Disciplined convex programming
and the \texttt{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 \textit{convexity calculus}: a collection of operations, combinations and transformations that are known to preserve convexity
- An introduction to convex optimization problems, or \textit{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 \textbf{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.
Convex optimization solvers

- **LP solvers**
  - lots available (GLPK, Excel, Matlab’s `linprog`, . . .)

- **cone solvers**
  - typically handle (combinations of) LP, SOCP, SDP cones
  - several available (SDPT3, SeDuMi, CSDP, . . .)

- **general convex solvers**
  - some available (CVXOPT, MOSEK, . . .)

- plus lots of special purpose or application specific solvers

- could write your own

(we’ll study, and write, solvers later in the quarter)
Introduction

Optimization concerns the minimization or maximization of functions. The Optimization Toolbox consists of functions that perform minimization (or maximization) on general nonlinear functions. Functions for nonlinear equation solving and least-squares (data-fitting) problems are also provided.

The tables below show the functions available for minimization, equation solving, and solving least squares or data fitting problems.

Table 1-1: Minimization

<table>
<thead>
<tr>
<th>Type</th>
<th>Notation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar Minimization</td>
<td>( \min_a f(a) \text{ such that } a_1 &lt; a &lt; a_2 )</td>
<td>fminbnd</td>
</tr>
<tr>
<td>Unconstrained</td>
<td>( \min_x f(x) )</td>
<td>fminunc, fminsearch</td>
</tr>
<tr>
<td>Minimization</td>
<td>( \min_x f^T x ) \text{ such that } A \cdot x \leq b, \ Aeq \cdot x = beq, \ l \leq x \leq u )</td>
<td>linprog</td>
</tr>
<tr>
<td>Linear Programming</td>
<td>( \min_x \frac{1}{2} x^T H x + f^T x ) \text{ such that } A \cdot x \leq b, \ Aeq \cdot x = beq, \ l \leq x \leq u )</td>
<td>quadprog</td>
</tr>
<tr>
<td>Quadratic Programming</td>
<td>( \min_x f(x) \text{ such that } c(x) \leq 0, \ ceq(x) = 0 ) \text{ and } A \cdot x \leq b, \ Aeq \cdot x = beq, \ l \leq x \leq u )</td>
<td>fmincon</td>
</tr>
<tr>
<td>Constrained</td>
<td>( \min_x f(x) \text{ such that } c(x) \leq 0, \ ceq(x) = 0, \ A \cdot x \leq b, \ Aeq \cdot x = beq, \ l \leq x \leq u )</td>
<td>fgoalattain</td>
</tr>
<tr>
<td>Goal Attainment</td>
<td>( \min_{x, \gamma} \text{ such that } F(x) - w \gamma \leq \text{goal} ) \text{ and } c(x) \leq 0, \ ceq(x) = 0, \ A \cdot x \leq b, \ Aeq \cdot x = beq, \ l \leq x \leq u )</td>
<td></td>
</tr>
</tbody>
</table>
### Table 1-1: Minimization (Continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Notation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimax</td>
<td>( \min_{x} \max_{{ F_i(x) } } ) such that ( c(x) \leq 0, \ ceq(x) = 0, \ A \cdot x \leq b, \ Aeq \cdot x = beq, \ l \leq x \leq u )</td>
<td><code>fminimax</code></td>
</tr>
<tr>
<td>Semi-infinite Minimization</td>
<td>( \min_{x} f(x) ) such that ( K(x, w) \leq 0 ) for all ( w ), ( c(x) \leq 0, \ ceq(x) = 0, \ A \cdot x \leq b, \ Aeq \cdot x = beq, \ l \leq x \leq u )</td>
<td><code>fseminf</code></td>
</tr>
</tbody>
</table>

### Table 1-2: Equation Solving

<table>
<thead>
<tr>
<th>Type</th>
<th>Notation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Equations</td>
<td>( C \cdot x = d ), ( n ) equations, ( n ) variables</td>
<td>( \backslash ) (slash)</td>
</tr>
<tr>
<td>Nonlinear Equation of One Variable</td>
<td>( f(a) = 0 )</td>
<td><code>fzero</code></td>
</tr>
<tr>
<td>Nonlinear Equations</td>
<td>( F(x) = 0 ), ( n ) equations, ( n ) variables</td>
<td><code>fsolve</code></td>
</tr>
</tbody>
</table>

