Cone Representations, Languages, and Compilers for Convex Optimization

Stephen Boyd

joint work with Michael Grant and Jacob Mattingley

Electrical Engineering Department, Stanford University

Berkeley Optimization Day, 3/6/2010

Outline

- Convex optimization
- Constructive convex analysis
- Cone programming and representations
- Transforming to cone program
- Parser/solvers
- Code generation

Convex optimization problem — standard form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{array}$$

with variable $x \in \mathbf{R}^n$

- objective and inequality constraint functions $f_0, \ldots, f_m$ are convex
- equality constraints are linear
- examples:
  - least-squares, least-squares with ℓ_1 regularization
  - linear program (LP), quadratic program (QP)
  - maximum entropy and related problems

Why convex optimization?

- beautiful, fairly complete, and useful theory
- solution algorithms that work well in theory and practice
- **many applications** recently discovered 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, SDP, . . .)
  - 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 that can be solved using standard solvers
  - transformation can be hard to find, cumbersome to carry out

this talk: methods to formalize and automate the last approach

Outline

- Convex optimization
- Constructive convex analysis
- Cone programming and representations
- Transforming to cone program
- Parser/solvers
- Code generation

How can you tell if a problem is convex?

need to check convexity of a function

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

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

Calculus rules

- *nonnegative scaling*: f convex, $\alpha \geq 0 \implies \alpha f$ convex
- *sum*: f, g 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
- *perspective transformation*: f convex $\implies tf(x/t)$ convex for $t > 0$

Examples

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

- piecewise-linear function: $\max_{i=1,\ldots,k}(a_i^T x + b_i)$
- ℓ_1-regularized least-squares cost: $\|Ax - b\|_2^2 + \lambda\|x\|_1$, with $\lambda \geq 0$
- sum of largest k elements of x: $x_{[1]} + \cdots + x_{[k]}$
- distance to convex set C: $\mathbf{dist}(x, C) = \inf_{y \in C} \|x - y\|_2$

A general composition rule

- $h(f_1(x), \ldots, f_k(x))$ is convex if h is, and for each i,
  - f_i is affine, or
  - f_i is convex and h is nondecreasing in its ith arg, or
  - f_i is concave and h is nonincreasing in its ith arg

- this one rule subsumes most of the others

- in turn, it can be derived from the partial minimization rule

Constructive convexity verification

- build parse tree for function (expression)
- leaves are variables or constants
- nodes are composition functions of children, following general rule
- example: $(x-y)^2/(1-\max(x,y))$ is convex (for $x<1,\ y<1$)
  - (leaves) x, y, and 1 are affine functions
  - $\max(x,y)$ 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, get convex function with $u=x-y$, $v=1-\max(x,y)$

Disciplined convex program

- convex optimization problem described as
  - objective: minimize (cvx expr) or maximize (ccv expr)
  - inequality constraints: cvx expr $\leq$ ccv expr or ccv expr $\geq$ ccv expr
  - equality constraints: aff expr $=$ aff expr
- (convex, concave, affine) expressions formed from constants, variables, and functions using general composition rule
- functions come from a library, with known convexity, monotonicity properties
- DCP is convex-by-construction (cf. posterior convexity analysis)

(Automatic) parsing of DCP

- it's (relatively) easy to parse a DCP, given function library
- DCP is 'syntactically convex'; convexity hinges only on convexity, monotonicity attributes of functions, not their detailed meaning
- gives basic method for problem convexity detection/certification
- we'll see later another use of the resulting parse trees . . .

