

CVXOPT: Convex optimization with Python

Joachim Dahl

Department of Communication Technology
Aalborg University

Lieven Vandenberghe

Department of Electrical Engineering
University of California, Los Angeles

Workshop on Machine Learning Open Source Software

NIPS 2006

Outline

- **Motivation**
- Matrix library
- Optimization solvers
- Applications and modeling

Convex optimization

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

$x = (x_1, \dots, x_n)$ is the optimization variable

f_0, \dots, f_m are convex functions

- Includes least-squares, linear programming, many other problem classes.
- Convex optimization problems are fundamentally tractable.

History

- 1940s: linear programming

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & a_i^T x \leq b_i, \quad i = 1, \dots, m \end{array}$$

- 1950s: quadratic programming
- 1960s: geometric programming
- 1990s: semidefinite programming, second-order cone programming, quadratically constrained quadratic programming, robust optimization, sum-of-squares programming, . . .

New applications since 1990

- Linear matrix inequality techniques in control
- Circuit design via geometric programming
- Support vector machine learning via quadratic programming
- Semidefinite programming relaxations in combinatorial optimization
- Applications in statistics and machine learning, signal processing, communications, image processing, quantum information theory, finance, structural optimization, . . .

Interior-point methods

Linear programming

- 1984 (Karmarkar): first practical polynomial-time algorithm
- 1984-1990: efficient implementations for large-scale LPs

Nonlinear convex optimization

- Around 1990 (Nesterov & Nemirovski): polynomial-time interior-point methods for nonlinear convex programming
- Since 1990: extensions and high-quality software packages

Embedded convex optimization

- Efficient, in theory and practice
- Performance is similar across wide range of problem dimensions, problem data, problem classes
- Result is independent of choice of starting point
- Detect infeasibility
- Controlled by a small number of easily tuned algorithm parameters

Hence, useful for **embedded** optimization, *i.e.*, integrated in applications.

Software for convex optimization

- Proprietary packages: CPLEX, MOSEK, . . .
- Free, based on Matlab: SeDuMi, SDPT3, YALMIP, CVX, . . .
- Free, in the form of C libraries: GLPK, DSDP, SDPA, . . .
- Exploit certain types of problem structure (sparsity).

In practice, integrating convex optimization in applications often requires:

- substantial coding effort for interfacing with application,
- proprietary software,
- implementations that exploit (non-sparse) problem structure.

CVXOPT

Goals

- Free, portable package for convex optimization.
- Simplify development of embedded convex optimization applications.
- Take advantage of Python's extension modules.
- Matrix library for prototyping new algorithms.

Not: general-purpose computing package like Matlab, Octave

Current release (0.8.1)

- Dense and sparse matrices
- Interfaces to BLAS, LAPACK, CHOLMOD, UMFPACK, FFTW.
- Python solver for linear and semidefinite programming
- Python solver for nonlinear convex optimization
- Interfaces to solvers in GLPK, MOSEK and DSDP
- Modeling for piecewise-linear convex optimization problems

Outline

- Motivation
- **Matrix library**
- Optimization solvers
- Applications and modeling

Dense and sparse matrices

Dense matrices

- Three types: integer, double, complex
- Two-dimensional arrays stored in column-major order
- Compatible with NumPy via Array Interface

Sparse matrices

- Two types: double, complex
- Compressed column storage format

Matrix arithmetic

Overloaded arithmetic: $A + B$, $A - B$, $A * B$, . . .

In-place addition and scalar multiplication: $A += B$, $A *= c$, . . .

- Allowed if the dimensions and type of A do not change
- Modify existing matrix object ($A += B$ is not the same as $A = A + B$)

Indexing: with one or two arguments, using index sets or Python slices

Matrix functions: elementwise multiplication, cosine, exp, . . .

