RBE502:Robot DevOps

Introduction to Continuous Optimization

by

Pi Thanacha Choopojcharoen

Agenda

- Preliminary
  - Vector, Matrices, Jacobian, Gradient, Hessian
- Purpose
- Structure
  - Design variables
  - Objective
  - Constraints

Preliminary

Preliminary : Set

A set is a collection of all possible values of a variable.

Example:

A set of all positive odd number $\mathcal{A}$ can be expressed as follows.

$$\mathcal{A} = \{1,3,5,7, \dots\} = \{2k - 1 | k \in \mathbb{Z}^+\}$$

Common Sets:

$\mathbb{N}^+$: Natural number (start with 1) [1-indexing]

$\mathbb{N}^0$: Natural number (start with 0) [0-indexing]

$\mathbb{Z}$: Integer

$\mathbb{R}^+$: Positive real number

$\mathbb{R}^+ \cup \{0\}, \mathbb{R}_{\geq 0}$: Non-negative real number

$\mathbb{R}$: Real number

$\mathbb{C}$: complex number

Preliminary : Cartesian Product

A Cartesian product of 2 sets is a combination of values of each sets.

Given a variable $a \in \mathcal{A}$ and $b \in \mathcal{B}$, the cartesian product between their sets are the following.

$$\mathcal{S} = \mathcal{A} \times \mathcal{B}$$

We can write a pair of variables in a form of vector $\boldsymbol{s}$ and express it as follows.

$$\mathbf{s} = \begin{bmatrix} a \\ b \end{bmatrix} \in \mathcal{A} \times \mathcal{B}$$

$\boldsymbol{s}$ is a collection of variables. Each component can take any values in their corresponding set.

Preliminary : Vector

A vector, $\mathbf{x}$, represented by bold straight lowercase font, is a list of one or more components (order does matter).

If $x_i \in \mathcal{X}_i \; \forall \; i = 0, \dots, n$,
then

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = [x_1 \quad x_2 \quad \cdots \quad x_n]^\top \in \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$$

“numpy” represents this is a form of “array”

Preliminary : Vector Operation

(Frobenius) Norm $\|\cdot\|_2$:

If $v_i \in \mathbb{V}_i$ and a vector $\mathbf{v} \in \mathbb{V}_1 \times \mathbb{V}_2 \times \cdots \times \mathbb{V}_n$

$$\|\mathbf{v}\|_2 = \sqrt{\mathbf{v}^\top \mathbf{v}} = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Example : Set of 3D-position

$$\mathbf{p} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}$$

Example : Set of 3D-position

$$\mathbf{p} \in \mathbb{R}^3$$

Array & vector with Python

```python
import numpy as np

math_object =dict()
math_object['array'] = np.array([1,2,3])
math_object['row vector'] = np.array([[1,2,3]])
math_object['column vector'] = np.array([[1,2,3]]).T

for key,value in math_object.items():
    print(f"Shape of {key}: {value.shape}")
```

Array & vector with Python

```
Shape of array: (3,)
Shape of row vector: (1, 3)
Shape of column vector: (3, 1)
```

```
import numpy as np

math_object =dict()
math_object['array'] = np.array([1,2,3])
math_object['row vector'] = np.array([[1,2,3]])
math_object['column vector'] = np.array([[1,2,3]]).T

for key,value in math_object.items():
    print(f"Shape of {key}: {value.shape}")
```

Array & vector with Python

```
Shape of array: (3,)
~~Shape of row vector: (1, 3)~~
Shape of column vector: (3, 1)
```

```python
import numpy as np

math_object =dict()
math_object['array'] = np.array([1,2,3])
~~math_object['row vector'] = np.array([[1,2,3]])~~
math_object['column vector'] = np.array([[1,2,3]]).T

for key,value in math_object.items():
    print(f"Shape of {key}: {value.shape}")
```

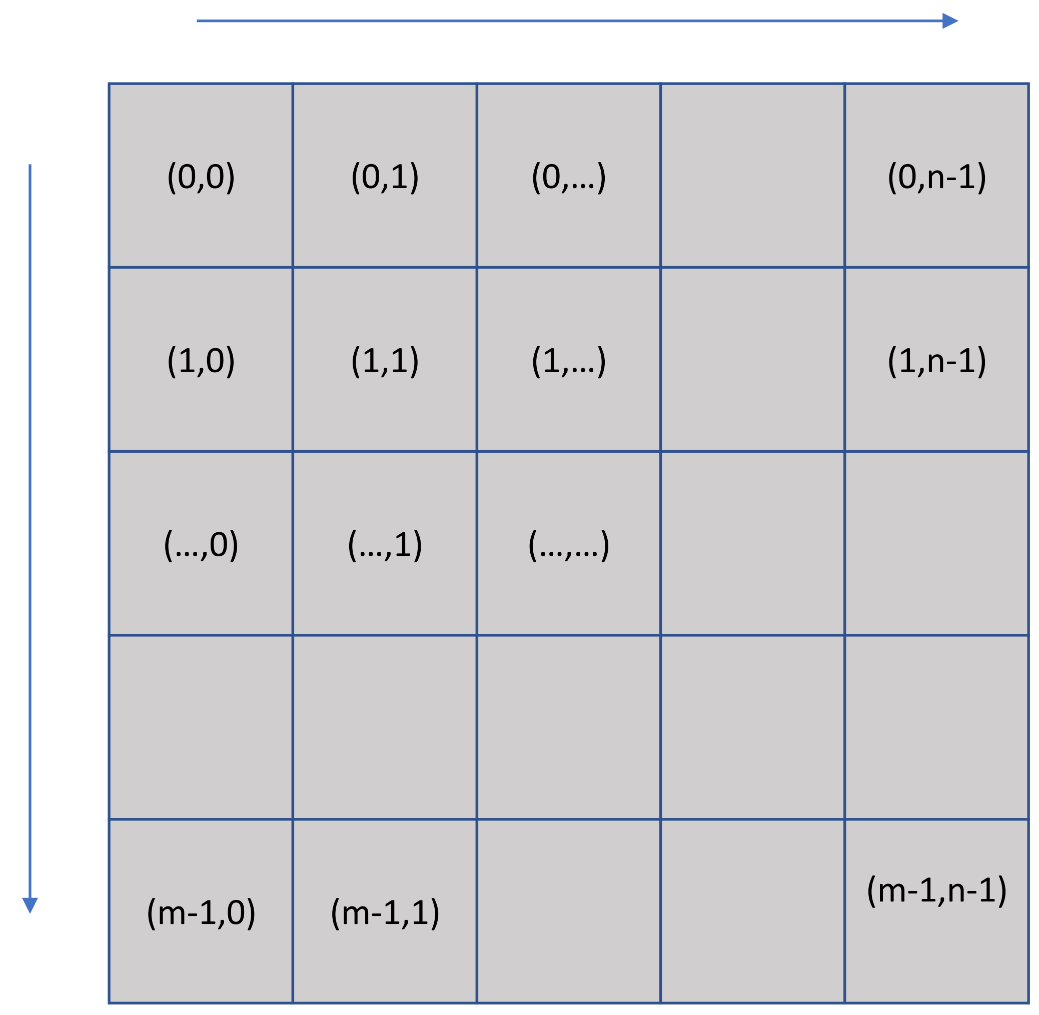
Example : 2D grid

Given a 2D finite grid, the position of the cell is indicated by a pair of indices $\mathbf{a}$.

$$\mathbf{a} = \begin{bmatrix} a_r \\ a_c \end{bmatrix}$$

There are only m rows and n columns. Both indices starts at zero and strictly increases to their maximum size.

$$\mathcal{A}_k = \{a \in \mathbb{N}_{\geq 0} | a < k\}$$



$$\mathbf{a} \in \mathcal{A}_m \times \mathcal{A}_n$$

Meshgrid with Python

```
m = 4
n = 5
A_r = range(m)
A_c = range(n)
A = []
for j in range(n):
    for i in range(m):
        A.append((A_r[i],A_c[j]))
# same effect
import numpy as np
a_r,a_c = np.meshgrid(A_r,A_c)
A = list(zip(a_r.reshape((1,-1))[0],a_c.reshape((1,-1))[0]))
```

Meshgrid with Python

```
m = 4
n = 5
A_r = range(m)
A_c = range(n)
A = []
for j in range(n):
    for i in range(m):
        A.append((A_r[i],A_c[j]))
# same effect
import numpy as np
a_r,a_c = np.meshgrid(A_r,A_c)
A = list(zip(a_r.reshape((1,-1))[0],a_c.reshape((1,-1))[0]))
```

```
[(0, 0),
(1, 0),
(2, 0),
(3, 0),
(0, 1),
(1, 1),
(2, 1),
(3, 1),
(0, 2),
(1, 2),
(2, 2),
(3, 2),
(0, 3),
(1, 3),
(2, 3),
(3, 3),
(0, 4),
(1, 4),
(2, 4),
(3, 4)]
```

Preliminary : Matrix

A matrix, $\mathbf{A}$, represented by bold straight uppercase font, is a 2-dimensional list one or more components (order does matter).

Given that $\mathbf{a}_i = [a_{i1} \quad a_{i2} \quad \cdots \quad a_{in}]^\top$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_\mathrm{m}^\top \end{bmatrix}$$

Preliminary : Common Matrices

$\mathbb{O}_{m\times n}$: matrix of size $m \times n$ with zero as its elements

```
>> np.zeros((m,n))
```

$\mathbf{1}_{m\times n}$: matrix of size $m \times n$ with ones as its elements

```
>> np.ones((m,n))
```

$\mathbb{I}_n$: an identity matrix of size $n \times n$

```
>> np.eye(n)
```

Preliminary : Common Matrix Operation

Matrix product

$$\mathbf{C} = \mathbf{A}\mathbf{B}$$

```
>> C = A@B
```

Hadamard (element-wise) product

$$\mathbf{C} = \mathbf{A} \circ \mathbf{B}$$

```
>> C = A*B
```

Transpose

$$\mathbf{B} = \mathbf{A}^{\top}$$

```
>> B= A.T
```

Inverse

$$\mathbf{B} = \mathbf{A}^{-1}$$

```
>> B= np.inv(A)
```

Matrix with Python

```
import numpy as np

A = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(A)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Preliminary : function

A general mathematic function $\mathbf{f}(\cdot)$ maps multiple inputs $\mathbf{x} \in \mathcal{X}$ to multiple outputs $\mathbf{y} \in \mathcal{Y}$

$$\mathbf{f}: \mathcal{X} \rightarrow \mathcal{Y}$$

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

Or

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2, \cdots, x_n) \\ f_2(x_1, x_2, \cdots, x_n) \\ \vdots \\ f_m(x_1, x_2, \cdots, x_n) \end{bmatrix}$$

** A ***<u>real-value function</u>*** is a function that can maps multiple inputs to a single value defined as follows

$$\mathbf{f}: \mathcal{X} \rightarrow \mathbb{R}$$

Example: Differential Drive Odometry

Given a position vector $\mathbf{p}$, we can compute the Euclidean distance to the origin using the following relationship.

$$\mathbf{f}(\mathbf{x},\mathbf{u}) = \begin{bmatrix} u_1 x_3 \\ u_1 x_4 \\ -u_2 x_4 \\ u_2 x_3 \end{bmatrix}$$

$$f: \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}^4$$

$$\mathcal{X} = \mathbb{R}^2 \times \mathbb{S}^1, \qquad \mathcal{U}_{\mathrm{i}} = \{u \in \mathbb{R} | |u| \leq \mathrm{i}_{\max}\}, \qquad \mathcal{U} = \mathcal{U}_v \times \mathcal{U}_{\boldsymbol{\omega}}$$

$$\mathbf{x} = \begin{bmatrix} p_x \\ p_y \\ c \\ s \end{bmatrix} \in \mathcal{X}, \qquad \mathbf{u} = \begin{bmatrix} v \\ \omega \end{bmatrix} \in \mathcal{U}$$

Example: Euclidean Distance

Given a position vector $\mathbf{p}$, we can compute the Euclidean distance to the origin using the following relationship.

$$f(\mathbf{p}) = \sqrt{\mathbf{p}^\top \mathbf{p}}$$

$$f\colon \mathbb{R}^3 \rightarrow \mathbb{R}_{\geq 0}$$

