Lecture #2: Introduction to Optimisation and Constraint-Satisfaction Problems

COMP3608 - Intelligent Systems
Inzamam Rahaman

Objectives of this lecture

- Many AI problems are either framed as optimisation or constraint satisfaction or use such techniques

- We need to understand these techniques and the underlying concepts before we proceed

Outline

1. What is Optimization?
    1. Important Definitions
2. Categories of optimisation
    1. Unconstrained vs constrained
    2. Continuous vs Discrete
3. Optimisation process
4. Problem conversion and relaxation
5. Gradient Descent
6. Backtracking

What is Optimisation?

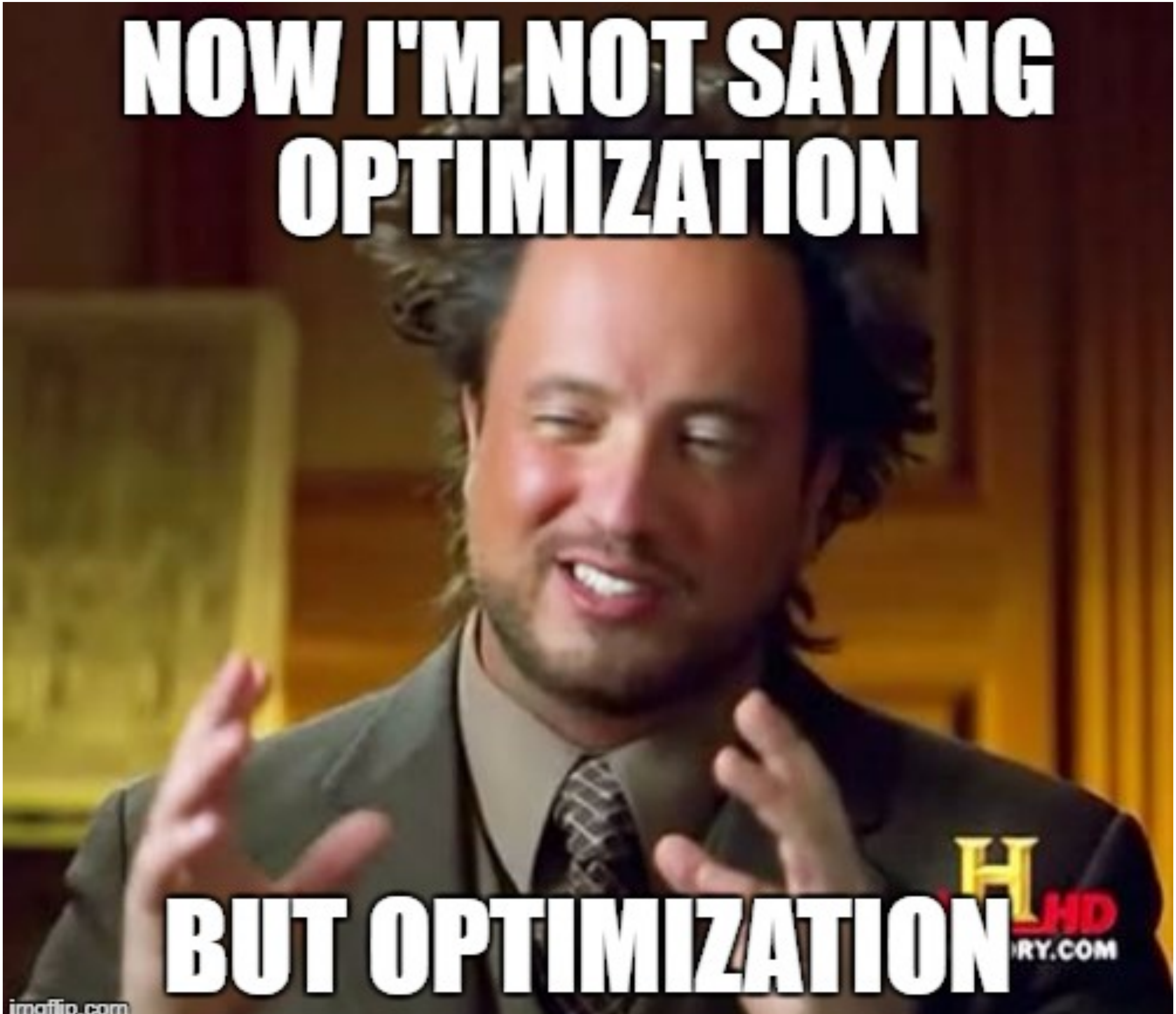
- We are constantly confronted by optimisation problems
  - Given a road network, how do I get from my start point to intended end point the quickest or cheapest?
  - How many hours should I devote to studying different courses to maximise my semester GPA?
  - How should different airline crews be scheduled to maximise profits while simultaneously minimising crew fatigue and staying in accord with labour laws and transportation regulations?

What is Optimisation?

- We are constantly confronted by optimisation problems
  - Given a road network, how do I get from my start point to intended end point the **quickest** or **cheapest**?
  - How many hours should I devote to studying different courses to **maximise** my semester GPA?
  - How should different airline crews be scheduled to **maximise** profits while simultaneously **minimising** crew fatigue and staying in accord with labour laws and transportation regulations?
- In the above, we want to make decisions that maximise or minimise some performance measure


ONE DOES NOT SIMPLY
MAKE DECISIONS WITHOUT OPTIMIZATION
quickmeme.c

Optimisation is a fundamental activity that "intelligent" systems must perform


NOW I'M NOT SAYING
OPTIMIZATION
BUT OPTIMIZATION

What is Optimisation?

- In the field of optimisation, we are concerned with getting through to the heart of a problem to get an idea about the performance measure to minimise or maximise and what decisions are available to us

- We then formulate this measure mathematically as a function or as a set of functions

- And characterise the limitations on our decisions

What is Optimisation?

$$\min f(x)$$

subject to

$$x \in \mathcal{X}$$

or

$$\max f(x)$$

subject to

$$x \in \mathcal{X}$$

What is Optimisation?

$$\min f(x)$$

subject to

$$x \in \mathcal{X}$$

Cost/Loss Function

Feasible Set or Region

Decision (Design) Variable

From a feasible set, we want to choose a design variable that minimises (or maximises) a function

What is Optimisation?

Objective Function

$$\max f(x)$$

subject to

$$x \in \mathcal{X}$$

Feasible Set or Region

Decision (Design) Variable

From a feasible set, we want to choose a design variable that minimises (or maximises) a function

Objective function is sometimes also called profit, reward, or utility function

A note on Objective functions

- We can minimise or maximise any function whose domain is a set with a partial ordering defined on it
- But, in real cases, we tend to only have functions with a scalar real number output
- Will assume that this is the range of our objective function from here on out

Definitions

- Suppose $x^* \in \mathscr{X}$ is the design point where the minimum (maximum) value of $f(x)$ occurs. We say that x^* is a minimiser (maximiser) of $f(x)$.

- Moreover, we can say

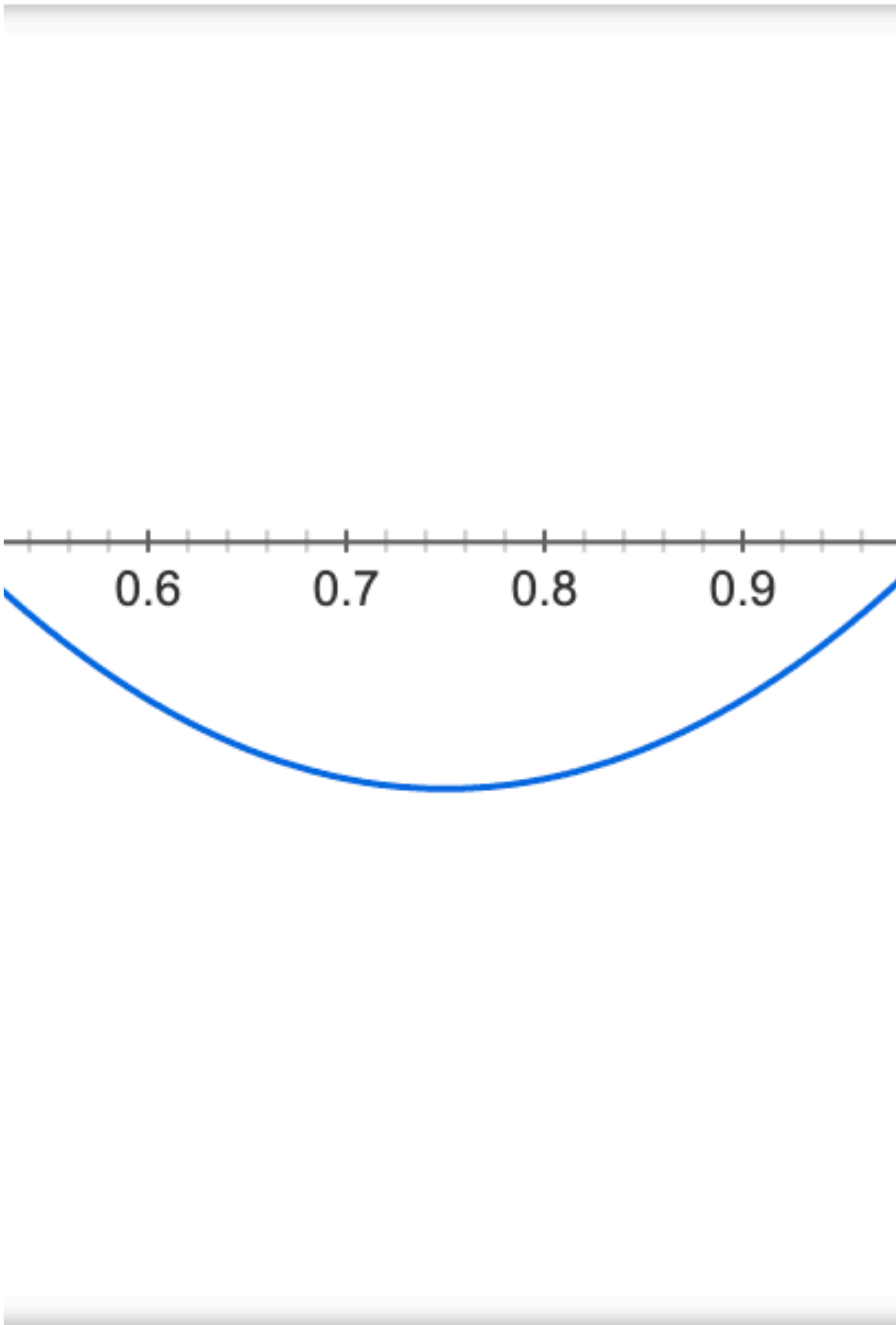
- $x^* = \underset{x \in \mathscr{X}}{\operatorname{argmin}} f(x)$ or $x^* = \underset{x \in \mathscr{X}}{\operatorname{argmax}} f(x)$

