Optimization for Robotics Summer School

Lecture 1 & 2: Introduction to Optimization

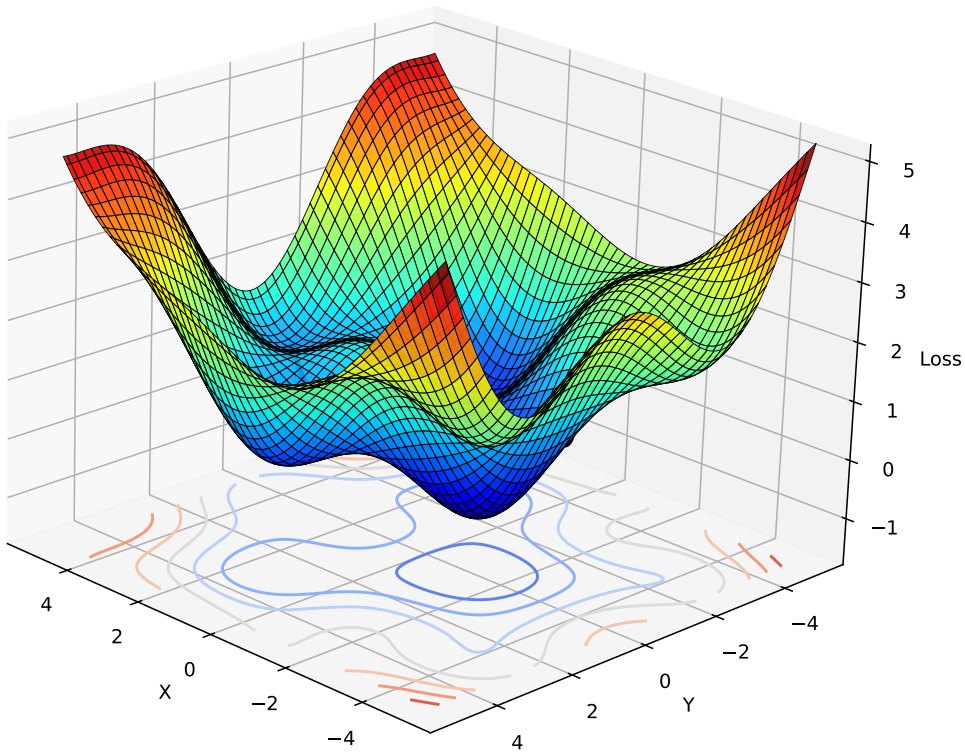
Konstantinos Chatzilygeroudis - costashatz@upatras.gr

Department of Electrical and Computer Engineering
University of Patras

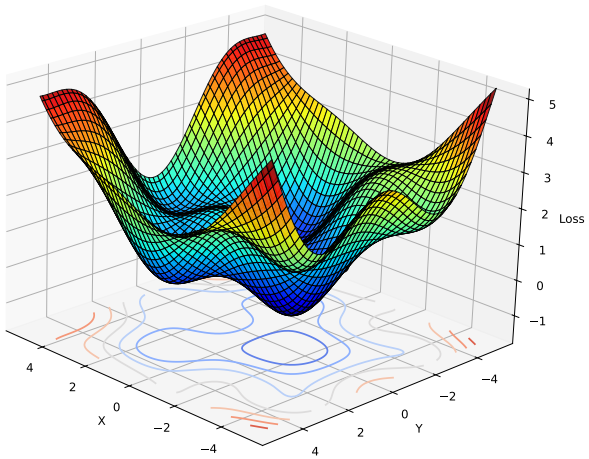
Template made by Panagiotis Papagiannopoulos

What is Optimization?

Optimization is the process of making something as effective or functional as possible.



What is Optimization?

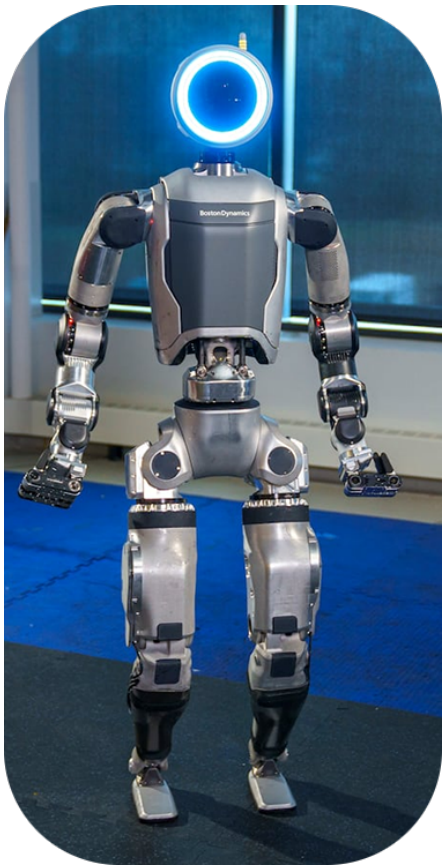


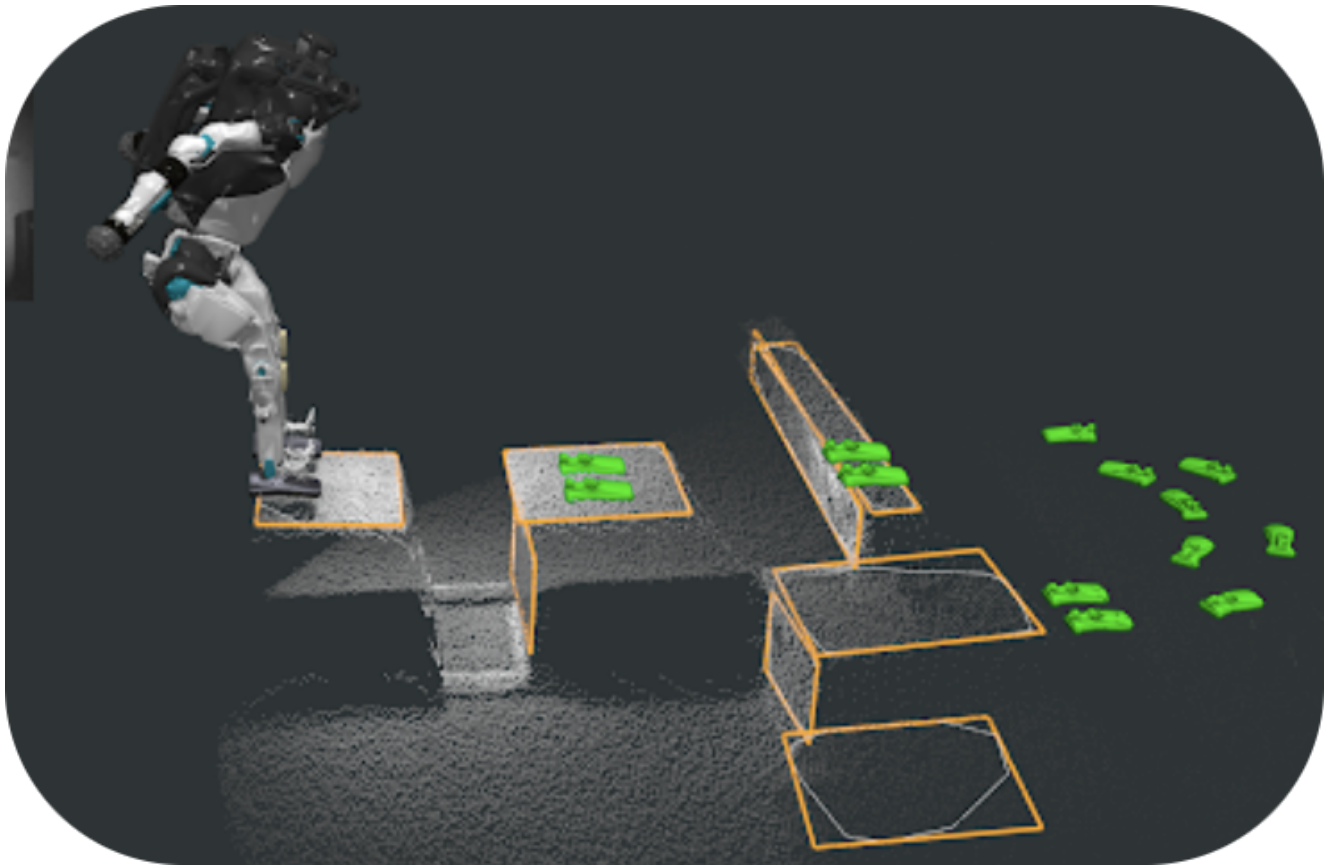
What is the goal of optimization?

- To find the best solution to a problem, which involves minimizing or maximizing a (set of) function(s) (**the objective function(s)**).

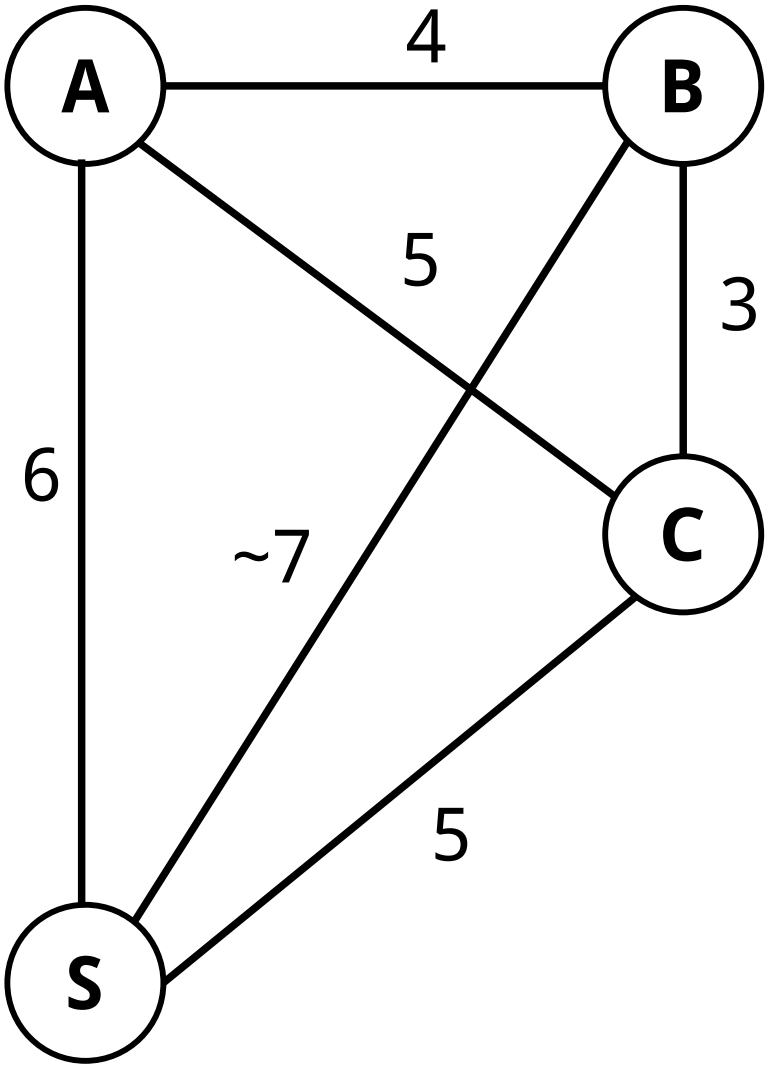
- To find the values of the *unknowns* (**the variables**) that lead to the optimal outcome, subject to certain **constraints**.

Why?



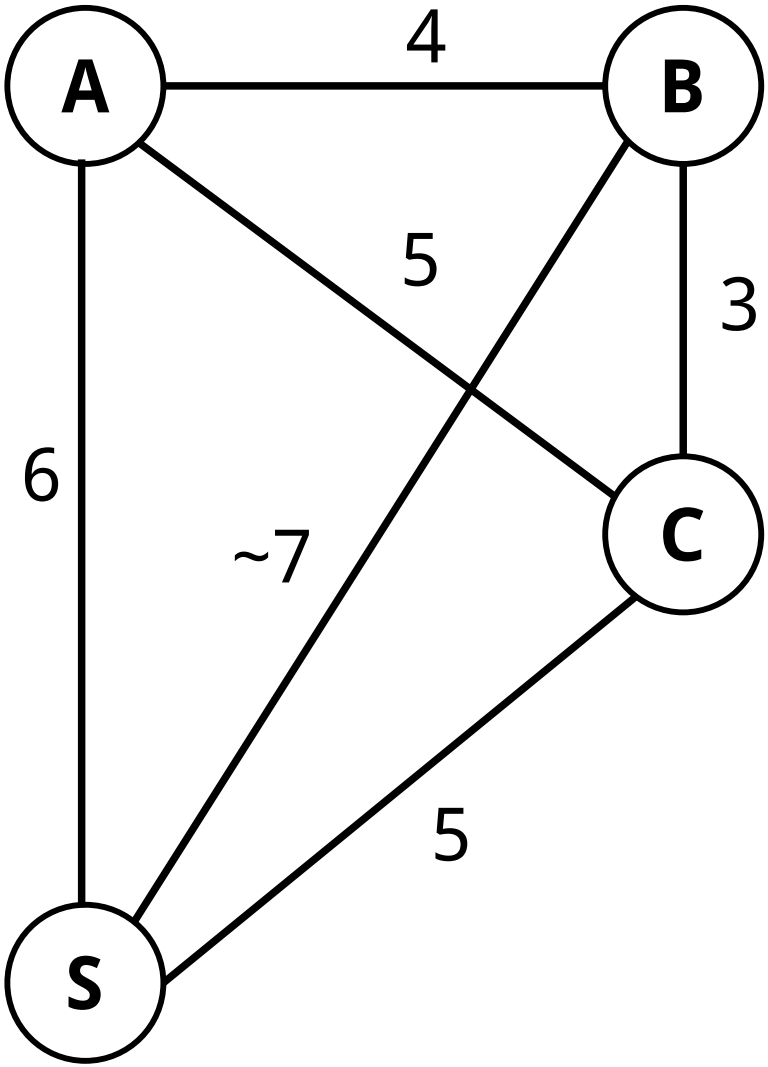


Discrete Optimization Example



Example: A delivery driver needs to drop off packages at 3 different houses, and they want to minimize total travel distance.

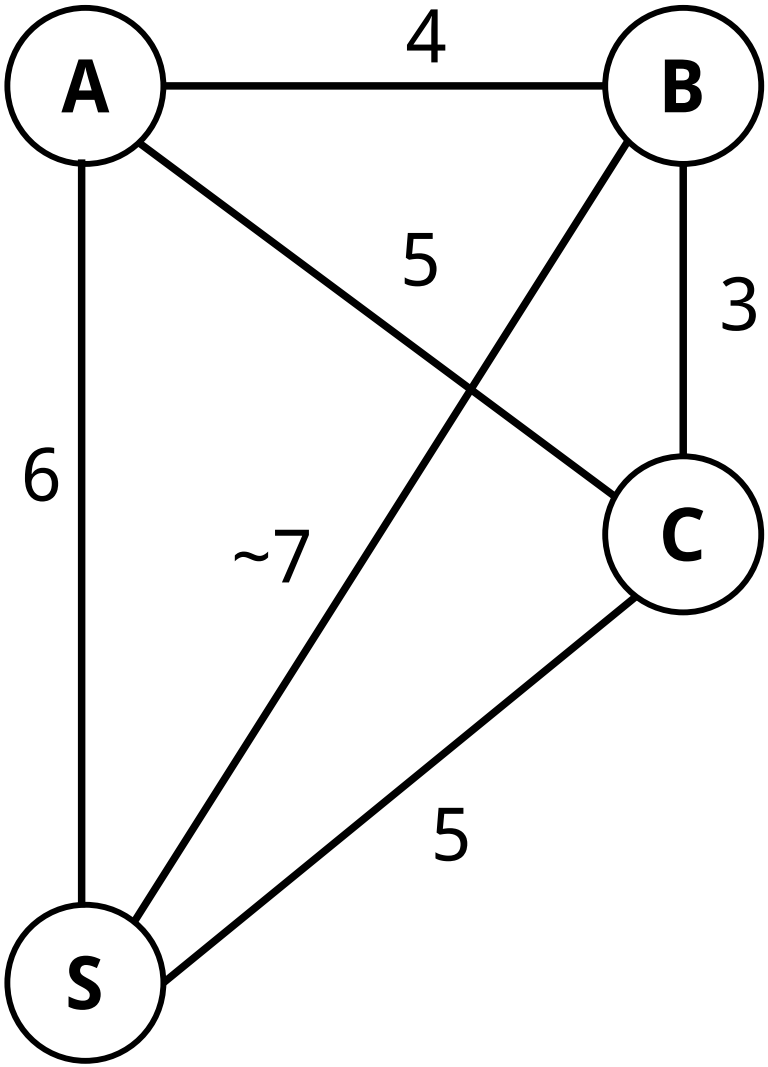
Discrete Optimization Example



Example: A delivery driver needs to drop off packages at 3 different houses, and they want to minimize total travel distance.

- The objective function is the total travel distance.
- We have a set of possible solutions. Can we enumerate them?

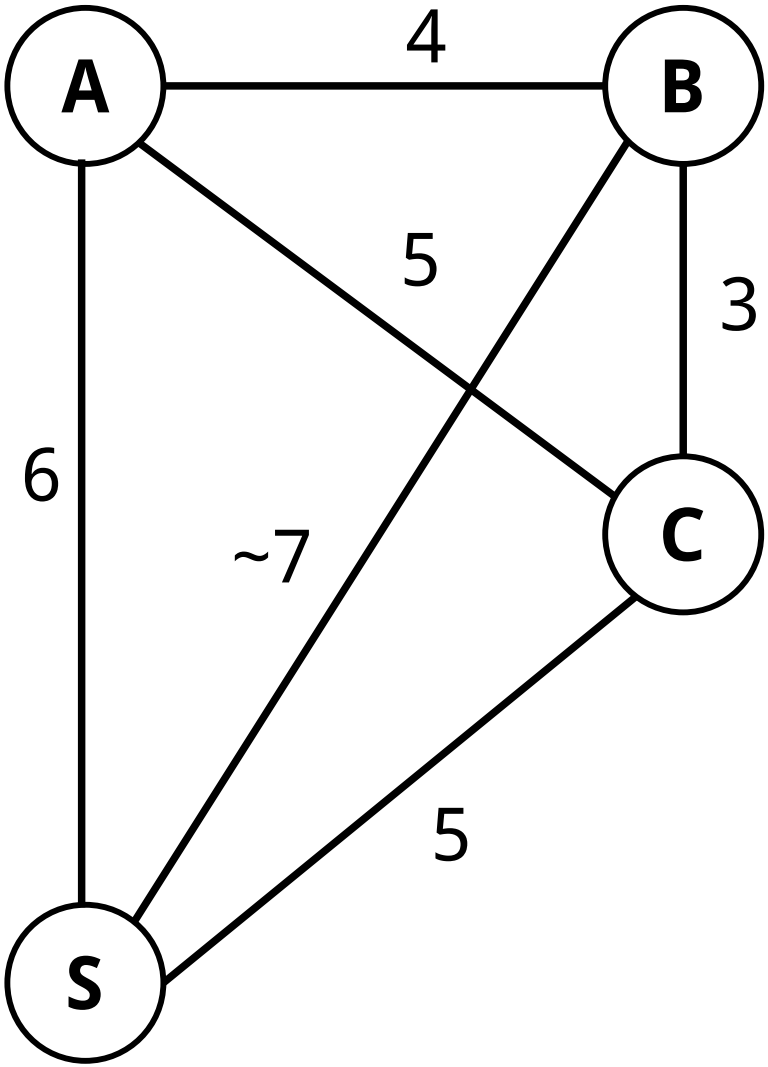
Discrete Optimization Example



Example: A delivery driver needs to drop off packages at 3 different houses, and they want to minimize total travel distance.