```
>> np.linalg.norm(p)
```

Preliminary: Jacobian Matrix

Given a function $\mathbf{f}: \mathcal{X} \rightarrow \mathcal{Y}$,

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

a Jacobian matrix of $\mathbf{f}$, with respect to $\mathbf{x}$ is defined as the following.

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}} \\ \frac{\partial f_2}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} : \mathcal{X} \rightarrow \mathbb{R}^{m \times \mathrm{n}}$$

Preliminary: Jacobian Matrix of real-value function

Given a real-value function $f: \mathcal{X} \to \mathbb{R}$,

$$y = f(\mathbf{x})$$

a Jacobian matrix of f , with respect to $\mathbf{x}$ is

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix} : \mathcal{X} \to \mathbb{R}^{1\times\mathrm{n}}$$

In some fields, it is interchangeable with “gradient”.

Preliminary: Gradient of a function

Given a real-value function $f : \mathcal{X} \to \mathbb{R}$,

$$y = f(\mathbf{x})$$

a gradient of f , with respect to $\mathbf{x}$ is

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left(\frac{\partial f}{\partial \mathbf{x}}\right)^{\top} : \mathcal{X} \to \mathbb{R}^{\mathrm{n}}$$

Jacobian of real-value function is a row vector.

Gradient of a function is a column vector.

Example : Jacobian of LTI system

Given a LTI system in the form of

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{A} \in \mathbb{R}^{n \times n}$$

The Jacobian of $\mathbf{f}$ w.r.t. to $\mathbf{x}$ is

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) = \frac{\partial}{\partial \mathbf{x}}(\mathbf{A}\mathbf{x}) = \mathbf{A}$$

The Jacobian of $\mathbf{f}$ w.r.t. to $\mathbf{u}$ is

$$\frac{\partial \mathbf{f}}{\partial \mathbf{u}} = \frac{\partial}{\partial \mathbf{u}}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) = \frac{\partial}{\partial \mathbf{u}}(\mathbf{B}\mathbf{u}) = \mathbf{B}$$

Example : Quadratic cost

Given a vector of errors $\mathbf{e} \in \mathbb{R}^n$, a quadratic cost is a sum of square of its components.

$$f(\mathbf{e}) = e_1^2 + e_2^2 + \cdots e_n^2 = \mathbf{e}^\top \mathbf{e}$$

The Jacobian of the quadratic cost with respect to the errors is

$$\frac{\partial f}{\partial \mathbf{e}} = \frac{\partial f}{\partial \mathbf{e}}(\mathbf{e}^\top \mathbf{e}) = 2\mathbf{e}^\top$$

The gradient of the quadratic cost is

$$\nabla_{\mathbf{e}} f(\mathbf{e}) = 2\mathbf{e}$$

What is the Jacobian of the gradient of the cost ?

$$\frac{\partial f}{\partial \mathbf{e}}\left(\nabla_{\mathbf{e}} f(\mathbf{e})\right) = ?$$

What is the Jacobian of the gradient of the cost ?

$$\frac{\partial}{\partial \mathbf{e}}\left(\nabla_{\mathbf{e}} f(\mathbf{e})\right) = \frac{\partial}{\partial \mathbf{e}}(2\mathbf{e}) = 2\frac{\partial \mathbf{e}}{\partial \mathbf{e}} = 2\mathbb{I}_n = \begin{bmatrix} 2 & 0 & \cdots & 0 \\ 0 & 2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 2 \end{bmatrix} \in \mathbb{R}^{n\times n}$$

Preliminary: Hessian Matrix

Given a real-value function $f : \mathcal{X} \rightarrow \mathbb{R}$,

$$y = f(\mathbf{x})$$

a Hessian matrix of f , with respect to $\mathbf{x}$ is the Jacobian of the gradient

$$\nabla_{\mathbf{x}}^2 f(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} \left(\nabla_{\mathbf{x}} f(\mathbf{x}) \right) : \mathcal{X} \rightarrow \mathbb{R}^{\mathrm{n} \times n}$$

Preliminary: Common Derivatives

$$\frac{\partial}{\partial \mathbf{x}}(\mathbf{A}\mathbf{x}) = \mathbf{A}$$

$$\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}^\top \mathbf{A}\mathbf{x}) = \mathbf{x}^\top(\mathbf{A} + \mathbf{A}^\top)$$

$$\nabla_{\mathbf{x}}^2(\mathbf{x}^\top \mathbf{A}\mathbf{x}) = \mathbf{A} + \mathbf{A}^\top$$

Preliminary: Linear Approximation

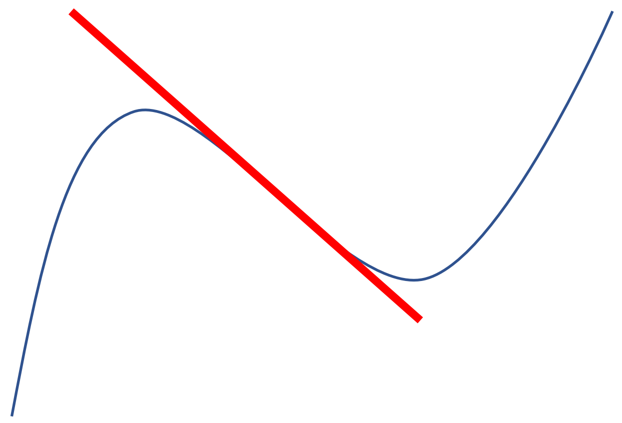
Given a real-value function $f: \mathcal{X} \to \mathbb{R}$,

$$y = f(\mathbf{x})$$

A linear approximation of $f(\mathbf{x})$ at $\mathbf{x}^*$ is the followings.

$$y \approx \mathbf{f}(\mathbf{x}^*) + \mathbf{J} \cdot (\mathbf{x} - \mathbf{x}^*)$$

$$\mathbf{J} = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^*}$$

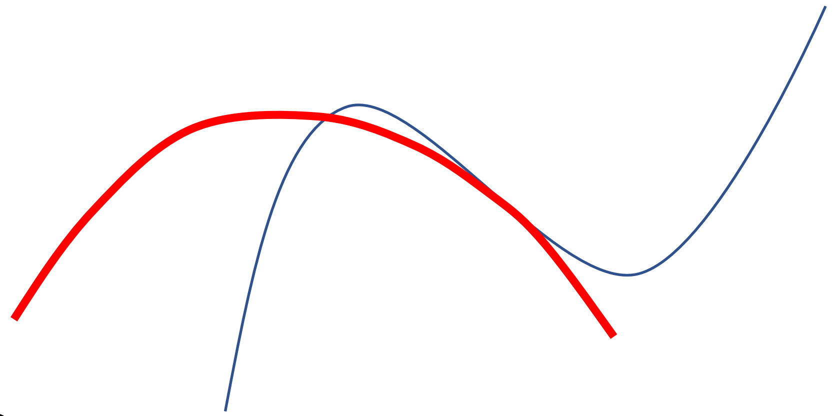


Preliminary: Quadratic Approximation

Given a real-value function $f: \mathcal{X} \to \mathbb{R}$,

$$y = f(\mathbf{x})$$

A quadratic approximation of $f(\mathbf{x})$ at $\mathbf{x}^*$ is the followings.



$$y \approx \mathbf{f}(\mathbf{x}^*) + \mathbf{J} \cdot (\mathbf{x} - \mathbf{x}^*) + (\mathbf{x} - \mathbf{x}^*)^\top \mathbf{H} \cdot (\mathbf{x} - \mathbf{x}^*)$$

$$\mathbf{J} = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^*}, \qquad \mathbf{H} = \nabla_{\mathbf{x}}^2 f(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}$$

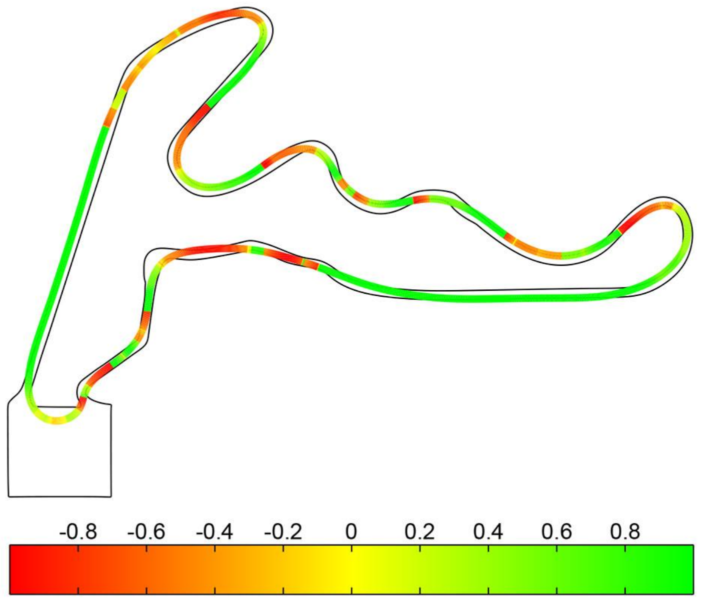
Purpose

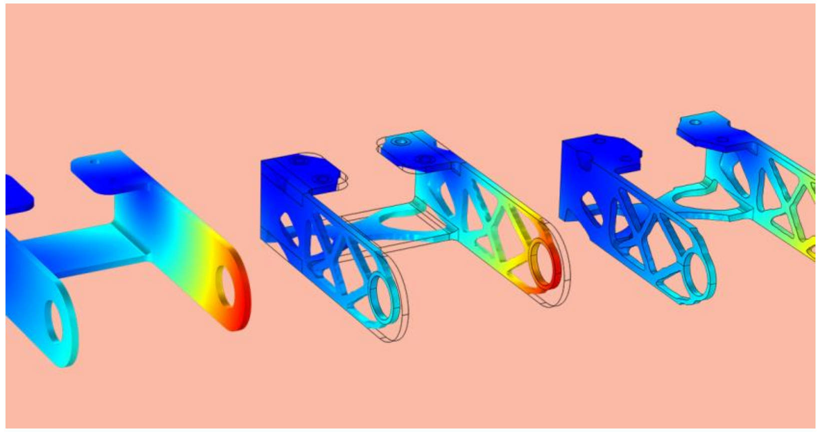
Optimization

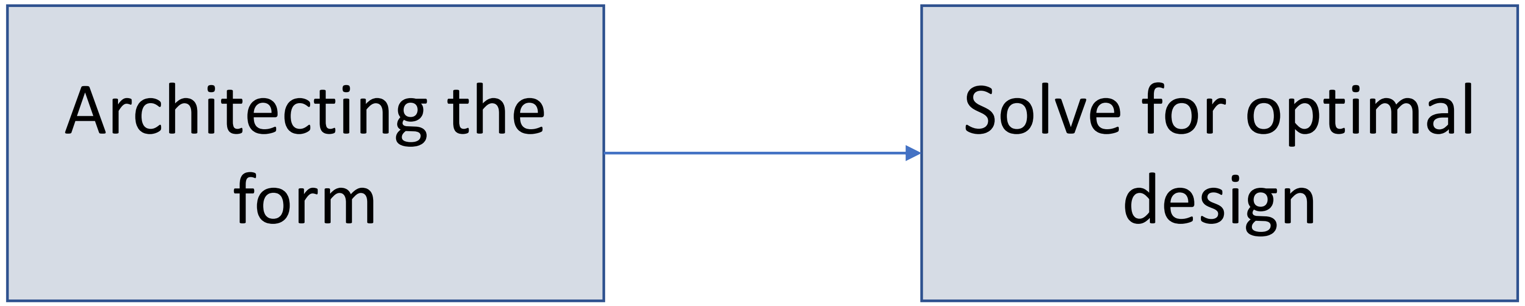
"**Mathematical optimization** or **mathematical programming** is the selection of a best element, with regard to some criterion, from some set of available alternatives"

For example:

- Best time to hit the brake
- Minimize materials while maintain structural integrity
- Best model parameter that minimize the loss function







- Identify goals
- Identify constraints
- Formulate problems

- Solving methods (Solvers)

Structure

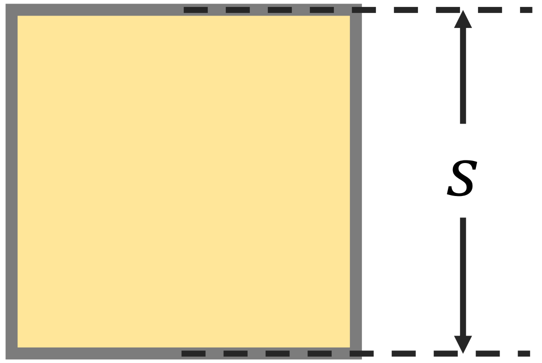
Structure of Optimization Problem

1. Design Variables
2. Objective Function
3. Constraints*

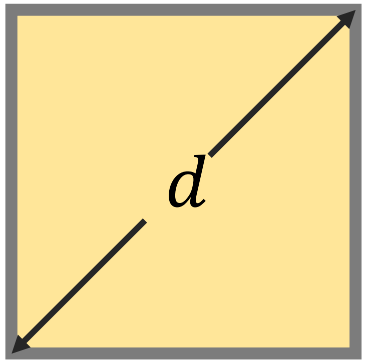
Design Variables (Decision Variables)

- Things that we try to choose
- A set of design variables can be generated by the concept generation process

Example : Designing a square room



$$A(s) = s^2$$
$$P(s) = 4s$$



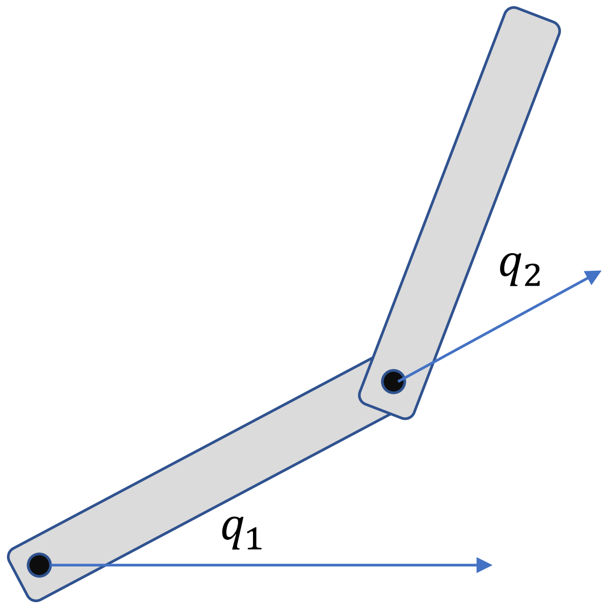
$$A(d) = 0.5d^2$$
$$P(s) = 2\sqrt{2}d$$

Example :

Given an equation of forward kinematics of 2DOF planar manipulator

$$\mathbf{p} = \begin{bmatrix} l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) \\ l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \end{bmatrix}$$

What is the decision variables ?



Example : Parameter Estimation

Given an equation of forward kinematics of 2DOF planar manipulator

$$\mathbf{p}(\mathbf{q}) = \begin{bmatrix} l_1\cos(q_1) + l_2\cos(q_1 + q_2) \\ l_1\sin(q_1) + l_2\sin(q_1 + q_2) \end{bmatrix}$$

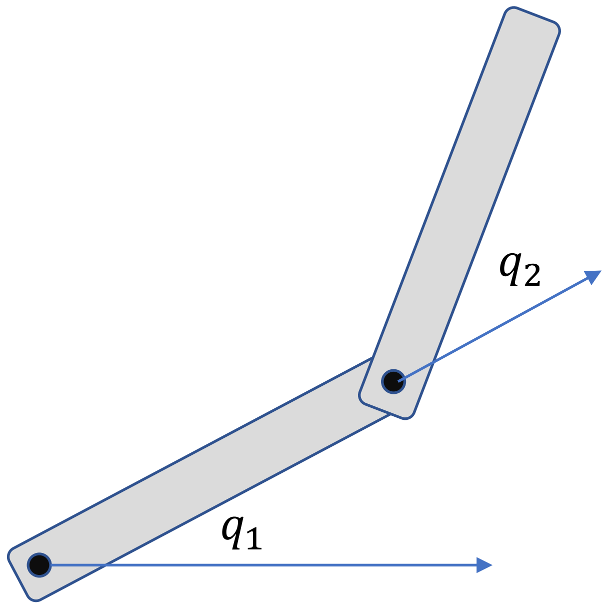
And given a data set $\langle \mathbf{q}_i, \mathbf{p}_i \rangle$

