Domain Specific Languages
for Convex Optimization

Stephen Boyd
joint work with E. Chu, J. Mattingley, M. Grant
Electrical Engineering Department, Stanford University

ROKS 2013, Leuven, 9 July 2013
Outline

Convex optimization

Constructive convex analysis

Cone representation

 Canonicalization

Parser/solvers and parser/generators

Conclusions
Outline

Convex optimization

Constructive convex analysis

Cone representation

Canonicalization

Parser/solvers and parser/generators

Conclusions
Convex optimization problem — standard form

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

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
Convex optimization problem — conic form

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

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

- \( \mathcal{K} \) is convex cone
  - \( x \in \mathcal{K} \) is a generalized nonnegativity constraint
- linear objective, equality constraints

- special cases:
  - \( \mathcal{K} = \mathbb{R}_+^n \): linear program (LP)
  - \( \mathcal{K} = \mathbb{S}_+^n \): semidefinite program (SDP)
- the modern canonical form
Why convex optimization?

- beautiful, fairly complete, and useful theory
Why convex optimization?

- beautiful, fairly complete, and useful theory
- solution algorithms that work well in theory and practice
  - convex optimization is actionable
Why convex optimization?

- beautiful, fairly complete, and useful theory
- solution algorithms that work well in theory and practice
  - convex optimization is actionable
- many applications in
  - control
  - combinatorial optimization
  - signal and image processing
  - communications, networks
  - circuit design
  - machine learning, statistics
  - 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

- transform your problem into a standard form, and use a standard solver
  - extends reach of problems solvable by standard solvers

- **this talk:** methods to formalize and automate last approach
Outline

Convex optimization

Constructive convex analysis

Cone representation

Canonicalization

Parser/solvers and parser/generators

Conclusions
How can you tell if a problem is convex?

approaches:

- use basic definition, first or second order conditions, e.g.,
  $\nabla^2 f(x) \succeq 0$

- via convex calculus: construct $f$ using
  - library of basic functions that are convex
  - calculus rules or transformations that preserve convexity
Convex functions: Basic examples

- $x^p$ ($p \geq 1$ or $p \leq 0$), $-x^p$ ($0 \leq p \leq 1$)
- $e^x$, $-\log x$, $x \log x$
- $a^T x + b$
- $x^T P x$ ($P \succeq 0$)
- $\|x\|$ (any norm)
- $\max(x_1, \ldots, x_n)$
Convex functions: Less basic examples

- $x^T x/y$ ($y > 0$), $x^T Y^{-1} x$ ($Y \succ 0$)
- $\log(e^{x_1} + \cdots + e^{x_n})$
- $-\log \Phi(x)$ ($\Phi$ is Gaussian CDF)
- $\log \det X^{-1}$ ($X \succ 0$)
- $\lambda_{\text{max}}(X)$ ($X = X^T$)
Calculus rules

- **nonnegative scaling**: \( f \) convex, \( \alpha \geq 0 \) \( \implies \alpha f \) convex

- **sum**: \( f, h \) convex \( \implies f + g \) convex

- **affine composition**: \( f \) convex \( \implies f(Ax + b) \) convex

- **pointwise maximum**: \( f_1, \ldots, f_m \) convex \( \implies \max_i f_i(x) \) convex

- **partial minimization**: \( f(x, y) \) convex \( \implies \inf_y f(x, y) \) convex

- **composition**: \( h \) convex increasing, \( f \) convex \( \implies h(f(x)) \) convex
Examples

from basic functions and calculus rules, we can show convexity of . . .

- piecewise-linear function: \( \max_{i=1,...,k}(a_i^T x + b_i) \)
- \( \ell_1 \)-regularized least-squares cost: \( \|Ax - b\|_2^2 + \lambda \|x\|_1 \), with \( \lambda \geq 0 \)
- sum of largest \( k \) elements of \( x \): \( x[1] + \cdots + x[k] \)
A general composition rule

\[ h(f_1(x), \ldots, f_k(x)) \] 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
- this one rule subsumes most of the others
- in turn, it can be derived from the partial minimization rule
Constructive convexity verification

