

CS2040S

Data Structures and Algorithms

Welcome!

Summary

Monday:

Algorithm Analysis:

- *Asymptotic notation*
- *Models of computation*

Binary Search:

- *Finding an element in array.*
- *Invariants*

Peak Finding

- *Finding a (local) maximum*
- *1d-algorithm*
- *2d-algorithm*

Today: Optimization

Finding a maximum / minimum

- *Key aspect of machine learning*
- *Binary search isn't always good*

Newton's method

- *Second order approach*

Gradient descent

- *First order approach*

Announcements / Reminders

Problem sets:

Problem Set 1 was due Monday.

Problem Set 2 is due next Thursday. (Happy Lunar New Year!)

Optional Practice Problems

- Good practice!
- Make sure you understand the techniques.
- Extra coding experience.
- Extra algorithm experience.

Announcements / Reminders

Tutorials:

Read your e-mail / announcements and follow instructions.

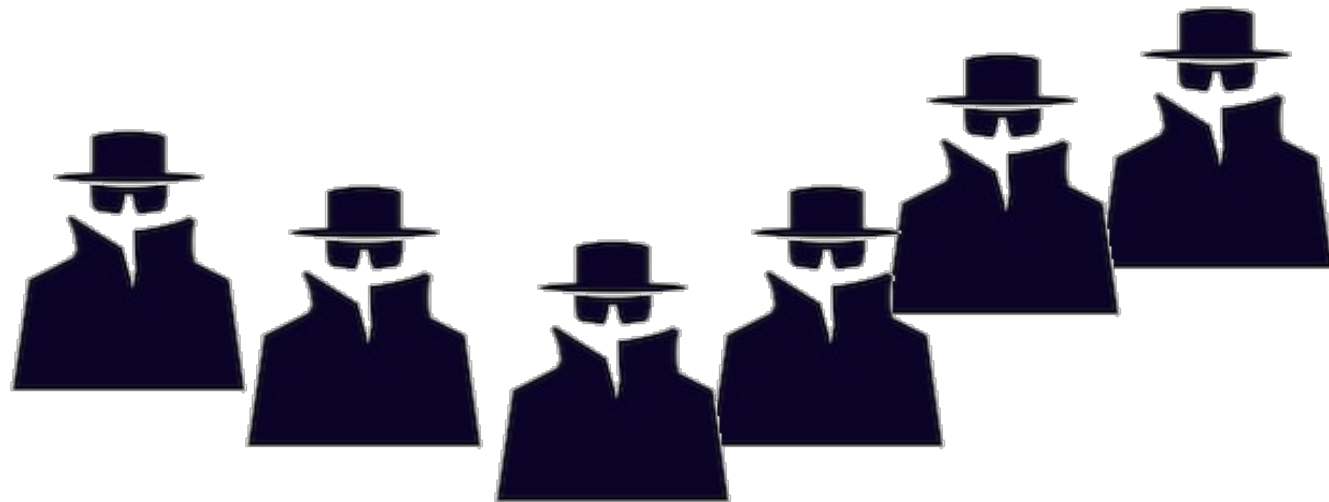
A mutually-agreed swap is approved by default.

Recitations:

We will get to it, after tutorials are resolved.

Puzzle of the Week

There are N students in CS2040S and K of them are spies. Your job is to identify all the spies.



Puzzle of the Week

To catch a spy:



You can send some students on a mission.



If all **K** spies are on the mission, they will meet.

You learn if the meeting occurred or not.



You learn nothing else.



Puzzle of the Week

Find:



The best strategy you can to catch all the spies. (Write a program!)



Announcements / Reminders

Competition:

Find the spies!

Open on Coursemology now:

- Optional.
- Write a program to implement your best spy catching strategy.
- We will test it on various spy rings.
- Fewest queries / fewest dollars wins!
- (And a small bonus for participating.)

Binary Search

$O(\log n)$ time

Sorted array: $A[1..n]$

2	4	4	5	6	7	8	9	11	17	23	28
---	---	---	---	---	---	---	---	----	----	----	----

```
Search(A, key, n)
```

```
    begin = 0
```

```
    end = n-1
```

```
    while begin < end do:
```

```
        mid = begin + (end-begin)/2;
```

```
        if key <= A[mid] then
```

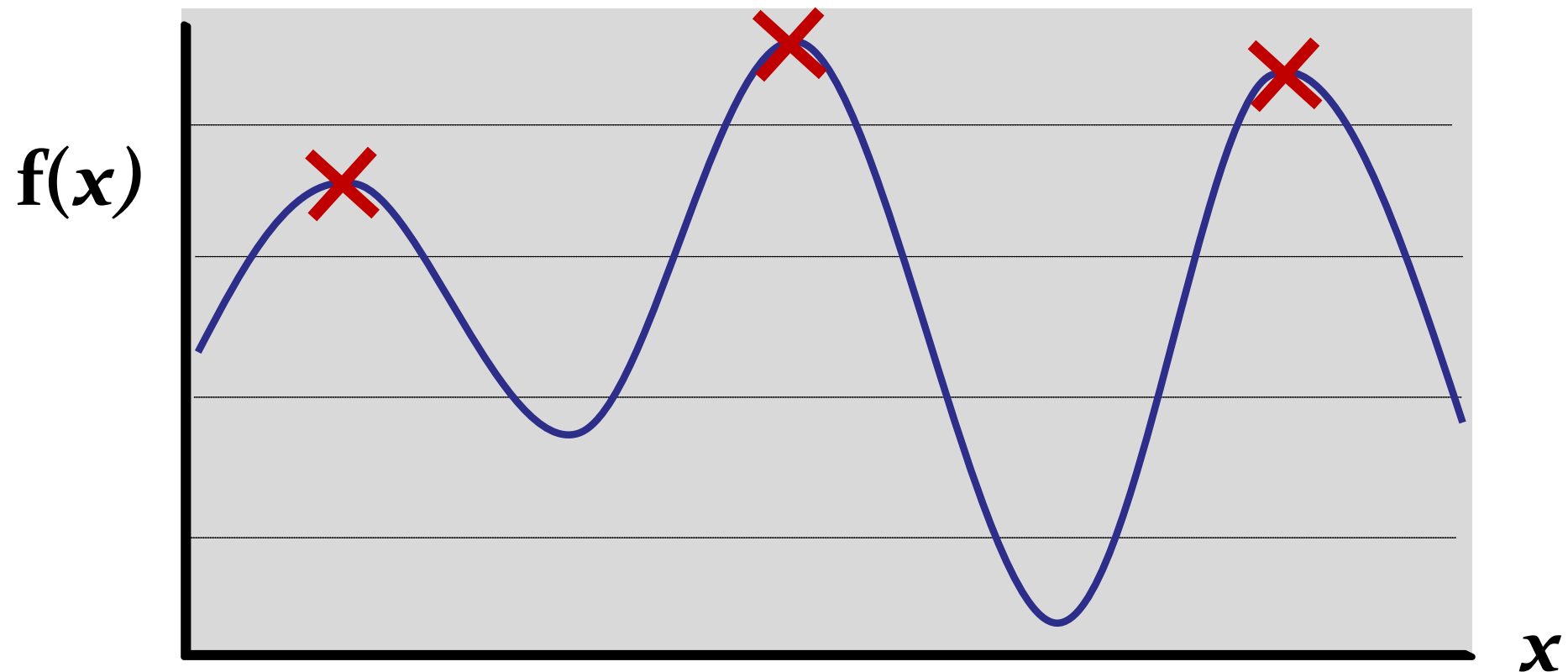
```
            end = mid
```

```
        else begin = mid+1
```

```
    return A[begin]
```

Peak Finding

Input: Some function $f(x)$



Output: local maximum

Peak Finding

$O(\log n)$ time

Input: Some array $A[1..n]$

2	4	9	2	11	6	23	4	6	8	17	5
---	---	---	---	----	---	----	---	---	---	----	---

FindPeak(A, n)

if $A[n/2]$ is a peak **then return** $n/2$

else if $A[n/2+1] > A[n/2]$ **then**

FindPeak ($A[n/2+1..n], n/2$)

else if $A[n/2-1] > A[n/2]$ **then**

FindPeak ($A[1..n/2-1], n/2$)

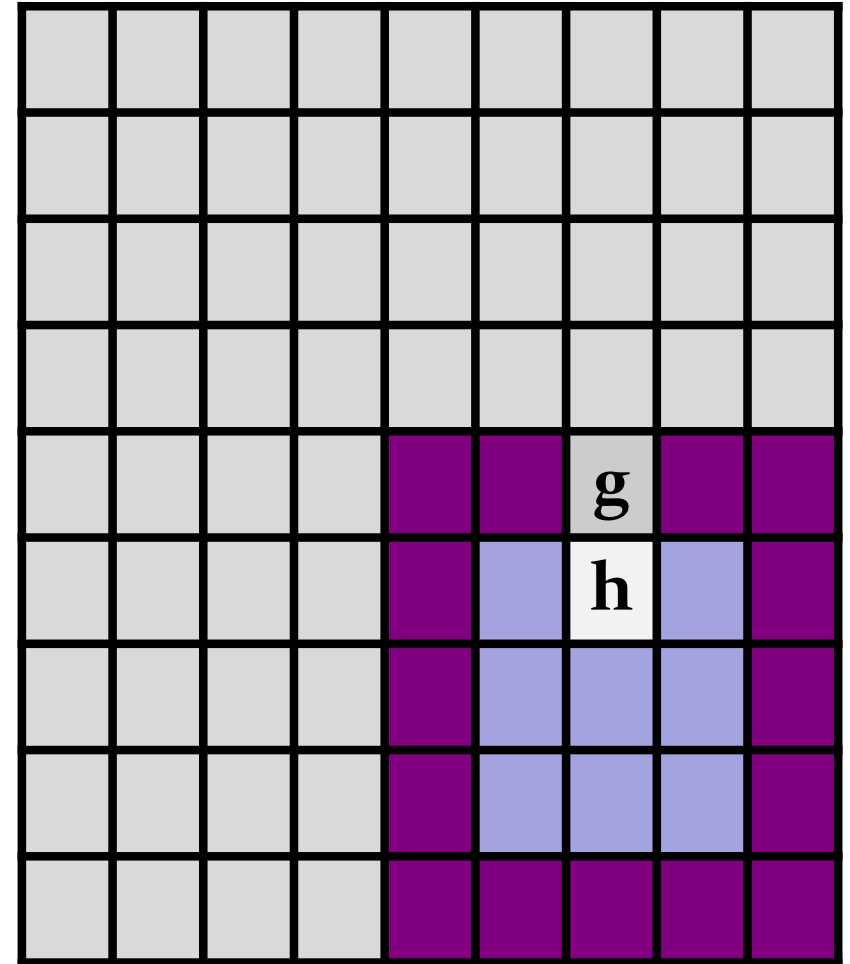
2D Algorithm

$O(m + n)$ time

Divide-and-Conquer

1. Find MAX element on border + cross.
2. If found a peak, DONE.
3. Else:

Recurse on quadrant containing element bigger than MAX.



Binary Search

Sorted array: $A[1..n]$

2	4	4	5	6	7	8	9	11	17	23	28
---	---	---	---	---	---	---	---	----	----	----	----

Not just for searching arrays:

- Assume a complicated function:

`int complicatedFunction(int s)`

- Assume the function is always increasing:

`complicatedFunction(i) < complicatedFunction(i+1)`

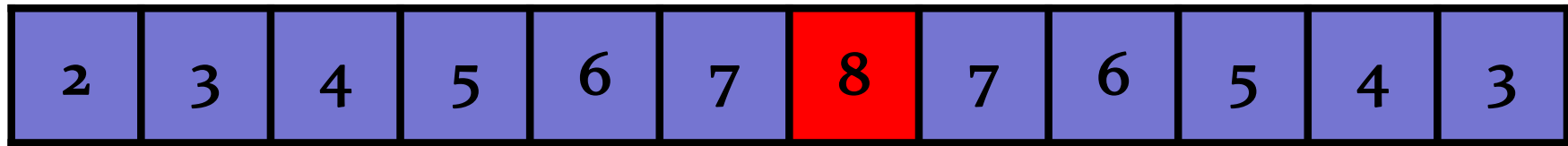
- Find the minimum value j such that:

`complicatedFunction(j) > 100`

Finding a maximum / minimum

Convex / concave function : $f[1..n]$

Problem Set 2



Not just for arrays:

- Assume a complicated function:

`int complicatedFunction(int s)`

- Assume the function is convex (or concave)
- Find the maximum / minimum value:

`minj(complicatedFunction(j))`

Why function maximization?

Two short answers:

Optimization is everywhere

- Maximize revenue
- Minimize cost
- Minimize time
- ...

Why function maximization?

Two short answers:

Optimization is everywhere

- Maximize revenue
- Minimize cost
- Minimize time
- ...

Machine learning

- All about optimization!
- Find the best model for your data
- Find the best neural network for your problem.

Machine Learning Example

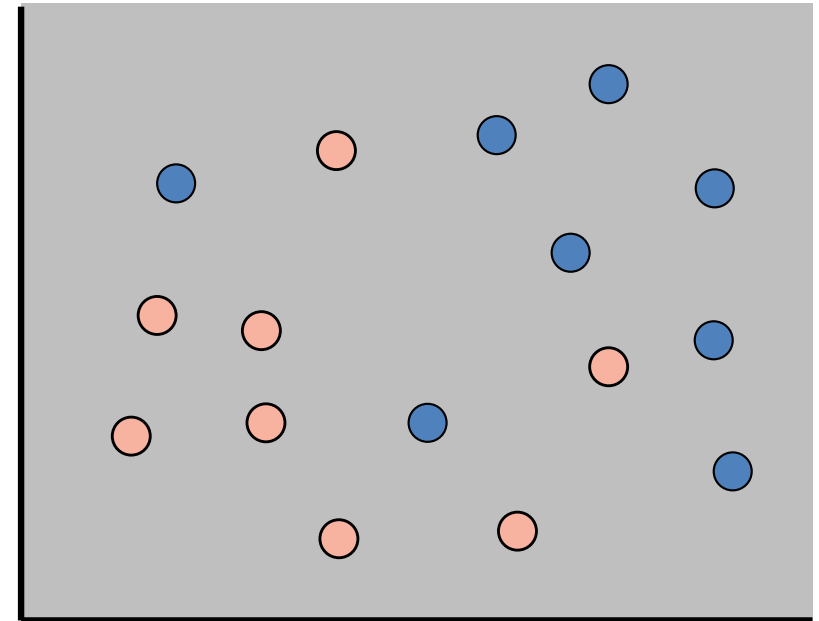
Classifier:

Input:

- A set of data points
- Each point has a label: 0 or 1

Output:

- A function that classifies points.

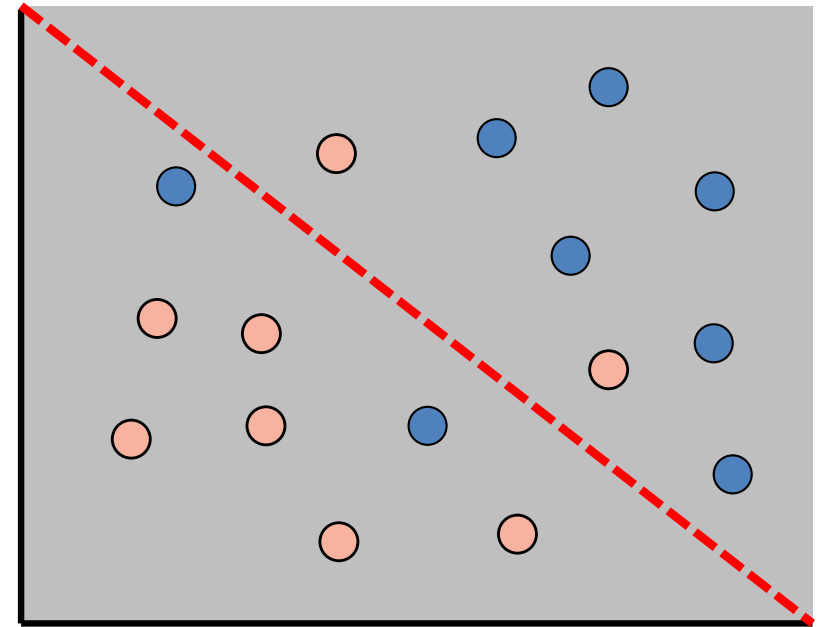


Machine Learning Example

Linear classifier:

Idea:

- Find a line: $ax + by = c$
- To classify point (x_j, y_j) :
 - if $(ax_j + by_j > c)$: **BLUE**
 - if $(ax_j + by_j < c)$: **RED**

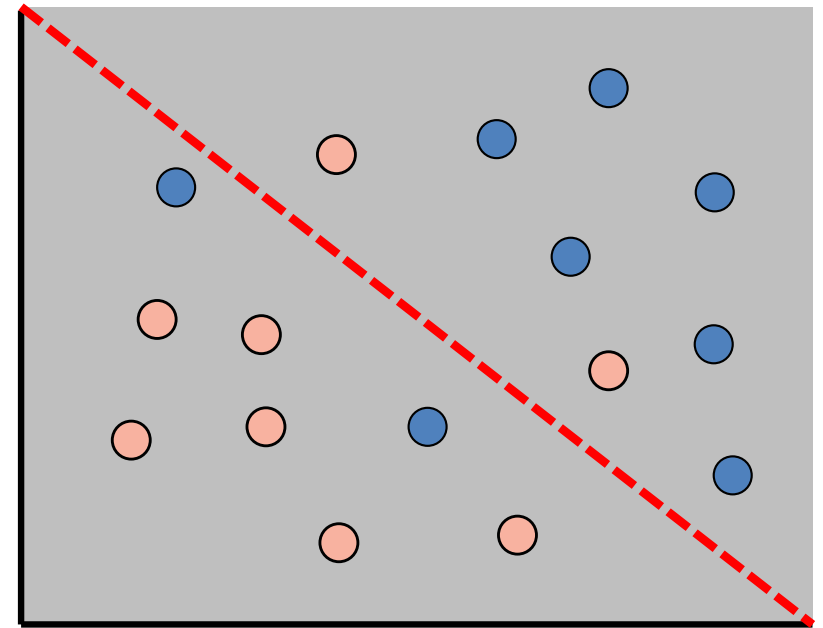


Machine Learning Example

Linear classifier:

Idea:

- Find a line: $ax + by + c = 0$
- To classify point (x_j, y_j) :
 - if $(ax_j + by_j + c > 0)$: **BLUE**
 - if $(ax_j + by_j + c < 0)$: **RED**



color for point (x, y)

Question:

Given data $(x_1, y_1, z_1), (x_2, y_2, z_1), \dots, (x_n, y_n, z_1)$

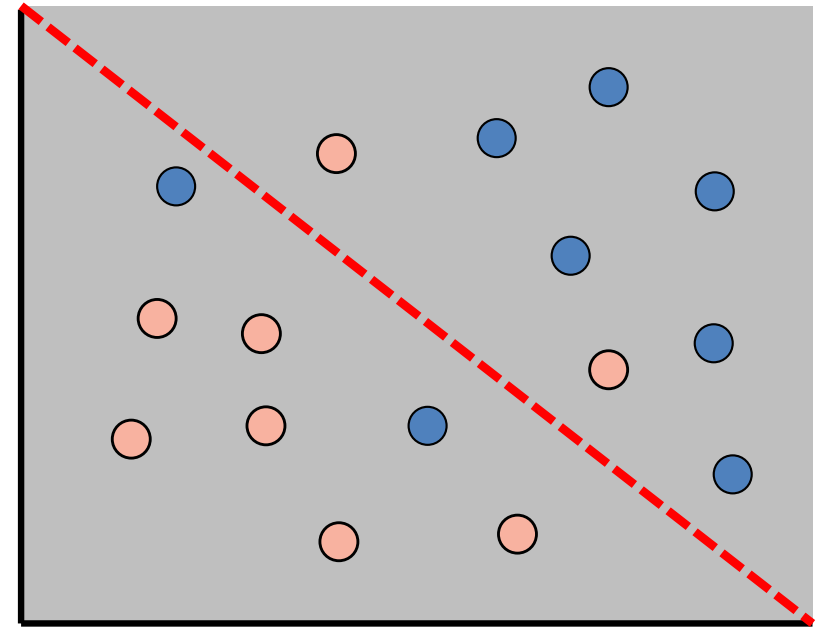
Find *best* values of (a, b, c)

Machine Learning Example

Linear classifier:

Idea:

Find a line: $ax + by + c = 0$



One way to make that precise:

Given data $(x_1, y_1, z_1), (x_2, y_2, z_1), \dots, (x_n, y_n, z_1)$

$$\max_{a,b,c} \prod_{i=1}^n h_{a,b,c}(x_i, y_i)^{z_i} \cdot (1 - h_{a,b,c}(x_i, y_i))^{1-z_i}$$

Likelihood function:

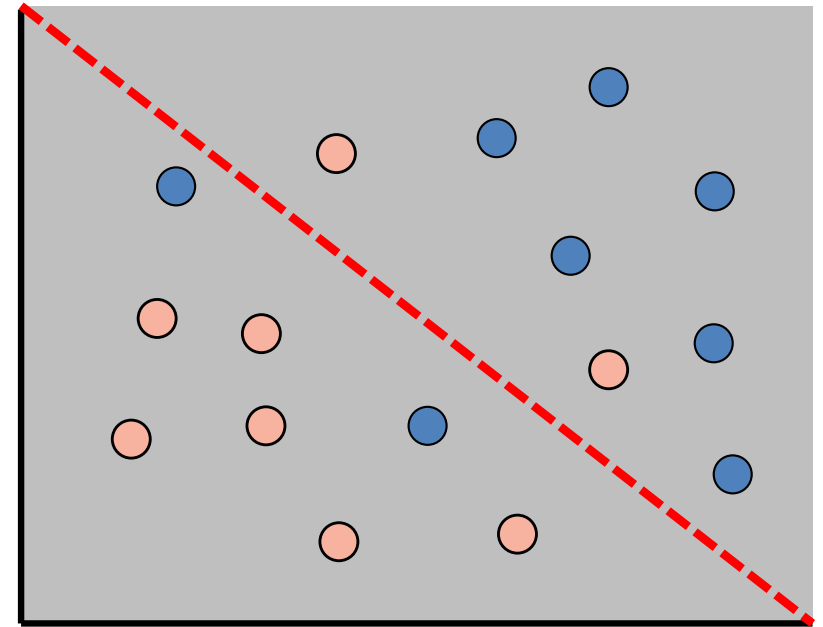
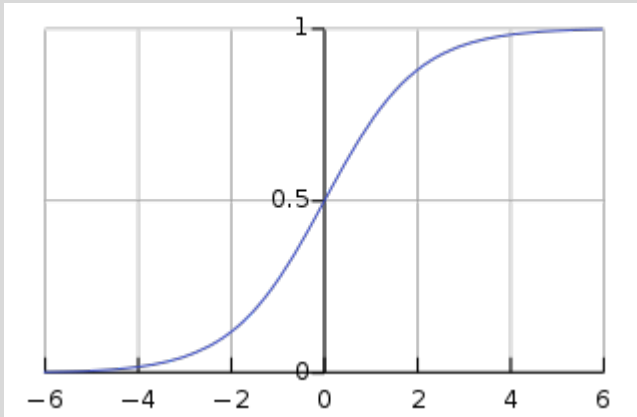
$h(x, y)$ = probability that point (x, y) is blue/1.

$1 - h(x, y)$ = probability that point (x, y) is red/0.

