Convex Optimization with Abstract Linear Operators

Stephen Boyd and Steven Diamond

EE & CS Departments

Stanford University

ICASSP, Shanghai, March 22 2016

Outline

Convex Optimization

Examples

Matrix-Free Methods

Summary

Outline

Convex Optimization

Examples

Matrix-Free Methods

Summary

Convex optimization problem — Classical form

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

- variable $x \in \mathbf{R}^n$
- equality constraints are linear
- $f_0, \ldots, f_m$ are **convex**: for $\theta \in [0, 1]$,

  $$f_i(\theta x + (1-\theta) y) \leq \theta f_i(x) + (1-\theta) f_i(y)$$

  i.e., f_i have nonnegative (upward) curvature

Convex optimization — Cone form

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

- variable $x \in \mathbf{R}^n$
- $K \subset \mathbf{R}^n$ is a proper cone
    - K nonnegative orthant $\longrightarrow$ LP
    - K Lorentz cone $\longrightarrow$ SOCP
    - K positive semidefinite matrices $\longrightarrow$ SDP
- the ‘modern’ canonical form

Why

- beautiful, nearly complete theory
    - duality, optimality conditions, . . .

Why

- beautiful, nearly complete theory
  - duality, optimality conditions, . . .
- effective algorithms, methods (in theory and practice)
  - get **global solution** (and optimality certificate)
  - polynomial complexity

Why

- beautiful, nearly complete theory
  - duality, optimality conditions, . . .
- effective algorithms, methods (in theory and practice)
  - get **global solution** (and optimality certificate)
  - polynomial complexity
- conceptual unification of many methods

Why

- beautiful, nearly complete theory
  - duality, optimality conditions, . . .
- effective algorithms, methods (in theory and practice)
  - get **global solution** (and optimality certificate)
  - polynomial complexity
- conceptual unification of many methods
- **lots of applications** (many more than previously thought)

Application areas

- machine learning, statistics
- finance
- supply chain, revenue management, advertising
- control
- signal and image processing, vision
- networking
- circuit design
- mechanical structure design
- combinatorial optimization
- quantum mechanics

Medium-scale solvers

- 1000s–10000s variables, constraints
- reliably solved by interior-point methods on single machine (especially for problems in standard cone form)
- exploit problem sparsity

Medium-scale solvers

- 1000s–10000s variables, constraints
- reliably solved by interior-point methods on single machine (especially for problems in standard cone form)
- exploit problem sparsity

- *no algorithm tuning/babysitting needed*
- not quite a technology, but getting there
- used in control, finance, engineering design, ...

Large-scale solvers

- 100k – 1B variables, constraints
- solved using custom (often problem specific) methods
  - limited memory BFGS
  - stochastic subgradient
  - block coordinate descent
  - operator splitting methods
- (when possible) exploit fast transforms (FFT, . . .)

- require custom implementation, tuning for each problem
- used in machine learning, image processing, . . .

Modeling languages

- (new) high level language support for convex optimization
  - describe problem in high level language
  - description automatically transformed to a standard form
  - solved by standard solver, transformed back to original form

Modeling languages

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\boxed{\begin{array}{l}\texttt{u} = \dots\\ \texttt{v} = \dots\\ \texttt{problem} = \dots\end{array}} \arrow[r, "\text{canonicalize}"] & \boxed{\begin{array}{ll}\text{min.} & c^Tx\\ \text{s.t.} & x \in K\\ & Ax = b\end{array}} \arrow[d, "\text{solve}"] \\
\boxed{\begin{array}{l}\texttt{u} = (0.59, \dots\\ \texttt{v} = (1.9, \dots\end{array}} & \boxed{\begin{array}{c}x = (1.58, \dots\\ \vdots\end{array}} \arrow[l, "\text{unpack}"']
\end{tikzcd}$$

Implementations

convex optimization modeling language implementations

- YALMIP, CVX (Matlab)
- CVXPY (Python)
- Convex.jl (Julia)

widely used for applications with medium scale problems

CVX

(*Grant & Boyd, 2005*)

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

- `A`, `b`, `gamma` are constants (`gamma` nonnegative)
- after `cvx_end`
  - problem is converted to standard form and solved
  - variable `x` is over-written with (numerical) solution

CVXPY

(*Diamond & Boyd, 2013*)

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

- `A`, `b`, `gamma` are constants (`gamma` nonnegative)
- `solve` method converts problem to standard form, solves, assigns `value` attributes

Modeling languages

- enable rapid prototyping (for small and medium problems)
- ideal for teaching (can do a lot with short scripts)
- shifts focus from *how to solve* to *what to solve*