Basic Linear Algebra Subprograms (BLAS) interface

- Larger set of arithmetic operations (e.g., $A := \alpha A + \beta xy^T, \dots$)
- Separate routines for symmetric, triangular, banded matrices
- Always work 'in-place', i.e., without creating new matrices
- Some functions overloaded to allow sparse arguments

LAPACK interface

- LU, Cholesky, LDL^T , QR factorization
- Solution of dense linear equations
- Linear least-squares and least-norm problems
- Symmetric and Hermitian eigenvalue decomposition
- Singular value decomposition

Sparse linear equations

- Interface to UMFPACK (LU factorization) and CHOLMOD (Cholesky)
- Separate functions for symbolic and numeric factorization

Example: compute

$$A^{-T} B^{-1} A^{-1} b$$

where A , B have same sparsity pattern

```
Fs = umfpack.symbolic(A)
FA = umfpack.numeric(A, Fs)
FB = umfpack.numeric(B, Fs)
umfpack.solve(A, FA, b)
umfpack.solve(B, FB, b)
umfpack.solve(A, FA, b, trans='T')
```

Differences with Matlab and Octave

- Only a small subset of Matlab/Octave functionality
- Direct interface to BLAS, LAPACK, CHOLMOD, UMFPACK
- Function arguments are passed by reference
- Python scoping rules are different

Example: solve positive definite $Ax = b$ with different righthand sides

```
def solver(A):
    lapack.potrf(A)                # A = L*LT;  A := L
    def f(x): lapack.potrs(A,x)    # x := L-T*L-1*x
    return f
....
f = solver(A)
f(x)    # x := A-1*x
f(y)    # y := A-1*y
```

Outline

- Motivation
- Matrix library
- **Optimization solvers**
- Applications and modeling

Nonlinear convex optimization

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

- Simple solver: user provides functions for evaluating constraint functions and derivatives
- Expert solver: user also provides function for solving KKT systems
- Interfaces for quadratic programming and geometric programming

Example (simple solver): logistic regression

$$\text{minimize } f(x) = c^T x + \sum_{i=1}^m \log(1 + \exp((Ax)_i))$$

```
m, n = A.size
def F(x=None, z=None):
    if x is None: return 0, matrix(0.0, (n,1))
    w = exp(A*x)
    # function value and gradient
    f = c.T*x + sum(log(1+w))
    grad = c + A.T * div(w, 1+w)
    if z is None: return f, grad.T
    # Hessian
    H = A.T * spmatrix(div(w, (1+w)**2), range(m), range(m)) * A
    return f, grad.T, z[0]*H

sol = solvers.cp(F)
```

Example (expert solver): 1-norm regularized regression

$$\text{minimize } \|Ax - y\|_2^2 + \|x\|_1 \quad (A \in \mathbf{R}^{m \times n}, n \gg m)$$

- Equivalent to a quadratic program.
- Every iteration involves solving a set of linear (KKT) equations:

$$\begin{bmatrix} \lambda A^T A & 0 & I & -I \\ 0 & 0 & -I & -I \\ I & -I & -D_1^{-1} & 0 \\ -I & -I & 0 & -D_2^{-1} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z_1 \\ \Delta z_2 \end{bmatrix} = b$$

Reduces to a system of the form $(ADA^T + I)v = r$.

- Structure can be exploited by providing a routine for KKT system.

Linear cone programming

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

Inequalities include vector (componentwise) inequalities and matrix inequalities.

- Simple solvers: use standard matrix format
- Expert solver: user provides function for solving KKT system

Example: 1-norm approximation

$$\text{minimize } \|Pu - q\|_1$$

Equivalent to the LP

$$\begin{aligned} &\text{minimize } \mathbf{1}^T y \\ &\text{subject to } -y \preceq Pu - q \preceq y \end{aligned}$$

Every iteration involves solving sets of linear equations:

$$\begin{bmatrix} 0 & 0 & P^T & -P^T \\ 0 & 0 & -I & -I \\ P & -I & -D_1^{-1} & 0 \\ -P & -I & 0 & -D_2^{-1} \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta y \\ \Delta z_1 \\ \Delta z_2 \end{bmatrix} = b.$$

