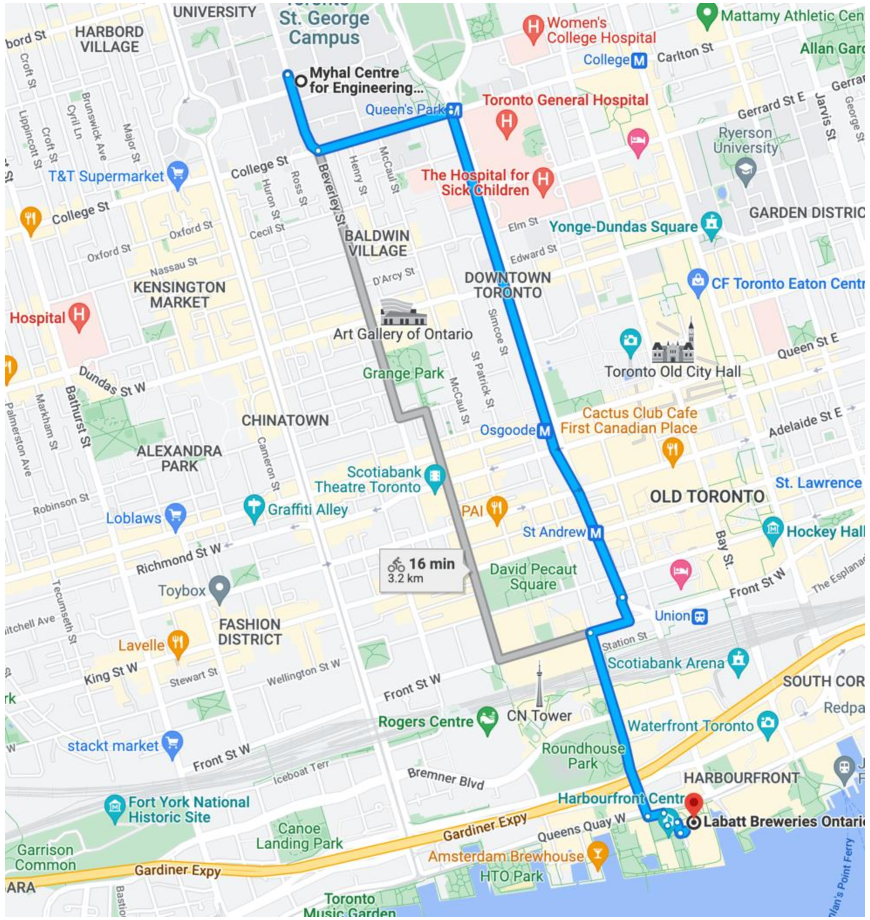
Optimization, Constraints and Linear Programming

Alex Olson

What is Optimization?

- Major field within Data Analytics, Operations Research and Management Science
- Basic idea: find the values of the decision variables that maximize (or minimize) the objective value, while staying within the constraints
- How do I find the shortest route to bike to the harbourfront, without breaking traffic laws?

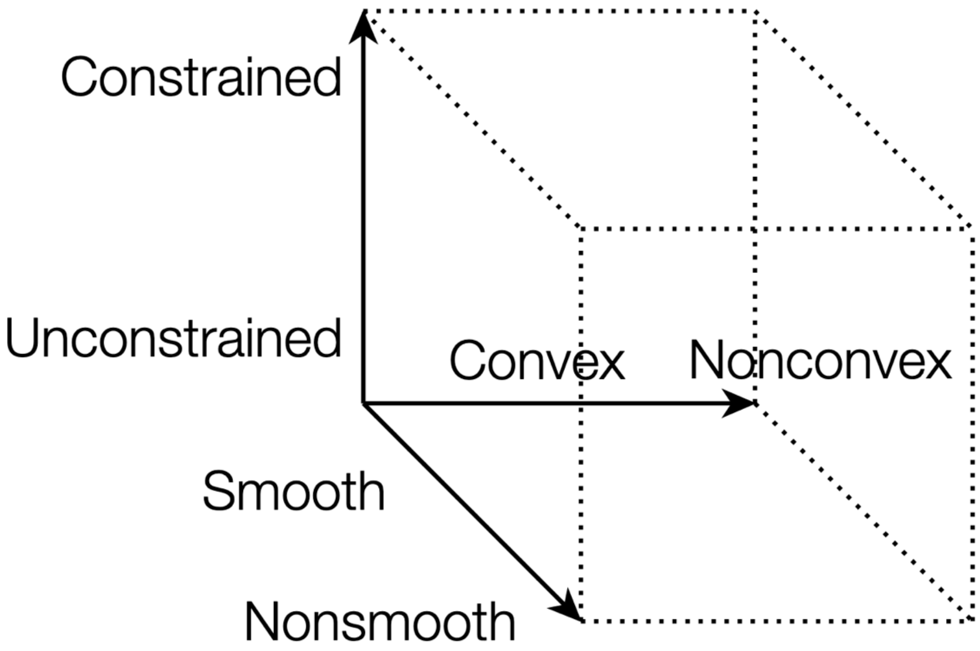


What is Optimization?

- In machine learning, we usually want to <u>minimize</u> the result of a loss function
- A huge number of ML problems can be solved using optimization
  - e.g. regression, classification, maximum likelihood
- If we can use optimization, we get access to powerful tools which can find our answer

Classes of optimization problem

- Many different types of problem can be framed as an optimization problem
- Three main distinctions help to define them
- Constrained vs Unconstrained
- Convex vs Nonconvex
- Smooth vs Nonsmooth (less important)



Constrained vs Unconstrained

- Constraints are conditions on what answers are acceptable
- When finding the shortest driving route, you are really finding the shortest *legal* driving route
- When scheduling employees, have to factor in their availability

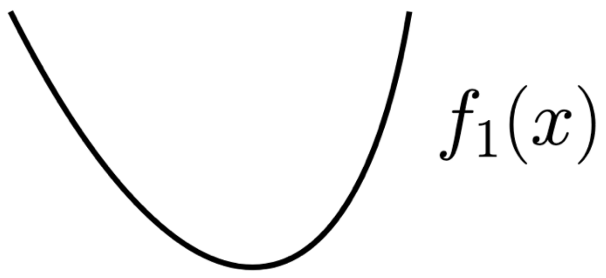
$$\underset{x}{\text{minimize}} \quad f(x)$$

vs

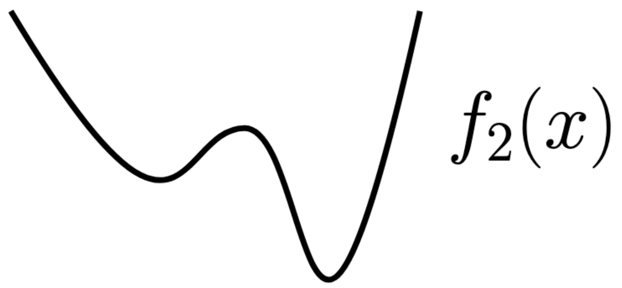
$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{aligned}$$

Convex vs Nonconvex

- A function is convex if there is exactly one "bottom" point – the global minimum
- This makes the problem much easier to solve because as long as the error is decreasing, you are getting closer to the best answer
- If the function is nonconvex, you can be "tricked" by a local minimum



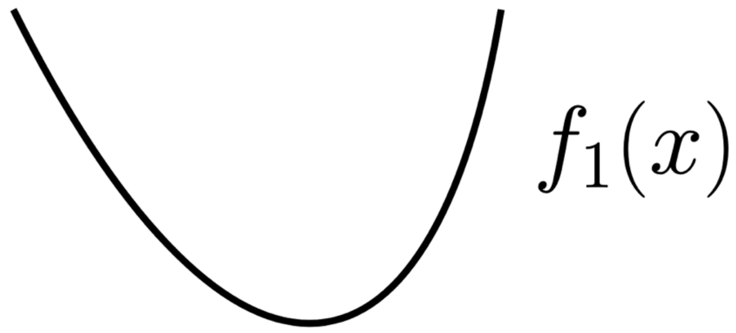
Convex function



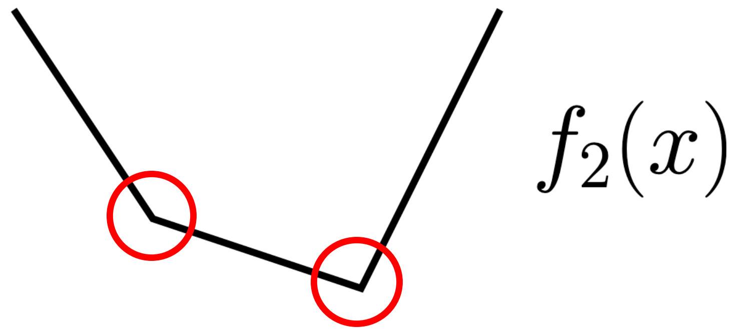
Nonconvex function

Smooth vs Nonsmooth

- Many modelling methods depend on calculating the derivative of the error – this tells us how to change our answer to get closer to the minimum
- If the function is nonsmooth, there are points (red) where it is not possible to differentiate