Outline

- Convex optimization
- Constructive convex analysis
- Cone programming and representations
- Transforming to cone program
- Parser/solvers
- Code generation

Convex optimization problem — conic form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \mathcal{K} \end{array}$$

with variable $x \in \mathbf{R}^n$

- objective is linear
- constraints are linear equalities and (generalized) nonnegativity
- $\mathcal{K}$ is convex cone
- examples:
  - LP: $\mathcal{K} = \mathbf{R}^n_+$
  - semidefinite program (SDP): $\mathcal{K} = \mathbf{S}^n_+$ (PSD matrices)

Cone programming

- symmetric cone programming: $\mathcal{K}$ is product of
  - $\mathbf{R}^k$ ('unconstrained variables')
  - nonnegative orthant $\mathbf{R}^k_+$
  - Lorentz cones $\mathcal{L}^k = \{(z,t) \in \mathbf{R}^k \times \mathbf{R} \mid \|z\|_2 \leq t\}$
  - semidefinite cones $\mathbf{S}^k_+$

  of various dimensions
- exponential cone: $\mathcal{E} = \{(x,y,t) \mid \exp(x/t) \leq y/t,\ t > 0\}$
- with these cones, can express almost any convex problem that arises in applications (we'll see how, shortly)

Cone programming solvers

- theory, algorithms for cone programming well developed in last 10 years
- software for symmetric cone programming widely available and used
  - SeDuMi, SDPT3 (open source; Matlab/C)
  - CSDP, SDPA (open source; C)
  - CVXOPT (open source; Python/C)
  - MOSEK, PENSDP (commercial)
- our goal: solve convex optimization problems by reduction to cone programs

Cone representation

(*Nesterov, Nemirovsky*)

cone representation of (convex) function f:

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

$$\begin{array}{ll} \text{minimize} & c^T x + d^T y + e \\ \text{subject to} & A \left[\begin{array}{c} x \\ y \end{array} \right] = b, \quad \left[\begin{array}{c} x \\ y \end{array} \right] \in \mathcal{K} \end{array}$$

  - cone program in (x, y), we but minimize only over y

- $i.e.$, we define f by optimizing over *some* variables in a cone program

Examples

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

$$\begin{array}{ll} \text{minimize} & -t \\ \text{subject to} & \begin{bmatrix} x & t \\ t & y \end{bmatrix} \succeq 0 \end{array}$$

  with variable t

- $f(x, y) = x^T x/y$ is optimal value of SDP

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \begin{bmatrix} tI & x \\ x^T & y \end{bmatrix} \succeq 0 \end{array}$$

  with variable t

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

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T\lambda - k\nu \\ \text{subject to} & x + \nu\mathbf{1} = \lambda - \mu \\ & \lambda \succeq 0, \quad \mu \succeq 0 \end{array}$$

  with variables λ, μ, ν

- $f(p, q) = p\log(p/q)$ is optimal value of exponential cone program

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & (-t, q, p) \in \mathcal{E} \quad (\Leftrightarrow \quad \exp(-t/p) \leq q/p) \end{array}$$

  with variable t

SDP representations

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

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

some of these representations are not obvious . . .

Outline

- Convex optimization
- Constructive convex analysis
- Cone programming and representations
- Transforming to cone program
- Parser/solvers
- Code generation

Transforming to cone program: Example

- example: ℓ_1-regularized least-norm problem

$$\text{minimize} \quad \|Ax - b\|_2 + \lambda\|x\|_1$$

  with variable $x \in \mathbf{R}^n$ (convex, but not a cone program . . .)

- introduce new variables $t \in \mathbf{R}$, $z \in \mathbf{R}^m$, $x_+,\ x_- \in \mathbf{R}^n$:

$$\begin{array}{ll} \text{minimize} & t + \lambda(\mathbf{1}^T x_+ + \mathbf{1}^T x_-) \\ \text{subject to} & \|z\|_2 \leq t, \quad x_+ \succeq 0, \quad x_- \succeq 0 \\ & z = Ax - b, \quad x = x_+ - x_- \end{array}$$

  . . . a cone program with $x \in \mathbf{R}^n$, $(x_+, x_-) \in \mathbf{R}^{2n}_+$, $(z, t) \in \mathcal{L}^m$

