COMPSCI 389
Introduction to Machine Learning

Gradient Descent

Prof. Philip S. Thomas (pthomas@cs.umass.edu)

Optimization Perspective

- Recall:
$$\text{argmin}_w \ L(w, D)$$
- Viewing $L(w, D)$ as a function, f, of just the weights (and a fixed data set):
$$\text{argmin}_w \ f(w)$$
- Note that this is equivalent to maximizing a different function, where $g = -f$
$$\text{argmax}_w \ g(w)$$
- We could also write x instead of w:
$$\text{argmin}_x \ f(x)$$
- The function being optimized (minimized or maximized) is called the **objective function** (optimization terminology).
  - In this case, our objective function is a **loss function** (machine learning terminology).
- **Question**: How do we find the input that minimizes a function?

Local Search Methods

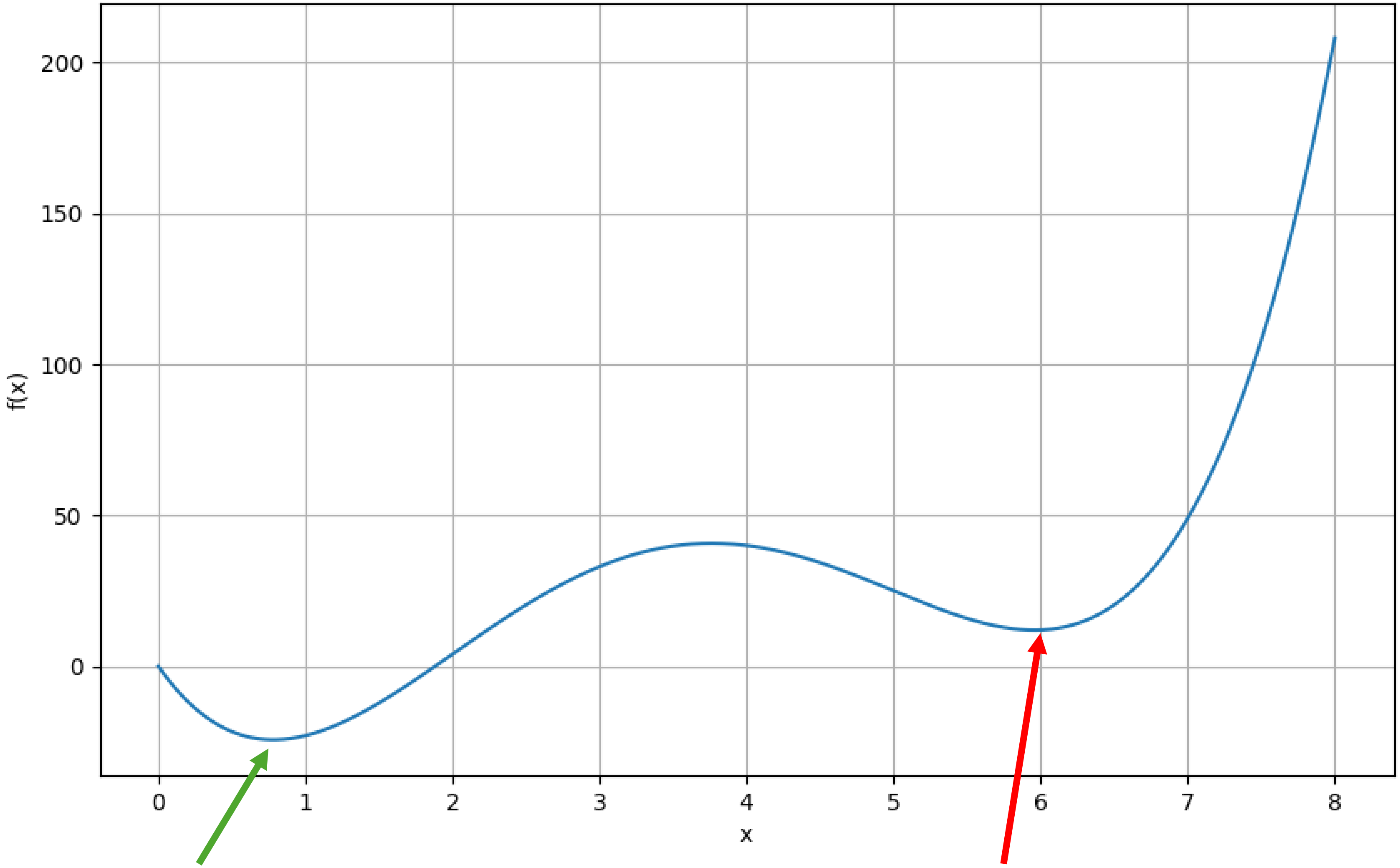
- Start with some initial input, x_0
- Search for a nearby input, x_1, that decreases f:
$$f(x_1) < f(x_0)$$
- Repeat, finding a nearby input x_{i+1} that decreases f (for each iteration i):
$$f(x_{i+1}) < f(x_i)$$
- Stop when:
  - You cannot find a new input that decreases f
  - The decrease in f becomes very small
  - The process runs for some predetermined amount of time
- Called "local search methods" because they search locally around some current point, x_i.

“Find a nearby point that decreases f”

- We will consider gradient-based optimizers.
- At any input/point x, we can query:
  - $f(x)$: The value of the objective function at the point
  - $\frac{df(x)}{dx}$: The derivative of the objective function at the point
    - This is the **gradient**, and is also written as $\nabla f(x)$