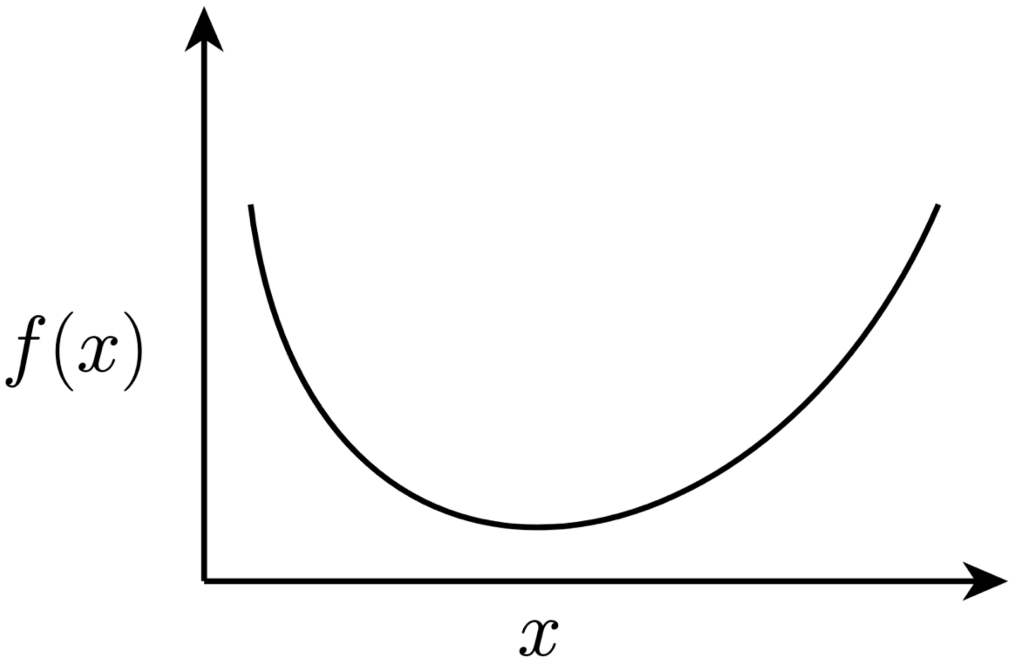
Smooth function



Nonsmooth function

Solving an optimization problem

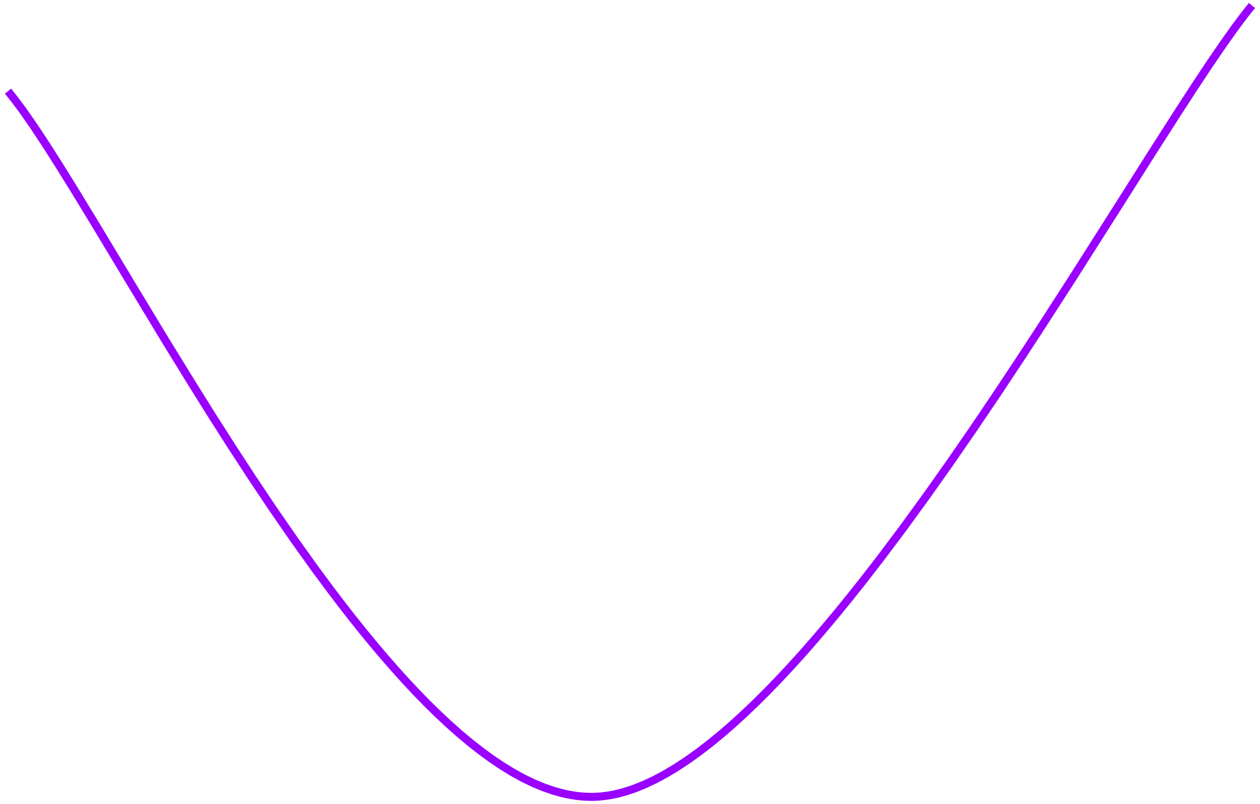
- Let's start with the simplest case: unconstrained, convex, smooth function
- We just need to find the point where the curve is flat (i.e. derivative is zero) - this is the minimum
- If the function is <u>very</u> simple, we can just calculate this value directly
- Otherwise, we can use gradient descent



Gradient descent

For some loss function $L(\mathbf{w})$, gradient $\nabla L(\mathbf{w})$ points towards in direction of steepest ascent.

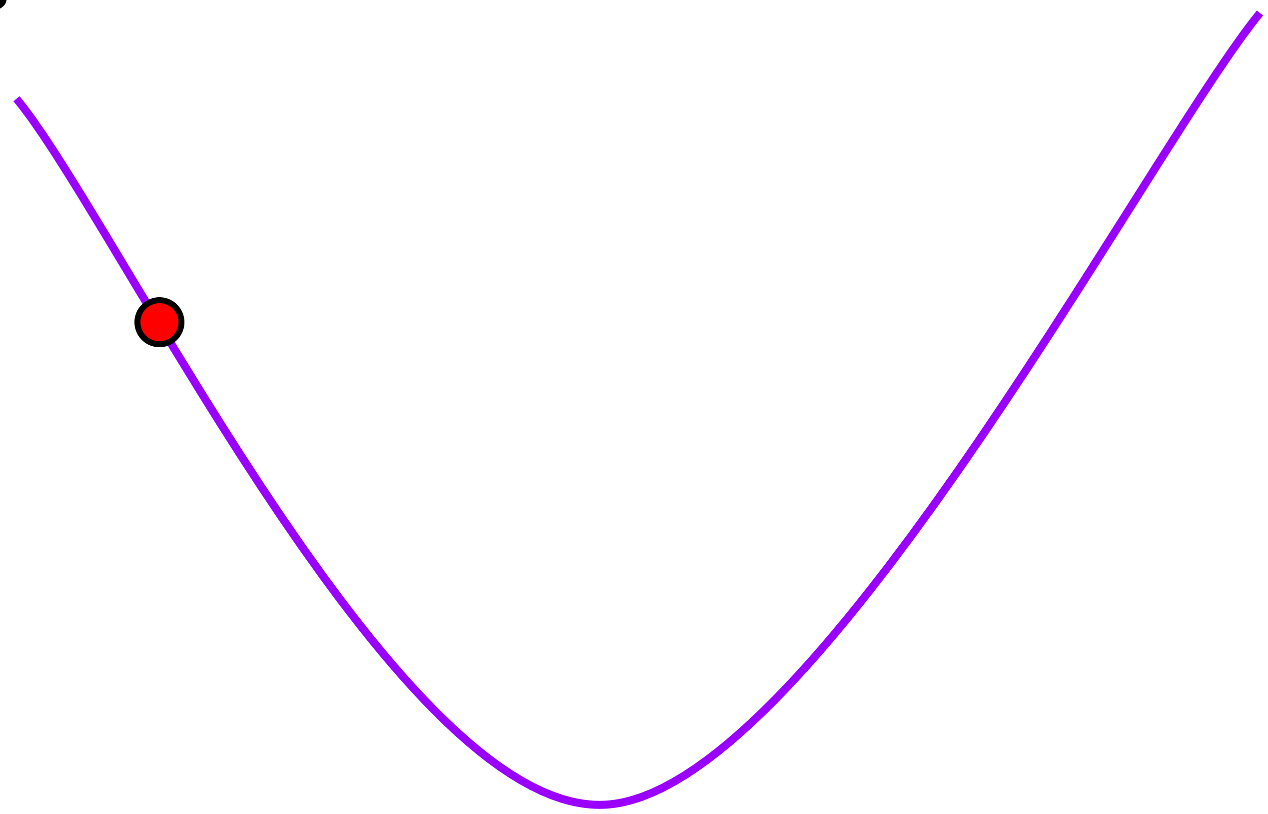
In 1d, either points left or right



Gradient descent

For some loss function L(**w**), gradient ∇L(**w**) points towards in direction of steepest ascent.

