Domain Specific Languages for Convex Optimization

Stephen Boyd

joint work with M. Grant, S. Diamond

Electrical Engineering Department, Stanford University

Springer Lectures, UC Berkeley, 4/10/15

Outline

Outline

Convex optimization

Constructive convex analysis

Cone representation

Canonicalization

Modeling frameworks

Conclusions

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 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

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

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

- $\mathcal{K}$ is convex cone
  - $x \in \mathcal{K}$ is a generalized nonnegativity constraint
- linear objective, equality constraints
- special cases:
  - $\mathcal{K} = \mathbf{R}^n_+$: linear program (LP)
  - $\mathcal{K} = \mathbf{S}^n_+$: semidefinite program (SDP)
- the modern canonical form

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

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

- $x^T x/y$ ($y > 0$), $x^T Y^{-1} x$ ($Y \succ 0$)
- $\log(e^{x_1} + \cdots + e^{x_n})$
- $-\log \Phi(x)$ (Φ is Gaussian CDF)
- $\log\det X^{-1}$ ($X \succ 0$)
- $\lambda_{\max}(X)$ ($X = X^T$)
- $f(x) = x_{[1]} + \cdots + x_{[k]}$ (sum of largest k entries)

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 $\longrightarrow f(Ax + b)$ 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

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 $x < 1$, $y < 1$

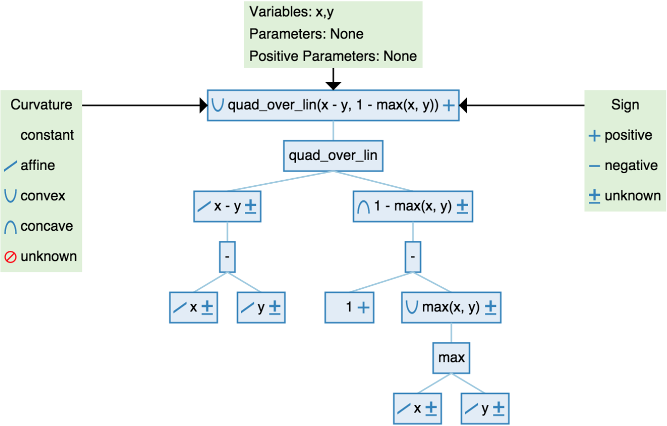
$$\frac{(x-y)^2}{1-\max(x,y)}$$

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

analyzed by `dcp.stanford.edu` (*Diamond 2014*)



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 {scalar convex expression} or
  - maximize {scalar concave expression}

- zero or more **constraints**, with form
  - {convex expression} <= {concave expression} or
  - {concave expression} >= {convex expression} or
  - {affine expression} == {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 $\sqrt[4]{\cdot}$, or $\exp \cdot$ and $(\cdot)_+$, since their attributes match

Outline

Cone representation

(*Nesterov, Nemirovsky*)

cone representation of (convex) function f:

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

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

  - 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

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

  with variable t

- $f(x) = x_{[1]} + \cdots + x_{[k]}$ 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

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

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

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

  - 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 constaints
- DCP ensures equivalence of resulting cone program

Outline

Example

- constrained least-squares problem with ℓ_1 regularization

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2^2 + \gamma\|x\|_1 \\ \text{subject to} & \|x\|_\infty \leq 1 \end{array}$$

  - variable $x \in \mathbf{R}^n$
  - constants/parameters A, b, $\gamma > 0$

CVX

- developed by M. Grant
- embedded in Matlab; targets multiple cone solvers
- CVX specification for example problem:

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

- here A, b, γ are **constants**

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

CVXPY

- developed by S. Diamond
- embedded in Python; targets multiple cone solvers

- CVXPY specification for example problem:

```
from cvxpy import *
x = Variable(n)
cost = sum_squares(A*x-b) + gamma*norm(x,1)
obj = Minimize(cost)
constr = [norm(x,"inf") <= 1]
prob = Problem(obj,constr)
opt_val = prob.solve()
solution = x.value
```

Parameters in CVXPY

- symbolic representations of constants
- can specify sign
- change value of constant without re-parsing problem

- computing a trade-off curve for example problem:

```
x_values = []
for val in numpy.logspace(-4, 2, num=100):
    gamma.value = val
    prob.solve()
    x_values.append(x.value)
```

Signed DCP in CVXPY

function	meaning	attributes
`norm(x, p)`	$\|x\|_p$, $p \geq 1$	cvx, nondecr for $x \geq 0$, nonincr for $x \leq 0$
`square(x)`	x^2	cvx, nondecr for $x \geq 0$, nonincr for $x \leq 0$
`huber(x)`	$\begin{cases} x^2, & \|x\| \leq 1 \\ 2\|x\| - 1, & \|x\| > 1 \end{cases}$	cvx, nondecr for $x \geq 0$, nonincr for $x \leq 0$

Outline

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
- modeling frameworks make rapid prototyping easy

References

- *Disciplined Convex Programming* (Grant, Boyd, Ye)
- *Graph Implementations for Nonsmooth Convex Programs* (Grant, Boyd)

- CVX (Grant, Boyd)
- CVXPY (Diamond, Boyd)