- The objective function is the total travel distance.
- We have a set of possible solutions. Can we enumerate them? **Yes!**

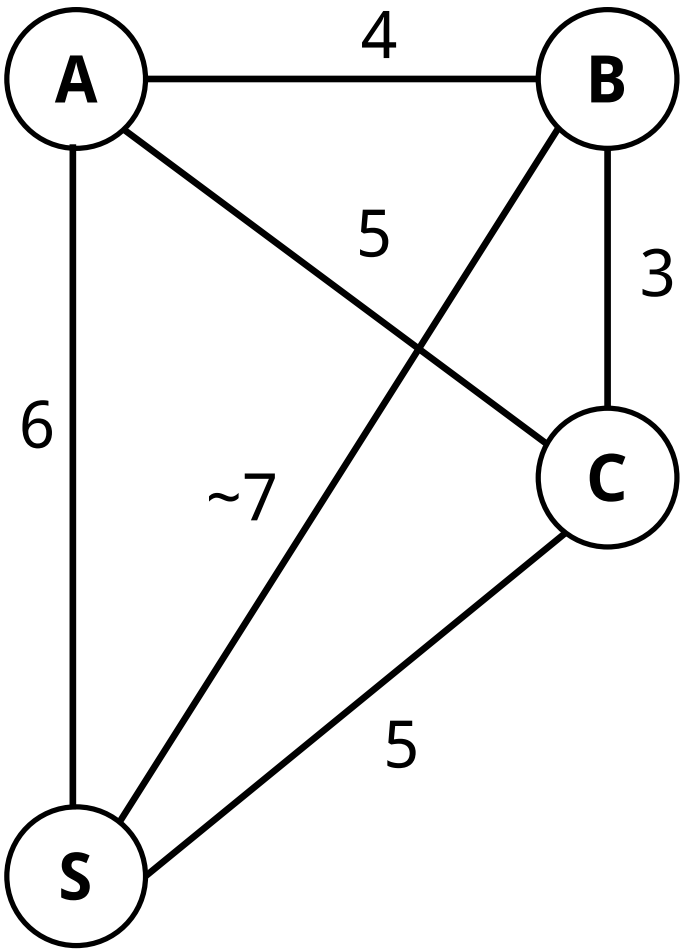
Discrete Optimization Example



Example: A delivery driver needs to drop off packages at 3 different houses, and they want to minimize total travel distance.

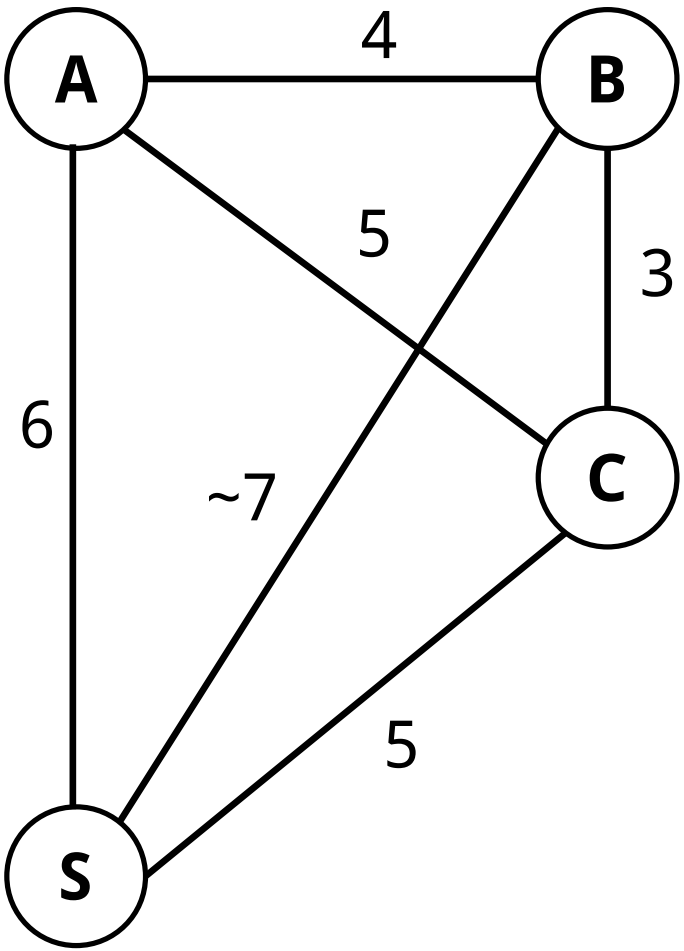
- The objective function is the total travel distance.
- We have a set of possible solutions. Can we enumerate them? **Yes!**
- What is the optimal solution?

Discrete Optimization Example (2)



Let's enumerate all possible solutions:

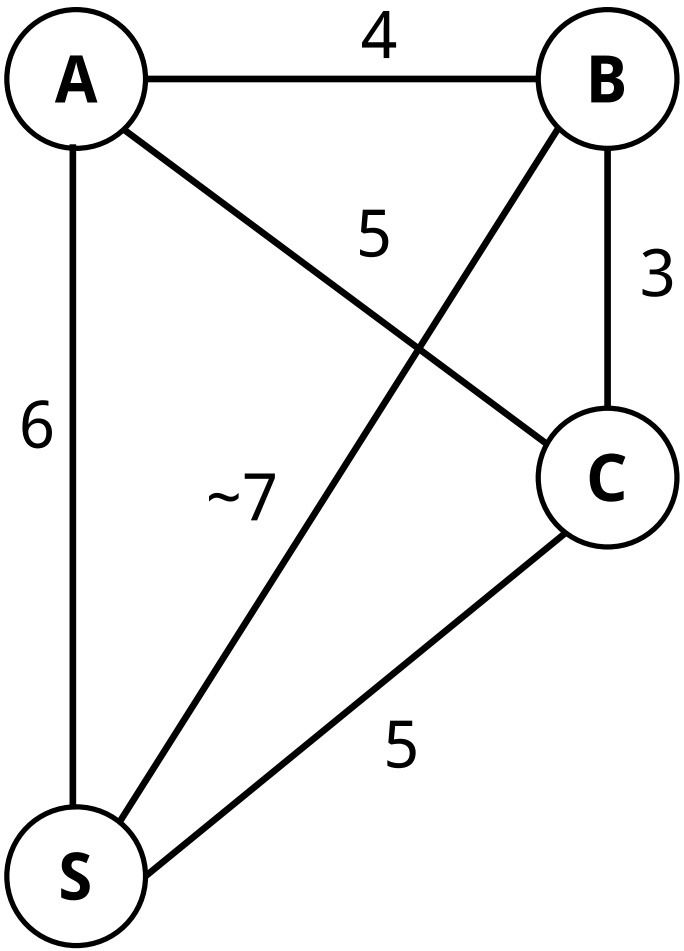
Discrete Optimization Example (2)



Let's enumerate all possible solutions:

1. $S \rightarrow A \rightarrow B \rightarrow C \rightarrow S$ (distance: 18)

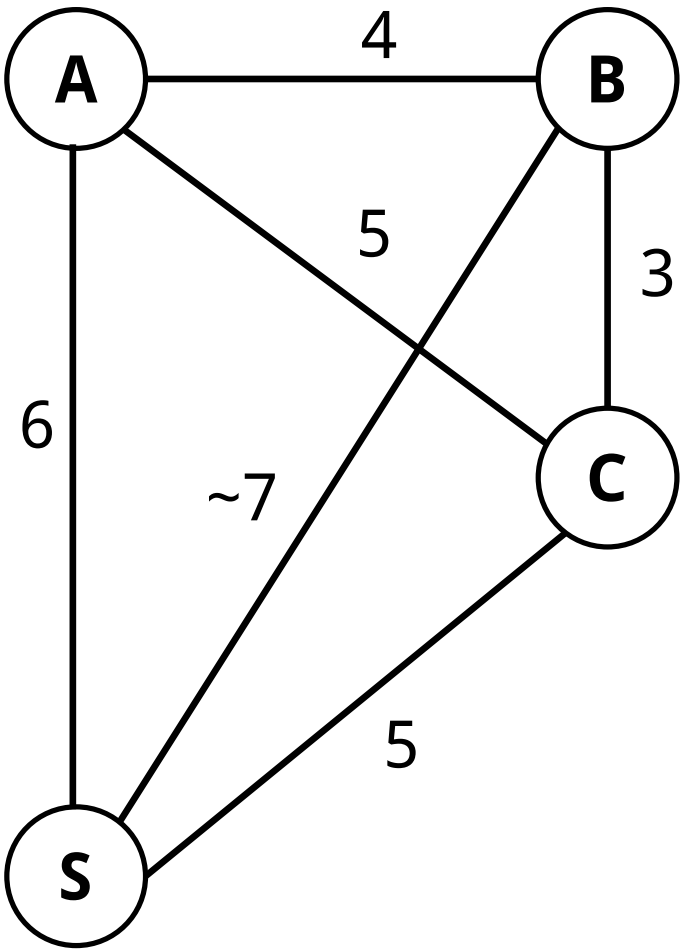
Discrete Optimization Example (2)



Let's enumerate all possible solutions:

1. $S \rightarrow A \rightarrow B \rightarrow C \rightarrow S$ (distance: 18)
2. $S \rightarrow A \rightarrow C \rightarrow B \rightarrow S$ (distance: 21)

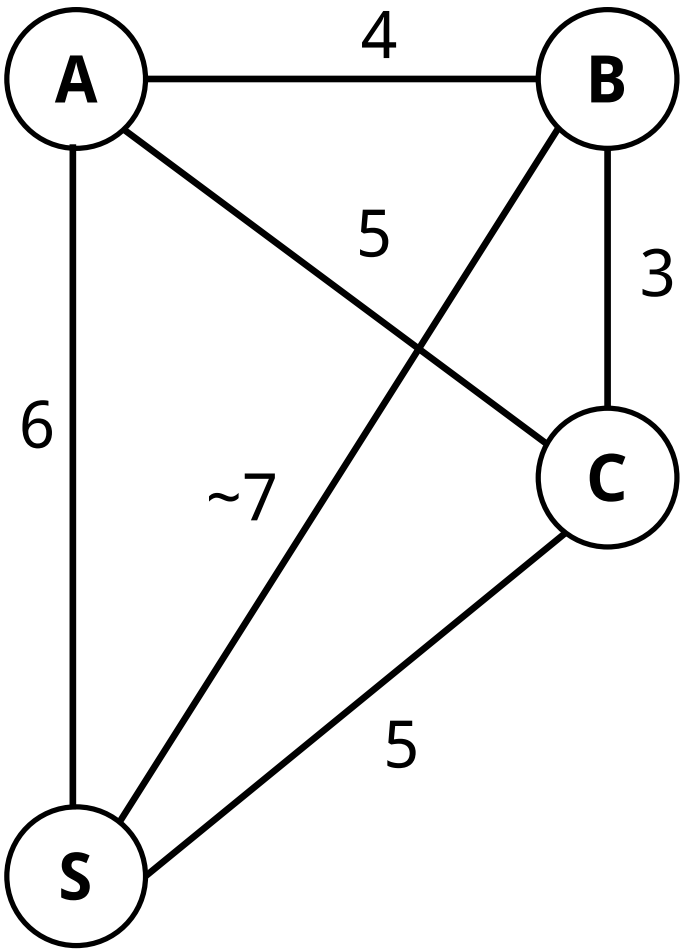
Discrete Optimization Example (2)



Let's enumerate all possible solutions:

1. $S \rightarrow A \rightarrow B \rightarrow C \rightarrow S$ (distance: 18)
2. $S \rightarrow A \rightarrow C \rightarrow B \rightarrow S$ (distance: 21)
3. $S \rightarrow B \rightarrow A \rightarrow C \rightarrow S$ (distance: 21)
4. $S \rightarrow B \rightarrow C \rightarrow A \rightarrow S$ (distance: 21)
5. $S \rightarrow C \rightarrow A \rightarrow B \rightarrow S$ (distance: 21)
6. $S \rightarrow C \rightarrow B \rightarrow A \rightarrow S$ (distance: 18)

Discrete Optimization Example (2)

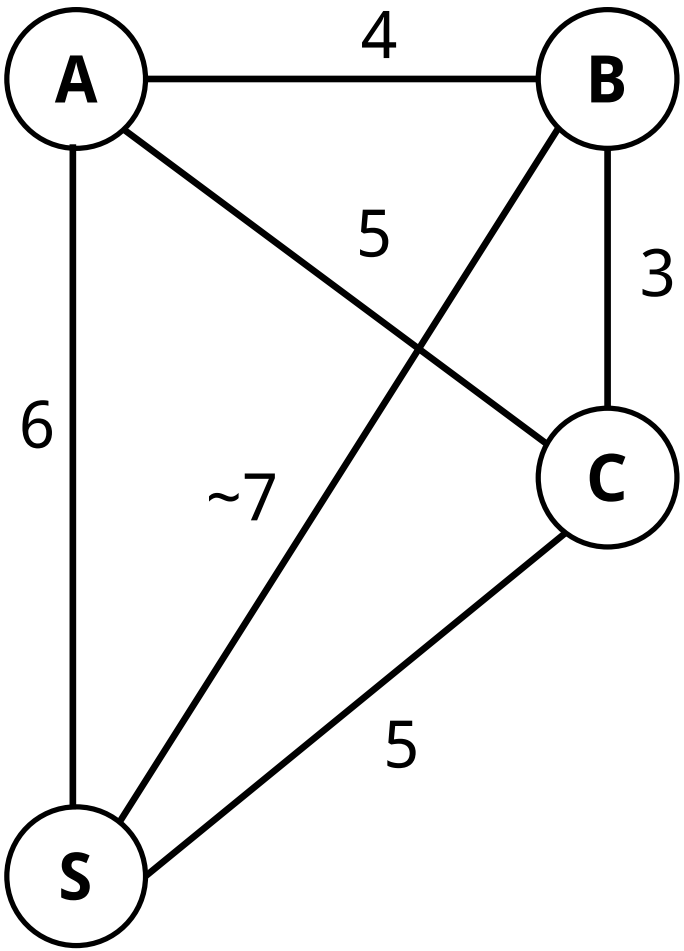


Let's enumerate all possible solutions:

1. $S \rightarrow A \rightarrow B \rightarrow C \rightarrow S$ (distance: 18)
2. $S \rightarrow A \rightarrow C \rightarrow B \rightarrow S$ (distance: 21)
3. $S \rightarrow B \rightarrow A \rightarrow C \rightarrow S$ (distance: 21)
4. $S \rightarrow B \rightarrow C \rightarrow A \rightarrow S$ (distance: 21)
5. $S \rightarrow C \rightarrow A \rightarrow B \rightarrow S$ (distance: 21)
6. $S \rightarrow C \rightarrow B \rightarrow A \rightarrow S$ (distance: 18)

The optimal solution is?

Discrete Optimization Example (2)

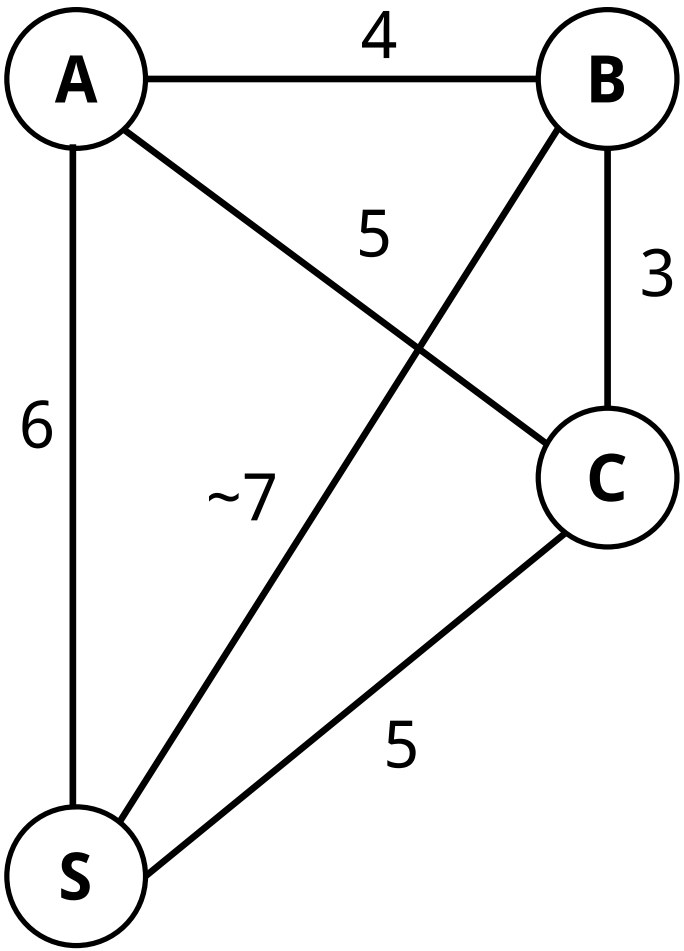


Let's enumerate all possible solutions:

1. $S \to A \to B \to C \to S$ (distance: 18)
2. $S \to A \to C \to B \to S$ (distance: 21)
3. $S \to B \to A \to C \to S$ (distance: 21)
4. $S \to B \to C \to A \to S$ (distance: 21)
5. $S \to C \to A \to B \to S$ (distance: 21)
6. $S \to C \to B \to A \to S$ (distance: 18)

The optimal solution is? We have two!

