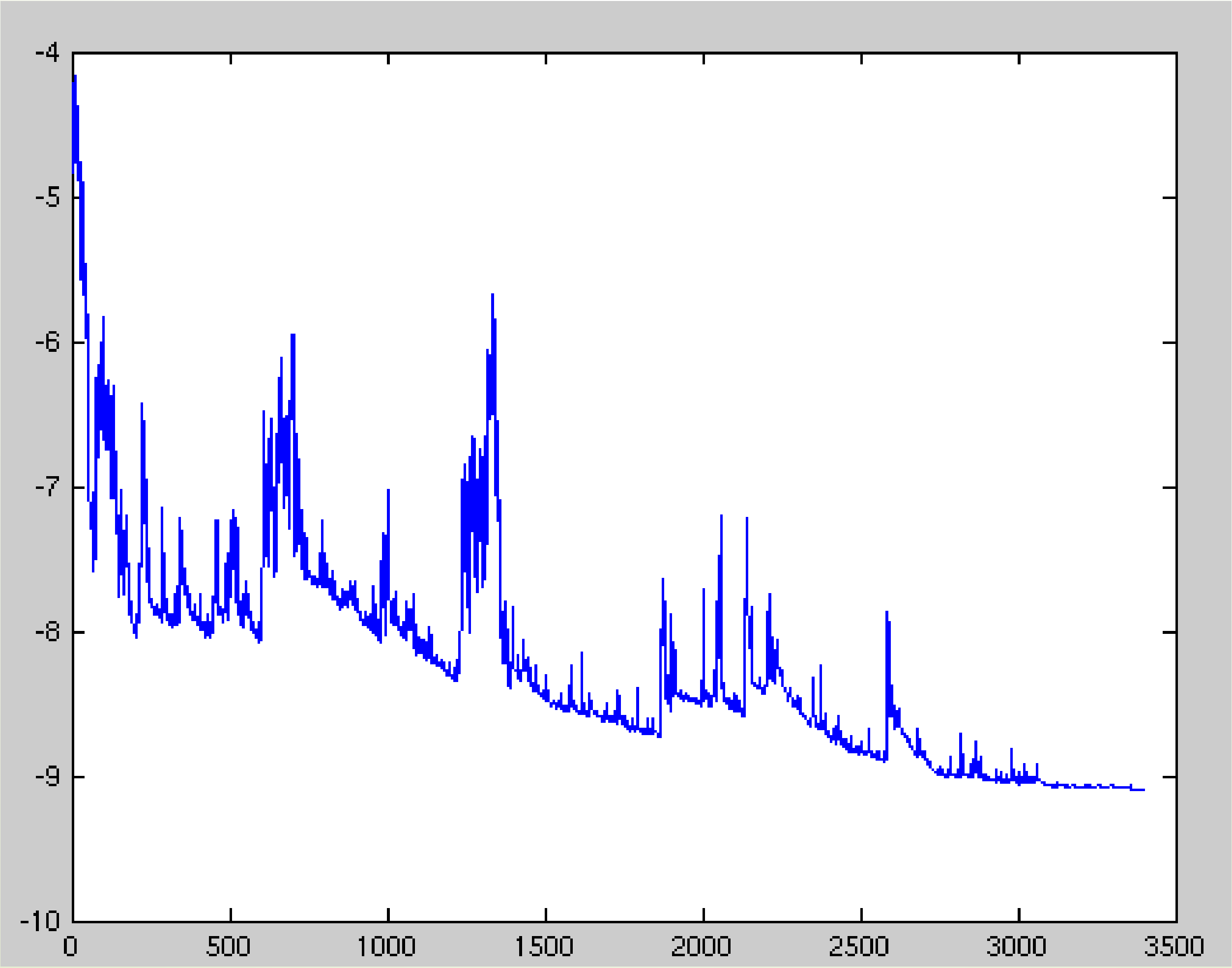
hàm mất mát cho toàn bộ dữ liệu sau khi *chỉ* sử dụng 50 điểm dữ liệu đầu tiên

thuật toán hội tụ khá nhanh đến vùng lân cận của nghiệm

Mini-batch Gradient Descent

- Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia toàn bộ dữ liệu thành các *mini-batch*
- Mỗi *mini-batch* có n điểm dữ liệu ($n = 50 \div 100$)