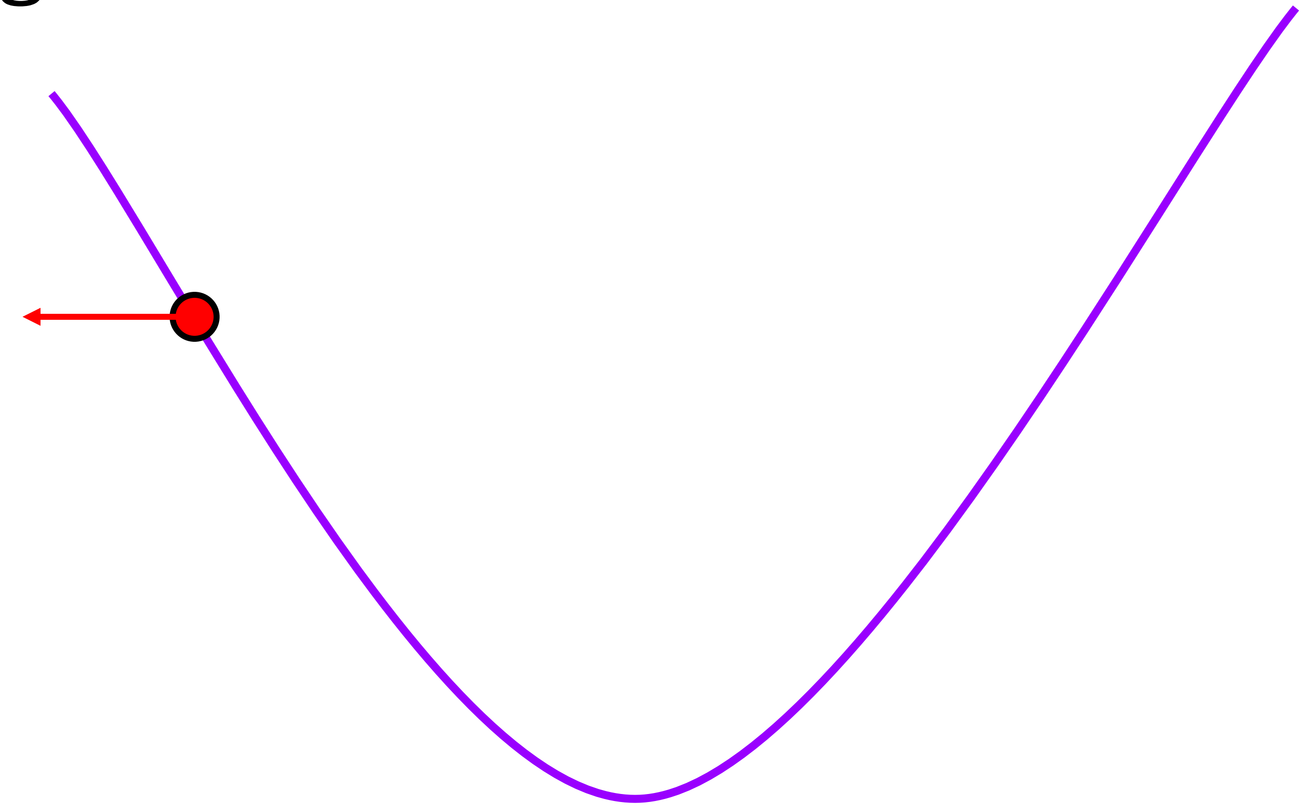
In 1d, either points left or right



Gradient descent

For some loss function L(**w**), gradient ∇L(**w**) points towards in direction of steepest ascent.

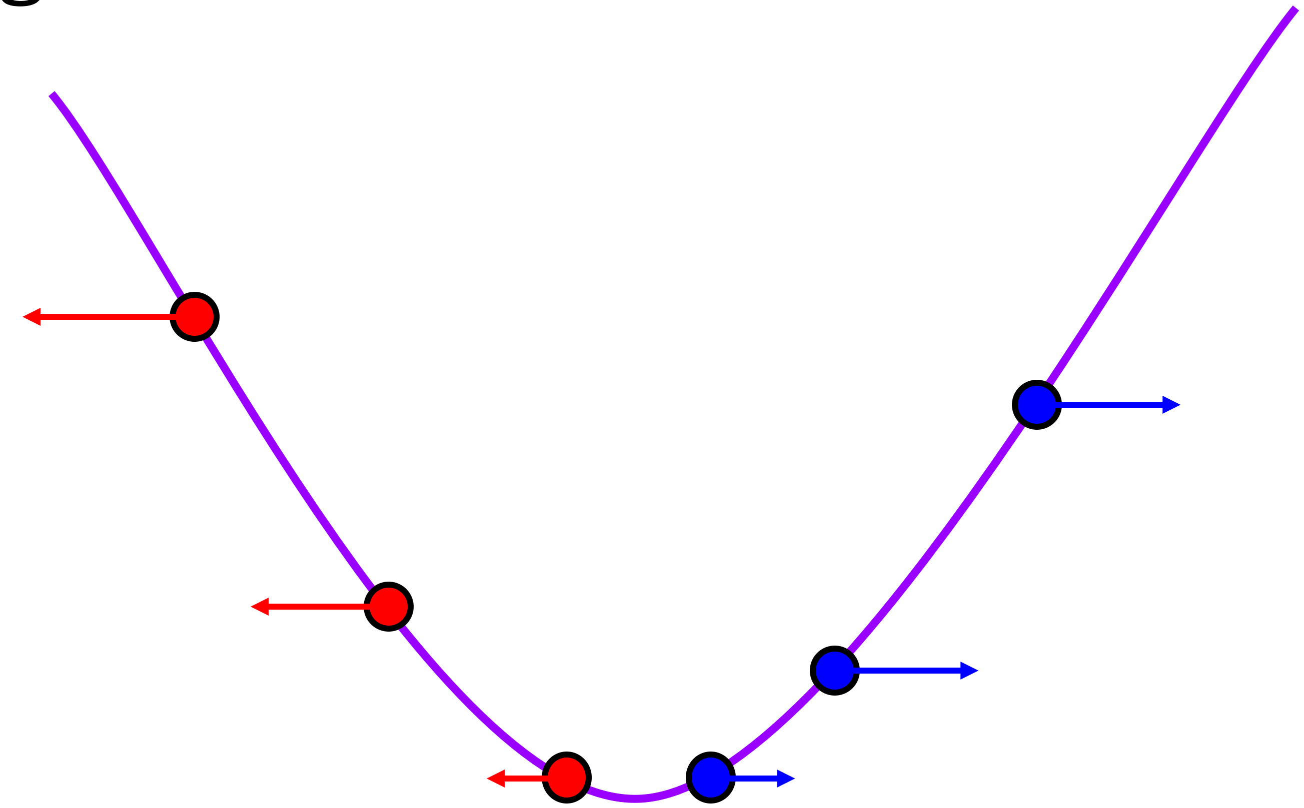
In 1d, either points left or right



Gradient descent

For some loss function L(**w**), gradient ∇L(**w**) points towards in direction of steepest ascent.

In 1d, either points left or right



Gradient descent

For some loss function L(**w**), gradient ∇L(**w**) points towards in direction of steepest ascent.