- x^* is the solution to optimisation problem

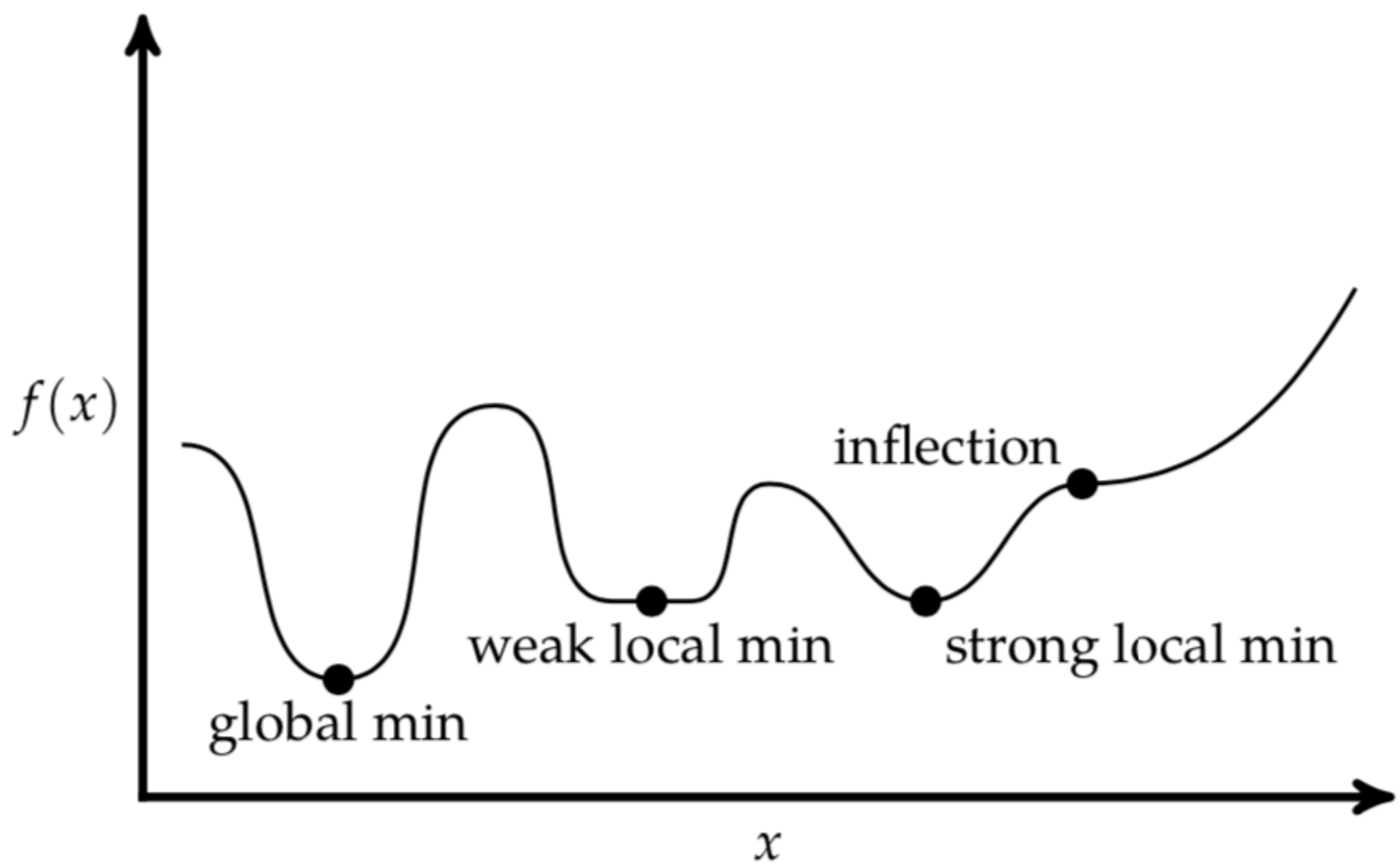
Example

- Consider
  $\min 2x^2 - 3x + 1$
  subject to
  $x \in \mathbb{R}$

- What is x^* and $f(x^*)$?

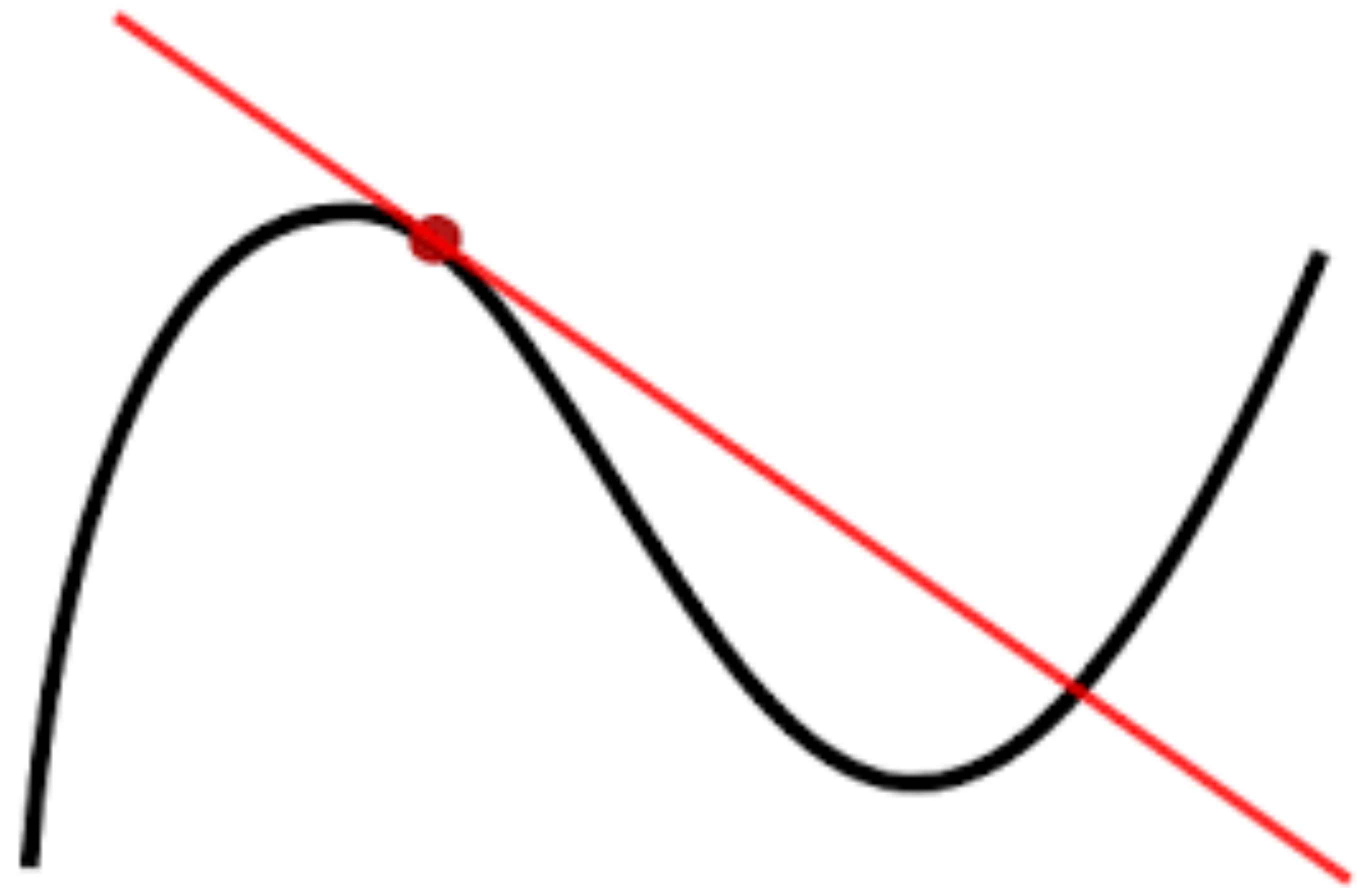


Definitions



Gradient

- Note that the derivative gives the gradient or slope of the tangent to the curve
- This means, that at an optimum, local or global, the derivative is 0
- Hence, we can use the derivative to solve for the optimum point
    - Can use the second order derivative to determine if maximum or minimum



Gradient

- Easy right!
- Note that we can't use this in every case
  - Some functions are not continuously differentiable over the feasible set :(
  - Gradient has no analytical solution :(
  - Gradient is expensive to compute :((Automatic differentiation can help us with this to some extent though)
- Will encounter many such cases in AI
  - Will learn how to cope
    - Gradient Descent (1st order method) - no analytical solution
    - Metaheuristics (0th order methods) - expensive or non-existent gradient
  - 2nd order methods (Newton-Raphson, qausi-Newton methods) also exist, but are expensive and not used that much in AI (yet)

Optimisation = search

- A common theme in my optimisation algorithms is that they are essentially searching for the optimal design point
- This is important to keep in mind when we look at meta-heuristics next class!