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta, x_{i:i+n}, y_{i:i+n})$$

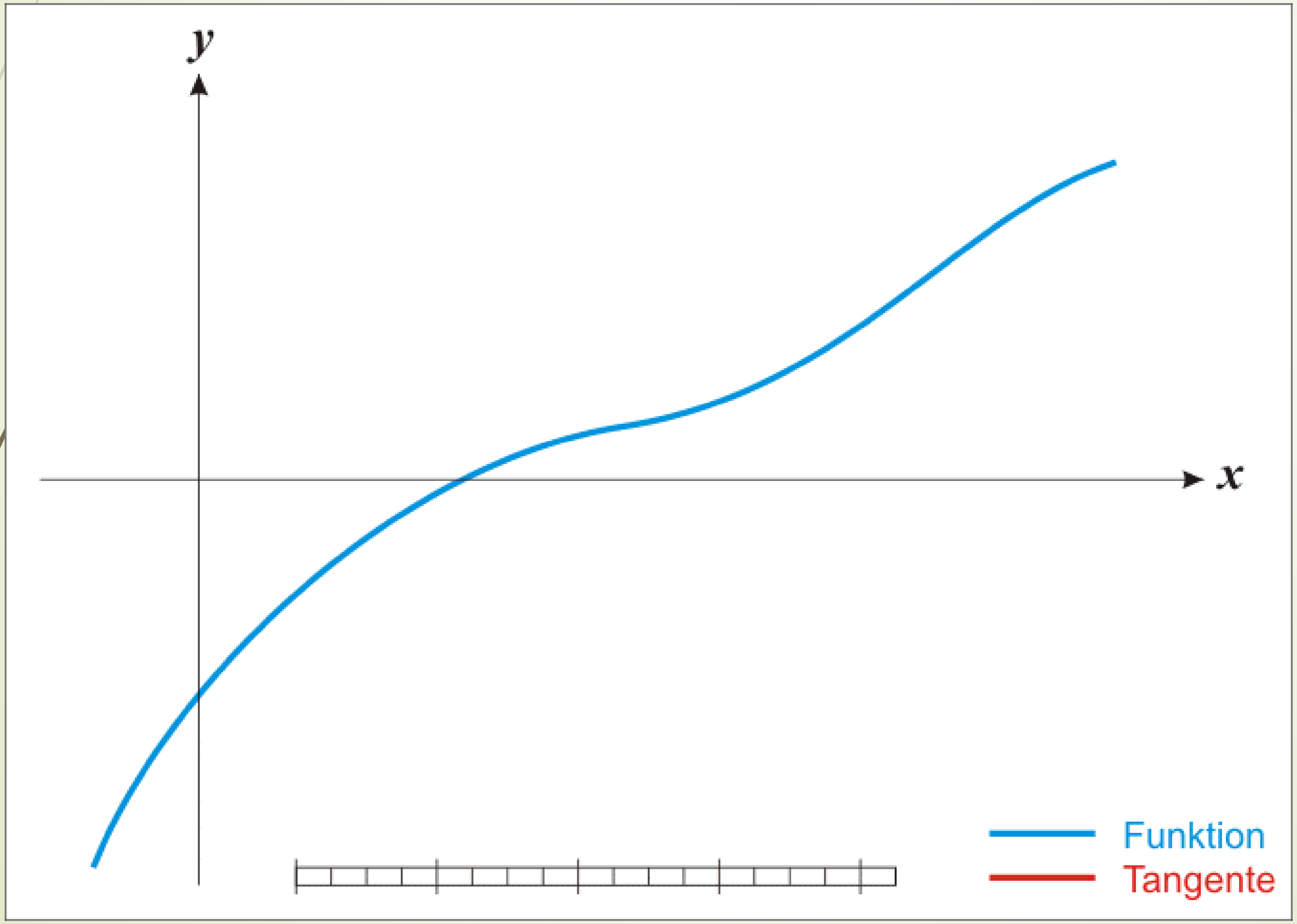


6. Điều kiện dừng (Stop Criteria)

- Giới hạn số vòng lặp
- So sánh gradient của nghiệm tại hai lần cập nhật liên tiếp
- So sánh giá trị của hàm mất mát của nghiệm tại hai lần cập nhật liên tiếp
- Trong SGD và mini-batch GD: so sánh nghiệm sau một vài lần cập nhật

7. Phương pháp Newton

$$y = f'(x_t)(x - x_t) + f(x_t) \Rightarrow x = x_t - \frac{f(x_t)}{f'(x_t)} \triangleq x_{t+1}$$



7. Phương pháp Newton

Bài toán tìm local minimum: $\boldsymbol{f'(x) = 0}$

$$\boxed{x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}} \Rightarrow \boxed{x_{t+1} = x_t - f''(x_t)^{-1} f'(x_t)}$$

$$\Downarrow$$

$$\boxed{\theta = \theta - \mathrm{H}(J(\theta))^{-1} f'(x_t)}$$

$H\big(J(\theta)\big)$là đạo hàm bậc hai của hàm mất mát
(còn gọi là Hessian matrix)

- Khi số chiều và số điểm dữ liệu lớn → đạo hàm bậc hai của hàm mất mát sẽ là một ma trận rất lớn, ảnh hưởng tới cả memory và tốc độ tính toán của hệ thống