Machine Learning Example

Choice of **h** function?

$$h_{a,b,c}(x,y) = \frac{1}{1 + e^{-(ax+by+c)}}$$



Given data $(x_1, y_1, z_1), (x_2, y_2, z_1), \dots, (x_n, y_n, z_1)$

$$\max_{a,b,c} \prod_{i=1}^n h_{a,b,c}(x_i, y_i)^{z_i} \cdot (1 - h_{a,b,c}(x_i, y_i))^{1-z_i}$$

Likelihood function:

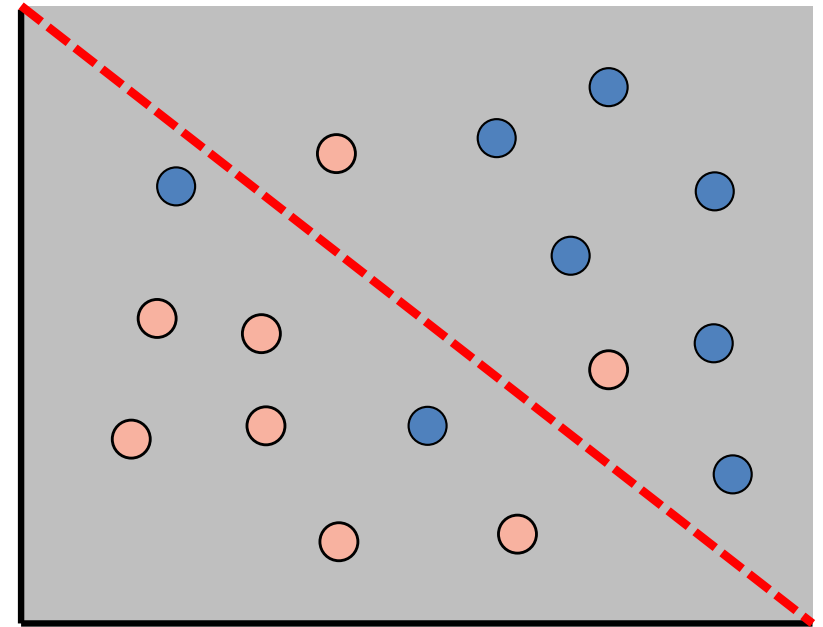
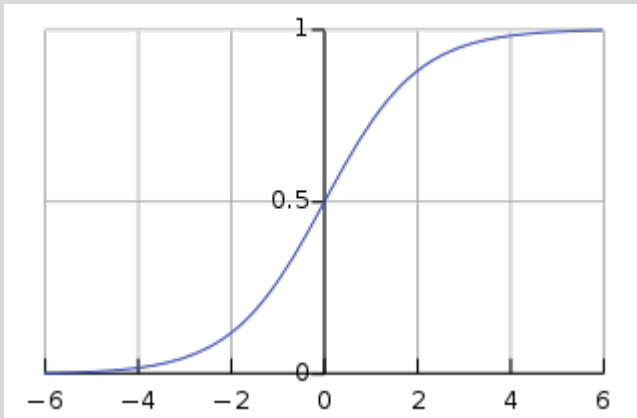
$h(x, y)$ = probability that point (x, y) is blue/1.

$1 - h(x, y)$ = probability that point (x, y) is red/0.

Machine Learning Example

Choice of **h** function?

$$h_{a,b,c}(x,y) = \frac{1}{1 + e^{-(ax+by+c)}}$$



Given data $(x_1, y_1, z_1), (x_2, y_2, z_1), \dots, (x_n, y_n, z_1)$

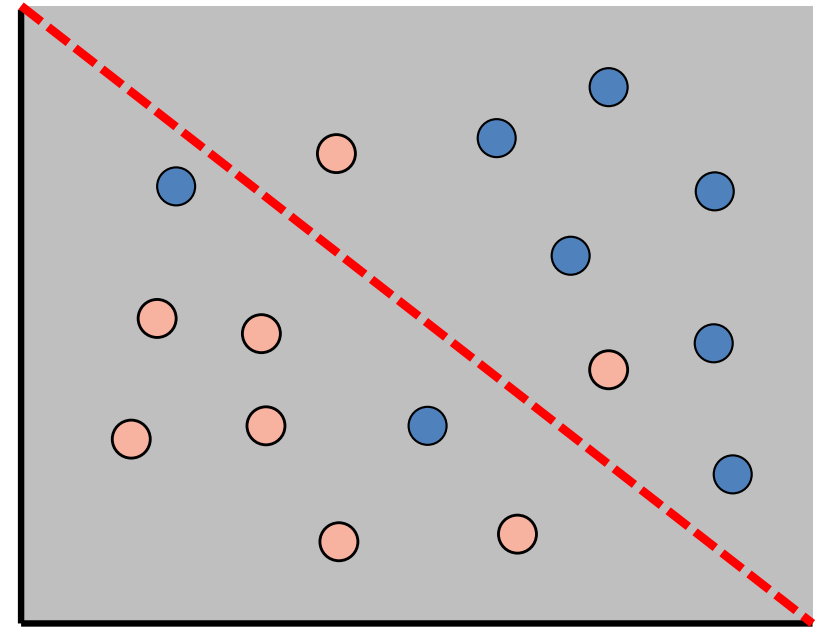
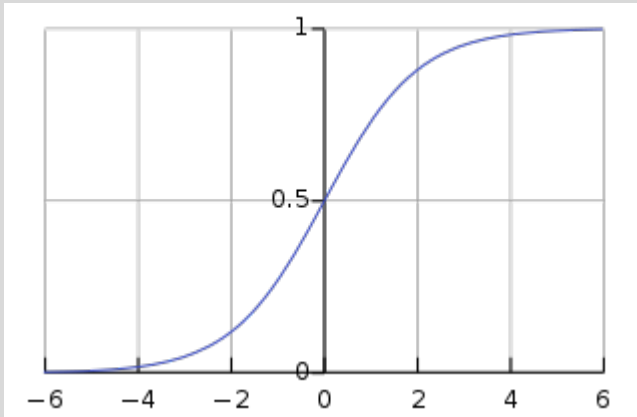
$$\max_{a,b,c} \sum_{i=1}^n z_i \log(h_{a,b,c}(x_i, y_i)) + (1 - z_i) \log(1 - h_{a,b,c}(x_i, y_i))$$

Log-Likelihood function:
log of the likelihood function.

Machine Learning Example

Choice of **h** function?

$$h_{a,b,c}(x,y) = \frac{1}{1 + e^{-(ax+by+c)}}$$



Given data $(x_1, y_1, z_1), (x_2, y_2, z_1), \dots, (x_n, y_n, z_1)$

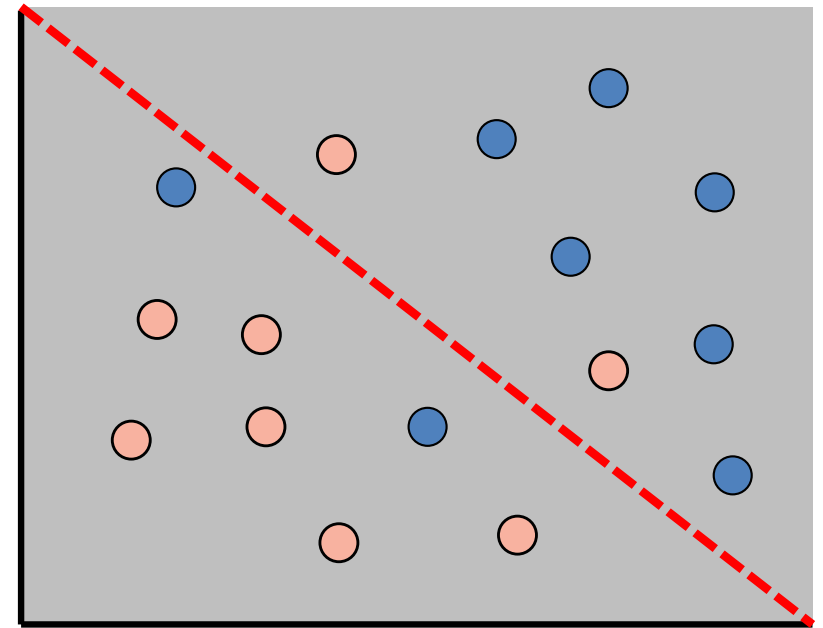
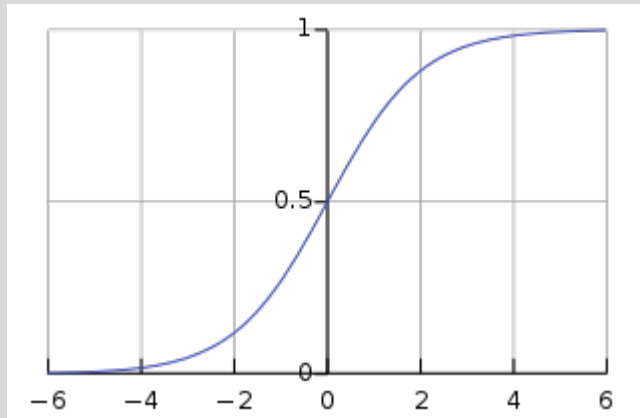
$$\max_{a,b,c} \sum_{i=1}^n z_i \log(h_{a,b,c}(x_i, y_i)) + (1 - z_i) \log(1 - h_{a,b,c}(x_i, y_i))$$

Logistic Regression

Machine Learning Example

Choice of **h** function?

$$h_{a,b,c}(x,y) = \frac{1}{1 + e^{-(ax+by+c)}}$$



$$\max_{a,b,c} \sum_{i=1}^n z_i \log(h_{a,b,c}(x_i, y_i)) + (1 - z_i) \log(1 - h_{a,b,c}(x_i, y_i))$$

Solution: **binary search?**

Finding a maximum / minimum

Convex / concave function : $f[1..n]$

3	4	5	7
2	8	12	9
1	4	8	7
0	2	6	3

What happens in d dimensions?

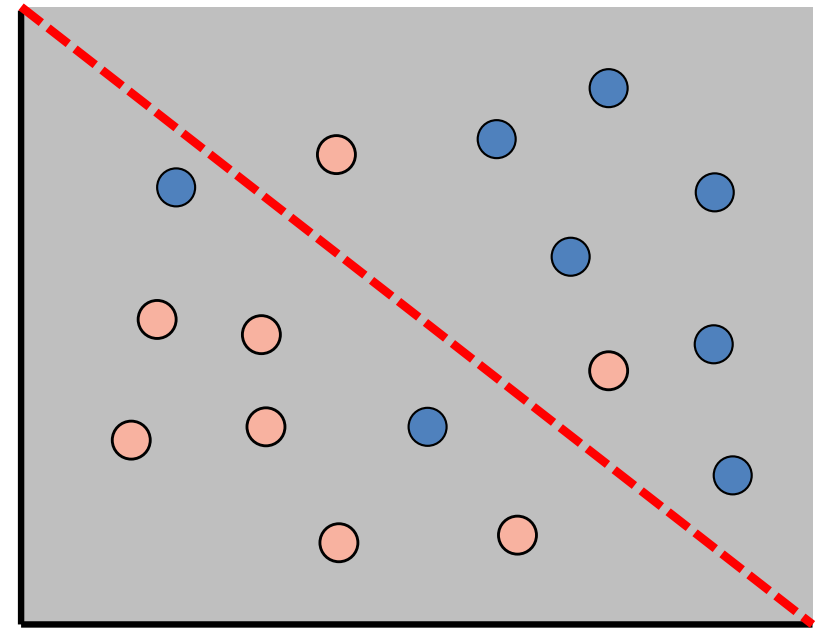
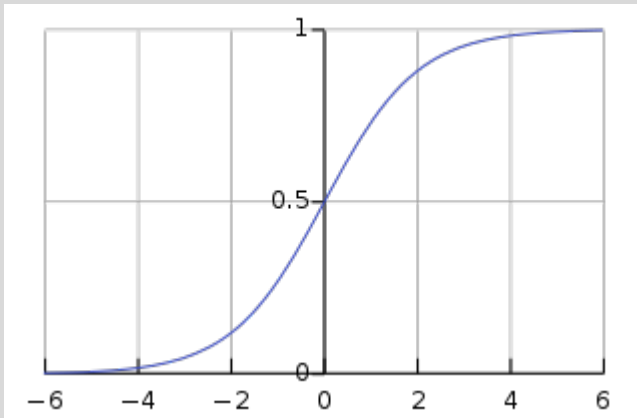
Dimension	Time
1	$O(\log n)$
2	$O(n)$
3	$O(n^2)$
...	...
d	$O(n^{d-1})$

Maximum distance
between two points in a
 d -dimensional cube
with side length n ::
 $O(dn)$

Machine Learning Example

Choice of **h** function?

$$h_{a,b,c}(x,y) = \frac{1}{1 + e^{-(ax+by+c)}}$$



$$\max_{a,b,c} \sum_{i=1}^n z_i \log(h_{a,b,c}(x_i, y_i)) + (1 - z_i) \log(1 - h_{a,b,c}(x_i, y_i))$$

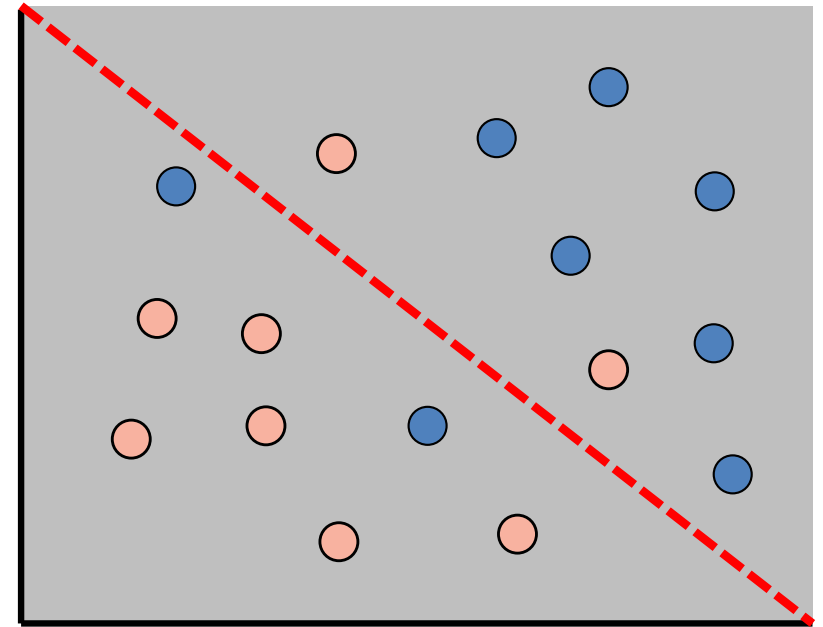
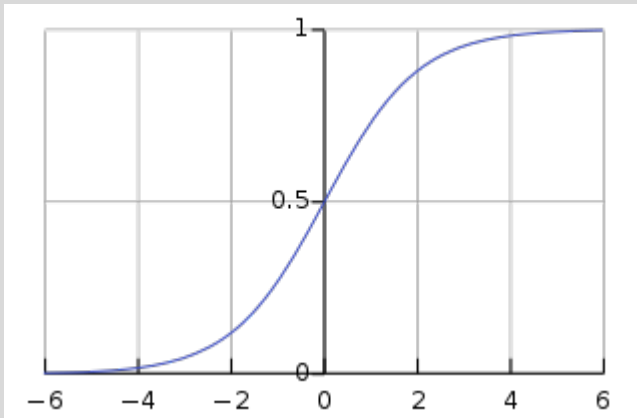
Solution: **binary search**?

- Slow (as dimensions get large).
- Not very robust to noise.

Machine Learning Example

Choice of **h** function?

$$h_{a,b,c}(x,y) = \frac{1}{1 + e^{-(ax+by+c)}}$$



Solution: set (partial) derivatives to zero

- ➔ Hard algebraically.
- ➔ Can approximate numerically
- ➔ E.g., Newton's method

$$\sum_{i=1}^n [z_i - h_{a,b,c}(x_i, y_i)] x_i = 0$$

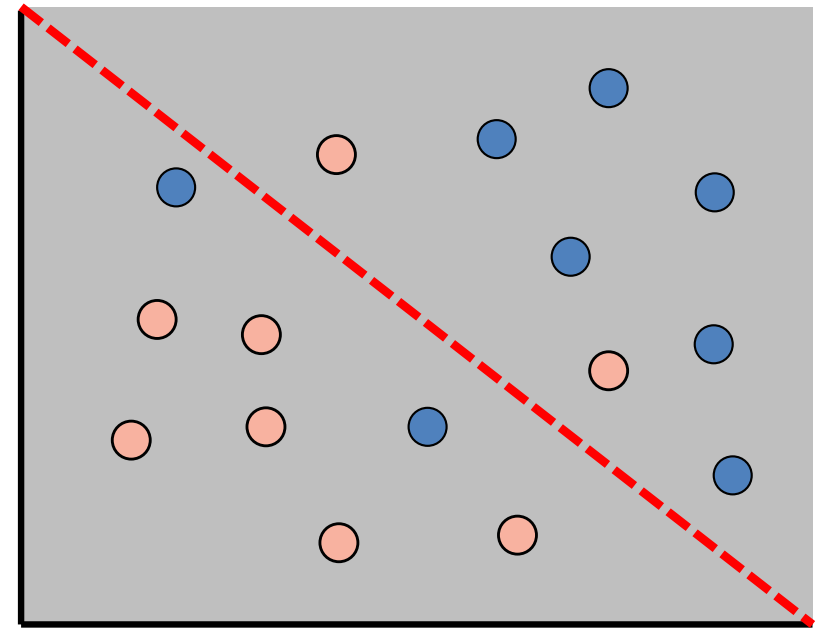
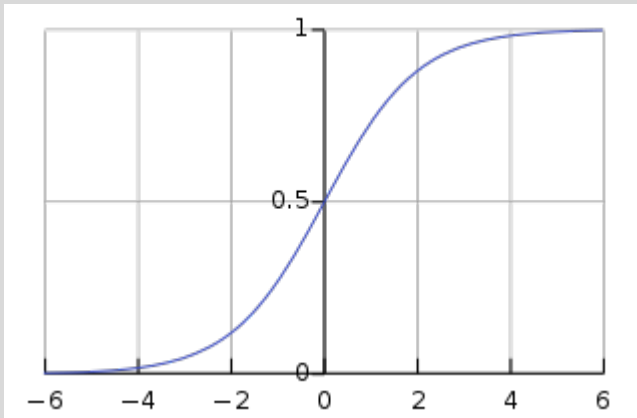
$$\sum_{i=1}^n [z_i - h_{a,b,c}(x_i, y_i)] y_i = 0$$

$$\sum_{i=1}^n [z_i - h_{a,b,c}(x_i, y_i)] = 0$$

Machine Learning Example

Choice of **h** function?

$$h_{a,b,c}(x,y) = \frac{1}{1 + e^{-(ax+by+c)}}$$



$$\max_{a,b,c} \sum_{i=1}^n z_i \log(h_{a,b,c}(x_i, y_i)) + (1 - z_i) \log(1 - h_{a,b,c}(x_i, y_i))$$

Solution: approximate directly

- ➔ Hard algebraically.
- ➔ Can approximate numerically
- ➔ E.g., gradient descent

Summary

Today: Optimization

Finding a maximum / minimum

- *Key aspect of machine learning*
- *Binary search isn't always good*

Newton's method

- *Second order approach*

Gradient descent

- *First order approach*

When I say:

“Find x s.t. $f(x) = a^3 - x^{1/3} = 0$ ”

You can hear:

“Find **parameters** so that the partial derivatives of the log-likelihood function equal **0**”

When I say:

“Find x to minimize $f(x) = 2x^2 + 4x + 9$ ”

You can hear:

“Find **parameters** to maximize the log-likelihood function”

Pop Quiz

What does this code do?

```
float HIDDEN(float number)
{
    int i;
    float x2, y;
    float threehalfs = 1.5f;


    x2 = number*0.5f;
    y = number;
    i = Float.toIntBits(number);    // evil floating bit level hacking
    i = 0x5f3759df - (i >> 1);      // what the *bleep*?
    y = Float.intBitsToFloat(i);
    y = y * (threehalfs - (x2*y*y))  // first iteration
    return y;
}
```

Pop Quiz

What does this code do?

```
float HIDDEN(float number)
{
    int i;
    float x2, y;
    float threehalfs = 1.5f;

    x2 = number*0.5f;
    y = number;
    i = Float.toIntBits(number); // evil floating bit level hacking
    i = 0x5f3759df - (i >> 1);   // what the *bleep*?
    y = Float.intBitsToFloat(i);
    y = y * (threehalfs - (x2*y*y)) // first iteration
    return y;
}
```



Original comments...

Pop Quiz

What does this code do?

```
float HIDDEN(float number)
```

```
{
```

```
    int i;
```

```
    float x2, y;
```

```
    float threehalfs = 1.5f;
```

```
    x2 = number*0.5f;
```

```
    y = number;
```

```
    i = Float.toIntBits(number);    // evil floating bit level hacking
```

```
    i = 0x5f3759df - (i >> 1);    // what the *bleep*?
```

```
    y = Float.intBitsToFloat(i);
```

```
    y = y * (threehalfs - (x2*y*y)) // first iteration
```

```
    return y;
```

```
}
```

Fake java:

Returns bit representation of a float.

IEEE float:

- 1 sign bit
- 8 (biased) exponent bits
- 23 significand bits

Fake java:

Turns bits back into a float.

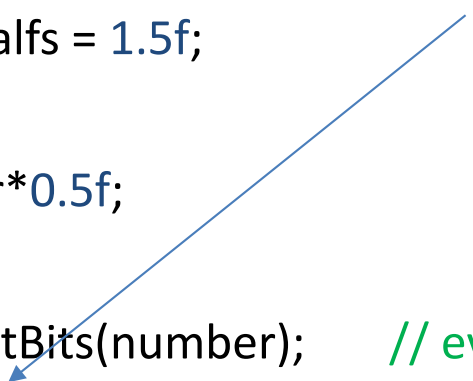
Pop Quiz

What does this code do?

```
float HIDDEN(float number)
{
    int i;
    float x2, y;
    float threehalfs = 1.5f;

    x2 = number*0.5f;
    y = number;
    i = Float.toIntBits(number);    // evil floating bit level hacking
    i = 0x5f3759df - (i >> 1);      // what the *bleep*?
    y = Float.intBitsToFloat(i);
    y = y * (threehalfs - (x2*y*y)) // first iteration
    return y;
}
```

Magic number!



Pop Quiz

What does this code do?