### Table 1-3: Least-Squares (Curve Fitting)

<table>
<thead>
<tr>
<th>Type</th>
<th>Notation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Least Squares</td>
<td>( \min_{x} | C \cdot x - d |_2^2 ), ( m ) equations, ( n ) variables</td>
<td>( \backslash ) (slash)</td>
</tr>
<tr>
<td>Nonnegative Linear Least Squares</td>
<td>( \min_{x} | C \cdot x - d |_2^2 ) such that ( x \geq 0 )</td>
<td><code>lsqnonneg</code></td>
</tr>
</tbody>
</table>
Overview

Most of these optimization routines require the definition of an M-file containing the function to be minimized. Alternatively, an inline object created from a MATLAB expression can be used. Maximization is achieved by supplying the routines with \(-f\), where \(f\) is the function being optimized.

Optimization options passed to the routines change optimization parameters. Default optimization parameters are used extensively but can be changed through an options structure.

Gradients are calculated using an adaptive finite-difference method unless they are supplied in a function. Parameters can be passed directly to functions, avoiding the need for global variables.

This User’s Guide separates “medium-scale” algorithms from “large-scale” algorithms. Medium-scale is not a standard term and is used here only to differentiate these algorithms from the large-scale algorithms, which are designed to handle large-scale problems efficiently.

Medium-Scale Algorithms

The Optimization Toolbox routines offer a choice of algorithms and line search strategies. The principal algorithms for unconstrained minimization are the Nelder-Mead simplex search method and the BFGS quasi-Newton method. For constrained minimization, minimax, goal attainment, and semi-infinite optimization, variations of Sequential Quadratic Programming are used.
Solver example: MATLAB’s linprog

A program for solving linear programs:

\[ x = \text{linprog}( c, A, b, A_{eq}, B_{eq}, l, u ) \]

Problems must be expressed in the following standard form:

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad Ax \preceq b \\
& \quad A_{eq} x = b_{eq} \\
& \quad l \preceq x \preceq u
\end{align*}
\]

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:

\[ x \text{ free} \implies x_+ - x_-, \quad x_+ \geq 0, \quad x_- \geq 0 \]

Eliminating inequality constraints using *slack variables*:

\[ a^T x \leq b \implies a^T x + s = b, \quad s \geq 0 \]

Splitting equality constraints into inequalities:

\[ a^T x = b \implies a^T x \leq b, \quad a^T x \geq b \]
Solver example: SeDuMi

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

\[ x = \text{sedumi}(A, b, c, K) \]

Solves problems of the form:

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad Ax = b \\
& \quad x \in \mathcal{K} \triangleq \mathcal{K}_1 \times \mathcal{K}_2 \times \cdots \times \mathcal{K}_L
\end{align*}
\]

where each set \( \mathcal{K}_i \subseteq \mathbb{R}^{n_i} \), \( i = 1, 2, \ldots, L \) 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: $\mathbb{R}^{n_i}$
- a nonnegative orthant: $\mathbb{R}_+^{n_i}$ (for linear inequalities)
- a real or complex second-order cone:
  \[
  \mathbb{Q}^n_c \triangleq \left\{ (x, y) \in \mathbb{R}^n \times \mathbb{R} \mid \|x\|_2 \leq y \right\}
  \]
  \[
  \mathbb{Q}_c^n \triangleq \left\{ (x, y) \in \mathbb{C}^n \times \mathbb{R} \mid \|x\|_2 \leq y \right\}
  \]
- a real or complex semidefinite cone:
  \[
  \mathbb{S}^n_+ \triangleq \left\{ X \in \mathbb{R}^{n \times n} \mid X = X^T, X \succeq 0 \right\}
  \]
  \[
  \mathbb{H}^n_+ \triangleq \left\{ X \in \mathbb{C}^{n \times n} \mid X = X^H, X \succeq 0 \right\}
  \]

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

\[ \text{minimize } \| Ax - b \| \]