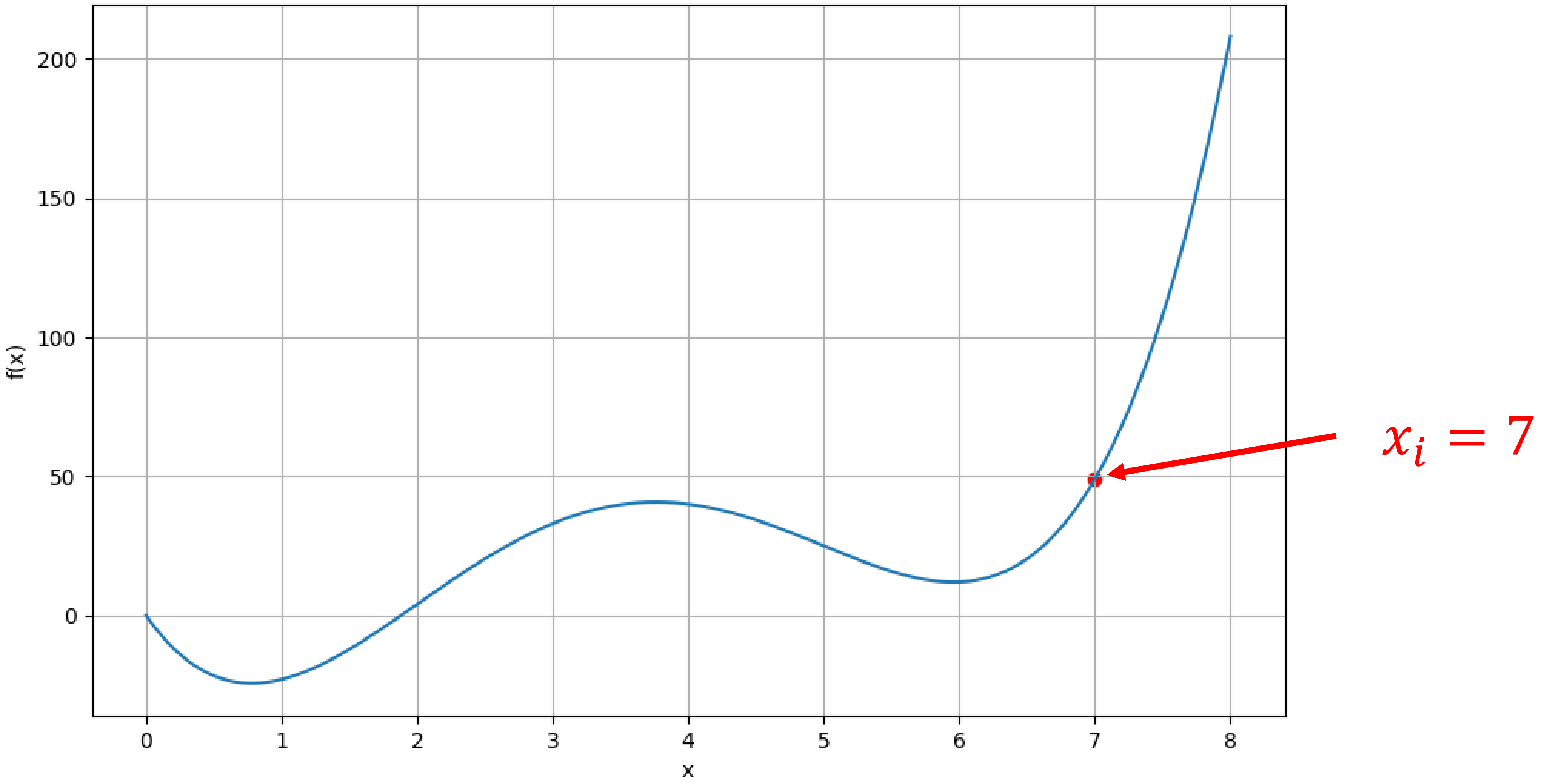
Question: Is a global minimum a local minimum?

Answer: Yes!



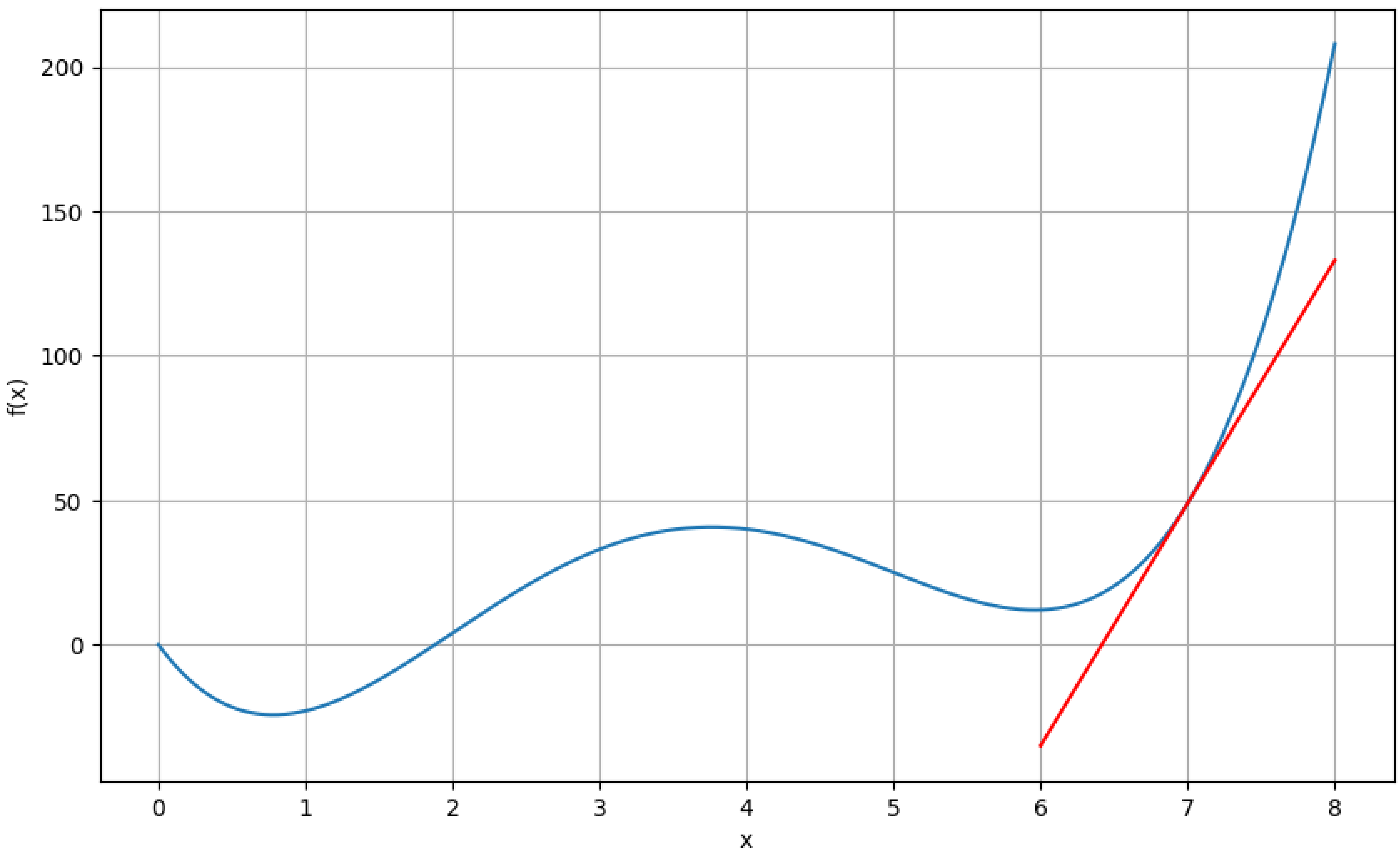
Global minimum: A location where the function achieves the lowest value (the argmin).

