Aalto University
School of Electrical
Engineering

# CVX Tutorial (Python), Lecture 6

## Convex Optimization D
## ELEC - E5424

October 31, 2023

# Introduction

- CVX is a modeling system for disciplined convex programming
- Disciplined convex programs (DCP), are convex optimization problems that are described using a limited set of construction rules
- CVX is available in Matlab and Python
- Do not use CVX to check if a problem is convex:
  - formulate a convex problem first
  - then, confirm with CVX that it is indeed convex
- CVX is not meant for large, high dimensional, problems.

# Installation

## Python

CVXPY can be installed using `pip install cvxpy`. Further instructions available here.

## MATLAB

Instructions for installation are available here. Though similar, syntax between Python and MATLAB are very different.

# Disciplined Convex Programming

- Disciplined convex programming is a methodology for constructing convex optimization problems proposed by Michael Grant, Stephen Boyd, and Yinyu Ye.
- Disciplined convex programming imposes a set of conventions or rules, which we call the DCP ruleset. Problems which adhere to the ruleset can be rapidly and automatically verified as convex and converted to a solvable form.
- Problems that violate the DCP ruleset are rejected, even when the problem is convex.

# Example 1: Least-Squares

Minimize $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$, where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{x} \in \mathbb{R}^n$, and $\boldsymbol{b} \in \mathbb{R}^m$

```
import cvxpy as cp
import numpy as np

m = 16
n = 10
A_mat = np.random.randn(m,n)
b = np.random.randn(m,1)

x = cp.Variable((n,1))
objective = cp.Minimize(cp.norm2(A_mat@x - b))
problem = cp.Problem(objective)
result = problem.solve()
```

# Example 1: Least-Squares

- If problem is feasible and bounded, the solved variables are stored in `x.value`
- The `result` contains the optimal value for the given problem. In practical problems this often correspond to a cost or error.
- Setting `problem.solve(verbose=True)` will give detailed information about the solver used, iterations taken etc.

# Example 2: Bound-Constrained LS

$$\begin{aligned} \text{Minimize} \quad & \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \\ \text{Subject to} \quad & l \preceq \boldsymbol{x} \preceq u \end{aligned}$$

```
x = cp.Variable((n,1))
objective = cp.Minimize(cp.norm2(A_mat@x - b))
constraints = [
    l <= x,
    x <= u,
]
problem = cp.Problem(objective, constraints)
result = problem.solve()
```

# Constraints

- CVX only supports equality constraints (==) and non-strict inequalities (<=, >=).
  - If absolutely necessary, strict inequalities can often be addressed with normalization.
- Be careful to not use the assignment operator for equality constraints (=)
- To be able to verify convexity of a problem, DCP sets further limitations to constraint definitions:
  - left- and right- hand sides of an equality constraint must be affine,
  - inequality constraints of form $f(x) <= g(x)$, or $g(x) >= f(x)$ are only accepted if $f$ is convex and $g$ concave.

# Inequality Constraints

- Inequality constraints (<= and >=) can be used for scalars and matrices.
- If one side is scalar and the other is a matrix, constraints are interpreted elementwise, i.e, each element of the matrix is constrained by the (same) scalar.
- Inequalities can not be used if either side is complex.

# Attributes

- CVX does not use a set membership operator such as $x \in \mathbb{R}$.
- Instead, variables (and parameters) can be given additional properties using **attributes**, for example:
  - $x \in \mathbb{R}^+$ can be expressed as `x = cp.Variable(1, pos=True)`
  - $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ Hermitian, can be expressed as `A = cp.Variable((n,n), hermitian=True)`

# DCP Ruleset

- The DCP ruleset breaks problems down to atomic functions.
- Each atomic function has a **sign** and **curvature**.
- Using composition rules from convex analysis, DCP tries to propagate the curvature and sign from atomic functions to the complete expression.

| **Curvature** | **Meaning** |
|---|---|
| constant | $f(x)$ independent of $x$ |
| affine | $f(\theta x + (1-\theta)y) = \theta f(x) + (1-\theta)f(y)$, $\forall x,\ y,\ \theta \in [0,1]$ |
| convex | $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$, $\forall x,\ y,\ \theta \in [0,1]$ |
| concave | $f(\theta x + (1-\theta)y) \geq \theta f(x) + (1-\theta)f(y)$, $\forall x,\ y,\ \theta \in [0,1]$ |
| unknown | DCP analysis can not determine curvature |

# Curvature rules

$f(expr_1, expr_2, \ldots, expr_3)$ is convex if $f$ is a convex function and **for each** $expr_i$ one of the following conditions hold:

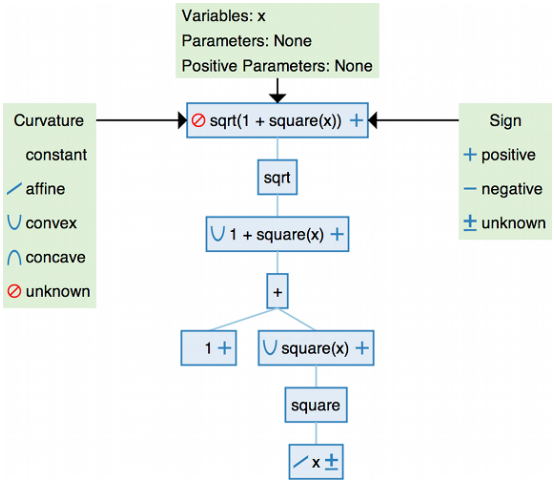
- $f$ is increasing in argument $i$ and $expr_i$ is convex.
- $f$ is decreasing in argument $i$ and $expr_i$ is concave.
- $expr_i$ is affine or constant.

Similar logic is applied to establishing concavity or affinity of a function. If a function does not fulfill conditions for convex, concave, or affine, it is unknown.

## Note

If any (sub)expression has unknown curvature, CVX can not solve the problem.

# Curvature analysis example




[Source]

# Problem Reformulation

DCP yields unknown curvature for $\sqrt{1 + x^2}$. However, we can represent the same problem with other atomic functions:

$$\sqrt{1 + x^2} = \sqrt{1^2 + x^2} = \|[1, x]\|_2$$

In CVXPY, `cp.sqrt(1 + cp.square(x))` becomes `cp.norm(cp.vstack(1, x), 2)`. We now have a known convex function (norm) with an affine (sub)expression. Based on the DCP ruleset this is convex.

# Advanced: Semidefinite Programming

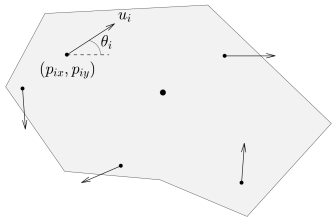
- CVXPY supports Semidefinite Programming (SDP) using additional LMI notation or attributes. Note that this feature differs from the SDP mode in MATLAB.
- In order to define Positive Semidefinite (PSD) matrix, we can either:
  - Define a variable with the PSD attribute `X = cp.Variable((n,n), PSD=True)`.
  - Introduce a PSD constraint with the special « or » operators.

```
X = cp.Variable((n,n), PSD=True)
constraints = [X » 0]
```

# Demo: Thruster Problem

# Problem Definition

Consider a rocket/satellite with $n$ thrusters at $p_i = (p_{ix}, p_{iy})$. Each thruster acts in the direction of $\theta_i$. The force of each thruster can be adjusted between 0 and 1.




## Problem 1:

Find Thruster forces $u_i$ that yield given desired (total) forces and torque on the rigid body, while minimizing fuel usage (if feasible).

# Alternate Formulations

**Problem 2:**
Find Thruster forces $u_i$ that yield given desired forces and torque, while minimizing fuel usage (if feasible). Each thruster consists of a pair pointing towards both $\theta_i$ and $-\theta_i$.

**Problem 3:**
Find Thruster forces $u_i$ that yield given desired forces and torque, while minimizing maximal force/torque error.

# Extra: Shadow Prices in LP

# Factory

A factory can manufacture two products which yield a profit of 20€ for product 1 and 50€ for product 2. Each product requires molding, assembly, painting, and inspection. These steps have the following (monthly) constraints:

$$
\begin{aligned}
x_1 + 3x_2 &\leq 300 \quad \text{[molding]},\\
x_1 + x_2 &\leq 300 \quad \text{[assembly]},\\
5x_1 + 2x_2 &\leq 500 \quad \text{[painting]},\\
2x_1 + 2x_2 &\leq 300 \quad \text{[inspection]},
\end{aligned}
$$

where $x_n$ is the quantity of product $n$ being produced. Maximize total (monthly) profit and compute the corresponding dual values. What do they describe?

# Optimization problem

$$
\begin{aligned}
\text{Maximize} \quad & 20x_1 + 50x_2 \\
\text{Subject to} \quad & x_1 + 3x_2 \leq 300, \\
& x_1 + x_2 \leq 300, \\
& 5x_1 + 2x_2 \leq 500, \\
& 2x_1 + 2x_2 \leq 300, \\
& x_1, x_2 \geq 0
\end{aligned}
$$