Optimisation Problem Conversion

- Some algorithms are designed to solve minimisation problems, other to solve maximisation
- We can “convert” between the two formulations by negating the objective or cost function
- The solutions of the original and the derived problem are the same!
  - Imagine reflecting a graph on the y axis
  - $$\operatorname*{argmin}_{x \in \mathcal{X}} f(x) = x^* = \operatorname*{argmax}_{x \in \mathcal{X}} \; - f(x)$$

Optimisation Problem Conversion

- Same thing applies if we add a constant, we shift the function up or down, but the solution point remains the same

- $$\underset{x \in \mathcal{X}}{\operatorname{argmin}} f(x) = x^* = \underset{x \in \mathcal{X}}{\operatorname{argmin}} f(x) + c, c \in \mathbb{R}$$

Classes of optimisation problem

- There are many ways to classify optimisation problems, e.g. convex opimtization, linear, integer, mixed-integer, etc…
- We shall focus on differentiating optimisation problems across two dimensions:
    - Unconstrained vs constrained
    - Continuous vs Discrete (Combinatorical)

Unconstrained vs Constrained

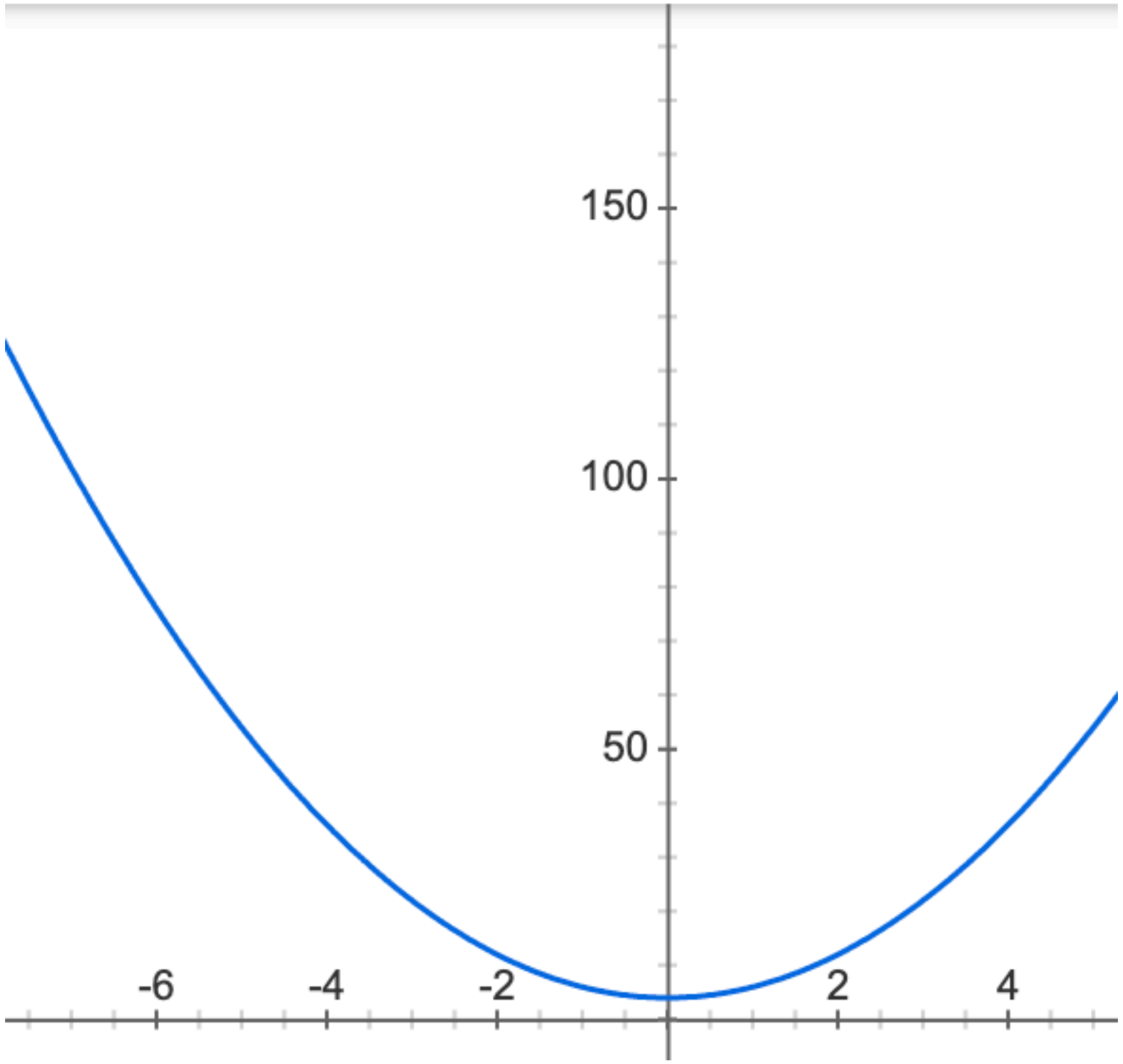
- Recall the “subject to” portion of the optimisation problem formulation
- We refer to a design point as being an element of a feasible set or feasible region
- So far $\mathscr{X} = \mathbf{dom}(f)$, but often times $\mathscr{X} \subset \mathbf{dom}(f)$
  - The former is unconstrained, the latter is constrained