Local minimum: A location where all nearby (adjacent) points have higher values.



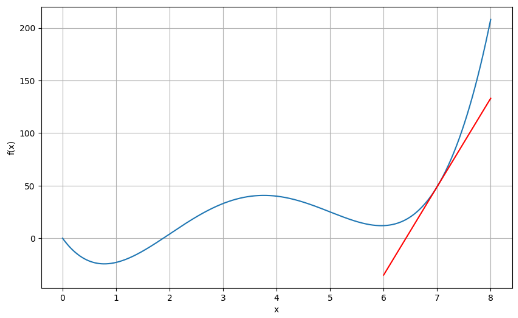
Question: How can we find a point x_{i+1} such that $f(x_{i+1}) < f(x_i)$? That is, a point that is “lower”?

Idea: Move a small amount “downhill”



Notice: The slope of the function tells us which direction is uphill / downhill.
Positive slope: Decrease x_i to get x_{i+1}. **Negative slope**: Increase x_i to get x_{i+1}.

Gradient Descent



- Take a step of length α (a small positive constant) in the opposite direction of the slope:

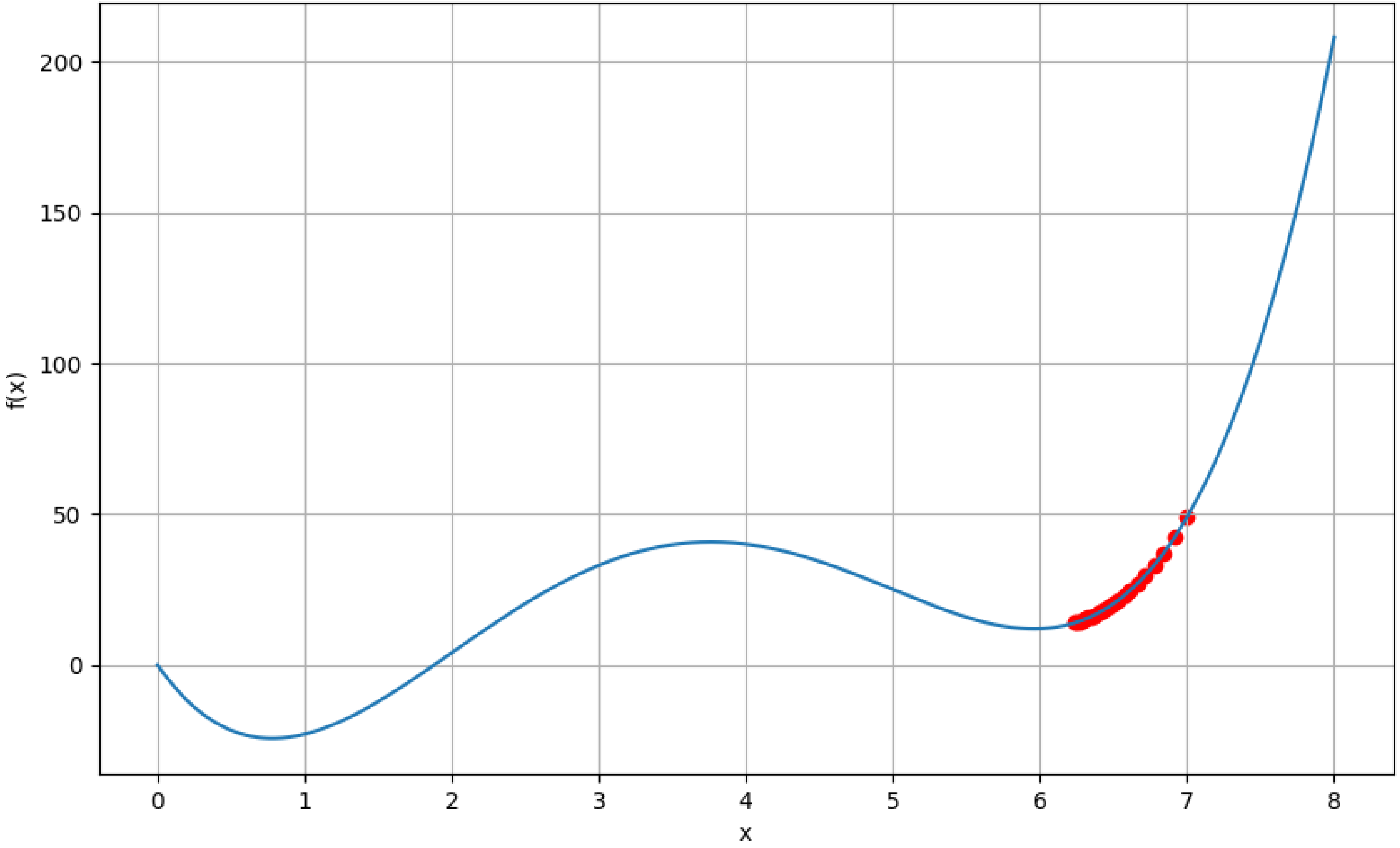
$$x_{i+1} = x_i - \alpha \times \text{slope}.$$

- **Note**: The slope is $\frac{df(x)}{dx}$, so we can write:

$$x_{i+1} = x_i - \alpha \frac{df(x)}{dx}.$$

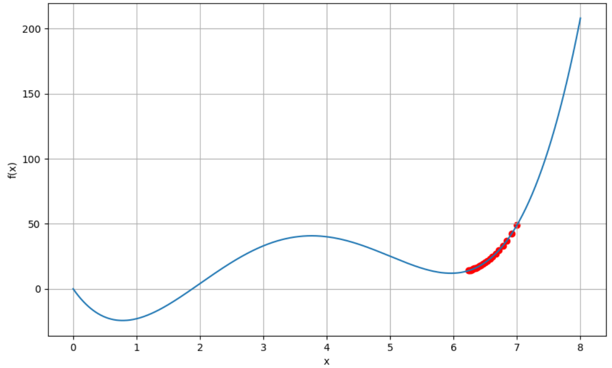
- α is a hyperparameter called the **step size** or **learning rate**.

