A Quick Guide on CVXPY

CS4017: Introduction to Optimization

Ziyi Yu

What is a solver?

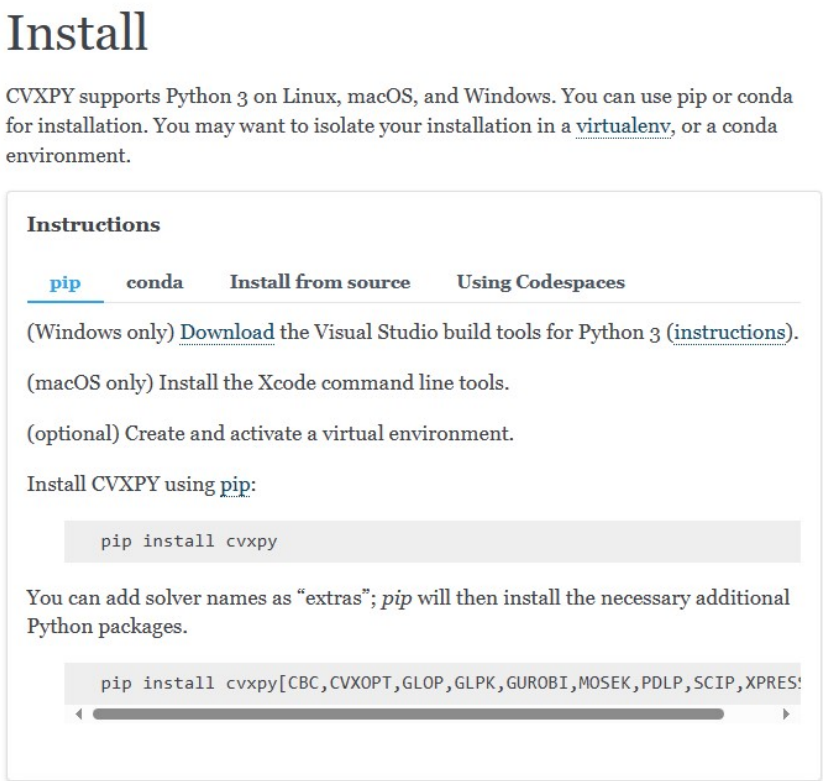
Given an optimization problem, how to compute the optimal value / optimal variable? For example,

$$\begin{aligned} \underset{x,\,y}{\text{minimize}} \quad & (x-y)^2 \\ \text{subject to} \quad & x+y=1, \\ & x-y \geq 1. \end{aligned}$$

A solver helps you with that. All you have to do is write your problem into a **proper format**, and then input to the solver.

CVXPY

CVX was first developed by Michael Grant (student of Stephen Boyd) in 2006. Nowadays there are many solvers, with interfaces to various programming languages.
Here we mainly focus on CVXPY: https://www.cvxpy.org/, a Python package.



Install

CVXPY supports Python 3 on Linux, macOS, and Windows. You can use pip or conda for installation. You may want to isolate your installation in a virtualenv, or a conda environment.

Instructions

pip | conda | Install from source | Using Codespaces

(Windows only) Download the Visual Studio build tools for Python 3 (instructions).

(macOS only) Install the Xcode command line tools.

(optional) Create and activate a virtual environment.

Install CVXPY using pip:

```
pip install cvxpy
```

You can add solver names as "extras"; *pip* will then install the necessary additional Python packages.

```
pip install cvxpy[CBC,CVXOPT,GLOP,GLPK,GUROBI,MOSEK,PDLP,SCIP,XPRES
```

The first example

$$\begin{array}{ll} \underset{x,\,y}{\text{minimize}} & (x-y)^2 \\ \text{subject to} & x+y=1, \\ & x-y \geq 1. \end{array}$$

```python
import cvxpy as cp

# Create two scalar optimization variables.
x = cp.Variable()
y = cp.Variable()

# Create two constraints.
constraints = [x + y == 1,
               x - y >= 1]

# Form objective.
obj = cp.Minimize((x - y)**2)

# Form and solve problem.
prob = cp.Problem(obj, constraints)
prob.solve()  # Returns the optimal value.
print("status:", prob.status)
print("optimal value", prob.value)
print("optimal var", x.value, y.value)
```

```
status: optimal
optimal value 0.999999999761
optimal var 1.00000000001 -1.19961841702e-11
```