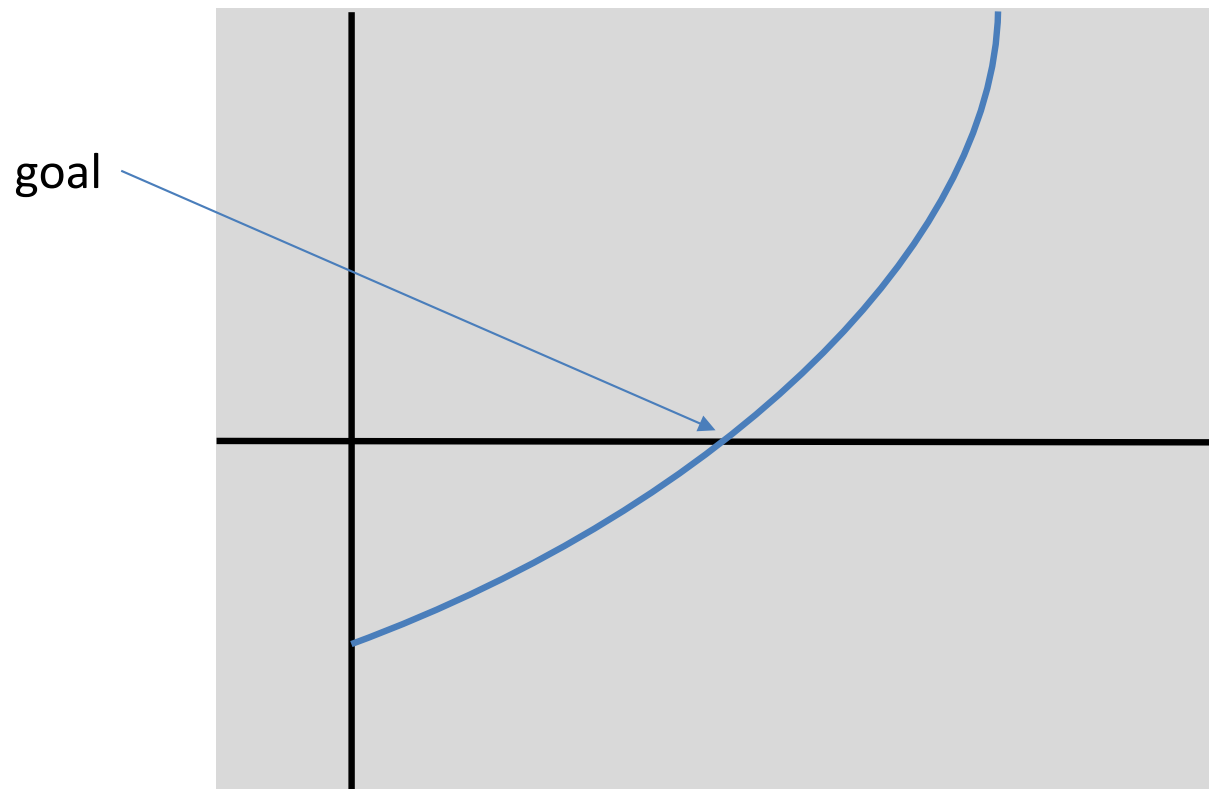
```
float HIDDEN(float number)
{
    int i;
    float x2, y;
    float threehalfs = 1.5f;

    x2 = number*0.5f;
    y = number;
    i = Float.toIntBits(number);    // evil floating bit level hacking
    i = 0x5f3759df - (i >> 1);      // what the *bleep*?
    y = Float.intBitsToFloat(i);
    y = y * (threehalfs - (x2*y*y))  // first iteration
    return y;
}
```

Newton's Method

Goal:

Find a root of $f(x) = 0$ via successive approximation:

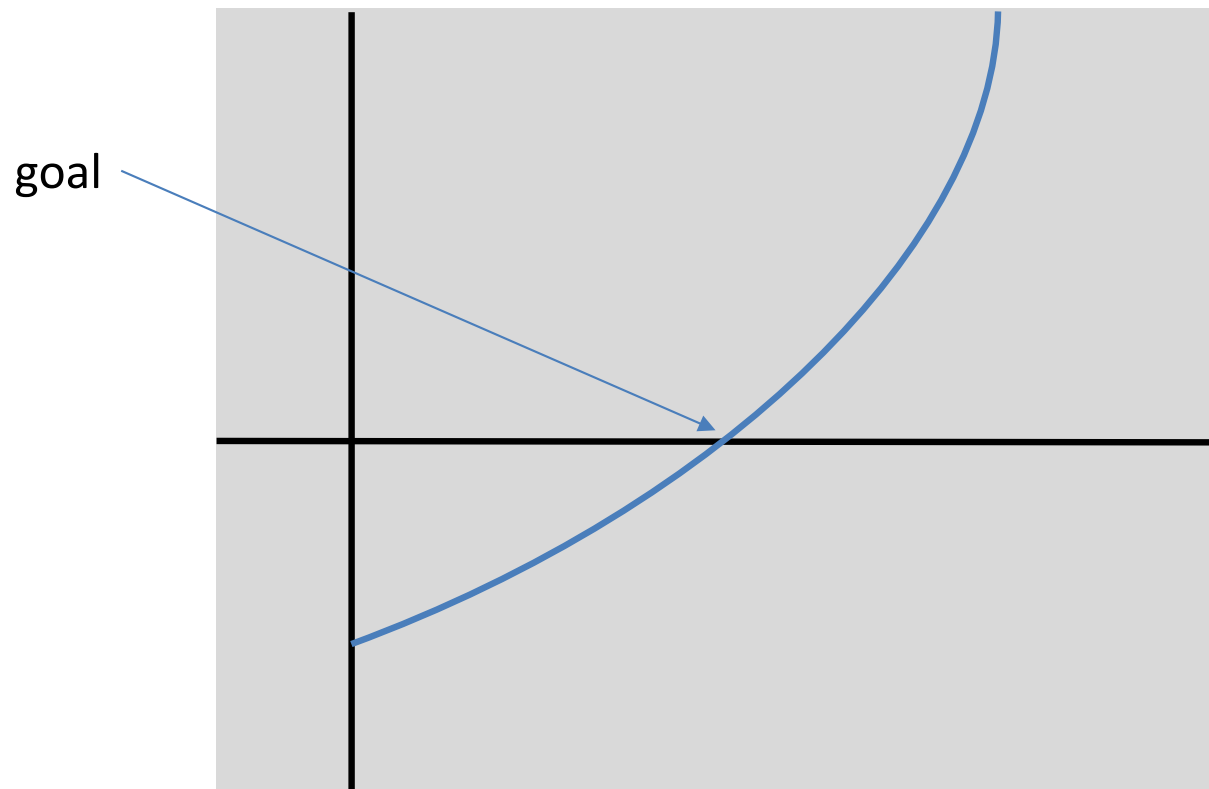


Newton's Method

Goal:

Note: for this example in one dimension, we could use binary search!

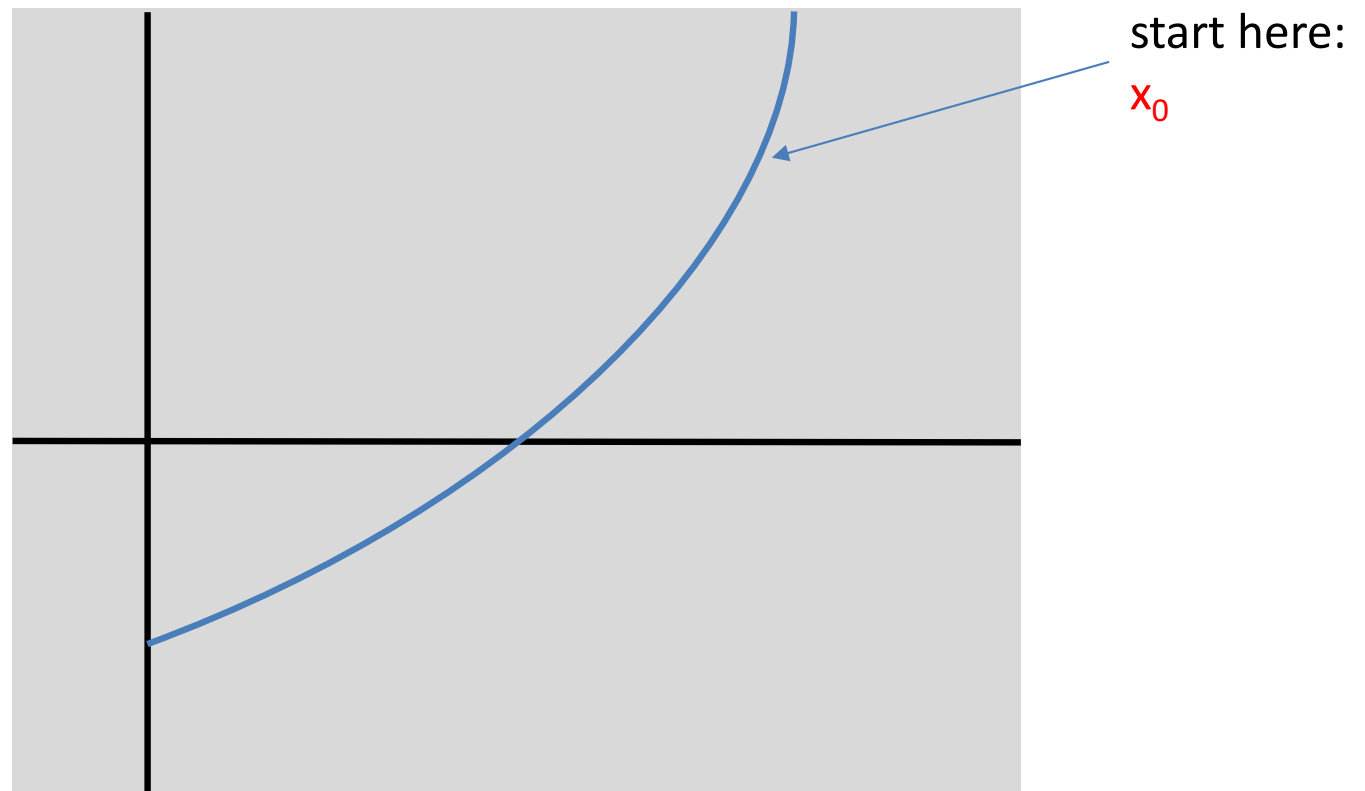
Find a root of $f(x) = 0$ via successive approximation:



Newton's Method

Goal:

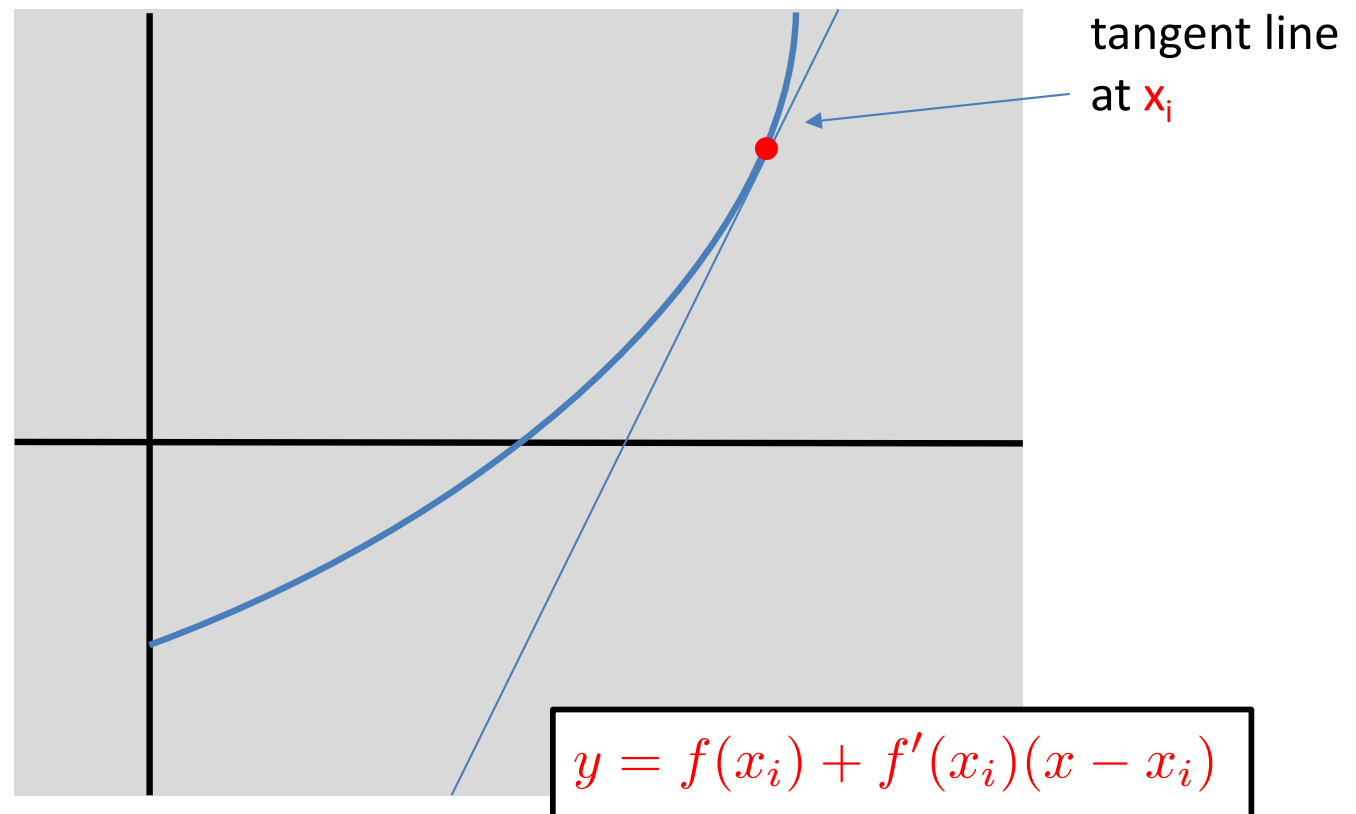
Find a root of $f(x) = 0$ via successive approximation:



Newton's Method

Goal:

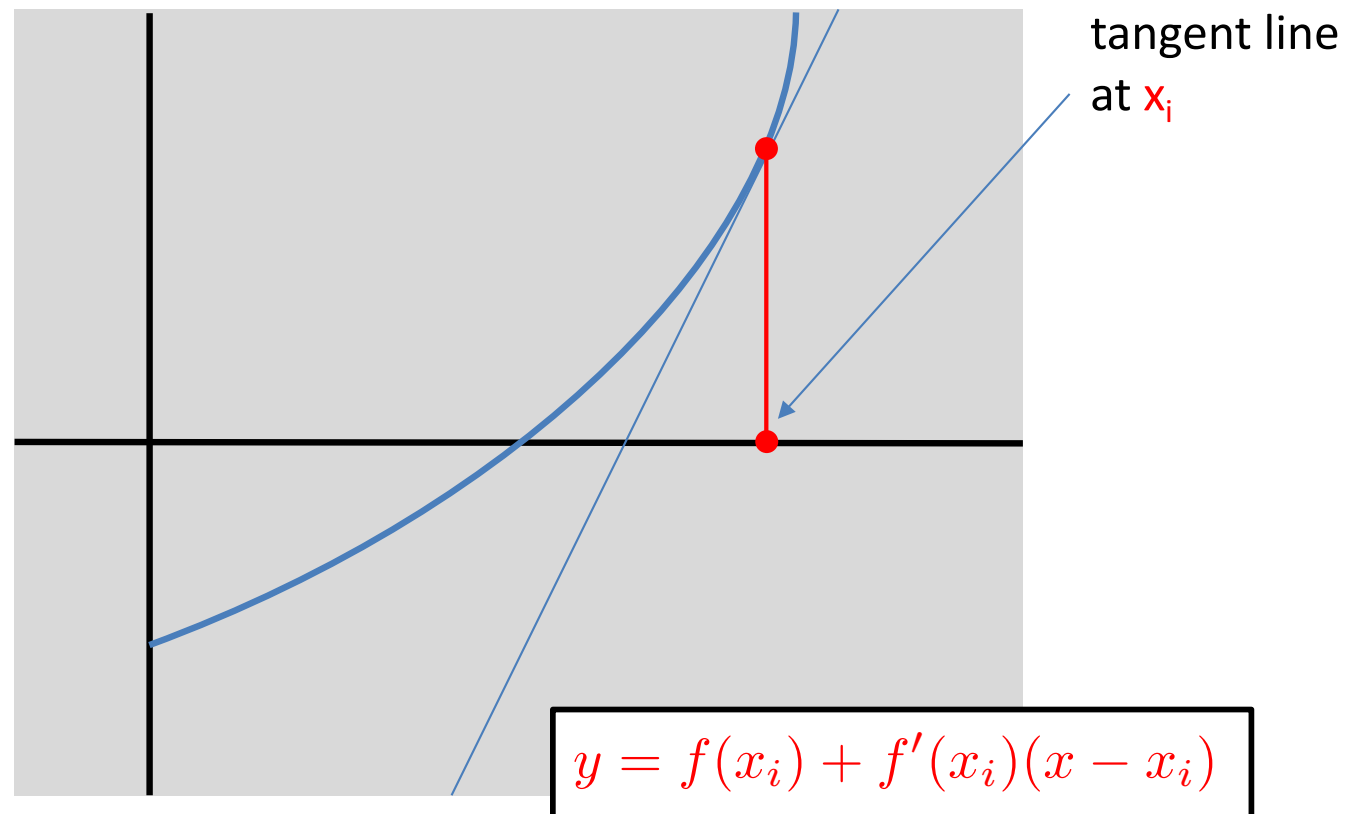
Find a root of $f(x) = 0$ via successive approximation:



Newton's Method

Goal:

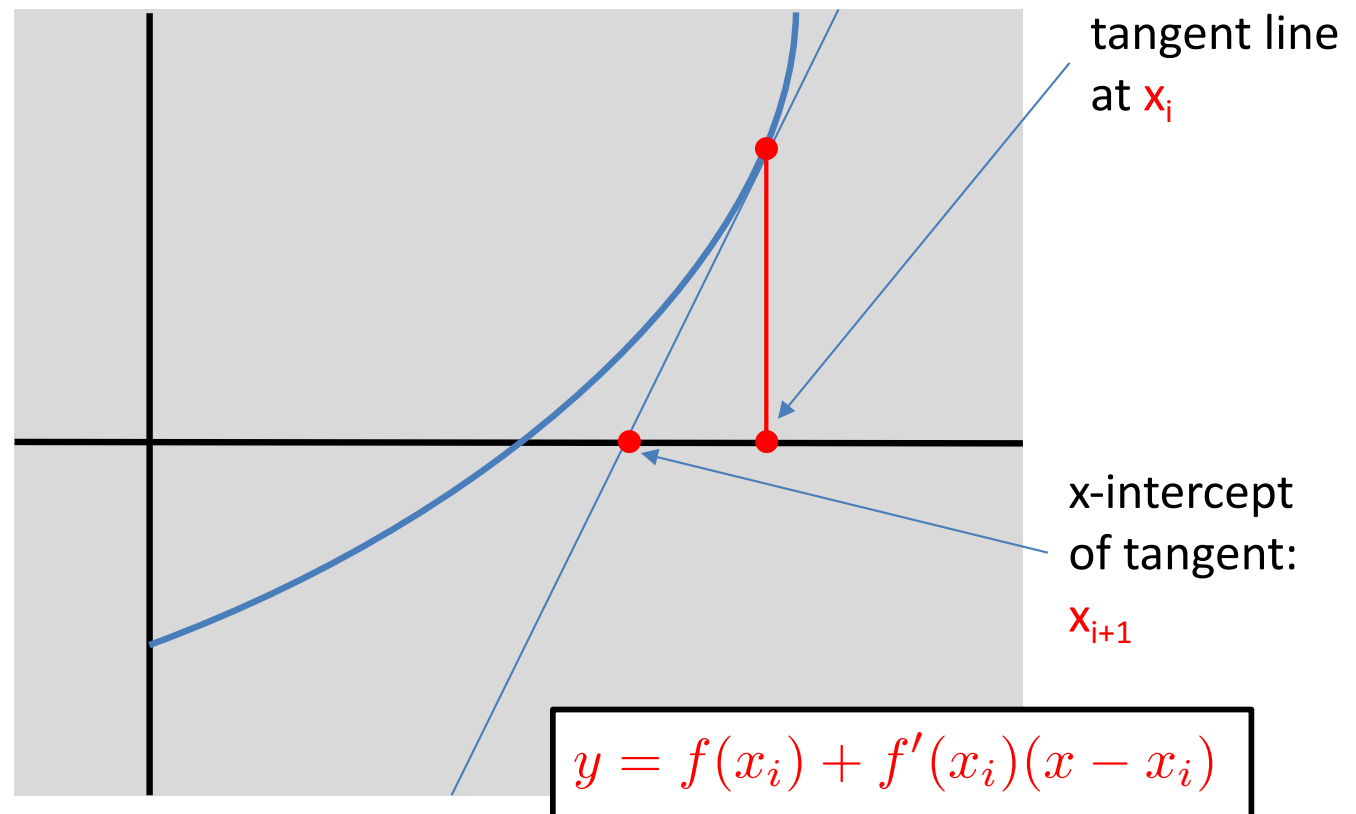
Find a root of $f(x) = 0$ via successive approximation:



Newton's Method

Goal:

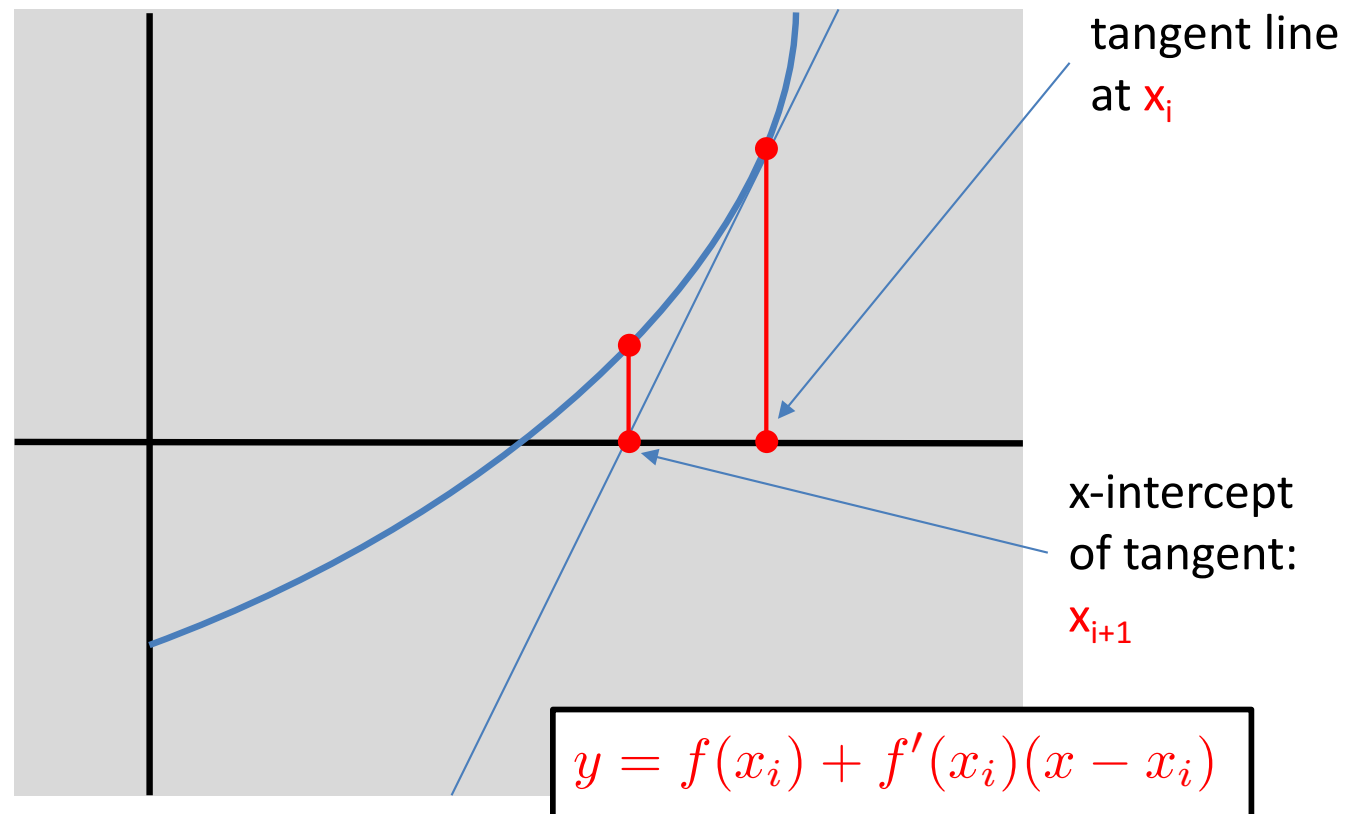
Find a root of $f(x) = 0$ via successive approximation:



Newton's Method

Goal:

