Convex Optimization in Python with CVXPY

Steven Diamond    Stephen Boyd
Stanford University

TCMM 2014, September 2014
Outline

Convex optimization

Convex modeling languages

CVXPY

Image in-painting

Trade-off curve, in parallel

Single commodity flow

Summary
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
Outline

Convex optimization

Convex modeling languages

CVXPY

Image in-painting

Trade-off curve, in parallel

Single commodity flow

Summary
Convex modeling languages

▶ long tradition of modeling languages for optimization
  ▶ cf. AMPL, GAMS
▶ modeling languages for convex optimization
  ▶ e.g., CVX, YALMIP, CVXGEN, QCML
▶ 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, nonnegative (convex), nonpositive (concave)
- expressions formed from
  - variables (curvature: affine, unknown sign)
  - constants (curvature: constant, known sign)
  - library of atoms with known curvature and sign (as function of their arguments)
- more at dcp.stanford.edu
Standard (conic) form

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad Ax = b \\
& \quad x \in K
\end{align*}
\]

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
Outline

Convex optimization

Convex modeling languages

CVXPY

Image in-painting

Trade-off curve, in parallel

Single commodity flow

Summary
CVXPY

a modeling language in Python for convex optimization

- translates from math to standard form used by solvers
- uses DCP to verify convexity
- open source all the way from the solvers
- supports parameterized problems
- mixes easily with general Python code, other libraries
- already used in many research projects and two classes
- over 7000 downloads on PyPi
CVXPY solvers

- all open source
- CVXOPT (Vandenberghe, Dahl, Andersen)
  - interior-point method
  - in Python
- ECOS (Domahidi)
  - interior-point method
  - compact, library-free C code
- SCS (O’Donoghue)
  - first-order method
  - native support of exponential cone
  - parallelism with OpenMP
CVXPY example

(constrained LASSO)

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

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

```
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

Convex optimization

Convex modeling languages

CVXPY

Image in-painting

Trade-off curve, in parallel

Single commodity flow

Summary
Image in-painting

- guess pixel values in obscured/corrupted parts of image

- *total variation in-painting*: choose pixel values $x_{ij} \in \mathbb{R}^3$ to minimize
  \[ TV(x) = \sum_{ij} \left\| \begin{bmatrix} x_{i+1,j} - x_{ij} \\ x_{i,j+1} - x_{ij} \end{bmatrix} \right\|_2 \]

- a convex problem
Example

- 512 × 512 color image
- denote corrupted pixels with $K \in \{0, 1\}^{512 \times 512}$
  - $K_{ij} = 1$ if pixel value is known
  - $K_{ij} = 0$ if unknown
- $X_{corr} \in \mathbb{R}^{512 \times 512 \times 3}$ is corrupted image
from cvxpy import *
variables = []
constr = []
for i in range(3):
    X = Variable(512, 512)
    variables += [X]
    constr += [mul_elemwise(K, X - X_corr[:,:,i]) == 0]
prob = Problem(Minimize(tv(*variables)), constr)
prob.solve(solver=SCS)
Example

Image in-painting

Original

Corrupted

Lorem ipsum dolor sit amet, adipiscing elit, sed diam non euismod tincidunt ut laoreet magna aliquam erat volutpat enim ad minim veniam, quis exerci tation ullamcorper sus lobo rites nisl ut aliquip ex ea conseqvaut. Duis autem vel eu dolor in hendrerit in vulputat esse molestie consequat, vel illum dolore eu feugiat nulla facilisi vero eros et accumsan et justo dignissim qui blandit praesent luptatum zzril dolores amets duis doran.
Example

Image in-painting
Example (80% of pixels removed)

Image in-painting
Example (80% of pixels removed)
Outline

Convex optimization

Convex modeling languages

CVXPY

Image in-painting

Trade-off curve, in parallel

Single commodity flow

Summary
Parameters

» symbolic representations of constants
» fixed sign and dimensions
» change value of constant without rebuilding problem
# Positive scalar parameter.
gamma = Parameter(sign="positive")

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

# Matrix parameter with negative entries.
G = Parameter(4, 7, sign="negative")

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

(LASSO)

\[
\begin{align*}
\text{minimize} \quad & \|Ax - b\|_2^2 + \gamma \|x\|_1 \\
\text{with variable} \quad & x \in \mathbb{R}^n
\end{align*}
\]

\[
x = \text{Variable}(n) \\
gamma = \text{Parameter(sign="positive")} \\
\text{error} = \text{sum_squares}(A*x-b) \\
\text{regularization} = \gamma \|x\|_1 \\
\text{prob} = \text{Problem(Minimize(error + regularization))}
\]
For loop style trade-off curve

compute a trade-off curve by updating parameter gamma

