A New Architecture for Optimization Modeling Frameworks

Matt Wytock, Steven Diamond, Felix Heide
and Stephen Boyd
Stanford University

November 14, 2016
Convex optimization problem

minimize \( f_0(x) \)

subject to \( f_i(x) \leq 0, \quad i = 1, \ldots, m \)

\[ Ax = b, \]

with variable \( x \in \mathbb{R}^n \)

- 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
The one rule that DCP is based on

\[ h(f_1(x), \ldots, f_k(x)) \] is convex when \( h \) is convex and for each \( i \)

- \( h \) is increasing in argument \( i \), and \( f_i \) is convex, or
- \( h \) is decreasing in argument \( i \), and \( f_i \) is concave, or
- \( f_i \) is affine

- there’s a similar rule for concave compositions
  (just swap convex and concave above)
Traditional architecture for optimization frameworks

- Problem
- Canonicalization
- Standard form
- Matrix stuffing
- Sparse matrices
- Solver
- Solution
Standard (conic) form

minimize \( c^T x \)
subject to \( A x = b \)
\( x \in K \)

with variable \( x \in \mathbb{R}^n \)

- \( K \) is convex cone
  - \( x \in K \) is a generalized nonnegativity constraint
- linear objective, equality constraints
- special cases:
  - \( K = \mathbb{R}^n_+ \): linear program (LP)
  - \( K = \mathbb{S}^n_+ \): semidefinite program (SDP)
- general interface for solvers
Traditional cone solvers

▶ CVXOPT (Vandenberghe, Dahl, Andersen)
  ▶ interior-point method
  ▶ Python
▶ ECOS (Domahidi)
  ▶ interior-point method
  ▶ supports exponential cone
  ▶ compact, library-free C code
▶ SCS (O’Donoghue)
  ▶ first-order method
  ▶ parallelism with OpenMP
  ▶ GPU support
▶ others: GLPK, MOSEK, GUROBI, Cbc, Elemental, . . .
▶ traditional architecture has been enormously successful
  ▶ solvers based on interior point methods highly robust
  ▶ solvers portable to new platforms with linear algebra libraries
    ▶ BLAS, LAPACK, SuiteSparse, etc.
Drawbacks of traditional architecture

- for large problems, direct solutions to linear systems involving the $A$ matrix can be very expensive
- first-order methods (SCS) allow the use of indirect methods for linear solver subroutine
- but, representing all linear operators as sparse matrices can be inefficient
  - e.g., FFT-based convolution
- also, (most) existing solvers do not take advantage of modern platforms, e.g., GPUs, distributed
Graph-based architecture
Computation graphs

- computation graph for $f(x, y) = x^2 + 2x + y$

- simple transformations produce computation graphs for function gradient and adjoint
  - key operations in first-order and indirect solvers
Computation graph frameworks

- huge momentum and engineering effort from deep learning community
  - TensorFlow, Theano, Caffe, Torch, . . .
- support a wide variety of computational environments
  - CPU, GPU, distributed clusters, phones, . . .
- given a computation graph, existing frameworks implement gradient descent
- for optimization, first-order and indirect solvers fit naturally
- limited support for sparse matrix factorizations, which are required by interior point methods, direct solvers
Generating solver graphs

- solver generation implemented with functions parameterized by graphs or graph generators
- e.g., conjugate gradient for solving linear system $Ax = b$

```python
def cg_solve(A, b, x_init, tol=1e-8):
    delta = tol*norm(b)
    def body(x, k, r_norm_sq, r, p):
        Ap = A(p)
        alpha = r_norm_sq / dot(p, Ap)
        x = x + alpha*p
        r = r - alpha*Ap
        r_norm_sq_prev = r_norm_sq
        r_norm_sq = dot(r,r)
        beta = r_norm_sq / r_norm_sq_prev
        p = r + beta*p
        return (x, k+1, r_norm_sq, r, p)
    def cond(x, k, r_norm_sq, r, p):
        return tf.sqrt(r_norm_sq) > delta
    r = b - A(x_init)
    loop_vars = (x_init, tf.constant(0), dot(r, r), r, r)
    return tf.while_loop(cond, body, loop_vars)[:3]
```
Software implementation and numerical examples

