Object-Oriented Convex Optimization with CVXPY

Stephen Boyd  Steven Diamond  Akshay Agrawal
Stanford University

BayOpt, Stanford, 5/19/18
Outline

Convex optimization

CVXPY 1.0

Parameters and warm-start

Distributed optimization

Summary
Convex optimization problem

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& \quad Ax = b,
\end{align*}
\]

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, CVXR
- 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)
Outline

Convex optimization

CVXPY 1.0

Parameters and warm-start

Distributed optimization

Summary
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
- mixes easily with general Python code, other libraries
- already used in many research projects, classes, companies
- over 100,000 downloads on PyPi
CVXPY 1.0

a complete redesign of CVXPY

▶ a modular framework for mapping problems into solver standard form
  ▶ recognizes QPs and targets specialized solvers
  ▶ supports complex numbers via rewriting as equivalent real-valued problem
▶ a unified system for defining variables and parameters with special properties, e.g.,
  ▶ nonnegative
  ▶ symmetric
  ▶ sparse
▶ full NumPy compatibility (matching syntax, etc.)
CVXPY 1.0 example

(constrained LASSO)

\[
\begin{align*}
\text{minimize} & \quad \|Ax - b\|_2^2 + \gamma\|x\|_1 \\
\text{subject to} & \quad 1^T x = 0, \quad \|x\|_{\infty} \leq 1
\end{align*}
\]

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

```python
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
```
Solvers

▶ ECOS (Domahidi)
   ▶ interior-point method
   ▶ supports exponential cone
   ▶ compact, library-free C code

▶ SCS (O’Donoghue)
   ▶ 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, …
Reductions

- mapping from problem to standard form is a series of reductions
- a reduction maps a problem into an equivalent one
- equivalent means a solution of one can be readily constructed from a solution of the other
- analogous to reduction in theoretical CS
Reductions

canonicalization

retrieval

$p_0 \quad p_1 \quad \ldots \quad p_n$

$s_0 \quad s_1 \quad \ldots \quad s_n$

solver
Example reductions

- flipping the objective from minimize to maximize
- adding slack variables
- changing variables
- monotone transformations of objective/constraints
- eliminating complex numbers
- dualizing
- pre-solve
Canonicalization of DCP programs

DCP programs are *canonicalized* to equivalent cone programs via the reductions

- conversion to Smith form
- relaxing convex equality constraints
- expanding nonlinear functions to graph implementations

(this was hard-coded in CVXPY < 1.0)
Overall framework

CVXPY 1.0 parses the problem, analyzes it, and dispatches to the most specialized solver reachable via known reductions.
Outline

Convex optimization

CVXPY 1.0

Parameters and warm-start

Distributed optimization

Summary
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:

```python
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

# 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))
Warm-start

- problem object caches solution and factorization
- solution used as initial guess
- factorization re-used if possible
- stateful serial approach versus stateless parallel approach
Performance

(LASSO)

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>for-loop</th>
<th>4 proc.</th>
<th>32 proc.</th>
<th>warm-start</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 core MacBook Pro</td>
<td>173 sec</td>
<td>70 sec</td>
<td>81 sec</td>
<td>43 sec</td>
</tr>
<tr>
<td>32 cores, Intel Xeon</td>
<td>527 sec</td>
<td>171 sec</td>
<td>45 sec</td>
<td>149 sec</td>
</tr>
</tbody>
</table>
Outline

Convex optimization

CVXPY 1.0

Parameters and warm-start

Distributed optimization

Summary
Sum of problems

- overload + for problem objects
  - objectives add (if both minimize or maximize)
  - constraints lists add (i.e., concatenate)

```python
problem1 = Problem(Minimize(abs(x+y), [x>=2]))
problem2 = Problem(Minimize(square(y+z), [y>=1]))
problem = problem1 + problem2
print(problem)
# Problem(Minimize(abs(x+y)+square(y+z)), [x>=2,y>=1])
```
Separable problems

- fully separable problems (can be) solved in parallel
- groups objective terms, constraints with same variables

```python
problem1 = Problem(Minimize(abs(x)), [x >= 2])
problem2 = Problem(Minimize(y**2), [y >= 1])