- slower than custom methods, but often not much

Modeling languages

- enable rapid prototyping (for small and medium problems)
- ideal for teaching (can do a lot with short scripts)
- shifts focus from *how to solve* to *what to solve*

- slower than custom methods, but often not much

- this talk:
  how to extend CVXPY to large problems, fast operators

Outline

Convex Optimization

Examples

Matrix-Free Methods

Summary

Image in-painting

- guess pixel values in obscured/corrupted parts of image
- *total variation in-painting*:
  choose pixel values $x_{ij} \in \mathbf{R}^3$ to minimize *total variation*

$$\mathrm{TV}(x) = \sum_{ij} \left\| \begin{bmatrix} x_{i+1,j} - x_{ij} \\ x_{i,j+1} - x_{ij} \end{bmatrix} \right\|_2$$

- a convex problem [Rudin et al. 92, Chan and Shen 01]

CVXPY code

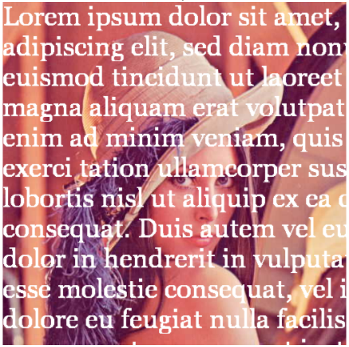
```
from cvxpy import *
R, G, B = Variable(n, n), Variable(n, n), Variable(n, n)
X = hstack(vec(R), vec(G), vec(B))
prob = Problem(Minimize(tv(R,G,B)),
               [X[known] == RGB[known],
                0 <= X, X <= 255])
prob.solve()
```

Example

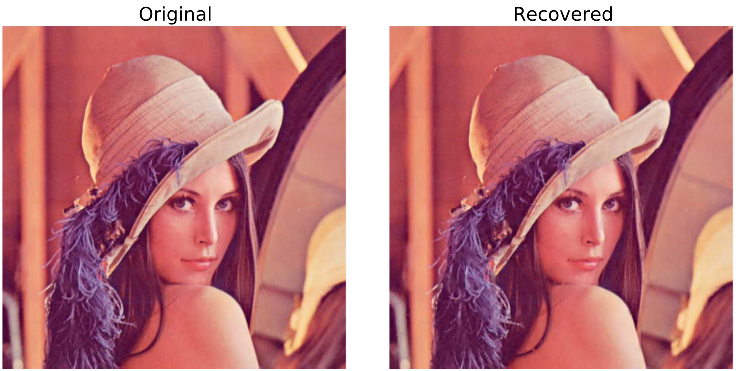
512 $\times$ 512 color image ($n \approx 800000$ variables)

Original

Corrupted

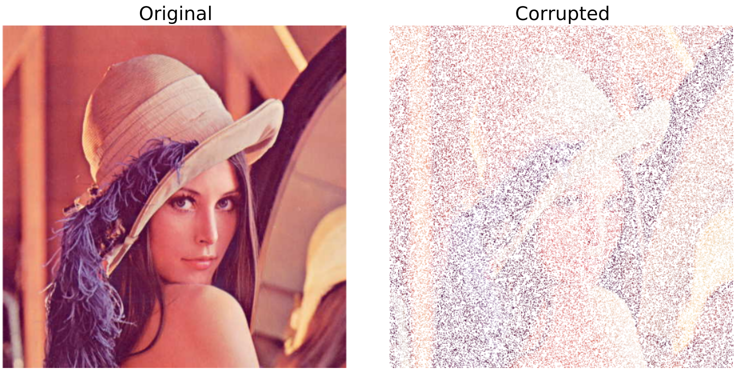

Lorem ipsum dolor sit amet,
adipiscing elit, sed diam non
euismod tincidunt ut laoreet
magna aliquam erat volutpat
enim ad minim veniam, quis
exerci tation ullamcorper sus
lobortis nisl ut aliquip ex ea
consequat. Duis autem vel eu
dolor in hendrerit in vulputa
esse molestie consequat, vel i
dolore eu feugiat nulla facilis