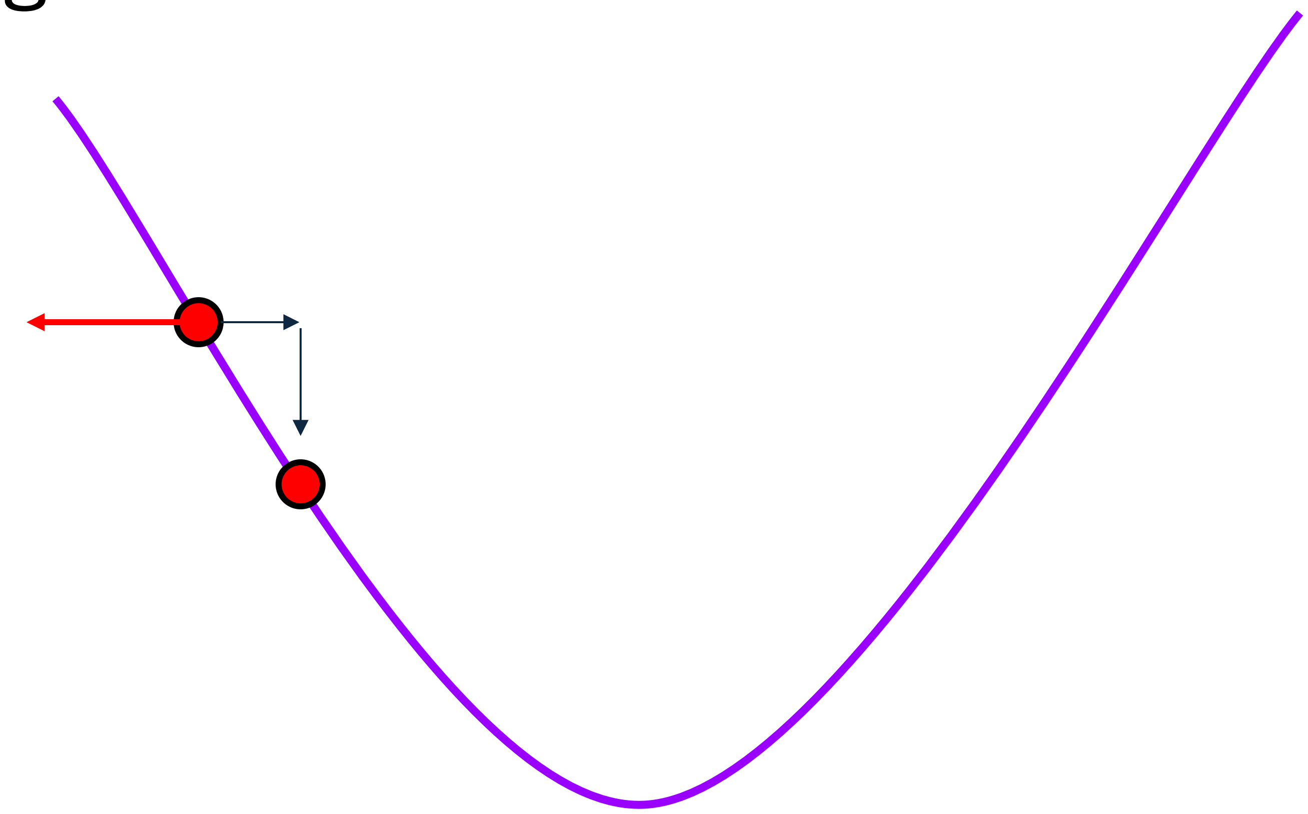
In 1d, either points left or right

Algorithm:

Take derivative

Move slightly in other direction

Repeat



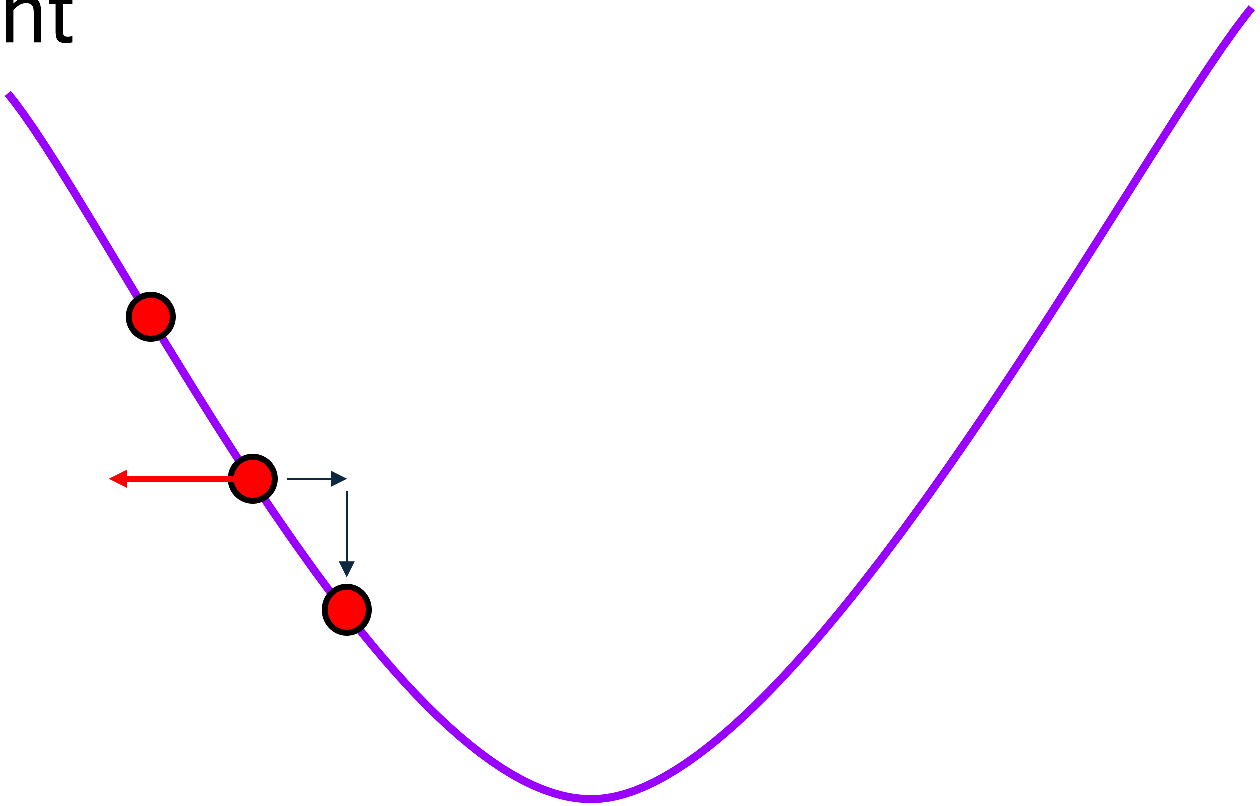
Gradient descent

For some loss function L(**w**), gradient ∇L(**w**) points towards in direction of steepest ascent.

In 1d, either points left or right

Algorithm:

Take derivative
Move slightly in other direction
Repeat



Gradient descent

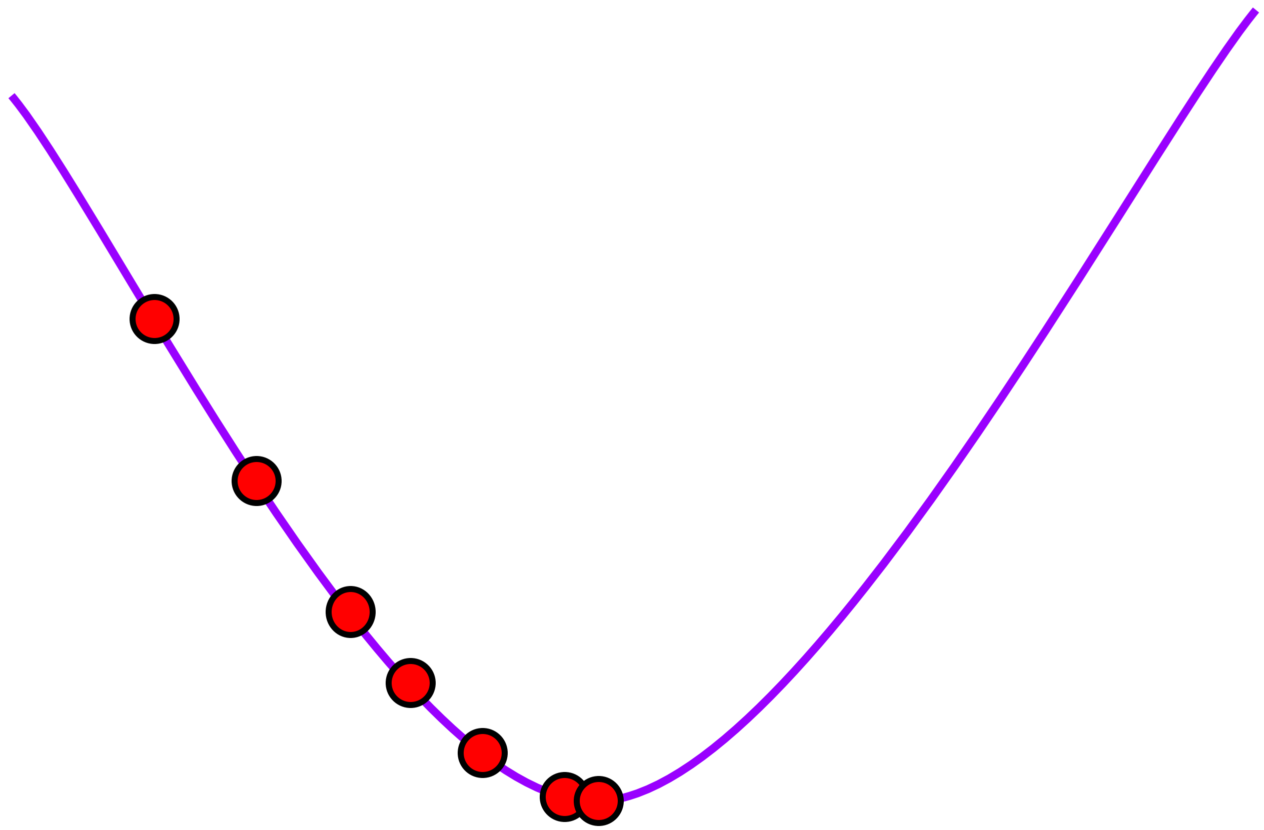
Algorithm:

Take derivative

Move slightly in other direction

Repeat

End up at local optima

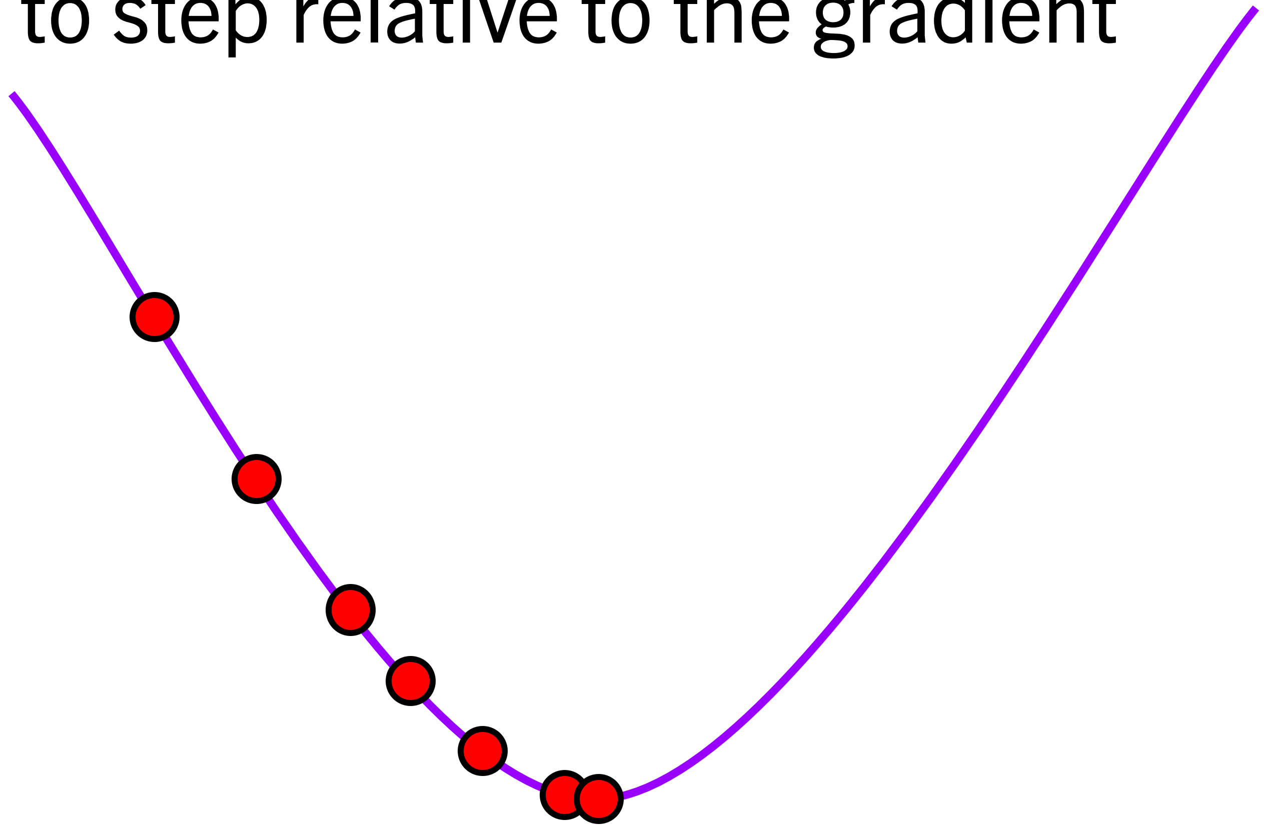


Gradient descent

Formally:

$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla L(\mathbf{w})$

Where η is *step size*, how far to step relative to the gradient



Optimization in action: Linear Programming

- As we've discussed, often optimizing a function can be extremely difficult
- Linear functions, even with constraints, are efficiently solvable (or can be approximated)
- If we can reformulate a problem to be described in a certain way, then it can be solved much more easily
- Note: the "programme" in linear programming is not the same as a computer program! It refers to planning.

What makes a linear programme

- Optimization problem consisting of
  - maximizing (or minimizing) a linear objective function
  - of n decision variables
  - subject to a set of constraints expressed by linear equations

  or inequalities.

- In linear programme: objective function + constraints are all linear
  Typically (not always): variables are non-negative

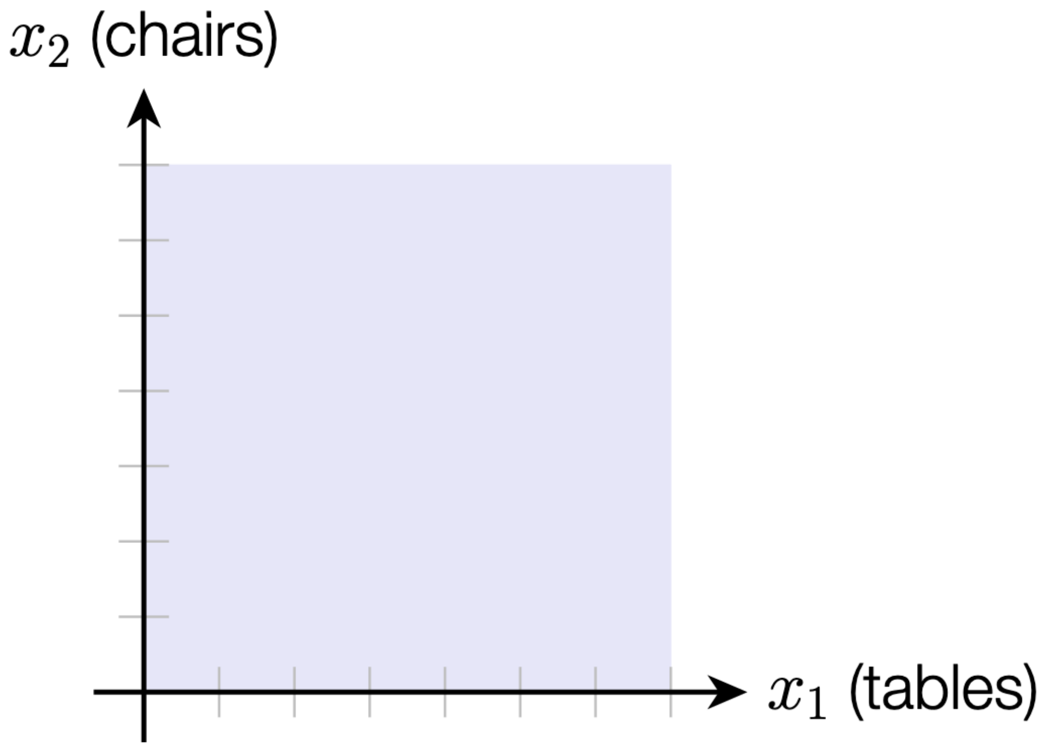
Working through an example

A large factory makes tables and chairs. Each table returns a profit of $200 and each chair a profit of $100.

Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood.

The factory has 6 units of metal and 9 units of wood.