An optimal value \( x^* \) minimizes the residuals

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

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 ($\ell_2$) 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:

\[
>> \ x = A \ \backslash \ b;
\]
Chebyshev ($\ell_\infty$) norm

\[
\begin{aligned}
\text{minimize} & \quad f(Ax - b) \triangleq \|Ax - b\|_\infty = \max_{1 \leq k \leq m} |a_k^T x - b| \\
\text{This can be expressed as a linear program:}
\end{aligned}
\]

\[
\begin{aligned}
\text{minimize} & \quad \begin{bmatrix} q \end{bmatrix} \\
\text{subject to} & \quad -q1 \leq Ax - b \leq +q1 \\
\end{aligned} 
\Rightarrow
\begin{aligned}
\text{minimize} & \quad \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ q \end{bmatrix} \\
\text{subject to} & \quad \begin{bmatrix} A & -1 \\ -A & -1 \end{bmatrix} \begin{bmatrix} x \\ q \end{bmatrix} \leq \begin{bmatrix} b \\ -b \end{bmatrix} 
\end{aligned}
\]

The linprog call:

\[
\begin{align*}
xq &= \text{linprog}( [\text{zeros(n,1);1}], \ldots \\
&\quad [A,-\text{ones(m,1);-A,-\text{ones(m,1)}], [b;-b] ); \\
x &= xq(1:n);
\end{align*}
\]
Manhattan ($\ell_1$) norm

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

LP formulation:

$$\minimize \quad 1^T w$$
$$\text{subject to} \quad -w \leq Ax - b \leq w$$

$$\implies \minimize \quad [0 \ 1]^T \begin{bmatrix} x \\ w \end{bmatrix}$$
$$\text{subject to} \quad \begin{bmatrix} A & -I \\ -A & -I \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} \preceq \begin{bmatrix} b \\ -b \end{bmatrix}$$

The `linprog` call:

```matlab
xw = linprog([zeros(n,1);ones(m,1)], ...
[A,-eye(m,1);-A,-eye(m,1)], [b;-b] );
x = xw(1:n);
```
Constrained Euclidean ($\ell_2$) norm

Add some constraints:

\[
\begin{align*}
\text{minimize} & \quad \|Ax - b\|_2 \\
\text{subject to} & \quad Cx = d \\
& \quad \ell \leq x \leq u
\end{align*}
\]

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

\[
\begin{align*}
\text{minimize} & \quad z \\
\text{subject to} & \quad Ax - b = y \\
& \quad Cx = d \\
& \quad x - s_\ell = \ell \\
& \quad x + s_u = u \\
& \quad s_\ell, s_u \geq 0 \\
& \quad \|y\|_2 \leq z \\
\end{align*}
\]

\[
\begin{align*}
\text{minimize} & \quad [0 \hspace{0.5cm} 0 \hspace{0.5cm} 0 \hspace{0.5cm} 0 \hspace{0.5cm} 1]^T \bar{x} \\
\text{subject to} & \quad \begin{bmatrix}
A & -I \\
C & -I \\
I & -I \\
I & I \\
\end{bmatrix} \begin{bmatrix}
\bar{x} \\
\end{bmatrix} = \begin{bmatrix}
b \\
d \\
\ell \\
u \\
\end{bmatrix} \\
\bar{x} & \in \mathbb{R}^n \times \mathbb{R}_+^n \times \mathbb{R}_+^n \times \mathbb{Q}^m
\end{align*}
\]

$s_\ell, s_u \in \mathbb{R}_+^n$ are slack variables, which are used quite often to convert inequalities to equations.
Constrained Euclidean (\(l^2\)) norm

The SeDuMi call:

\[
\begin{align*}
AA &= \begin{bmatrix} A, & \text{zeros}(m,n), & \text{zeros}(m,n), & -\text{eye}(m), & 0; \\
C, & \text{zeros}(p,n), & \text{zeros}(p,n), & \text{zeros}(p,n), & 0; \\
\text{eye}(n), & -\text{eye}(n), & \text{zeros}(n,n), & \text{zeros}(n,n), & 0; \\
\text{eye}(n), & \text{zeros}(n,n), & \text{eye}(n), & \text{zeros}(n,n), & 0 \\
\end{bmatrix}; \\
bb &= \begin{bmatrix} b; \\
d; \\
l; \\
u \end{bmatrix}; \\
cc &= \begin{bmatrix} \text{zeros}(3*n+m,1); \\
1 \end{bmatrix}; \\
K.f &= n; \\
K.l &= 2*n; \\
K.q &= m + 1; \\
xsyz &= \text{sedumi}(AA, bb, cc, K); \\
x &= xsyz(1:n);
\end{align*}
\]