Discrete Optimization Example (2)



Let's enumerate all possible solutions:

1. $S \to A \to B \to C \to S$ (distance: 18)
2. $S \to A \to C \to B \to S$ (distance: 21)
3. $S \to B \to A \to C \to S$ (distance: 21)
4. $S \to B \to C \to A \to S$ (distance: 21)
5. $S \to C \to A \to B \to S$ (distance: 21)
6. $S \to C \to B \to A \to S$ (distance: 18)

The optimal solution is? We have two!

In general, we cannot enumerate all possible solutions! What can we do?

Some notation

$$\boldsymbol{y} = f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}^M,$$

$\boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{y} \in \mathbb{R}^M$.

What is $\frac{\partial f}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$?

Some notation

$$\boldsymbol{y} = f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}^M,$$

$\boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{y} \in \mathbb{R}^M$.

What is $\frac{\partial f}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$?

$$\frac{\partial f}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial \boldsymbol{y}_1}{\partial \boldsymbol{x}_1} & \frac{\partial \boldsymbol{y}_1}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial \boldsymbol{y}_1}{\partial \boldsymbol{x}_N} \\ \frac{\partial \boldsymbol{y}_2}{\partial \boldsymbol{x}_1} & \frac{\partial \boldsymbol{y}_2}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial \boldsymbol{y}_2}{\partial \boldsymbol{x}_N} \\ \ddots & \ddots & \ddots & \ddots \\ \frac{\partial \boldsymbol{y}_M}{\partial \boldsymbol{x}_1} & \frac{\partial \boldsymbol{y}_M}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial \boldsymbol{y}_M}{\partial \boldsymbol{x}_N} \end{bmatrix} \in \mathbb{R}^{M \times N}$$

This is called the **Jacobian**.

Notation Continued

- When $M = 1$, then $\frac{\partial f}{\partial \boldsymbol{x}} \in \mathbb{R}^{1 \times N}$ (row vector),
- When $M = 1$, we usually denote the **gradient** of f with $\nabla_{\boldsymbol{x}} f$; in other words, $\nabla_{\boldsymbol{x}} f = \frac{\partial f}{\partial \boldsymbol{x}}^T \in \mathbb{R}^{N \times 1}$ (column vector),
- When $M = 1$, the **Hessian** of f is:

$$\nabla^2_{\boldsymbol{xx}} f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_N} \\ \ddots & \ddots & \ddots & \ddots \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \frac{\partial^2 f}{\partial x_N \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix}$$

Taylor Series/Linearization

We want to compute the Taylor expansion of f around $\bar{\boldsymbol{x}}$:

$$f(\boldsymbol{x}) \approx f(\bar{\boldsymbol{x}}) + \frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\bar{\boldsymbol{x}}} (\boldsymbol{x} - \bar{\boldsymbol{x}}) + \dots$$

We can write: $\boldsymbol{x} = \bar{\boldsymbol{x}} + \Delta \boldsymbol{x}$, **and thus**:

$$f(\boldsymbol{x}) = f(\bar{\boldsymbol{x}} + \Delta \boldsymbol{x}) \approx f(\bar{\boldsymbol{x}}) + \frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\bar{\boldsymbol{x}}} \Delta \boldsymbol{x} + \dots$$

Unconstrained Continuous Optimization

Unconstrained Optimization (minimization!):

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

- $\boldsymbol{x} \in \mathbb{R}^N$

Unconstrained Continuous Optimization

Unconstrained Optimization (minimization!):

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

- $\boldsymbol{x} \in \mathbb{R}^N$: **variables**
- $f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}$

Unconstrained Continuous Optimization

Unconstrained Optimization (minimization!):

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

- $\boldsymbol{x} \in \mathbb{R}^N$: **variables**
- $f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}$: **objective function**
- $\boldsymbol{x}_*$ is the (global) **optimal solution**

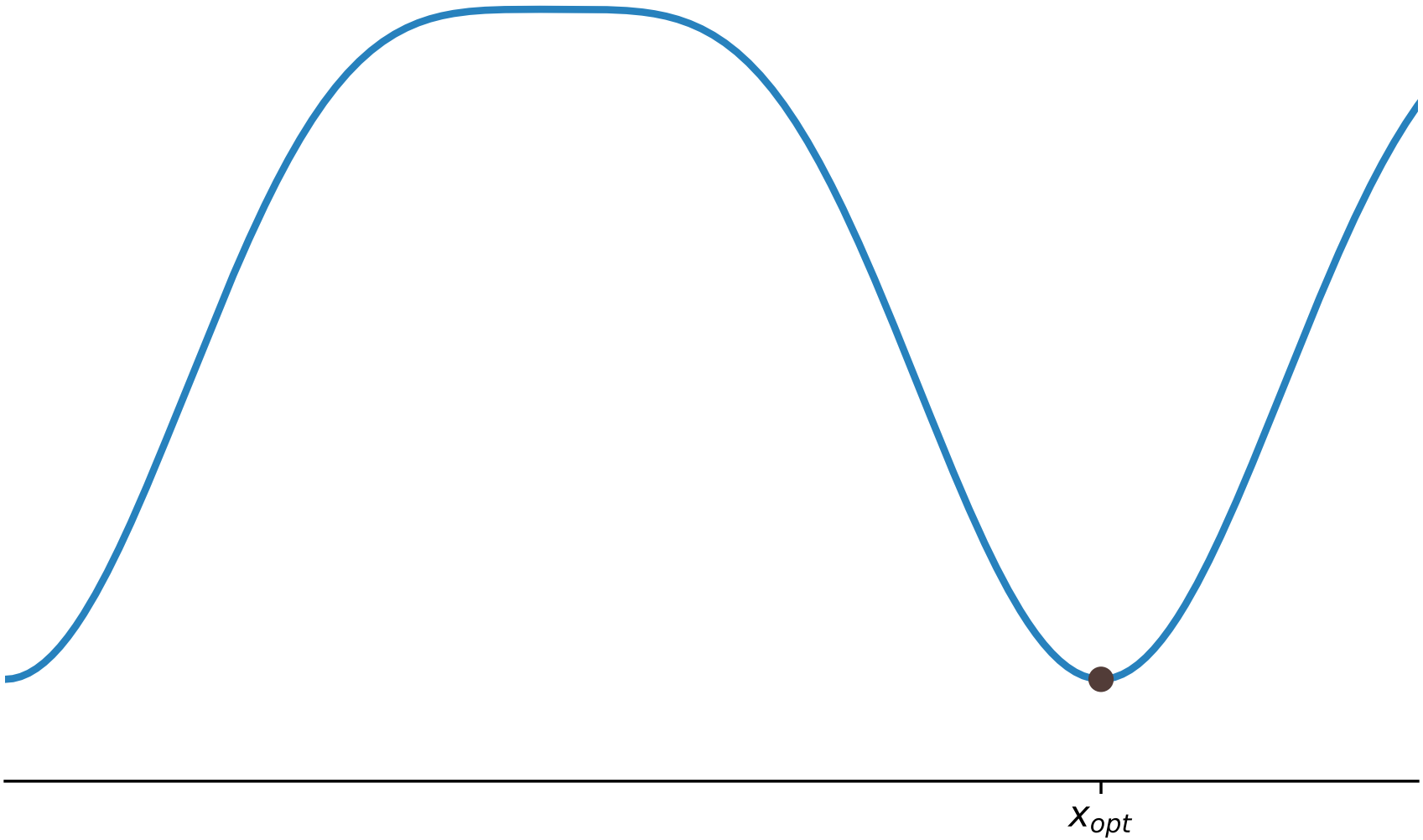
Unconstrained Continuous Optimization

Unconstrained Optimization (minimization!):

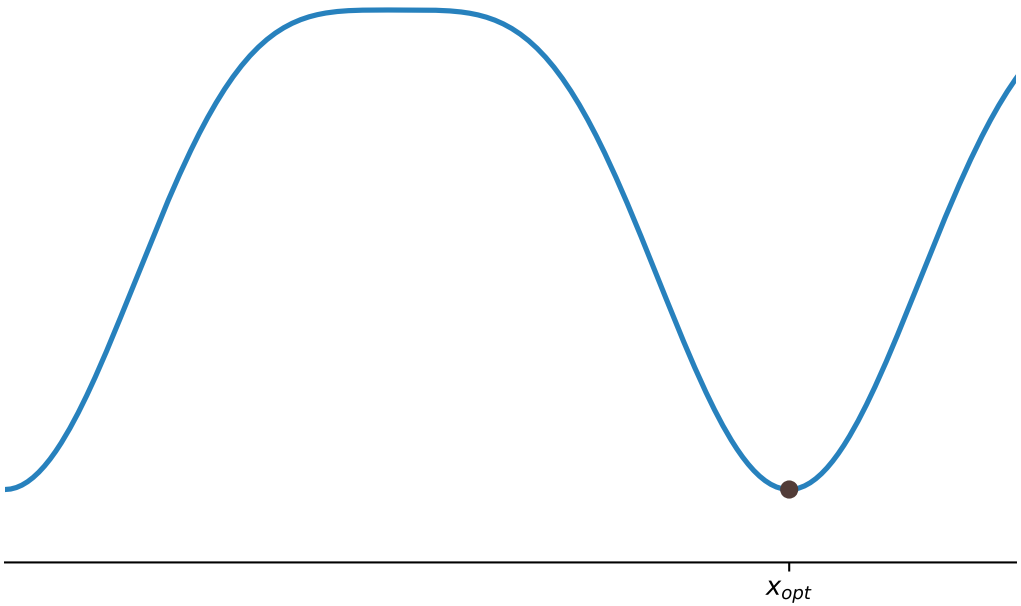
$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

- $\boldsymbol{x} \in \mathbb{R}^N$: **variables**
- $f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}$: **objective function**
- $\boldsymbol{x}_*$ is the (global) **optimal solution**, if $f(\boldsymbol{x}_*) \leqslant f(\boldsymbol{x}), \forall \boldsymbol{x}$
- $\boldsymbol{x}_*$ is a **local optimum**, if $f(\boldsymbol{x}_*) \leqslant f(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathcal{N}_{\boldsymbol{x}_*}$ (neighborhood of $\boldsymbol{x}_*$)

How does a solution look like?



How does a solution look like? (2)



Necessary Condition for a (local) solution (when f is smooth):

$$\left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_*} = 0 \Leftrightarrow \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_*) = 0$$

How does a solution look like? (3)

First Order Necessary Condition for a (local) minimum:

$$\left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_*} = 0 \Leftrightarrow \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_*) = 0$$

How does a solution look like? (3)

First Order Necessary Condition for a (local) minimum:

$$\left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_*} = 0 \Leftrightarrow \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_*) = 0$$

Second Order Necessary Condition for a (local) minimum:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_*) = 0$$

$$\nabla^2_{\boldsymbol{x}\boldsymbol{x}} f(\boldsymbol{x}_*) \text{ is psd}$$

Necessary and Sufficient Conditions

- **Necessary conditions** $\rightarrow$ if $\boldsymbol{x}_*$ is a (local) minimum, then these conditions hold;

Necessary and Sufficient Conditions

- **Necessary conditions** $\rightarrow$ if $\boldsymbol{x}_*$ is a (local) minimum, then these conditions hold;
- **Sufficient conditions**

Necessary and Sufficient Conditions

- **Necessary conditions** $\rightarrow$ if $\boldsymbol{x}_*$ is a (local) minimum, then these conditions hold;
- **Sufficient conditions** $\rightarrow$ if these conditions hold for $\boldsymbol{x}_\dagger$, then $\boldsymbol{x}_\dagger$ is a (local) minimum.

Enough of this! Let's see some action!

Problems related to optimization or maybe **useful to optimization**

Enough of this! Let's see some action!

Problems related to optimization or maybe **useful to optimization**

- **Root Finding:** Find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) = 0$,

Enough of this! Let's see some action!

Problems related to optimization or maybe **useful to optimization**

- **Root Finding:** Find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) = 0$,
- **Fixed Point:** Find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) = \boldsymbol{x}^*$,

Enough of this! Let's see some action!

Problems related to optimization or maybe **useful to optimization**

- **Root Finding:** Find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) = 0$,
- **Fixed Point:** Find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) = \boldsymbol{x}^*$,
- Many ways of solving this:
  - Fixed-Point Iteration
  - ...
  - Newton's Method

Newton's Method

- Linearize around current estimate, $\boldsymbol{x}_k$:

$$f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) \approx f(\boldsymbol{x}_k) + \frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k} \Delta\boldsymbol{x}$$

- We then set $f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) = 0$ and solve for $\Delta\boldsymbol{x}$:

$$f(\boldsymbol{x}_k) + \frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k} \Delta\boldsymbol{x} = 0$$

$$\Delta\boldsymbol{x} = -\left(\frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k}\right)^{-1} f(\boldsymbol{x}_k)$$

- We apply the correction $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}$

- Repeat until convergence

And so?

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}$$

Necessary Condition for solution (when f is smooth):

$$\frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}^*} = 0$$

And so?

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}$$

Necessary Condition for solution (when f is smooth):

$$\frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}^*} = 0$$

Idea: We can frame optimization as root finding in the gradient!

Optimization via Root Finding

Newton's Method for Optimization:

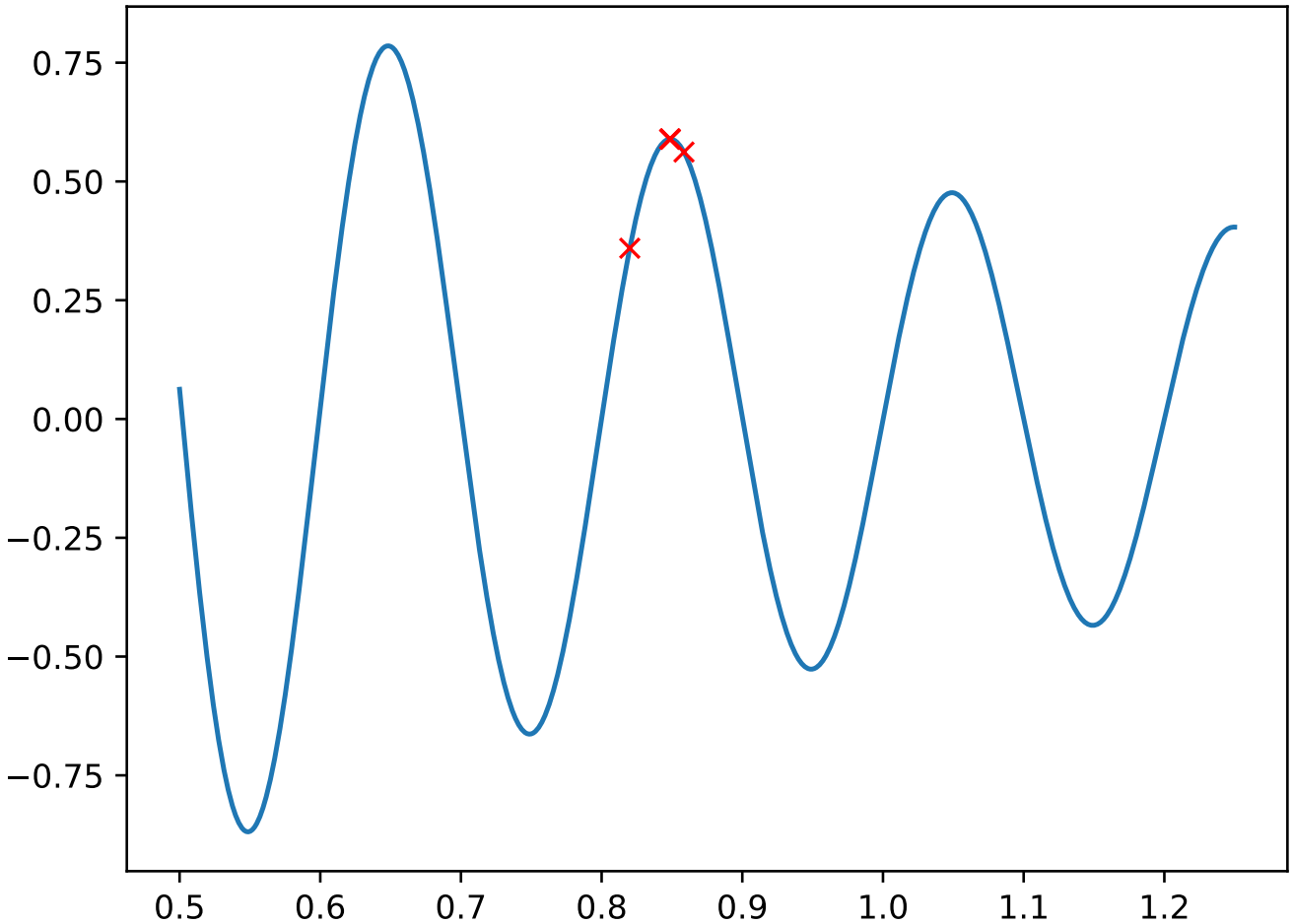
$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) \approx \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) + \frac{\partial}{\partial \boldsymbol{x}} \Big(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) \Big) \Delta\boldsymbol{x} = 0$$

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) \approx \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) + \nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}_k) \Delta\boldsymbol{x} = 0$$

$$\Delta\boldsymbol{x} = -\Big(\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}_k) \Big)^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}$$

Optimization via Root Finding - Code Example



Optimization via Root Finding (2)

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}$$

Necessary Condition for solution (when f is smooth):

$$\frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}^*} = 0$$

Not sufficient! Let's have a look at the 1D case:

Optimization via Root Finding (2)

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}$$

Necessary Condition for solution (when f is smooth):

$$\frac{\partial f}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}*} = 0$$

Not sufficient! Let's have a look at the 1D case:

$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{“descent”}} \underbrace{\left(\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}_k)\right)^{-1}}_{\text{“learning rate”}} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{“gradient”}}$$

Optimization via Root Finding (3)

Let's have a look at the 1D case:

$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{“descent”}} \underbrace{\left(\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}_k)\right)^{-1}}_{\text{“learning rate”}} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{“gradient”}}$$

Optimization via Root Finding (3)

Let's have a look at the 1D case:

$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{“descent”}} \underbrace{\left(\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}_k)\right)^{-1}}_{\text{“learning rate”}} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{“gradient”}}$$

- If $\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}) > 0$, then **“descent”**,
- If $\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}) < 0$, then **“ascent”**!