How many tables and chairs should the factory make to maximize profit?

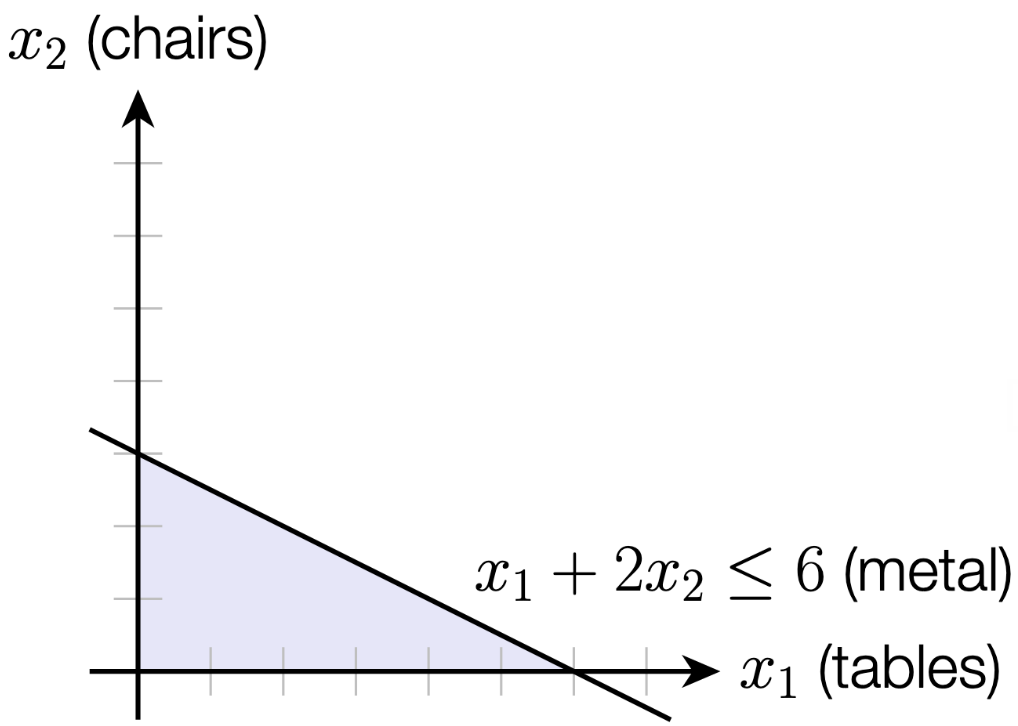


Working through an example

Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood.

The factory has 6 units of metal and 9 units of wood.

$$1x_1 + 2x_2 \leq 6$$

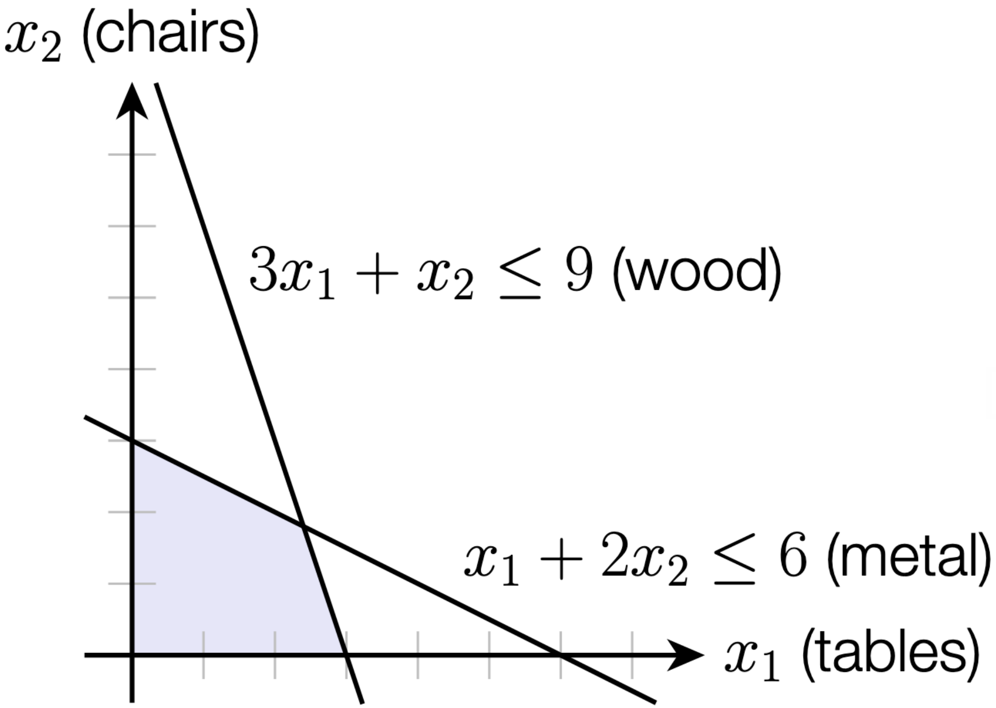


Working through an example

Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood.

The factory has 6 units of metal and 9 units of wood.

$$1x_1 + 2x_2 \leq 6$$
$$3x_1 + 1x_2 \leq 9$$

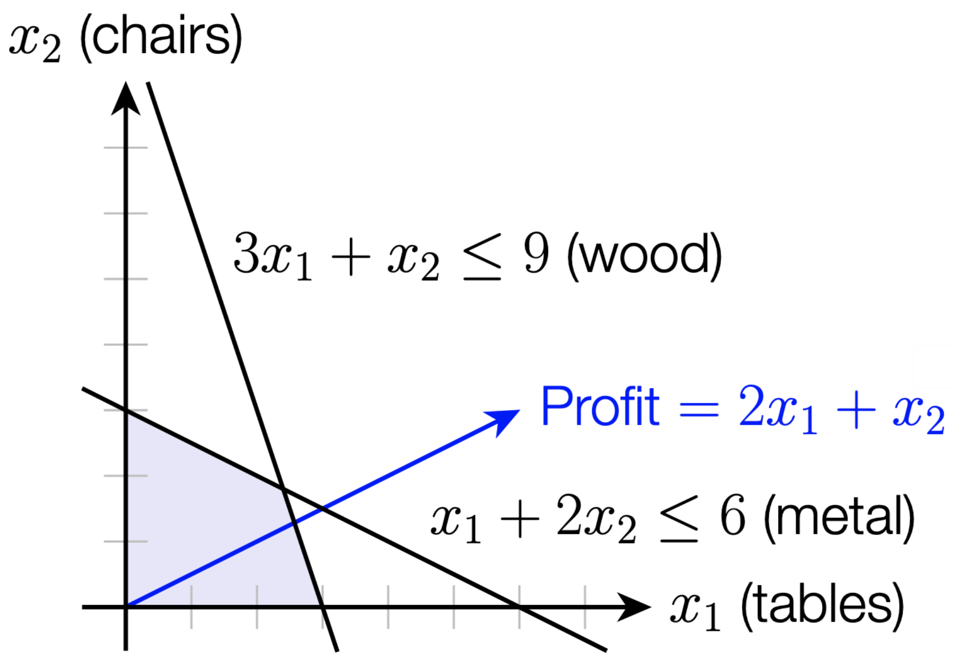


Working through an example

A large factory makes tables and chairs. Each table returns a profit of $200 and each chair a profit of $100.

How many tables and chairs should the factory make to maximize profit?

$$1x_1 + 2x_2 \leq 6$$
$$3x_1 + 1x_2 \leq 9$$
$$Profit = 2x_1 + 1x_2$$

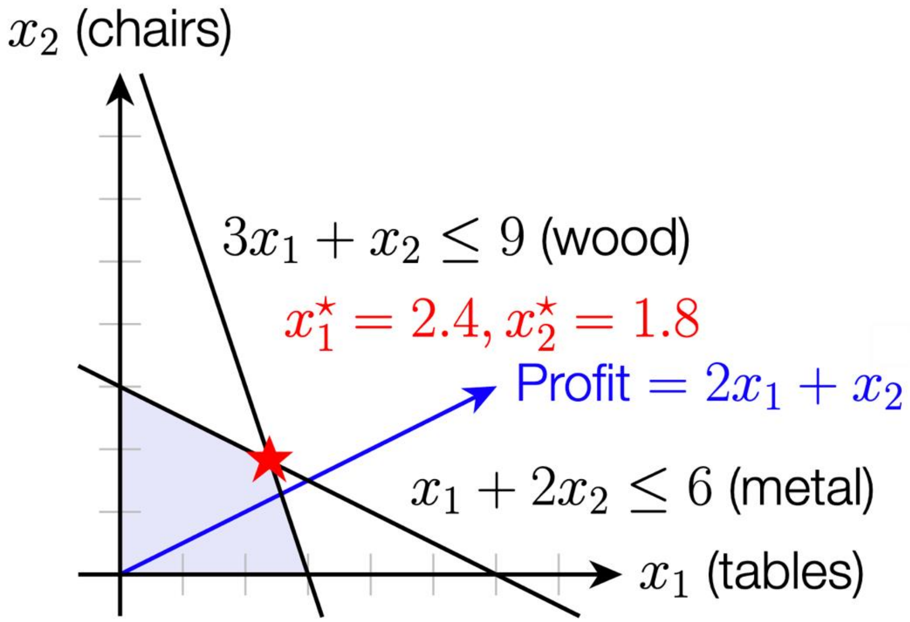


Working through an example

$$\underset{x_1, x_2}{\text{maximize}} \quad 2x_1 + x_2$$

$$\begin{aligned} \text{subject to} \quad & x_1 + 2x_2 \leq 6 \\ & 3x_1 + x_2 \leq 9 \\ & x_1, x_2 \geq 0 \end{aligned}$$