Constraints, Variables, and Parameters

`https://www.cvxpy.org/tutorial/intro/index.html`

1. Constraints: elementwise $==, <=, >=$; semidefinite $>>, <<$

Constraints, Variables, and Parameters

https://www.cvxpy.org/tutorial/intro/index.html

1. Constraints: elementwise $==, <=, >=$; semidefinite $>>, <<$
2. Variables: the decision variables, can be scalar, vector, or matrices (but not higher dimensions)

```python
# A scalar variable.
a = cp.Variable()

# Vector variable with shape (5,).
x = cp.Variable(5)

# Matrix variable with shape (5, 1).
x = cp.Variable((5, 1))

# Matrix variable with shape (4, 7).
A = cp.Variable((4, 7))
```

Constraints, Variables, and Parameters

https://www.cvxpy.org/tutorial/intro/index.html

1. Constraints: elementwise $==, <=, >=$; semidefinite $>>, <<$
2. Variables: the decision variables, can be scalar, vector, or matrices (but not higher dimensions)
3. Parameters: constants for a given problem instance: no need to re-write the code when the values change

```python
# Positive scalar parameter.
m = cp.Parameter(nonneg=True)

# Column vector parameter with unknown sign (by default).
c = cp.Parameter(5)

# Matrix parameter with negative entries.
G = cp.Parameter((4, 7), nonpos=True)

# Assigns a constant value to G.
G.value = -numpy.ones((4, 7))
```

You can initialize a parameter with a value. The following code segments are equivalent:

```python
# Create parameter, then assign value.
rho = cp.Parameter(nonneg=True)
rho.value = 2

# Initialize parameter with a value.
rho = cp.Parameter(nonneg=True, value=2)
```

Operators and functions

`https://www.cvxpy.org/tutorial/functions/index.html`

1. Operators: the same as Numpy: $+, -, *, /, @, .T, **$
2. Functions: most usual ones are already in-built

log_det(X)	$\log(\det(X))$	$X \in \mathbf{S}_+^n$	$\pm$ unknown	$\cap$ concave	None
log_sum_exp(X)	$\log\left(\sum_{ij} e^{X_{ij}}\right)$	$X \in \mathbf{R}^{m \times n}$	$\pm$ unknown	$\cup$ convex	$\nearrow$ incr.
matrix_frac(x, P)	$x^T P^{-1} x$	$x \in \mathbf{R}^n$ $P \in \mathbf{S}_{++}^n$	$+$ positive	$\cup$ convex	None
max(X)	$\max_{ij}\{X_{ij}\}$	$X \in \mathbf{R}^{m \times n}$	same as X	$\cup$ convex	$\nearrow$ incr.
mean(X)	$\frac{1}{mn}\sum_{ij}\{X_{ij}\}$	$X \in \mathbf{R}^{m \times n}$	same as X	$\nearrow$ affine	$\nearrow$ incr.
min(X)	$\min_{ij}\{X_{ij}\}$	$X \in \mathbf{R}^{m \times n}$	same as X	$\cap$ concave	$\nearrow$ incr.
mixed_norm(X, p, q)	$\left(\sum_k \left(\sum_l \lvert x_{k,l}\rvert^p\right)^{q/p}\right)^{1/q}$	$X \in \mathbf{R}^{n \times n}$	$+$ positive	$\cup$ convex	None
norm(x) norm(x, 2)	$\sqrt{\sum_i \lvert x_i\rvert^2}$	$X \in \mathbf{R}^n$	$+$ positive	$\cup$ convex	$\nearrow$ for $x_i \geq 0$ $\searrow$ for $x_i \leq 0$
norm(x, 1)	$\sum_i \lvert x_i\rvert$	$x \in \mathbf{R}^n$	$+$ positive	$\cup$ convex	$\nearrow$ for $x_i \geq 0$ $\searrow$ for $x_i \leq 0$
norm(x, "inf")	$\max_i\{\lvert x_i\rvert\}$	$x \in \mathbf{R}^n$	$+$ positive	$\cup$ convex	$\nearrow$ for $x_i \geq 0$ $\searrow$ for $x_i \leq 0$
norm(X, "fro")	$\sqrt{\sum_{ij} X_{ij}^2}$	$X \in \mathbf{R}^{m \times n}$	$+$ positive	$\cup$ convex	$\nearrow$ for $X_{ij} \geq 0$ $\searrow$ for $X_{ij} \leq 0$