Find a root of $f(x) = 0$ via successive approximation:

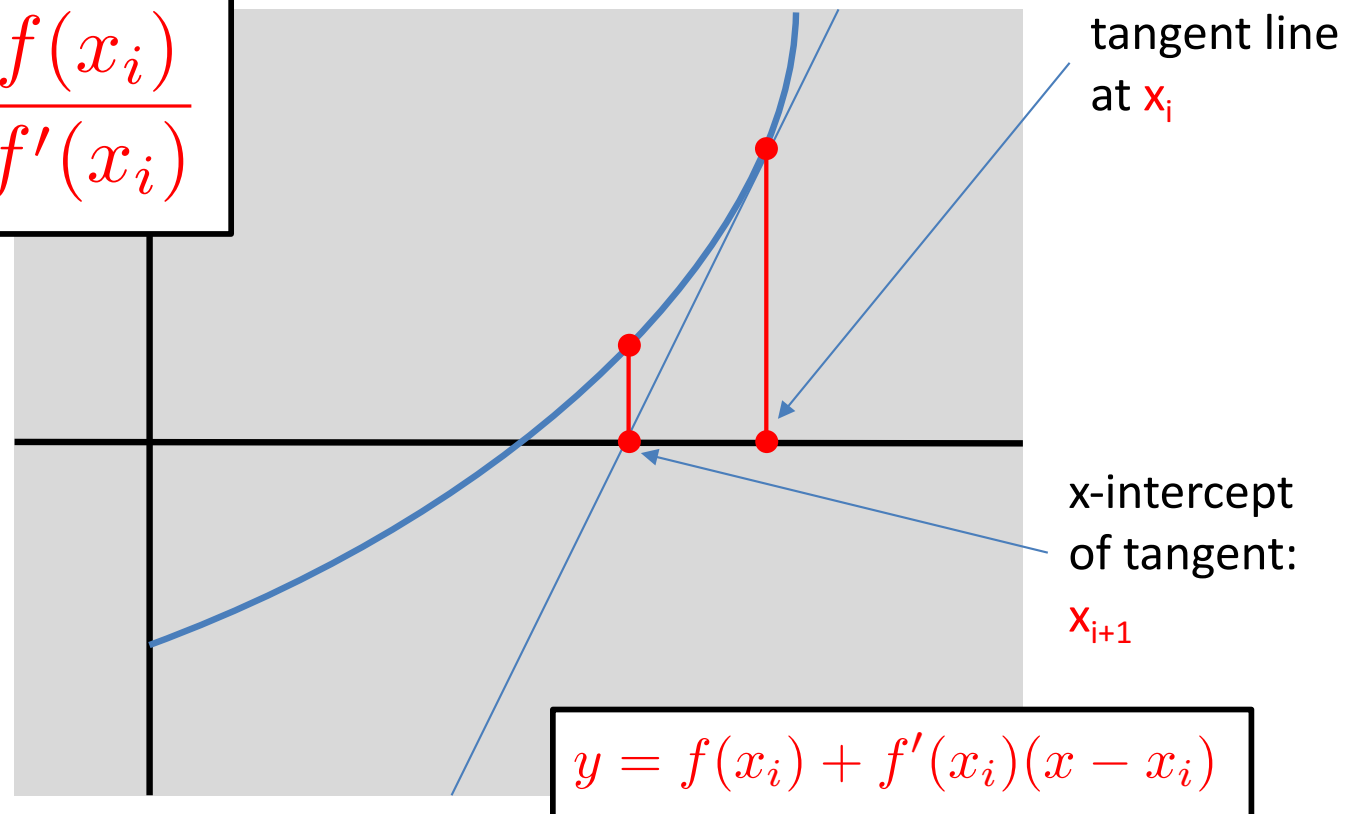


Newton's Method

Goal:

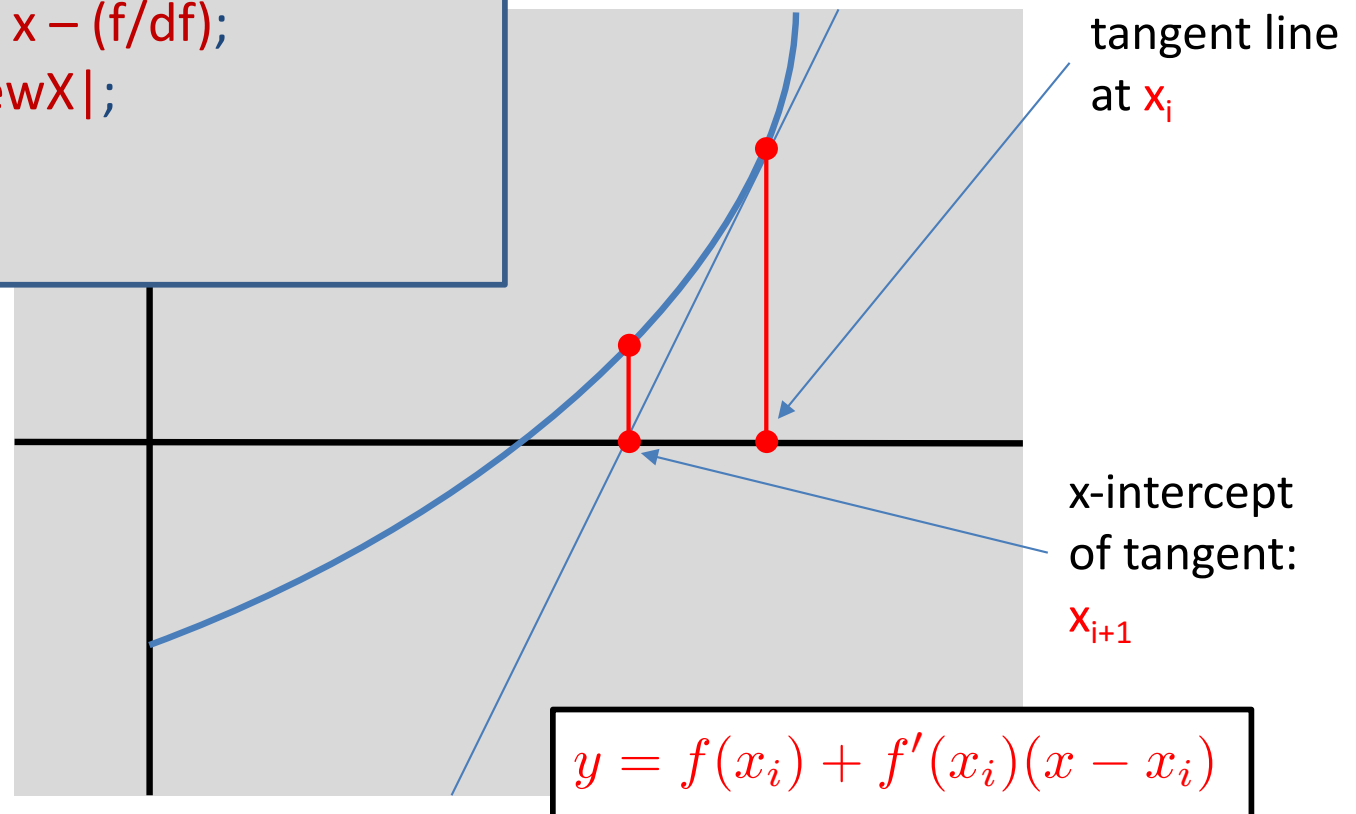
Find a root of $f(x) = 0$ via successive approximation:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$



Newton's Method

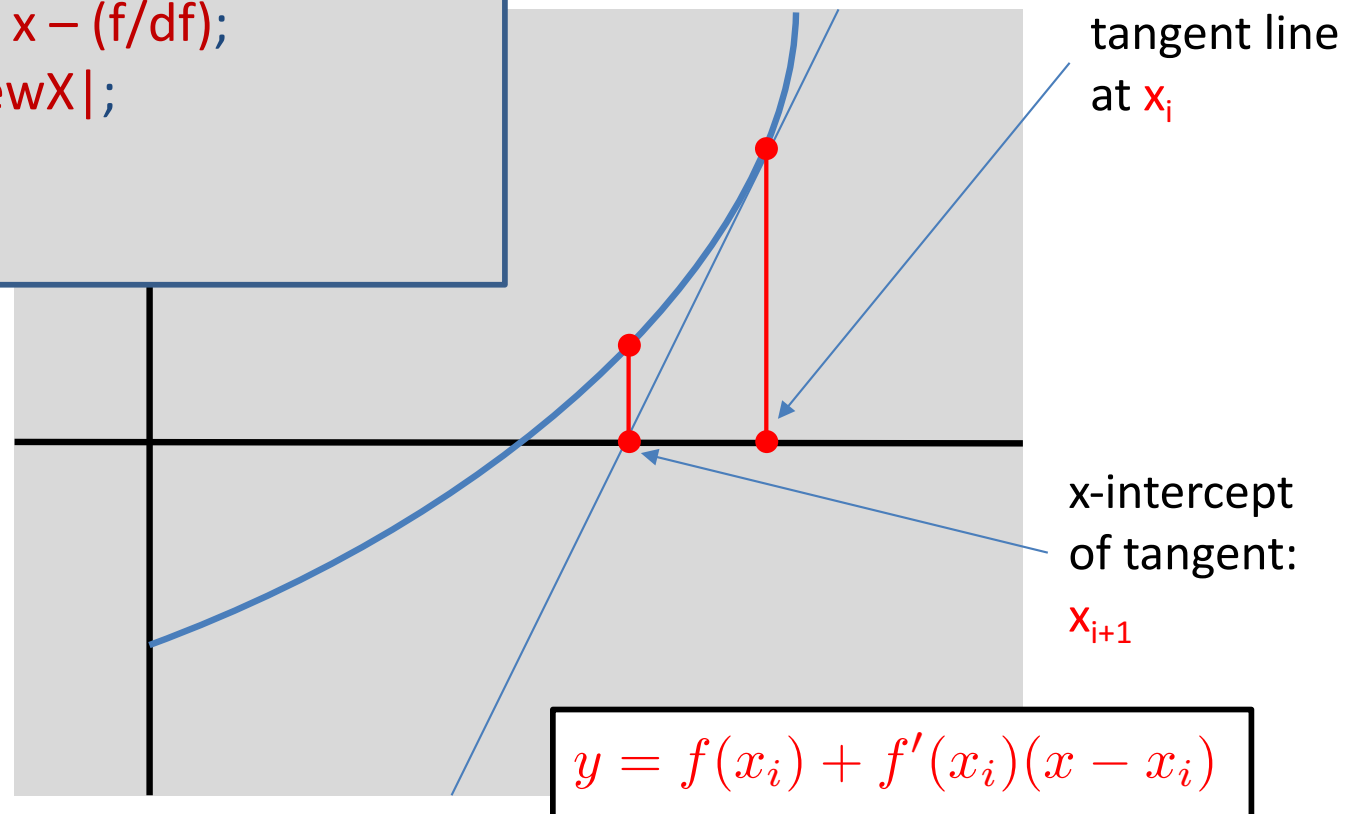
```
findRoot(float x, float error) {  
    diff = error+1;  
    while (diff > error){  
        float f = function(x);  
        float df = derivative(x);  
        float newX = x - (f/df);  
        diff = |x - newX|;  
        x = newX;  
    }  
    return x  
}
```



Newton's Method

```
findRoot(float x, float error) {  
    diff = error+1;  
    while (diff > error){  
        float f = function(x);  
        float df = derivative(x);  
        float newX = x - (f/df);  
        diff = |x - newX|;  
        x = newX;  
    }  
    return x  
}
```

How accurate an answer do you want?



Newton's Method

```
findRoot(float x, float error) {  
    diff = error+1;  
    while (diff > error){  
        float f = function(x);  
        float df = derivative(x);  
        float newX = x - (f/df);  
        diff = |x - newX|;  
        x = newX;  
    }  
    return x  
}
```

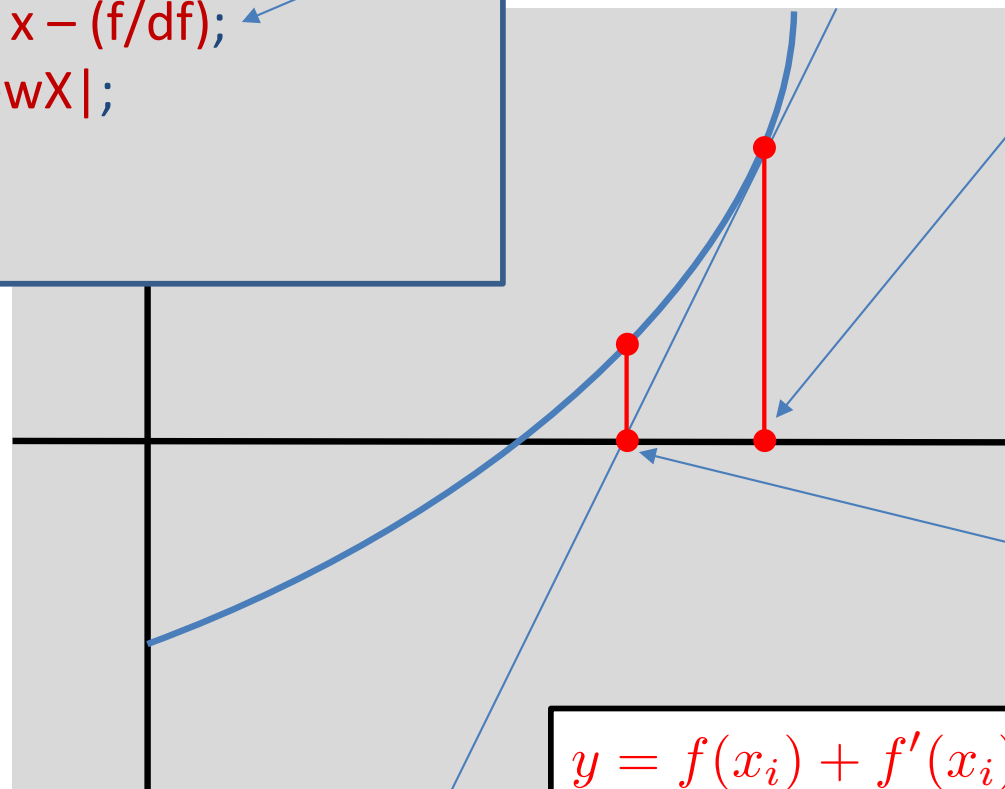
How accurate an answer do you want?

Newton step

tangent line
at x_i

x-intercept
of tangent:
 x_{i+1}

$$y = f(x_i) + f'(x_i)(x - x_i)$$



Newton's Method

Example: compute $a^{1/3}$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Function: $f(x) = a^{1/3} - x$

Derivative: $f'(x) = -1$

Newton's Method

Example: compute $a^{1/3}$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Function: $f(x) = a^{1/3} - x$

Derivative: $f'(x) = -1$

Update: $x_{i+1} = x_i + (a^{1/3} - x_i) = a^{1/3}$

Correct, but not very useful...

Newton's Method

Example: compute $a^{1/3}$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Function: $f(x) = a - x^3$

Derivative: $f'(x) = -3x^2$

Newton's Method

Example: compute $a^{1/3}$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Function: $f(x) = a - x^3$

Derivative: $f'(x) = -3x^2$

$$\begin{aligned}\text{Update: } x_{i+1} &= x_i - \frac{a - x_i^3}{-3x_i^2} = x_i + \frac{a}{3x_i^2} - (2/3)x_i \\ &= \frac{2x_i}{3} + \frac{a}{3x_i^2}\end{aligned}$$

Newton's Method

How fast does it converge?

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$x_{i+1} = \frac{2x_i}{3} + \frac{a}{3x_i^2}$$

Function: $f(x) = a - x^3$

Derivative: $f'(x) = -3x^2$

Quadratic convergence:

Every iteration, the number of correct digits doubles.

Iteration	Value
0	2
1	2.6750000000000003
2	2.5333260546772647
3	2.5251076782078234
4	2.525080872118423
5	2.525080871833849
TRUE:	2.525080871833849

Newton's Method

How fast does it converge?

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$x_{i+1} = \frac{2x_i}{3} + \frac{a}{3x_i^2}$$

Function: $f(x) = a - x^3$

Derivative: $f'(x) = -3x^2$

Quadratic convergence:

Every iteration, the number of correct digits doubles.

d digits of accuracy



$O(\log d)$ iterations

error $\leq 10^{-67}$



7 iterations

Convergence analysis

Assume:

$$\epsilon_i < 1$$

$$x_i = a^{1/3}(1 + \epsilon_i)$$

Newton's Method iteration

$$x_{i+1} = \frac{2x_i}{3} + \frac{a}{3x_i^2}$$

$$= \frac{2a^{1/3}(1 + \epsilon_i)}{3} + \frac{a}{3a^{2/3}(1 + \epsilon_i)^2}$$

plug in approximation assumption

$$= \frac{2a(1 + \epsilon_i)^3 + a}{3a^{2/3}(1 + \epsilon_i)^2}$$

algebra

$$= a^{1/3} \cdot \left[\frac{2(1 + \epsilon_i)^3 + 1}{3(1 + \epsilon_i)^2} \right]$$

$$= a^{1/3} \cdot \left[\frac{3(1 + \epsilon)^2 + 2(1 + \epsilon_i)^3 - 3(1 + \epsilon)^2 + 1}{3(1 + \epsilon)^2} \right]$$

$$= a^{1/3} \cdot \left[1 + \frac{2(1 + \epsilon_i)^3 - 3(1 + \epsilon)^2 + 1}{3(1 + \epsilon)^2} \right]$$

$$= a^{1/3} \cdot \left[1 + \frac{2\epsilon_i^3 + 3\epsilon_i^2}{3(1 + \epsilon)^2} \right]$$

quadratic convergence

$$\begin{aligned} \epsilon_{i+1} &= \frac{2\epsilon_i^3 + 3\epsilon_i^2}{3(1 + \epsilon)^2} \\ &\leq \left(\frac{5}{3} \right) \epsilon^2 \end{aligned}$$

Newton's Method

How fast does it converge?

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$x_{i+1} = \frac{2x_i}{3} + \frac{a}{3x_i^2}$$

Function: $f(x) = a - x^3$

Derivative: $f'(x) = -3x^2$

Quadratic convergence:

Every iteration, the number of correct digits doubles.

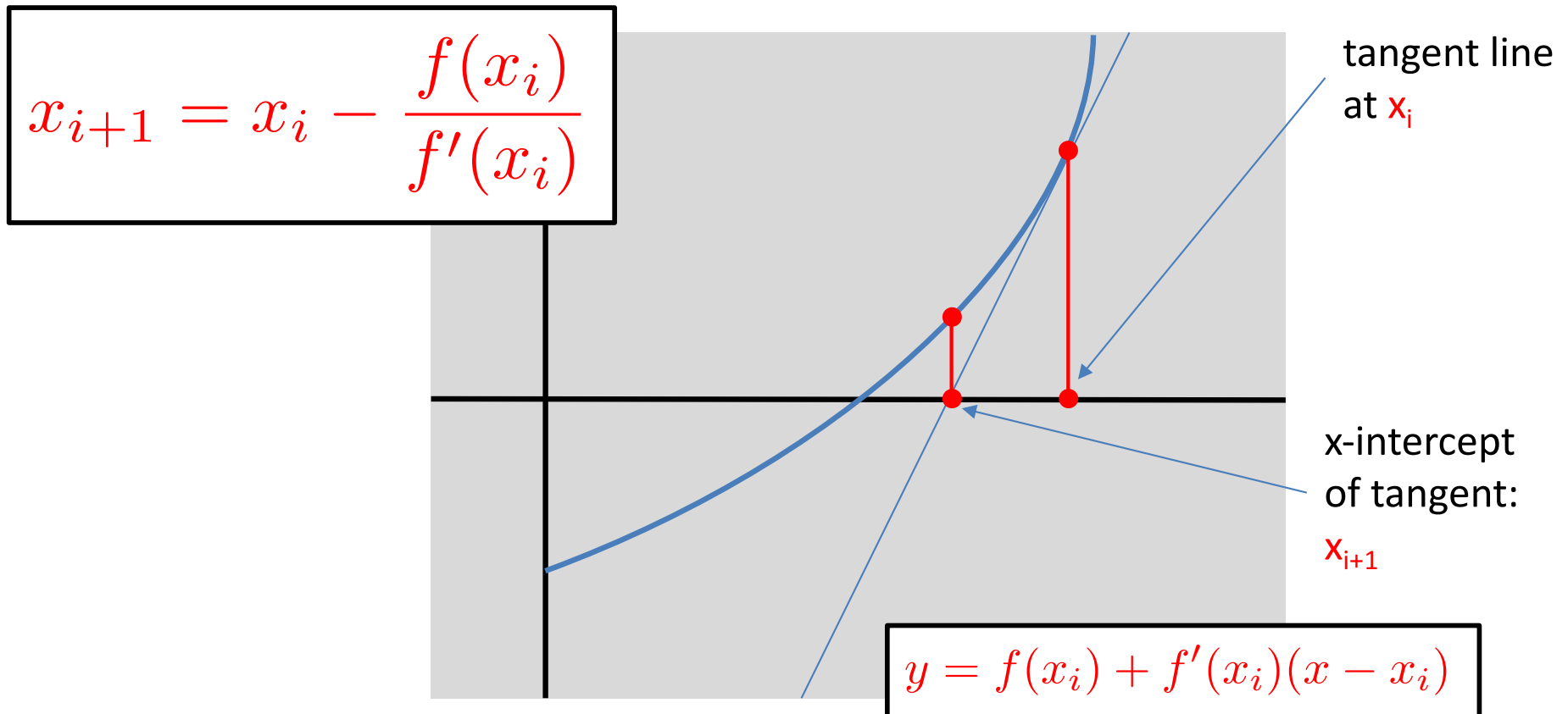
General conditions for quadratic convergence:

Function f continuously differentiable, derivative non-zero at root, has a second derivative at root.