Decision variables

$$l_1, \qquad l_2$$

$$\mathbf{p}(\mathbf{q}) = \begin{bmatrix} \cos(q_1) & \cos(q_1 + q_2) \\ \sin(q_1) & \sin(q_1 + q_2) \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \end{bmatrix}$$

$$\mathbf{e} = \mathbf{p}(\mathbf{q}_i : \langle l_1, l_2 \rangle) - \mathbf{p}_i$$



Example : Inverse Kinematics

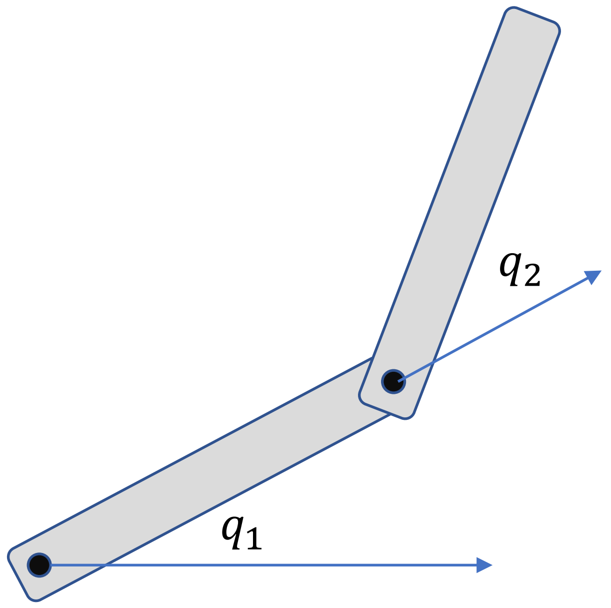
Given an equation of forward kinematics of 2DOF planar manipulator

$$\mathbf{p}(\mathbf{q}) = \begin{bmatrix} l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) \\ l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \end{bmatrix}$$

And given a position $\mathbf{p}$, find $\mathbf{q}$

Decision variables

$$q_1, \qquad q_2$$



Objective Function

- How can we measure that our design is the best
- A real-value function that maps the design variables to a real value

Let $\mathbf{w} \in \mathcal{W}$ be the design variables, an objective function in term of the design variables is defined as follows.

$$f : \mathcal{W} \to \mathbb{R}$$

Objective Function

- How can we measure that our design is the best
- A real-value function that maps the design variables to a real value
- Sometimes called Cost function

Let $\mathbf{w} \in \mathcal{W}$ be the design variables, an objective function in term of the design variables is defined as follows.

$$f : \mathcal{W} \to \mathbb{R}$$

$$f^* = \max_{\mathbf{w} \in \mathcal{W}} f(\mathbf{w})$$

Maximization problem

$$f^* = \min_{\mathbf{w} \in \mathcal{W}} f(\mathbf{w})$$

Minimization problem

Unconstrained Optimization

Searching for the optimizer $\mathbf{w}^*$ that minimizes (or maximize) the objective function without any constraints.

$$f^* = \min_{\mathbf{w} \in \mathcal{W}} f(\mathbf{w}) \qquad \mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w} \in \mathcal{W}} f(\mathbf{w})$$

Optimal cost

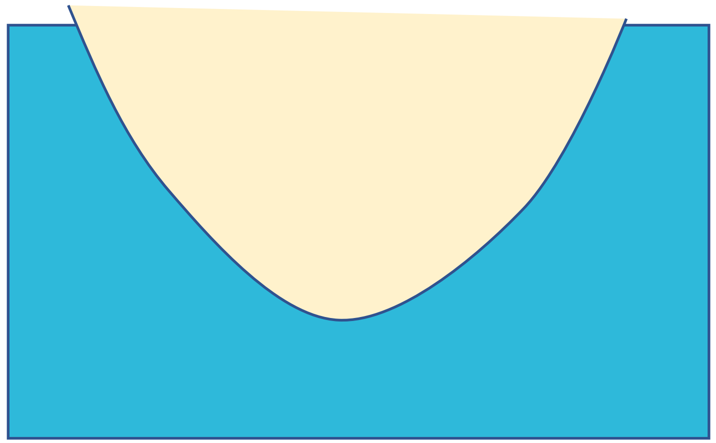
Optimizer

$$f^* = f(\mathbf{w})$$

Example: Best photograph spot

The expedition team is divided into 3 teams, the marine biologists who track the movement of whales, the photographers who are tasked to take the "best" photo of the whales, and the surveyors whose jobs are to look for the "best" photo-taking spot.

It's your first day of working for the surveyor team. After a quick run on the beach, you obtain the first set of data, the estimated map of the beach, which can be approximated as a parabola.



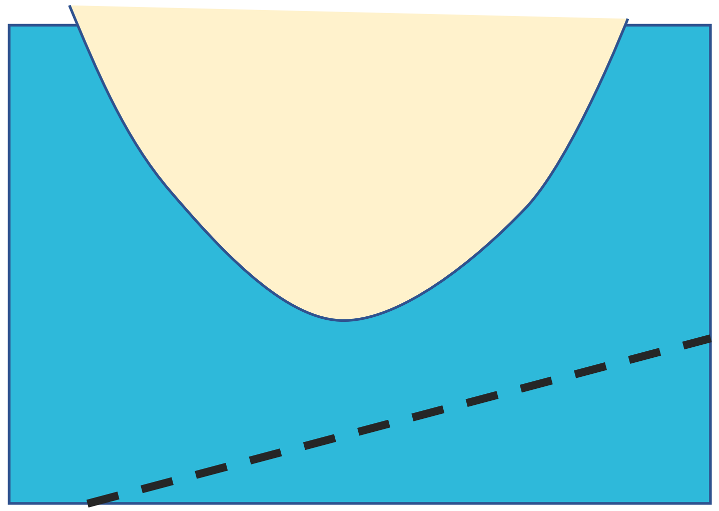
$$y_p(x_p) = \frac{1}{2}x_p^2 - 2$$

$\langle x_p, y_p \rangle$: represents a spot on the edge of the beach

Example: Best photograph spot

The marine biologist team has recently discovered a beluga traveling in a straight and constant trajectory. Its path can be mapped to a linear path.

What is the best photographing spot on the beach ?



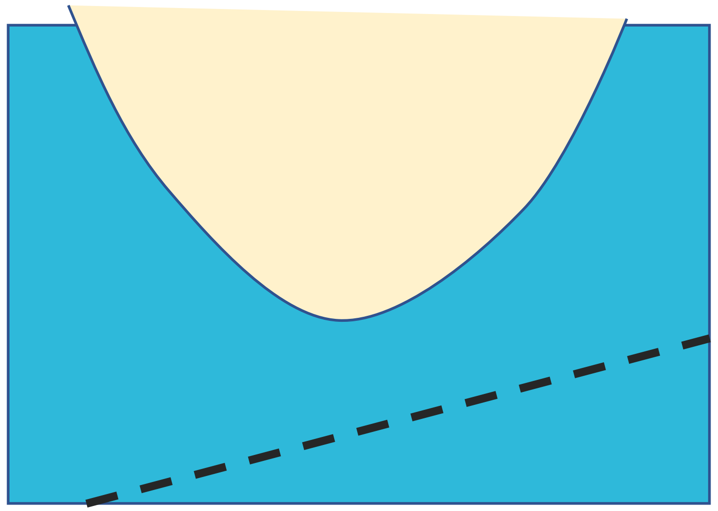
$$y_p(x_p) = \frac{1}{2}x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2}x_w - 5$$

Example: Best photograph spot

What is the best photographing spot on the beach ?

- What does it mean to be the best?
- What are the design variables?
- What is the objective function?



$$y_p(x_p) = \frac{1}{2}x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2}x_w - 5$$

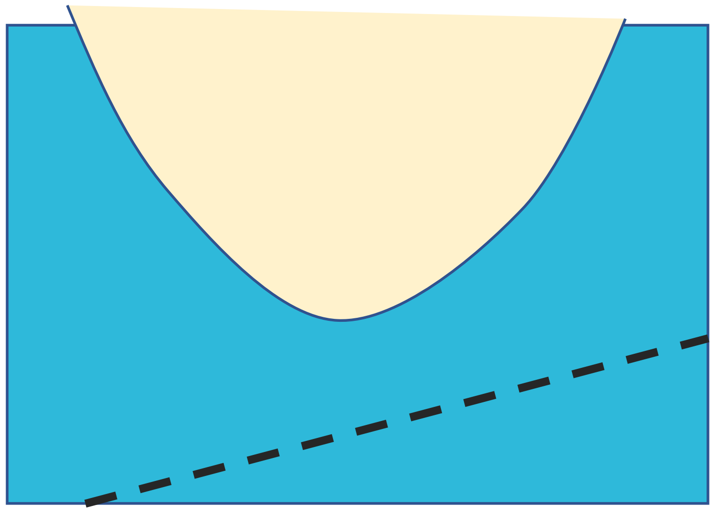
Example: Best photograph spot

Shortest distance between the edge of the beach and the whale’s position

Let x_p denotes the x-position of the photographer on the edge of the beach and

x_w denotes the x-position of the whale along its path.

$$f(x_p, x_w) = \sqrt{(x_p - x_w)^2 + \left(y_p(x_p) - y_w(x_w)\right)^2}$$



$$y_p(x_p) = \frac{1}{2}x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2}x_w - 5$$

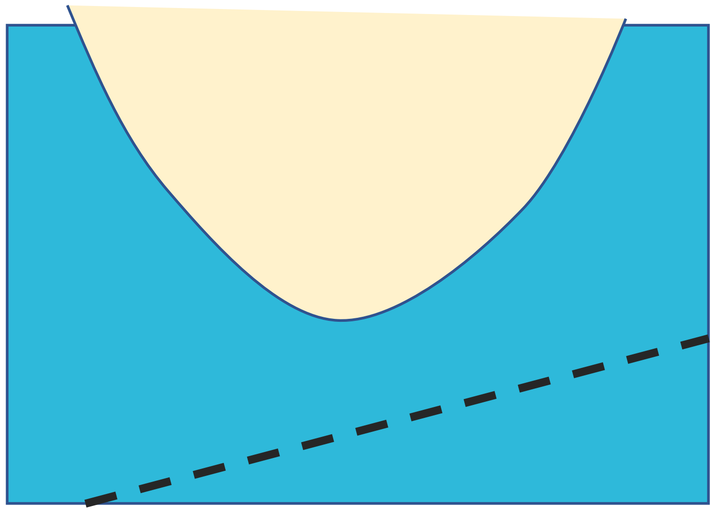
Example: Best photograph spot

Shortest distance between the edge of the beach and the whale's position

Let x_p denotes the x-position of the photographer on the edge of the beach and

x_w denotes the x-position of the whale along its path.

$$f(x_p, x_w) = (x_p - x_w)^2 + \left(y_p(x_p) - y_w(x_w)\right)^2$$



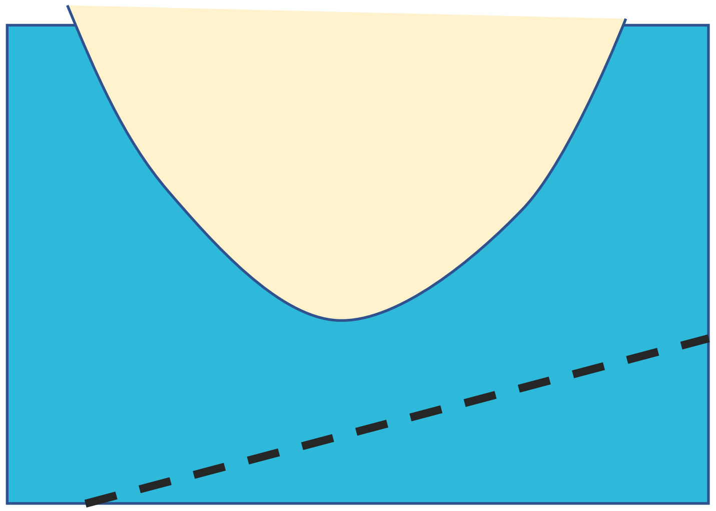
$$y_p(x_p) = \frac{1}{2}x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2}x_w - 5$$

Example: Best photograph spot

The optimization problem may look like this.

$$\langle x_p^*, x_w^* \rangle = \underset{\substack{x_p \in \mathbb{R} \\ x_w \in \mathbb{R}}}{\operatorname{argmin}} \left(x_p - x_w\right)^2 + \left(\frac{1}{2} x_p^2 - \frac{1}{2} x_w + 3\right)^2$$



$$y_p\left(x_p\right) = \frac{1}{2} x_p^2 - 2$$

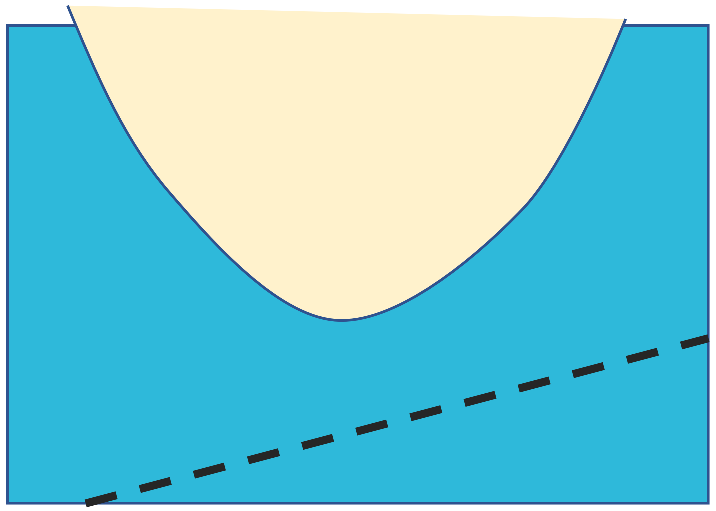
$$y_w(x_w) = \frac{1}{2} x_w - 5$$

Example: Best photograph spot

The optimization problem may look like this.

$$\langle x_p^*, x_w^* \rangle = \underset{\substack{x_p \in \mathbb{R} \\ x_w \in \mathbb{R}}}{\operatorname{argmin}} \left(x_p - x_w\right)^2 + \left(\frac{1}{2}x_p^2 - \frac{1}{2}x_w + 3\right)^2$$

$$\nabla_{\mathbf{w}} f(\mathbf{w})\Big|_{\mathbf{w}=\mathbf{w}^*} = 0$$



$$y_p(x_p) = \frac{1}{2}x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2}x_w - 5$$

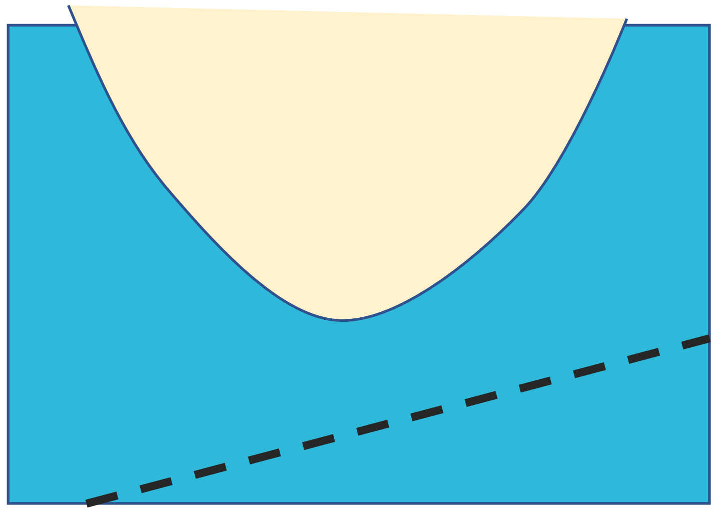
Example: Best photograph spot

The optimization problem may look like this.

$$\langle x_p^*, x_w^* \rangle = \underset{\substack{x_p \in \mathbb{R} \\ x_w \in \mathbb{R}}}{\operatorname{argmin}} (x_p - x_w)^2 + \left(\frac{1}{2}x_p^2 - \frac{1}{2}x_w + 3\right)^2$$

$$\frac{\partial f}{\partial x_p} = 2(x_p - x_w) + 2\left(\frac{1}{2}x_p^2 - \frac{1}{2}x_w + 3\right)(x_p)$$

$$\frac{\partial f}{\partial x_w} = -2(x_p - x_w) + 2\left(\frac{1}{2}x_p^2 - \frac{1}{2}x_w + 3\right)\left(-\frac{1}{2}\right)$$



$$y_p(x_p) = \frac{1}{2}x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2}x_w - 5$$

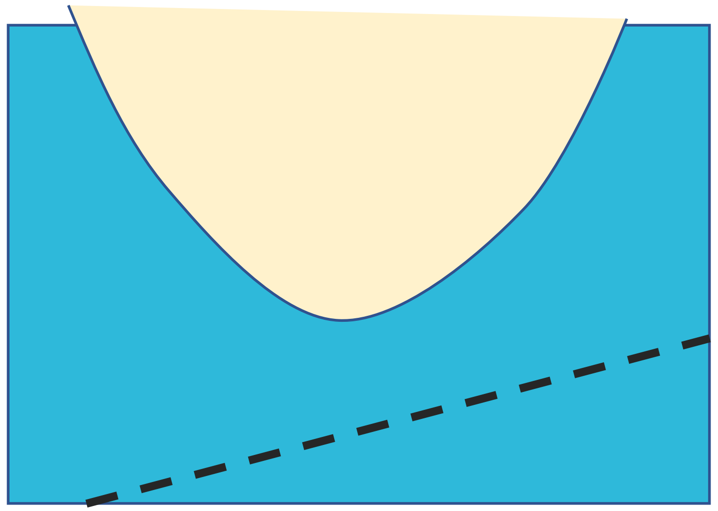
Example: Best photograph spot

$$2\left(x_p^* - x_w^*\right) + 2\left(\frac{1}{2}{x_p^*}^2 - \frac{1}{2}x_w^* + 3\right)\left(x_p^*\right) = 0$$

$$-2\left(x_p^* - x_w^*\right) + 2\left(\frac{1}{2}{x_p^*}^2 - \frac{1}{2}x_w^* + 3\right)\left(-\frac{1}{2}\right) = 0$$

$$\begin{gathered} {x_p^*}^3 - x_p^* x_w^* + 8x_p^* - 2x_w^* = 0 \\ -{x_p^*}^2 - 4x_p^* + 5x_w^* - 6 = 0 \end{gathered}$$

$$\begin{gathered} x_p^* = 0.5[\mathrm{m}], \qquad x_w^* = 1.65[\mathrm{m}] \\ y_p^* = -1.875[\mathrm{m}], \qquad y_w^* = -4.175[\mathrm{m}] \end{gathered}$$



$$y_p(x_p) = \frac{1}{2}x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2}x_w - 5$$

Example: Solving for zero gradient