Hopefully it is getting clear: this can be cumbersome
Transforming problems to standard form

- you’ve seen lots of tricks for transforming a problem into an equivalent one that has a standard form (e.g., LP, SDP)
- these tricks greatly extend the applicability of standard solvers
- writing code to carry out this transformation is often painful
- **modeling systems** can partly automate this step
Modeling systems

**a typical modeling system**

- automates most of the transformation to standard form; supports
  - declaring optimization variables
  - describing the objective function
  - describing the constraints
  - choosing (and configuring) the solver

- when given a problem instance, calls the solver

- interprets and returns the solver’s status (optimal, infeasible, . . . )

- (when solved) transforms the solution back to original form
Some current modeling systems

- AMPL & GAMS (proprietary)
  - developed in the 1980s, still widely used in traditional OR
  - no support for convex optimization

- YALMIP (‘Yet Another LMI Parser’)
  - first matlab-based object-oriented modeling system with special support for convex optimization
  - can use many different solvers; can handle some nonconvex problems

- CVXMOD/CVXOPT (in alpha)
  - python based, completely GPLed
  - cone and custom solvers

- CVX
  - matlab based, GPL, uses SDPT3/SeDuMi
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,

$$\sqrt{\sum(\text{square}(x))}$$

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
CVX

- uses DCP
- runs in Matlab, between the `cvx_begin` and `cvx_end` commands
- relies on SDPT3 or SeDuMi (LP/SOCP/SDP) solvers
- refer to user guide, online help for more info
- the CVX example library has more than a hundred examples
Using CVX

Typical CVX script:

cvx_begin

variable [name][size1],[size2]) (optional)[type]

(optional)minimize(convex scalar function of the variables)

constraints

cvx_end
Example: 

\[ \text{minimize} \quad \|Ax - b\|_2, \]

where \( x \in \mathbb{R}^n, \ b \in \mathbb{R}^m. \)

CVX source code:

```cvx
cvx_begin
    variable x(n)
    minimize(norm(A*x-b))
cvx_end
```

*Note:* This is for demonstration purposes only. It is perhaps easier to solve this least-squares problem using \( x=A\backslash b. \)
**Example:**

\[
\begin{align*}
& \text{minimize} & & 1^T x \\
& \text{subject to} & & Ax = b \\
& & & x \geq 0,
\end{align*}
\]

where \( x \in \mathbb{R}^n, b \in \mathbb{R}^m \).

CVX source code:

```cvx
cvx_begin
  variable x(n)
  minimize(ones(1,n)*x)
  subject to
    A*x == b
    x >= 0
cvx_end
```
Example:

$$\begin{align*}
\text{minimize} & \quad \max_{k=1,\ldots,m} \max(a_k^T x, 1/a_k^T x) \\
\text{subject to} & \quad 0 \leq x \leq 1,
\end{align*}$$

where $x \in \mathbb{R}^n$.

CVX source code:

```cvx
cvx_begin
    variable x(n)
    minimize(max(max([A*x inv_pos(A*x)]'))))
    subject to
        x >= 0
        x <= 1
cvx_end```

```
What CVX does

after cvx_end, CVX

- transforms problem into an LP
- calls solver SDPT3
- overwrites (object) x with (numeric) optimal value
- assigns problem optimal value to cvx_optval
- assigns problem status (which here is Solved) to cvx_status

(had problem been infeasible, cvx_status would be Infeasible and x would be NaN)
Variables and affine expressions

- declare variables with variable name[(dims)] [attributes]
  - variable x(3);
  - variable C(4,3);
  - variable S(3,3) symmetric;
  - variable D(3,3) diagonal;
  - variables y z;

- form affine expressions
  - A = randn(4, 3);
  - variables x(3) y(4);
  - 3*x + 4
  - A*x - y
  - x(2:3)
  - sum(x)
# Some functions