Gradient descent, $x_0 = 7, \alpha = 0.001$
$f(x) = x^4 - 14x^3 + 60x^2 - 70x$



Question: Why do the points get closer together when we use the same step size, α?

Why do the points get closer together when we use the same step size, α?

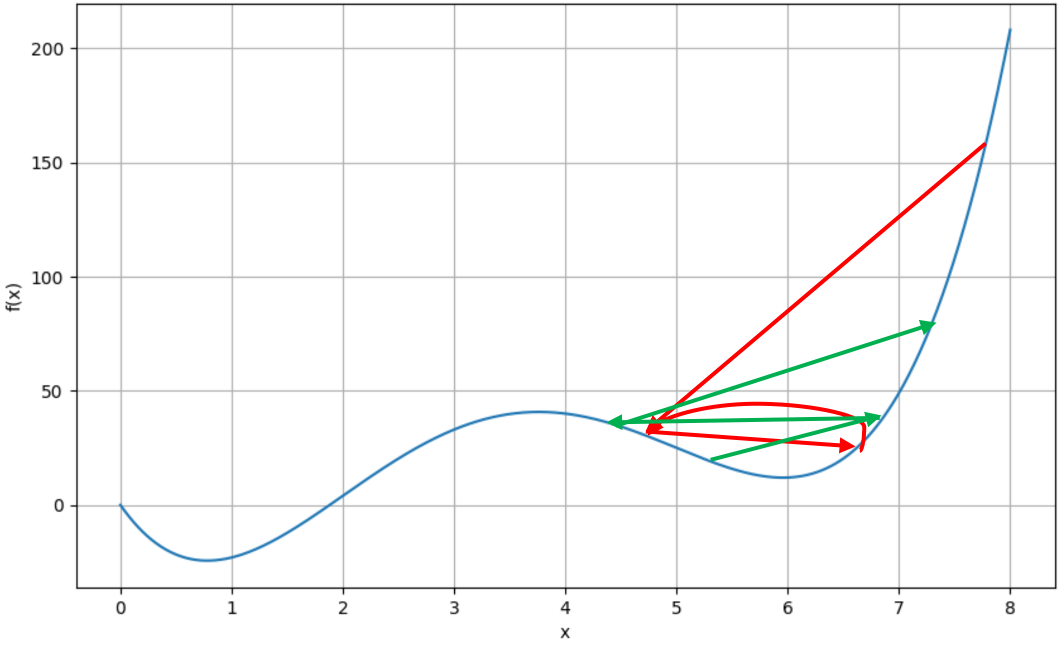


$$x_{i+1} = x_i - \alpha \frac{df(x)}{dx}$$

- As x_i approaches a local optimum, the slope goes to zero.
- This allows for "convergence" to a local optimum.
- Gradient descent can still overshoot the (local) minimum.
- If the step size is small enough (or decayed appropriately over time), gradient descent is guaranteed to converge to a local minimum.
  - If it overshoots a minimum by a small amount, it will reverse direction and move back towards the minimum.

Overshooting and Divergence

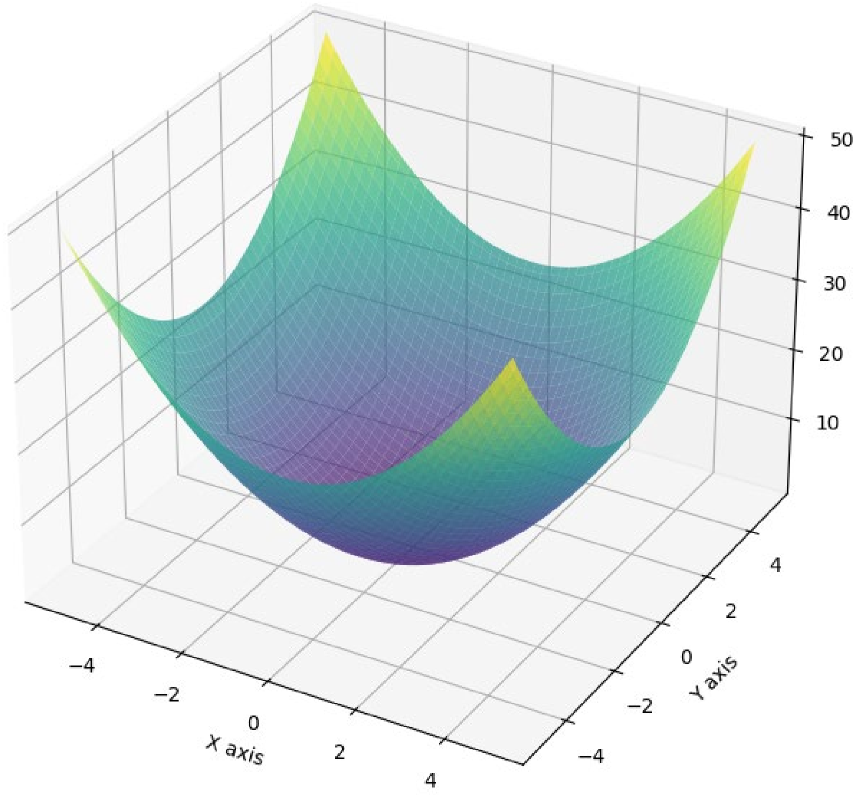
- If the step length was constant and too big, it could forever over-shoot the (local) minimum, diverging or oscillating (not making progress towards the local minimum).