Newton's Method

Goal:

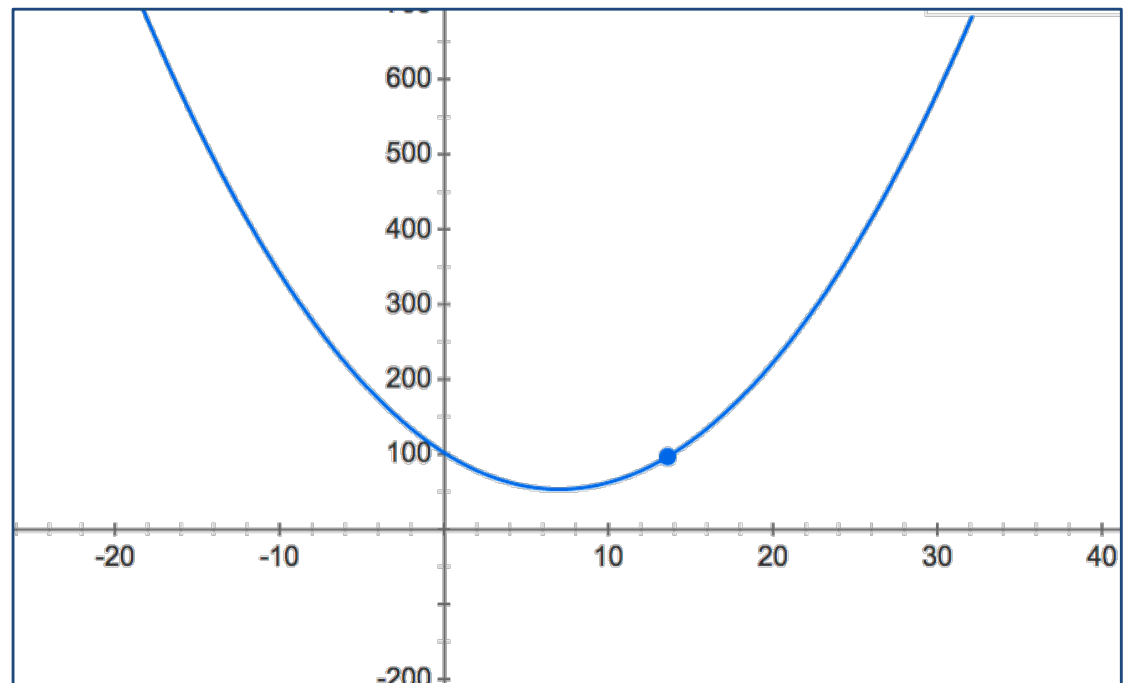
Find a root of $f(x) = 0$ via successive approximation:



Newton's Method Part 2

Goal:

Find the minimum value of $f(x)$ via successive approximation



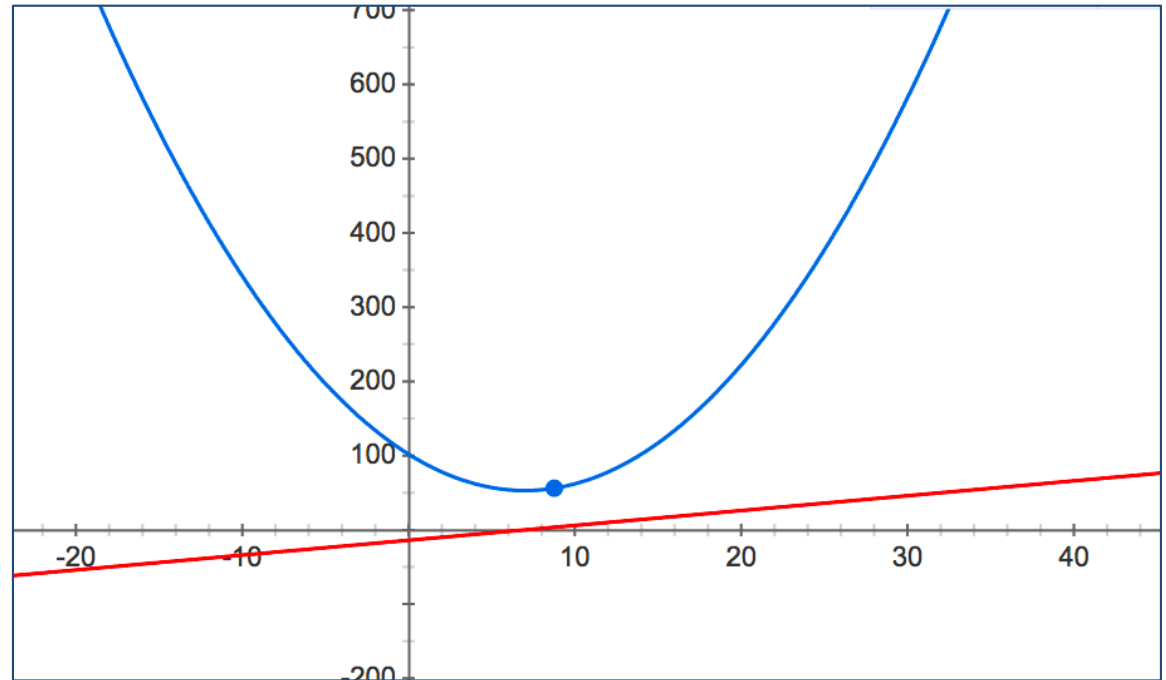
Newton's Method Part 2

Goal:

Find the minimum value of $f(x)$ via successive approximation:

Reduction:

Find a root of $f'(x) = 0$.



Newton's Method Part 2

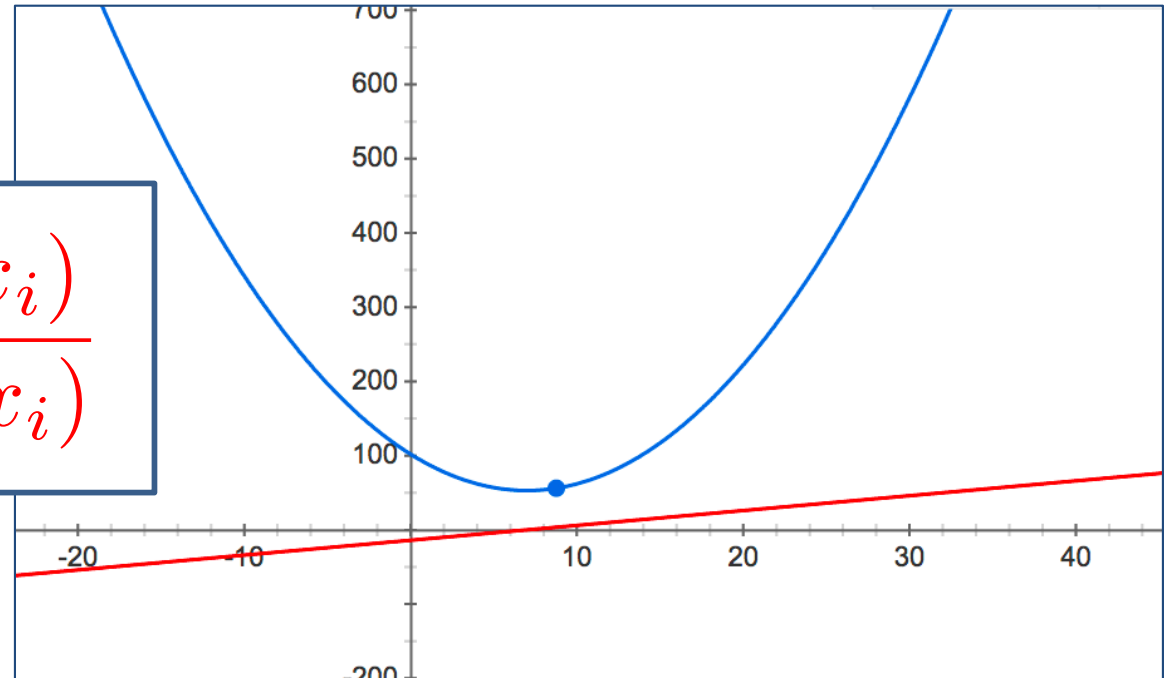
Goal:

Find the minimum value of $f(x)$ via successive approximation:

Reduction:

Find a root of $f'(x) = 0$.

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

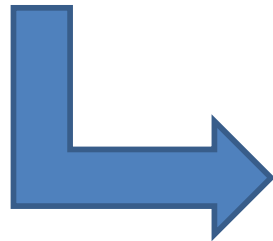


Newton's Method Continued

Example:

Find the maximum value of $f(\mathbf{x})$ via successive approximation:

$$\max_{a,b,c} \sum_{i=1}^n z_i \log(h_{a,b,c}(x_i, y_i)) + (1 - z_i) \log(1 - h_{a,b,c}(x_i, y_i))$$



$$\sum_{i=1}^n [z_i - h_{a,b,c}(x_i, y_i)] x_i = 0$$

$$\sum_{i=1}^n [z_i - h_{a,b,c}(x_i, y_i)] y_i = 0$$

$$\sum_{i=1}^n [z_i - h_{a,b,c}(x_i, y_i)] = 0$$

Newton's Method Continued

What about higher dimensions?

Find the minimum value of $f(x)$ via successive approximation:

Reduction:

Find a root of $f'(x) = 0$.

$$x_{i+1} = x_i - (\nabla f(x_i))(\nabla^2 f(x_i))^{-1}$$

derivative \rightarrow gradient (vector)

second derivative \rightarrow Hessian (matrix)
(division \rightarrow inverse)

Newton's Method Summary

Pros:

Fast convergence.

Relatively simple.

Compare to binary search:

- $O(d^c)$ cost per iteration, d is the dimension
- Quadratic convergence → few iterations

Cons:

Slow computation in higher dimensions.

Requires access to derivative, second derivative of f .

Example problem:

Computing the inverse of the Hessian...

Summary

Today: Optimization

Finding a maximum / minimum

- *Key aspect of machine learning*
- *Binary search isn't always good*

Newton's method

- *Second order approach*

Gradient descent

- *First order approach*

When I say:

“Find x s.t. $f(x) = a^3 - x^{1/3} = 0$ ”

You can hear:

“Find **parameters** so that the partial derivatives of the log-likelihood function equal **0**”

When I say:

“Find x to minimize
 $f(x) = 2x^2 + 4x + 9$ ”

You can hear:

“Find **parameters** to maximize the log-likelihood function”

Newton's Method → Gradient Descent

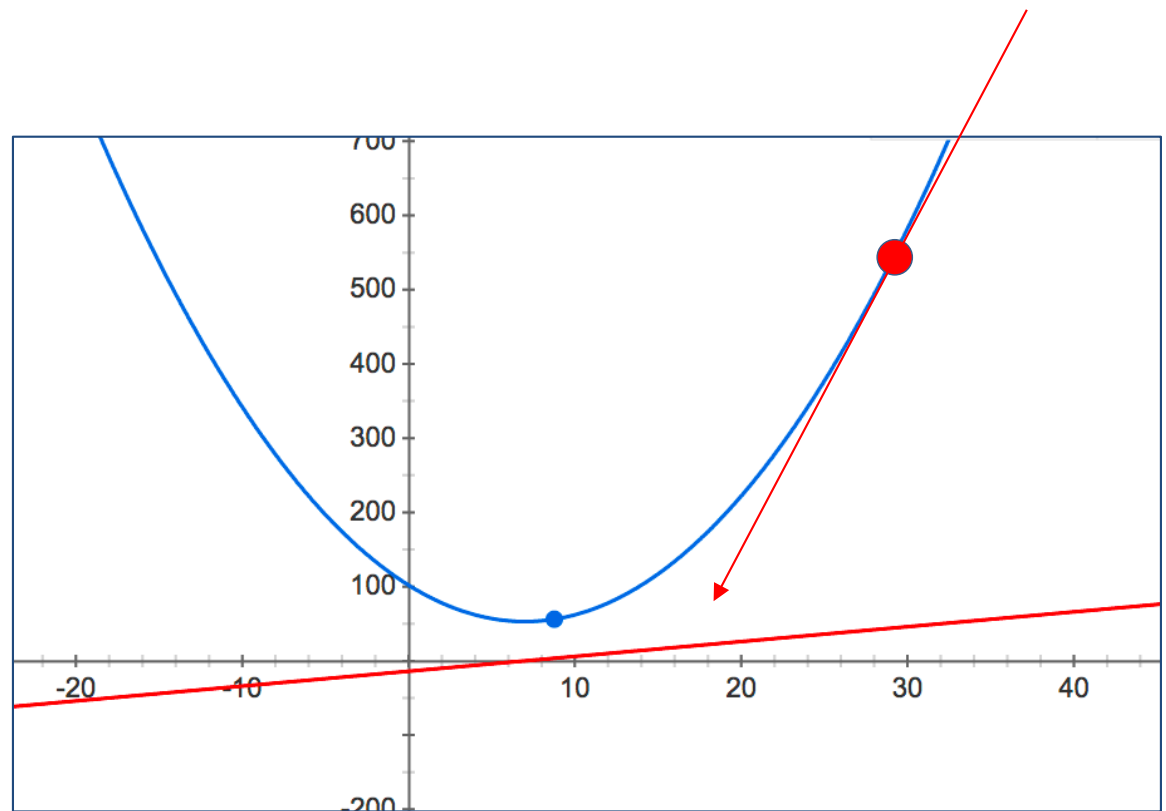
Goal:

Find the minimum value of $f(x)$ via successive approximation:

Reduction:

Find a root of $f'(x) = 0$.

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$



Newton's Method → Gradient Descent

Basic idea of Newton's Method:

- Gradient tells you which direction to move.
- Hessian tells you how far to move.

Problem:

Hessian is expensive to compute.

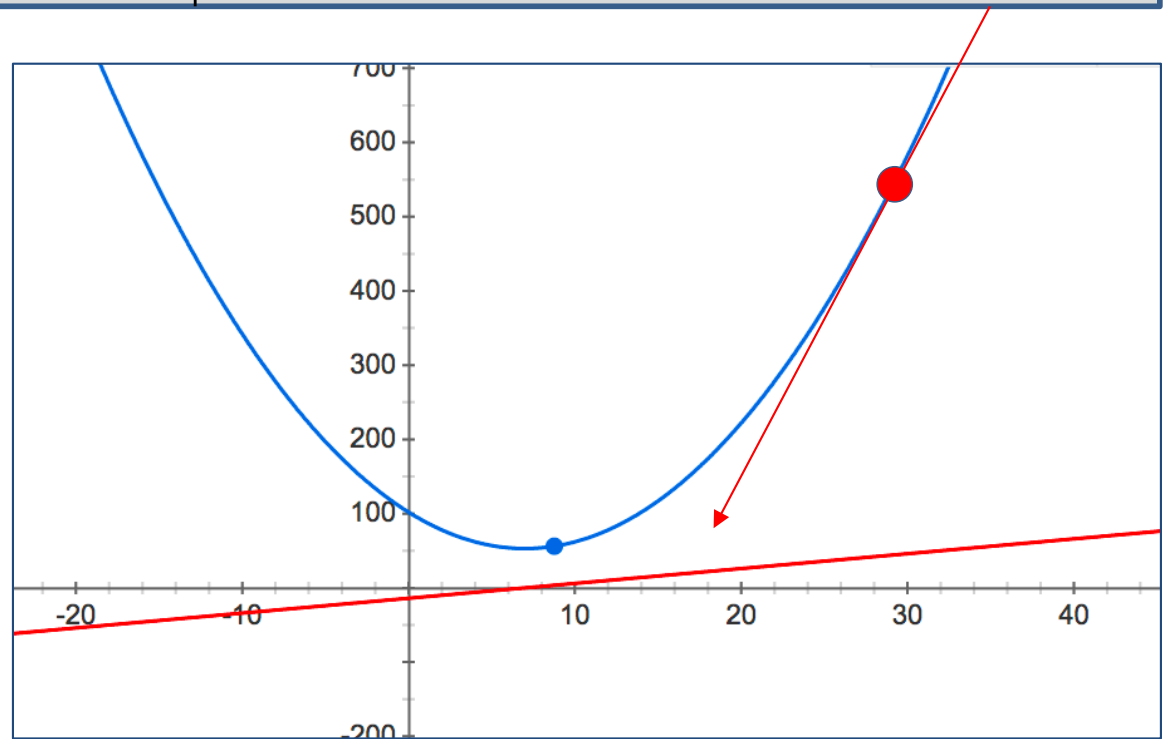
Solution:

Replace it with a constant.

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

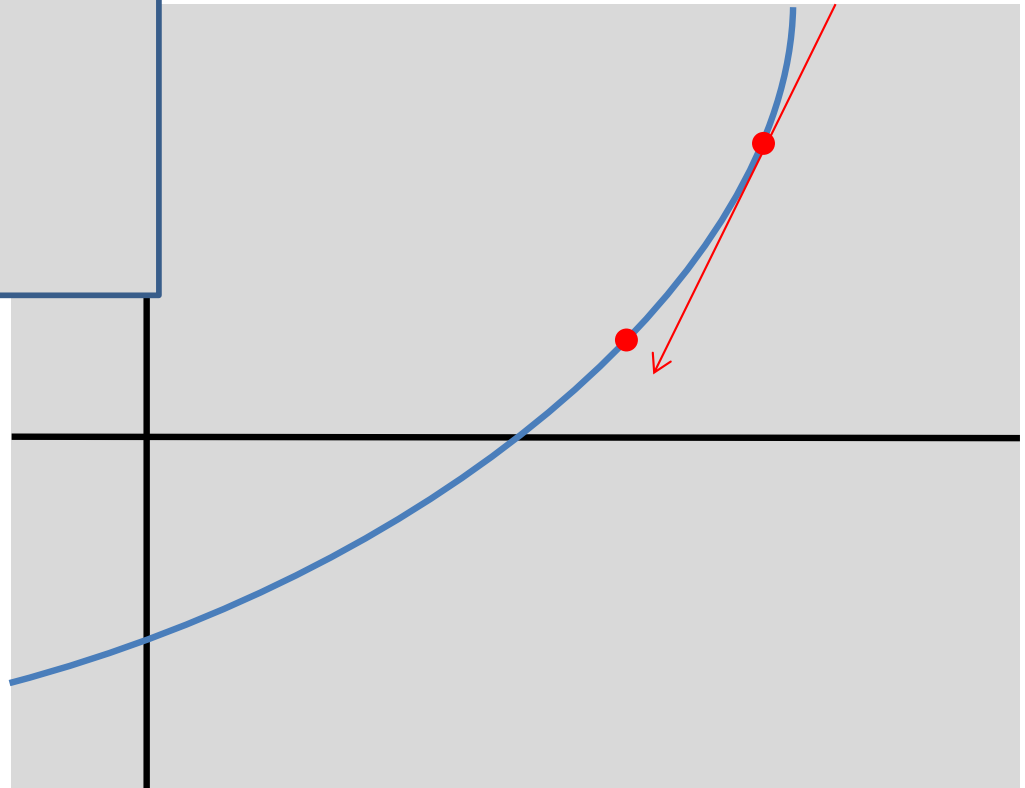


$$x_{i+1} = x_i - \gamma f'(x)$$



Gradient Descent

```
findMin(float x, float step, float error) {  
    diff = error+1;  
    while (diff > error){  
        float df = derivative(x);  
        float newX = x - (step*df);  
        diff = |x - newX|;  
        x = newX;  
    }  
    return x  
}
```



Gradient Descent

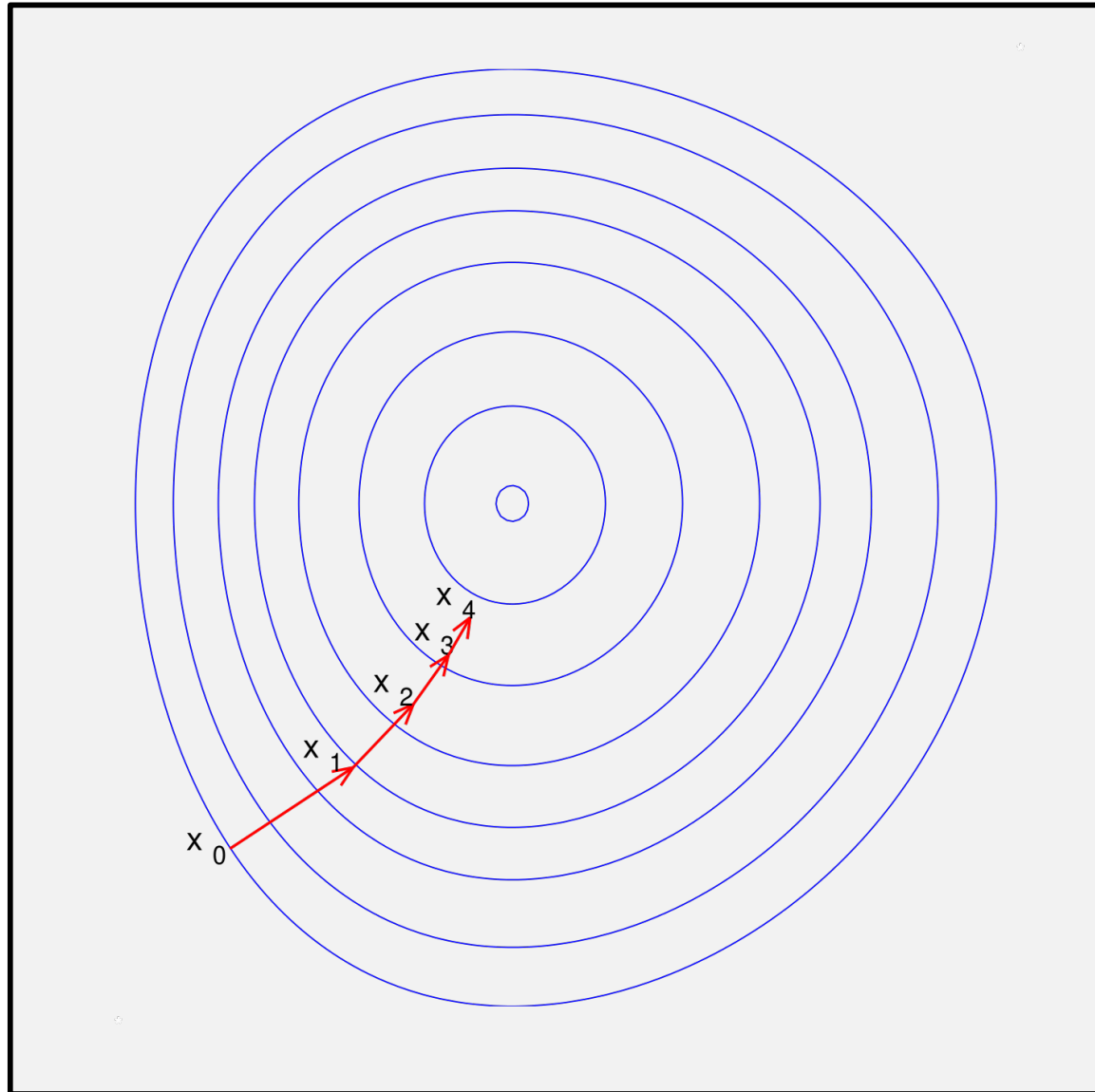


Image from:
Wikipedia

Gradient Descent

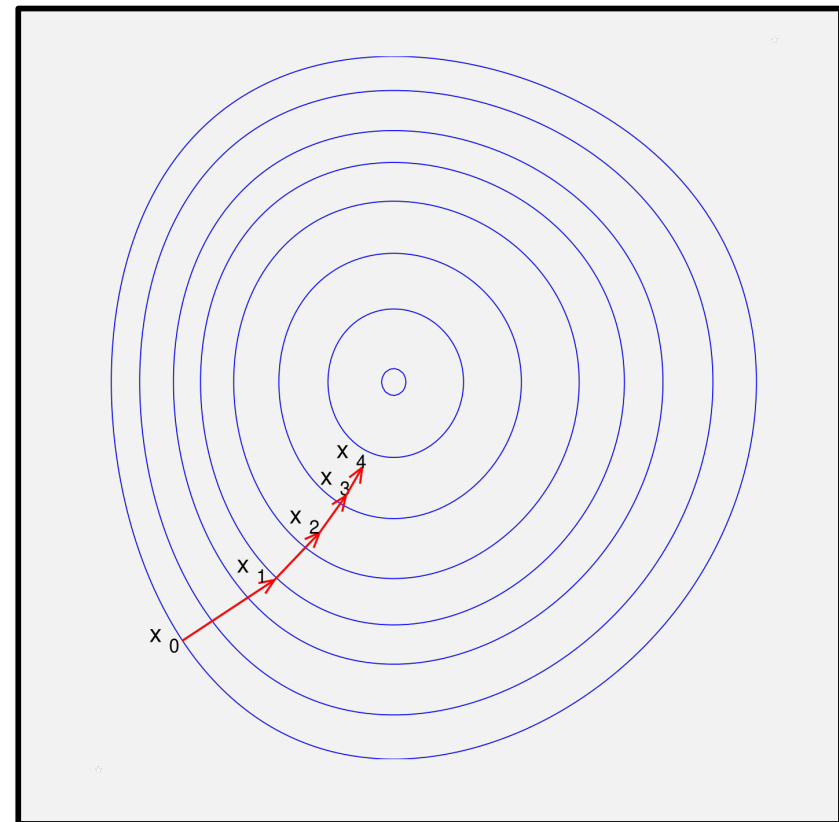
Goal: (in d dimensions)

Find the minimum value of $f(x)$ via successive approximation:

$$x_{i+1} = x_i - \gamma \nabla(f(x))$$

d-dimensional vectors

scalar



Gradient Descent

Goal: (in d dimensions)

Find the minimum value of $f(a,b,c)$ via successive approximation:

$$(a, b, c)_{i+1} = (a, b, c) - \gamma \nabla(f(a, b, c))$$

Notice:

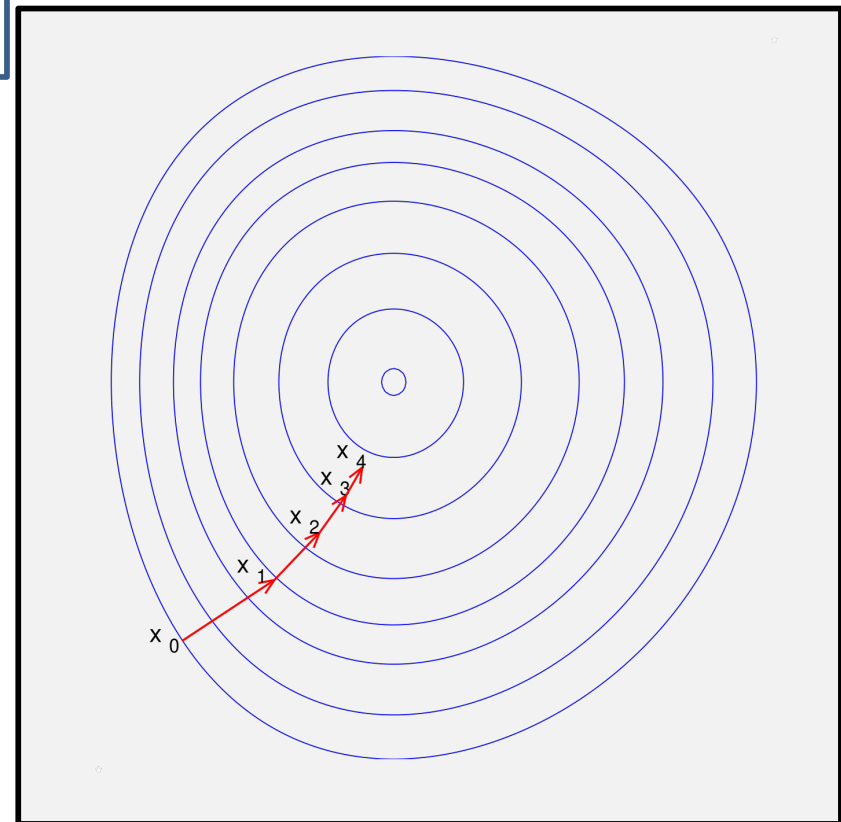
Each derivative computation may involve iterating over your entire data set!