Optimization via Root Finding (3)

Let's have a look at the 1D case:

$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{“descent”}} \underbrace{\left(\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}_k)\right)^{-1}}_{\text{“learning rate”}} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{“gradient”}}$$

- If $\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}) > 0$, then **“descent”**,
- If $\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}) < 0$, then **“ascent”**!

In $\mathbb{R}^N$:

- If $\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}) \succ 0$ (positive definite), **“descent”**,
- If $\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}) \prec 0$ (negative definite), **“ascent”**!

"Damped" Newton's Method

So, how do we solve this?

"Damped" Newton's Method

So, how do we solve this? Many was! Simplest yet effective is to add regularization or "damping":

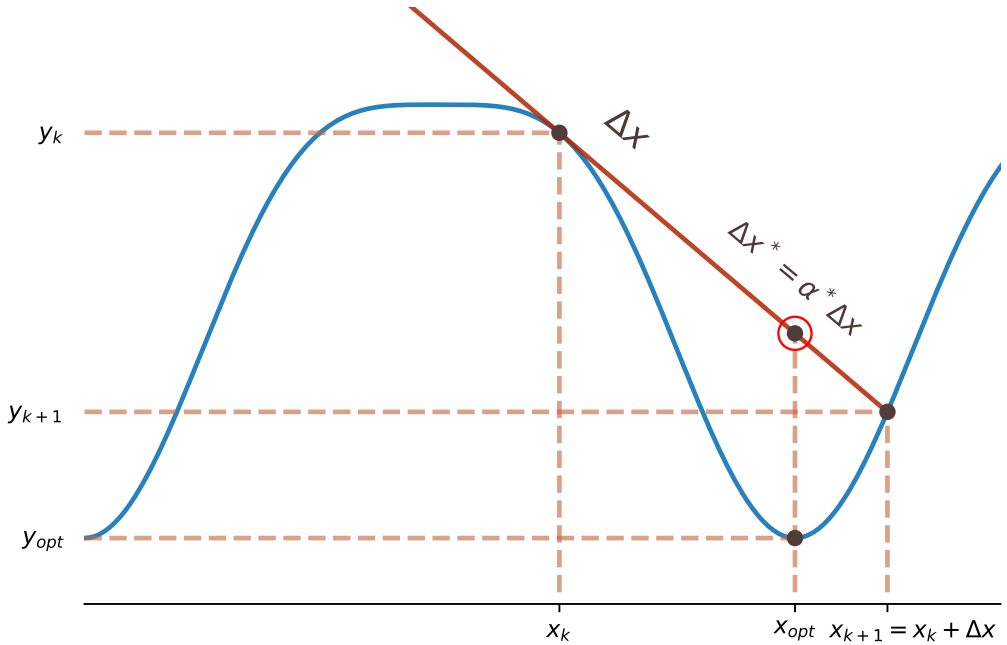
$$\boldsymbol{H} = \nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}_k)$$
$$\text{while } \boldsymbol{H} \preceq 0 :$$
$$\quad \boldsymbol{H} = \boldsymbol{H} + \beta \boldsymbol{I}$$
$$\Delta \boldsymbol{x} = -\boldsymbol{H}^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}$$

Backtracking Line Search

- Often $\Delta \boldsymbol{x}$ overshoots the real optimum. Why?

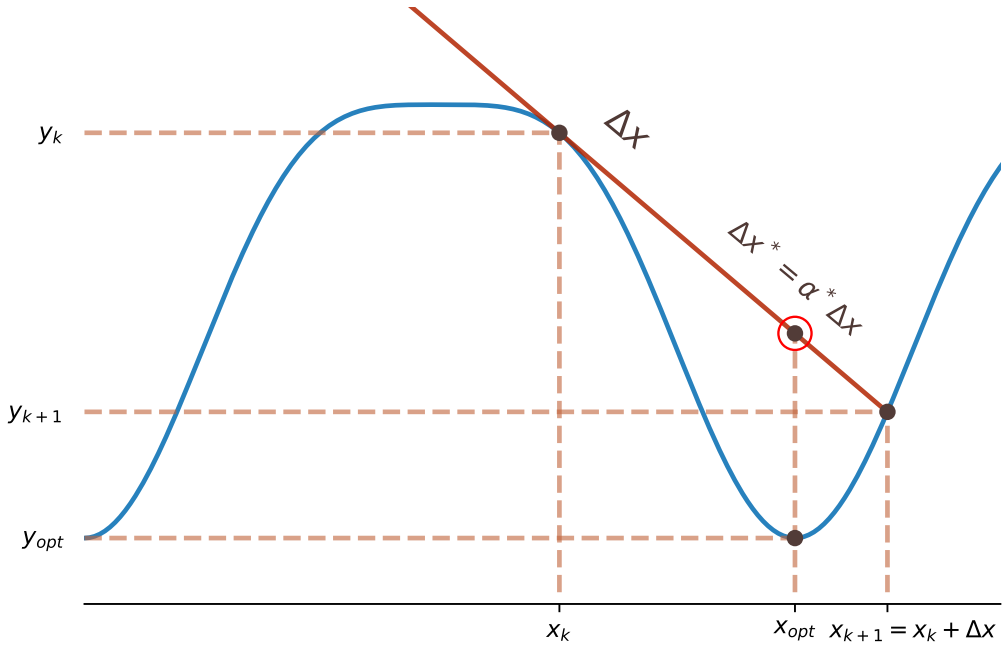
Backtracking Line Search

- Often $\Delta \boldsymbol{x}$ overshoots the real optimum. Why?
  - We are linearizing and taking a step on this line!!



Backtracking Line Search

- Often $\Delta \boldsymbol{x}$ overshoots the real optimum. Why?
  - We are linearizing and taking a step on this line!!



- To fix this issue:
  - We check if $f(\boldsymbol{x}_k + \Delta \boldsymbol{x})$ is good
  - If not, we "backtrack"; aka, we decrease $\Delta \boldsymbol{x}$
  - Repeat until we have a nice reduction

Line Search - Armijo Rule

Armijo Rule:

$$\begin{aligned} \alpha &= 1 \\ \text{while } & f(\boldsymbol{x}_k + \alpha \Delta \boldsymbol{x}) > f(\boldsymbol{x}_k) + b\alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)^T \Delta \boldsymbol{x} : \\ & \alpha = c\alpha \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha \Delta \boldsymbol{x} \end{aligned}$$

with $0 < c < 1$, α is like a "step-size", and b is the "tolerance". Usually, $c = 0.5$ and b in range from $1e^{-4}$ to 0.1.

Line Search - Armijo Rule

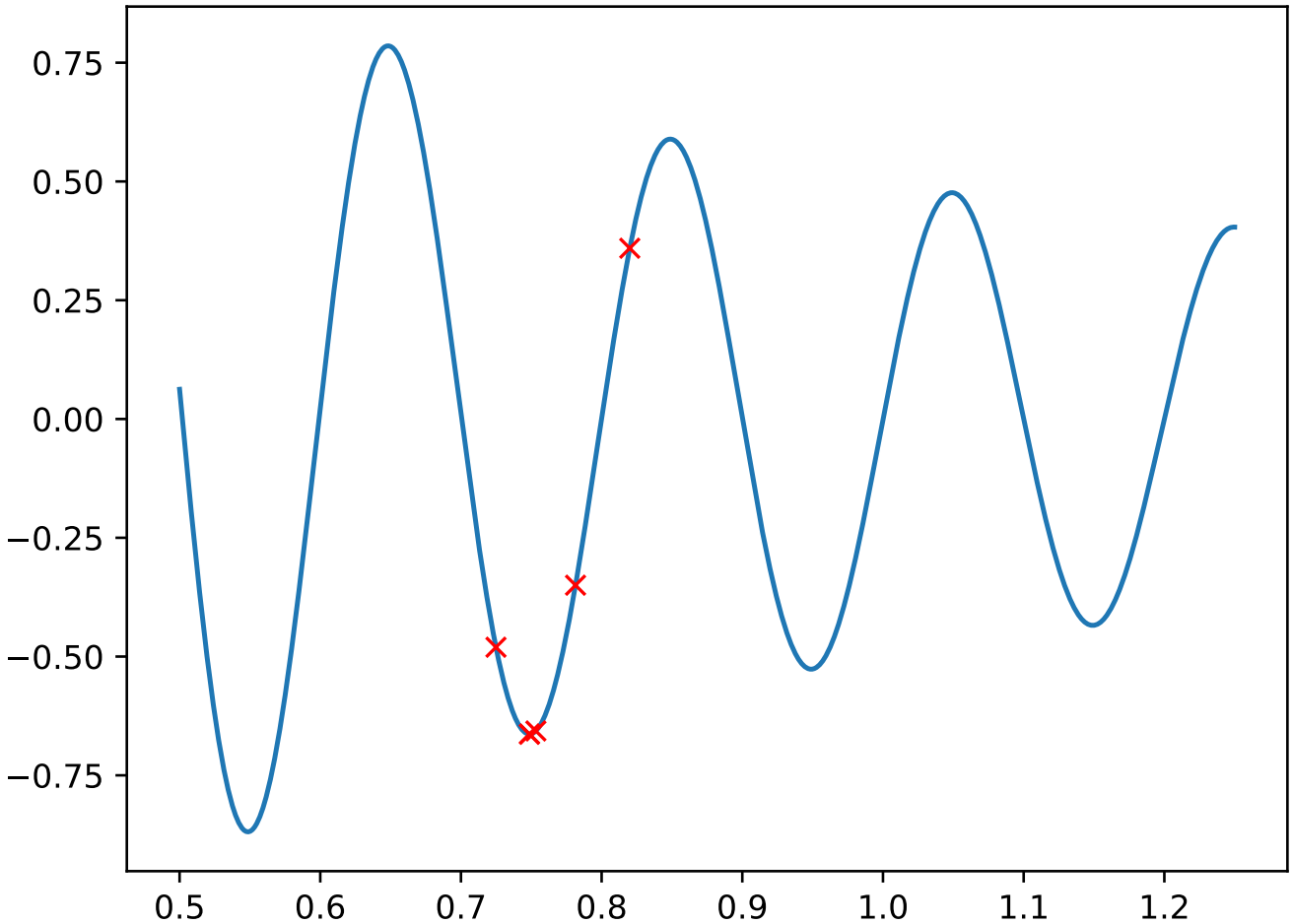
Armijo Rule:

$$
\begin{aligned}
\alpha &= 1 \\
\text{while } & f(\boldsymbol{x}_k + \alpha \Delta \boldsymbol{x}) > f(\boldsymbol{x}_k) + b\alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)^T \Delta \boldsymbol{x} : \\
& \alpha = c\alpha \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha \Delta \boldsymbol{x}
\end{aligned}
$$

with $0 < c < 1$, α is like a "step-size", and b is the "tolerance". Usually, $c = 0.5$ and b in range from $1e^{-4}$ to 0.1.

Intuition: The actual step needs to agree with the Taylor expansion (linearization) up to a tolerance.

Newton's Method with tricks - Code Example



Newton's Method and Optimization

Newton's Method:

- Quadratic Convergence
- Very accurate
- $O(N^3)$ for solving the linear system
  - We can improve this by exploiting the structure of the problem
- **Newton's Method for Optimization:**
  - Local! It will only find the "closest" minimum point
  - **We need to "fix" it in order to optimize!**

Gradient Descent

What if we do not have the Hessian?

- We can approximate it: **Quasi-Newton Methods** (e.g. BFGS)
- We can use the gradient only!

[1] Toussaint, Marc. 2012. Some Notes on Gradient Descent.

Gradient Descent

What if we do not have the Hessian?

- We can approximate it: **Quasi-Newton Methods** (e.g. BFGS)
- We can use the gradient only!

Let's go back to our update equation:

$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{"descent"}} \underbrace{\alpha}_{\text{"learning rate"}} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{"gradient"}}$$

How to choose the "learning rate"?

[1] Toussaint, Marc. 2012. Some Notes on Gradient Descent.

Gradient Descent

What if we do not have the Hessian?

- We can approximate it: **Quasi-Newton Methods** (e.g. BFGS)
- We can use the gradient only!

Let's go back to our update equation:

$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{"descent"}} \underbrace{\alpha}_{\text{"learning rate"}} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{"gradient"}}$$

How to choose the "learning rate"?

1. Initialize $\alpha = \alpha_0$
2. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)$
3. if $f(\boldsymbol{x}_{k+1}) > f(\boldsymbol{x}_k)$ then $\alpha = \alpha\beta$ $(0 < \beta < 1)$ and $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$
4. if $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$ then $\alpha = \alpha\zeta$ $(\zeta > 1)$
5. $k = k + 1$ and back to step 2 until we converge

It is guaranteed to always converge to a local minimum[1]! Usually $\beta = 0.5, \zeta = 1.2$.

[1] Toussaint, Marc. 2012. Some Notes on Gradient Descent.

Gradient Descent With Momemtum

$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{“descent”}} \underbrace{\alpha}_{\text{“learning rate”}} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{“gradient”}}$$

How to choose the “learning rate”?

- ...

Gradient Descent With Momemtum

$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{“descent”}} \underbrace{\alpha}_{\text{“learning rate”}} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{“gradient”}}$$

How to choose the “learning rate”?

- ...
- Momemtum:
  1. Initialize $\boldsymbol{m}_0 = \mathbf{0}, \eta \in [0, 1]$
  2. $\boldsymbol{m}_{k+1} = \eta \boldsymbol{m}_k + (1 - \eta)\alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)$
  3. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{m}_{k+1}$
  4. $k = k + 1$ and back to step 2 until we converge

Faster convergence and “smooths out” the noise in gradient steps.

Adam Optimizer

$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{“descent”}} \underbrace{\alpha}_{\text{“learning rate”}} \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{“gradient”}}$$

How to choose the “learning rate”?

Adam Optimizer

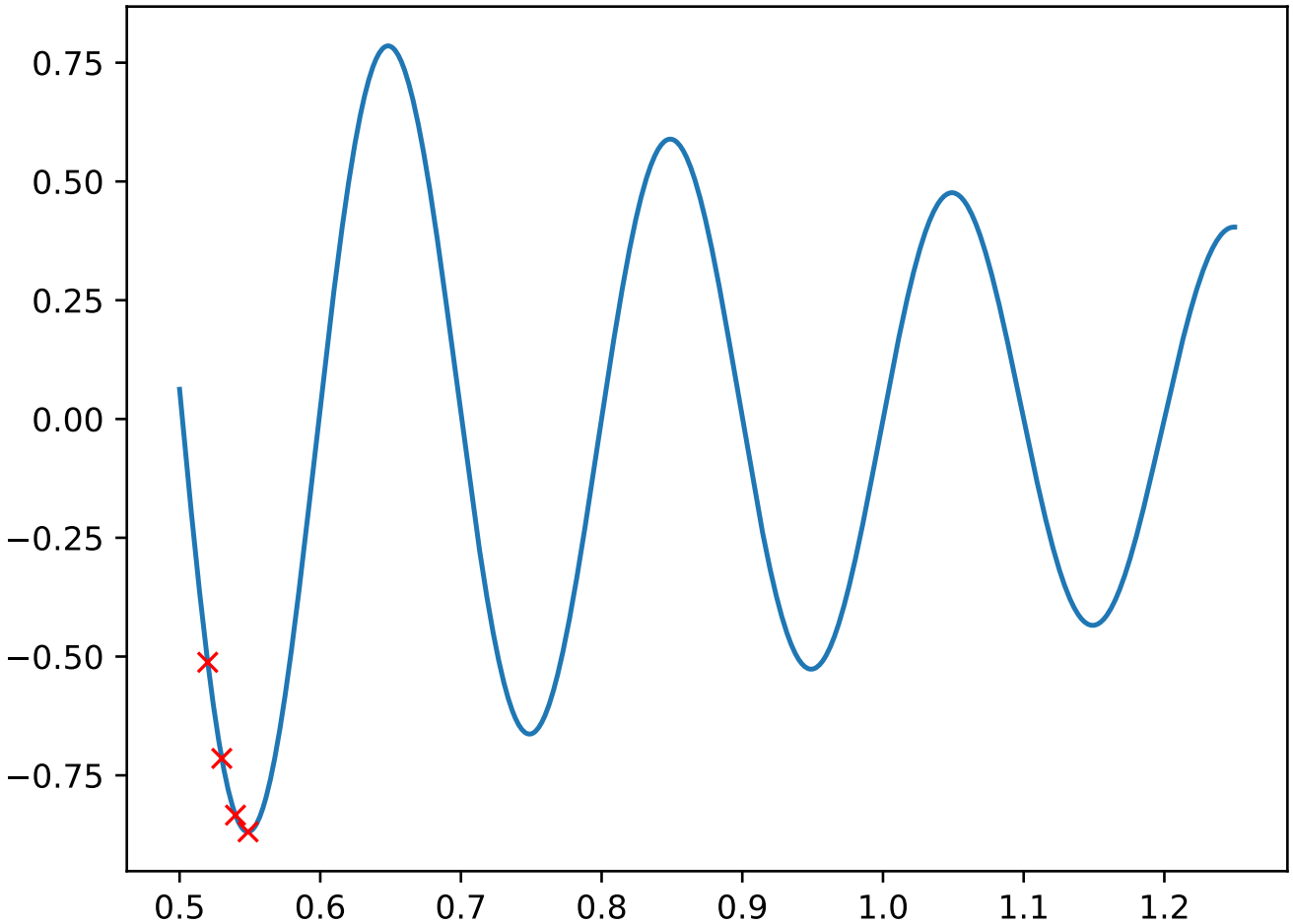
$$\Delta \boldsymbol{x} = \underbrace{-}_{\text{“descent”}} \quad \underbrace{\alpha}_{\text{“learning rate”}} \quad \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)}_{\text{“gradient”}}$$

How to choose the “learning rate”?

- **Adam Optimizer**:
  1. Initialize $\boldsymbol{m}_0 = \boldsymbol{v}_0 = \boldsymbol{0}, \beta_1, \beta_2 \in [0, 1), \epsilon > 0$
  2. $\boldsymbol{g}_{k+1} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)$
  3. $\boldsymbol{m}_{k+1} = \beta_1 \boldsymbol{m}_k + (1 - \beta_1)\boldsymbol{g}_{k+1}$
  4. $\boldsymbol{v}_{k+1} = \beta_2 \boldsymbol{v}_k + (1 - \beta_2)\boldsymbol{g}_{k+1}^2$
  5. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \dfrac{\frac{\boldsymbol{m}_{k+1}}{1-\beta_1^{k+1}}}{\sqrt{\frac{\boldsymbol{v}_{k+1}}{1-\beta_2^{k+1}}} + \epsilon}$
  6. $k = k + 1$ and back to step 2 until we converge

One of the most versatile first order optimizers. *Not always the best choice, but surely the first one!* Usually $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

Gradient Descent - Code Example



Stochastic Gradient Descent

- We cannot always compute the *exact* $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)$:
  - We do not have access and we approximate it via finite differences
  - Computation limitations: too big of a dataset!
  - Computation limitations: too expensive to compute the actual one, we use an approximated version!
  - ...

