Tools and Methods for Large-Scale Convex Optimization

Eric Chu, Brendan O'Donoghue, Alex Domahidi, Neal Parikh, and Stephen Boyd
Information Systems Laboratory, Department of Electrical Engineering, Stanford University

Convex optimization

- many applications
  - machine learning, statistics
  - control, signal, image processing
  - networking
  - finance, supply chain
  - smart grid
  - engineering design
  - and many others ...
- but, barriers to widespread use
  - modeling
  - custom implementation
  - problem scale

Motivation and goal

motivation:

- want to solve **arbitrary-scale** optimization problems, *e.g.*,
  - machine learning/statistics with huge datasets
  - dynamic optimization on large-scale networks

goal:

- ideally, a system that
  - has CVX-like interface
  - targets modern large-scale and heterogeneous computing platforms
  - scales arbitrarily
  - is rapidly deployable

Primal-dual cone problems

- (primal) cone program and its dual

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax + s = b \\ & s \in \mathcal{K} \end{array} \qquad \begin{array}{ll} \text{maximize} & -b^T y \\ \text{subject to} & A^T y + c = 0 \\ & y \in \mathcal{K}^* \end{array}$$

- primal variables (x, s), dual variable y
- $\mathcal{K}$ is convex cone, $\mathcal{K}^*$ its dual cone
- **duality gap** for any feasible (x, s, y): $c^T x + b^T y \geq 0$
  - $-b^T y$ is lower bound on optimal value of primal problem
  - $c^T x$ is upper bound on optimal value of dual problem
- feasible (x, s, y) with zero duality gap is **solution**

Primal-dual solution

- find $z = (x, s, y)$ satisfying $z \in \mathcal{E} \cap \mathcal{C}$

$$\mathcal{E} = \{z \mid Gz = h\}, \qquad \mathcal{C} = \mathbf{R}^n \times \mathcal{K} \times \mathcal{K}^*$$

$$G = \begin{bmatrix} A & I & 0 \\ 0 & 0 & -A^T \\ c^T & 0 & b^T \end{bmatrix}, \qquad h = \begin{bmatrix} b \\ c \\ 0 \end{bmatrix}$$

Primal-dual operator splitting

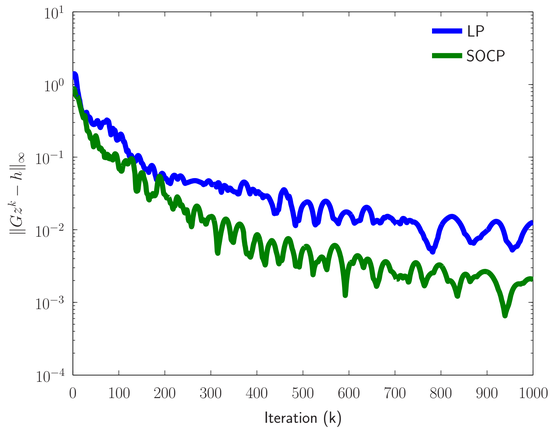
- operator splitting method for finding $z \in \mathcal{E} \cap \mathcal{C}$
- iterate

$$\begin{aligned} z^{k+1/2} &= \Pi_{\mathcal{E}}(z^k - u^k) \\ z^{k+1} &= \Pi_{\mathcal{C}}(z^{k+1/2} + u^k) \\ u^{k+1} &= u^k + z^{k+1/2} - z^{k+1} \end{aligned}$$

- $\Pi_{\mathcal{E}}$ ($\Pi_{\mathcal{C}}$) is projection onto $\mathcal{E}$ ($\mathcal{C}$)
- $z^{k+1/2} \in \mathcal{E}$, *i.e.*, satisfies equality constraints
- $z^k \in \mathcal{C}$, *i.e.*, satisfies cone constraints
- when $\mathcal{E} \cap \mathcal{C} \neq \emptyset$, *i.e.*, problem is solvable, $\|z^k - z^{k+1/2}\| \to 0$

Convergence of PDOS cone solver

LP and SOCP with 50 variables, 100 constraints



Modeling language

- uses disciplined convex programming (DCP) rules to verify curvature of expressions
- canonicalizes DCP-compliant optimization problems into second-order cone programs

```
# import QCML parsing library (named 'Scoop' for now)
from scoop import Scoop
p = Scoop()     # QCML parser object
p.rewrite("""   # QCML begin
  variable x
  variable z

  minimize square(x) - sqrt(z)
    x + z == 1
""")            # QCML end
```

- canonicalized as:

```
variable _z scalar
variable _x scalar
variable t0 scalar   # square(x)
variable t1 scalar   # sqrt(z)

minimize -t1 + t0
  norm([-t0 + 1.0; 2.0*_x]) <= t0 + 1.0
  norm(-_z + 1.0, 2.0*t1) <= _z + 1.0
  -_z + -_x + 1.0 == 0
```

- use as Python function (interpreted)

```
f = p.generate()   # returns solver function
sol = f()   # solves the problem instance
```

- use as external source code (compiled)

```
p.generate_source()   # writes solver source to file
```

Conclusions

scalability achieved by pairing modeling tool and code generator with large-scale solver

- solver source code (C) available at

  http://github.com/cvxgrp/pdos

- core C code is fewer than 100 lines
- competitive for low accuracy solutions

- modeling language source code (Python) available at

  http://github.com/cvxgrp/scoop

- minimal implementation of DCP rules
- full rewriting
- fewer than 1000 lines of Python