- optimal x for cone program is optimal for original problem

Transforming to cone program: General case

- start with convex optimization problem $\mathcal{P}_0$
- find sequence of transformations that yields cone program $\mathcal{P}_K$

$$\mathcal{P}_0 \rightarrow \mathcal{P}_1 \rightarrow \cdots \rightarrow \mathcal{P}_K$$

- solve $\mathcal{P}_K$ efficiently
- transform solution of $\mathcal{P}_K$ back to solution of original problem $\mathcal{P}_0$

Problem transformations

(a.k.a. re-writing methods, reductions)

- there are many such problem transformations, some obvious, many not
- idea goes back at least to 1940s (Dantzig, for LP)
- standard optimization curriculum covers transformation 'tricks'
  (to be carried out by hand, $i.e.$, graduate student)

Convex calculus rules and problem transformations

- for most of the convex calculus rules, there is an associated problem transformation that 'undoes' the rule

- example: suppose $\max\{f_1(x), f_2(x)\}$ appears as term in objective or constraint function, with $f_1,\ f_2$ convex

- problem transformation:
  - replace $\max\{f_1(x), f_2(x)\}$ with a new variable t
  - add new (convex) constraints $f_1(x) \leq t$, $f_2(x) \leq t$

- yields equivalent problem
  (solution x of new problem is solution of original problem)

General composition rule and problem transformations

- suppose we encounter expression $h(f_1(x), \ldots, f_k(x))$, convex by general composition rule

- problem transformation:
  - replace expression with new variable t
  - add new variables $s_1, \ldots, s_k$ and (convex) constraints

$$\begin{array}{ll} f_i(x) \leq s_i & f_i \text{ convex} \\ f_i(x) \geq s_i & f_i \text{ concave} \\ f_i(x) = s_i & f_i \text{ affine} \end{array}$$

  - add (convex) constraint $h(s_1, \ldots, s_k) \leq t$

- yields equivalent problem

Cone representations and problem transformations

- suppose f has cone representation

$$\begin{array}{ll} \text{minimize} & c^T x + d^T y + e \\ \text{subject to} & A \begin{bmatrix} x \\ y \end{bmatrix} = b, \quad \begin{bmatrix} x \\ y \end{bmatrix} \in \mathcal{K} \end{array}$$

- when we encounter $f(x)$, we
  - add new variable y, and constraints above
  - replace $f(x)$ with $c^T x + d^T y + e$
- yields equivalent problem

Transforming from DCP to cone problem

- start with convex program in DCP form with cone-representable functions

- parse tree gives
  - proof of convexity
  - set of transformations that reduce original problem to an equivalent cone program

Outline

- Convex optimization
- Constructive convex analysis
- Cone programming and representations
- Transforming to cone program
- Parser/solvers
- Code generation

Parser/solver

- specify convex problem in natural (DCP) form
  - declare optimization variables
  - form convex objective and constraints using functions from a library, general composition rule
  - library functions defined by cone representations
- parser/solver
  - parses problem description (certifying convexity)
  - automatically transforms to equivalent cone problem
  - solves cone problem
  - transforms back to get solution of original problem

Parser/solver

- implemented using object-oriented methods and/or compiler-compilers
- huge gain in productivity
  - rapid prototyping
  - teaching
  - trying out research ideas
- transformation can introduce many new variables and constraints, but performance penalty small if sparsity is exploited

History

- general purpose optimization modeling systems AMPL, GAMS (1970s)
- systems for SDPs/LMIs (1990s): `sdpsol` (Wu, Boyd), `lmilab` (Gahinet, Nemirovsky), `lmitool` (El Ghaoui)
- `yalmip` (Löfberg 2000–)
- automated convexity checking (Crusius PhD thesis 2002)
- disciplined convex programming (Grant, Boyd, Ye 2004)
- `cvx` (Grant, Boyd 2005)
- `cvxopt` (Dahl, Vandenberghe 2005)
- `ggplab` (Mutapcic, Koh, et al 2006)

