# Convex Optimization in Python with CVXPY

**Steven Diamond**     Stephen Boyd
Stanford University

SciPy 2018

# Outline

# Convex optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b, \end{array}$$

with variable $x \in \mathbf{R}^n$

- objective and inequality constraints $f_0, \ldots, f_m$ are convex

  for all $x$, $y$, $\theta \in [0, 1]$,

  $$f_i(\theta x + (1 - \theta)y) \leq \theta f_i(x) + (1 - \theta) f_i(y)$$

  *i.e.*, graphs of $f_i$ curve upward
- equality constraints are linear

## Why convex optimization?

- beautiful, fairly complete, and useful theory
- solution algorithms that work well in theory and practice
- **many applications** in
  - machine learning, statistics
  - control
  - signal, image processing
  - networking
  - engineering design
  - finance

. . . and many more

# How do you solve a convex problem?

- use someone else's ('standard') solver (LP, QP, SOCP, ...)
  - easy, but your problem **must** be in a standard form
  - cost of solver development amortized across many users
- write your own (custom) solver
  - lots of work, but can take advantage of special structure
- use a convex modeling language
  - transforms user-friendly format into solver-friendly standard form
  - extends reach of problems solvable by standard solvers

# Convex modeling languages

- long tradition of modeling languages for optimization
  - AMPL, GAMS
- modeling languages for convex optimization
  - CVX, YALMIP, CVXGEN, CVXPY, Convex.jl, RCVX
- function of a convex modeling language:
  - check/verify problem convexity
  - convert to standard form

# Disciplined convex programming (DCP)

- system for constructing expressions with known curvature
  - constant, affine, convex, concave
- expressions formed from
  - variables
  - constants and parameters
  - library of functions with known curvature, monotonicity, sign
- basis of all convex modeling systems
- more at dcp.stanford.edu

# Outline

# CVXPY

a modeling language in Python for convex optimization

- developed by Diamond & Boyd, 2014–
- uses signed DCP to verify convexity
- open source all the way to the solvers
- supports parameters
- mixes easily with general Python code, other libraries
- used in many research projects, classes, companies
- thousands of users

## Solvers

- ECOS (Domahidi)
  - cone solver
  - interior-point method
  - compact, library-free C code
- SCS (O'Donoghue)
  - cone solver
  - first-order method
  - parallelism with OpenMP
  - GPU support
- OSQP (Stellato, Banjac, Goulart)
  - first-order method
  - targets QPs and LPs
  - code generation support
- others: CVXOPT, GLPK, MOSEK, GUROBI, Cbc, ...

# CVXPY example

(constrained LASSO)

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2^2 + \gamma\|x\|_1 \\ \text{subject to} & \mathbf{1}^T x = 0, \quad \|x\|_\infty \leq 1 \end{array}$$

with variable $x \in \mathbf{R}^n$

```
from cvxpy import *
x = Variable(n)
cost = sum_squares(A*x-b) + gamma*norm(x,1)
obj = Minimize(cost)
constr = [sum_entries(x) == 0, norm(x,"inf") <= 1]
prob = Problem(obj, constr)
opt_val = prob.solve()
solution = x.value
```

# Outline

## Parameters in CVXPY

- symbolic representations of constants
- can specify sign (for use in DCP analysis)
- change value of constant without re-parsing problem

- for-loop style trade-off curve:

```
x_values = []
for val in numpy.logspace(-4, 2, 100):
    gamma.value = val
    prob.solve()
    x_values.append(x.value)
```

## Parallel style trade-off curve

```python
# Use tools for parallelism in standard library.
from multiprocessing import Pool

# Function maps gamma value to optimal x.
def get_x(gamma_value):
    gamma.value = gamma_value
    result = prob.solve()
    return x.value

# Parallel computation with N processes.
pool = Pool(processes = N)
x_values = pool.map(get_x, numpy.logspace(-4, 2, 100))
```

## Performance

(LASSO)

$$\text{minimize} \quad \|Ax - b\|_2^2 + \gamma\|x\|_1$$

with variable $x \in \mathbf{R}^n$

---

- $A \in \mathbf{R}^{1000 \times 500}$, 100 values $\gamma$
- single thread time for one LASSO: 1.6 seconds (OSQP)

| | for-loop | 4 proc. | 32 proc. | warm-start |
|---|---|---|---|---|
| 4 core MacBook Pro | 180 sec | 70 sec | 81 sec | 31 sec |
| 32 cores, Intel Xeon | 285 sec | 89 sec | 31 sec | 48 sec |