Stochastic GD

Example:

$$\sum_{i=1}^n [z_i - h_{a,b,c}(x_i, y_i)] x_i$$
$$\sum_{i=1}^n [z_i - h_{a,b,c}(x_i, y_i)] y_i$$
$$\sum_{i=1}^n [z_i - h_{a,b,c}(x_i, y_i)]$$



Gradient Descent

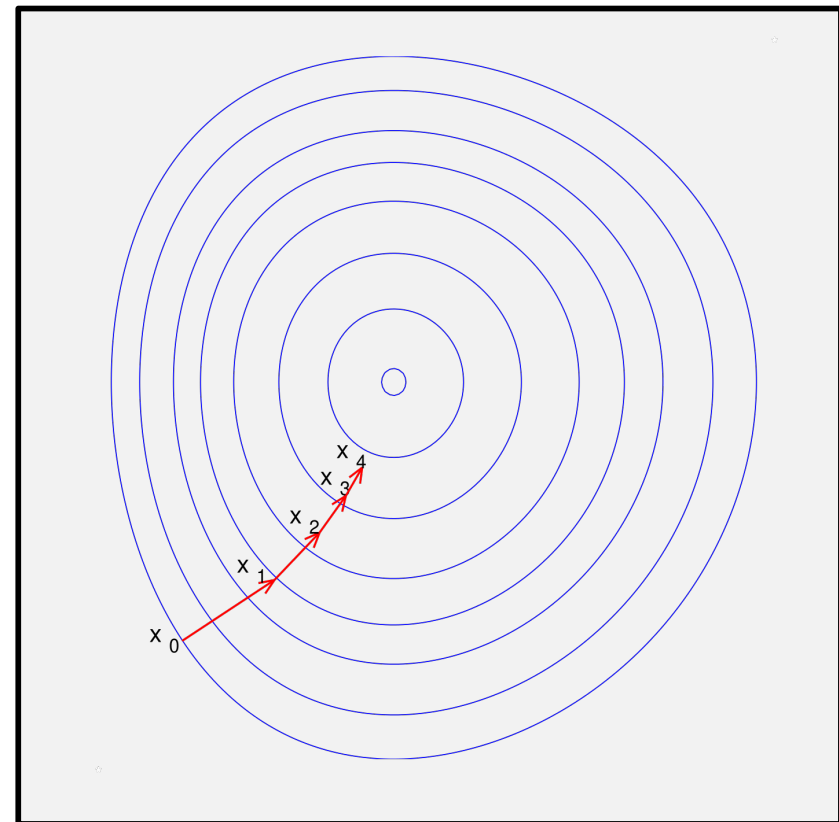
Goal: (in d dimensions)

Find the minimum value of $f(x)$ via successive approximation:

$$x_{i+1} = x_i - \gamma \nabla(f(x))$$

Cost per iteration:

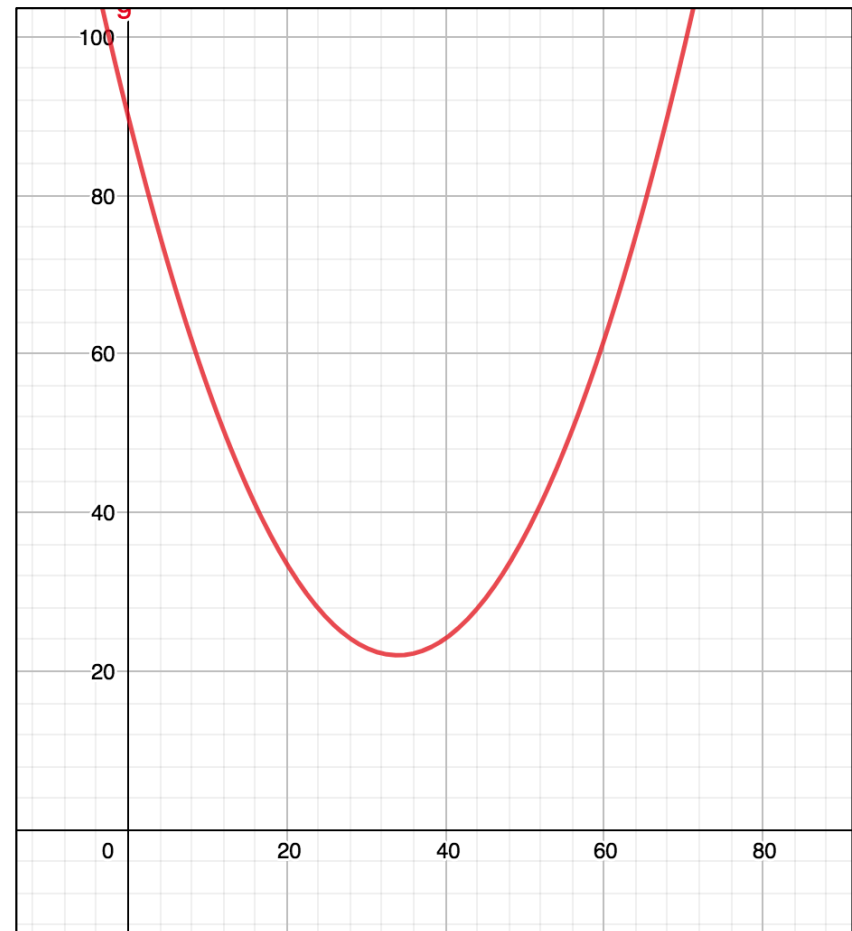
- d computations of partial derivatives
- d multiplications
- d subtractions



Gradient Descent

Convergence:

A function f is convex if...

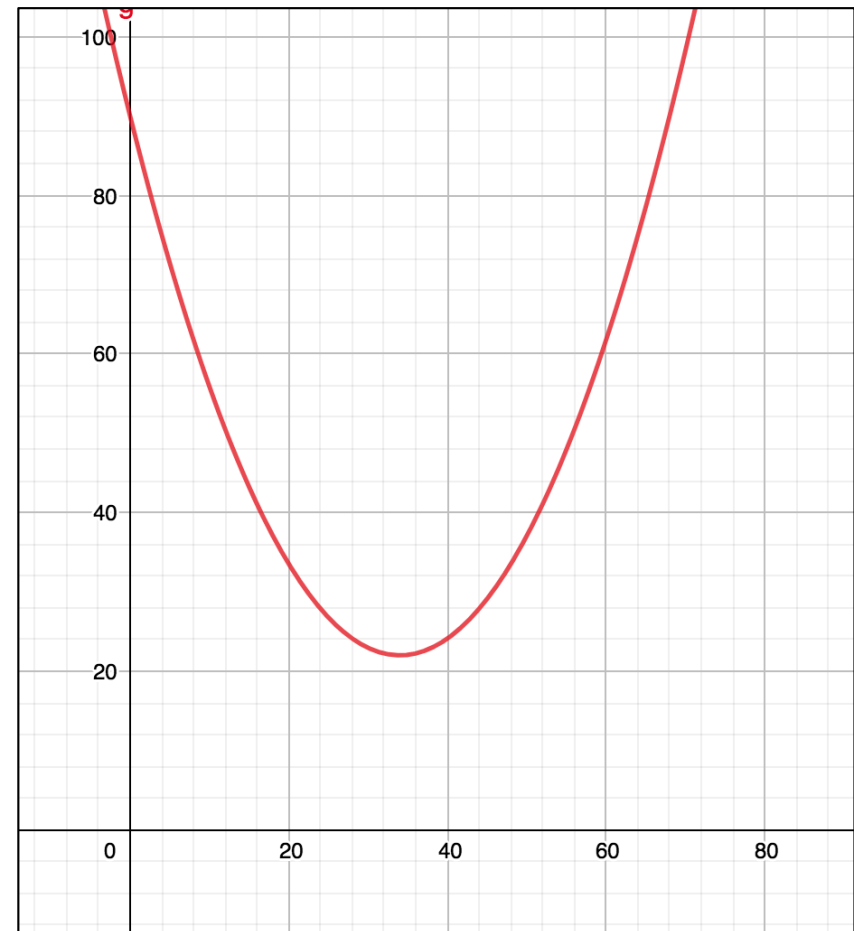


Gradient Descent

Convergence:

A function f is convex if...

→ has a minimum



Gradient Descent

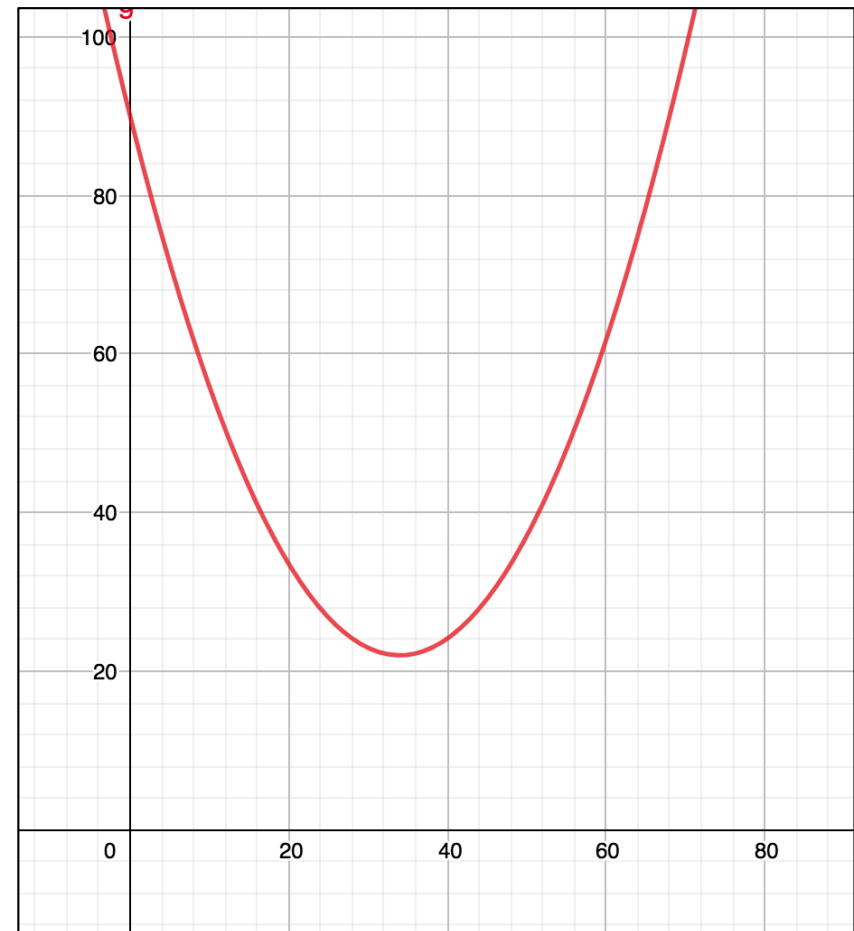
Convergence:

A function **f** is convex if...

A function **f** has a gradient that is **L**-Lipschitz if...

(gradient doesn't change too fast)

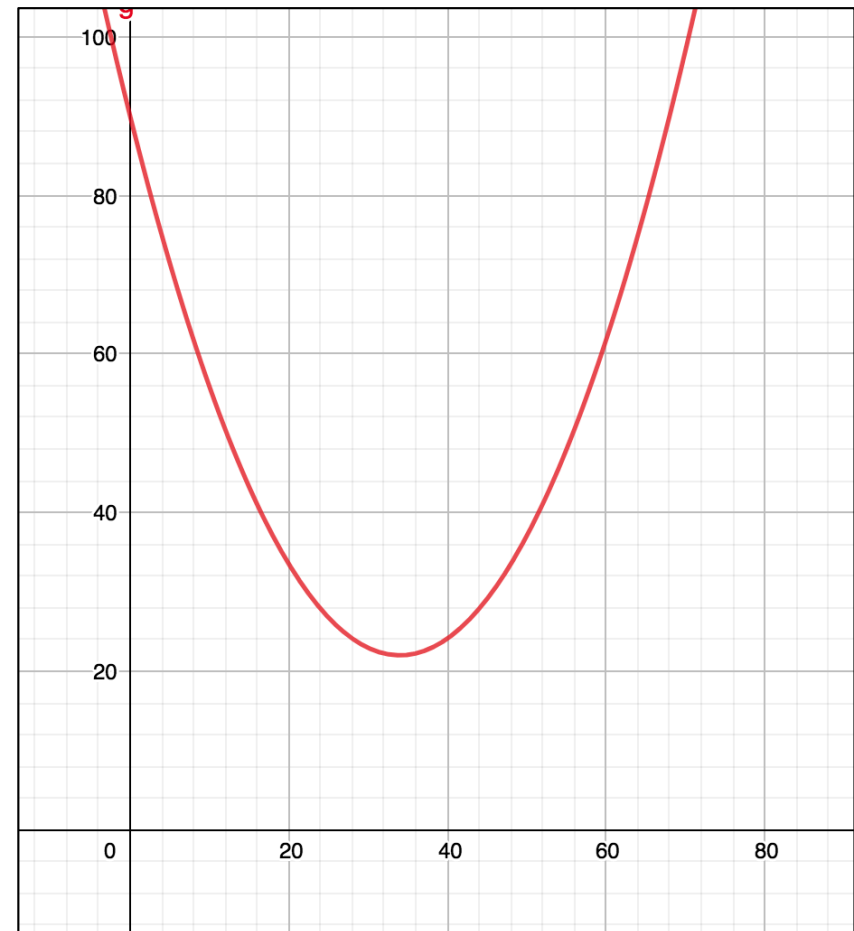
$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y||$$



Gradient Descent

Convergence:

If f is convex and differentiable
and its gradient is L -Lipschitz, and
if step size $\gamma \leq 1/L$:

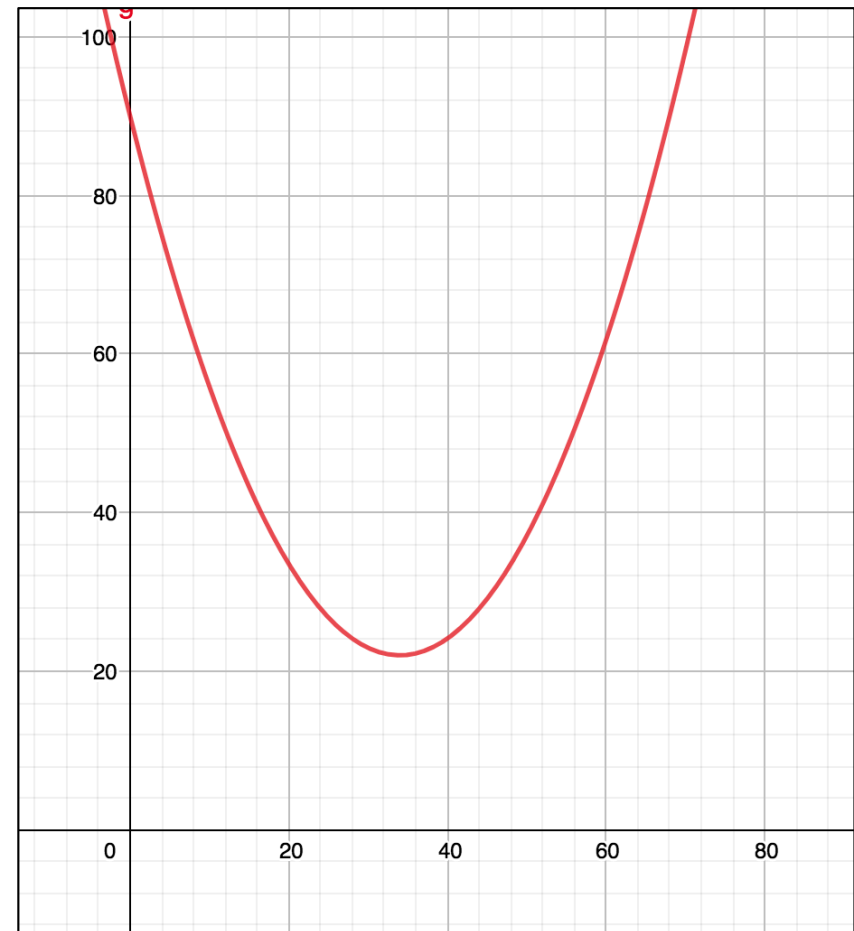


Gradient Descent

Convergence:

If f is convex and differentiable
and its gradient is L -Lipschitz, and
if step size $\gamma \leq 1/L$:

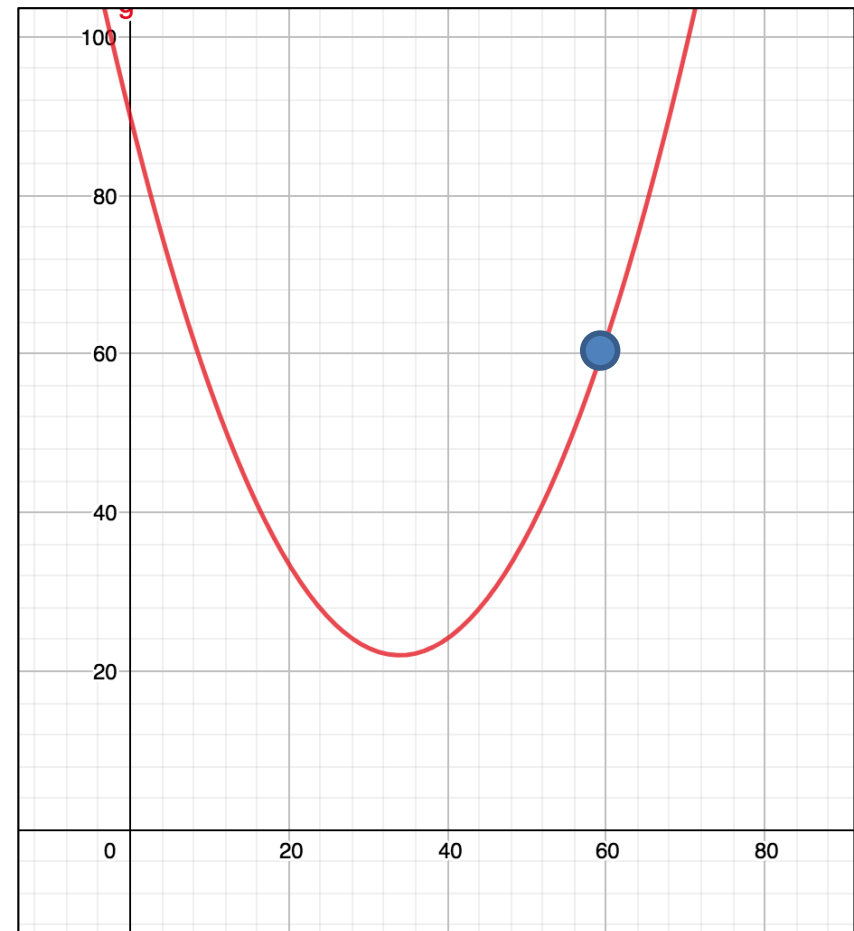
Gradient descent will converge!