Example



Example

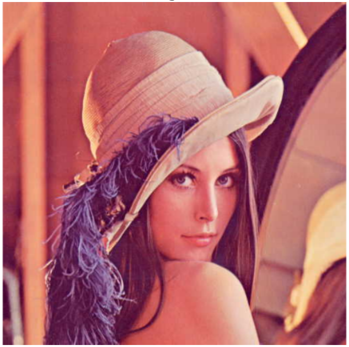
80% of pixels removed



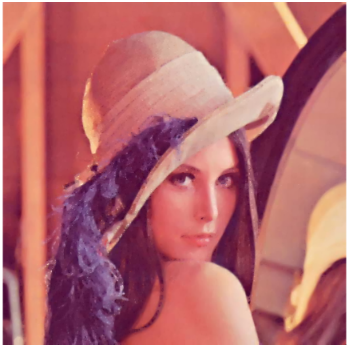
Example

80% of pixels removed

Original



Recovered



Colorization

- given B&W (scalar) pixel values, and a few colored pixels
- choose color pixel values $x_{ij} \in \mathbf{R}^3$ to minimize $\mathrm{TV}(x)$ subject to given B&W values
- a convex problem [Blomgren and Chan 98]

CVXPY code

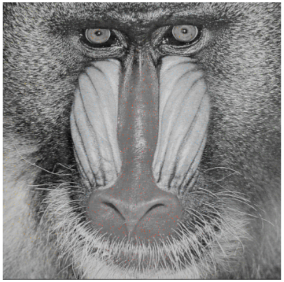
```
from cvxpy import *
R, G, B = Variable(n, n), Variable(n, n), Variable(n, n)
X = hstack(vec(R), vec(G), vec(B))
prob = Problem(Minimize(tv(R,G,B)),
              [0.299*R + 0.587*G + 0.114*B == BW,
               X[known] == RGB[known],
               0 <= X, X <= 255])
prob.solve()
```

Example

512 $\times$ 512 B&W image, with some color pixels given

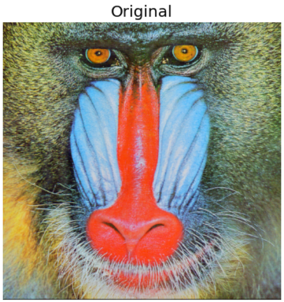
Original

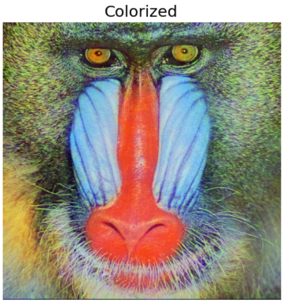
Black and White



Example

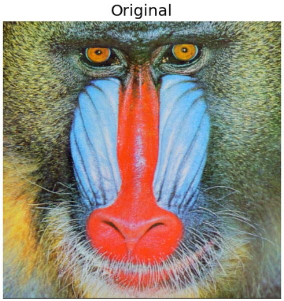
2% color pixels given

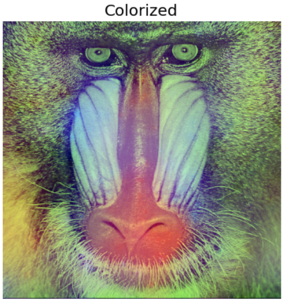




Example

0.1% color pixels given





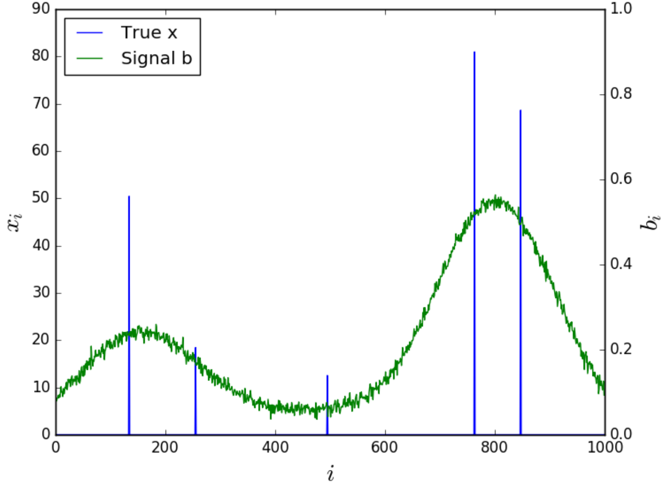
Nonnegative deconvolution

$$\begin{array}{ll} \text{minimize} & \|c * x - b\|_2 \\ \text{subject to} & x \geq 0 \end{array}$$

variable $x \in \mathbf{R}^n$; data $c \in \mathbf{R}^n$, $b \in \mathbf{R}^{2n-1}$