Unconstrained vs constrained - Example

$$\min 2x^2 + 4$$

subject to

$$x \in \mathbb{R}$$



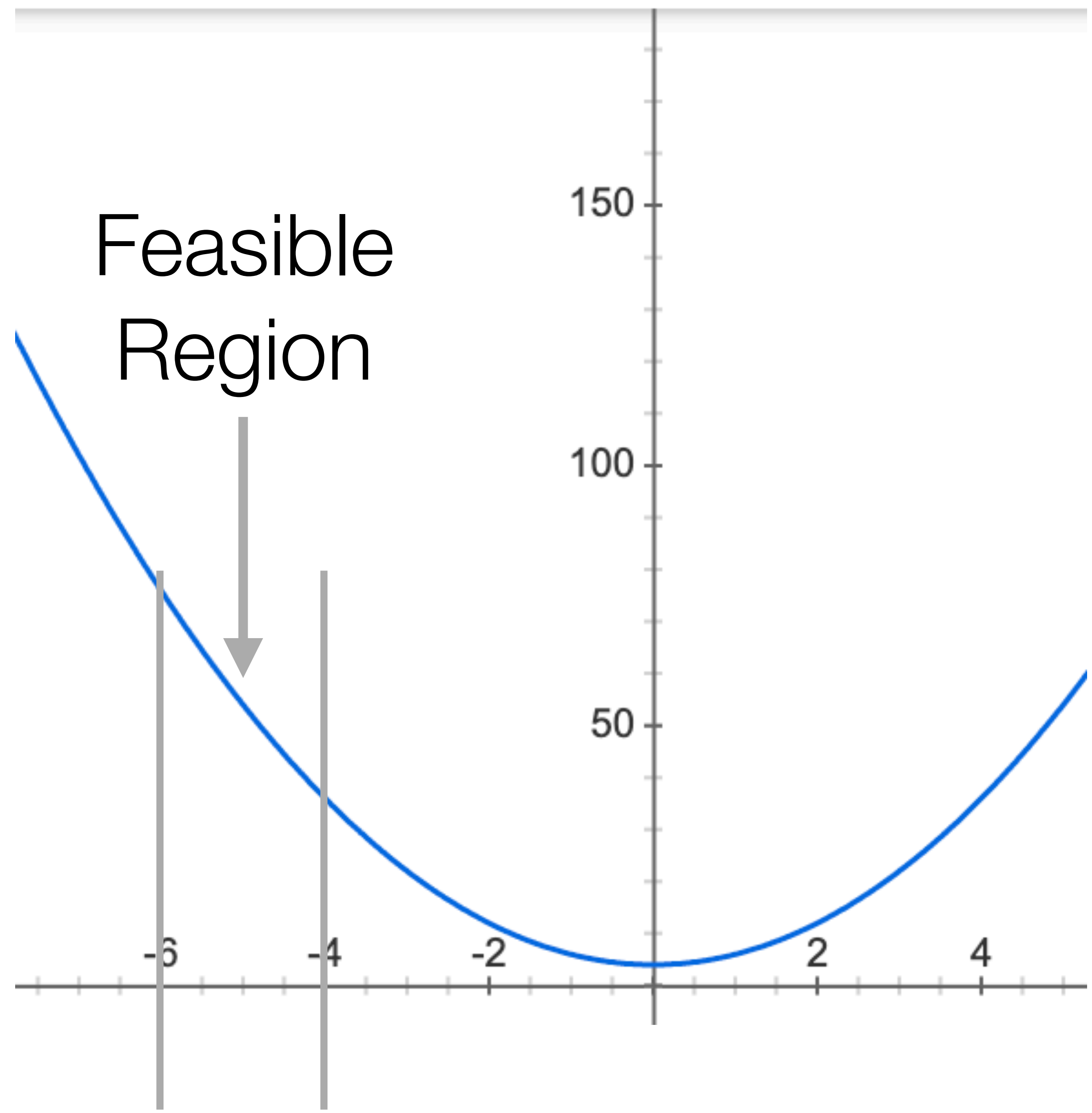
Unconstrained vs constrained - Example

$$\min 2x^2 + 4$$

subject to

$$-6 \leq x \leq 4$$

$$x \in \mathbb{R}$$



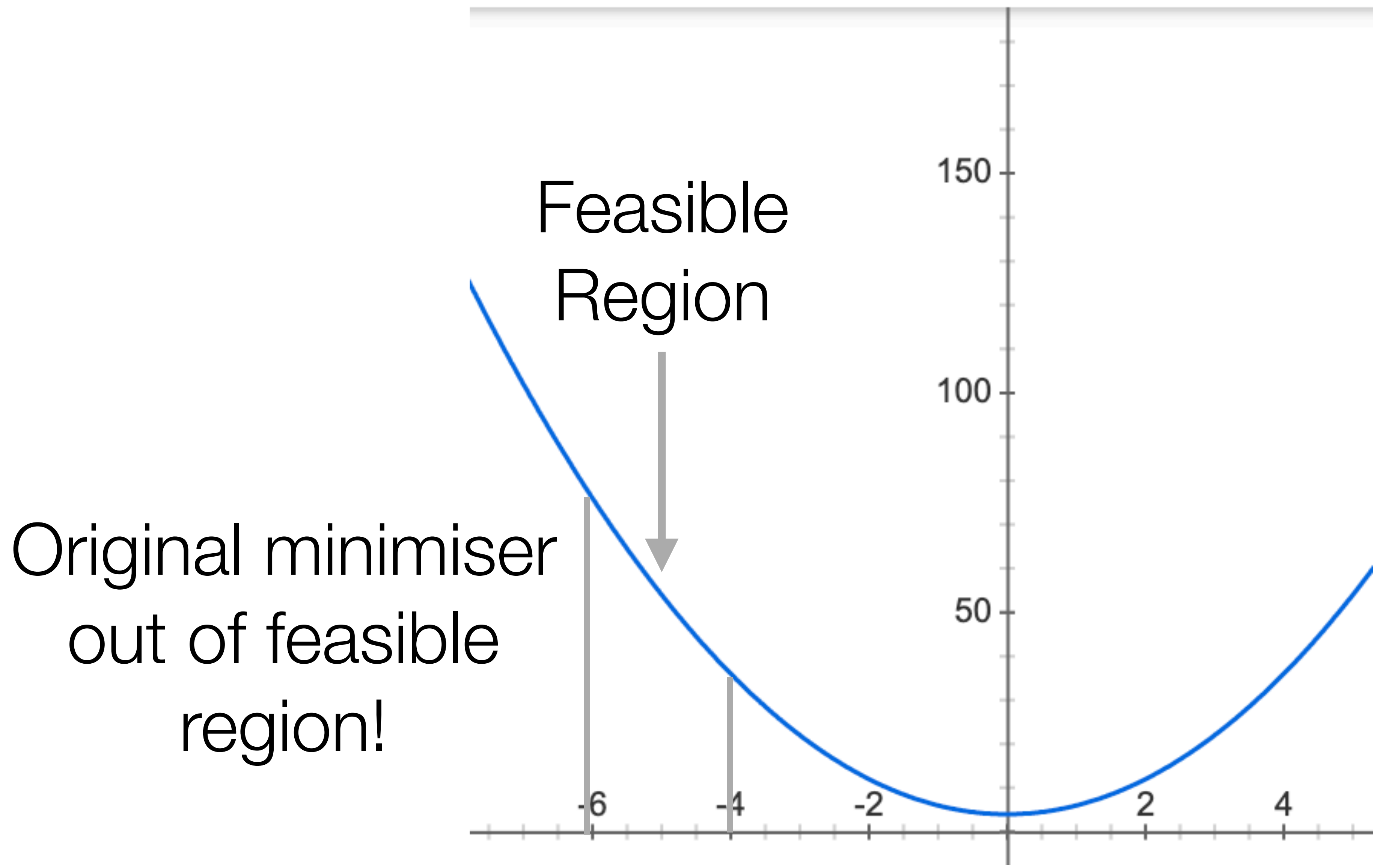
Unconstrained vs constrained - Example

$$\min 2x^2 + 4$$

subject to

$$-6 \leq x \leq 4$$

$$x \in \mathbb{R}$$

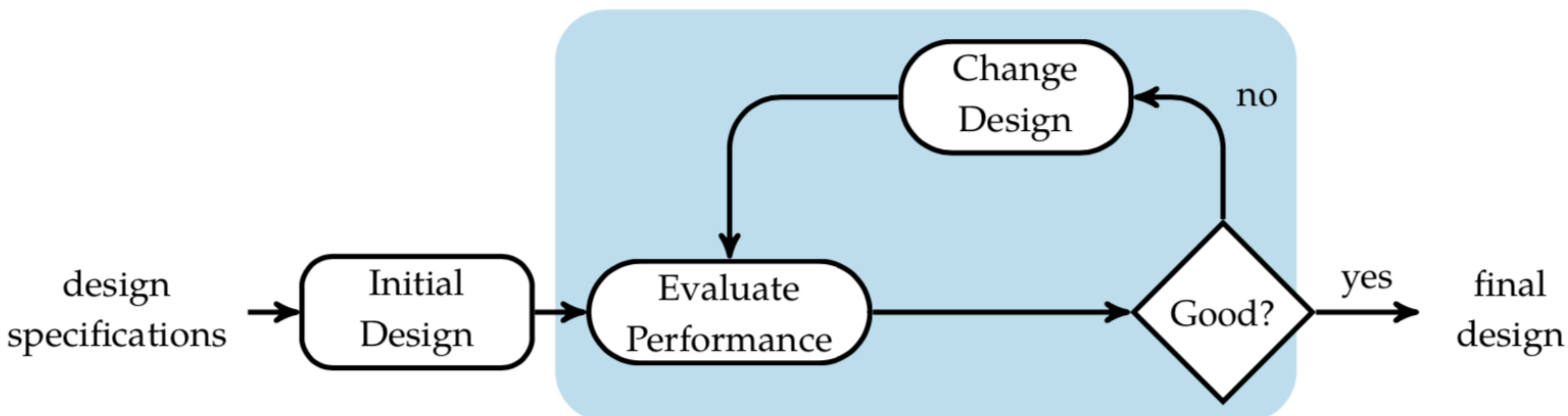


Constraint-Satisfaction Problems

- Sometimes we don’t care about the value of the objective function or have no such functions
- But we do have constraints
- We call such instances a CSP
- Examples:
  - Graph colouring
  - Halls’ marriage problem
  - n-Queens Problem
- MiniZinc is great at these!

Continuous vs Discrete

- The characteristics of the objective function's domain impacts the methods we can use greatly
- Continuous - real numbers, real vectors, real matrices
- Discrete - integers, integer vectors, integer matrices
- Many algorithms exploit continuity. So discrete optimisation is more difficult :(
    - Integer linear programming is in the class NP


design
specifications
Initial
Design
Evaluate
Performance
Change
Design
no
Good?
yes
final
design

Let’s formulate a problem!

- Wyndor Glass Ltd. produces two products A and B. A makes $2000 of profit per unit, and B makes $3000 of profit per unit. There are three plants that manufacture three different components that are used to manufacture A and B. Plant 1 can only operate for 10 hours per day. Plant 2 can only operate for 6 hours a day. Plant 3 can only operate for 15 hours a day. Each unit requires different amounts of a plants services. These are summarised in the following table

Let’s formulate a problem

Plant\Time Needed	Product A	Product B
Plant 1	2	3
Plant 2	1	0.3
Plant 3	2	1.5

What is this is an optimisation problem?

Problem Conversion

- Sometimes we can convert an
  - Constrained version to an unconstrained version
  - Discrete problem to a continuous problem
  - **to approximate a solution**
- Sometimes with provable guarantees :D
  - Won’t look at the guarantees much

Constrained to unconstrained

- Some methods we will look at will have different ways to dealing with converting constrained to unconstrained
- But in general, we can use penalty methods
- Basic idea:
    - Keep count of the number of constraints violated
    - multiply by some penalty factor
    - add to cost (minimisation problem) or subtract from objective (maximisation problem)

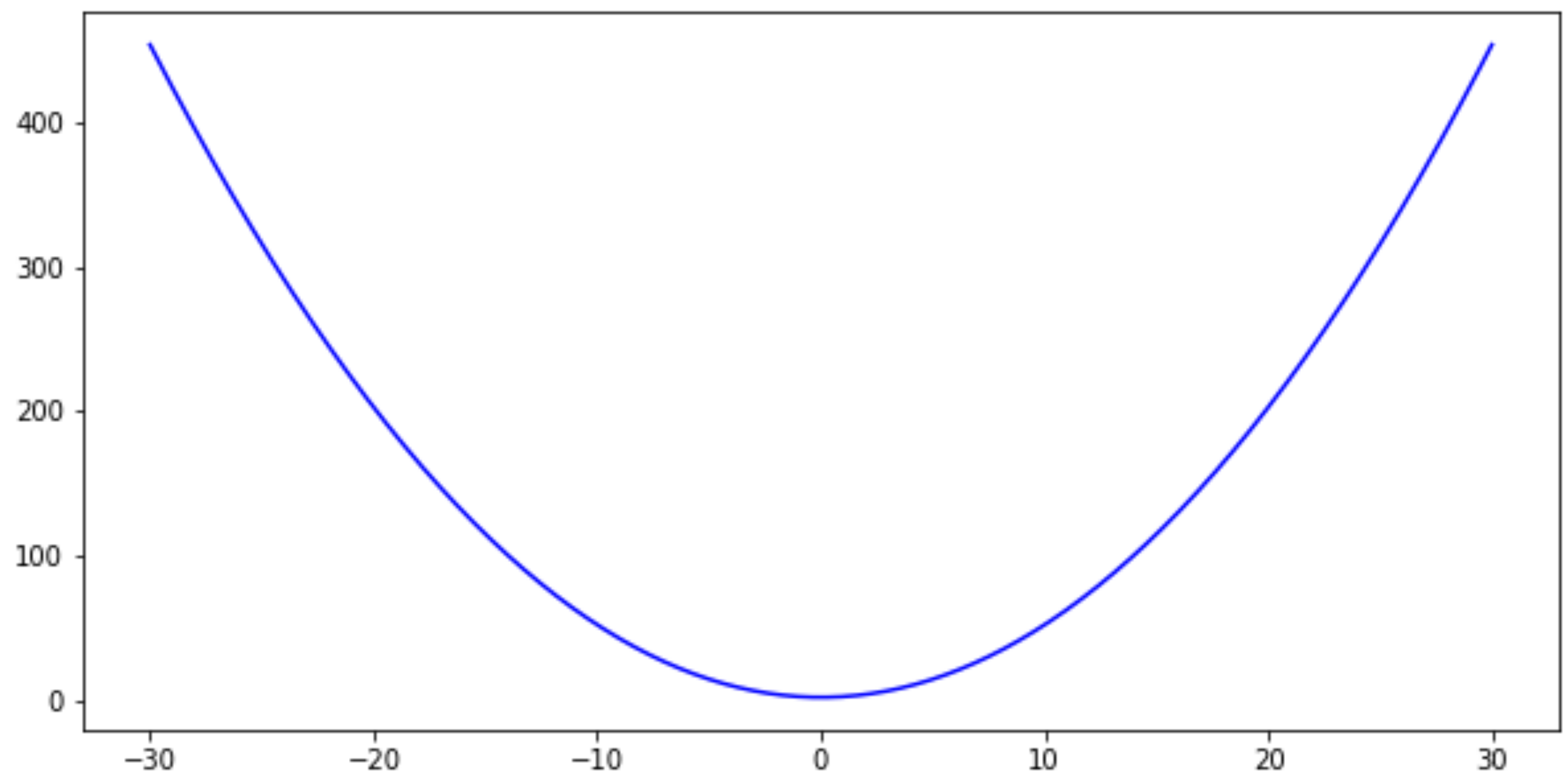
Gradient Descent (and friends)

- Iterative method for minimisation of convex functions using gradient
- Many improvements, but will look at simple variation for now
- Start and random answer and iteratively refine answer until convergence criteria (usually number of iterations (called epochs) is met)
- Used when solving for root of gradient is not feasible or possible
- The basis of many machine learning algorithms