Gradient Descent

Convergence:

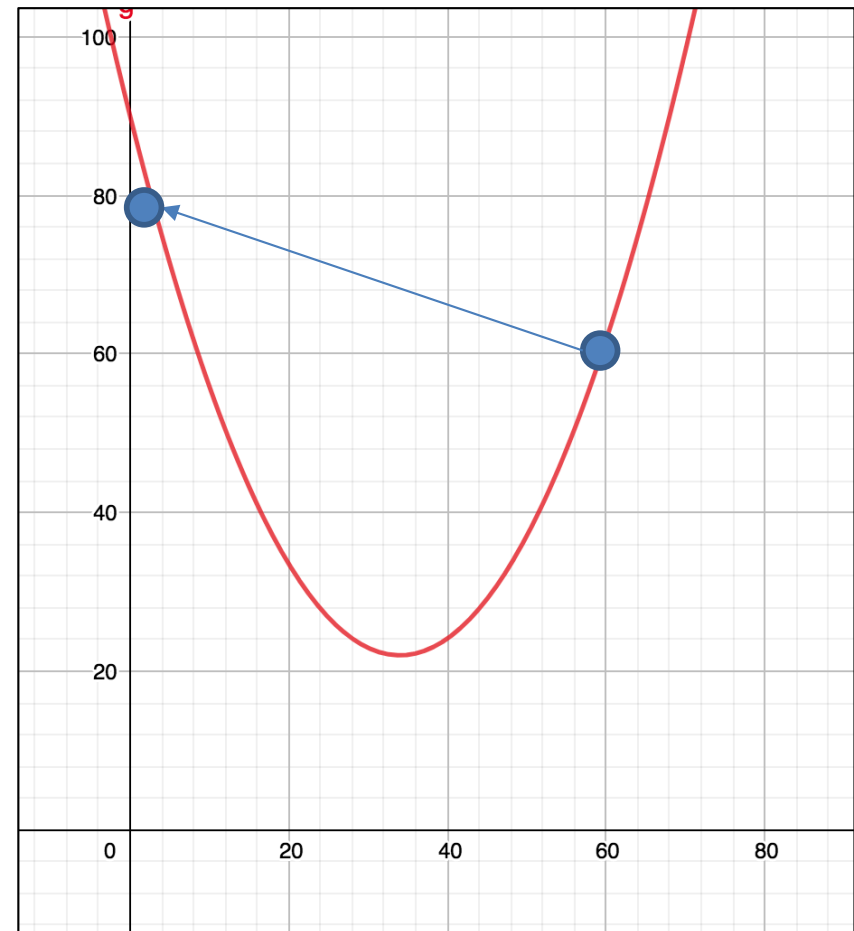
If step size is too large, may not converge:



Gradient Descent

Convergence:

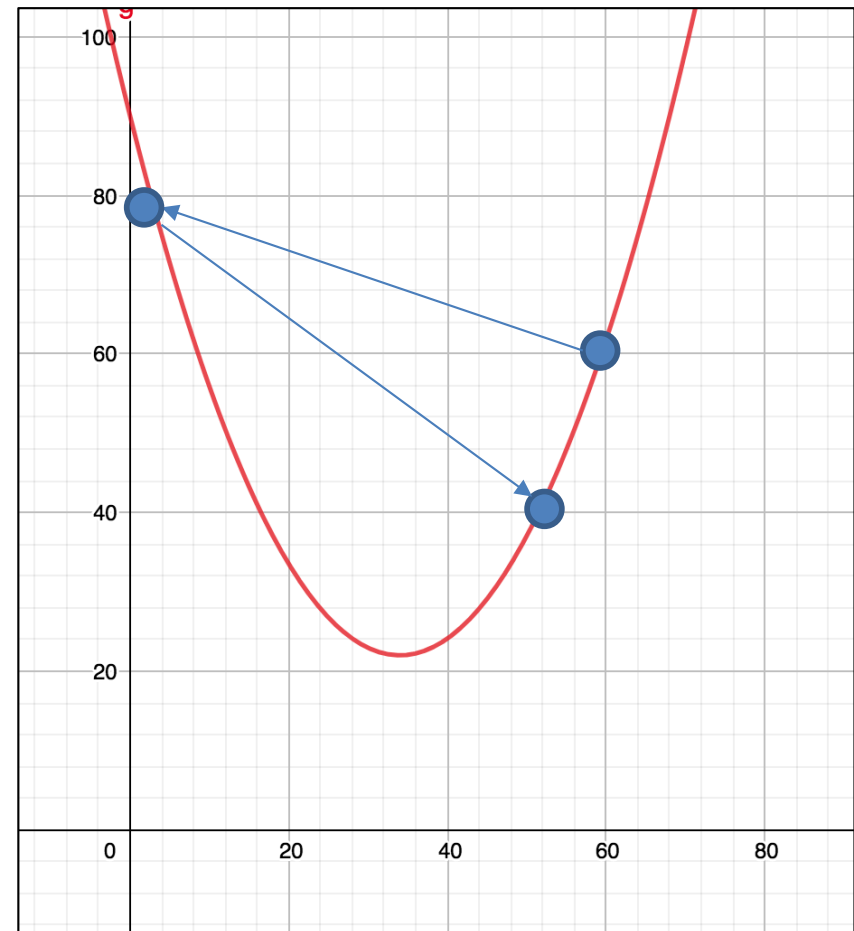
If step size is too large, may not converge:



Gradient Descent

Convergence:

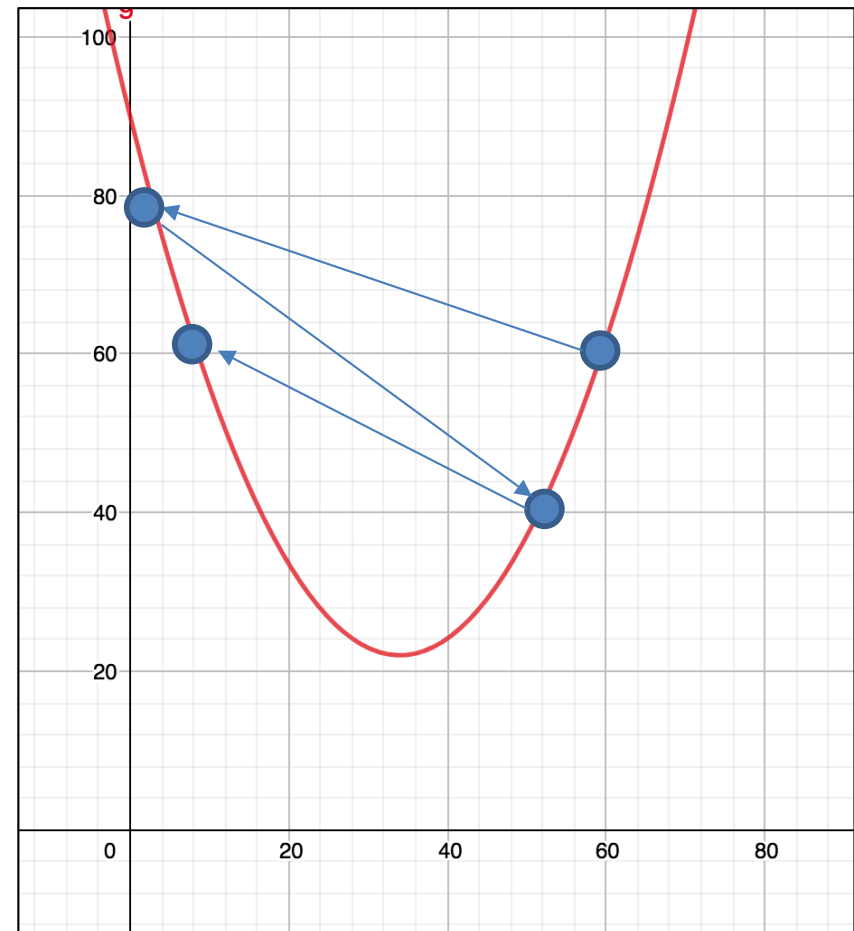
If step size is too large, may not converge:



Gradient Descent

Convergence:

If step size is too large, may not converge:

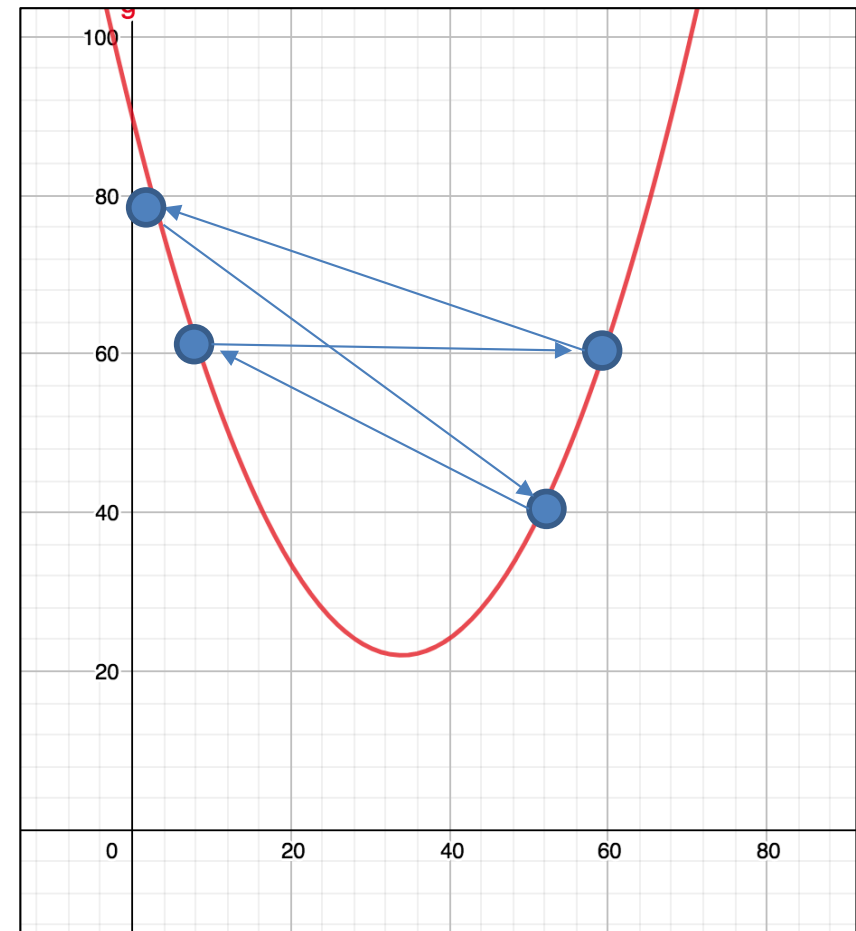


Gradient Descent

Convergence:

If step size is too large, may not converge:

(This is a fake example. Try it yourself!)



Gradient Descent

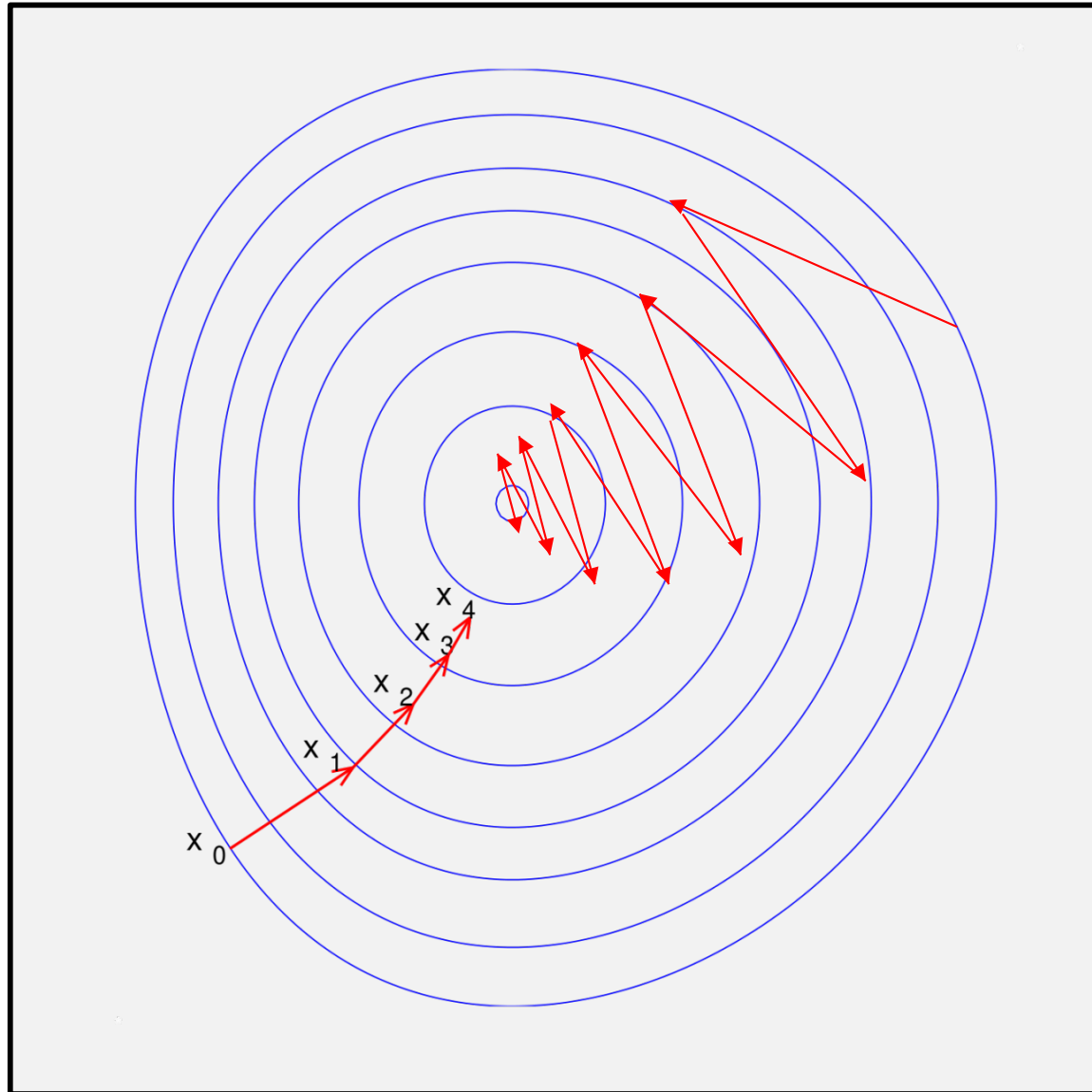


Image from:
Wikipedia

Gradient Descent

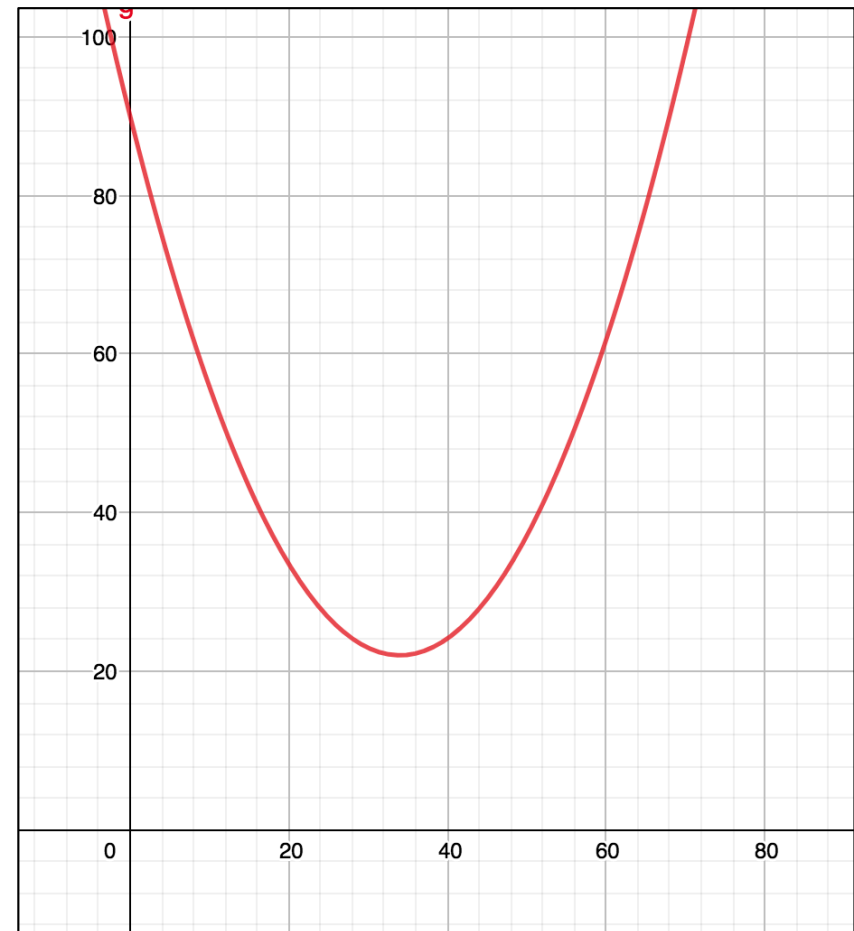
Convergence:

If f is convex and differentiable
and its gradient is L -Lipschitz, and
if step size $\gamma \leq 1/L$:

Gradient descent will converge!

After t iterations:

$$f(x_t) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\gamma t}$$



Gradient Descent

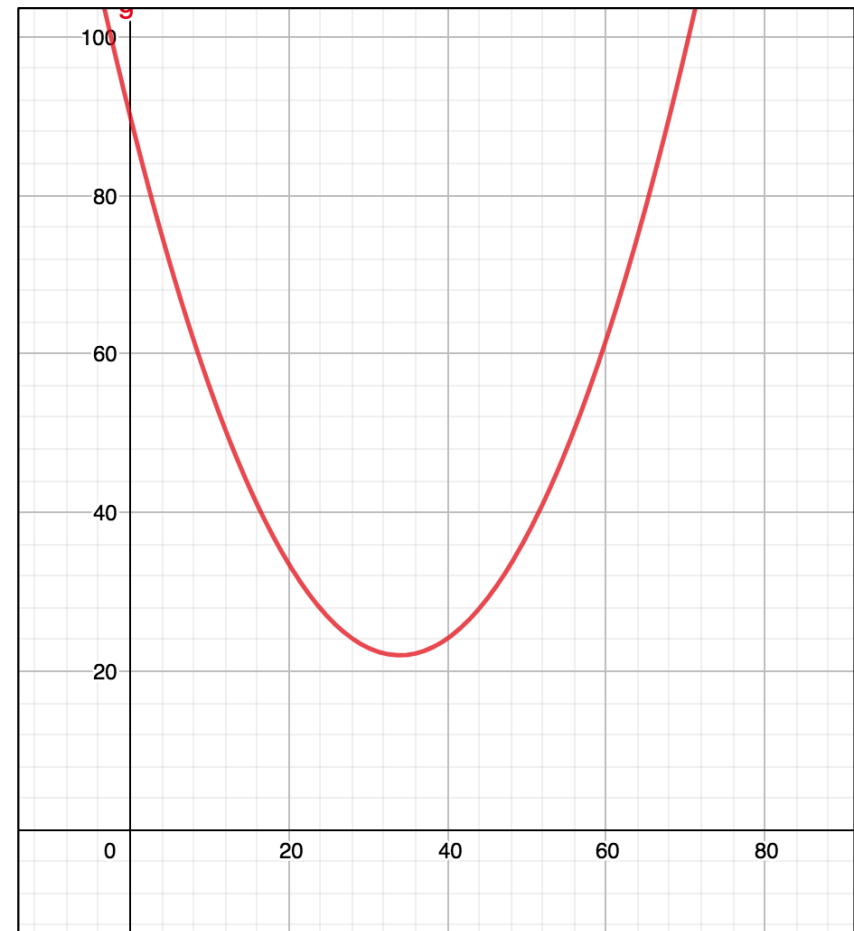
Convergence:

If f is convex and differentiable
and its gradient is L -Lipschitz, and
if step size $\gamma \leq 1/L$:

Gradient descent will converge!

To achieve error ϵ :

$$t \geq \frac{\|x_0 - x^*\|^2}{2\gamma\epsilon} \geq \frac{L\|x_0 - x^*\|^2}{2\epsilon}$$



Gradient Descent

Convergence:

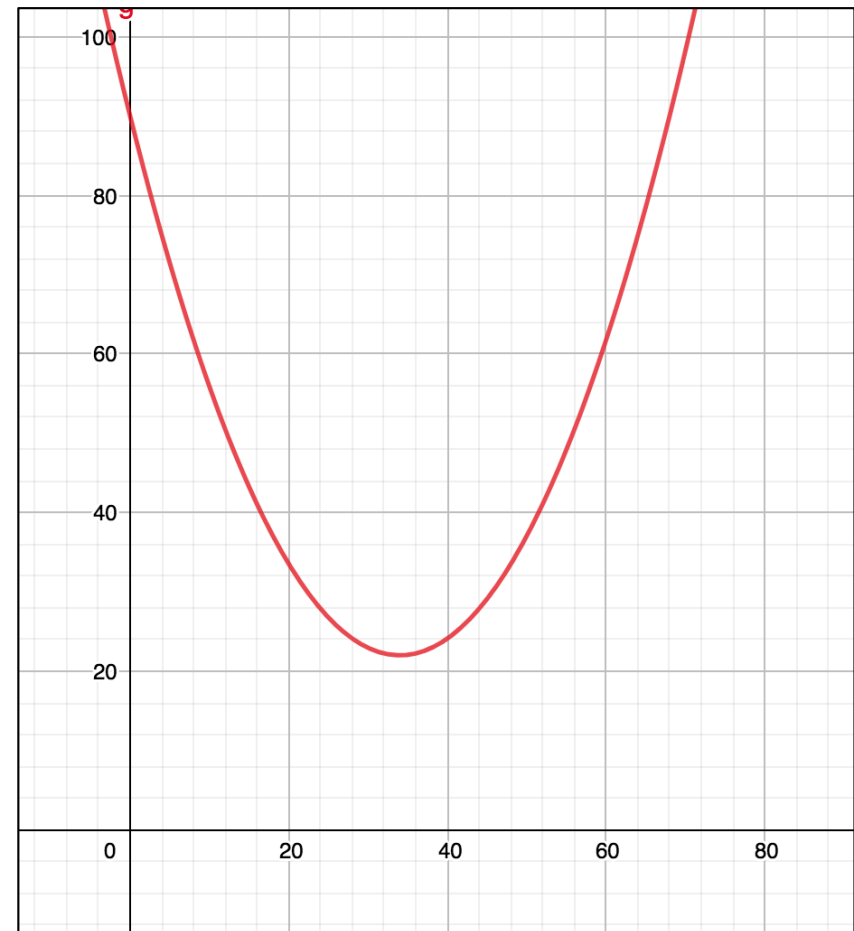
If f is convex and differentiable
and its gradient is L -Lipschitz, and
if step size $\gamma \leq 1/L$:

Gradient descent will converge!

To achieve error ϵ :

$$t \geq \frac{\|x_0 - x^*\|^2}{2\gamma\epsilon} \geq \frac{L\|x_0 - x^*\|^2}{2\epsilon}$$

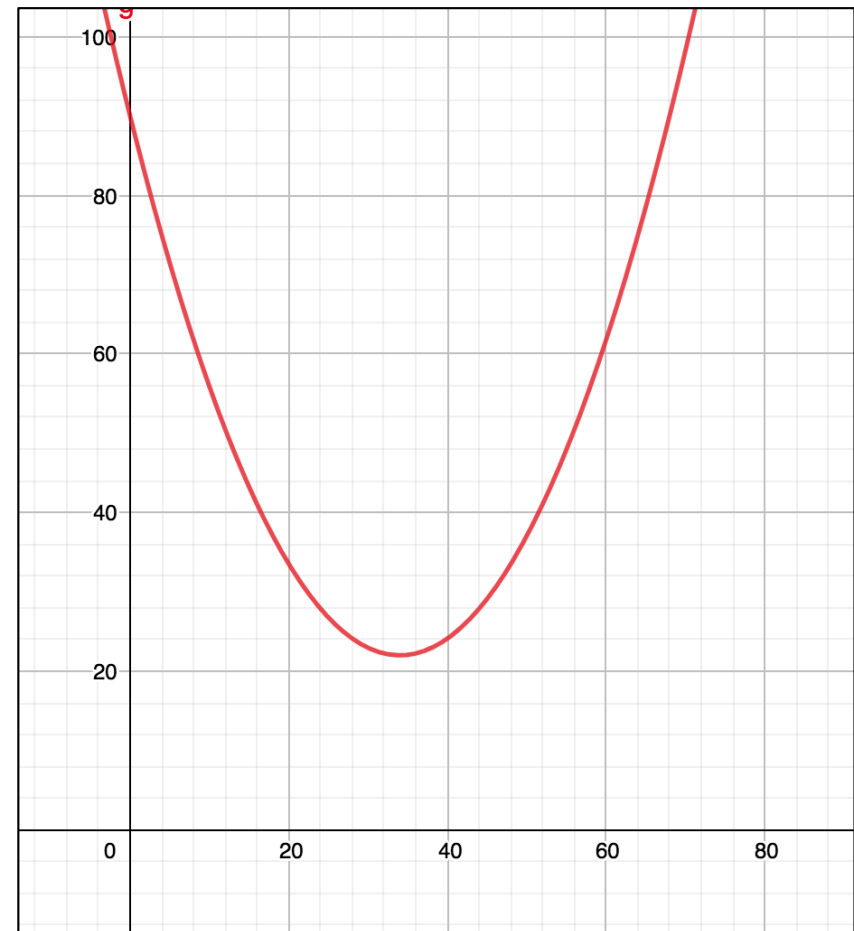
More iterations than Newton's method...
... but faster computation per iteration.



Gradient Descent

Convergence:

How to choose step size γ ?