Multidimensional Gradient Descent

- What if the function, f, takes many inputs?
  - Our loss function, $L(w, D)$ takes the weight vector w as input
    - We view D as fixed.
  - For now, consider a function $f(x, y)$, where x and y are two real numbers.

$$f(x, y) = x^2 + y^2$$



Consider the point (3,3)

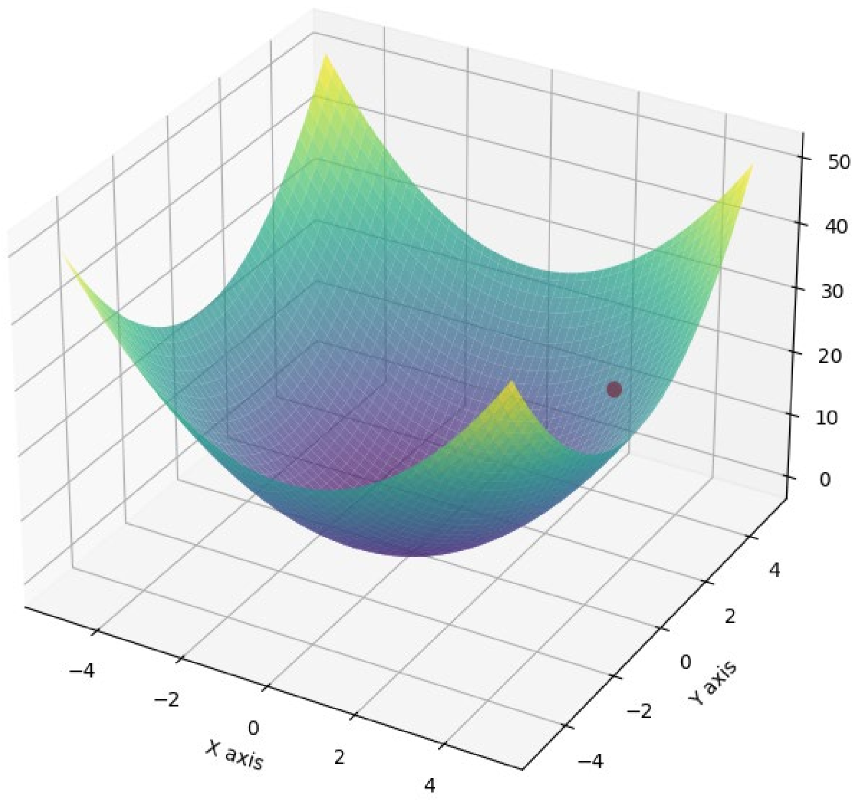
Question: How can we find a new point that is "downhill"?

Idea: Compute the slope along each axis!

x-slope: $\frac{\partial f(x,y)}{\partial x}$
y-slope: $\frac{\partial f(x,y)}{\partial y}$

The **gradient** is the concatenation of the slopes along each dimension/axis:

$$\nabla f(x) = \left[\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right]$$



The Gradient

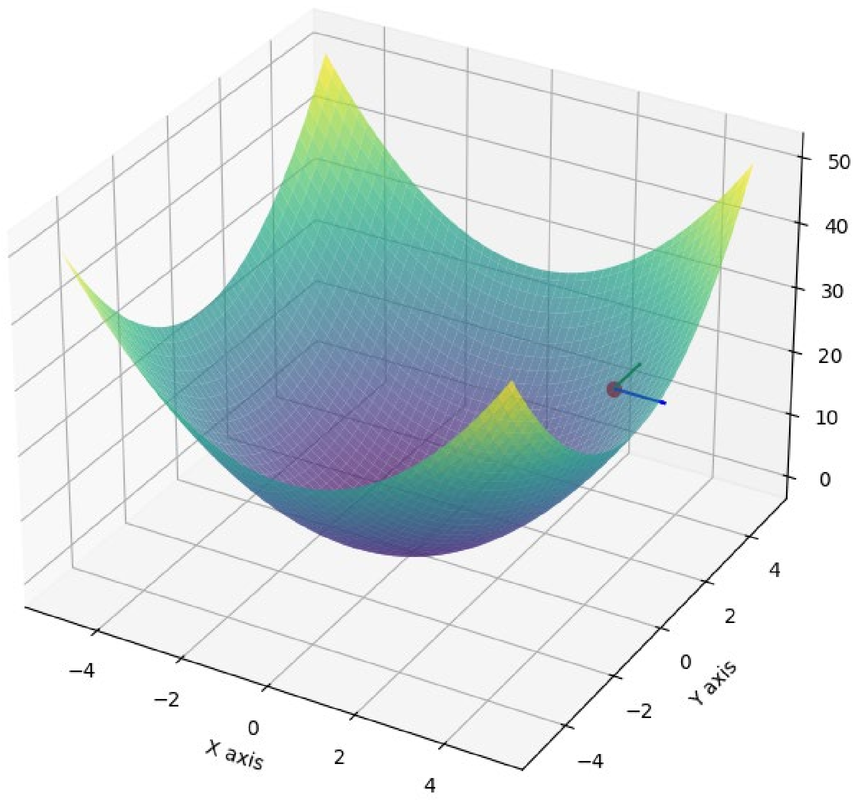
Question: How can we find a new point that is "downhill"?

Idea: Compute the slope along each axis!

x-slope: $\frac{\partial f(x,y)}{\partial x}$
y-slope: $\frac{\partial f(x,y)}{\partial y}$

The **gradient** is the concatenation of the slopes along each dimension/axis:

$$\nabla f(x) = \left[\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right]$$



Note: The gradient is also called the "**direction of steepest ascent**". It indicates how to change each input to go up-hill as quickly as possible.