<table>
<thead>
<tr>
<th>function</th>
<th>meaning</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm(x, p)</td>
<td>$|x|_p$</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>$\lambda_{\max}(X)$ ($X = X^T$)</td>
<td>cvx</td>
</tr>
<tr>
<td>huber(x)</td>
<td>$\begin{cases} x^2, &amp;</td>
<td>x</td>
</tr>
</tbody>
</table>
Composition rules

• can combine atoms using valid composition rules, \textit{e.g.}:
  – a convex function of an affine function is convex
  – the negative of a convex function is concave
  – a convex, nondecreasing function of a convex function is convex
  – a concave, nondecreasing function of a concave function is concave

• for convex $h$, $h(g_1, \ldots, g_k)$ is recognized as convex if, for each $i$,
  – $g_i$ is affine, or
  – $g_i$ is convex and $h$ is nondecreasing in its $i$th arg, or
  – $g_i$ is concave and $h$ is nonincreasing in its $i$th arg

• for concave $h$, $h(g_1, \ldots, g_k)$ is recognized as concave if, for each $i$,
  – $g_i$ is affine, or
  – $g_i$ is convex and $h$ is nonincreasing in $i$th arg, or
  – $g_i$ is concave and $h$ is nondecreasing in $i$th arg
Valid (recognized) examples

u, v, x, y are scalar variables; X is a symmetric $3 \times 3$ variable

- **convex:**
  - $\text{norm}(A \cdot x - y) + 0.1 \cdot \text{norm}(x, 1)$
  - $\quad \text{quad}\_\text{over}\_\text{lin}(u - v, 1 - \text{square}(v))$
  - $\quad \text{lambda}\_\text{max}(2 \cdot X - 4 \cdot \text{eye}(3))$
  - $\quad \text{norm}(2 \cdot X - 3, 'fro')$

- **concave:**
  - $\text{min}(1 + 2 \cdot u, 1 - \text{max}(2, v))$
  - $\quad \text{sqrt}(v) - 4.55 \cdot \text{inv}\_\text{pos}(u - v)$
Rejected examples

u, v, x, y are scalar variables

• neither convex nor concave:
  – \text{square}(x) - \text{square}(y)
  – \text{norm}(A*x - y) - 0.1*\text{norm}(x, 1)

• rejected due to limited DCP ruleset:
  – \sqrt{\text{sum}(\text{square}(x)))} (is convex; could use \text{norm}(x))
  – \text{square}(1 + x^2) (is convex; could use \text{square\_pos}(1 + x^2), or
    \ 1 + 2*\text{pow\_pos}(x, 2) + \text{pow\_pos}(x, 4))
Sets

• some constraints are more naturally expressed with convex sets

• sets in CVX work by creating unnamed variables constrained to the set

• examples:
  - semidefinite(n)
  - nonnegative(n)
  - simplex(n)
  - lorentz(n)

• semidefinite(n), say, returns an unnamed (symmetric matrix) variable that is constrained to be positive semidefinite
Using the semidefinite cone

variables: \(X\) (symmetric matrix), \(z\) (vector), \(t\) (scalar)
constants: \(A\) and \(B\) (matrices)

- \(X == \) semidefinite(n)
  - means \(X \in S_n^+\) (or \(X \succeq 0\))

- \(A*X*A' - X == B*semidefinite(n)*B'\)
  - means \(\exists Z \succeq 0\) so that \(AXA^T - X = BZB^T\)

- \([X \ z; \ z^T \ t] == \) semidefinite(n+1)
  - means
    \[
    \begin{bmatrix}
    X & z \\
    z^T & t
    \end{bmatrix} \succeq 0
    \]
Objectives and constraints

- **objective** can be
  - minimize(convex expression)
  - maximize(concave expression)
  - omitted (feasibility problem)

- **constraints** can be
  - convex expression <= concave expression
  - concave expression >= convex expression
  - affine expression == affine expression
  - omitted (unconstrained problem)
More involved example

A = randn(5);
A = A’*A;
cvx_begin
    variable X(5, 5) symmetric;
    variable y;
    minimize(norm(X) - 10*sqrt(y))
    subject to
        X - A == semidefinite(5);
        X(2,5) == 2*y;
        X(3,1) >= 0.8;
        y <= 4;
    cvx_end
Defining new functions