Reduces to a system of the form

$$P^T D P \Delta u = r$$

```

def l1(P, q):
    m, n = P.size
    def Fi(x, y, alpha=1.0, beta=0.0, trans='N'):
        if trans=='N':
            ... # evaluate  $y := \alpha * [P, -I; -P, -I] * x + \beta * y$ 
        else:
            ... # evaluate transpose

    def kkt_solver(di): # Return a solver for KKT system
        # Compute and factor  $A = P' * D * P$ 
        ...
        def f(x,y,z): # Solve KKT system using factorization
            ...
        return f

    c = matrix(size=(m+n,1)); c[:n], c[n:] = 0.0, 1.0
    h = matrix(size=(2*m,1)); h[:m], h[m:] = q, -q
    sol = solvers.conelp(c, kkt_solver, G1=Fi, h1=h)

```

Outline

- Motivation
- Matrix library
- Optimization solvers
- **Applications and modeling**

Applications

- Interactive shell
- Spreadsheet interfaces (gnumeric)
- Database applications
- Networking and distributed applications
- Graphical user interfaces
- Web interfaces
- Modeling via built-in Python interpreter

Piecewise-linear optimization

- Python object for (vector) variables

```
x = variable(5, 'x')
```

- Create PWL functions by overloaded arithmetic operations, `max()`, `min()`, `abs()`, `sum()`, indexing and slicing

```
f = sum(abs(A*x+b))
```

- Create constraints by overloaded comparison operators

```
c1 = (f <= 1)  
c2 = (sum(x) == 1)
```