- start with function given as **expression**
- build parse tree for expression
  - leaves are variables or constants/parameters
  - nodes are functions of children, following general rule
- tag each subexpression as convex, concave, affine, constant
  - variation: tag subexpression signs, use for monotonicity
  - *e.g.*, $(\cdot)^2$ is increasing if its argument is nonnegative
- sufficient (but not necessary) for convexity
Example

for $x < 1, y < 1$

\[
\frac{(x - y)^2}{1 - \max(x, y)}
\]

is convex

- (leaves) $x$, $y$, and 1 are affine expressions
- $\max(x, y)$ is convex; $x - y$ is affine
- $1 - \max(x, y)$ is concave
- function $u^2/v$ is convex, monotone decreasing in $v$ for $v > 0$
  hence, convex with $u = x - y$, $v = 1 - \max(x, y)$
Example

- \( f(x) = \sqrt{1 + x^2} \) is convex

- but cannot show this using constructive convex analysis
  - (leaves) 1 is constant, \( x \) is affine
  - \( x^2 \) is convex
  - \( 1 + x^2 \) is convex
  - but \( \sqrt{1 + x^2} \) doesn’t match general rule

- writing \( f(x) = \| (1, x) \|_2 \), however, works
  - \( (1, x) \) is affine
  - \( \| (1, x) \|_2 \) is convex
Disciplined convex programming (DCP)

- framework for describing convex optimization problems
- based on constructive convex analysis
- sufficient but not necessary for convexity
- basis for several domain specific languages and tools for convex optimization
Disciplined convex program: Structure

A DCP has

- zero or one **objective**, with form
  - minimize \( \{ \text{scalar convex expression} \} \) or
  - maximize \( \{ \text{scalar concave expression} \} \)

- zero or more **constraints**, with form
  - \( \{ \text{convex expression} \} \leq \{ \text{concave expression} \} \) or
  - \( \{ \text{concave expression} \} \geq \{ \text{convex expression} \} \) or
  - \( \{ \text{affine expression} \} = \{ \text{affine expression} \} \)
Disciplined convex program: Expressions

- expressions formed from
  - variables,
  - constants/parameters,
  - and functions from a library
- library functions have known convexity, monotonicity, and sign properties
- all subexpressions match general composition rule
Disciplined convex program

- a valid DCP is
  - convex-by-construction (cf. posterior convexity analysis)
  - ‘syntactically’ convex (can be checked ‘locally’)

- convexity depends only on attributes of library functions, and not their meanings
  - e.g., could swap $\sqrt{\cdot}$ and $4\sqrt{\cdot}$, or $\exp \cdot$ and $(\cdot)_+$, since their attributes match
Outline

Convex optimization

Constructive convex analysis

Cone representation

Canonicalization

Parser/solvers and parser/generators

Conclusions
Cone representation

(Nesterov, Nemirovsky)

**cone representation** of (convex) function $f$:

- $f(x)$ is optimal value of cone program

\[
\begin{align*}
\text{minimize} \quad & c^T x + d^T y + e \\
\text{subject to} \quad & A \begin{bmatrix} x \\ y \end{bmatrix} = b, \quad \begin{bmatrix} x \\ y \end{bmatrix} \in \mathcal{K}
\end{align*}
\]

- cone program in $(x, y)$, we but minimize only over $y$
- *i.e.*, we define $f$ by partial minimization of cone program
Examples

- \( f(x) = -(xy)^{1/2} \) is optimal value of SDP

\[
\begin{align*}
\text{minimize} & \quad -t \\
\text{subject to} & \quad \begin{bmatrix} x & t \\ t & y \end{bmatrix} \succeq 0
\end{align*}
\]

with variable \( t \)

- \( f(x) = x[1] + \cdots + x[k] \) is optimal value of LP

\[
\begin{align*}
\text{minimize} & \quad 1^T \lambda - k \nu \\
\text{subject to} & \quad x + \nu 1 = \lambda - \mu \\
& \quad \lambda \succeq 0, \quad \mu \succeq 0
\end{align*}
\]

with variables \( \lambda, \mu, \nu \)
SDP representations

Nesterov, Nemirovsky, and others have worked out SDP representations for many functions, e.g.,

- $x^p$, $p \geq 1$ rational
- $-(\det X)^{1/n}$
- $\sum_{i=1}^{k} \lambda_i(X) \ (X = X^T)$
- $\|X\| = \sigma_1(X) \ (X \in \mathbb{R}^{m \times n})$
- $\|X\|_* = \sum_i \sigma_i(X) \ (X \in \mathbb{R}^{m \times n})$

some of these representations are not obvious . . .
Canonicalization

- start with problem in DCP form, with cone representable library functions
- automatically transform to equivalent cone program
Canonicalization: How it’s done