Disciplined Convex Programming

Disciplined Convex Programming (DCP) helps CVXPY to check if the optimization problems are convex.
`https://www.cvxpy.org/tutorial/dcp/index.html`

1. Expressions: consist of variables, parameters (constants), operators, and functions.

```python
import cvxpy as cp

# Create variables and parameters.
x, y = cp.Variable(), cp.Variable()
a, b = cp.Parameter(), cp.Parameter()

# Examples of CVXPY expressions.
3.69 + b/3
x - 4*a
sqrt(x) - minimum(y, x - a)
maximum(2.66 - sqrt(y), square(x + 2*y))
```

Disciplined Convex Programming

Disciplined Convex Programming (DCP) helps CVXPY to check if the optimization problems are convex.

`https://www.cvxpy.org/tutorial/dcp/index.html`

1. Expressions: consist of variables, parameters (constants), operators, and functions.
2. Sign: flag each expression as *non-negative*, *non-positive*, *zero*, or *unknown*

```
x = cp.Variable()
a = cp.Parameter(nonpos=True)
c = numpy.array([1, -1])

print("sign of x:", x.sign)
print("sign of a:", a.sign)
print("sign of square(x):", cp.square(x).sign)
print("sign of c*a:", (c*a).sign)
```

```
sign of x: UNKNOWN
sign of a: NONPOSITIVE
sign of square(x): NONNEGATIVE
sign of c*a: UNKNOWN
```

Disciplined Convex Programming

Disciplined Convex Programming (DCP) helps CVXPY to check if the optimization problems are convex.