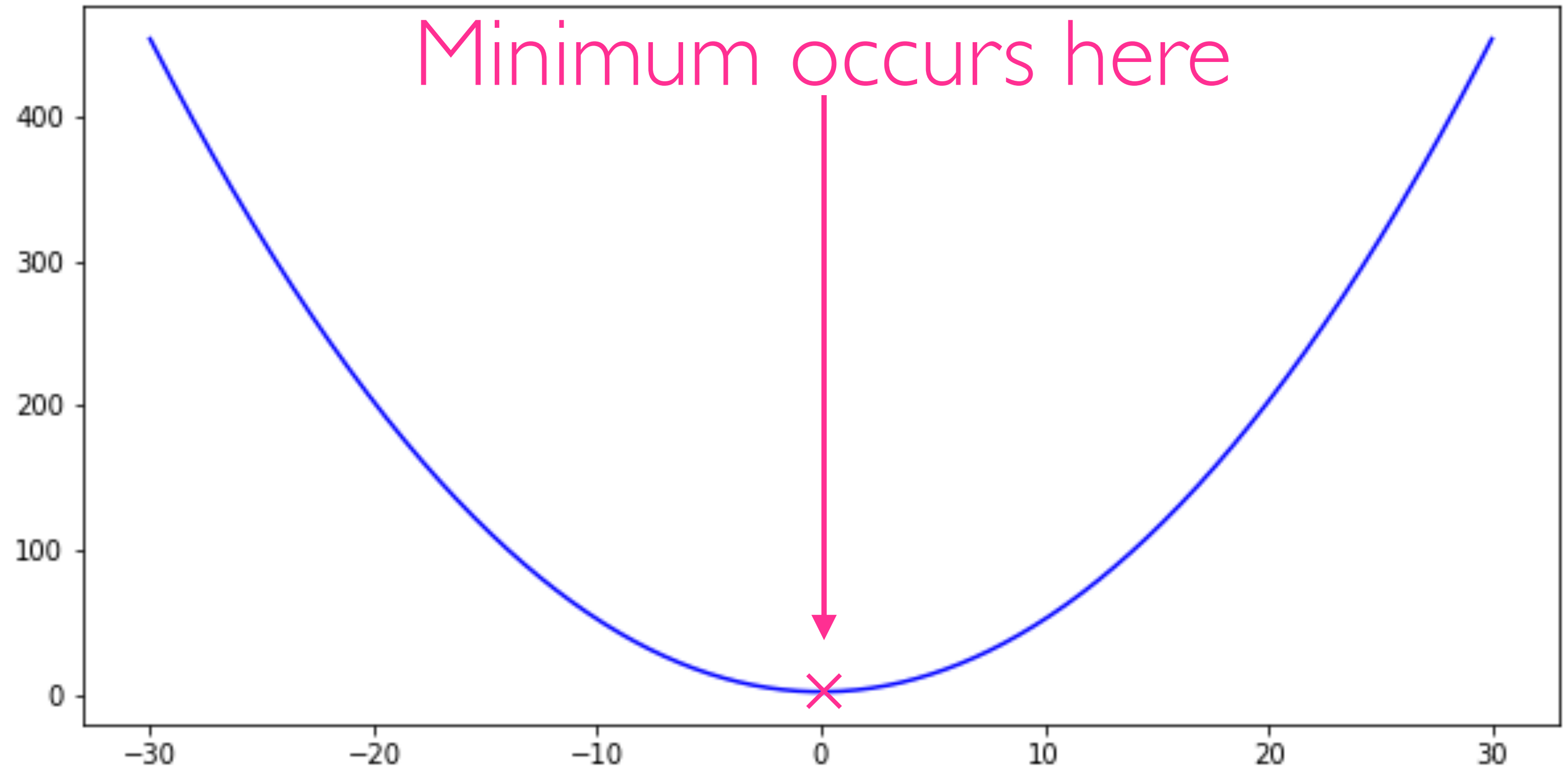
Gradient Descent

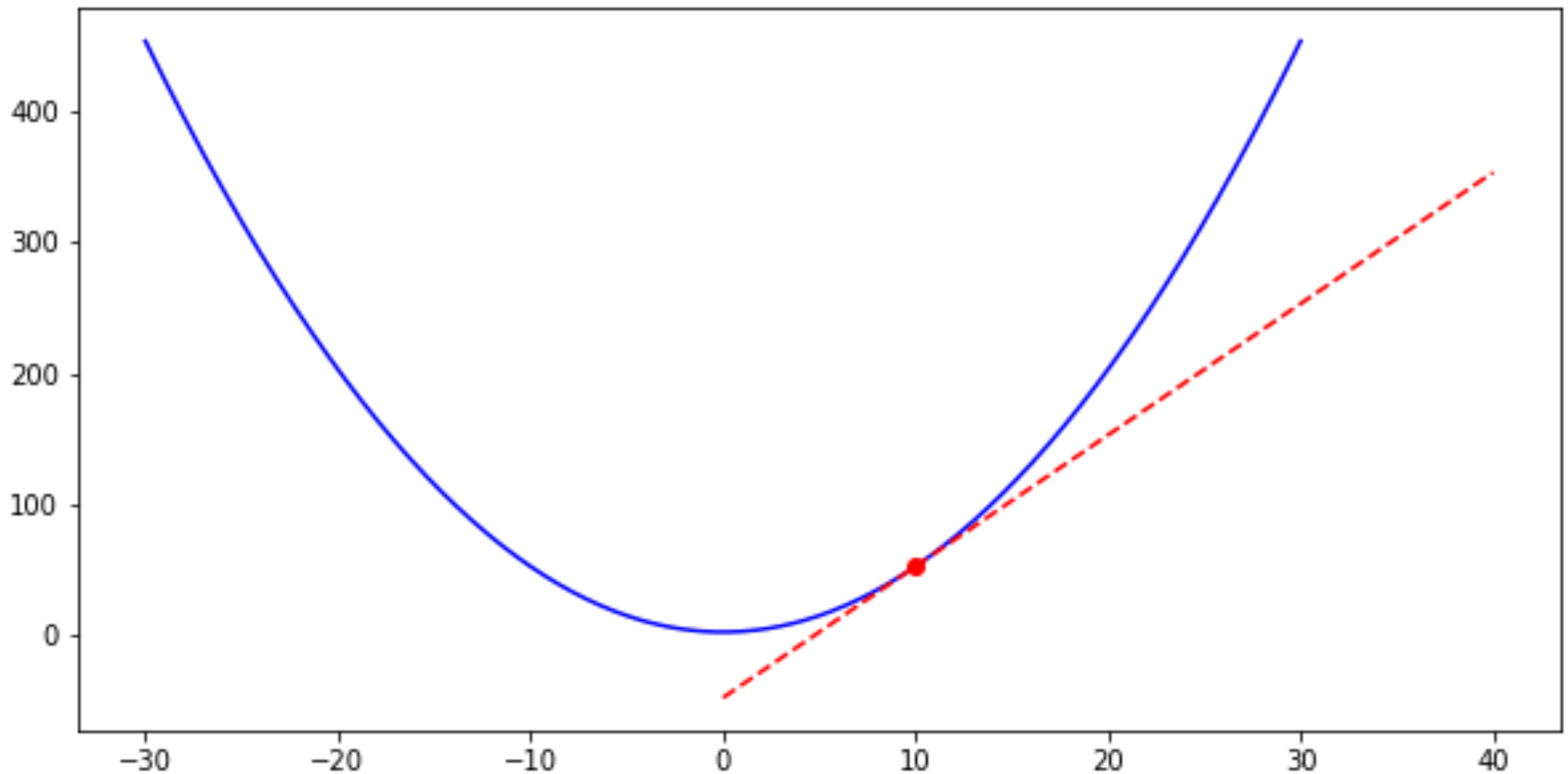
- Core idea: gradient of tangent gives us direction of steepest ascent
- Core idea: negation of gradient, should give us direction of steepest descent