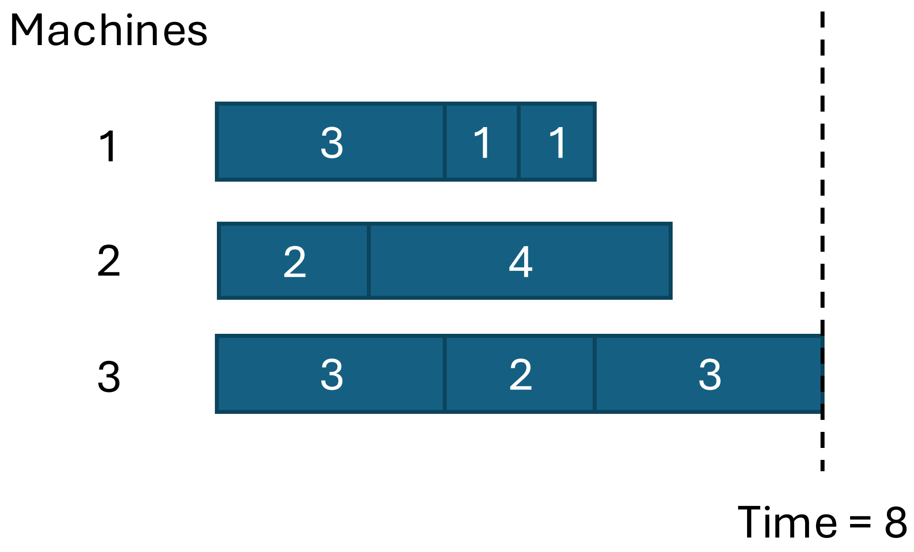
$$1x_1 + 2x_2 \leq 6$$
$$3x_1 + 1x_2 \leq 9$$
$$Profit = 2x_1 + 1x_2$$



Lab Part 1

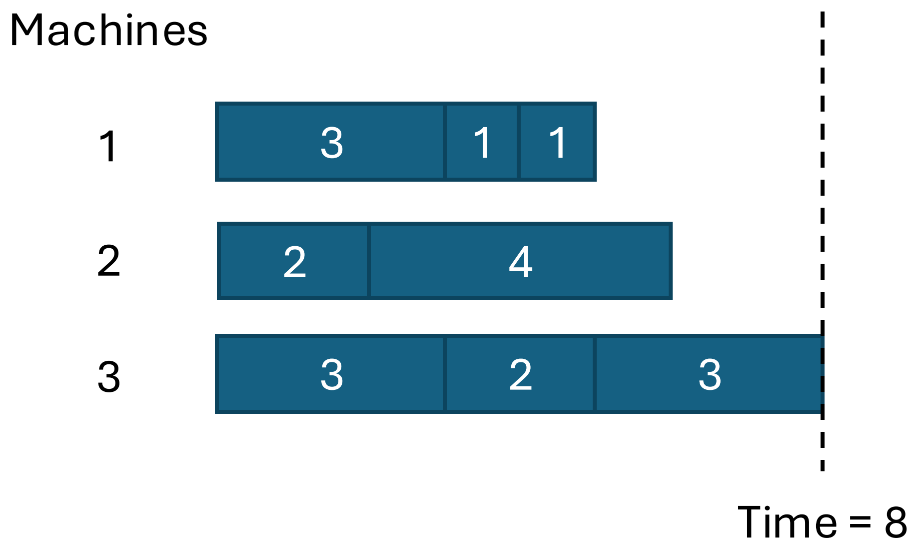
Example 2: Scheduling

- 3 machines must complete 8 tasks, each of which takes a varying amount of time.
- How do we assign the tasks so that the total time spent is as short as possible?



Example 2: Scheduling

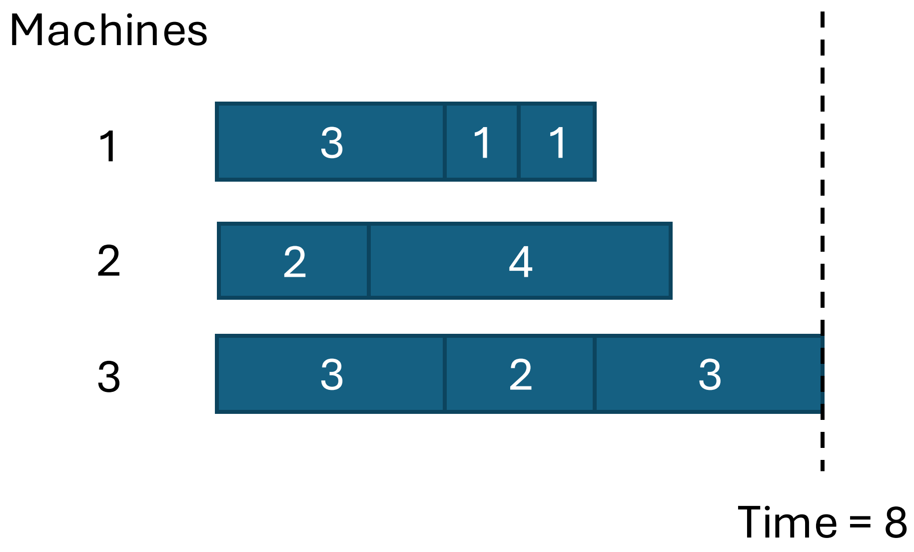
- 3 machines: $j_1 \ldots j_3$
- 8 tasks: $i_1 \ldots i_8$
- Each task i lasts t_i units of time



Example 2: Scheduling

- 3 machines: $j_1 \dots j_3$
- 8 tasks: $i_1 \dots i_8$
- Each task i lasts t_i units of time

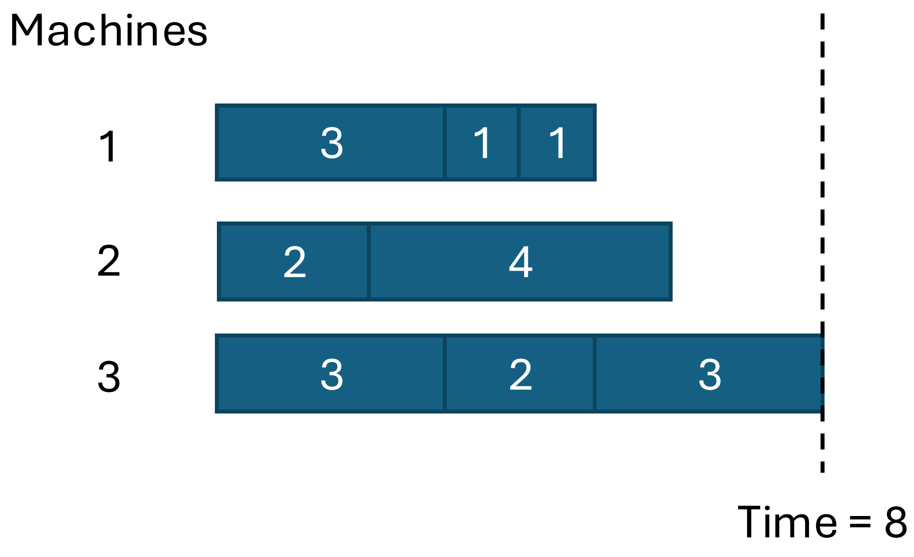
- The total amount of time needed by any one machine must be less than or equal to the total time needed
- Every task must be assigned exactly one time



Example 2: Scheduling

- 3 machines: $j_1 \ldots j_3$
- 8 tasks: $i_1 \ldots i_8$
- Each task i lasts t_i units of time