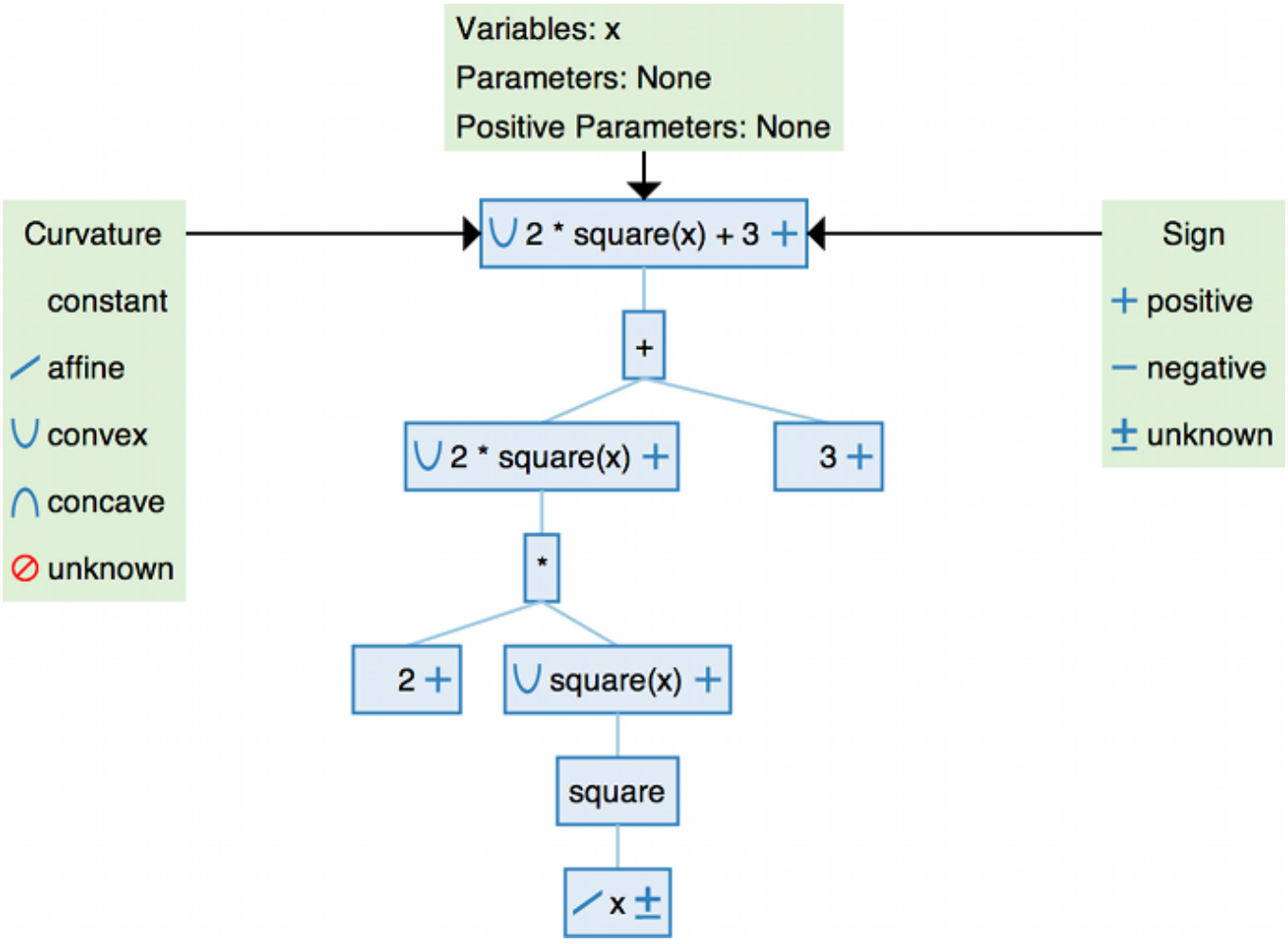
`https://www.cvxpy.org/tutorial/dcp/index.html`

1. Expressions: consist of variables, parameters (constants), operators, and functions.
2. Sign: flag each expression as *non-negative*, *non-positive*, *zero*, or *unknown*
3. Curvature: apply composition theorem to decide if the expression is *convex*, *concave*, *affine*, or *unknown*. This is based on the sign, convexity, and monotonicity of each sub-expression.