$$f(x) = \frac{1}{2}x^2 + 3$$

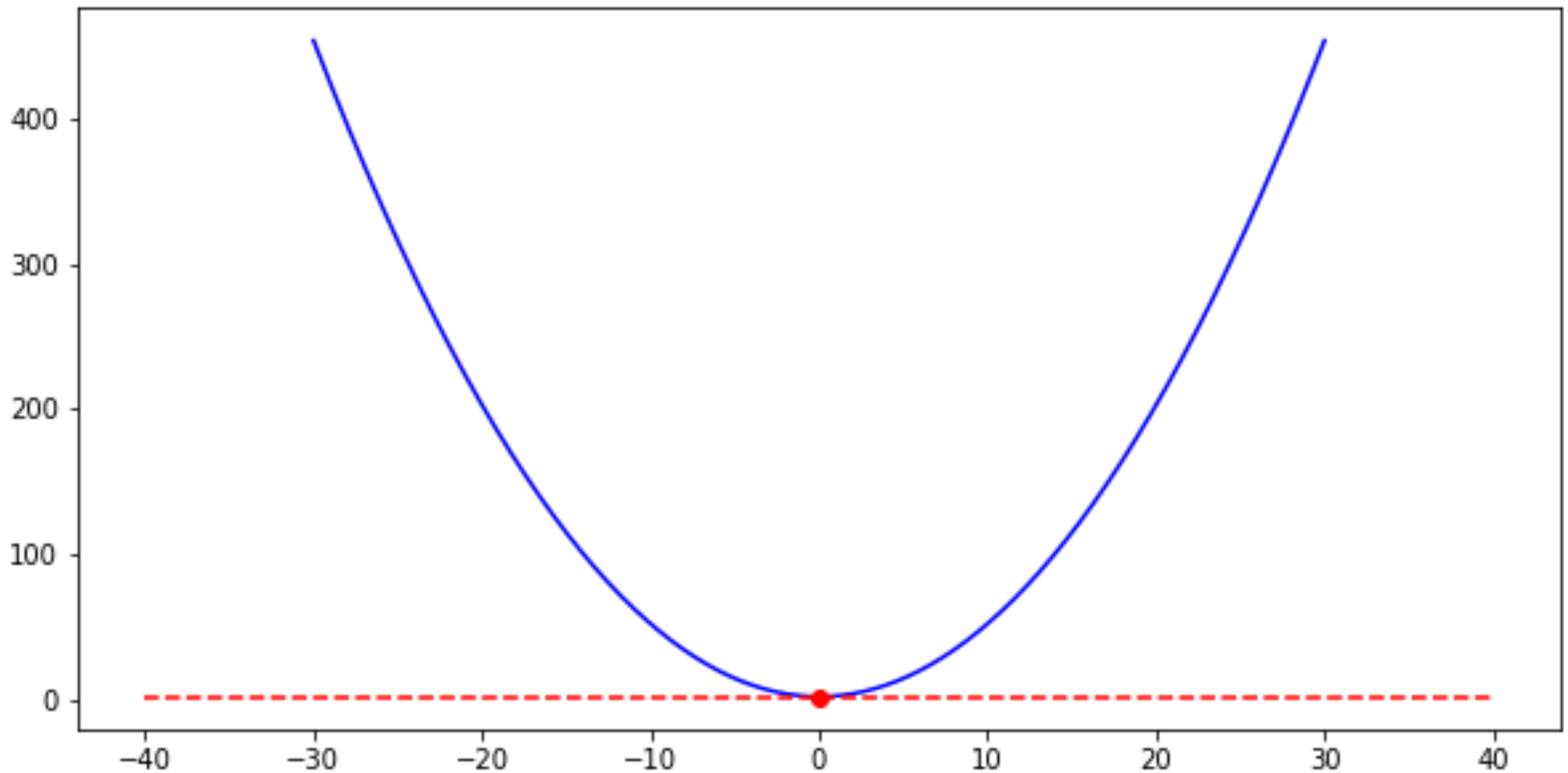


$$f(x) = \frac{1}{2}x^2 + 3$$





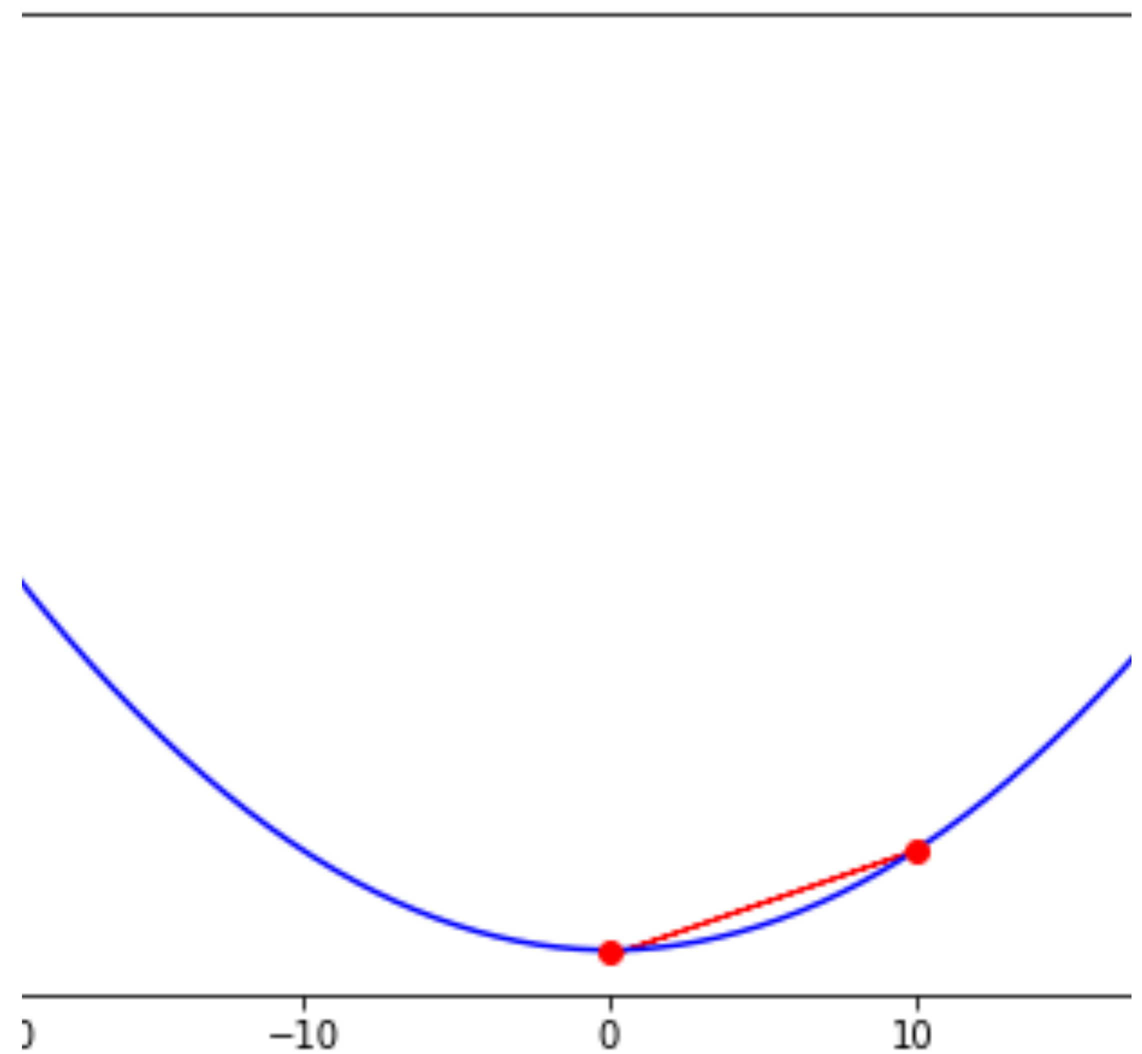
400
300
200
100
0
−30
−20
−10
0
10
20
30
40



400
300
200
100
0
−40
−30
−20
−10
0
10
20
30
40

Gradient Descent

- Choose random point, say $x = 10$, and a step size $\alpha = 1$
- Compute gradient at point, $f'(x)$
  - $f'(10) = 10$
- Move in the direction opposite to $f'(x)$ scaled by α
  - $-\alpha f'(10) = 10$
- Compute new point
  - $x = 10 - 10 = 0$



Gradient Descent Algorithm

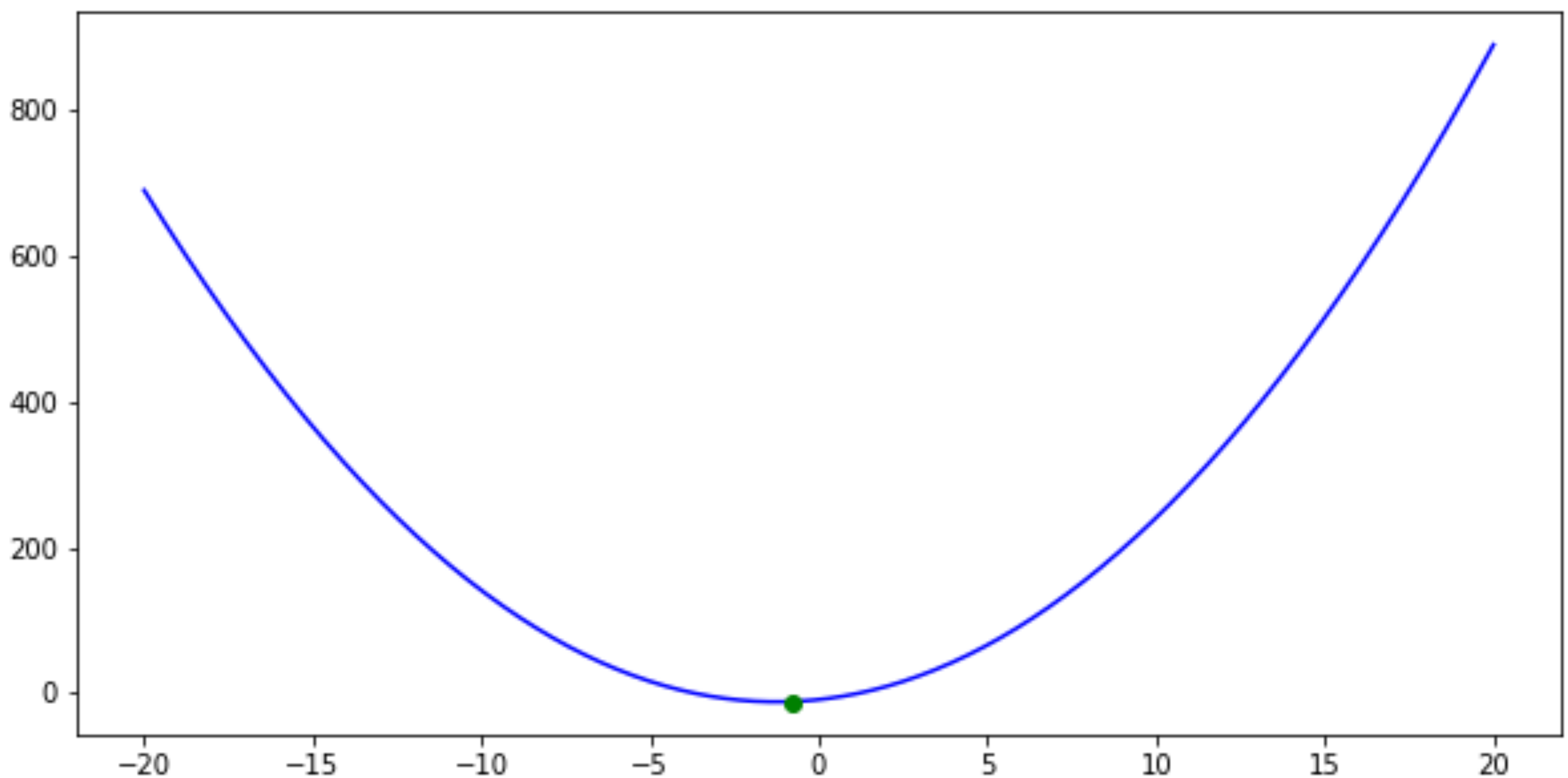
```
function gd(f, f’, α , lo=100, hi=100):
  x = uniform_random(lo, hi)
  grad_t = f’(x)
  while not converged:
    x = x - αf’(x)
  return x
```