Stochastic Gradient Descent

- We cannot always compute the *exact* $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)$:
  - We do not have access and we approximate it via finite differences
  - Computation limitations: too big of a dataset!
  - Computation limitations: too expensive to compute the actual one, we use an approximated version!
  - ...
- **Stochastic Gradient Descent** (SGD) is just acknowledging this fact!
- Under assumptions and mitigation measures, we can still theoretically converge to a local optimum

SGD - Verdict

- **Pros:**
  - Can handle large datasets
  - Can handle noisy gradients
  - Can handle non-smooth functions

SGD - Verdict

- **Pros:**
  - Can handle large datasets
  - Can handle noisy gradients
  - Can handle non-smooth functions
- **Cons:**
  - Convergence is not guaranteed
  - May oscillate around the optimum
  - Requires careful tuning of hyperparameters

Gradient Approximation via Finite Differences

- **What if we cannot compute the gradient at all?**

Gradient Approximation via Finite Differences

- **What if we cannot compute the gradient at all?**
- We know that $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \lim_{h \to 0} \frac{f(\boldsymbol{x}+h)-f(\boldsymbol{x})}{h}$

Gradient Approximation via Finite Differences

- **What if we cannot compute the gradient at all?**
- We know that $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \lim_{h \to 0} \frac{f(\boldsymbol{x}+h)-f(\boldsymbol{x})}{h}$
- **Idea:** Let's use $\frac{f(\boldsymbol{x}+h)-f(\boldsymbol{x})}{h}$ as an approximation of the gradient!
- As we make $\boldsymbol{h}$ smaller, we should be getting better approximations! Up to a limit! We have finite precision!
- In N-D, this looks like this:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \approx \begin{bmatrix} \frac{f(\boldsymbol{x}+h\boldsymbol{e}_1)-f(\boldsymbol{x})}{h} \\ \vdots \\ \frac{f(\boldsymbol{x}+h\boldsymbol{e}_D)-f(\boldsymbol{x})}{h} \end{bmatrix}$$

where $\boldsymbol{e}_i$ is a direction vector (aka all zeros instead of the i-th component that is equal to 1).

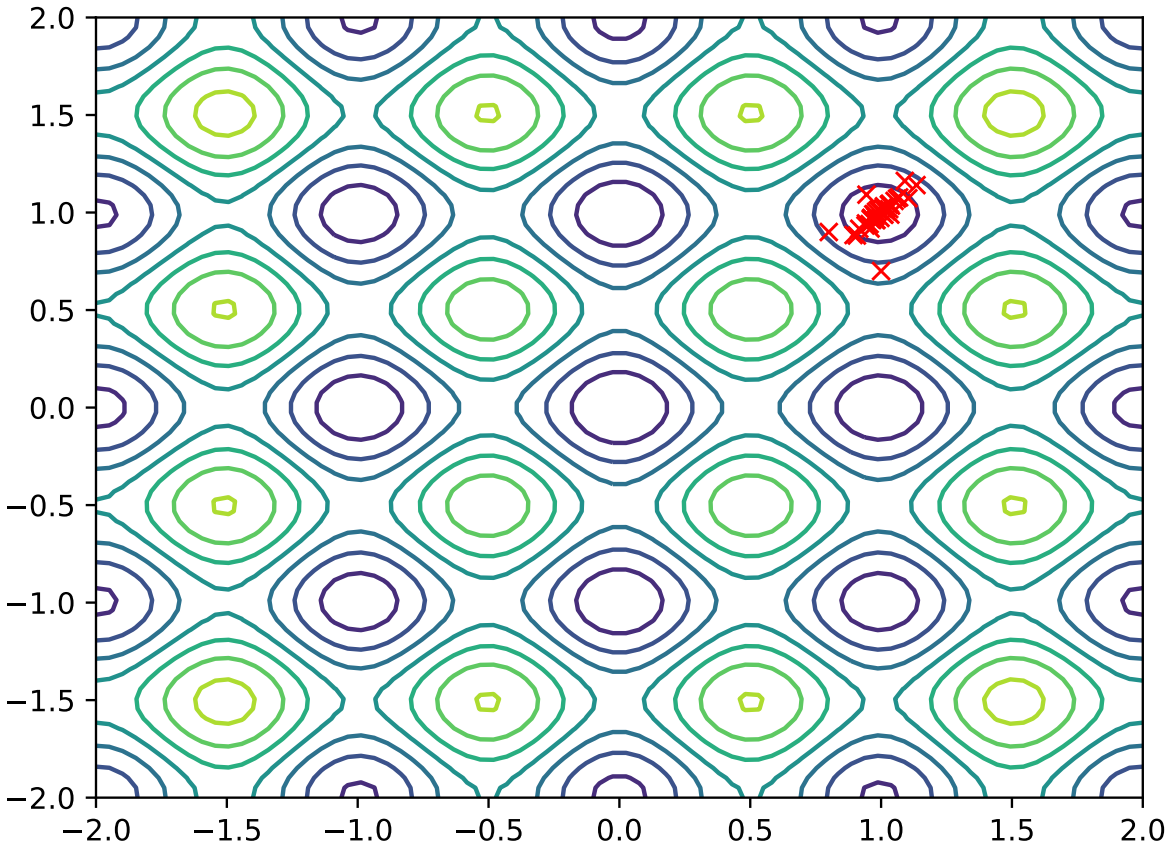
Gradient Approximation via Finite Differences (2)

- In practice we use $\frac{f(\boldsymbol{x}+h)-f(\boldsymbol{x}-h)}{2h}$
- This has an error $O(h^2)$
- Numerically more stable (aka we can use smaller h)
- In N-D, this looks like this:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \approx \begin{bmatrix} \frac{f(\boldsymbol{x}+h\boldsymbol{e}_1)-f(\boldsymbol{x}-h\boldsymbol{e}_1)}{2h} \\ \vdots \\ \frac{f(\boldsymbol{x}+h\boldsymbol{e}_D)-f(\boldsymbol{x}-h\boldsymbol{e}_D)}{2h} \end{bmatrix}$$

where $\boldsymbol{e}_i$ is a direction vector (aka all zeros instead of the i-th component that is equal to 1).

Gradient Approximation - Code Example (2)



Gradient Approximation via Finite Differences (3)

Pros:

- Simple
- Easy to implement
- Accurate results for small h

Cons:

- We need $2D$ function calls for one gradient approximation
- If your function is "expensive", this is a bad way of approximating the gradient
- Many numerical errors can arise

Gradient Approximation with Random Samples

- How can we use the finite difference idea and random sampling to approximate more efficiently high-dimensional gradients?
- **Simple idea:**

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \approx \sum_{i=1}^{N} \frac{(f(\boldsymbol{x}+\sigma\boldsymbol{\epsilon}_i) - f(\boldsymbol{x}-\sigma\boldsymbol{\epsilon}_i))}{2N\sigma}\boldsymbol{\epsilon}_i$$

where $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

Gradient Approximation with Random Samples

- How can we use the finite difference idea and random sampling to approximate more efficiently high-dimensional gradients?
- **Simple idea:**

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \approx \sum_{i=1}^{N} \frac{(f(\boldsymbol{x}+\sigma\boldsymbol{\epsilon}_i) - f(\boldsymbol{x}-\sigma\boldsymbol{\epsilon}_i))}{2N\sigma}\boldsymbol{\epsilon}_i$$

  where $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

- **Why is this efficient?**

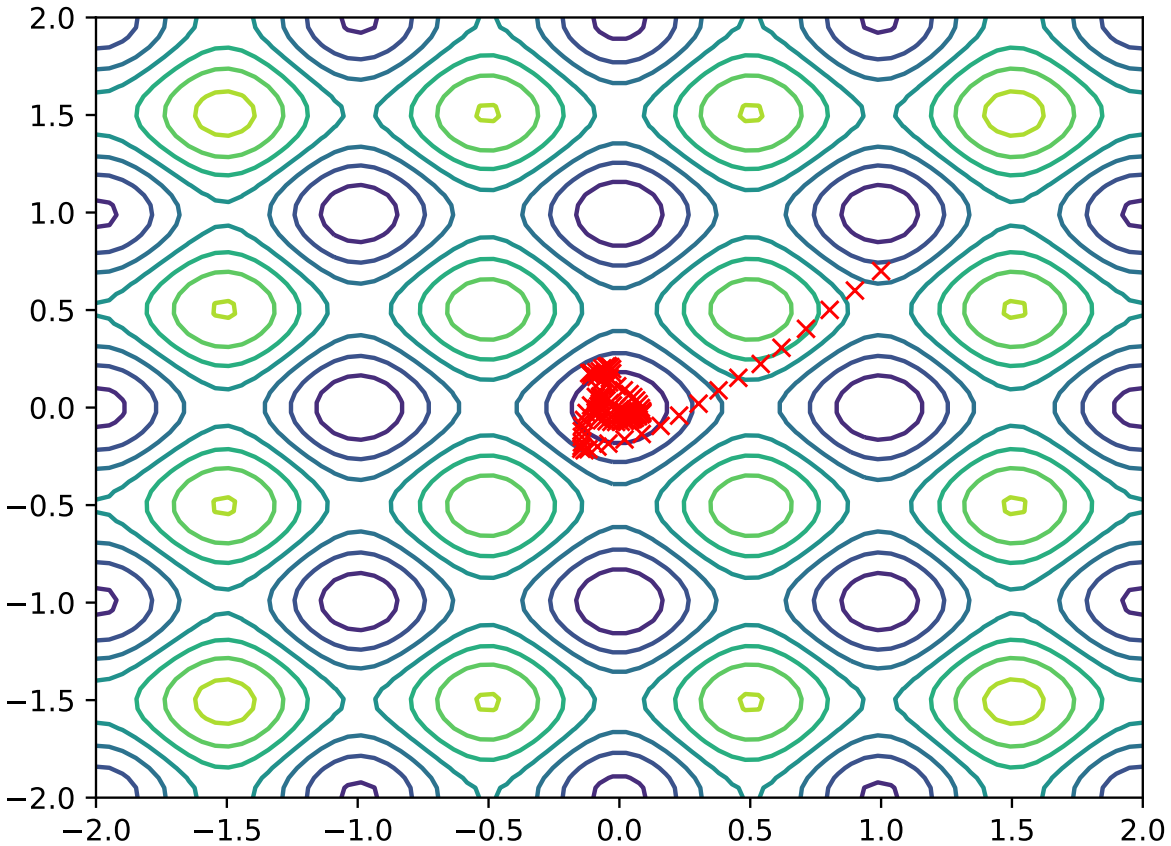
Gradient Approximation with Random Samples

- How can we use the finite difference idea and random sampling to approximate more efficiently high-dimensional gradients?
- **Simple idea:**

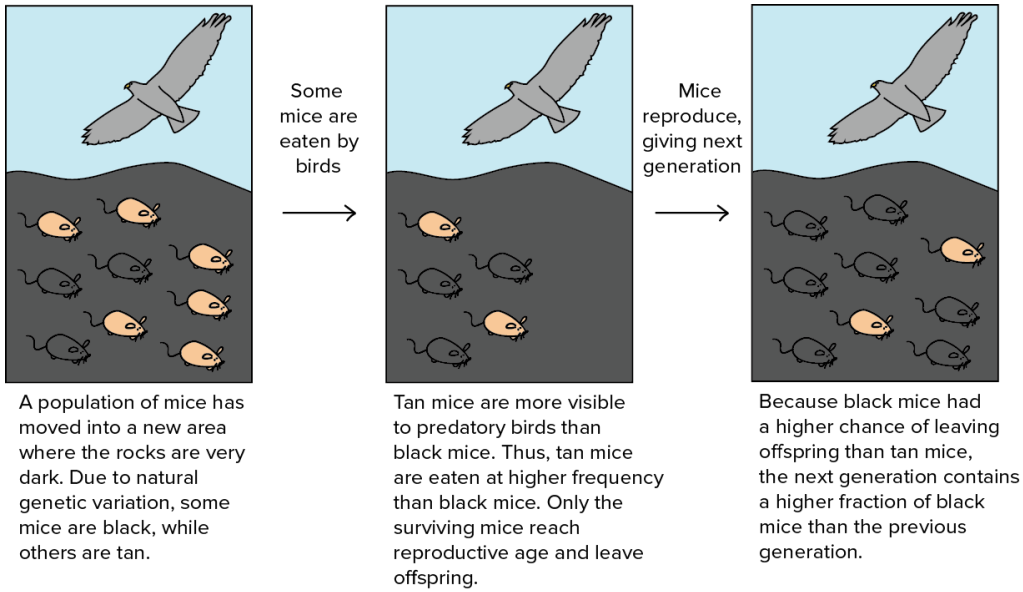
$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \approx \sum_{i=1}^{N} \frac{(f(\boldsymbol{x}+\sigma\boldsymbol{\epsilon}_i) - f(\boldsymbol{x}-\sigma\boldsymbol{\epsilon}_i))}{2N\sigma}\boldsymbol{\epsilon}_i$$

  where $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.
- **Why is this efficient?**
- **Why is this effective?**

Gradient with Random Samples - Code Example



Evolutionary Algorithms



Source: Khan Academy "Darwin, evolution, & natural selection"

- The previous approximation is part of an *evolutionary algorithm* called **Antithetic Evolution Strategies**!!
- **Evolutionary Algorithms** (EAs) are inspired by the natural evolution of fittest.
- **Evolution Strategies** are EAs that model the population as a parameteric distribution (e.g. Gaussian) and optimize its parameters.

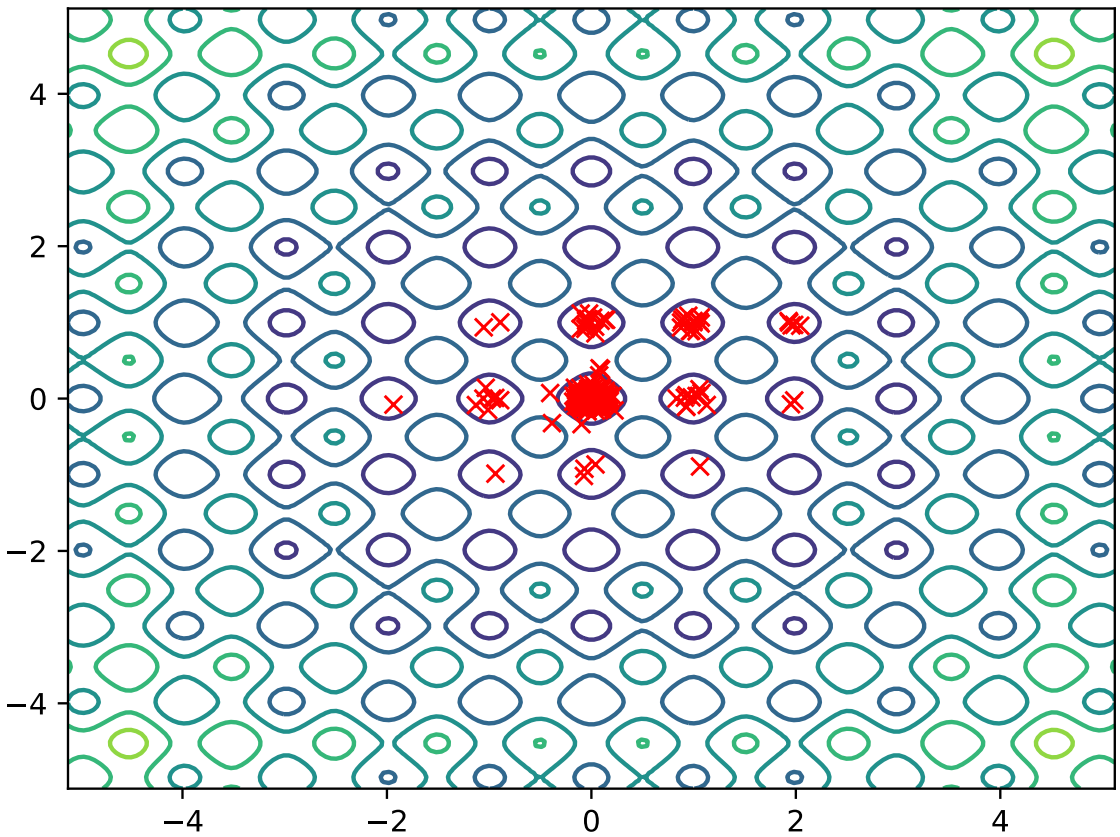
Evolutionary Algorithms - General Structure

1. Generate or set the initial population
2. Evaluate each individual in the population
3. Select individuals for reproduction (*parents*)
4. Breed new individuals (*offspring*) or update existing individuals through crossover and mutation operations
5. Discard the least-fit individuals or replace them with new individuals
6. Possibly re-generate the population
7. Go back to step 2 (repeat until convergence/stopping criteria)

A Simple Genetic Algorithm

1. Generate or set the initial population of size N, $\mathcal{P} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$
2. Evaluate each individual in the population, $\mathcal{F} = \{f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)\}$
3. Sort $\mathcal{F}$ and select the best $M < N$ individuals for reproduction, $\mathcal{P}_{\text{elite}} = \{\boldsymbol{x}_1^{\text{elite}}, \ldots, \boldsymbol{x}_M^{\text{elite}}\}$
4. Set $\mathcal{O} = \varnothing$
5. For each $\boldsymbol{x}_i^{\text{elite}} \in \mathcal{P}_{\text{elite}}$:
   1. Select randomly another $\boldsymbol{x}_j^{\text{elite}} \in \mathcal{P}_{\text{elite}}$ $(i \neq j)$
   2. Compute: $\boldsymbol{x}_i^{\text{offspring}} = \boldsymbol{x}_i^{\text{elite}} + \underbrace{\epsilon_1}_{\text{mutation}} + \underbrace{\epsilon_2(\boldsymbol{x}_j^{\text{elite}} - \boldsymbol{x}_i^{\text{elite}})}_{\text{crossover}}$, where $\epsilon_1 \sim \mathcal{N}(\mathbf{0}, \sigma_1 \boldsymbol{I}), \epsilon_2 \sim \mathcal{N}(\mathbf{0}, \sigma_2 \boldsymbol{I})$.
   3. Add $\boldsymbol{x}_i^{\text{offspring}}$ to $\mathcal{O}$
6. Replace the lowest scoring individuals of $\mathcal{P}$ with $\mathcal{O}$
7. Go back to step 2 (repeat until convergence/stopping criteria)