```python
x_values = []
for val in numpy.logspace(-4, 2):
    gamma.value = val
    prob.solve()
    x_values.append(x.value)
```
Trade-off curve for LASSO
Entries of $x$ versus $\gamma$: (regularization path)
Parallel style trade-off curve

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

# Assign a value to gamma and find the 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))
Performance

- Lasso with $A \in \mathbb{R}^{1000 \times 500}$, 100 values of $\gamma$
- single thread time for one LASSO: 4 seconds
- performance using solver SCS:

<table>
<thead>
<tr>
<th></th>
<th>For loop</th>
<th>4 processes</th>
<th>32 processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 core MacBook Pro</td>
<td>403 sec</td>
<td>147 sec</td>
<td>136 sec</td>
</tr>
<tr>
<td>32 cores, Intel Xeon</td>
<td>619 sec</td>
<td>175 sec</td>
<td>56 sec</td>
</tr>
</tbody>
</table>
Outline

Convex optimization

Convex modeling languages

CVXPY

Image in-painting

Trade-off curve, in parallel

Single commodity flow

Summary
Single commodity flow

- directed graph with \( p \) nodes, \( n \) edges
- flow \( f_i \) on edge \( i \)
- external source/sink flow \( s_j \) at node \( j \)
- single commodity flow problem:

\[
\text{minimize} \quad \sum_{i=1}^{n} \phi_i(f_i) + \sum_{j=1}^{p} \psi_j(s_j), \\
\text{subject to} \quad \text{zero net flow at each node}
\]

- variables are \( f_i, s_j \)
- \( \phi_i \) convex flow cost functions
- \( \psi_j \) convex source cost functions
- can include constraints in \( \phi_i, \psi_j \)
Matrix representation

- node incidence matrix $A \in \mathbb{R}^{p \times n}$

$$A_{ij} = \begin{cases} 
+1 & \text{edge } i \text{ leaves node } j \\
-1 & \text{edge } i \text{ enters node } j \\
0 & \text{otherwise.}
\end{cases}$$

- zero net flow at each node: $Af = s$

- final problem:

$$\text{minimize } \sum_{i=1}^{n} \phi_i(f_i) + \sum_{j=1}^{p} \psi_j(s_j),$$

subject to $Af = s$
Object-oriented representation

- node object includes source, cost, source/net flow constraints
- edge object includes flow, cost, flow constraints
- solve the problem:
  ```python
  cost = sum([object.cost for object in nodes + edges])
  obj = Minimize(cost)
  constraints = []
  for object in nodes + edges:
    constraints += object.constraints()
  Problem(obj, constraints).solve()
  ```

Single commodity flow
class Node(object):
    def __init__(self, cost):
        self.source = Variable()
        self.cost = cost(self.source)
        self.edge_flows = []

    def constraints(self):
        """The constraint net flow == 0."""
        net_flow = sum(self.edge_flows) + self.source
        return [net_flow == 0]
class Edge(object):
    def __init__(self, cost):
        self.flow = Variable()
        self.cost = cost(self.flow)

    def connect(self, in_node, out_node):
        """Connects two nodes via the edge."""
        in_node.edge_flows.append(-self.flow)
        out_node.edge_flows.append(self.flow)
Example

- 7-by-7 grid of nodes
- 1 unit of flow sent from source $s_1$ to sink $s_n$:
  - $s_1 = +1$
  - $s_n = -1$
  - $s_i = 0$ for $i = 2, \ldots, n - 1$
- flow cost $\phi_i(f_i) = w_i (|f_i| + \lambda f_i^2)$
  - weights $w_i > 0$ randomly chosen
Shortest path ($\lambda = 0$)
Diffusion \((\lambda = +\infty)\)
Diffusion with sparsity ($\lambda = 1$)
Outline

Convex optimization

Convex modeling languages

CVXPY

Image in-painting

Trade-off curve, in parallel

Single commodity flow

Summary
Summary

- convex optimization is easy with CVXPY
- mixes well with high level Python
  - parallelism
  - object oriented design
- building block for
  - distributed optimization
  - nonconvex optimization
Future work

- not just for prototyping
- speed and scalability
- abstract linear operators