Example: CVX

- parser/solver written in Matlab (M. Grant)
- convex problem, with variable $x \in \mathbf{R}^n$; A, b, λ, F, g constants

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2 + \lambda\|x\|_1 \\ \text{subject to} & Fx \leq g \end{array}$$

- CVX specification:

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

when CVX processes this specification, it

- parses it (which verifies convexity of problem)
- generates equivalent cone problem (here, an SOCP)
- solves it using `SDPT3` or `SeDuMi`
- transforms solution back to original problem

the CVX code is easy to read, understand, modify

The same example, transformed by 'hand'

transform problem to SOCP, call `SeDuMi`, reconstruct solution:

```
% Set up big matrices.
[m,n] = size(A); [p,n] = size(F);
AA = [speye(n), -speye(n), speye(n), sparse(n,p+m+1); ...
      F, sparse(p,2*n), speye(p), sparse(p,m+1); ...
      A, sparse(m,2*n+p), speye(m), sparse(m,1)];
bb = [zeros(n,1); g; b];
cc = [zeros(n,1); gamma*ones(2*n,1); zeros(m+p,1); 1];
K.f = m; K.l = 2*n+p; K.q = m + 1;      % specify cone
xx = sedumi(AA, bb, cc, K);             % solve SOCP
x = x(1:n);                             % extract solution
```

Some functions in the CVX library

function	meaning	attributes
`norm(x, p)`	$\|x\|_p$, $p \geq 1$	cvx
`square(x)`	x^2	cvx
`square_pos(x)`	$(x_+)^2$	cvx, nondecr
`pos(x)`	x_+	cvx, nondecr
`sum_largest(x,k)`	$x_{[1]} + \cdots + x_{[k]}$	cvx, nondecr
`sqrt(x)`	$\sqrt{x}$, $x \geq 0$	ccv, nondecr
`inv_pos(x)`	$1/x$, $x > 0$	cvx, nonincr
`max(x)`	$\max\{x_1, \ldots, x_n\}$	cvx, nondecr
`quad_over_lin(x,y)`	x^2/y, $y > 0$	cvx, nonincr in y
`lambda_max(X)`	$\lambda_{\max}(X)$, $X = X^T$	cvx
`huber(x)`	$\begin{cases} x^2, & \|x\| \leq 1 \\ 2\|x\| - 1, & \|x\| > 1 \end{cases}$	cvx

Outline

- Convex optimization
- Constructive convex analysis
- Cone programming and representations
- Transforming to cone program
- Parser/solvers
- Code generation

Solving specific problems

in developing a custom solver for a specific application, we can

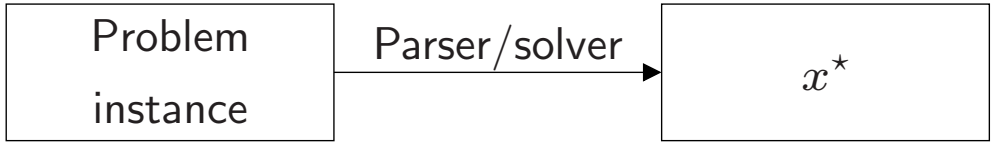
- optimize transformation to cone form
- exploit structure very efficiently
- determine ordering, memory allocation beforehand
- cut corners in algorithm, $e.g.$, terminate early
- use warm start