Genetic Algorithm - Code Example

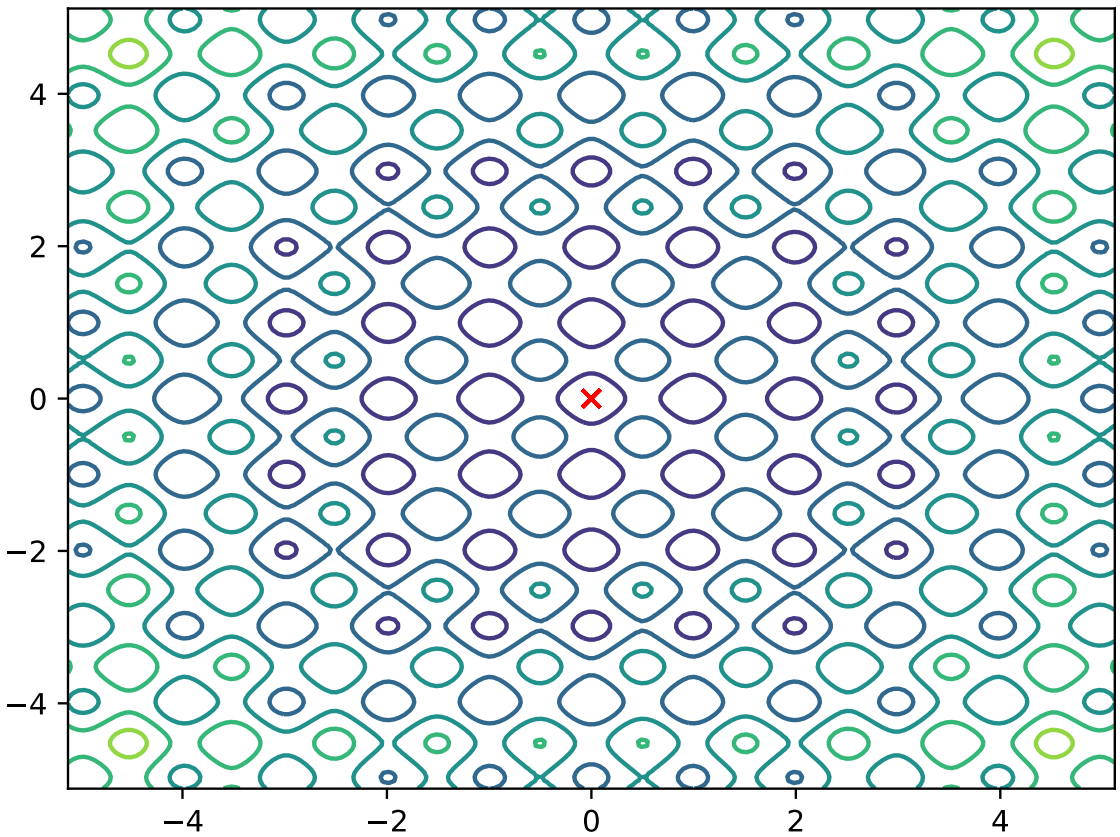


Cross Entropy Method (CEM)

Cross Entropy Method (CEM) or Simple Evolution Strategies:

1. The population is a Gaussian distribution: $\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$
2. Generate a population of size N, $\mathcal{P} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, where $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\sigma}_k^2))$
3. Evaluate each individual in the population, $\mathcal{F} = \{f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)\}$
4. Select the best $M < N$ individuals from $\mathcal{F}$: $\mathcal{P}_{\text{elite}} = \{\boldsymbol{x}_1^{\text{elite}}, \ldots, \boldsymbol{x}_M^{\text{elite}}\}$
5. $\boldsymbol{\mu}_{k+1} = \frac{1}{M} \sum_{\boldsymbol{x}_i^{\text{elite}} \in \mathcal{P}_{\text{elite}}} \boldsymbol{x}_i^{\text{elite}}$ (Sample mean)
6. $\boldsymbol{\sigma}_{k+1}^2 = \frac{1}{M} \sum_{\boldsymbol{x}_i^{\text{elite}} \in \mathcal{P}_{\text{elite}}} (\boldsymbol{x}_i^{\text{elite}} - \boldsymbol{\mu}_k)^2$ (Sample variance)
7. Go back to step 2 (repeat until convergence/stopping criteria)

Cross Entropy Method - Code Example



Covariance Matrix Adaptation ES (CMA-ES)

- Similar to CEM
- We use a full Covariance matrix, not just a diagonal
- Effective heuristics for adapting the Covariance matrix
- Versatile black-box optimizer!
- Very good performance with small populations!

Zero-Order-Optimization - Verdict

- **Pros:**
  - No need for gradients
  - Can handle non-smooth functions
  - Can handle noisy functions
  - Global optimization

Zero-Order-Optimization - Verdict

- **Pros:**
  - No need for gradients
  - Can handle non-smooth functions
  - Can handle noisy functions
  - Global optimization
- **Cons:**
  - Slow convergence
  - Requires many function evaluations
  - Tricky to scale to high dimensions

Constrained Optimization: Equality Constraints

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$\text{s.t. } h(\boldsymbol{x}) = 0$$

where $f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}, h(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}^M$.

First Order Necessary Conditions:

$$1)\ \frac{\partial f}{\partial \boldsymbol{x}} + \boldsymbol{\lambda}^T \frac{\partial h}{\partial \boldsymbol{x}} = 0$$
$$2)\ h(\boldsymbol{x}) = 0$$

$\boldsymbol{\lambda} \in \mathbb{R}^M$

Constrained Optimization: Equality Constraints

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$\text{s.t. } h(\boldsymbol{x}) = 0$$

where $f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}, h(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}^M$.

First Order Necessary Conditions:

$$1)\ \frac{\partial f}{\partial \boldsymbol{x}} + \boldsymbol{\lambda}^T \frac{\partial h}{\partial \boldsymbol{x}} = 0$$
$$2)\ h(\boldsymbol{x}) = 0$$

$\boldsymbol{\lambda} \in \mathbb{R}^M$ and $\boldsymbol{\lambda}$ is called the **"Lagrange multiplier"** or **"dual variable"**.

Constrained Optimization: Equality Constraints (2)

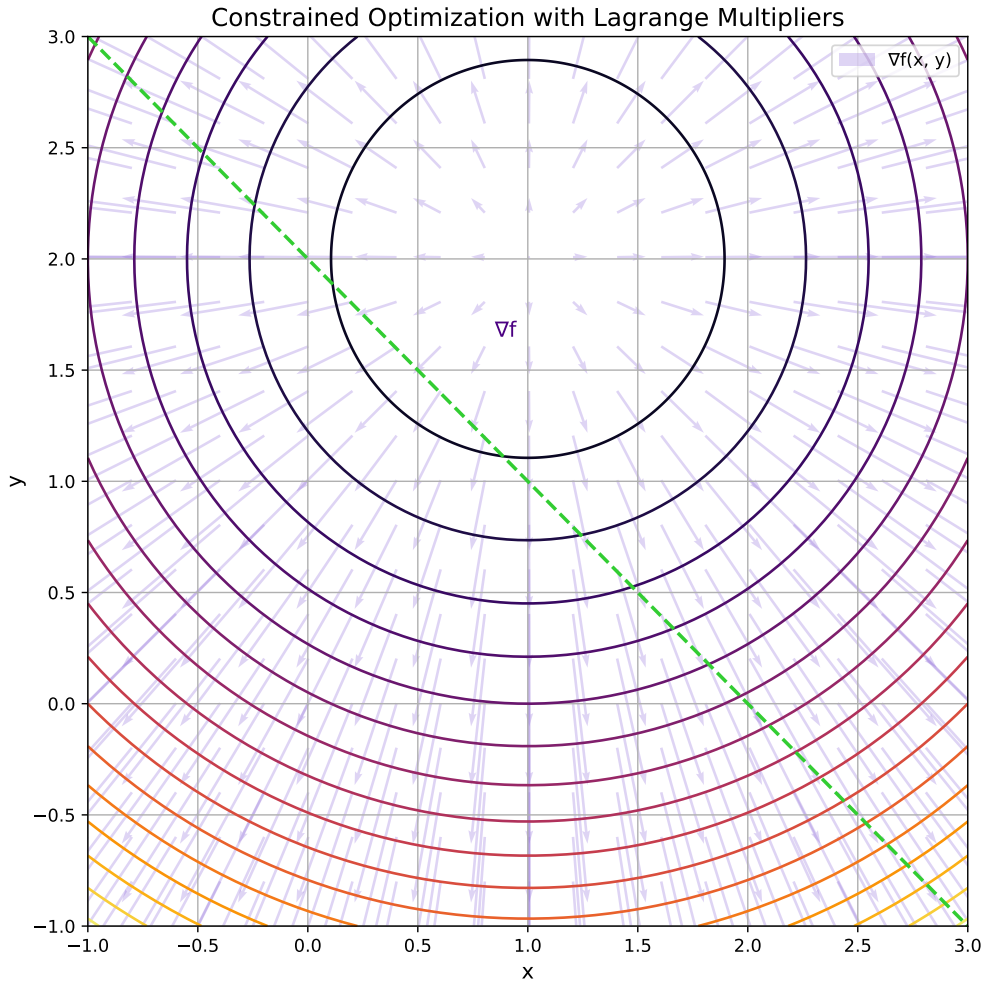
$$\min_{x,y} \; (x-1)^2 + (y-2)^2$$

$$\text{s.t. } \; x + y - 2 = 0$$

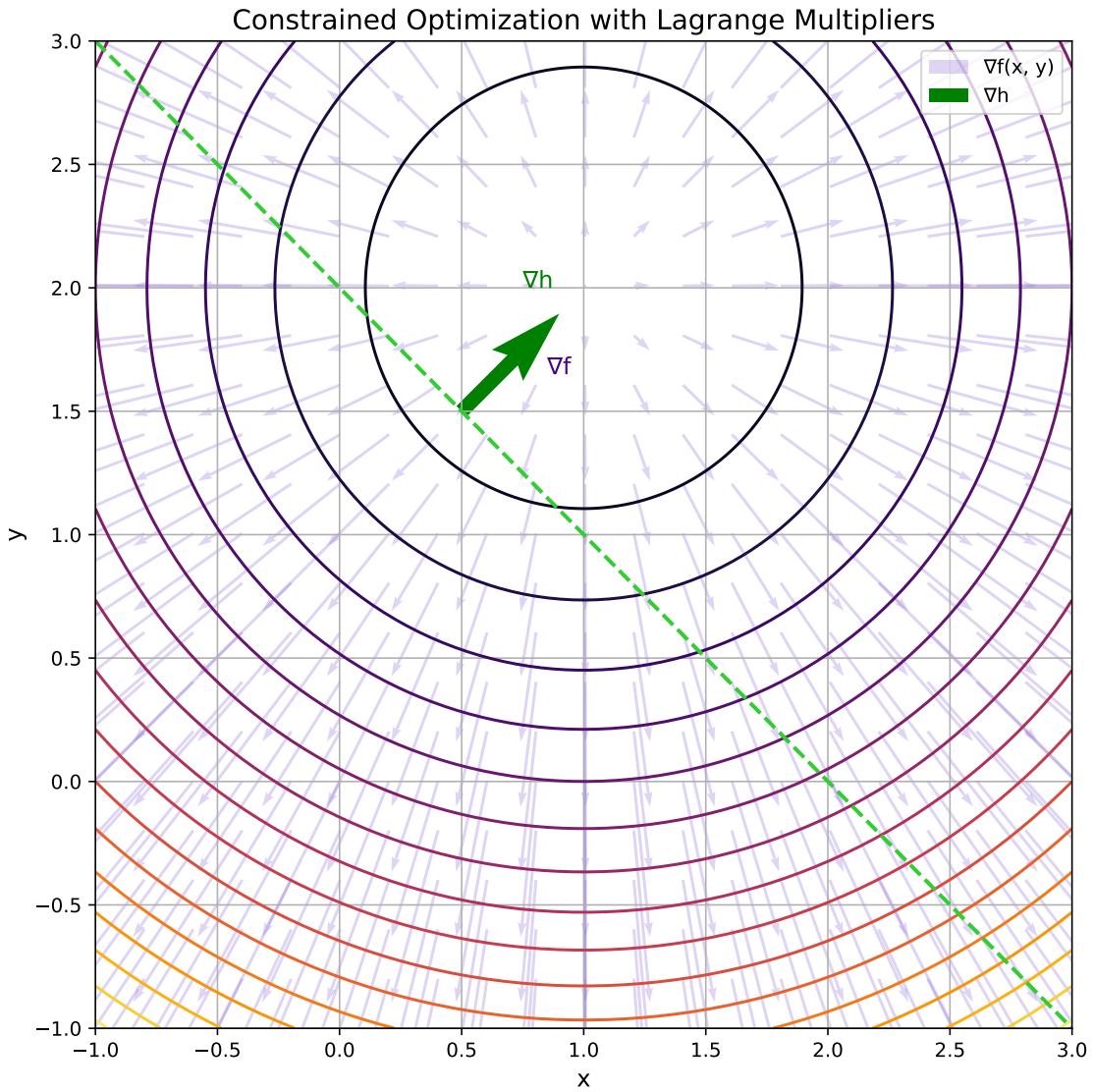
Constrained Optimization: Equality Constraints (2)

$$\min_{x,y} \; (x-1)^2 + (y-2)^2$$

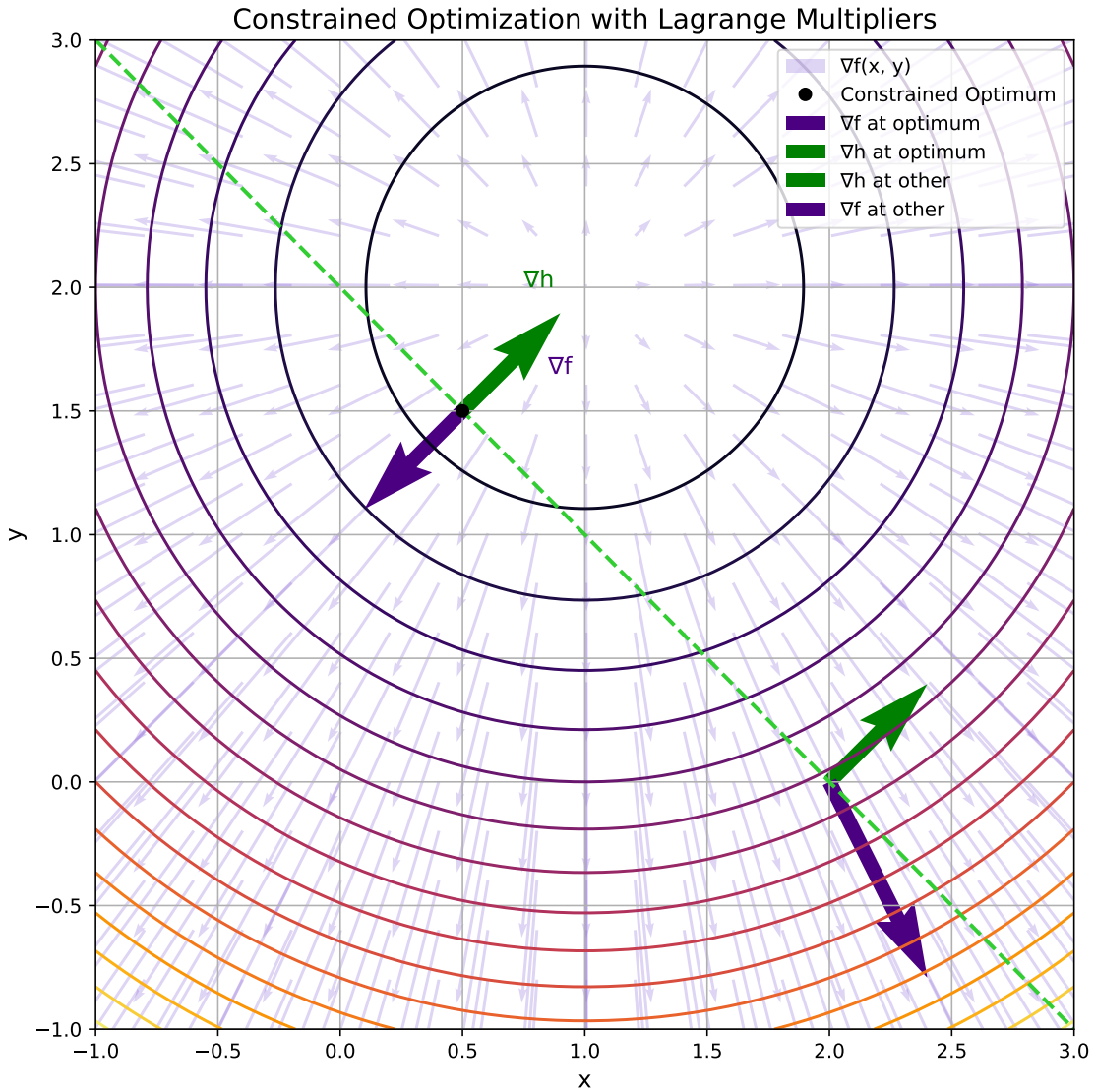
$$\text{s.t.} \; x + y - 2 = 0$$



Constrained Optimization: Equality Constraints (3)



Constrained Optimization: Equality Constraints (4)



Lagrangian for Equality Constraints

We define the **Lagrangian** as follows:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T h(\boldsymbol{x})$$

Now we can say:

$$1)\ \nabla_{\boldsymbol{x}} \mathcal{L} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \left(\frac{\partial h}{\partial \boldsymbol{x}}\right)^T \boldsymbol{\lambda} = 0$$

$$2)\ \nabla_{\boldsymbol{\lambda}} \mathcal{L} = h(\boldsymbol{x}) = 0$$

We have recovered the original conditions! We call these conditions **"KKT Conditions"**.

Optimization with Equality Constraints

OK! How can we find the solution to this?

OK! How can we find the solution to this?