- based on CVXPY, a convex optimization modeling framework
- solves convex problems using TensorFlow
- implements a variant of SCS, a first-order method
- linear subproblems solved with conjugate gradient
- experiment platform details
  - 32-core Intel Xeon 2.2Ghz processor
  - nVidia Titan X GPU with 12GB RAM
Nonnegative deconvolution example

\[
\begin{align*}
\text{minimize} \quad & \|c \ast x - b\|_2 \\
\text{subject to} \quad & x \geq 0,
\end{align*}
\]

with variable \( x \in \mathbb{R}^n \), problem data \( c \in \mathbb{R}^n \), \( b \in \mathbb{R}^{2n-1} \)

```python
from cvxpy import *
from cvxflow import scs_tf
x = Variable(n)
f = norm(conv(c, x) - b, 2)
prob = Problem(Minimize(f), [x >= 0])
scs_tf.solve(prob)
```
Comparison on nonnegative deconvolution

<table>
<thead>
<tr>
<th>Input size</th>
<th>Memory usage (GB)</th>
<th>GPU solve time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.36</td>
<td>2</td>
</tr>
<tr>
<td>1000</td>
<td>0.9</td>
<td>5.7</td>
</tr>
<tr>
<td>10000</td>
<td>1</td>
<td>10.4</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>13</td>
</tr>
</tbody>
</table>

- SCS Native
- SCS TensorFlow
Conclusions

- convex optimization is useful
- convex modeling languages make it easy
- graph-based architectures help it scale
- open source Python libraries available
  - cvxpy: cvxpy.org
  - cvxflow: github.com/cvxgrp/cvxflow
More details for nonnegative deconvolution

<table>
<thead>
<tr>
<th></th>
<th>small</th>
<th>medium</th>
<th>large</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables $n$</td>
<td>101</td>
<td>1001</td>
<td>10001</td>
</tr>
<tr>
<td>constraints $m$</td>
<td>300</td>
<td>3000</td>
<td>30000</td>
</tr>
<tr>
<td>nonzeros in $A$</td>
<td>9401</td>
<td>81601</td>
<td>6922001</td>
</tr>
</tbody>
</table>

**SCS native**

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
<th>CPU</th>
<th>GPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>solve time</td>
<td>0.1 secs</td>
<td>2.0 secs</td>
<td>2.2 secs</td>
<td>2.0 secs</td>
<td>260 secs</td>
</tr>
<tr>
<td>matrix build time</td>
<td>0.01 secs</td>
<td>0.01 secs</td>
<td>0.6 secs</td>
<td>0.6 secs</td>
<td>52 secs</td>
</tr>
<tr>
<td>memory usage</td>
<td>360 MB</td>
<td>360 MB</td>
<td>470 MB</td>
<td>470 MB</td>
<td>10.4 GB</td>
</tr>
<tr>
<td>objective</td>
<td>$1.38 \times 10^0$</td>
<td>$4.57 \times 10^0$</td>
<td>$1.41 \times 10^1$</td>
<td>$1.41 \times 10^1$</td>
<td></td>
</tr>
<tr>
<td>SCS iterations</td>
<td>380</td>
<td>100</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg. CG iterations</td>
<td>8.44</td>
<td>2.95</td>
<td>3.01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SCS TensorFlow**

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
<th>CPU</th>
<th>GPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>solve time</td>
<td>3.4 secs</td>
<td>5.7 secs</td>
<td>5.7 secs</td>
<td>3.2 secs</td>
<td>88 secs</td>
</tr>
<tr>
<td>graph build time</td>
<td>0.8 secs</td>
<td>0.8 secs</td>
<td>0.8 secs</td>
<td>0.8 secs</td>
<td>13 secs</td>
</tr>
<tr>
<td>memory usage</td>
<td>895 MB</td>
<td>984 MB</td>
<td>894 MB</td>
<td>984 MB</td>
<td>1.3 GB</td>
</tr>
<tr>
<td>objective</td>
<td>$1.38 \times 10^0$</td>
<td>$4.57 \times 10^0$</td>
<td>$1.41 \times 10^1$</td>
<td>$1.41 \times 10^1$</td>
<td></td>
</tr>
<tr>
<td>SCS iterations</td>
<td>480</td>
<td>100</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg. CG iterations</td>
<td>2.75</td>
<td>2.00</td>
<td>2.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>