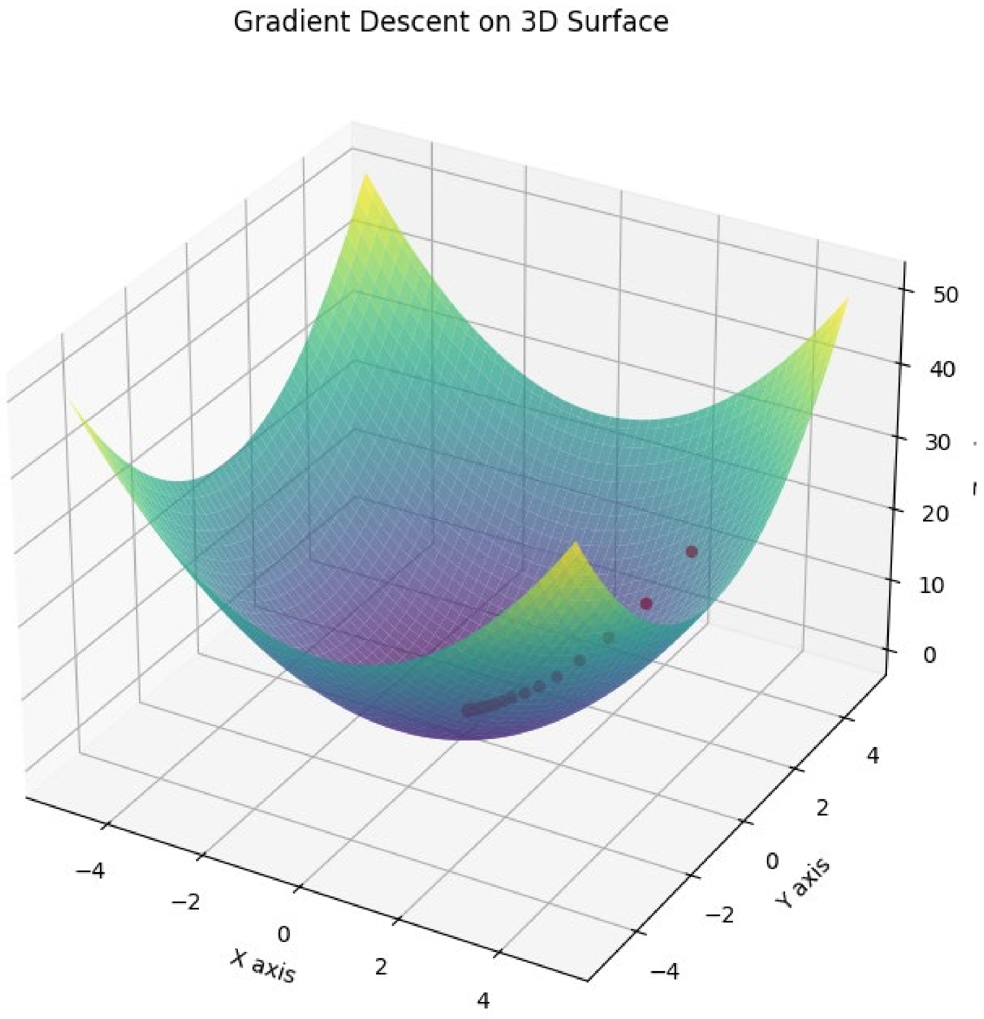
Gradient Descent: Move both x and y in the negative direction of their slopes. That is, move in the opposite direction of the gradient:

$$x_{i+1} = x_i - \alpha \frac{\partial f(x_i, y_i)}{\partial x_i}$$
$$y_{i+1} = y_i - \alpha \frac{\partial f(x_i, y_i)}{\partial y_i}$$

OR

$$(x_{i+1}, y_{i+1}) = (x_i, y_i) - \alpha \nabla f(x_i, y_i)$$

Gradient Descent on $f(x, y) = x^2 + y^2$
$(x_0, y_0) = (3,3), \alpha = 0.7$



Pseudocode: Gradient Descent on $f(x)$

- **Hyperparameter**: Step size α. Typically a small constant like $0.1, 0.01, 0.001, \ldots$
- **Assumption**: f is a function that takes a vector (or single real number) as input and produces a single real number as output.
- **Assumption**: f is smooth (differentiable)
- **Method**:
  - Select an arbitrary initial point, x_0 (a vector).
  - For each iteration i, set $x_{i+1} = x_i - \alpha \nabla f(x_i)$. Equivalently, for each element of x_i (indexed by j):

$$x_{i+1,j} = x_{i,j} - \alpha \frac{\partial f(x_i)}{\partial x_{i,j}}$$

  - Stop when progress becomes slow or after some fixed amount of time.

Gradient Descent: Adaptive Step Sizes

- Tuning the step size, α, can be challenging.
- **Adaptive step size** methods measure properties of the function over time to adapt the step size automatically.
  - Many methods (ADAGRAD, ADAM, etc.)
  - Some change not only the length of the step, but also the *direction* of the step!
  - Details beyond the scope of this course.

Gradient Descent for Minimizing Sample MSE (Linear Parametric Model)

$$\operatorname{argmin}_w L(w, D)$$

- Initialize w_0 arbitrarily.
- Iterate:

$$w_{i+1} \leftarrow w_i - \alpha \frac{\partial L(w_i, D)}{\partial w_i}$$

- Equivalently, for each weight (indexed by j):

$$w_{i+1,j} \leftarrow w_{i,j} - \alpha \frac{\partial L(w_i, D)}{\partial w_{i,j}}$$

- To implement this, we need to know $\frac{\partial L(w_i,D)}{\partial w_{i,j}}$

What is $\frac{\partial L(w_i,D)}{\partial w_{i,j}}$?

$$L(w_i, D) = \frac{1}{n}\sum_{i'=1}^{n}\left(y_{i'} - \sum_{j'=1}^{d} w_{i,j'}\phi_{j'}(x_{i'})\right)^2$$

Question: Why $\Sigma_{j'}$ rather than Σ_j?

Answer: We already used the symbol j to denote the weight we are taking the derivative with respect to. So, we use a different symbol for the index of the summation.

$$\frac{\partial L(w_i,D)}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}}\frac{1}{n}\sum_{i'=1}^{n}\left(y_{i'} - \sum_{j'=1}^{d} w_{i,j'}\phi_{j'}(x_{i'})\right)^2$$

$$\frac{\partial L(w_i,D)}{\partial w_{i,j}} = \frac{1}{n}\sum_{i'=1}^{n}\frac{\partial}{\partial w_{i,j}}\left(y_{i'} - \sum_{j'=1}^{d} w_{i,j'}\phi_{j'}(x_{i'})\right)^2$$

$$\frac{\partial L(w_i,D)}{\partial w_{i,j}} = \frac{1}{n}\sum_{i'=1}^{n}2\left(y_{i'} - \sum_{j'=1}^{d} w_{i,j'}\phi_{j'}(x_{i'})\right)\frac{\partial}{\partial w_{i,j}}\left(y_{i'} - \sum_{j'=1}^{d} w_{i,j'}\phi_{j'}(x_{i'})\right)$$

$$\frac{\partial L(w_i,D)}{\partial w_{i,j}} = \frac{-1}{n}\sum_{i'=1}^{n}2\left(y_{i'} - \sum_{j'=1}^{d} w_{i,j'}\phi_{j'}(x_{i'})\right)\frac{\partial}{\partial w_{i,j}}\sum_{j'=1}^{d} w_{i,j'}\phi_{j'}(x_{i'})$$

$$\frac{\partial L(w_i,D)}{\partial w_{i,j}} = \frac{-1}{n}\sum_{i'=1}^{n}2\left(y_{i'} - \sum_{j'=1}^{d} w_{i,j'}\phi_{j'}(x_{i'})\right)\phi_j(x_{i'})$$

$$\frac{\partial}{\partial w_{i,j}}\sum_{j'=1}^{d} w_{i,j'}\phi_{j'}(x_{i'}) = \frac{\partial}{\partial w_{i,j}} w_{i,j}\,\phi_j(x_{i'}) = \phi_j(x_{i'})$$