```python
from scipy.optimize import fsolve
def gradient_cost(w):
    x_p = w[0]
    x_w = w[1]
    f = []
    f.append(x_p**3-x_p*x_w+8*x_p-2*x_w)
    f.append(-x_p**2-4*x_p+5*x_w-6)
    return f
w_0 = [0,0]
solution = fsolve(gradient_cost,w_0)
print(f"Optimizer:{solution}")
```

```
Optimizer:[0.5  1.65]
```

Example: Using minimize

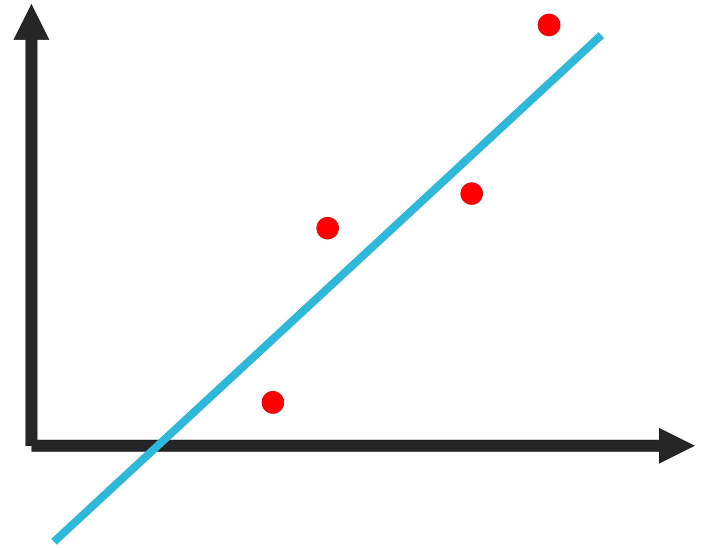
```
from scipy.optimize import minimize
def cost(w):
    y_p = lambda x_p : 0.5*(x_p**2)-2
    y_w = lambda x_w : 0.5*x_w-5
    x_p = w[0]
    x_w = w[1]
    return (x_p-x_w)**2+(y_p(x_p)-y_w(x_w))**2
w_0 = [0,0]
solution = minimize(cost,w_0)
print(f"Optimal cost:{solution.fun}")
print(f"Optimizer:{solution.x}")
```