- for each (non-affine) library function $f(x)$ appearing in parse tree, with cone representation

\[
\begin{align*}
&\text{minimize } \quad c^T x + d^T y + e \\
&\text{subject to } \quad A \begin{bmatrix} x \\ y \end{bmatrix} = b, \quad \begin{bmatrix} x \\ y \end{bmatrix} \in \mathcal{K}
\end{align*}
\]

- add new variable $y$, and constraints above
- replace $f(x)$ with affine expression $c^T x + d^T y + e$

- yields problem with linear equality and cone constraints
- DCP ensures equivalence of resulting cone program
Outline

Convex optimization

Constructive convex analysis

Cone representation

Canonicalization

Parser/solvers and parser/generators

Conclusions
Parser/solvers and parser/generators

- **parser/solver** (CVX, YALMIP)
  - canonicalize problem *instance* (with numeric parameters)
  - solve using cone program solver
Parser/solvers and parser/generators

- **parser/solver** (CVX, YALMIP)
  - canonicalize problem *instance* (with numeric parameters)
  - solve using cone program solver

- **parser/generator** (CVXGEN, QCML)
  - canonicalize problem *family* (with symbolic parameters)
  - generate mapping from original problem to cone program
  - connect to generic (QCML) or custom (CVXGEN) cone program solver
Example

- constrained least-squares problem with $\ell_1$ regularization

\[
\text{minimize} \quad \|Ax - b\|_2^2 + \lambda \|x\|_1 \\
\text{subject to} \quad \|x\|_\infty \leq 1
\]

- variable $x \in \mathbb{R}^n$
- constants/parameters $A, b, \lambda > 0$
CVX

- parser/solver (M. Grant)
- embedded in Matlab; targets multiple cone solvers

- CVX specification for example problem:

```matlab
cvx_begin
    variable x(n)  % declare vector variable
    minimize (sum(square(A*x-b,2)) + lambda*norm(x,1))
    subject to norm(x,inf) <= 1
cvx_end
```

- here \( A, b, \lambda \) are constants
Some functions in the CVX library

<table>
<thead>
<tr>
<th>function</th>
<th>meaning</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm(x, p)</td>
<td>$|x|_p$, $p \geq 1$</td>
<td>cvx</td>
</tr>
<tr>
<td>square(x)</td>
<td>$x^2$</td>
<td>cvx</td>
</tr>
<tr>
<td>square_pos(x)</td>
<td>$(x_+)^2$</td>
<td>cvx, nondecr</td>
</tr>
<tr>
<td>pos(x)</td>
<td>$x_+$</td>
<td>cvx, nondecr</td>
</tr>
<tr>
<td>sum_largest(x,k)</td>
<td>$x[1] + \cdots + x[k]$</td>
<td>cvx, nondecr</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>$\sqrt{x}$, $x \geq 0$</td>
<td>ccv, nondecr</td>
</tr>
<tr>
<td>inv_pos(x)</td>
<td>$1/x$, $x &gt; 0$</td>
<td>cvx, nonincr</td>
</tr>
<tr>
<td>max(x)</td>
<td>$\max{x_1, \ldots, x_n}$</td>
<td>cvx, nondecr</td>
</tr>
<tr>
<td>quad_over_lin(x,y)</td>
<td>$x^2/y$, $y &gt; 0$</td>
<td>cvx, nonincr in y</td>
</tr>
<tr>
<td>lambda_max(X)</td>
<td>$\lambda_{\text{max}}(X)$, $X = X^T$</td>
<td>cvx</td>
</tr>
<tr>
<td>huber(x)</td>
<td>$\begin{cases} x^2, &amp;</td>
<td>x</td>
</tr>
</tbody>
</table>
CVXGEN

- parser-generator (J. Mattingley)
- domain specific input
- emits flat C source that solves problem family
- goal:
  - spend (perhaps much) time generating code
  - save (hopefully much) time solving problem instances
CVXGEN specification

- CVXGEN specification for example problem:

parameters
  lambda positive
  A(m,n)
  b(m)
end

variables
  x(n)
end

minimize
  sum(square(A*x - b)) + lambda*norm1(x)
subject to
  norm_inf(x) <= 1
end

- here A, b, λ are **symbolic parameters**
Sample solve times for CVXGEN generated code

(on quad-core 3.4GHz Xeon with 16GB of RAM)