Gradient Descent - Step Size or Learning Rate

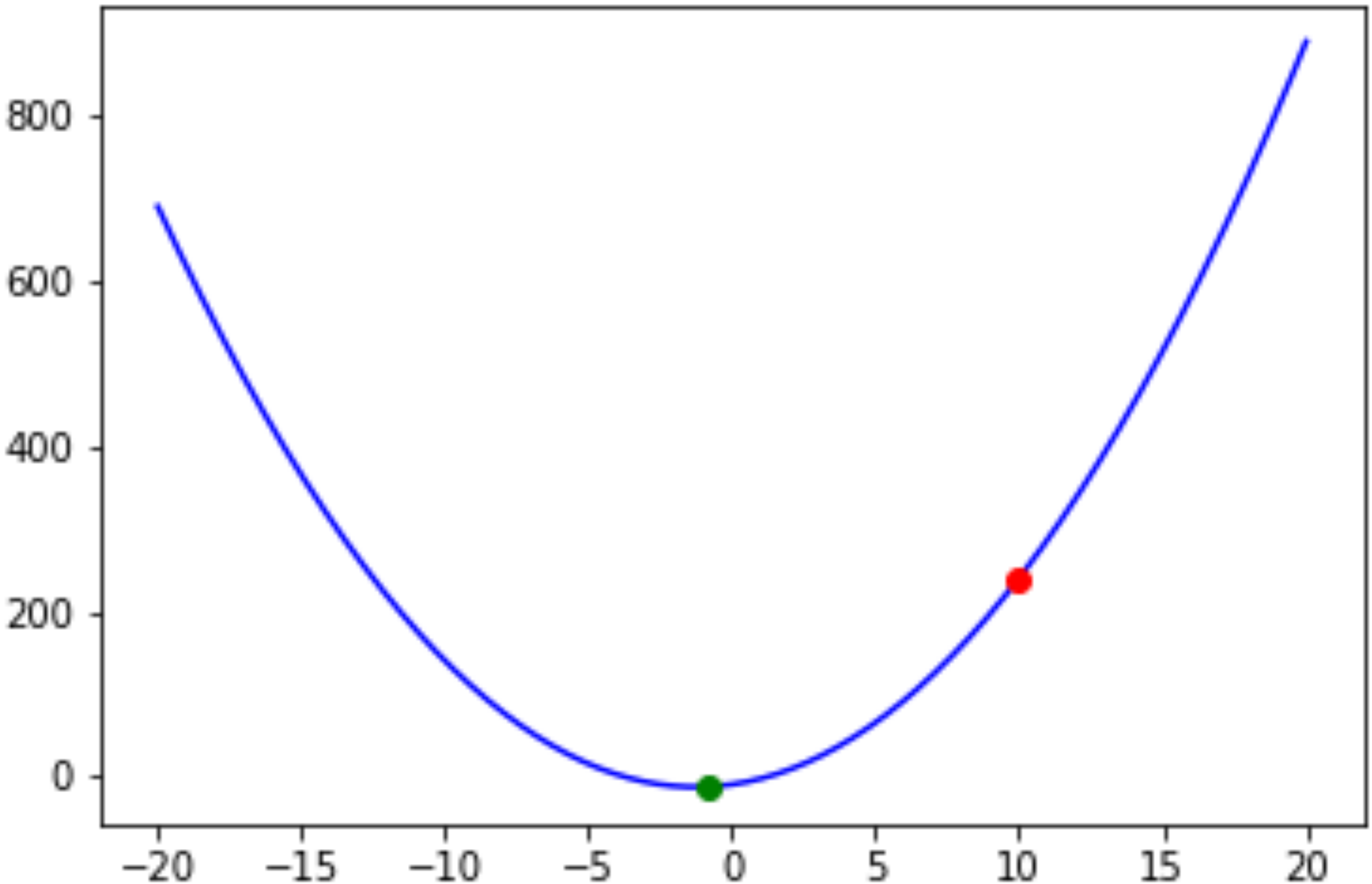
- The step size, α, also called the learning rate can have impact on convergence

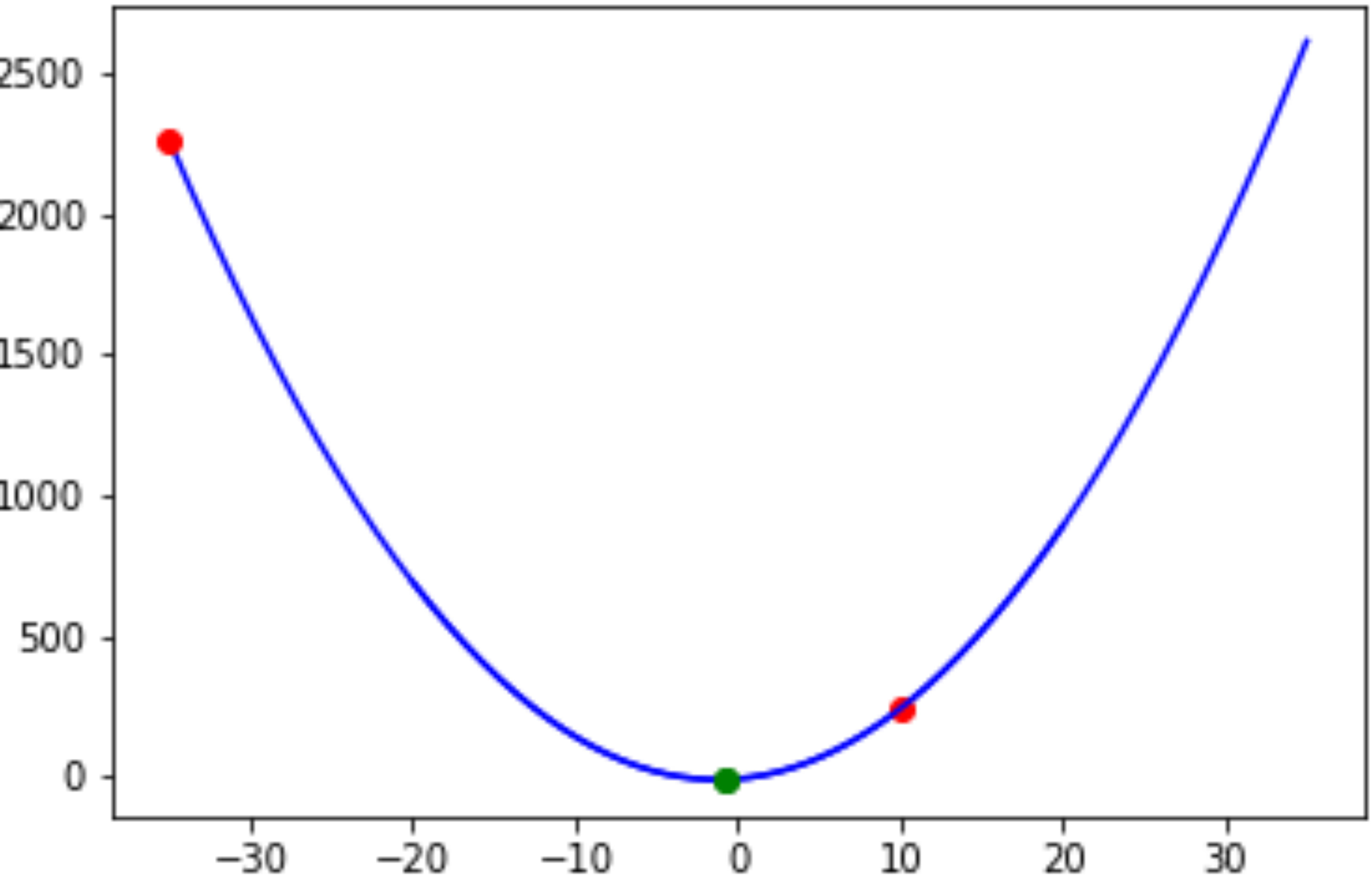
- Too large an α, we don't converge

- Too small an α, we converge slowly

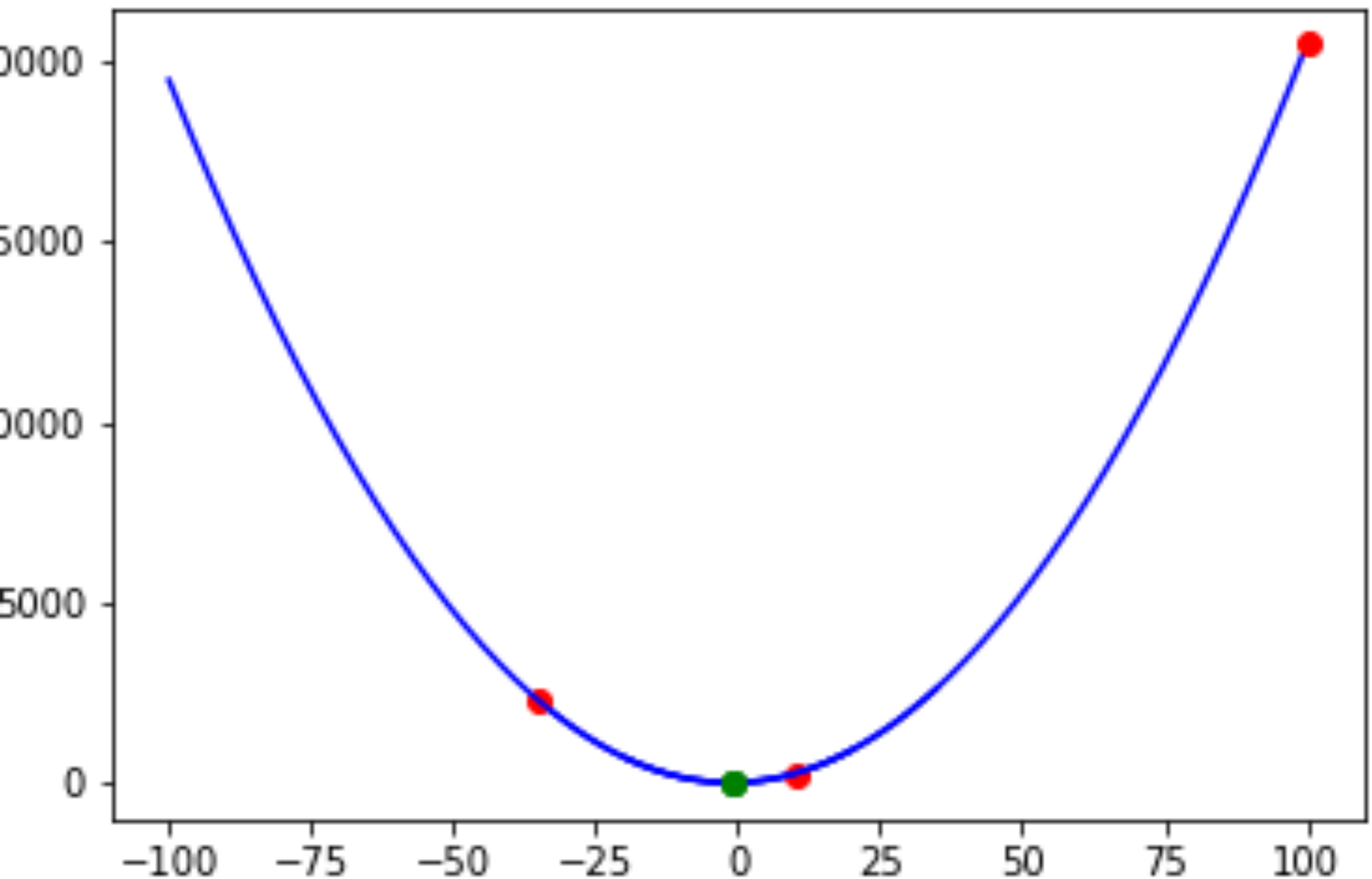


800
600
400
200
0
−20
−15
−10
−5
0
5
10
15
20


800
600
400
200
0
−20
−15
−10
−5
0
5
10
15
20



2500
2000
1500
1000
500
0
−30
−20
−10
0
10
20
30


0000
5000
0000
5000
0
−100
−75
−50
−25
0
25
50
75
100

Gradient Descent

- Large learning rates cause us to jump to far
- We can miss the optimal point and end up moving away from it or bounce around it
- Some modifications of gradient descent adjust the learning rate depending on the progress of the algorithm
  - AdaGrad, ADAM, RMSProp, etc…

