Disciplined convex programming and the `cvx` modeling framework

Michael C. Grant
Information Systems Laboratory
Stanford University

April 27, 2006

Agenda

Today's topics:

- **disciplined convex programming**, a methodology for practical convex optimization; and
- **the** `cvx` **modeling system**: a MATLAB-based software package which supports this methodology

Where you are now

What you have seen thus far:

- Fundamental definitions of convex sets and convex/concave functions
- An extensive *convexity calculus*: a collection of operations, combinations and transformations that are known to preserve convexity
- An introduction to convex optimization problems, or *convex programs*
- Several specific classes of convex programs: LPs, SDPs, GPs, SOCPs... hopefully you can identify a problem from one of these classes when you see one

You have not yet seen:

- How convex optimization problems are **solved**

(well, that and a few other things)

Solvers

A *solver* is an engine for solving a particular type of mathematical problem, such as a convex program

Solvers typically handle only a certain *class* of problems, such as LPs, or SDPs, or GPs

They also require that problems be expressed in a *standard form*

Most problems do not immediately present themselves in standard form, so they must be *transformed* into standard form

Solver example: MATLAB's `linprog`

A program for solving linear programs:

```
x = linprog( c, A, b, A_eq, B_eq, l, u )
```

Problems must be expressed in the following standard form:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b \\ & A_{\text{eq}} x = b_{\text{eq}} \\ & \ell \preceq x \preceq u \end{array}$$

As standard forms go, this one is quite flexible: in fact, the first step `linprog` often does is to convert this problem into a more restricted standard form!

Conversion to standard form: common tricks

Representing free variables as the difference of nonnegative variables:

$$x \text{ free} \quad \Longrightarrow \quad x_+ - x_-,\ x_+ \geq 0,\ x_- \geq 0$$

Eliminating inequality constraints using *slack variables*:

$$a^T x \leq b \quad \Longrightarrow \quad a^T x + s = b,\ s \geq 0$$

Splitting equality constraints into inequalities:

$$a^T x = b \quad \Longrightarrow \quad a^T x \leq b,\ a^T x \geq b$$

Solver example: SeDuMi

A program for solving LPs, SOCPs, SDPs, and related problems:

```
x = sedumi( A, b, c, K )
```

Solves problems of the form:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \mathcal{K} \triangleq \mathcal{K}_1 \times \mathcal{K}_2 \times \cdots \times \mathcal{K}_L \end{array}$$

where each set $\mathcal{K}_i \subseteq \mathbf{R}^{n_i}$, $i = 1, 2, \ldots, L$ is chosen from a very short list of cones (see next slide...)

The Matlab variable `K` gives the number, types, and dimensions of the cones $\mathcal{K}_i$

Cones supported by SeDuMi

- free variables: $\mathbf{R}^{n_i}$
- a nonnegative orthant: $\mathbf{R}_+^{n_i}$ (for linear inequalities)
- a real or complex second-order cone:

$$\mathbf{Q}^n \triangleq \{\, (x, y) \in \mathbf{R}^n \times \mathbf{R} \mid \|x\|_2 \leq y \,\}$$
$$\mathbf{Q}_c^n \triangleq \{\, (x, y) \in \mathbf{C}^n \times \mathbf{R} \mid \|x\|_2 \leq y \,\}$$

- a real or complex semidefinite cone:

$$\mathbf{S}_+^n \triangleq \left\{\, X \in \mathbf{R}^{n \times n} \mid X = X^T,\ X \succeq 0 \,\right\}$$
$$\mathbf{H}_+^n \triangleq \left\{\, X \in \mathbf{C}^{n \times n} \mid X = X^H,\ X \succeq 0 \,\right\}$$

The cones must be arranged in this order: $i.e.$, the free variables first, then the nonnegative orthants, then the second-order cones, then the semidefinite cones

Example: Norm approximation

Consider the problem

$$\text{minimize} \quad \|Ax - b\|$$

An optimal value x^* minimizes the resiuduals

$$r_k \triangleq a_k^T x - b_k \quad k = 1, 2, \ldots m$$

according to the measure defined by the norm $\|\cdot\|$

Obviously, the value of x^* depends significantly upon the choice of that norm...