- can make a new function using existing atoms

- **example:** the convex deadzone function

\[
f(x) = \max\{|x| - 1, 0\} = \begin{cases} 
0, & |x| \leq 1 \\
x - 1, & x > 1 \\
1 - x, & x < -1 
\end{cases}
\]

- create a file `deadzone.m` with the code

  ```matlab
  function y = deadzone(x)
  y = max(abs(x) - 1, 0)
  ```

- `deadzone` makes sense both within and outside of CVX
Implementing atoms

Obviously, the usefulness of disciplined convex programming depends upon the size of the atom library available to us... or at least, on the ability to expand that library as needed.

The manner in which atoms are implemented depends heavily on the capabilities of the underlying solver.

Currently, cvx uses (SeDuMi), whose standard form permits four constraint types:

- Equality constraints
- Inequality constraints
- Second-order cone constraints
- Linear matrix inequalities

How do you implement a variety of functions under these limited conditions?
Graph implementations

A fundamental principle of convex analysis: the close relationship between convex functions and convex sets via the epigraph

The epigraph of a function $f : \mathbb{R}^n \to \mathbb{R}$ is

$$\text{epi } f \triangleq \{ (x, y) \in \mathbb{R}^n \times \mathbb{R} \mid f(x) \leq y \}$$

A function is convex if and only if its epigraph is a convex set.

For concave functions, the corresponding construction is the hypograph

$$\text{hypo } g \triangleq \{ (x, y) \in \mathbb{R}^n \times \mathbb{R} \mid g(x) \geq y \}$$

A function is concave if and only if its hypograph is a convex set.
The absolute value function (blue line) and its epigraph (grey region):

\[ f(x) \triangleq |x| \]
\[ \text{epi } f \triangleq \{ (x, y) \mid |x| \leq y \} \]
\[ = \{ (x, y) \mid x \leq y, -x \leq y \} \]

Key observation: for the purposes of convex optimization, the epigraph/hypograph can be used to provide a complete description of a function.

Graph implementations are, in effect, rules for “rewriting” a function in terms of other atoms by describing its epigraph or hypograph.
Example: the two-element maximum

\[ f(x, y) \triangleq \max\{x, y\} = \inf \{ z \mid x \leq z, y \leq z \} \]

function cvx_optval = max( x, y )
cvx_begin
    variable z
    minimize( z )
    subject to
        x <= z;
        y <= z;
cvx_end
Example: $\ell_1$ norm

\[
f(x) \triangleq \|x\|_1 = \inf \left\{ 1^T v \mid -v \leq x \leq v \right\}
\]

function cvx_optval = norm_1( x )
n = length( x );
cvx_begin
    variable v(n)
    minimize( sum(v) )
    subject to
        +x <= v;
        -x <= v;
    cvx_end

This implementation effectively encapsulates the very transformation employed in the norm approximation example
Example: $\ell_2$ norm

\[ f(x) \triangleq \|x\|_2 \implies \text{epi } f = \mathcal{Q}^n \]

Obviously, the second-order (Lorentz) cone can represent the 2-norm:

```matlab
function cvx_optval = norm_2( x )
n = length( x );
cvx_begin
    variable y
    { x, y } == lorentz( n );
cvx_end
```

(In the actual implementation, the complex case is handled too)
Example: Simple squaring

SeDuMi does not even support the basic convex function $f(x) = x^2$—or does it?

$$x^2 \leq y \implies \begin{bmatrix} y & x \\ x & 1 \end{bmatrix} \succeq 0$$

So $f(x)$ can be represented using a semidefinite program:

```matlab
function cvx_optval = square( x )
    cvx_begin sdp
        variable y
        minimize( y )
        [ y, x ; x, 1 ] >= 0
    cvx_end
```

Of course, if SeDuMi were replaced with a different solver, this implementation might not be necessary
Example: Square root

Graph implementations can freely refer to other graph implementations

For example, the square root function can be implemented using `square`:

\[ f(x) \triangleq \sqrt{x} \geq y \quad \implies \quad y^2 \leq x \]

```matlab
function cvx_optval = sqrt( x )
    cvx_begin
        variable y
        maximize( y )
        subject to
            square( y ) <= x
    cvx_end
```
Example: eigenvalue functions