```
Optimal cost:6.612500000012421
Optimizer:[0.49999998 1.64999683]
```

Example: Least Square

Given a set of data pair $\langle x_i, y_i \rangle$, find the "**best**" linear equation that represent the data.

$$y = mx + c$$



Example: Least Square

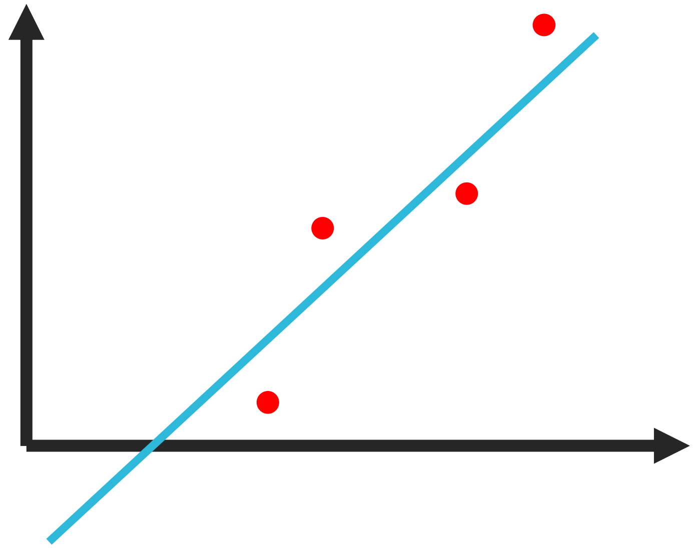
Given a set of data pair $\langle x_i, y_i \rangle$, find the **"best"** linear equation that represent the data.

$$y = mx + c$$

The best linear equation is the one the provide least amount of the square of the errors combined.

$$\text{cost} = \sum_{i=1}^{N} (mx_i + c - y_i)^2$$

What are the design variables ?



Example: Least Square

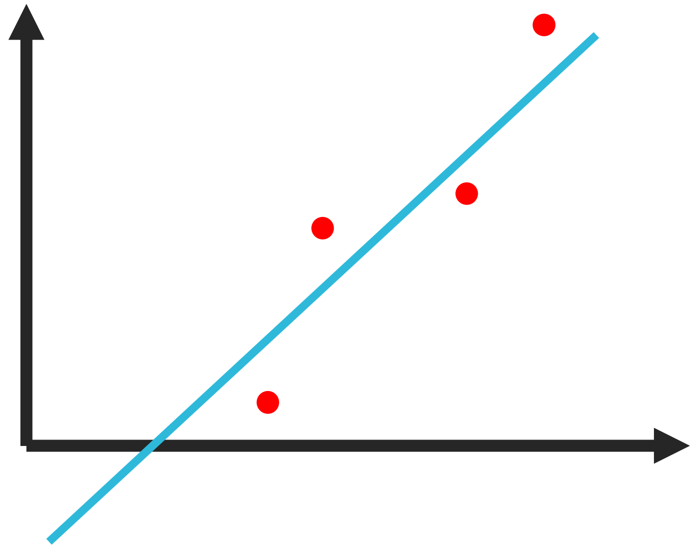
Given a set of data pair $\langle x_i, y_i \rangle$, find the **"best"** linear equation that represent the data.

$$y = mx + c$$

The best linear equation is the one the provide least amount of the square of the errors combined.

$$\text{cost} = \sum_{i=1}^{N} (mx_i + c - y_i)^2$$

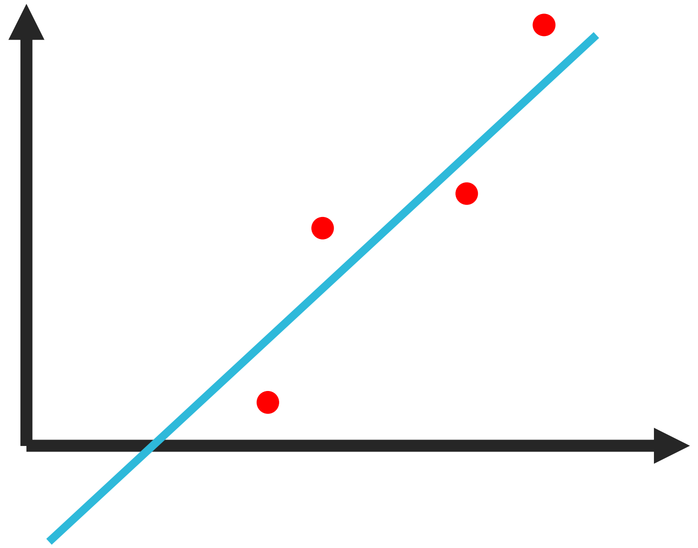
The design variables are m and c



Example: Least Square

The optimization problem can be written as follows.

$$\langle m^*, c^* \rangle = \underset{m,c}{\operatorname{argmin}} \sum_{i=1}^{N} (mx_i + c - y_i)^2$$



Example: Least Square

The problem has closed form solution.

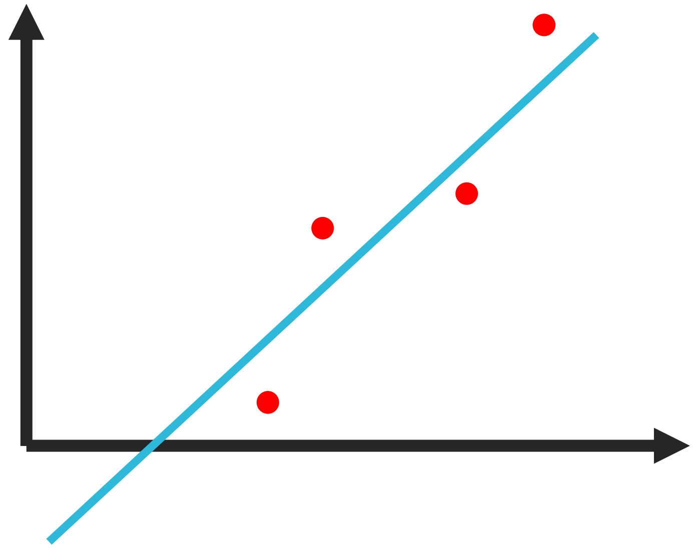
$$mx_i + c - y_i = \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} - y_i$$

Then

$$\begin{bmatrix} mx_1 + c - y_1 \\ mx_2 + c - y_2 \\ \vdots \\ mx_N + c - y_N \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \mathbf{Aw} - \mathbf{b}$$

The optimization can be expressed as

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{Aw} - \mathbf{b})^{\top} (\mathbf{Aw} - \mathbf{b})$$



Example: Least Square

Using vector and matrix differentiation, we can find the optimizer.

$$\nabla_{\mathbf{w}}\left((\mathbf{Aw}-\mathbf{b})^\top(\mathbf{Aw}-\mathbf{b})\right) = 2\mathrm{A}^\top(\mathbf{Aw}-\mathbf{b})$$

The optimizer may occur when the Jacobian is 0. Type equation here.

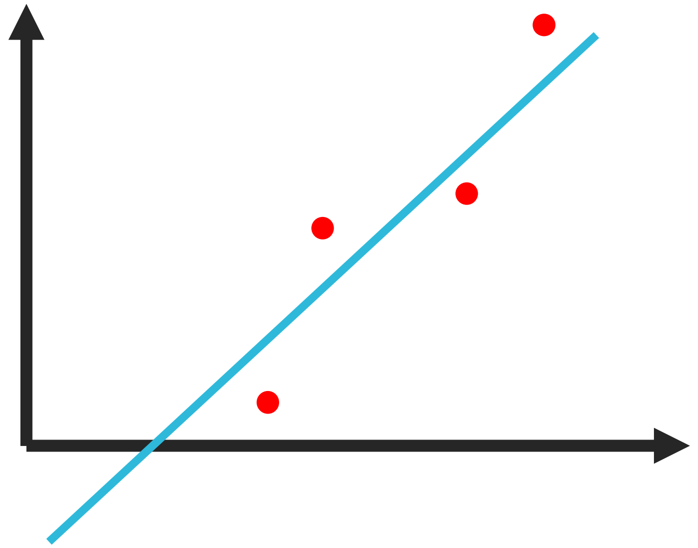
$$2\mathrm{A}^\top(\mathbf{Aw}^*-\mathbf{b}) = \mathbf{0}$$

$$\mathbf{A}^\top\mathbf{Aw}^* = \mathbf{A}^\top\mathbf{b}$$

$$\mathbf{w}^* = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{b}$$

$$\mathbf{w}^* = \mathbf{A}^\dagger\mathbf{b}$$

Left pseudoinverse

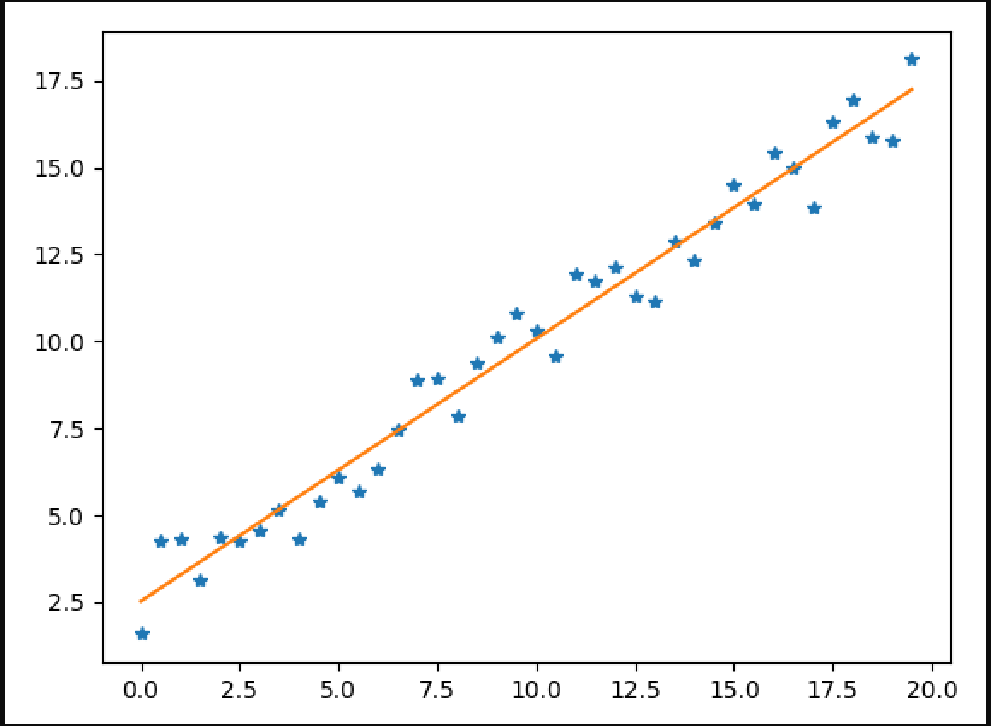


Example: Simulate data set

```
# simulate data set
m = 0.75
c = 2.5
x = np.arange(0,20,0.5)
y = m*x+c+3*(np.random.random(len(x))-0.5)
```

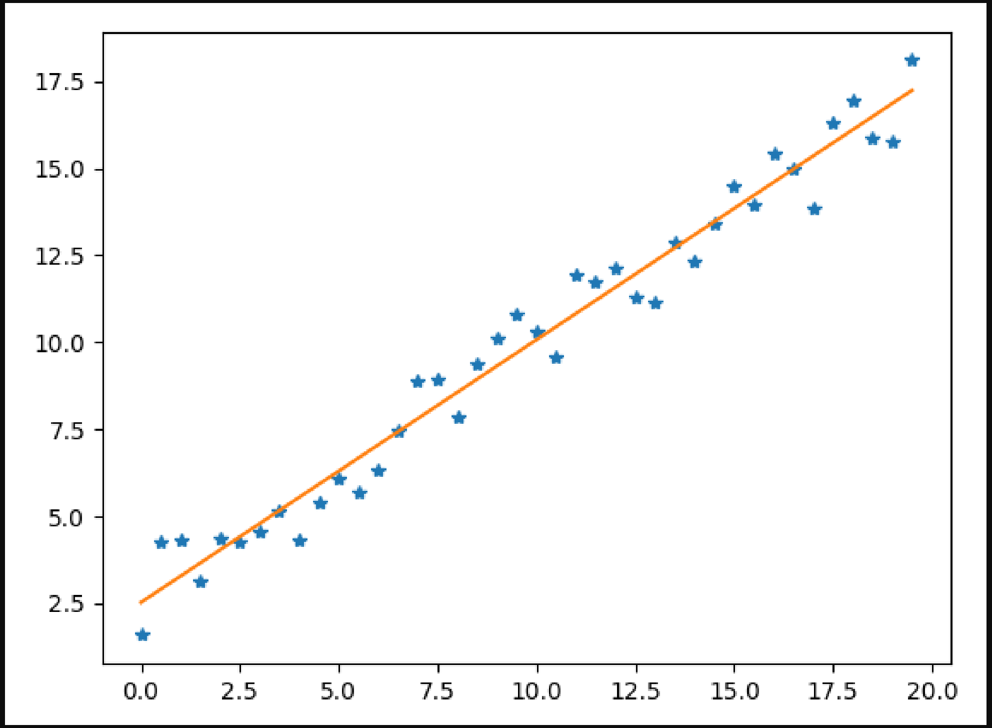
Example: Using minimize (overkill)

```
from scipy.optimize import minimize
import numpy as np
import matplotlib.pyplot as plt
# solve for best m and c
def cost(w,A,b):
    dy = A@np.array(w)-b
    return np.dot(dy,dy)
A = np.array([x,np.ones(len(x))]).T
b = np.array(y)
cost_Ab = lambda w : cost(w,A,b)
solution = minimize(cost_Ab,[0,0])
print(f"m is {solution.x[0]}, c is {solution.x[1]}")
plt.plot(x,y,'*')
plt.plot(x,solution.x[0]*x+solution.x[1])
plt.show()
```



Example: Using closed form solution