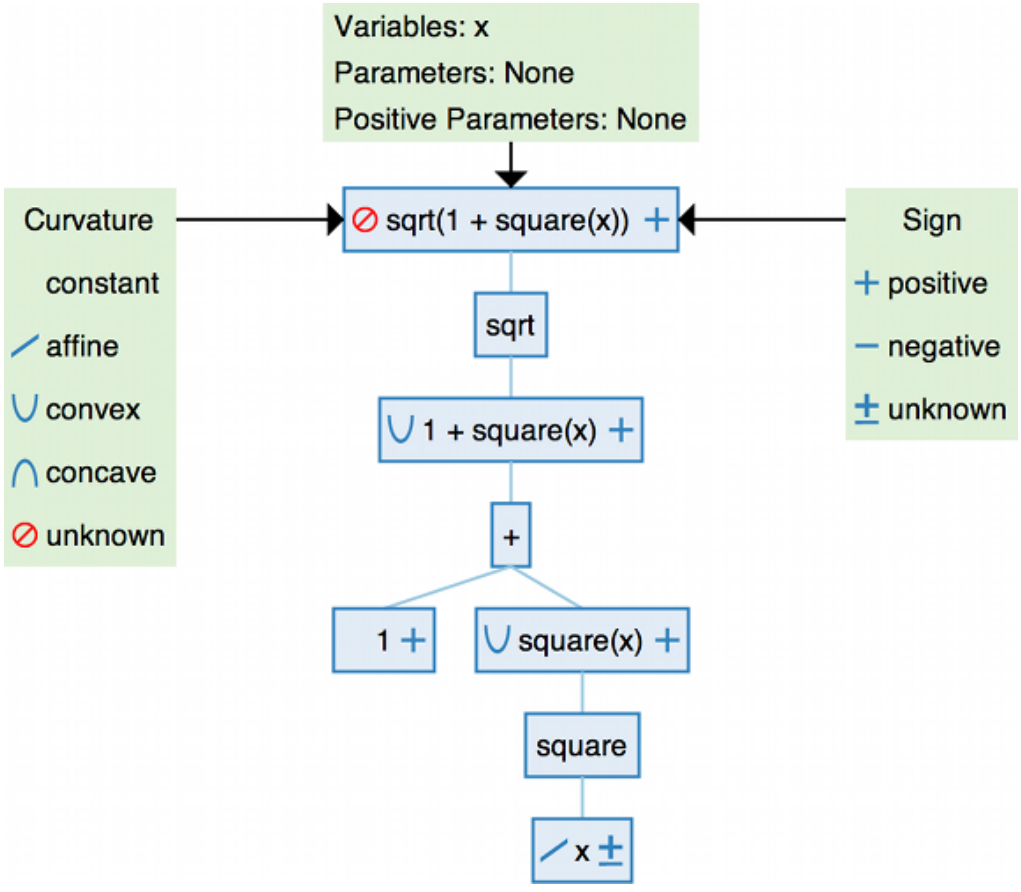
```
x = cp.Variable()
a = cp.Parameter(nonneg=True)

print("curvature of x:", x.curvature)
print("curvature of a:", a.curvature)
print("curvature of square(x):", cp.square(x).curvature)
print("curvature of sqrt(x):", cp.sqrt(x).curvature)
```

```
curvature of x: AFFINE
curvature of a: CONSTANT
curvature of square(x): CONVEX
curvature of sqrt(x): CONCAVE
```

DCP Examples



DCP Examples



Some expressions may not work. For example, sqrt() is *concave* and *increasing*, so it accepts only concave argument. By changing to another expression, it will be identified as convex.

```
print("sqrt(1 + square(x)) curvature:",
      cp.sqrt(1 + cp.square(x)).curvature)
print("norm(hstack([1, x]), 2) curvature:",
      cp.norm(cp.hstack([1, x]), 2).curvature)
```

```
sqrt(1 + square(x)) curvature: UNKNOWN
norm(hstack(1, x), 2) curvature: CONVEX
```

- Previously: $\sqrt{1+x^2}$
- Now:

$$\| [1, x] \|_2 = \left\| \begin{bmatrix} 1 \\ x \end{bmatrix} \right\|_2$$

DCP Examples

A problem is constructed from an objective and a list of constraints. If a problem follows the DCP rules, it is guaranteed to be convex and solvable by CVXPY. The DCP rules require that the problem objective have one of two forms:

- Minimize(convex)
- Maximize(concave)