<table>
<thead>
<tr>
<th>problem</th>
<th>vars</th>
<th>constrs</th>
<th>SDPT3 (ms)</th>
<th>CVXGEN (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>portfolio</td>
<td>110</td>
<td>111</td>
<td>350</td>
<td>0.4</td>
</tr>
<tr>
<td>svm</td>
<td>111</td>
<td>200</td>
<td>510</td>
<td>0.6</td>
</tr>
<tr>
<td>generator</td>
<td>286</td>
<td>620</td>
<td>470</td>
<td>1.5</td>
</tr>
<tr>
<td>battery</td>
<td>144</td>
<td>289</td>
<td>310</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Quadratic cone modeling language (QCML)

- parser/generator (E. Chu)
- domain specific input; parser embedded in Python
- targets CVXOPT in Python
- can generate source code for several targets
- goal: seamless transition from prototyping to code generation
**QCML specification**

- full Python source

```python
from qcml import QCML
p = QCML()  # QCML parser object
p.parse(""")  # QCML begin
    dimensions m n
parameters A(m,n) b(m)
parameter lambda positive
variable x(n)
minimize sum(square(A*x - b)) + lambda*norm1(x)
    norm_inf(x) <= 1
""")  # QCML end
# canonicalize the problem
p.canonicalize()
```

Parser/solvers and parser/generators 38
Using QCML as parser/solver

- once canonicalized, create a Python solver

```python
p.codegen("cvxopt")  # creates Python source code
f = p.solver          # bytecode for solver function
```
Using QCML as parser/solver

- once canonicalized, create a Python solver
  
  ```python
  p.codegen("cvxopt")  # creates Python source code
  f = p.solver          # bytecode for solver function
  ```

- f is a Python function mapping parameters into solutions
  
  ```python
  sol = f(params)       # solution for problem instance
  ```
  
  - params is a dictionary holding parameter values
  - sol is a dictionary holding optimal value, solver status, . . .
Using QCML as parser/solver

- once canonicalized, create a Python solver
  
  ```python
  p.codegen("cvxopt")  # creates Python source code
  f = p.solver            # bytecode for solver function
  ```

- `f` is a Python function mapping parameters into solutions
  
  ```python
  sol = f(params)  # solution for problem instance
  ```
  
  - `params` is a dictionary holding parameter values
  - `sol` is a dictionary holding optimal value, solver status, ...

- combine canonicalize, codegen, and solver
  
  ```python
  sol = p.solve(params)
  ```
  
  - recreates CVX-like functionality
Using QCML as parser/generator

- once canonicalized, create external source code
  
  ```python
p.codegen("ecos")  # creates C solver source code
  ```
Using QCML as parser/generator

- once canonicalized, create external source code
  ```python
  p.codegen("ecos")  # creates C solver source code
  ```

- generates folder with
  - C source that maps problem parameters into SOCP
  - C source that maps SOCP solution into problem solution
  - Makefile

- links with external solver, in this case, ECOS
Using QCML as parser/generator

- once canonicalized, create external source code
  \[ \text{p.codegen("ecos")} \quad \text{# creates C solver source code} \]

- generates folder with
  - C source that maps problem parameters into SOCP
  - C source that maps SOCP solution into problem solution
  - Makefile

- links with external solver, in this case, ECOS

- recreates CVXGEN-like functionality
Using QCML as parser/generator

- once canonicalized, create external source code
  
  ```python
  p.codegen("ecos")  # creates C solver source code
  ```

- generates folder with
  - C source that maps problem parameters into SOCP
  - C source that maps SOCP solution into problem solution
  - Makefile

- links with external solver, in this case, ECOS

- recreates CVXGEN-like functionality

- (eventually) target custom deployment context
  - embedded systems, GPGPU, clusters, ...
Outline

Convex optimization

Constructive convex analysis

Cone representation

Canonicalization

Parser/solvers and parser/generators

Conclusions
Conclusions

- DCP is a formalization of constructive convex analysis
  - simple method to certify problem as convex
  - basis of several domain specific languages for convex optimization

- parser/solvers make rapid prototyping easy

- parser/generators yield solvers that
  - are extremely fast
  - can be embedded in real-time applications

- hybrid solution unifies prototyping and deployment
References

- *Disciplined Convex Programming* (Grant, Boyd, Ye)
- *Graph Implementations for Nonsmooth Convex Programs* (Grant, Boyd)
- *Automatic Code Generation for Real-Time Convex Optimization* (Mattingley, Boyd)
- *Code Generation for Embedded Second-Order Cone Programming* (Chu, Parikh, Domahidi, Boyd)

- CVX (Grant, Boyd)
- CVXGEN (Mattingley, Boyd)
- QCML (Chu, Boyd)