Gradient Descent - Vectors or Multivariate

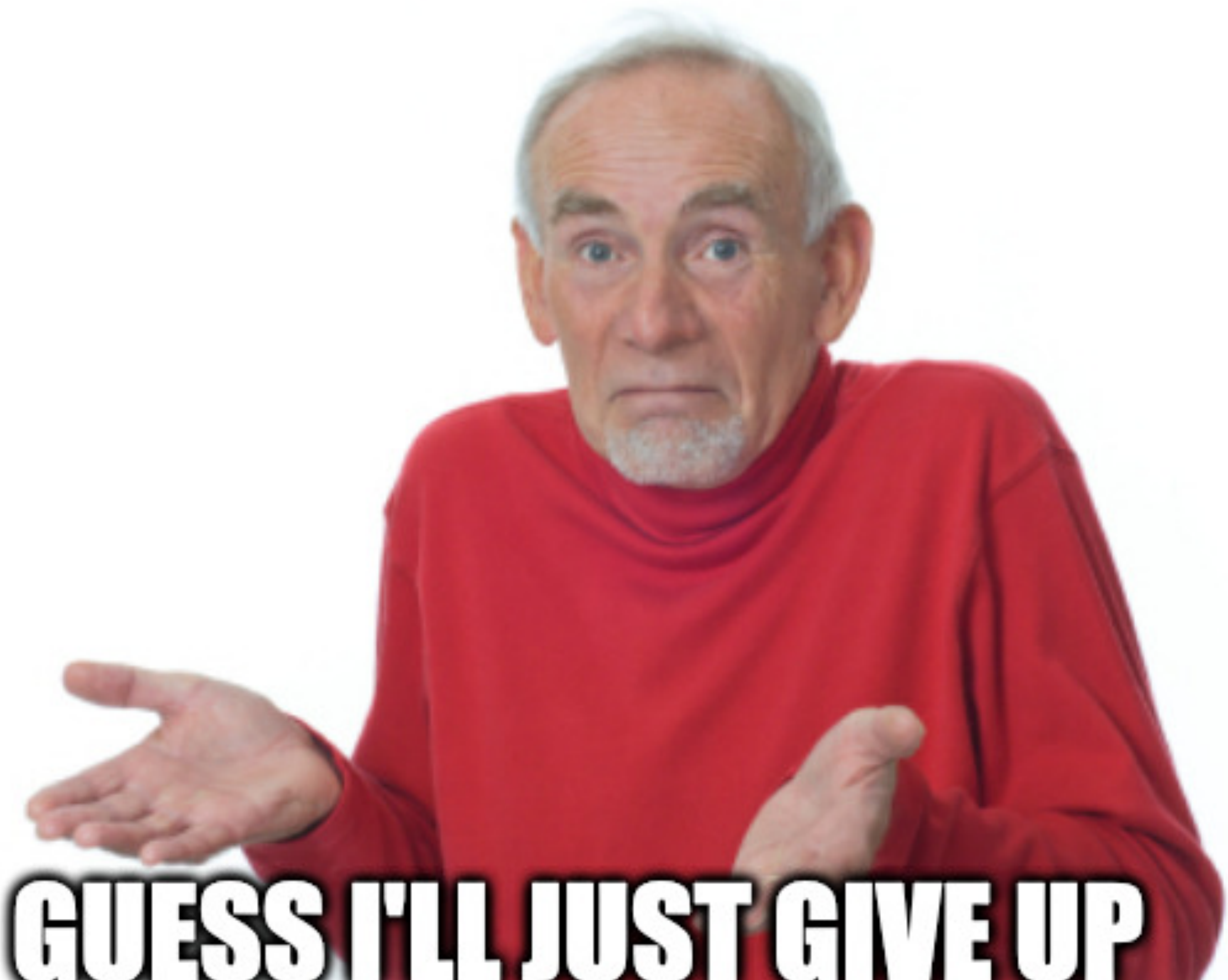
- Gradient Descent is trivially extensible to multivariable or vector cases?
- Just use partial derivative or vector derivate instead!
- Will see examples in the lab using PyTorch and by hand

Backtracking

- Can be used to solve CSPs
- Suppose that we start in state, S_0 that is in accord with our constraints but incomplete. We need to take m actions or make m decisions to reach S_m such that we find S_m that obeys our constraints
- We have an action set A, $|A| = n$ of actions that we can take. Our actions are labeled $a_1, a_2, \ldots, a_n$

Backtracking

- Suppose that we take action a_1, and this leads us into state S_1. S_1 is a valid state. We now need to move onto S_2
- Suppose that we try all actions and all possible S_3 s are invalid
- What do we do?

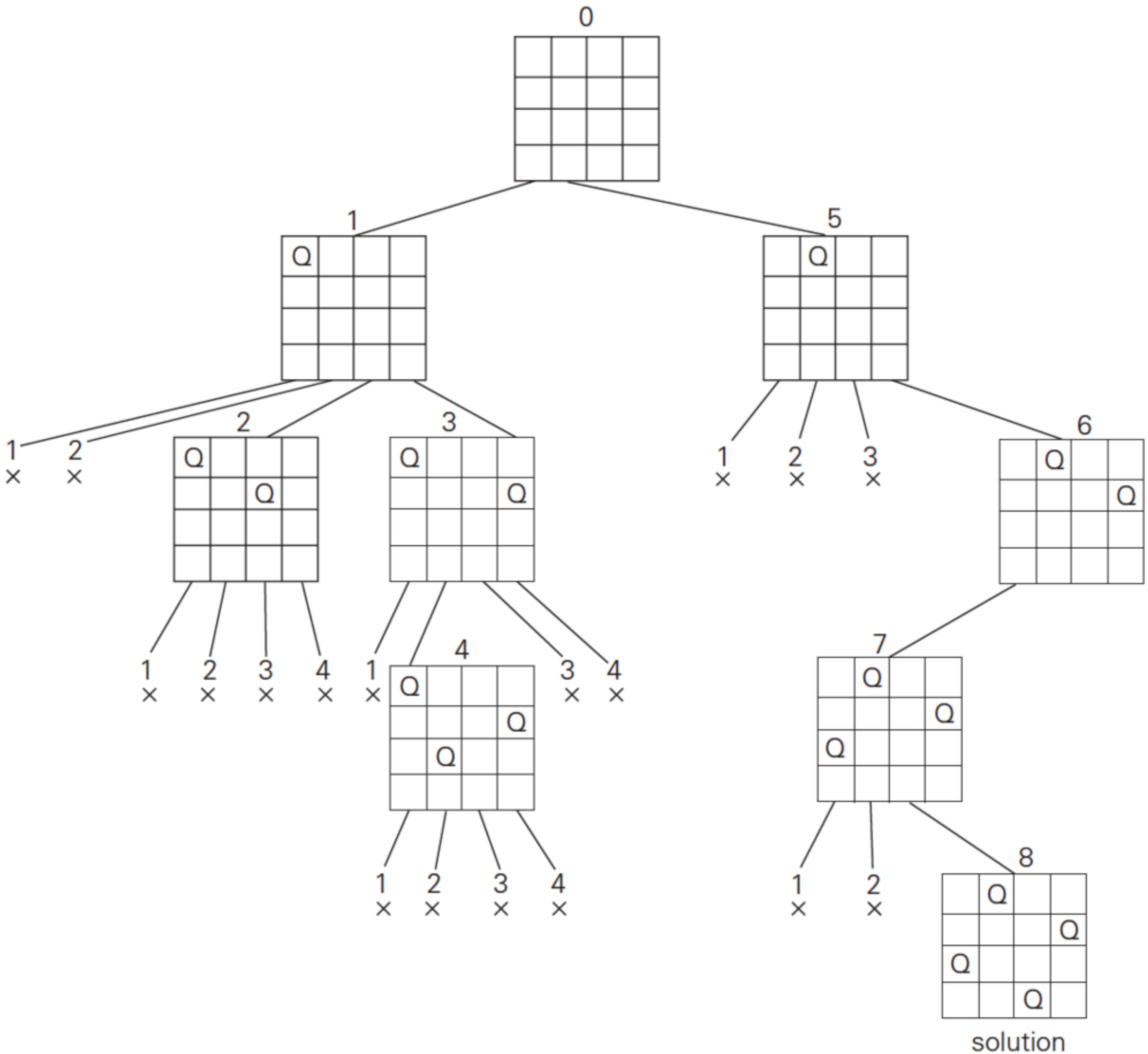

GUESS I'LL JUST GIVE UP

Backtracking

- No!
- We assume then that S_2 was a bad-turn or dead-end.
- So we ***backtrack*** to S_1 and start from where we left of in our action set - a_1. We now consider a_2. And repeat until we either reach a valid S_m and report success or backtrack to S_0, exhaust all of our actions and report failure

Backtracking - N Queens

- Consider an $n \times n$ chessboard.
- We want to find a way to place n queens on it such that no queen can attack any other queen
- Remember, that in chess, a queen can move diagonally, horizontally, or vertically any number of squares


0
1
2
3
4
5
6
7
8
solution

MiniZinc

- MiniZinc is a DSL for solving optimisation problems
- Allows us to move between mathematical formulation and working code easily
- Will use for CSP and for some optimisation problems
- Will look at it in lab next week

```
enum DISH;
int: capacity;
array[DISH] of int: satisf;
array[DISH] of int: size;

array[DISH] of var int: amt;

constraint forall(i in DISH)(amt[i] >= 0);
constraint sum(i in DISH)(size[i] * amt[i]) <= capacity;

solve maximize sum(i in DISH)(satisf[i] * amt[i]);

output ["AMOUNT = ", show(amt), "\n"];
```