- Define optimization problem by specifying objective and constraints

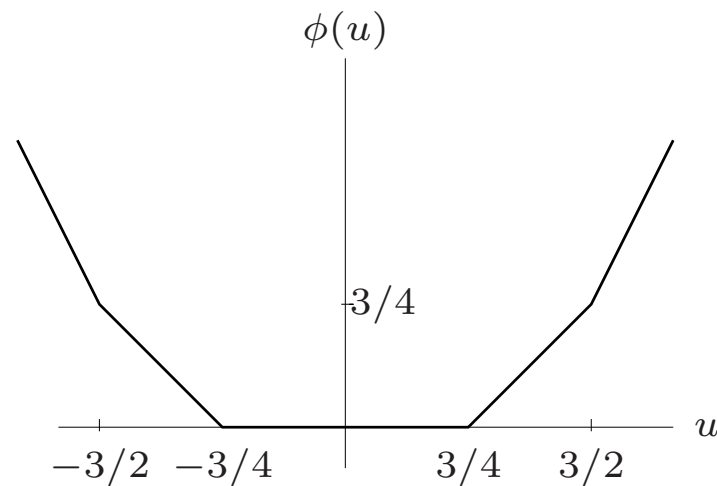
```
prob = op(x[1], [c1, c2])
```

Example: Penalty approximation

Chebyshev norm: minimize $\|Ax - b\|_\infty$

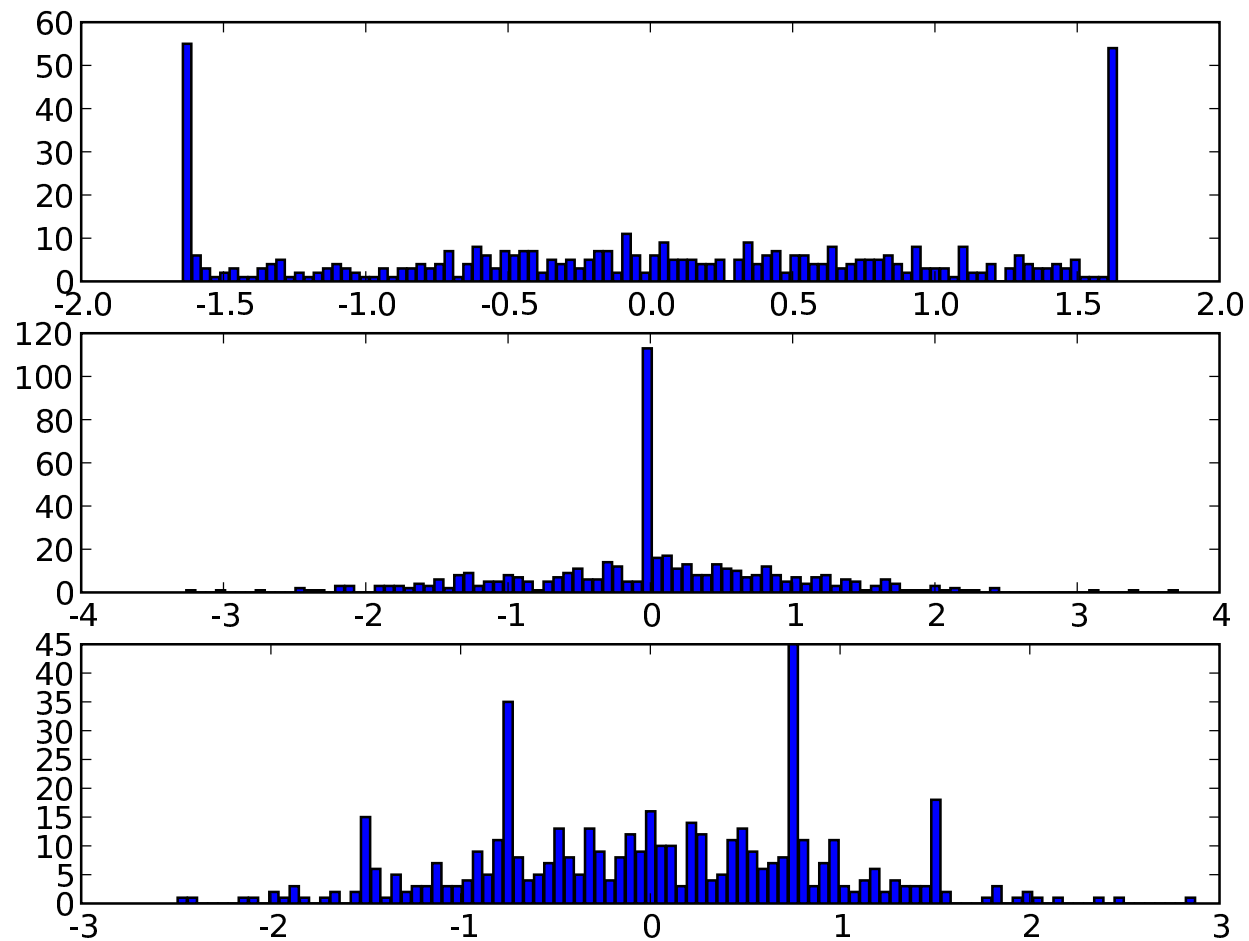
1-norm: minimize $\|Ax - b\|_1$

Piecewise-linear penalty: minimize $\sum_k \phi((Ax - b)_k)$



Can be expressed as LPs, by introducing auxiliary variables and constraints.

Example: distribution of residuals $Ax - b$ ($A \in \mathbf{R}^{500 \times 200}$).



1. Minimize $\|Ax - b\|_\infty$:

```
x1 = variable(n)
prob1 = op(max(abs(A*x1-b)))
prob1.solve()
```

2. Minimize $\|Ax - b\|_1$:

```
x2 = variable(n)
prob2 = op(sum(abs(A*x2-b)))
prob2.solve()
```

3. Minimize $\sum_k \phi((Ax - b)_k)$:

```
x3 = variable(n)
prob3 = op(sum(max(0, abs(A*x3-b)-0.75, 2*abs(A*x3-b)-2.25)))
prob3.solve()
```

Summary

Applications

- Embedded convex optimization via Python's extension libraries
- In interactive mode, a user-friendly package for convex optimization
- Prototyping of numerical algorithms (linear algebra and optimization)

Some future plans

- Iterative linear algebra
- Second-order cone programming
- Python implementation of CVX modeling tool (Grant, Boyd, Ye)

www.ee.ucla.edu/~vandenbe/cvxopt