The only valid constraints under the DCP rules are

- affine == affine
- convex <= concave
- concave >= convex

Select a proper solver

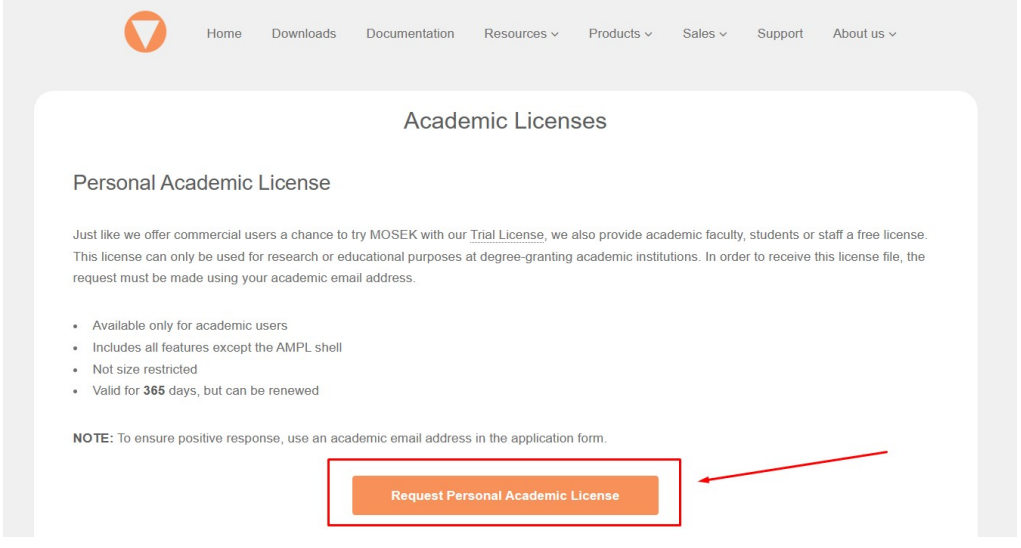
	LP	QP	SOCP	SDP	EXP	POW	MIP
CBC	X						X
CLARABEL	X	X	X	X	X	X	
COPT	X	X	X	X			X*
GLOP	X						
GLPK	X						
GLPK_MI	X						X
OSQP	X	X					
PIQP	X	X					
PROXQP	X	X					
PDLP	X						
CPLEX	X	X	X				X
NAG	X	X	X				
ECOS	X	X	X		X		
GUROBI	X	X	X				X
MOSEK	X	X	X	X	X	X	X**
CVXOPT	X	X	X	X			
SDPA ***	X	X	X	X			
SCS	X	X	X	X	X	X	
SCIP	X	X	X				X
XPRESS	X	X	X				X
SCIPY	X						X*

```
# Solve with MOSEK.
prob.solve(solver=cp.MOSEK)
print("optimal value with MOSEK:", prob.value)
```

See more in Link

Install MOSEK

1. Run ‘pip install cvxpy[MOSEK]‘
2. Goto `https://www.mosek.com/products/academic-licenses/`
   - 2.1 Request Personal Academic License, fill your information and receive an e-mail
   - 2.2 Accept the agreement and receive a ‘mosek.lic’, put it into proper place (follow the instruction in the e-mail)
3. Use MOSEK in your code via ‘prob.solve(solver=cp.MOSEK)‘



Examples: Quadratic Program

The standard form of quadratic program is:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & (1/2)x^\top Px + q^\top x + r \\ \text{subject to} & Gx \preceq h, \\ & Ax = b, \end{array}$$

with $P \in \mathbb{S}_+^n$.

```python
# Import packages.
import cvxpy as cp
import numpy as np

# Generate a random non-trivial quadratic program.
m = 15
n = 10
p = 5
np.random.seed(1)
P = np.random.randn(n, n)
P = P.T @ P
q = np.random.randn(n)
G = np.random.randn(m, n)
h = G @ np.random.randn(n)
A = np.random.randn(p, n)
b = np.random.randn(p)

# Define and solve the CVXPY problem.
x = cp.Variable(n)
prob = cp.Problem(cp.Minimize((1/2)*cp.quad_form(x, P) + q.T @ x),
                  [G @ x <= h,
                   A @ x == b])
prob.solve()

# Print result.
print("\nThe optimal value is", prob.value)
print("A solution x is")
print(x.value)
print("A dual solution corresponding to the inequality constraints is")
print(prob.constraints[0].dual_value)
```

```
The optimal value is 86.89141585569918
A solution x is
[-1.68244521  0.29769913 -2.38772183 -2.79986015  1.18270433 -0.20911897
 -4.50993526  3.76683701 -0.45770675 -3.78589638]
A dual solution corresponding to the inequality constraints is
[ 0.          0.          0.          0.          0.         10.45538054
  0.          0.          0.         39.67365045  0.          0.
  0.         20.79927156  6.54115873]
```