If $X$ is symmetric:

$$\lambda_{\min}(X) \geq y \iff X - yI \succeq 0$$
$$\lambda_{\max}(X) \leq y \iff yI - X \succeq 0$$

function cvx_optval = lambda_min( X )
n = size(X,1);
cvx_begin
    variable y
    minimize( y )
    X - y * eye( n ) == semidefinite(n);
cvx_end
Example: matrix fractional function

\[ f : \mathbb{R}^n \times \mathbb{S}^n \rightarrow \mathbb{R}, \quad f(x,Y) \triangleq \begin{cases} 
 x^T Y^{-1} x & Y \succ 0 \\
 +\infty & \text{otherwise}
\end{cases} \]

Convex in both \( x \) and \( Y \), and can be represented using an LMI:

\[
 f(x,Y) \leq z \iff \begin{bmatrix} Y & x \\ x^T & z \end{bmatrix} \succeq 0
\]

```matlab
function cvx_optval = matrix_frac( x, Y )
n = length(x);
cvx_begin
    variable z
    minimize( z )
    subject to
        [(Y+Y')/2 x; x' z] == semidefinite(n+1);
cvx_end
```

Disciplined convex programming and the cvx modeling framework
Incomplete specifications

cvx actually supports a more general concept we call *incomplete specifications*, which is

\[ f(x) = \inf_y \{ g(x, y) \mid (x, y) \in S \} \quad \text{or} \quad f(x) = \sup_y \{ g(x, y) \mid (x, y) \in S \} \]

\((f, g \text{ convex})\) \quad \text{or} \quad \text{((f, g concave))}

where \(S\) is a convex set

Graph implementations are just special cases of this more general framework
Example: Huber penalty

\[ f(x) \triangleq \begin{cases} 
  x^2 & |x| \leq 1 \\
  2|x| - 1 & |x| > 1 
\end{cases} \]

Useful for robust regression—to suppress outliers
**Example: Huber penalty**

\[
f(x) \triangleq \begin{cases} 
  x^2 & |x| \leq 1 \\
  2|x| - 1 & |x| \geq 1 
\end{cases}
\]

\[
= \inf \left\{ w^2 + 2v \mid |x| \leq w + v, \ w \leq 1, \ v \geq 0 \right\}
\]

```matlab
function cvx_optval = huber( x )
cvx_begin
    variables v w
    minimize( square( w ) + 2 * v )
    subject to
        abs( x ) <= w + v
        w <= 1
        v >= 0
    cvx_end
```
Example: distance from an ellipsoid

Let $f_E(x)$ be the distance from a point $x$ to an ellipsoid $E$, where $E$ is described by a pair $(P, y_c) \in \mathbb{R}^{n \times n} \times \mathbb{R}^n$:

\[
f(x, y) \triangleq \inf \{ \|x - y\| \mid y \in E \}
\]

\[
E \triangleq \{ y \in \mathbb{R}^n \mid \|P(y - y_c)\|_2 \leq 1 \}
\]

Then $g(x, y) \triangleq \|x - y\|_2$, $S \triangleq \mathbb{R}^n \times E$:

```matlab
function cvx_optval = dist_ellipse( x, P, yc )
n = length( x );
cvx_begin
    variable y(n)
    minimize( norm( x - y, 2 ) )
    subject to
        norm( P * ( y - yc ), 2 ) <= 1;
cvx_end
```

Disciplined convex programming and the cvx modeling framework 35

Computational Methods in Optimization - CVX - 55 of 56
CVX hints/warnings

- watch out for = (assignment) versus == (equality constraint)

- X >= 0, with matrix X, is an elementwise inequality

- X >= semidefinite(n) means: X is elementwise larger than some positive semidefinite matrix (which is likely not what you want)

- writing subject to is unnecessary (but can look nicer)

- make sure you include brackets around objective functions
  - yes: minimize(c'*x)
  - no: minimize c'*x

- double inequalities like 0 <= x <= 1 don’t work; use 0 <= x; x <= 1 instead

- log, exp, entropy-type functions not yet implemented in CVX