Gradient Descent for Minimizing Sample MSE (Linear Parametric Model)

- For each weight (indexed by j):

$$w_{i+1,j} \leftarrow w_{i,j} - \alpha \frac{\partial L(w_i, D)}{\partial w_{i,j}}$$

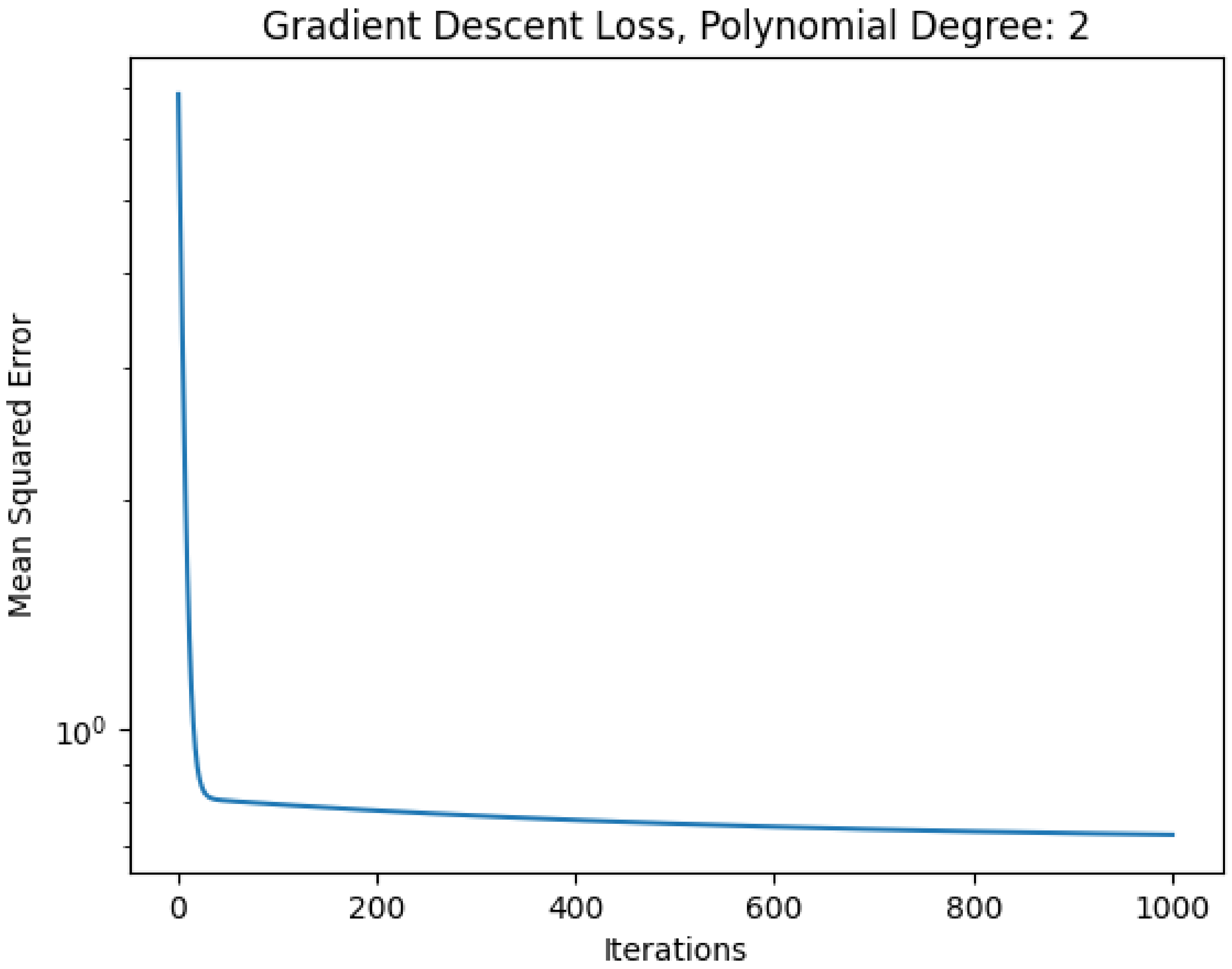
- Where:

$$\frac{\partial L(w_i, D)}{\partial w_{i,j}} = \frac{-1}{n} \sum_{i=1}^{n} 2 \left(y_i - \sum_{j'=1}^{d} w_{i,j'} \phi_{j'}(x_i) \right) \phi_j(x_i)$$

- So, for each weight (indexed by j):

$$w_{i+1,j} \leftarrow w_{i,j} + \alpha \frac{1}{n} \sum_{i=1}^{n} 2 \left(y_i - \sum_{j'=1}^{d} w_{i,j'} \phi_{j'}(x_i) \right) \phi_j(x_i)$$

GPA Data set


Gradient Descent Loss, Polynomial Degree: 2
Mean Squared Error
10^0
0
200
400
600
800
1000
Iterations

```
Iteration 0/1000, Loss: 8.4351
Iteration 1/1000, Loss: 6.8922
Iteration 2/1000, Loss: 5.6614
Iteration 3/1000, Loss: 4.6794
Iteration 4/1000, Loss: 3.8960
Iteration 5/1000, Loss: 3.2710
Iteration 6/1000, Loss: 2.7724
Iteration 7/1000, Loss: 2.3746
Iteration 8/1000, Loss: 2.0572
Iteration 9/1000, Loss: 1.8040
Iteration 10/1000, Loss: 1.6019
Iteration 11/1000, Loss: 1.4407
Iteration 12/1000, Loss: 1.3120
Iteration 13/1000, Loss: 1.2093
Iteration 14/1000, Loss: 1.1274
Iteration 15/1000, Loss: 1.0619
Iteration 16/1000, Loss: 1.0097
Iteration 17/1000, Loss: 0.9680
Iteration 18/1000, Loss: 0.9347
Iteration 19/1000, Loss: 0.9081
Iteration 20/1000, Loss: 0.8868
Iteration 21/1000, Loss: 0.8698
Iteration 22/1000, Loss: 0.8562
Iteration 23/1000, Loss: 0.8453
Iteration 24/1000, Loss: 0.8366
...
Iteration 997/1000, Loss: 0.7177
Iteration 998/1000, Loss: 0.7177
Iteration 999/1000, Loss: 0.7176
Iteration 1000/1000, Loss: 0.7176
```