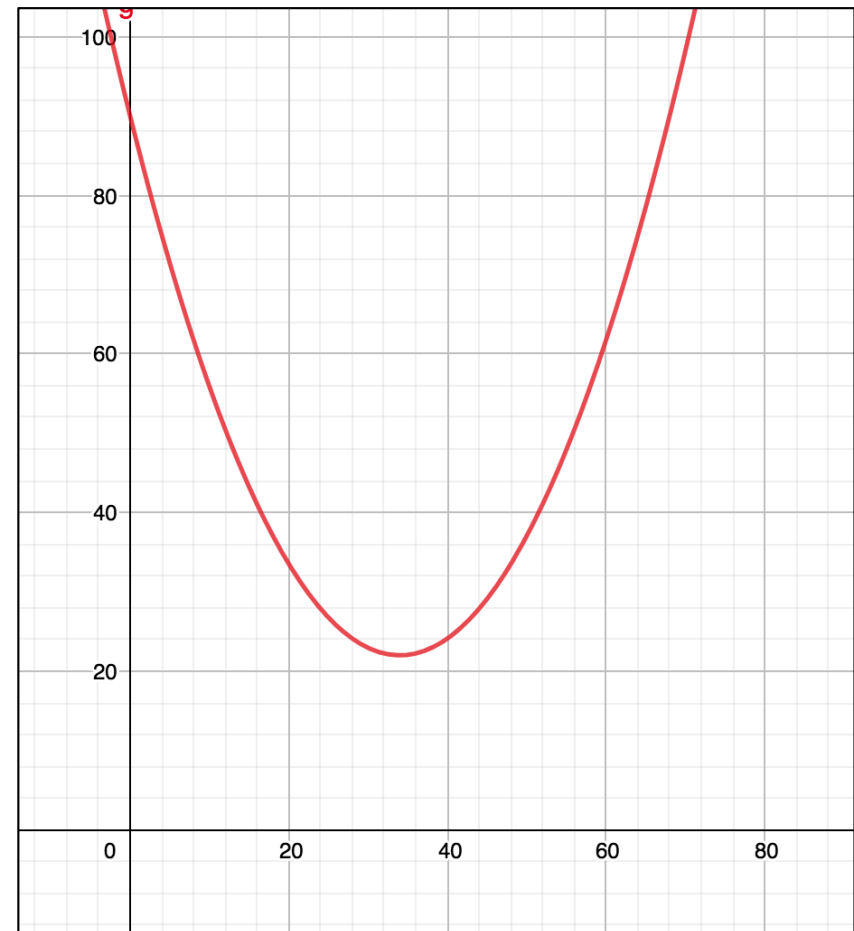


Gradient Descent

Convergence:

How to choose step size γ ?

1. Look at function and compute L.



Gradient Descent

Convergence:

How to choose step size γ ?

2. Adjust step size γ in each step:

γ = very small value

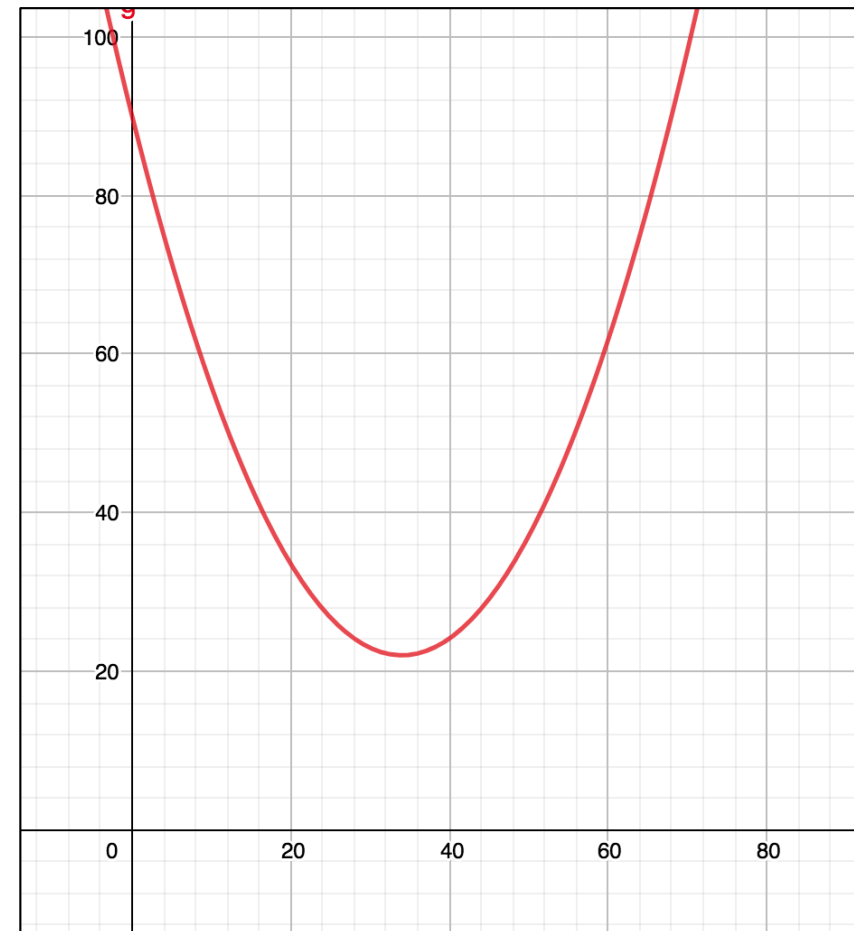
repeat:

Run t iterations of $GD(\gamma)$.

If converged, stop.

Else $\gamma = \gamma/2$

Kind of like a binary
search on γ !



Gradient Descent

Convergence:

How to choose step size γ ?

2. Adjust step size γ in each step:

γ = big value

repeat:

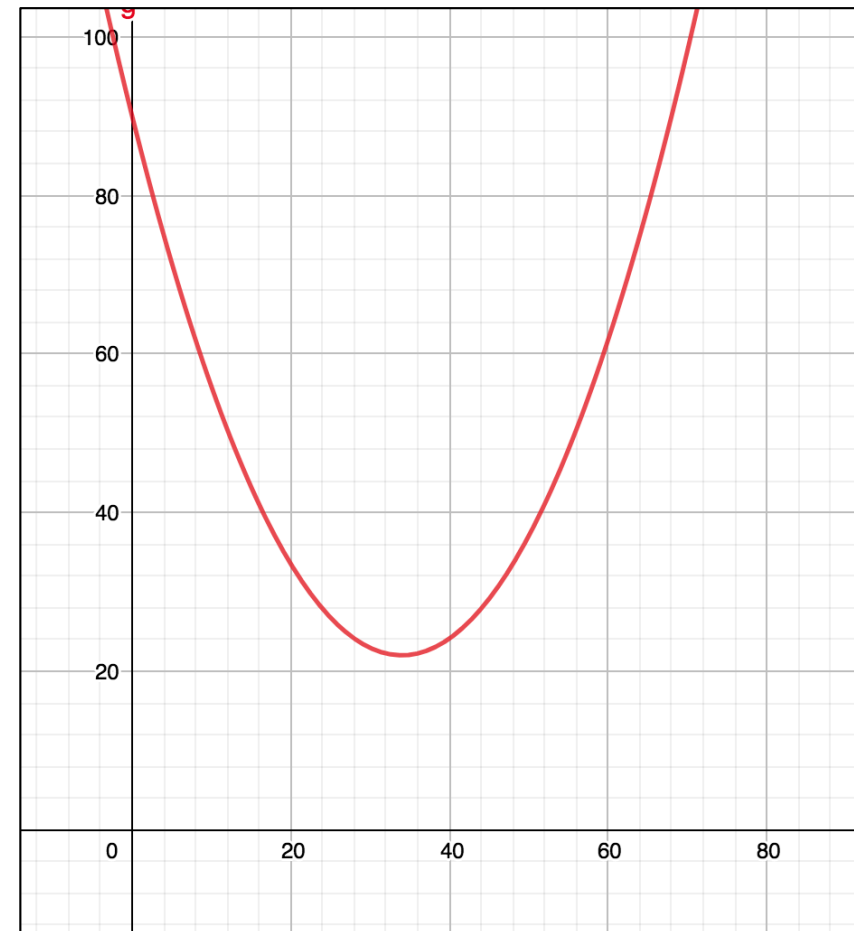
Run t iterations of $GD(\gamma)$.

If converged, stop.

Else $\gamma = \gamma/2$

What is a good value to start with?

How many iterations is enough?



Gradient Descent

Convergence:

How to choose step size γ ?

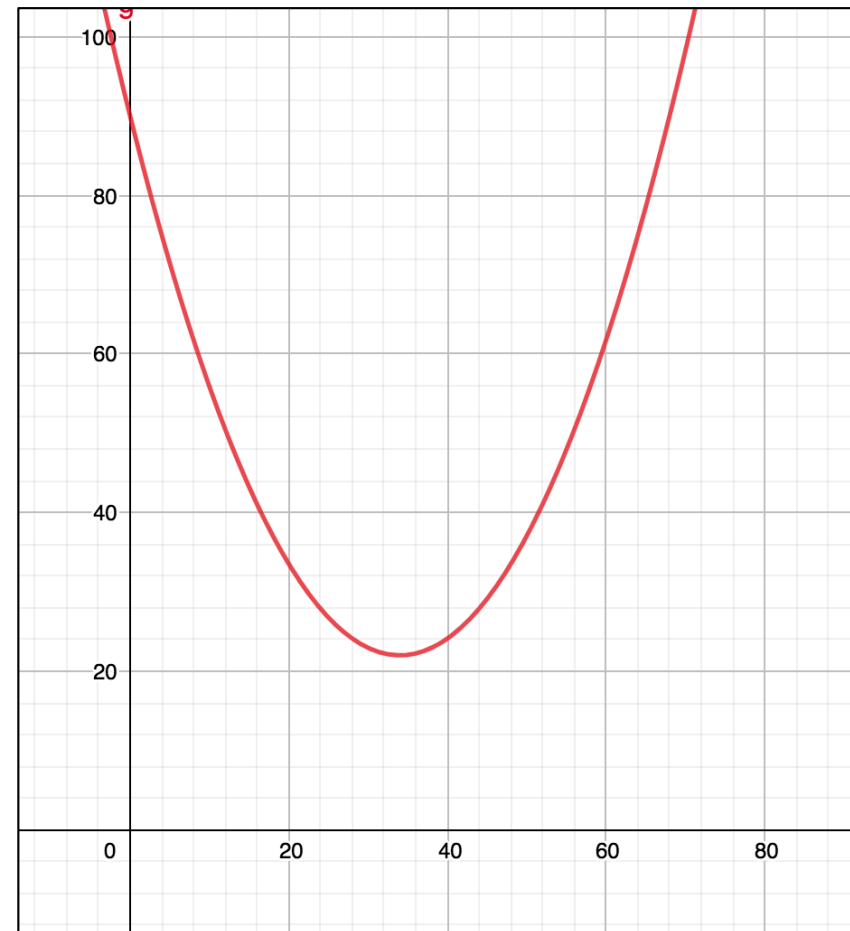
3. Compute an optimal step size for each iteration (*line search*):

$$\min_{\gamma} [f(x_i - \gamma \nabla f(x_i))]$$

For (d=1), same as computing answer.

For (d > 1), it reduces to a one-dimensional minimization problem.

→ line search



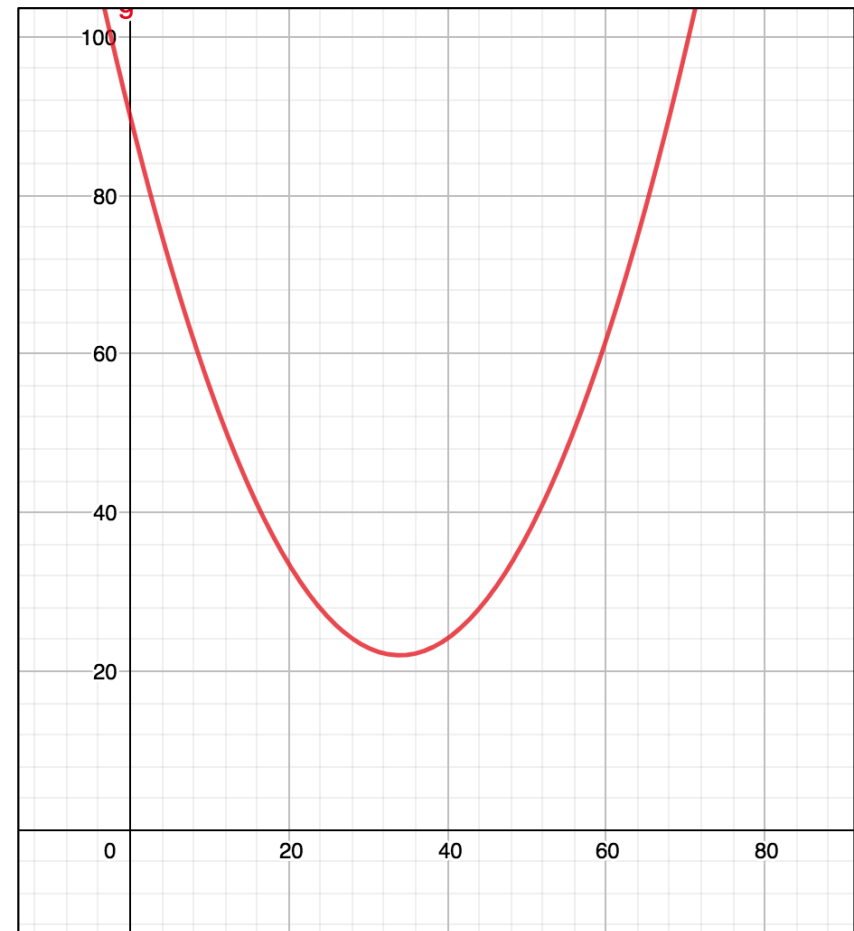
Gradient Descent

Convergence:

How to choose step size γ ?

- 4. Backtracking
- 5. Other
- 6. Other
- 7. Other
- 8. Other
- 9. Compute the inverse Hessians

Much of the cleverness in implementing gradient descent lies in adjusting the step-size in a clever way.



Gradient Descent

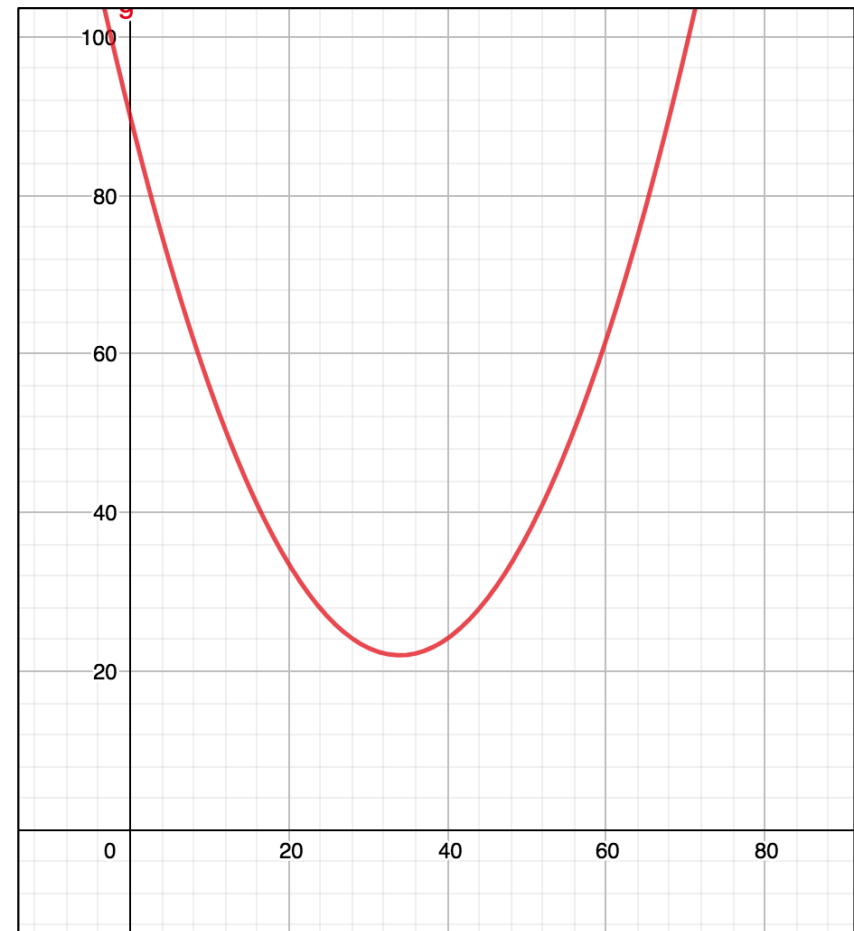
Convergence:

If function is strongly convex,
then convergence is much faster...

“Linear” \rightarrow rate = $\log 1/\epsilon$

Also see:

Accelerated gradient descent
(Nesterov’s method), and many,
many more variations

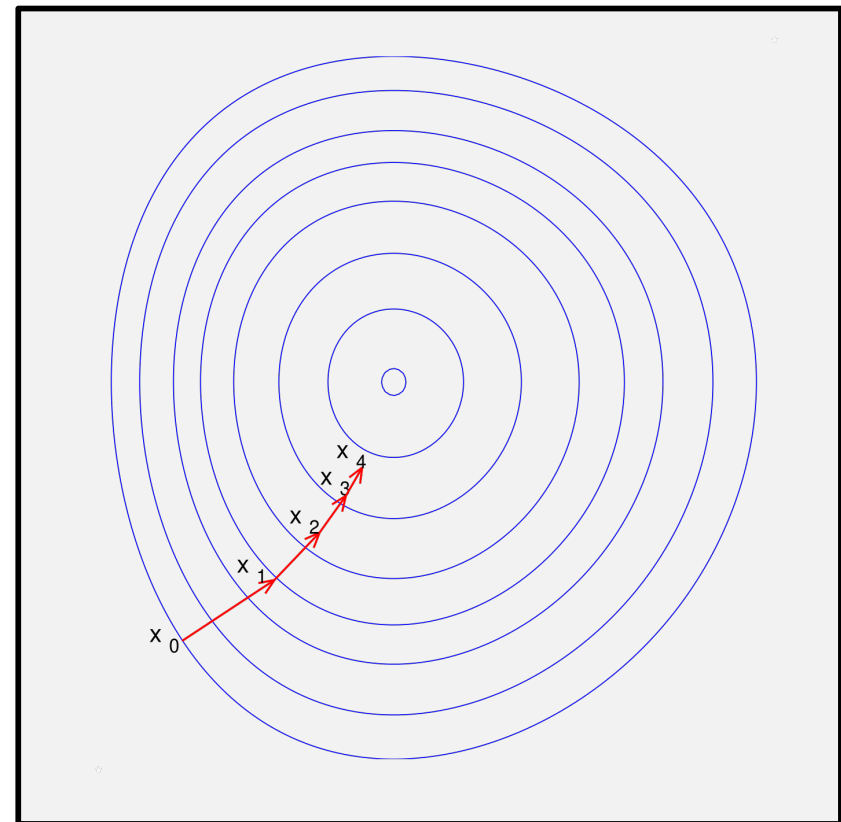


Gradient Descent

Goal: (in d dimensions)

Find the minimum value of $f(x)$ via successive approximation:

$$x_{i+1} = x_i - \gamma_i \nabla(f(x))$$



Trade-Offs

Newton's Method

Fewer iterations

- *Convergence is quadratic*

More expensive computation

- *Requires inverse Hessian*

Requires access to second derivative

Gradient Descent

More iterations

- *Convergence is linear (when the function is strongly convex).*

Cheap computation

- *Just the gradient*

Requires choices of step size

- *Backtracking*
- *Line search*
- *Etc.*

Summary

Today: Optimization

Finding a maximum / minimum

- *Key aspect of machine learning*
- *Binary search isn't always good*

Newton's method

- *Second order approach*

Gradient descent

- *First order approach*

Next time:

Sorting

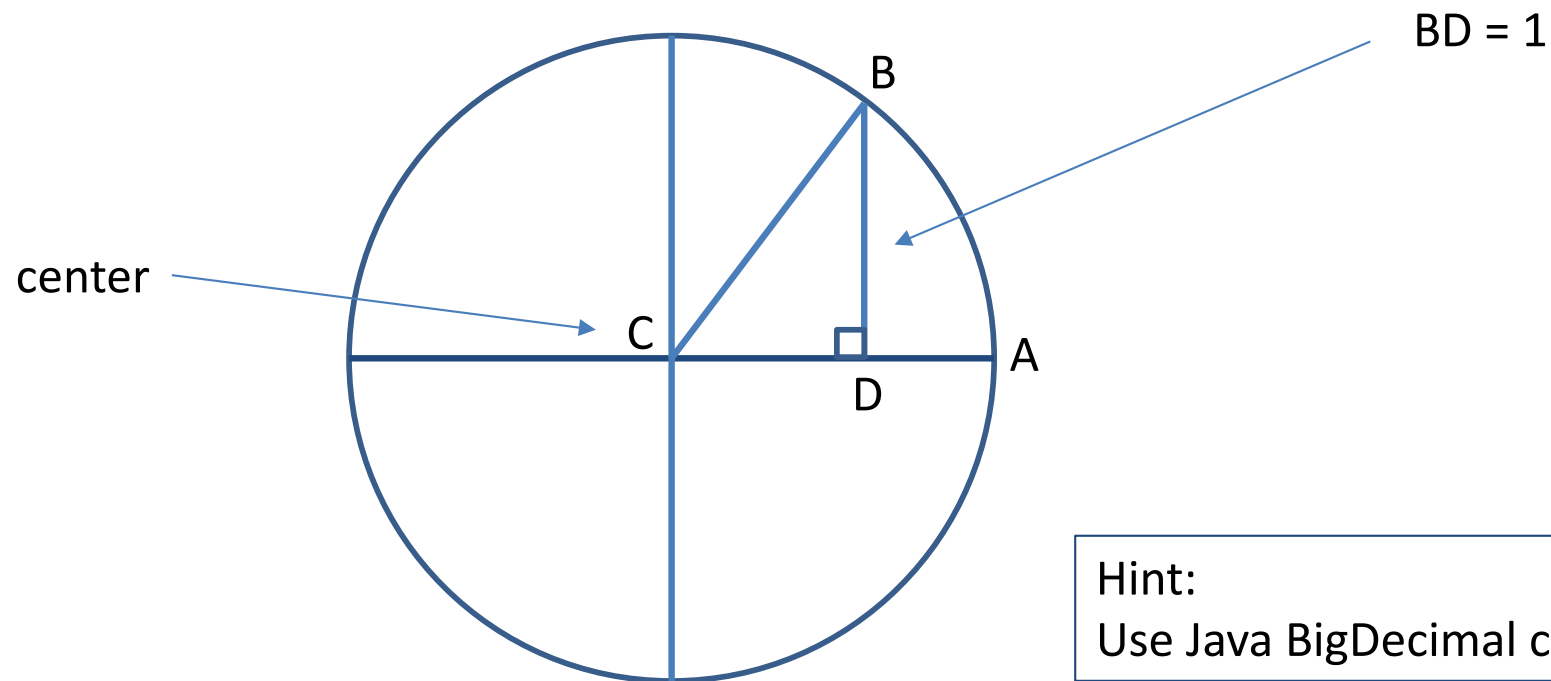
- *BubbleSort*
- *InsertionSort*
- *SelectionSort*
- *MergeSort*
- *QuickSort*

Properties of sorting algorithms

- *Stability*
- *Input-sensitive performance*
- *Etc.*

A Geometry Challenge

Compute AD (using, e.g., Newton's method, to 500 digits)



diameter = 1,000,000,000,000

What pattern do you see encoded in the result?