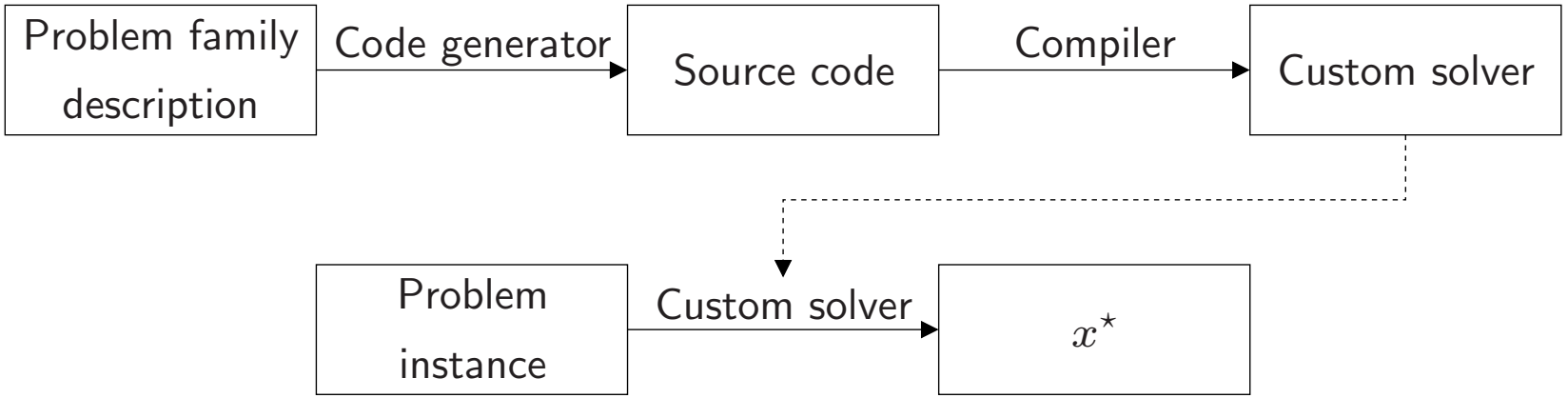
to get **very fast** solver

Convex optimization solver generation

- specify convex problem **family** via (an extension of) DCP
  - declare optimization variables, *parameters*
  - form convex objective and constraints from library of functions, using general composition rule
  - *parameter constraints* used in parsing
- code generator
  - analyzes problem structure (dimensions, sparsity, . . .)
  - chooses elimination ordering
  - generates solver code for specific problem family
- idea:
  - spend (perhaps much) time generating code
  - save (hopefully much) time solving problem instances

Parser/solver vs. code generation





Example: CVXMOD

- written in Python (J. Mattingley)
- QP family, with variable $x \in \mathbf{R}^n$, parameters P, q, g, h

$$\begin{array}{ll} \text{minimize} & x^T P x + q^T x \\ \text{subject to} & Gx \leq h, \quad Ax = b \end{array}$$

- CVXMOD specification:

```
A = matrix(...); b = matrix(...)
P = param('P', n, n, psd=True); q = param('q', n)
G = param('G', m, n); h = param('h', m)
x = optvar('x', n)
qpfam = problem(minimize(quadform(x, P) + tp(q)*x),
                [G*x <= h, A*x == b])
```

`cvxmod` code generation

- generate solver for problem family `qpfam` with

```
qpfam.codegen()
```

- output includes `qpfam/solver.c` and ancillary files

- solve instance with (C function call)

```
status = solver(params, vars, work);
```

`cvxmod` code generator

(preliminary implementation)

- handles problems transformable to QP
- primal-dual interior-point method
- direct LDL^T factorization of KKT matrix
- (slow) method to determine variable ordering (at code generation time)
- explicit factorization code generated

Sample solve times for CVXMOD generated code

problem family	vars	constrs	SDPT3 (ms)	`cvxmod` (ms)
`control1`	140	190	250	0.4
`control2`	360	1080	1400	2.0
`control3`	1110	3180	3400	13.2
`order_exec`	20	41	490	0.05
`net_utility`	50	150	130	0.23
`actuator`	50	106	300	0.17
`robust_kalman`	95	45	120	0.12

Conclusions

- DCP is a formalization of constructive convex analysis
  - simple method to certify problem as convex
  - useful as language for describing convex problems
- parser/solvers make rapid prototyping easy
- new code generation methods yield solvers that
  - are extremely fast
  - can be embedded in real-time applications

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)
- CVX (Grant, Boyd)
- CVXMOD (Mattingley, Boyd)

all available on-line, but CVXMOD code gen not yet ready for prime-time