# Solve in parallel
(problem1 + problem2).solve(parallel=True)

# Solve serially
problem1.solve()
problem2.solve()
```
**fix function**

- replaces variables in a given list with parameters with the same value
- create expression

  # An expression with variables x and z.
  expr1 = sum_squares(A*x - b) + norm(z, 1)
  expr1.variables() # [x, z]

- use fix function

  # Fix expr1 with respect to z.
  # z is replaced with Parameter(value=z.value).
  expr2 = fix(expr1, [z])
  expr2.variables() # [x]
Alternating direction method of multipliers

- problem

\[
\begin{align*}
\text{minimize} & \quad f(x) + g(z) \\
\text{subject to} & \quad Ax + Bz = c
\end{align*}
\]

- augmented Lagrangian

\[
L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \left(\frac{\rho}{2}\right)\|Ax + Bz - c\|_2^2
\]

- ADMM:

\[
\begin{align*}
x^{k+1} & := \arg\min_x L_\rho(x, z^k, y^k) \\
z^{k+1} & := \arg\min_z L_\rho(x^{k+1}, z, y^k) \\
y^{k+1} & := y^k + \rho(Ax^{k+1} + Bz^{k+1} - c)
\end{align*}
\]
Generic ADMM in CVXPY

- form and solve original problem

  ```python
  Problem(Minimize(f + g), [A*x + B*z == c]).solve()
  ```

- ADMM in CVXPY:

  ```python
  resid = A*x + B*z - c
  y = Parameter(m, value=zeros(m))
  aug_lagr = f+g+y.T*resid+(rho/2)*sum_squares(resid)
  for k in range(MAX_ITERS):
    Problem(Minimize(fix(aug_lagr, [z]))).solve()
    Problem(Minimize(fix(aug_lagr, [x]))).solve()
    y.value += rho*resid.value
  ```
Consensus optimization

- want to solve problem with $N$ objective terms

\[
\text{minimize } \sum_{i=1}^{N} f_i(x)
\]

- e.g., $f_i$ is the loss function for $i$th block of training data

- consensus form:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{N} f_i(x_i) \\
\text{subject to} & \quad x_i = z
\end{align*}
\]

- $x_i$ are local variables
- $z$ is the global variable
- $x_i = z$ are consistency or consensus constraints
ADMM consensus

- alternating direction method of multipliers (ADMM) consensus:
  \[
  x_{i}^{k+1} := \arg\min_{x_{i}} \left( f_i(x_i) + \frac{\rho}{2} \|x_i - \bar{x}^k + u_i^k\|_2^2 \right)
  \]
  \[
  u_{i}^{k+1} := u_{i}^k + x_{i}^{k+1} - \bar{x}_{k+1}
  \]

- \( \bar{x}^k = (1/N) \sum_{i=1}^{N} x_i^k \)
- parameter \( \rho \geq 0 \)

- split across \( N \) worker processes

- in each iteration
  - update \( x_i \) locally (in each worker process, in parallel)
  - gather \( x_i \) on master process, average to get \( \bar{x} \)
  - scatter \( \bar{x} \) to workers
  - update \( u_i \) locally

Distributed optimization 31
ADMM consensus: the workers

- launch $N$ worker processes
- pipe to communicate with master

```
def run_worker(f, pipe):
    xbar = Parameter(n, value=zeros(n))
    u = Parameter(n, value=zeros(n))
    f += (rho/2)*sum_squares(x - xbar + u)
    prox = Problem(Minimize(f))
    # ADMM loop.
    while True:
        prox.solve()
        pipe.send(x.value)
        xbar.value = pipe.recv()
        u.value += x.value - xbar.value
```
ADMM consensus: the master

- master gathers $x_i$ and scatters $\bar{x}$

```python
# pipes = list with pipe to each worker
for i in range(MAX_ITER):
    # Gather and average xi
    xbar = sum([pipe.recv() for pipe in pipes]) / N
    # Scatter xbar
    [pipe.send(xbar) for pipe in pipes]
```
Consensus SVM

- data \((a_i, b_i), i = 1, \ldots, N, a_i \in \mathbb{R}^n, b_i \in \{-1, +1\}\)
- linear classifier \(\text{sign}(a^T w + v)\), with weight \(w\), offset \(v\)
- choose \(w, v\) to minimize \(\frac{1}{N} \sum_{i=1}^{N} (1 - b_i(a_i^T w + v))_+ + \lambda \|w\|^2_2\)
- split data and use ADMM consensus to solve
Example

- $N = 10^6$ samples
- $n = 10^3$ (dense) features
- 6 hours to solve serially using CVXPY and SCS
- 20 seconds to solve with ADMM consensus
  - split over 100 processes
  - 32 cores, Intel Xeon CPUs
  - 2 sec per ADMM iteration
  - 10 iterations to converge

Distributed optimization
Example

Distributed optimization
Outline

Convex optimization

CVXPY 1.0

Parameters and warm-start

Distributed optimization

Summary

Summary
Summary

- object-oriented convex optimization
  - is close to the mathematics
  - ...and extremely practical
- CVXPY mixes well with high level Python
  - parallelism
  - object oriented design
- CVXPY is building block for
  - distributed optimization
  - nonconvex optimization
  - domain-specific application packages