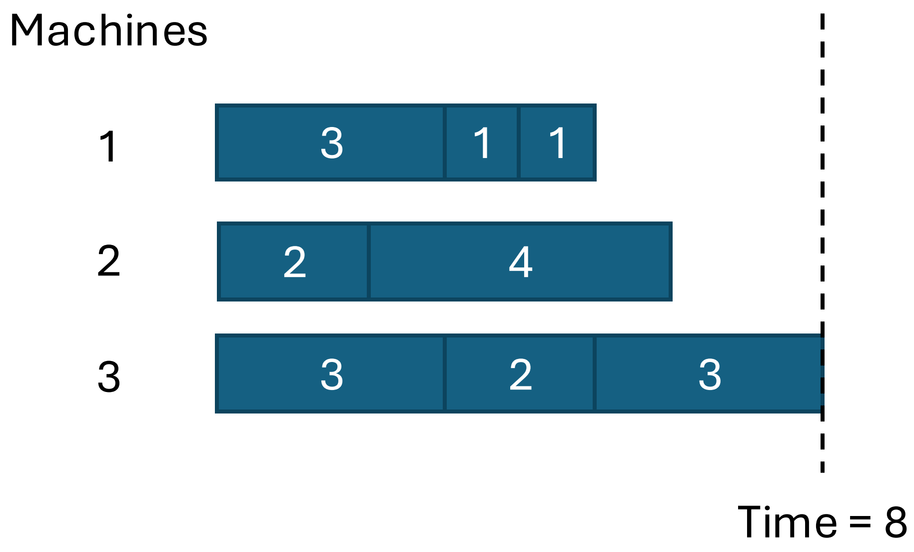
- The total amount of time needed must be less than or equal to the amount of time needed by one machine
- Every task must be assigned exactly one time
- $x_i^j = 1$ if task i is assigned to machine j



Example 2: Scheduling

- 3 machines: $j_1 \ldots j_3$
- 8 tasks: $i_1 \ldots i_8$
- Each task i lasts t_i units of time

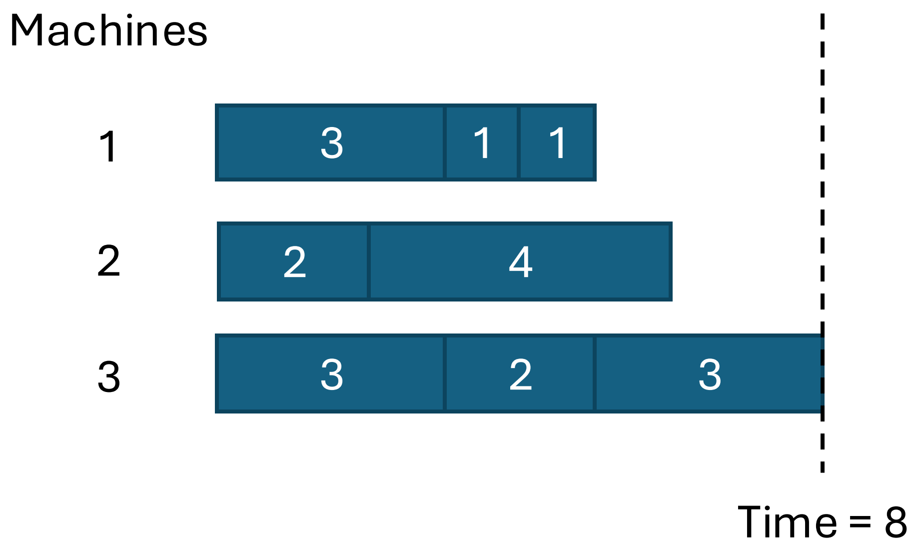
- The total amount of time needed must be less than or equal to the amount of time needed by one machine
- Every task must be assigned exactly one time
- $x_i^j = 1$ if task i is assigned to machine j



$\sum_i (t_i * x_i^j) \leq t_{total}$ (For each machine j)

Example 2: Scheduling

- 3 machines: $j_1 \dots j_3$
- 8 tasks: $i_1 \dots i_8$
- Each task i lasts t_i units of time



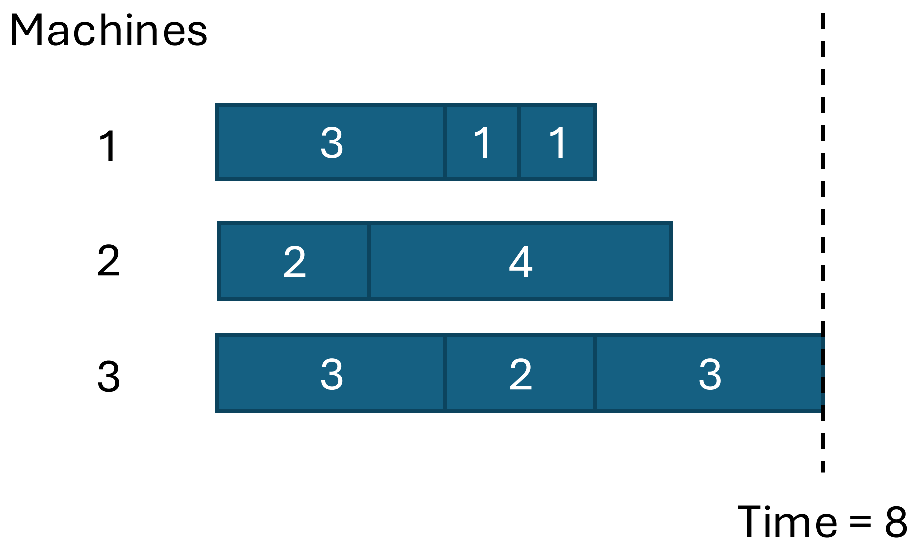
- The total amount of time needed must be less than or equal to the amount of time needed by one machine
- Every task must be assigned exactly one time
- $x_i^j = 1$ if task i is assigned to machine j

$\sum_i (t_i * x_i^j) \leq t_{total}$ (For each machine j)

$\sum_j x_i^j = 1$ (For each task i)

Example 2: Scheduling

- 3 machines: $j_1 \dots j_3$
- 8 tasks: $i_1 \dots i_8$
- Each task i lasts t_i units of time



- The total amount of time needed must be less than or equal to the amount of time needed by one machine
- Every task must be assigned exactly one time
- $x_i^j = 1$ if task i is assigned to machine j

$\sum_i (t_i * x_i^j) \leq t_{total}$ (For each machine j)

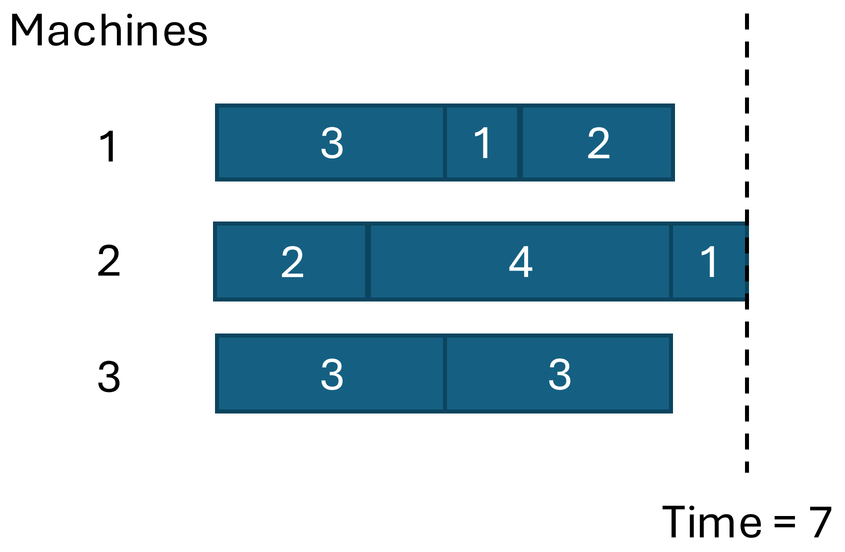
$\sum_j x_i^j = 1$ (For each task i)

Minimize t_{total}!

Example 2: Scheduling

- 3 machines: $j_1 \ldots j_3$
- 8 tasks: $i_1 \ldots i_8$
- Each task i lasts t_i units of time

- The total amount of time needed must be less than or equal to the amount of time needed by one machine
- Every task must be assigned exactly one time
- $x_i^j = 1$ if task i is assigned to machine j



$\sum_i (t_i * x_i^j) \leq t_{total}$ (For each machine j)

$\sum_j x_i^j = 1$ (For each task i)

Minimize t_{total}!

Example 2: Scheduling

- Capable of handling even more complex requirements
- e.g. preferred jobs to certain machines, or jobs that must be completed before others can begin