...and so does the process of conversion to standard form

Euclidean (ℓ_2) norm

$$\text{minimize} \quad \|Ax - b\|_2 = \left(\sum_{k=1}^{m} (a_k^T x - b_k)^2\right)^{1/2}$$

No need to use `linprog` or SeDuMi here: this is a *least squares* problem, with an analytic solution

$$x^* = (A^T A)^{-1} A^T b$$

In MATLAB or Octave, a single command computes the solution:

```
>> x = A \ b;
```

Chebyshev (ℓ_∞) norm

$$\text{minimize} \quad f(Ax-b) \triangleq \|Ax-b\|_\infty = \max_{1\le k\le m} |a_k^T x - b|$$

This can be expressed as a linear program:

$$\begin{array}{ll} \text{minimize} & q \\ \text{subject to} & -q\mathbf{1} \le Ax-b \le +q\mathbf{1} \end{array} \quad \Longrightarrow \quad \begin{array}{ll} \text{minimize} & \begin{bmatrix} \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x \\ q \end{bmatrix} \\ \text{subject to} & \begin{bmatrix} A & -\mathbf{1} \\ -A & -\mathbf{1} \end{bmatrix} \begin{bmatrix} x \\ q \end{bmatrix} \le \begin{bmatrix} b \\ -b \end{bmatrix} \end{array}$$

The `linprog` call:

```
xq = linprog( [zeros(n,1);1], ...
              [A,-ones(m,1);-A,-ones(m,1)], [b;-b] );
x = xq(1:n);
```

Manhattan (ℓ_1) norm

$$\text{minimize} \quad f(Ax-b) \triangleq \|Ax-b\|_1 = \sum_{k=1}^{m} |a_k^T x - b|$$

LP formulation:

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T w \\ \text{subject to} & -w \leq Ax - b \leq w \end{array} \quad \Longrightarrow \quad \begin{array}{ll} \text{minimize} & \begin{bmatrix} \mathbf{0} & \mathbf{1} \end{bmatrix}^T \begin{bmatrix} x \\ w \end{bmatrix} \\ \text{subject to} & \begin{bmatrix} A & -I \\ -A & -I \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} \leq \begin{bmatrix} b \\ -b \end{bmatrix} \end{array}$$

The `linprog` call:

```
xw = linprog( [zeros(n,1);ones(m,1)], ...
              [A,-eye(m,1);-A,-eye(m,1)], [b;-b] );
x = xw(1:n);
```

Largest-L norm

Let $|w|_{[k]}$ represent the k-th largest element (in magnitude) of $w \in \mathbf{R}^m$:

$$|w|_{[1]} \geq |w|_{[2]} \geq \cdots \geq |w|_{[m]}$$

Then define the *largest-L norm* as

$$\|w\|_{[L]} \triangleq |w|_{[1]} + |w|_{[2]} + \cdots + |w|_{[L]} \triangleq \sum_{k=1}^{L} \sigma_k(\mathbf{diag}(w))$$

for any integer $1 \leq L \leq m$. Special cases include

$$\|w\|_{[1]} \equiv \|w\|_\infty, \quad \|w\|_{[m]} \triangleq \|w\|_1$$

but novel results are produced for all $1 < L < m$

Largest-L norm (continued)

This norm can be also represented in a linear program:

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T v + Lq \\ \text{subject to} & -v - \mathbf{1}q \leq Ax - b \leq +v + \mathbf{1}q \\ & v \geq 0 \end{array}$$

The `linprog` call:

```
xvq = linprog( [zeros(n,1);ones(m,1);L], ...
               [A,-eye(m),ones(m,1);-A,-eye(m),-ones(m,1)], ...
               [], [], [-Inf*ones(n,1);zeros(m,1);-Inf] );
x = xvq(1:n);
```

This is not much more complex than the ℓ_1 and ℓ_∞ cases—but of course, you have to know this trick (and few do)

Constrained Euclidean (ℓ_2) norm

Add some constraints:

$$\begin{array}{ll} \text{minimize} & \|Ax-b\|_2 \\ \text{subject to} & Cx = d \\ & \ell \leq x \leq u \end{array}$$