Idea! Let's use Newton's method!

$$\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}_k+\Delta\boldsymbol{x},\boldsymbol{\lambda}_k+\Delta\boldsymbol{\lambda})=\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)+\frac{\partial^2\mathcal{L}}{\partial\boldsymbol{x}^2}\Big|_{\boldsymbol{x}_k,\boldsymbol{\lambda}_k}\Delta\boldsymbol{x}+\underbrace{\frac{\partial^2\mathcal{L}}{\partial\boldsymbol{x}\partial\boldsymbol{\lambda}}\Big|_{\boldsymbol{x}_k,\boldsymbol{\lambda}_k}}_{\left(\frac{\partial h}{\partial\boldsymbol{x}}\Big|_{\boldsymbol{x}_k}\right)^T}\Delta\boldsymbol{\lambda}=0$$

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{x}_k+\Delta\boldsymbol{x},\boldsymbol{\lambda}_k+\Delta\boldsymbol{\lambda})=h(\boldsymbol{x}_k)+\frac{\partial h}{\partial\boldsymbol{x}}\Big|_{\boldsymbol{x}_k}\Delta\boldsymbol{x}=0$$

Optimization with Equality Constraints (2)

$$\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}_k+\Delta\boldsymbol{x},\boldsymbol{\lambda}_k+\Delta\boldsymbol{\lambda})=\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)+\frac{\partial^2\mathcal{L}}{\partial\boldsymbol{x}^2}\Big|_{\boldsymbol{x}_k,\boldsymbol{\lambda}_k}\Delta\boldsymbol{x}+\left(\frac{\partial h}{\partial\boldsymbol{x}}\Big|_{\boldsymbol{x}_k}\right)^T\Delta\boldsymbol{\lambda}=0$$

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{x}_k+\Delta\boldsymbol{x},\boldsymbol{\lambda}_k+\Delta\boldsymbol{\lambda})=h(\boldsymbol{x}_k)+\frac{\partial h}{\partial\boldsymbol{x}}\Big|_{\boldsymbol{x}_k}\Delta\boldsymbol{x}=0$$

This is a linear system in $\boldsymbol{z}=\begin{bmatrix}\Delta\boldsymbol{x}\\ \Delta\boldsymbol{\lambda}\end{bmatrix}$:

$$\begin{bmatrix}\frac{\partial^2\mathcal{L}}{\partial\boldsymbol{x}^2}\Big|_{\boldsymbol{x}_k,\boldsymbol{\lambda}_k} & \left(\frac{\partial h}{\partial\boldsymbol{x}}\Big|_{\boldsymbol{x}_k}\right)^T\\ \frac{\partial h}{\partial\boldsymbol{x}}\Big|_{\boldsymbol{x}_k} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{x}\\ \Delta\boldsymbol{\lambda}\end{bmatrix}=\begin{bmatrix}-\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)\\ -h(\boldsymbol{x}_k)\end{bmatrix}$$

This is often called the **"KKT System"**.

Gauss-Netwon Method

$$\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{x}^2} = \nabla^2_{\boldsymbol{xx}} f + \frac{\partial}{\partial \boldsymbol{x}} \Big[(\frac{\partial h}{\partial \boldsymbol{x}})^T \boldsymbol{\lambda} \Big]$$

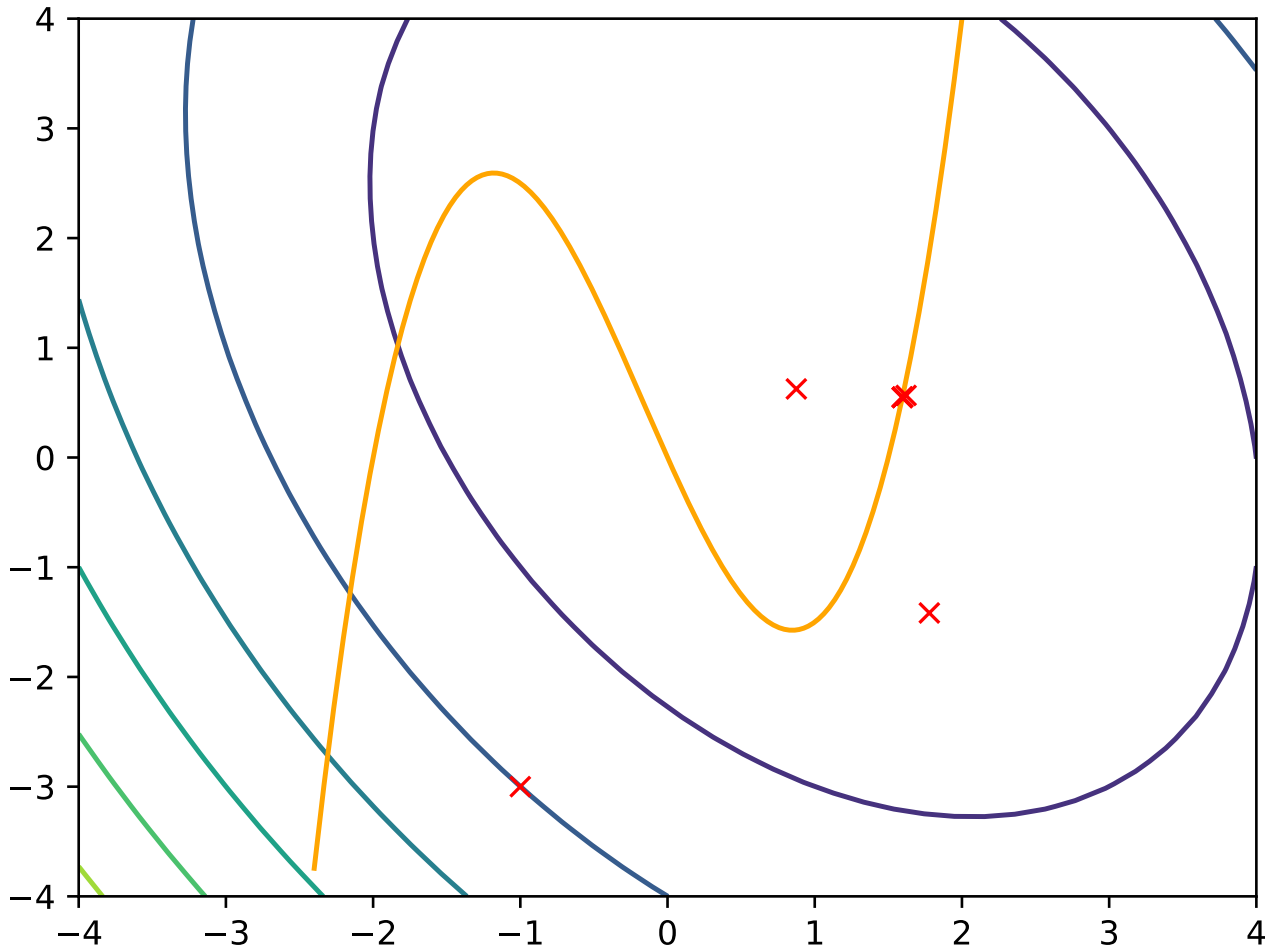
The second term here is:

- A third rank tensor
- Expensive to compute (there are "complicated" ways to compute this faster, e.g. pullback and auto-diff)
- Can be "ugly"/ill-conditioned

In practice, we often remove this term:

- The algorithm is now called "Gauss-Newton"
- In theory it has slower convergence
- In practice it takes more iterations, but each iteration is faster!

Gauss-Netwon Method - Code Example



Constrained Optimization: Inequality Constraints

$$\min_{\boldsymbol{x}} \; f(\boldsymbol{x})$$
$$\text{s.t. } \; g(\boldsymbol{x}) \leqslant 0$$

where $f(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}, g(\boldsymbol{x}) : \mathbb{R}^N \rightarrow \mathbb{R}^M$. We define the **Lagrangian** as follows:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\mu}^T g(\boldsymbol{x})$$

First Order Necessary (KKT) Conditions:

$$1) \; \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \left(\frac{\partial g}{\partial \boldsymbol{x}}\right)^T \boldsymbol{\mu} = 0, \qquad \text{“stationarity”}$$

$$2) \; g(\boldsymbol{x}) \leqslant 0, \qquad \text{“primal feasibility”}$$

$$3) \; \boldsymbol{\mu} \geqslant 0, \qquad \text{“dual feasibility”}$$

$$4) \; \boldsymbol{\mu}^T g(\boldsymbol{x}) = 0, \qquad \text{“complementarity”}$$

Constrained Optimization: Inequality Constraints (2)

We cannot solve the above in closed form as in the equality case!!

Many methods to solve this:

- Augmented Lagrangian Method (ALM)
- Sequential Quadratic Programming (SQP)
- Interior Point Methods
- ...

Augmented Lagrangian Method

We define the **Augmented Lagrangian** as:

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\mu}^T g(\boldsymbol{x}) + \frac{\rho}{2}\|\max(0, g(\boldsymbol{x}))\|^2$$

$\frac{\rho}{2}\|\max(0, g(\boldsymbol{x}))\|^2$ is called the **penalty** term.

Pseudocode:

$$1)\ \boldsymbol{x}_{k+1} = \min_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{\mu}_k)$$

$$2)\ \boldsymbol{\mu}_{k+1} = \max\Big(\mathbf{0}, \boldsymbol{\mu}_k + \rho g(\boldsymbol{x}_{k+1})\Big)$$

$$3)\ \rho = \alpha\rho, \text{ with } \alpha \in \mathbb{R}^+$$

Usually $\alpha \approx 10$ and we use (2) only when the constraints are violated. The method comes with strong convergence guarantees.

How can we solve (1)?

Augmented Lagrangian Method - Minimizing $\mathcal{L}_\rho$

- We need to solve $\min_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{\mu})$
- Important to note that $\boldsymbol{\mu}$ and ρ are constants!

Augmented Lagrangian Method - Minimizing $\mathcal{L}_\rho$

- We need to solve $\min_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{\mu})$
- Important to note that $\boldsymbol{\mu}$ and ρ are constants!
- Let's apply Newton's Method:

$$\nabla_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}_k + \Delta \boldsymbol{x}, \boldsymbol{\mu}_k) \approx$$

$$\nabla_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}_k, \boldsymbol{\mu}_k) + \frac{\partial}{\partial \boldsymbol{x}} \Big(\nabla_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}_k, \boldsymbol{\mu}_k) \Big) \Delta \boldsymbol{x} = 0$$

- We need the gradient of $\mathcal{L}_\rho$:

$$\begin{aligned} \nabla_{\boldsymbol{x}} \mathcal{L}_\rho((\boldsymbol{x}, \boldsymbol{\mu})) &= \left(\frac{\partial f}{\partial \boldsymbol{x}} + \boldsymbol{\mu}^T \frac{\partial g}{\partial \boldsymbol{x}} + \rho g(\boldsymbol{x})^T \frac{\partial g}{\partial \boldsymbol{x}} \right)^T \\ &= \left(\frac{\partial f}{\partial \boldsymbol{x}} + \Big(\boldsymbol{\mu} + \rho g(\boldsymbol{x}) \Big)^T \frac{\partial g}{\partial \boldsymbol{x}} \right)^T = 0 \end{aligned}$$

Augmented Lagrangian Method - Minimizing $\mathcal{L}_\rho$ (2)

$$\nabla_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}_k + \Delta\boldsymbol{x}, \boldsymbol{\mu}_k) \approx$$

$$\nabla_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}_k, \boldsymbol{\mu}_k) + \frac{\partial}{\partial \boldsymbol{x}} \Big(\nabla_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}_k, \boldsymbol{\mu}_k) \Big) \Delta\boldsymbol{x} = 0$$

We can use the Gauss-Newton version and:

$$\frac{\partial}{\partial \boldsymbol{x}} \Big(\nabla_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}_k, \boldsymbol{\mu}_k) \Big) \approx \nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}_k) + \rho \Big(\frac{\partial g}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k} \Big)^T \frac{\partial g}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k}$$

$$\Delta\boldsymbol{x} = -\Big(\nabla^2_{\boldsymbol{xx}} f(\boldsymbol{x}_k) + \rho \Big(\frac{\partial g}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k} \Big)^T \frac{\partial g}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k} \Big)^{-1} \nabla_{\boldsymbol{x}} \mathcal{L}_\rho(\boldsymbol{x}_k, \boldsymbol{\mu}_k)$$

Merit Functions

- **How can we apply line search in the constrained case?**

Merit Functions

- **How can we apply line search in the constrained case?**
- We define a **merit function**, $P(\boldsymbol{x})$
- Then we apply the Armijo rule to it

$$
\begin{aligned}
\alpha &= 1 \\
\text{while } P(\boldsymbol{x}_k + \alpha \Delta \boldsymbol{x}) &> P(\boldsymbol{x}_k) + b\alpha \nabla_{\boldsymbol{x}} P(\boldsymbol{x}_k)^T \Delta \boldsymbol{x} : \\
\alpha &= c\alpha \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha \Delta \boldsymbol{x}
\end{aligned}
$$

Constrained Minimization

Full constrained minimization:

$$\begin{aligned} \min_{\boldsymbol{x}} \; & f(\boldsymbol{x}) \\ \text{s.t.} \; & g(\boldsymbol{x}) \leqslant 0 \\ & h(\boldsymbol{x}) = 0 \end{aligned}$$

The **Lagrangian** is given:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T h(\boldsymbol{x}) + \boldsymbol{\mu}^T g(\boldsymbol{x})$$

First Order Necessary (KKT) Conditions:

1) $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \left(\frac{\partial h}{\partial \boldsymbol{x}}\right)^T \boldsymbol{\lambda} + \left(\frac{\partial g}{\partial \boldsymbol{x}}\right)^T \boldsymbol{\mu} = 0,$ "stationarity"

2) $g(\boldsymbol{x}) \leqslant 0$ and $h(\boldsymbol{x}) = 0,$ "primal feasibility"

3) $\boldsymbol{\mu} \geqslant 0,$ "dual feasibility"

4) $\boldsymbol{\mu}^T g(\boldsymbol{x}) = 0,$ "complementarity"

Usual Merit Functions

- KKT residual

$$P(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2} \| \begin{bmatrix} \nabla_{\boldsymbol{x}} \mathcal{L} \\ h(\boldsymbol{x}) \\ \min(0, g(\boldsymbol{x})) \end{bmatrix} \|^2$$

- Weighted Sum

$$P(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \frac{1}{2} w \| \begin{bmatrix} h(\boldsymbol{x}) \\ \min(0, g(\boldsymbol{x})) \end{bmatrix} \|^2$$

- Augmented Lagrangian

$$P(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathcal{L}_{\rho}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$$

Quadratic Programming (QP)

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x}$$
$$\text{s.t. } \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}$$
$$\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d} \leqslant \boldsymbol{0}$$

where $\boldsymbol{x}, \boldsymbol{q} \in \mathbb{R}^N$, $\boldsymbol{Q} \succ 0 \in \mathbb{R}^{N \times N}$. Let's solve this with the **Augmented Lagrangian Method:**

Quadratic Programming (QP)

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x}$$
$$\text{s.t. } \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}$$
$$\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d} \leqslant \boldsymbol{0}$$

where $\boldsymbol{x}, \boldsymbol{q} \in \mathbb{R}^N$, $\boldsymbol{Q} \succ 0 \in \mathbb{R}^{N\times N}$. Let's solve this with the **Augmented Lagrangian Method:**

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \boldsymbol{\mu}^T(\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 + \frac{\rho}{2}\|\max(\boldsymbol{0}, \boldsymbol{C}\boldsymbol{x} - \boldsymbol{d})\|^2$$

KKT Conditions:

1) $\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q} + \boldsymbol{A}^T\boldsymbol{\lambda} + \boldsymbol{C}^T\boldsymbol{\mu} = 0$
2) $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}$ and $\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d} \leqslant \boldsymbol{0}$
3) $\boldsymbol{\mu} \geqslant \boldsymbol{0}$
4) $\boldsymbol{\mu}^T(\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d}) = \boldsymbol{0}$

General Constrained Optimization

What other methods are there?

- **Active Set Methods**:
  - Try to guess which constraints are active (equal to zero) or not
  - Solve equality constrained optimization only with active constraints
  - Need good heuristic for identifying potentially active constraints
- **Barrier/Interior Point (IP) Methods:**
  - Add inequality constraints as "barriers"/penalties in the cost function. For example, $f'(\boldsymbol{x}) = f(\boldsymbol{x}) - \frac{1}{\rho}\log(-g(\boldsymbol{x}))$
  - Very good for convex problems

General Constrained Optimization (2)

What other methods are there?

- **Penalty Methods:**
  - Add constraint violation into the cost function. For example, $f'(\boldsymbol{x}) = f(\boldsymbol{x}) + \frac{\rho}{2}\|\max(\mathbf{0}, g(\boldsymbol{x}))\|^2$
  - Similar to IP methods, but there is a key difference: IP methods blow up at the boundary, penalty methods start pushing only if there is a violation!
  - Can easily get ill-conditions
  - Cannot get good accuracy
- **Sequential Quadratic Programming (SQP):**
  - Sequential QP problems
  - General idea:
    - Fit a quadratic approximation of the cost function
    - Linearize the constraints
    - Solve the produced QP with slack variables for feasibility
    - Iterate until convergence

Final Verdict on Optimization

- Newton's Method is very effective; Gauss-Newton the one used mostly in practice
- Transforming your problem into well known forms (e.g. QP) can greatly help
- Stochastic Gradient Descent is very effective for large scale or noisy problems
- Evolutionary Algorithms are very effective for non-smooth problems, but do not scale easily on large scale problems
- A lot of good libraries

Deterministic Optimal Control

Continuous Time:

$$\underset{\boldsymbol{x}(t),\boldsymbol{u}(t)}{\operatorname{argmin}} \mathcal{J}(\boldsymbol{x}(t),\boldsymbol{u}(t)) = \int_{t_0}^{t_f} \ell(\boldsymbol{x}(t),\boldsymbol{u}(t))dt + \ell_F(\boldsymbol{x}(t_f))$$

$$\text{s.t.} \qquad \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),\boldsymbol{u}(t))$$

where

- $\boldsymbol{x}(t) \in \mathbb{R}^N, \boldsymbol{u}(t) \in \mathbb{R}^M$ are the state and control trajectories
- $\mathcal{J}(\boldsymbol{x}(t),\boldsymbol{u}(t))$ is the "cost function"
- $\ell(\boldsymbol{x}(t),\boldsymbol{u}(t))$ is the "stage cost"
- $\ell_F(\boldsymbol{x}(t_f))$ is the "terminal cost"
- $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),\boldsymbol{u}(t))$ are the "dynamics constraints"
- We can potentially add more constraints (e.g. torque limits)

Deterministic Optimal Control (2)

- The continuous version is **“infinite dimensional”**

Deterministic Optimal Control (2)

- The continuous version is **"infinite dimensional"**
- The solution is open loop control trajectories!
- Only very few problems can be solved analytically

Deterministic Optimal Control (2)

- The continuous version is **"infinite dimensional"**
- The solution is open loop control trajectories!
- Only very few problems can be solved analytically
- Let's discretize!