# Outline

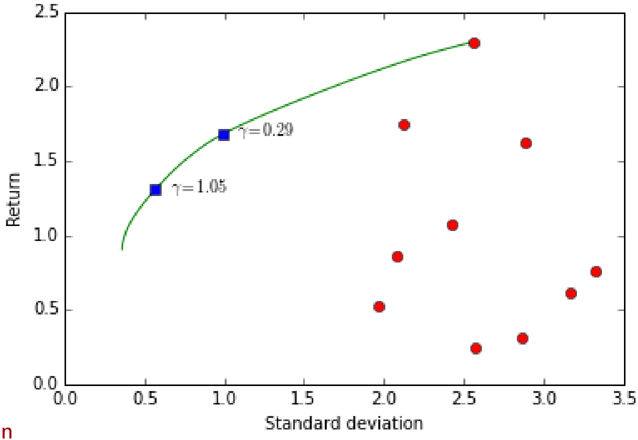
# Portfolio optimization

$$
\begin{array}{ll}
\text{maximize} & \mu^T w - \gamma w^T \Sigma w \\
\text{subject to} & \mathbf{1}^T w = 1, \quad w \in \mathcal{W}
\end{array}
$$

- variable $w \in \mathbf{R}^n$ is *portfolio allocation vector*
- $\mathcal{W}$ is set of allowed portfolios
- common case: $\mathcal{W} = \mathbf{R}^n_+$ (long only portfolio)
- $\gamma > 0$ is the *risk aversion parameter*
- $\mu^T w - \gamma w^T \Sigma w$ is *risk-adjusted return*
- varying $\gamma$ gives optimal *risk-return trade-off*

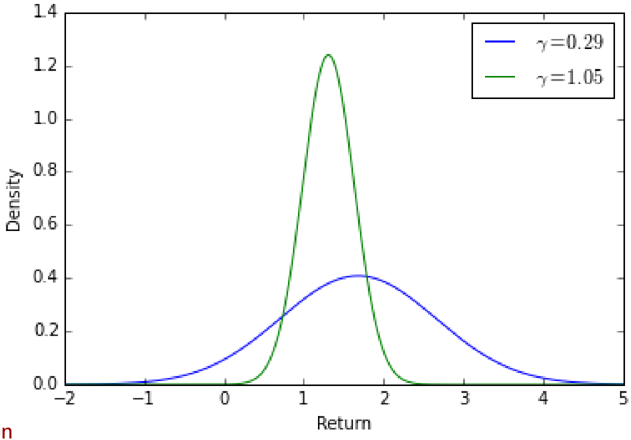
## Example

optimal risk-return trade-off for 10 assets, long only portfolio



## Example

return distributions for two risk aversion values



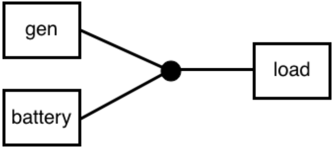
# Outline

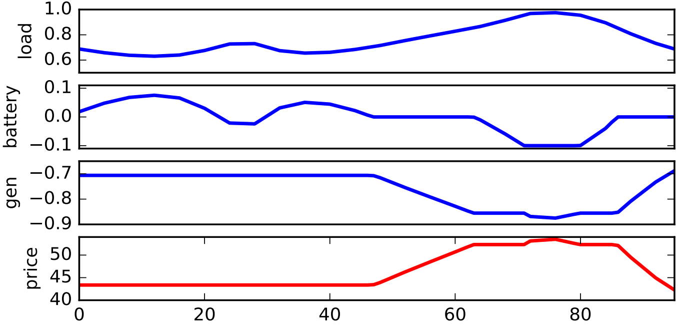
# Dynamic energy management

- devices interchange power at nets over multiple periods
  - generators
  - loads (fixed, deferrable, curtailable, ...)
  - storage systems (battery, pumped hydro, ...)
  - thermal/HVAC
  - transmission lines
- each device has objective function and constraints
- power conserved at nets

- minimize total system cost to get optimal power schedules
- net conservation dual variables are locational marginal prices (LMPs)

# Dynamic energy management example



## Dynamic energy management package

- CVXPY extension for dynamic energy management (Wytock, Diamond, Boyd, 2016)

```
from dem import *
load = FixedLoad(power=p_load)  # pre-specified load
gen = Generator(power_max=2, alpha=30, beta=1)
battery = Storage(discharge_max=0.1, charge_max=0.1,
                  energy_max=1.6)
net = Net([load.terminals[0], gen.terminals[0],
           battery.terminals[0]])
network = Group([load, gen, battery], [net])
network.optimize()
plot(net.price)  # plot LMP at net
```

# Outline

## Summary

- convex optimization in Python is easy with CVXPY
  - code follows the math
  - simple rules for verifying convexity
- CVXPY mixes well with high level Python
  - parallelism
  - object oriented design
- CVXPY is building block for
  - nonconvex optimization (DCCP, NCVX)
  - domain-specific application packages (CVXPortfolio)
- Installation instructions at cvxpy.org
- Projects at github.com/cvxgrp/