```
import numpy as np
import matplotlib.pyplot as plt
# solve for best m and c
A = np.array([x,np.ones(len(x))]).T
b = np.array(y)
#solution = np.linalg.lstsq(A,b)
w = np.linalg.pinv(A)@b
print(f"m is {w[0]}, c is {w[1]}")
plt.plot(x,y,'*')
plt.plot(x,w[0]*x+w[1])
plt.show()
```



Challenge 1: Parameter Estimation

Given an equation of forward kinematics of 2DOF planar manipulator

$$\mathbf{p}(\mathbf{q}) = \begin{bmatrix} l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) \\ l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \end{bmatrix}$$

And given a data set $\langle \mathbf{q}_i, \mathbf{p}_i \rangle$

Use Least-square Estimation to determine l_1 and l_2

(The dataset is not provided. You can simulate the dataset by assigning l_1 and l_2 and add some noises to the positions)

Constraints

- A set of limitations on the design variables
- Something that we cannot violate

Equality constraints: a set of constraints that must match the limitation exactly

$$\mathbf{g}(\mathbf{w}) = \mathbf{0}$$

or

$$\begin{bmatrix} g_1(w_1, w_2, \dots, w_n) \\ g_2(w_1, w_2, \dots, w_n) \\ \vdots \\ g_{n_g}(w_1, w_2, \dots, w_n) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Inequality constraints: a set of constraints that lies within the limit

$$\mathbf{h}(\mathbf{w}) \leq \mathbf{0}$$

or

$$\begin{bmatrix} h_1(w_1, w_2, \dots, w_n) \\ h_2(w_1, w_2, \dots, w_n) \\ \vdots \\ h_{n_h}(w_1, w_2, \dots, w_n) \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Constrained Optimization

Searching for the optimizer $\mathbf{w}^*$ that minimizes (or maximize) the objective function subject to given constraints.

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathcal{W}}{\operatorname{argmin}} f(\mathbf{w})$$

$$\text{s.t.}$$

$$\mathbf{g}(\mathbf{w}) = \mathbf{0}$$
$$\mathbf{h}(\mathbf{w}) \leq \mathbf{0}$$

Example: Best photograph spot

The optimization problem may look like this.

$$\langle x_p^*, x_w^* \rangle = \underset{\substack{x_p \in \mathbb{R} \\ x_w \in \mathbb{R}}}{\operatorname{argmin}} (x_p - x_w)^2 + \left(\frac{1}{2}x_p^2 - \frac{1}{2}x_w + 3\right)^2$$

$$\Downarrow$$

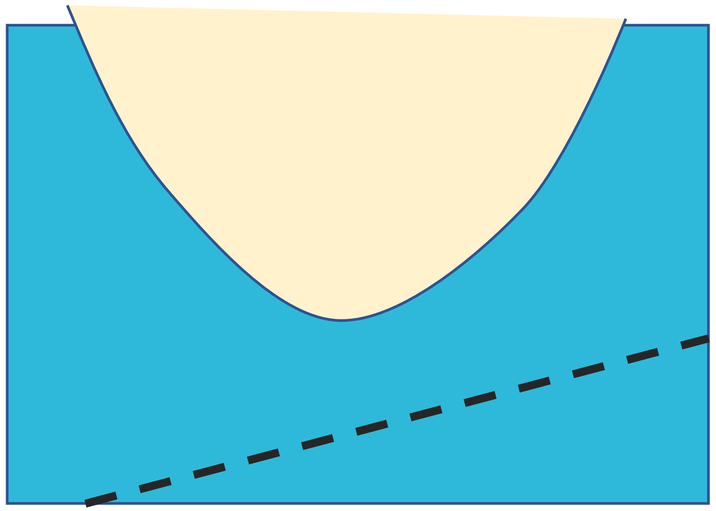
$$\langle x_p^*, x_w^* \rangle = \underset{x_p, x_w, y_p, y_w}{\operatorname{argmin}} \; (x_p - x_w)^2 + (y_p - y_w)^2$$

s.t.

$$\begin{bmatrix} \frac{1}{2}x_p^2 - 2 - y_p \\ \frac{1}{2}x_w - 5 \; - y_p \end{bmatrix} = 0$$

More design variables

Simpler cost



$$y_p(x_p) = \frac{1}{2}x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2}x_w - 5$$

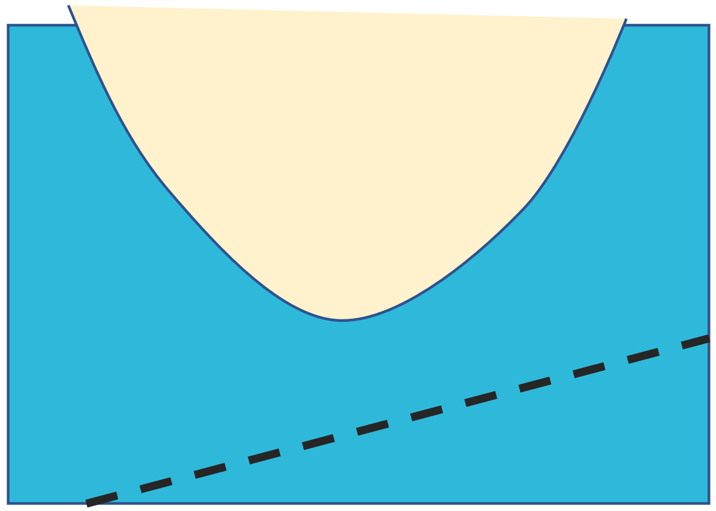
Example: Solving by hand using Lagrange Multiplier

We cannot simply solve for zero-gradient since it might not satisfy the constraints. We can introduce a Lagrange multipliers $\boldsymbol{\lambda}$ as another set of independent variables. The we can rewrite the objective function as a new function f'.

$$f'(\mathbf{w}, \boldsymbol{\lambda}) = f(\mathbf{w}) + \boldsymbol{\lambda}^\top \mathbf{g}(\mathbf{w})$$

If $\mathbf{w}$ satisfies the constraints $\mathbf{g}(\mathbf{w}) = 0$ optimizing f' is essentially the same as optimizing f. Now we can solve for zero-gradient of f' with respect to both $\mathbf{w}$ and $\boldsymbol{\lambda}$.

$$\nabla_{\mathbf{w}} f'(\mathbf{w}, \boldsymbol{\lambda}) = \nabla_{\mathbf{w}} f(\mathbf{w}) + \left(\frac{\partial \mathbf{g}}{\partial \boldsymbol{w}}\right)^\top \boldsymbol{\lambda}$$
$$\nabla_{\boldsymbol{\lambda}} f'(\mathbf{w}, \boldsymbol{\lambda}) = \mathbf{g}(\mathbf{w})$$



$$y_p(x_p) = \frac{1}{2} x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2} x_w - 5$$

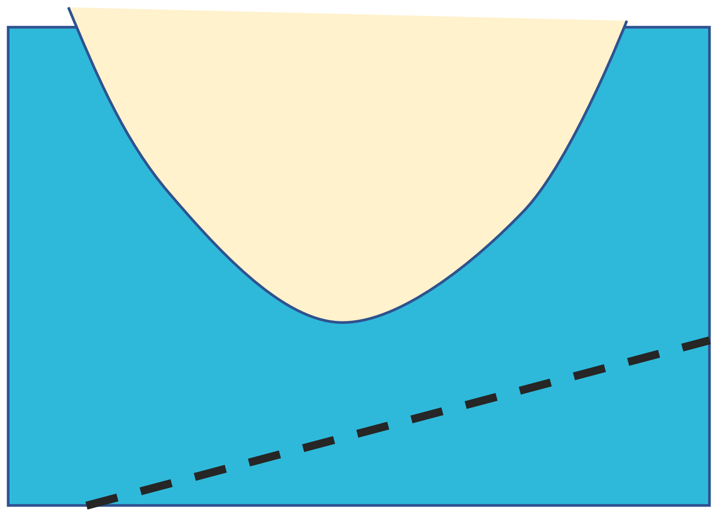
Example: Solving by hand using Lagrange Multiplier

$$\nabla_{\mathbf{w}} f(\mathbf{w}) + \left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)^{\top} \boldsymbol{\lambda} = \begin{bmatrix} 2(x_p - x_w) \\ -2(x_p - x_w) \\ 2(y_p - y_w) \\ -2(y_p - y_w) \end{bmatrix} + \begin{bmatrix} x_p & 0 \\ 0 & \frac{1}{2} \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$$

$$\nabla_{\boldsymbol{\lambda}} f'(\mathbf{w}, \boldsymbol{\lambda}) = \begin{bmatrix} \frac{1}{2} x_p^2 - 2 - y_p \\ \frac{1}{2} x_w - 5 \ - y_w \end{bmatrix}$$

Setting both functions to zero, we can determine the optimizer and its corresponding Lagrange multipliers.

$$\begin{bmatrix} 2(x_p^* - x_w^*) + x_p^* \lambda_1^* \\ -2(x_p^* - x_w^*) + \frac{1}{2} \lambda_2^* \\ 2(y_p^* - y_w^*) - \lambda_1^* \\ -2(y_p^* - y_w^*) - \lambda_2^* \\ \frac{1}{2} {x_p^*}^2 - 2 - y_p^* \\ \frac{1}{2} x_w^* - 5 \ - y_w^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



$$y_p(x_p) = \frac{1}{2} x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2} x_w - 5$$

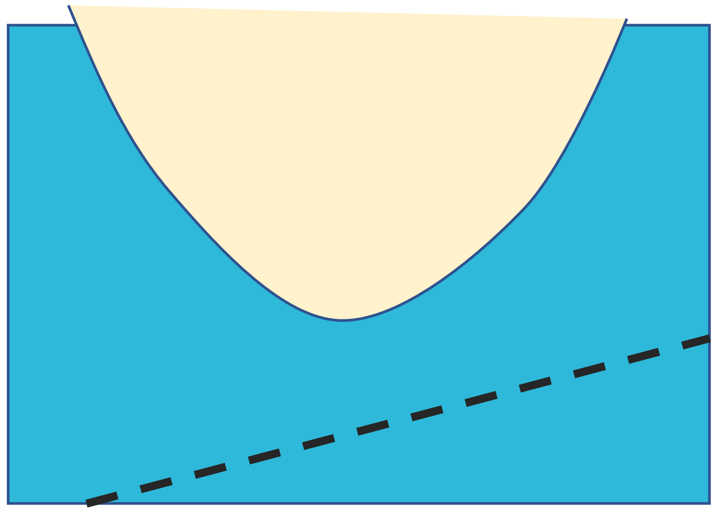
Example: Solving by hand using Lagrange Multiplier

$$\nabla_{\mathbf{w}} f(\mathbf{w}) + \left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right)^{\top} \boldsymbol{\lambda} = \begin{bmatrix} 2(x_p - x_w) \\ -2(x_p - x_w) \\ 2(y_p - y_w) \\ -2(y_p - y_w) \end{bmatrix} + \begin{bmatrix} x_p & 0 \\ 0 & \frac{1}{2} \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$$

$$\nabla_{\boldsymbol{\lambda}} f'(\mathbf{w}, \boldsymbol{\lambda}) = \begin{bmatrix} \frac{1}{2} x_p^2 - 2 - y_p \\ \frac{1}{2} x_w - 5 \; - y_w \end{bmatrix}$$

Setting both functions to zero, we can determine the optimizer and its corresponding Lagrange multipliers.

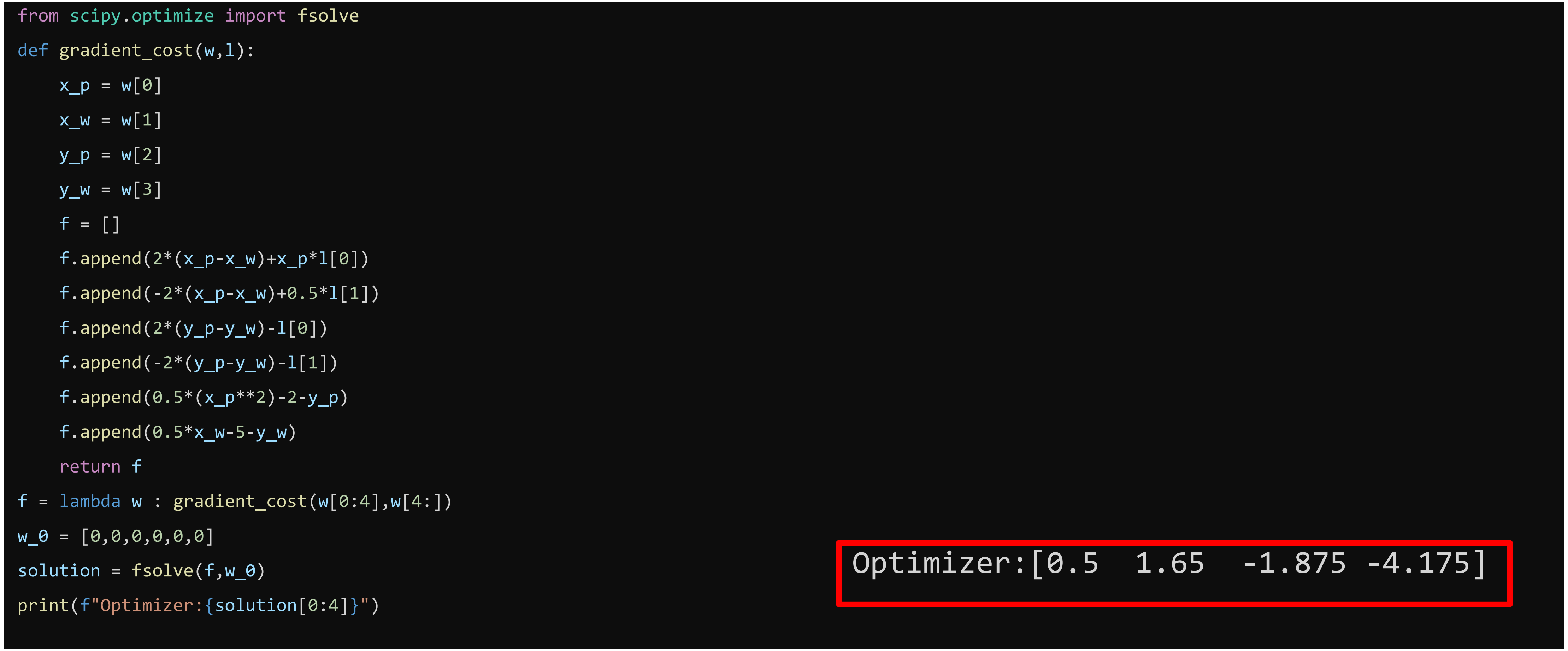
$$\begin{bmatrix} 2(x_p^* - x_w^*) + x_p^* \lambda_1^* \\ -2(x_p^* - x_w^*) + \frac{1}{2} \lambda_2^* \\ 2(y_p^* - y_w^*) - \lambda_1^* \\ -2(y_p^* - y_w^*) - \lambda_2^* \\ \frac{1}{2} {x_p^*}^2 - 2 - y_p^* \\ \frac{1}{2} x_w^* - 5 \; - y_w^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



$$y_p(x_p) = \frac{1}{2} x_p^2 - 2$$

$$y_w(x_w) = \frac{1}{2} x_w - 5$$

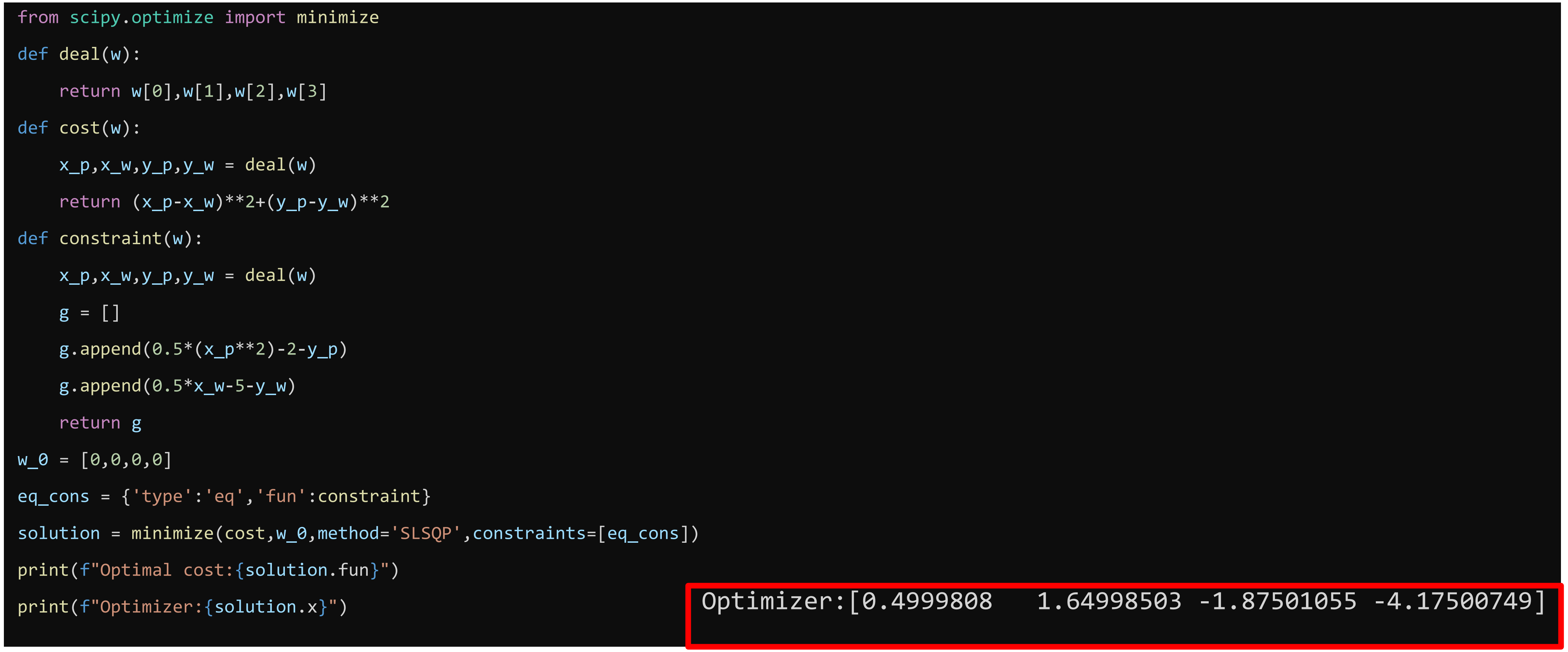
Example: Solving for zero gradient