Deterministic Optimal Control - Discrete

Discrete Time:

$$\underset{\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}}{\text{argmin}} \; \mathcal{J}(\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}) = \sum_{k=1}^{K-1} \ell(\boldsymbol{x}_k,\boldsymbol{u}_k) + \ell_F(\boldsymbol{x}_K)$$

$$\text{s.t.} \qquad \boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k,\boldsymbol{u}_k)$$

- $\boldsymbol{x}_k \in \mathbb{R}^N$ and $\boldsymbol{u}_k \in \mathbb{R}^M$ are vectors
- The is now **"finite dimensional"**
- The solution is open loop control trajectories!
- We usually call $\boldsymbol{x}_k, \boldsymbol{u}_k$ **"knot points"**

Linear Quadratic Regulator (LQR)

What is the Linear Quadratic Regulator (LQR) problem?

Linear Quadratic Regulator (LQR)

What is the Linear Quadratic Regulator (LQR) problem?

$$\underset{\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}}{\text{argmin}} \; \mathcal{J}(\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}) = \sum_{k=1}^{K-1}\left(\frac{1}{2}\boldsymbol{x}_k^T\boldsymbol{Q}_k\boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T\boldsymbol{R}_k\boldsymbol{u}_k\right) + \frac{1}{2}\boldsymbol{x}_K^T\boldsymbol{Q}_K\boldsymbol{x}_K$$

$$\text{s.t.} \quad \boldsymbol{x}_{k+1} = \boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k$$

$$\boldsymbol{Q}_k \succeq 0$$

$$\boldsymbol{R}_k \succ 0$$

- Widely used in many real applications
- The "workhorse" of optimal control
- We know "everything" about it!
- Infinite variations and extensions!
- **Time Invariant** if: $\boldsymbol{A}_k = \boldsymbol{A}, \boldsymbol{B}_k = \boldsymbol{B}, \boldsymbol{Q}_k = \boldsymbol{Q}, \boldsymbol{R}_k = \boldsymbol{R}, \forall k$

LQR as a Quadratic Programming Problem

Looking at the LQR problem, it really looks like a QP. Can we write it like one?

LQR as a Quadratic Programming Problem

Looking at the LQR problem, it really looks like a QP. Can we write it like one? Yes we can! We assume that $\boldsymbol{x}_1$ (initial conditions) is given, and define:

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{x}_K \end{bmatrix}, \boldsymbol{H} = \begin{bmatrix} \boldsymbol{R}_1 & & & & \\ & \boldsymbol{Q}_2 & & & \\ & & \boldsymbol{R}_2 & & \\ & & & \ddots & \\ & & & & \boldsymbol{Q}_K \end{bmatrix}$$

Now we can define:

$$\begin{aligned} \underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) &= \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H} \boldsymbol{z} \\ \text{s.t.} \quad & \text{“dynamics constraints”} \end{aligned}$$

LQR as a Quadratic Programming Problem (2)

For the dynamics constraints we have:

$$\underbrace{\begin{bmatrix} \boldsymbol{B}_1 & -\boldsymbol{I} & \boldsymbol{0} & \dots & & & \\ \boldsymbol{0} & \boldsymbol{A}_2 & \boldsymbol{B}_2 & -\boldsymbol{I} & \boldsymbol{0} & \dots & \\ & & & \ddots & & & \\ & & & & \boldsymbol{A}_{K-1} & \boldsymbol{B}_{K-1} & -\boldsymbol{I} \end{bmatrix}}_{\boldsymbol{G}} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{x}_K \end{bmatrix} = \underbrace{\begin{bmatrix} -\boldsymbol{A}_1\boldsymbol{x}_1 \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}}_{\boldsymbol{d}}$$

Now we have a full QP:

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z}$$

$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

LQR as a Quadratic Programming Problem (2)

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H}\boldsymbol{z}$$

$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

Can we solve this?

LQR as a Quadratic Programming Problem (2)

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H}\boldsymbol{z}$$
$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

Can we solve this? Of course we can! The Lagrangian is:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H}\boldsymbol{z} + \boldsymbol{\lambda}^T(\boldsymbol{G}\boldsymbol{z} - \boldsymbol{d})$$

LQR as a Quadratic Programming Problem (2)

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H}\boldsymbol{z}$$
$$\text{s.t.} \qquad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

Can we solve this? Of course we can! The Lagrangian is:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{z}^T \boldsymbol{H}\boldsymbol{z} + \boldsymbol{\lambda}^T(\boldsymbol{G}\boldsymbol{z} - \boldsymbol{d})$$

KKT Conditions:

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \boldsymbol{H}\boldsymbol{z} + \boldsymbol{G}^T\boldsymbol{\lambda} = \boldsymbol{0}$$
$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

LQR as a Quadratic Programming Problem (2)

$$\underset{\boldsymbol{z}}{\text{argmin}} \mathcal{J}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z}$$
$$\text{s.t.} \quad \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

Can we solve this? Of course we can! The Lagrangian is:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} + \boldsymbol{\lambda}^T(\boldsymbol{G}\boldsymbol{z} - \boldsymbol{d})$$

KKT Conditions:

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \boldsymbol{H}\boldsymbol{z} + \boldsymbol{G}^T\boldsymbol{\lambda} = \boldsymbol{0}$$
$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \boldsymbol{G}\boldsymbol{z} - \boldsymbol{d} = \boldsymbol{0}$$

KKT System:

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{G}^T \\ \boldsymbol{G} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{d} \end{bmatrix}$$

LQR as a Quadratic Programming Problem (2)

$$\underset{z}{\text{argmin}} \mathcal{J}(z) = \frac{1}{2} z^T H z$$
$$\text{s.t.} \qquad Gz - d = 0$$

Can we solve this? Of course we can! The Lagrangian is:

$$\mathcal{L}(z, \lambda) = \frac{1}{2} z^T H z + \lambda^T (Gz - d)$$

KKT Conditions:

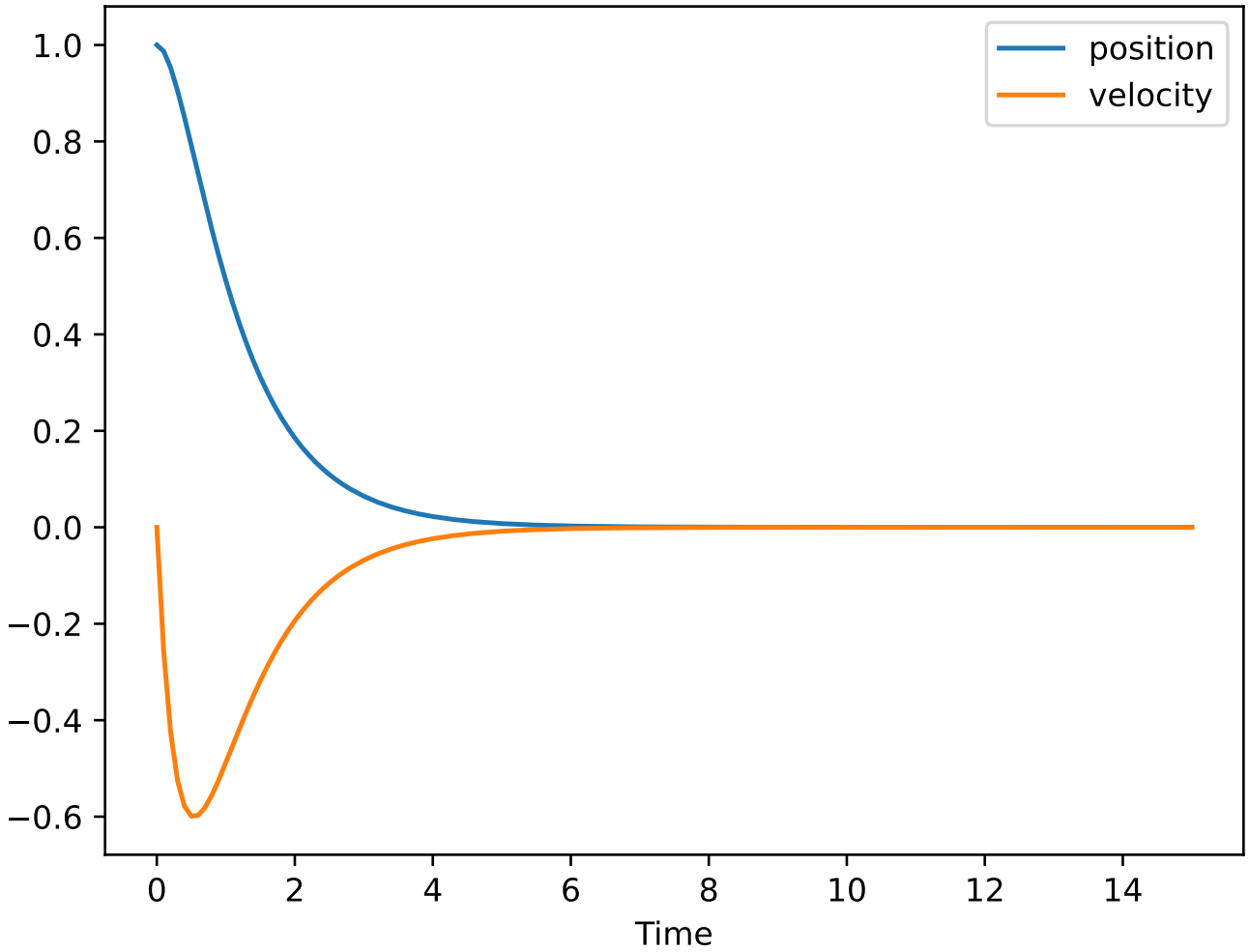
$$\nabla_z \mathcal{L} = Hz + G^T \lambda = 0$$
$$\nabla_\lambda \mathcal{L} = Gz - d = 0$$

KKT System:

$$\begin{bmatrix} H & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} z \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix}$$

We have the exact solution by solving ONE linear system!

LQR as a QP - Code Example



LQR via Riccati Recursion

Let's write the KKT System in "detail" (for $K = 4$):

$$
\left[\begin{array}{cccccccc}
R_1 & 0 & \cdots & & 0 & B_1^T & \cdots & 0 \\
0 & Q_2 & 0 & \cdots & 0 & -I & A_2^T & \vdots \\
\vdots & 0 & R_2 & 0 & \cdots & 0 & B_2^T & \\
 & \vdots & 0 & Q_3 & 0 & \cdots & -I & A_3^T \\
 & & \vdots & 0 & R_3 & 0 & \cdots & B_3^T \\
0 & 0 & & \vdots & 0 & Q_4 & & -I \\
\hdashline
B_1 & -I & 0 & 0 & \vdots & 0 & \ddots & \vdots \\
\vdots & A_2 & B_2 & -I & 0 & \vdots & & \\
0 & \cdots & & A_3 & B_3 & -I & \cdots & 0
\end{array}\right]
\left[\begin{array}{c} u_1 \\ x_2 \\ u_2 \\ x_3 \\ u_3 \\ x_4 \\ \hdashline \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{array}\right]
=
\left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \hdashline -A_1 x_1 \\ 0 \\ 0 \end{array}\right]
$$

LQR via Riccati Recursion (2)

We first solve the last line of the upper block:

$$\boldsymbol{Q}_4\boldsymbol{x}_4 - \boldsymbol{\lambda}_4 = \boldsymbol{0} \Rightarrow \boldsymbol{\lambda}_4 = \boldsymbol{Q}_K\boldsymbol{x}_4$$

Then we move one line up:

$$\boldsymbol{R}_3\boldsymbol{u}_3 + \boldsymbol{B}_3^T\boldsymbol{\lambda}_4 = \boldsymbol{R}_3\boldsymbol{u}_3 + \boldsymbol{B}_3^T\boldsymbol{Q}_K\boldsymbol{x}_4 = \boldsymbol{0}$$

$$\Rightarrow \boldsymbol{R}_3\boldsymbol{u}_3 + \boldsymbol{B}_3^T\boldsymbol{Q}_K(\boldsymbol{A}_3\boldsymbol{x}_3 + \boldsymbol{B}_3\boldsymbol{u}_3) = \boldsymbol{0}$$

$$\Rightarrow \boldsymbol{u}_3 = -\underbrace{(\boldsymbol{R}_3 + \boldsymbol{B}_3^T\boldsymbol{Q}_K\boldsymbol{B}_3)^{-1}\boldsymbol{B}_3^T\boldsymbol{Q}_K\boldsymbol{A}_3}_{\boldsymbol{K}_3}\boldsymbol{x}_3$$

One more line:

$$\boldsymbol{Q}_3\boldsymbol{x}_3 - \boldsymbol{\lambda}_3 + \boldsymbol{A}_3^T\boldsymbol{\lambda}_4 = \boldsymbol{0} \Rightarrow \boldsymbol{Q}_3\boldsymbol{x}_3 - \boldsymbol{\lambda}_3 + \boldsymbol{A}_3^T\boldsymbol{Q}_K\boldsymbol{x}_4 = \boldsymbol{0}$$

$$\Rightarrow \boldsymbol{Q}_3\boldsymbol{x}_3 - \boldsymbol{\lambda}_3 + \boldsymbol{A}_3^T\boldsymbol{Q}_K(\boldsymbol{A}_3\boldsymbol{x}_3 + \boldsymbol{B}_3\boldsymbol{u}_3) = \boldsymbol{0}$$

$$\Rightarrow \boldsymbol{\lambda}_3 = \underbrace{\left(\boldsymbol{Q}_3 + \boldsymbol{A}_3^T\boldsymbol{Q}_K(\boldsymbol{A}_3 - \boldsymbol{B}_3\boldsymbol{K}_3)\right)}_{\boldsymbol{P}_3}\boldsymbol{x}_3$$

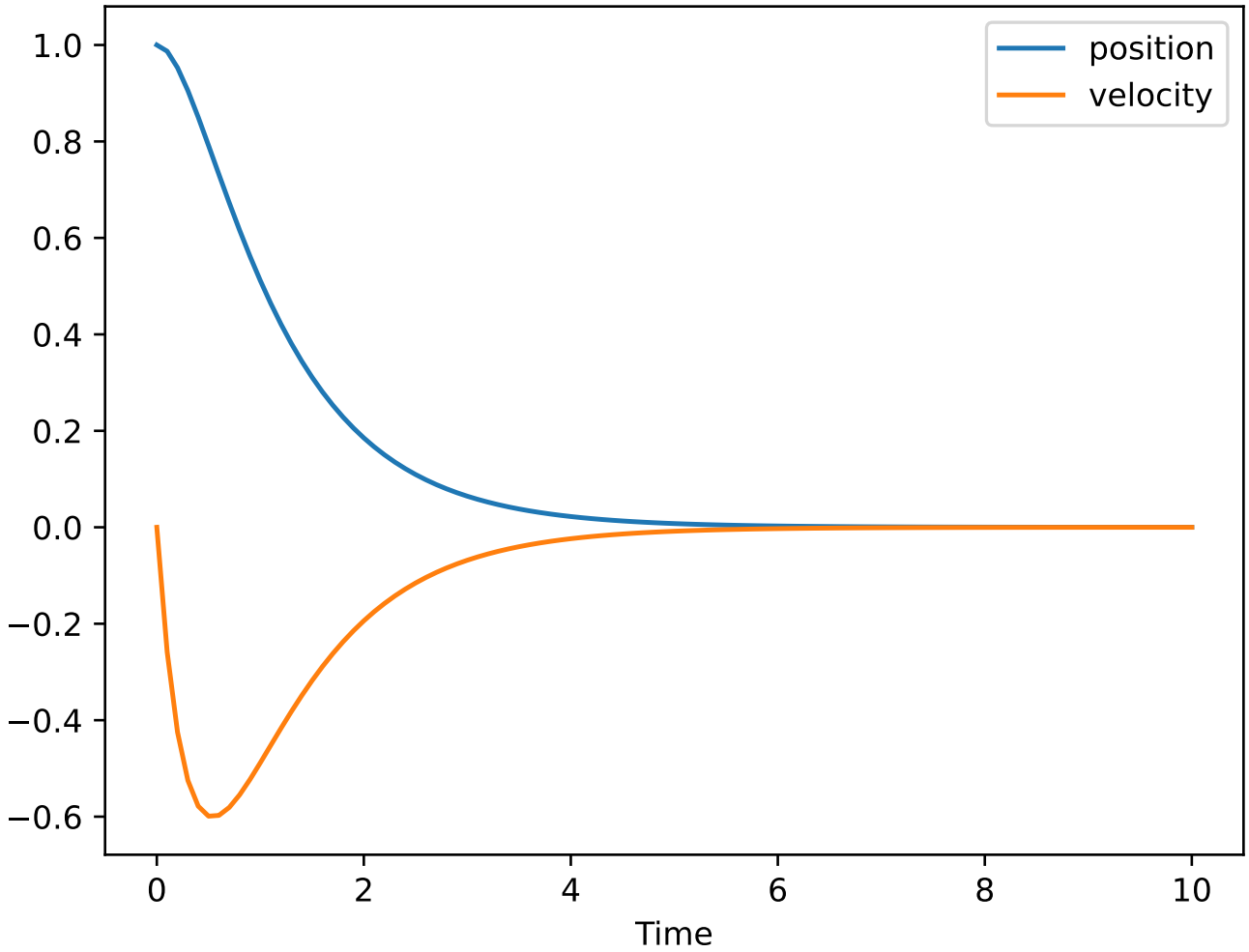
LQR via Riccati Recursion (3)

The recursion is now revealed:

$$\boldsymbol{P}_K = \boldsymbol{Q}_K$$
$$\boldsymbol{K}_k = (\boldsymbol{R}_k + \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{B}_k)^{-1} \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{A}_k$$
$$\boldsymbol{P}_k = \boldsymbol{Q}_k + \boldsymbol{A}_k^T \boldsymbol{P}_{k+1} (\boldsymbol{A}_k - \boldsymbol{B}_k \boldsymbol{K}_k)$$

- QP has complexity $O(K^3(N+M)^3)$
- Riccati recursion has complexity $O(K(N+M)^3)$
- The Riccati recursion is the optimal exploitation of the sparsity of the KKT system!
- **We also get for free a feedback policy: $\boldsymbol{u} = -\boldsymbol{K}\boldsymbol{x}$!**

LQR via Riccati Recursion - Code Example



Thank you

Any Questions?