Examples: Semidefinite Program

The standard form of semidefinite program is:

$$\begin{array}{ll} \underset{X \in \mathbb{S}^n}{\text{minimize}} & \mathbf{tr}(CX) \\ \text{subject to} & \mathbf{tr}(A_i X) = b_i,\ i = 1, \ldots, p, \\ & X \succeq 0, \end{array}$$

with $C, A_1, \ldots, A_p \in \mathbb{S}^n$.

```python
# Import packages.
import cvxpy as cp
import numpy as np

# Generate a random SDP.
n = 3
p = 3
np.random.seed(1)
C = np.random.randn(n, n)
A = []
b = []
for i in range(p):
    A.append(np.random.randn(n, n))
    b.append(np.random.randn())

# Define and solve the CVXPY problem.
# Create a symmetric matrix variable.
X = cp.Variable((n,n), symmetric=True)
# The operator >> denotes matrix inequality.
constraints = [X >> 0]
constraints += [
    cp.trace(A[i] @ X) == b[i] for i in range(p)
]
prob = cp.Problem(cp.Minimize(cp.trace(C @ X)),
                  constraints)
prob.solve()

# Print result.
print("The optimal value is", prob.value)
print("A solution X is")
print(X.value)
```

```
The optimal value is 2.654348003008652
A solution X is
[[ 1.6080571  -0.59770202 -0.69575904]
 [-0.59770202  0.22228637  0.24689205]
 [-0.69575904  0.24689205  1.39679396]]
```

Examples: Least Squares

The goal of least squares is to

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2^2.$$

```python
# Import packages.
import cvxpy as cp
import numpy as np

# Generate data.
m = 20
n = 15
np.random.seed(1)
A = np.random.randn(m, n)
b = np.random.randn(m)

# Define and solve the CVXPY problem.
x = cp.Variable(n)
cost = cp.sum_squares(A @ x - b)
prob = cp.Problem(cp.Minimize(cost))
prob.solve()

# Print result.
print("\nThe optimal value is", prob.value)
print("The optimal x is")
print(x.value)
print("The norm of the residual is ", cp.norm(A @ x - b, p=2).value)
```

```
The optimal value is 7.005909828287484
The optimal x is
[ 0.17492418 -0.38102551  0.34732251  0.0173098  -0.0845784  -0.08134019
  0.293119    0.27019762  0.17493179 -0.23953449  0.64097935 -0.41633637
  0.12799688  0.1063942  -0.32158411]
The norm of the residual is  2.6468679280023557
```

Find out more

`https://www.cvxpy.org/examples/index.html`

Basic examples

- Least squares [.ipynb]
- Linear program [.ipynb]
- Quadratic program [.ipynb]
- Second-order cone program [.ipynb]
- Semidefinite program [.ipynb]
- Mixed-integer quadratic program [.ipynb]
- Control
- Portfolio optimization
- Worst-case risk analysis
- Model fitting
- Optimal advertising
- Total variation in-painting [.ipynb]

Disciplined geometric programming

- DGP fundamentals [.ipynb]
- Maximizing the volume of a box [.ipynb]
- Power control [.ipynb]
- Perron-Frobenius matrix completion [.ipynb]
- Rank-one nonnegative matrix factorization [.ipynb]

Disciplined quasiconvex programming

- Concave fractional function [.ipynb]
- Minimum-length least squares [.ipynb]
- Hypersonic shape design [.ipynb]

Derivatives

- Fundamentals [.ipynb]
- Queuing design [.ipynb]
- Structured prediction [.ipynb]

Machine learning

- Ridge regression [.ipynb]
- Lasso regression [.ipynb]
- Logistic regression [.ipynb]
- SVM classifier [.ipynb]
- Huber regression
- Quantile regression

Finance

- Portfolio optimization
- Cryptocurrency trading
- Entropic Portfolio Optimization
- Portfolio Optimization using SOC constraints
- Gini Mean Difference Portfolio Optimization
- Kurtosis Portfolio Optimization
- Relativistic Value at Risk Portfolio Optimization
- Approximate Kurtosis Portfolio Optimization

Advanced

- Object-oriented convex optimization [.ipynb]
- Consensus optimization [.ipynb]
- Method of multipliers [.ipynb]

Thank you

Any questions?