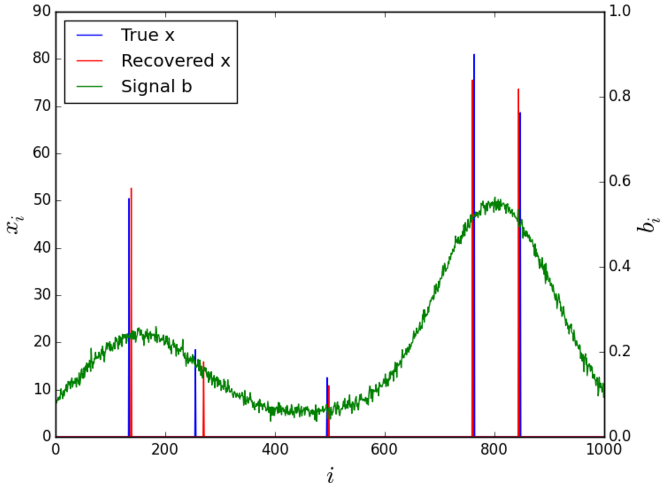
```
from cvxpy import *
x = Variable(n)
cost = norm(conv(c, x) - b)
prob = Problem(Minimize(cost),
               [x >= 0])
prob.solve()
```

Example



True x
Signal b
x_i
b_i
i
0
200
400
600
800
1000
90
80
70
60
50
40
30
20
10
0
1.0
0.8
0.6
0.4
0.2
0.0

Example


True x
Recovered x
Signal b
x_i
b_i
i
0
200
400
600
800
1000
0
10
20
30
40
50
60
70
80
90
0.0
0.2
0.4
0.6
0.8
1.0

Outline

Convex Optimization

Examples

Matrix-Free Methods

Summary

Abstract linear operator

linear function $f(x) = Ax$

- idea: *don't form, store, or use* the matrix A
- *forward-adjoint oracle (FAO)*: access f only via its
  - forward operator, $x \rightarrow f(x) = Ax$
  - adjoint operator, $y \rightarrow f^*(y) = A^T y$
- we are interested in cases where this is more efficient (in memory or computation) than forming and using A
- key to scaling to (some) large problems

Examples of FAOs

convolution, DFT	$O(n \log n)$
Gauss, Wavelet, and other transforms	$O(n)$
Lyapunov, Sylvester mappings $X \to AXB$	$O(n^{1.5})$
sparse matrix multiply	$O(\mathbf{nnz}(A))$
inverse of sparse triangular matrix	$O(\mathbf{nnz}(A))$

Compositions of FAOs

- represent linear function f as computation graph
    - graph inputs represent x
    - graph outputs represent y
    - nodes store FAOs
    - edges store partial results
- to evaluate $f(x)$: evaluate node forward operators in order
- to evaluate $f^*(y)$: evaluate node adjoints in reverse order

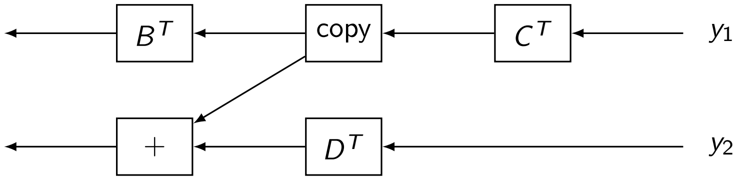
Forward graph

$$Ax = \begin{bmatrix} C(Bx_1 + x_2) \\ Dx_2 \end{bmatrix}$$

$$\begin{tikzcd}
x_1 \arrow[r] & \boxed{B} \arrow[r] & \boxed{+} \arrow[r] & \boxed{C} \arrow[r] & {} \\
x_2 \arrow[r] & \boxed{\text{copy}} \arrow[r] \arrow[ur] & \boxed{D} \arrow[rr] & & {}
\end{tikzcd}$$

Adjoint graph

$$A^T y = \left[\begin{array}{c} B^T C^T y_1 \\ C^T y_1 + D^T y_2 \end{array} \right]$$



Matrix-free methods

- *matrix-free algorithm* uses FAO representations of linear functions
- oldest example: conjugate gradients (CG)
  - minimizes $\|Ax - b\|_2^2$ using only $x \rightarrow Ax$ and $y \rightarrow A^T y$
  - in theory, finite algorithm
  - in practice, not so much
- many matrix-free methods for other convex problems (Pock-Chambolle, Beck-Teboulle, Osher, Gondzio, ...)
- can deliver modest accuracy in 100s or 1000s of iterations
- need good preconditioner, tuning

Matrix-free cone solvers

- matrix-free interior-point [Gondzio]
- matrix-free SCS [Diamond, O'Donoghue, Boyd] (serial CPU implementation)
- matrix-free POGS [Fougner, Diamond, Boyd] (GPU implementation)

- for use as a modeling language back end, we are interested only in *general preconditioners*

Matrix-free CVXPY

preliminary version [Diamond]

- canonicalizes to a matrix-free cone program
- solves using matrix-free SCS or POGS