```
from scipy.optimize import fsolve
def gradient_cost(w,l):
    x_p = w[0]
    x_w = w[1]
    y_p = w[2]
    y_w = w[3]
    f = []
    f.append(2*(x_p-x_w)+x_p*l[0])
    f.append(-2*(x_p-x_w)+0.5*l[1])
    f.append(2*(y_p-y_w)-l[0])
    f.append(-2*(y_p-y_w)-l[1])
    f.append(0.5*(x_p**2)-2-y_p)
    f.append(0.5*x_w-5-y_w)
    return f
f = lambda w : gradient_cost(w[0:4],w[4:])
w_0 = [0,0,0,0,0,0]
solution = fsolve(f,w_0)
print(f"Optimizer:{solution[0:4]}")
```

```
Optimizer:[0.5   1.65   -1.875 -4.175]
```

Example: using minimize



```python
from scipy.optimize import minimize
def deal(w):
    return w[0],w[1],w[2],w[3]
def cost(w):
    x_p,x_w,y_p,y_w = deal(w)
    return (x_p-x_w)**2+(y_p-y_w)**2
def constraint(w):
    x_p,x_w,y_p,y_w = deal(w)
    g = []
    g.append(0.5*(x_p**2)-2-y_p)
    g.append(0.5*x_w-5-y_w)
    return g
w_0 = [0,0,0,0]
eq_cons = {'type':'eq','fun':constraint}
solution = minimize(cost,w_0,method='SLSQP',constraints=[eq_cons])
print(f"Optimal cost:{solution.fun}")
print(f"Optimizer:{solution.x}")
```

```
Optimizer:[0.4999808   1.64998503 -1.87501055 -4.17500749]
```

Example: Inverse Kinematics

Given a 3DOF planar manipulator with the following Forward Kinematics equation.

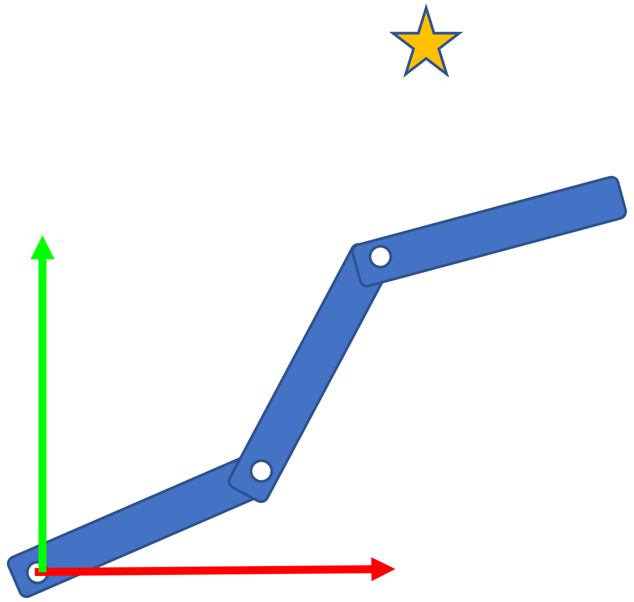
$$\mathbf{p} = \mathbf{fk}(\mathbf{q}) = \begin{bmatrix} l_1\cos(q_1) + l_2\cos(q_1+q_2) + l_3\cos(q_1+q_2+q_3) \\ l_1\sin(q_1) + l_2\sin(q_1+q_2) + l_3\sin(q_1+q_2+q_3) \end{bmatrix}$$

And joint limits

$$\mathbf{q}_{\min} \leq \mathbf{q} \leq \mathbf{q}_{\max}$$

And a target position $\mathbf{p}_{\mathrm{t}}$, find the best joint configuration $\mathbf{q}^*$.

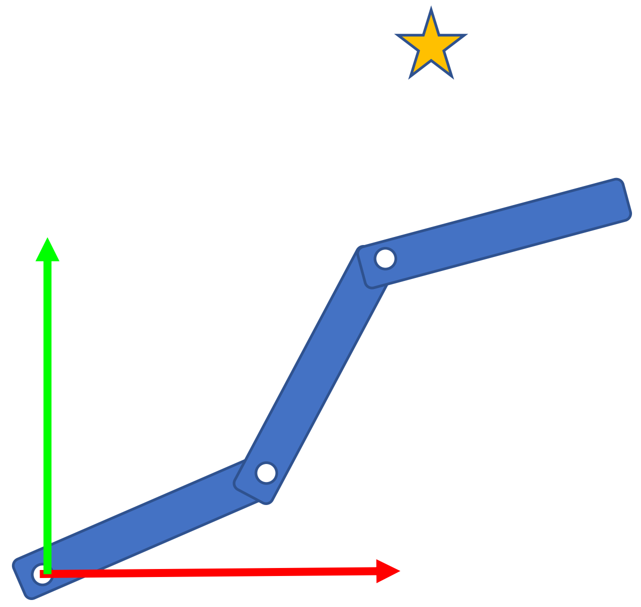
What is the "best" joint configuration ?



Example: Inverse Kinematics

Best joint configuration $\mathbf{q}^*$

- Least movement from the previous joint configuration $\mathbf{q}_0$
- Safest : furthest away from the joint limits



Example: Least movement from the previous joint configuration $\mathbf{q}_0$

Given an initial joint configuration $\mathbf{q}_0$,

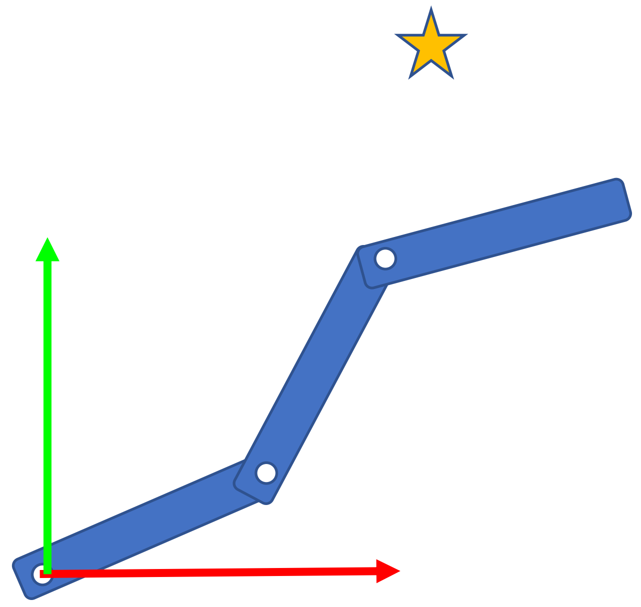
A vector of error $\tilde{\mathbf{q}}$ is simply the difference between joint configuration that we choose and the initial joint configuration.

$$\tilde{\mathbf{q}} = \mathbf{q} - \mathbf{q}_0$$

We choose the objective to be sum of the square of the errors.

$$f(\mathbf{q}) = (\mathbf{q} - \mathbf{q}_0)^\top(\mathbf{q} - \mathbf{q}_0) = \tilde{\mathbf{q}}^\top\tilde{\mathbf{q}}$$

This is what we have to minimize.



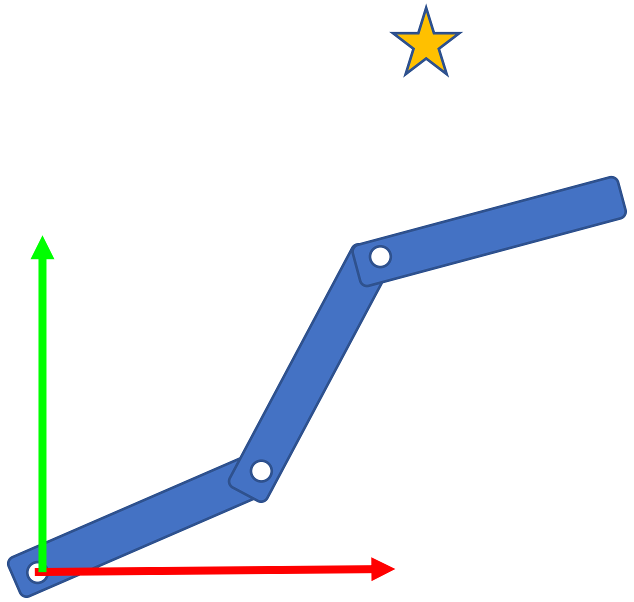
Example: Least movement from the previous joint configuration $\mathbf{q}_0$

However, $\mathbf{q}$ cannot be anything. The corresponding position must equal to the target position $\mathbf{p}_\mathrm{t}$. This can be written as equality constraints.

$$\mathbf{fk}(\mathbf{q}) - \mathbf{p}_\mathrm{t} = \mathbf{0}$$

The joint configuration must also lie in the joint limits.

$$\begin{bmatrix} \mathbf{q} - \mathbf{q}_\mathrm{max} \\ \mathbf{q}_\mathrm{min} - \mathbf{q} \end{bmatrix} \leq \mathbf{0}$$



Example: Least movement from the previous joint configuration $\mathbf{q}_0$

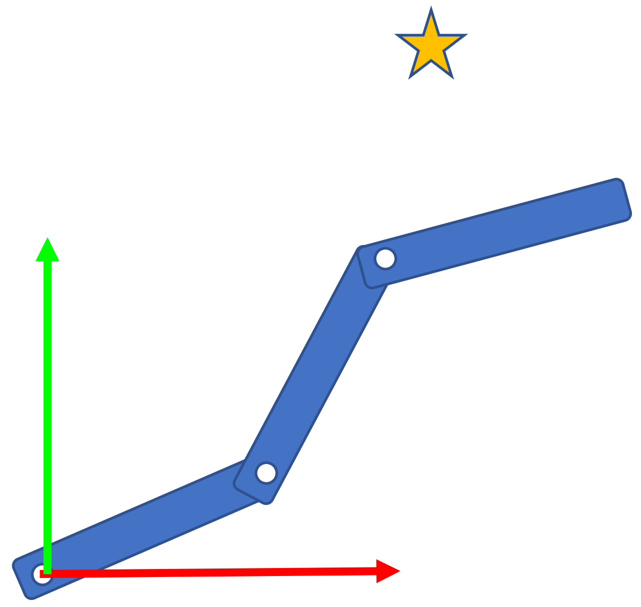
We finally formulate an optimization problem as the following.

$$\mathbf{q}^* = \underset{\mathbf{q}}{\operatorname{argmin}} (\mathbf{q} - \mathbf{q}_0)^\top (\mathbf{q} - \mathbf{q}_0)$$

s.t.

$$\mathbf{fk}(\mathbf{q}) - \mathbf{p}_\mathrm{t} = \mathbf{0}$$

$$\begin{bmatrix} \mathbf{q} - \mathbf{q}_\mathrm{max} \\ \mathbf{q}_\mathrm{min} - \mathbf{q} \end{bmatrix} \leq \mathbf{0}$$



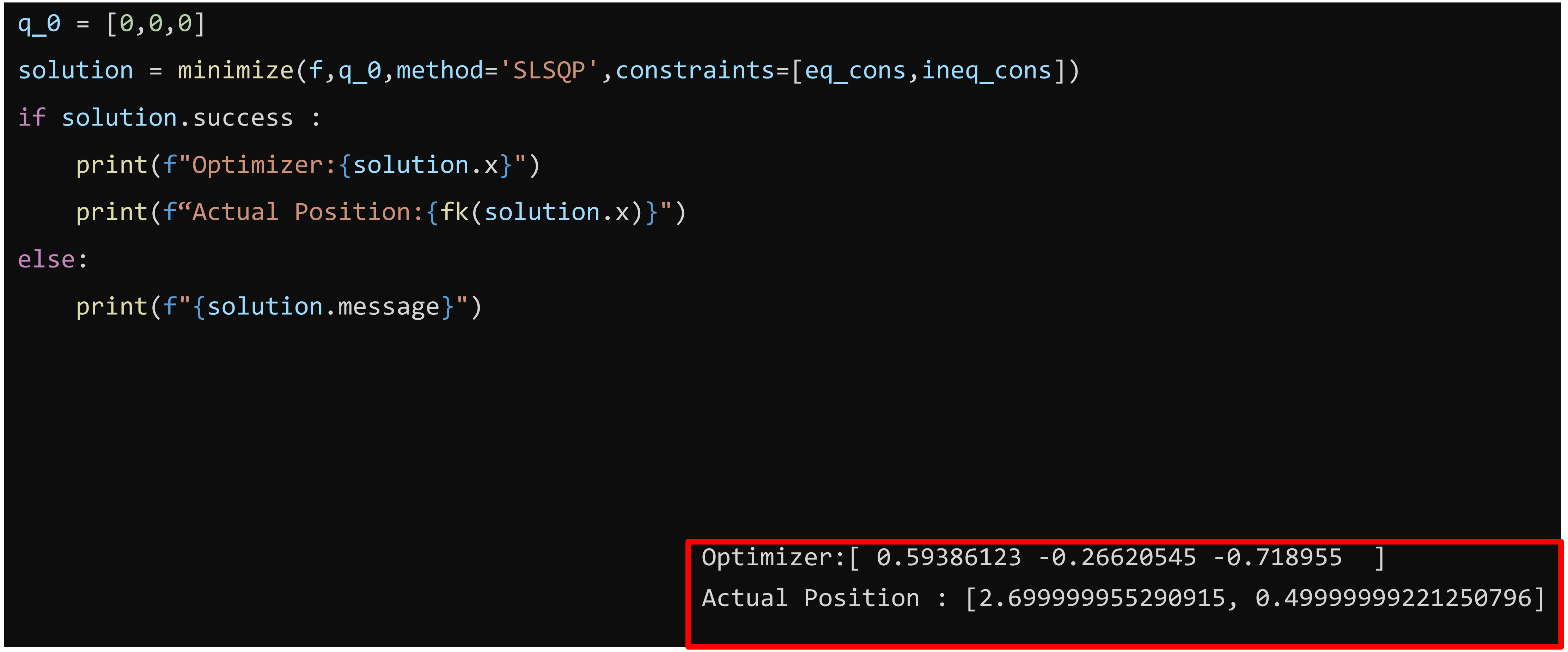
Example: setting up the given parameters