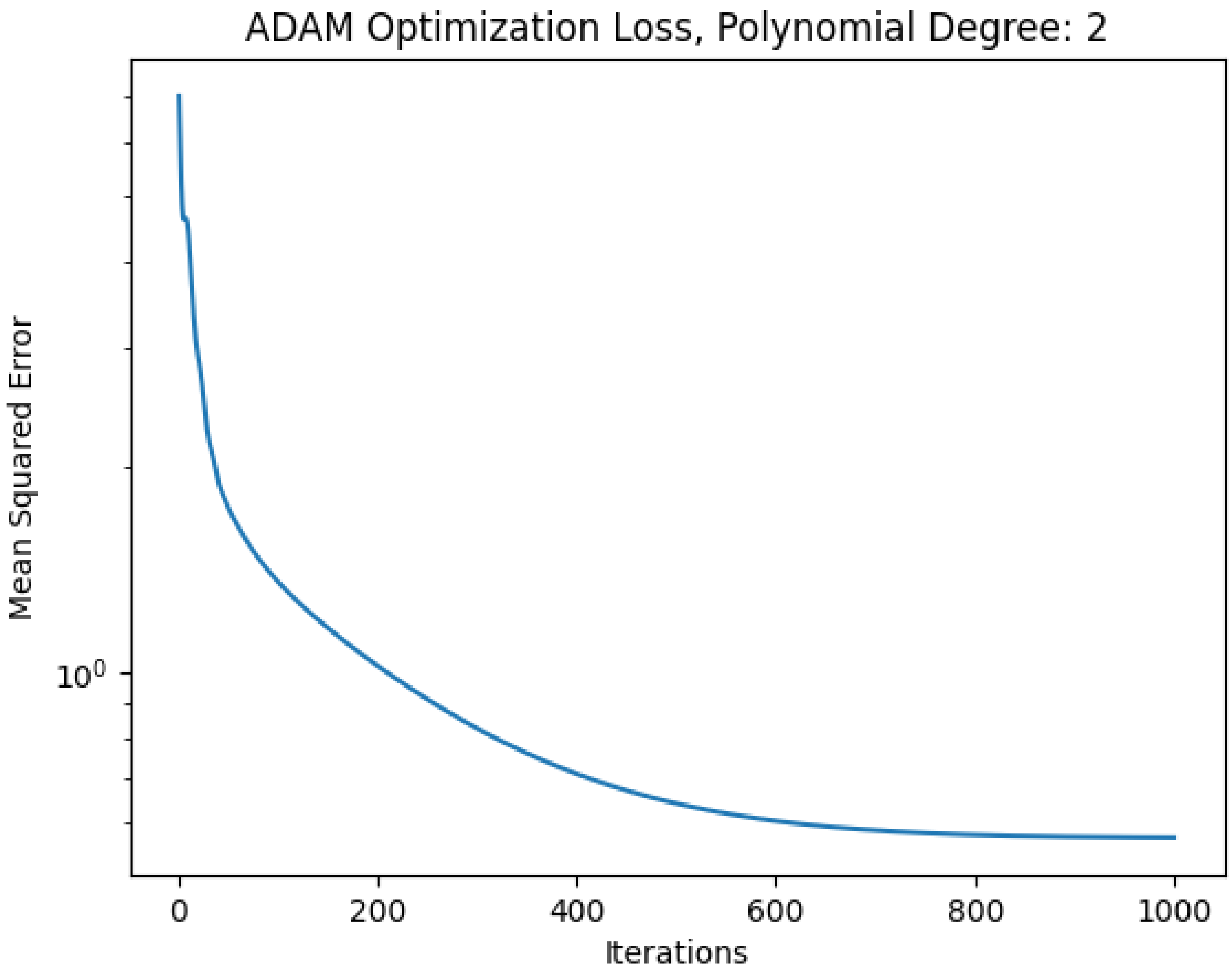
Not very good!

```
Test MSE: 0.7856 Standard
Error of MSE: 0.0084
```

Least Squares with Linear Parametric Model

- **Question**: Why was the final MSE so large (0.78)?
  - Other methods achieved ~0.57
- **Answer**:
  - Better weights likely exist!
  - Gradient descent was making very slow progress at the end.
- **Idea**: Let's try using an adaptive step size method, ADAM.

```
Iteration 1/1000, Loss: 7.0300
Iteration 2/1000, Loss: 5.9808
Iteration 3/1000, Loss: 5.2636
Iteration 4/1000, Loss: 4.8402
Iteration 5/1000, Loss: 4.6492
Iteration 6/1000, Loss: 4.6073
Iteration 7/1000, Loss: 4.6240
Iteration 8/1000, Loss: 4.6272
Iteration 9/1000, Loss: 4.5771
Iteration 10/1000, Loss: 4.4633
Iteration 11/1000, Loss: 4.2945
Iteration 12/1000, Loss: 4.0891
Iteration 13/1000, Loss: 3.8682
Iteration 14/1000, Loss: 3.6514
Iteration 15/1000, Loss: 3.4540
Iteration 16/1000, Loss: 3.2858
Iteration 17/1000, Loss: 3.1506
Iteration 18/1000, Loss: 3.0462
Iteration 19/1000, Loss: 2.9662
Iteration 20/1000, Loss: 2.9017
Iteration 21/1000, Loss: 2.8433
Iteration 22/1000, Loss: 2.7831
Iteration 23/1000, Loss: 2.7164
Iteration 24/1000, Loss: 2.6418
Iteration 25/1000, Loss: 2.5612
...
Iteration 997/1000, Loss: 0.5650
Iteration 998/1000, Loss: 0.5650
Iteration 999/1000, Loss: 0.5650
Iteration 1000/1000, Loss: 0.5649
```



Much better!

```
Test MSE: 0.5791
Standard Error of MSE: 0.0073
```

End