Matrix-free CVXPY

preliminary version [Diamond]

- canonicalizes to a matrix-free cone program
- solves using matrix-free SCS or POGS

our (modest?) goals: MF-CVXPY should often

- work without algorithm tuning
- be no more than 10$\times$ slower than a custom method

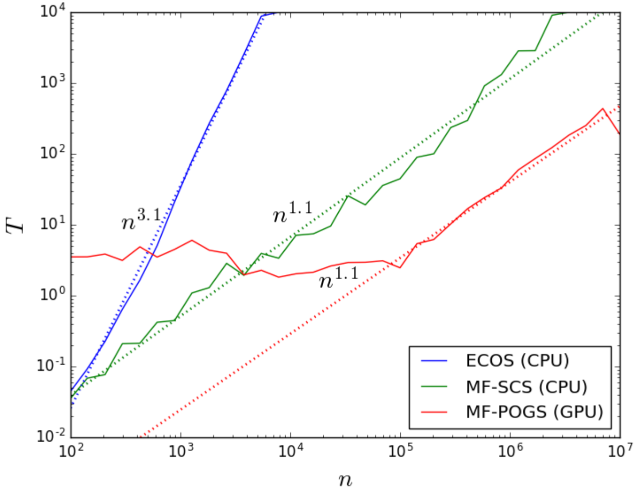
Example: Nonnegative deconvolution

$$\begin{array}{ll} \text{minimize} & \|c * x - b\|_2 \\ \text{subject to} & x \geq 0 \end{array}$$

variable $x \in \mathbf{R}^n$; data $c \in \mathbf{R}^n$, $b \in \mathbf{R}^{2n-1}$

- standard (matrix) method
  - represent $c*$ as $(2n-1) \times n$ Toeplitz matrix
  - memory is order n^2, solve is order n^3
- matrix-free method
  - represent $c*$ as FAO (implemented via FFT)
  - memory is order n, solve is order $n \log n$

Nonnegative deconvolution timings



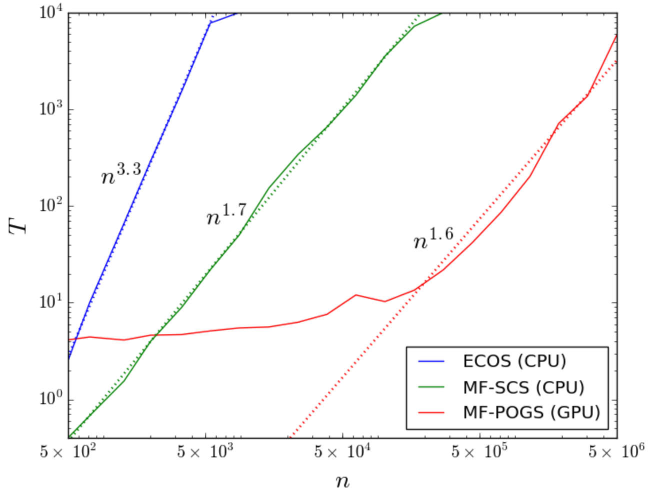
Sylvester LP

$$\begin{array}{ll} \text{minimize} & \mathbf{Tr}(D^T X) \\ \text{subject to} & AXB \leq C \\ & X \geq 0, \end{array}$$

variable $X \in \mathbf{R}^{p \times q}$; data $A \in \mathbf{R}^{p \times p}$, $B \in \mathbf{R}^{q \times q}$, $C, D \in \mathbf{R}^{p \times q}$
$n = pq$ variables, $2n$ linear inequalities

- standard method
  - represent $f(X) = AXB$ as $pq \times pq$ Kronecker product
  - memory is order n^2, solve is order n^3
- matrix-free method
  - represent $f(X) = AXB$ as FAO
  - memory is order n, solve is order $n^{1.5}$

Sylvester LP timings



Outline

Convex Optimization

Examples

Matrix-Free Methods

Summary

Summary

- convex optimization problems **arise in many applications**
- small and medium size problems **can be solved effectively and conveniently** using domain-specific languages, general solvers

Summary

- convex optimization problems **arise in many applications**
- small and medium size problems **can be solved effectively and conveniently** using domain-specific languages, general solvers
- we hope to extend this to large scale problems, fast operators

Resources

all available online

- *Convex Optimization* (book)
- *EE364a* (course slides, videos, code, homework, . . .)
- CVX, CVXPY, Convex.jl, SCS, POGS (code)
- preliminary version of MF-CVXPY (and SCS and POGS):
  `https://github.com/SteveDiamond/cvxpy`