```python
import numpy as np
p_t = [2.7,0.5]
q_0 = [np.pi/4, np.pi/3 , np.pi/12]
q_max = [np.pi,np.pi/2,np.pi/2]
q_min = [-np.pi,-np.pi/2,-np.pi/2]
```

Example: defining objective & constraints

```python
def cost(q,q_0):
    dq = np.array(q)-np.array(q_0)
    return np.dot(dq,dq)
def fk(q):
    l = [1,1,1]
    p = []
    p.append(l[0]*np.cos(q[0])+l[1]*np.cos(q[0]+q[1])+l[2]*np.cos(q[0]+q[1]+q[2]))
    p.append(l[0]*np.sin(q[0])+l[1]*np.sin(q[0]+q[1])+l[2]*np.sin(q[0]+q[1]+q[2]))
    return p
f = lambda q : cost(q,q_0)
eqcon = lambda q : np.array(fk(q))-np.array(p_t)
incon = lambda q : np.hstack((np.array(q_max)-np.array(q),np.array(q)-np.array(q_min)))
ineq_cons = {'type':'ineq','fun':incon}
eq_cons = {'type':'eq','fun':eqcon}
```

Example: defining objective & constraints



```
q_0 = [0,0,0]
solution = minimize(f,q_0,method='SLSQP',constraints=[eq_cons,ineq_cons])
if solution.success :
    print(f"Optimizer:{solution.x}")
    print(f“Actual Position:{fk(solution.x)}")
else:
    print(f"{solution.message}")
```

```
Optimizer:[ 0.59386123 -0.26620545 -0.718955   ]
Actual Position : [2.699999955290915, 0.49999999221250796]
```

Example: Safest configuration

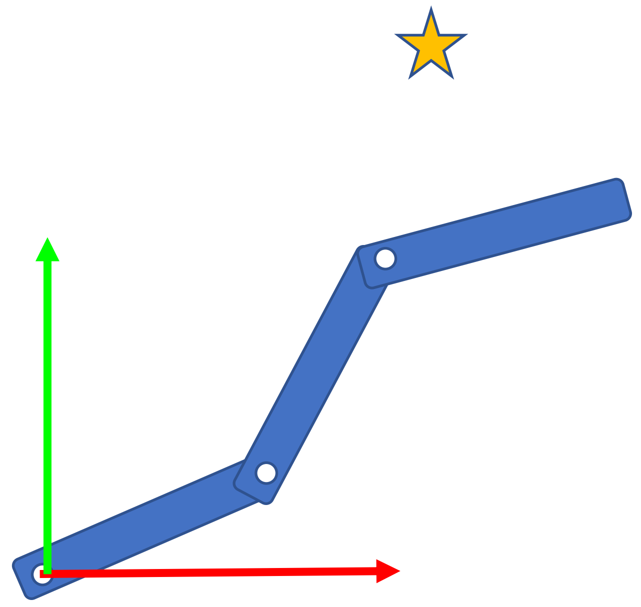
Sometimes, the best means the safest. Hitting the joint limits can cause serious damage to the robot's hardware. So, the safest configuration is the configuration that stay furthest away from the limits yet satisfies the forward kinematics.

We can reinterpret the "safest" configuration as staying closest to the mid-range configuration $\mathbf{q}_{\text{mid}}$, which is defined as follows.

$$\mathbf{q}_{\text{mid}} = \frac{1}{2}(\mathbf{q}_{\text{max}} - \mathbf{q}_{\text{min}})$$

A new vector of error $\tilde{\mathbf{q}}$ is simply the difference between joint configuration that we choose and the mid-range joint configuration.

$$\tilde{\mathbf{q}} = \mathbf{q} - \mathbf{q}_{\text{mid}}$$



Example: Safest configuration

We can choose the objective function to be the same as before.

$$\text{cost} = (\mathbf{q} - \mathbf{q}_{\text{mid}})^\top (\mathbf{q} - \mathbf{q}_{\text{mid}})$$

or

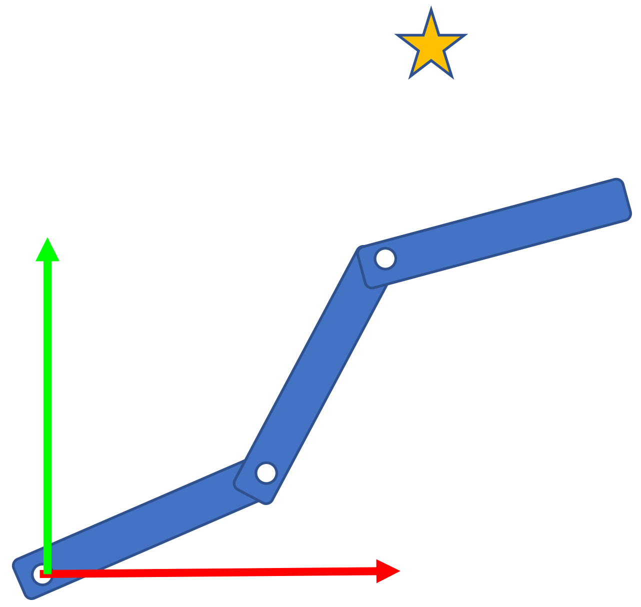
$$\text{cost} = \sum_{i=1}^{n} (q_i - q_{\text{mid},i})^2$$

However, some joints have wider ranges than the others. Therefore, we can weigh the cost by their ranges.

$$\text{cost} = \sum_{i=1}^{n} \left(\frac{1}{q_{\max,i} - q_{\min,i}} \right) (q_i - q_{\text{mid},i})^2$$

or

$$\text{cost} = (\mathbf{q} - \mathbf{q}_{\text{mid}})^\top \mathbf{diag}(\mathbf{q}_{\text{max}} - \mathbf{q}_{\text{min}})^{-1} (\mathbf{q} - \mathbf{q}_{\text{mid}})$$



Example: Safest configuration

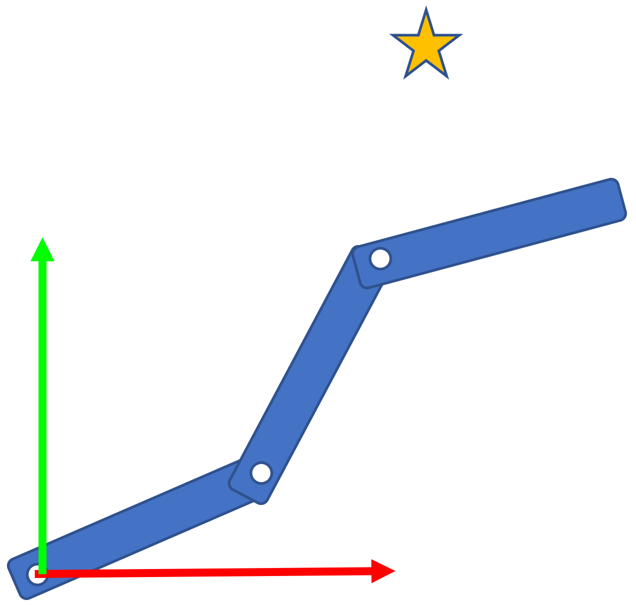
We finally formulate an optimization problem as the following.

$$\mathbf{q}^* = \underset{\mathbf{q}}{\operatorname{argmin}}(\mathbf{q} - \mathbf{q}_{\text{mid}})^\top \mathbf{diag}(\mathbf{q}_{\text{max}} - \mathbf{q}_{\text{min}})^{-1}(\mathbf{q} - \mathbf{q}_{\text{mid}})$$

s.t.

$$\mathbf{fk}(\mathbf{q}) - \mathbf{p}_{\text{t}} = \mathbf{0}$$

$$\begin{bmatrix} \mathbf{q} - \mathbf{q}_{\text{max}} \\ \mathbf{q}_{\text{min}} - \mathbf{q} \end{bmatrix} \leq \mathbf{0}$$



Challenge 2: Safest Configuration

Solve the following optimization problem using scipy's minimize.

$$\mathbf{q}^* = \underset{\mathbf{q}}{\operatorname{argmin}} (\mathbf{q} - \mathbf{q}_{\mathrm{mid}})^\top \mathbf{diag}(\mathbf{q}_{\mathrm{max}} - \mathbf{q}_{\mathrm{min}})^{-1} (\mathbf{q} - \mathbf{q}_{\mathrm{mid}})$$

s.t.

$$\mathbf{fk}(\mathbf{q}) - \mathbf{p}_{\mathrm{t}} = \mathbf{0}$$

$$\begin{bmatrix} \mathbf{q} - \mathbf{q}_{\mathrm{max}} \\ \mathbf{q}_{\mathrm{min}} - \mathbf{q} \end{bmatrix} \leq \mathbf{0}$$