This is not a least-squares problem—but it is an SOCP, and SeDuMi can handle it, once it is converted to standard form:

$$\begin{array}{ll} \text{minimize} & z \\ \text{subject to} & Ax - b = y \\ & Cx = d \\ & x - s_\ell = \ell \\ & x + s_u = u \\ & s_\ell, s_u \geq 0 \\ & \|y\|_2 \leq z \end{array} \quad \Longrightarrow \quad \begin{array}{ll} \text{minimize} & \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 \end{bmatrix}^T \bar{x} \\ \text{subject to} & \begin{bmatrix} A & & & -I & \\ C & & & & \\ I & -I & & & \\ I & & I & & \end{bmatrix} \bar{x} = \begin{bmatrix} b \\ d \\ \ell \\ u \end{bmatrix} \\ & \bar{x} \in \mathbf{R}^n \times \mathbf{R}^n_+ \times \mathbf{R}^n_+ \times \mathbf{Q}^m \end{array}$$

$s_l, s_u \in \mathbf{R}^n_+$ are *slack variables*, which are used quite often to convert inequalities to equations

Constrained Euclidean (ℓ_2) norm

The SeDuMi call:

```
AA = [ A,       zeros(m,n), zeros(m,n), -eye(m),    0 ;
       C,       zeros(p,n), zeros(p,n), zeros(p,n), 0 ;
       eye(n), -eye(n),    zeros(n,n), zeros(n,n), 0 ;
       eye(n), zeros(n,n), eye(n),     zeros(n,n), 0 ];
bb = [ b ; d ; l ; u ];
cc = [ zeros( 3 * n + m, 1 ) ; 1 ];
K.f = n; K.l = 2 * n; K.q = m + 1;
xsyz = sedumi( AA, bb, cc, K );
x = xsyz( 1 : n );
```

Hopefully it is getting clear: this can be cumbersome

Modeling frameworks

A *modeling framework* simplifies the use of a numerical technology by shielding the user from the underlying mathematical details. Frameworks for LP, QP, NLP, and SDP are directly responsible for increased popularity of these problem classes

Most common modeling frameworks provide a language that allows problems to be specified in a natural syntax, and automatically call an underlying numerical solver

Examples:

- AMPL, GAMS, AIMMS, LINDO, $etc.$: LP, QP, NLP
- SDPSOL, LMITOOL, LMILAB: SDP
- YALMIP: LP, SOCP, SDP

`cvx` is designed to support convex optimization; or more specifically, *disciplined convex programming*

Disciplined convex programming

People don't simply write down optimization problems and hope that they are convex; instead, they draw from a "mental library" of functions and sets with known convexity properties, and combine them in ways that convex analysis guarantees will produce convex results...

...$i.e.$, using the calculus rules discussed in previous lectures

Disciplined convex programming formalizes this methodology

Two key elements:

- An expandable *atom library*: a collection of functions and sets, or *atoms*, with known properties of convexity, monotonicity, and range.

- A ruleset which governs how those atoms can be used and combined to form a valid problem

Compliant problems are called *disciplined convex programs*, or DCPs

The DCP ruleset

A subset of the calculus rules you have already learned:

- objective functions: convex for minimization, concave for maximization
- constraints:
  - equality and inequality constraints
  - each must be separately convex
- expressions:
  - addition, subtraction, scalar multiplication
  - composition with affine functions
  - nonlinear compositions (with appropriate monotonicity conditions). including elementwise maximums, $etc.$

Not included (for now): perspective transformations, conjugates, pointwise supremum, quasiconvexity

Is the ruleset too limited?

These rules constitute a set of *sufficient* conditions for convexity. For example,

```
sqrt( sum( square( x ) ) )
```

is convex, but does not obey the ruleset, because it is the composition of a concave nondecreasing function and a convex function

In this case, the workaround is simple: use `norm( x )` instead

In some cases, it may be necessary to add new functions (more later)

A practical hurdle? Perhaps, but it does mirror the mental process

Disciplined convex programming: benefits

- Verification of convexity is replaced with enforcement of the ruleset, which can be performed reliably and quickly
- Conversion to solvable form is fully automated
- Because the atom library is extensible, generality is not compromised
- New atoms can be created by experts and shared with novices

Demonstration