On affine conic and disjunctive constraints in Mosek version 10

June 15, 2022

Erling D. Andersen
(joint work with Sven Wiese)
MOSEK ApS,
Company WWW: https://mosek.com

www.mosek.com

Outline

Section 1

Background

Mosek

- A software package for solving:
  - Linear and **conic** problems.
  - Convex quadratically constrained problems.
  - Also mixed-integer versions of the above.
- Current version: 9.3.
- Current beta version: 10.0.

Section 2

Outline

Todays topics

- Conic optimization
  - What is it?
  - Why?
  - New interface to specify conic constraints.
- Disjunctive constraints
  - What is it?
  - Why?
  - How to specify disjunctive constraints.

Section 3

Conic optimization

Generic conic optimization problem
Primal form

$$
\begin{array}{lll}
\text{minimize} & \sum_k (c^k)^T x^k & \\
\text{subject to} & \sum_k A^k x^k & = \quad b, \\
 & x^k \in \mathcal{K}^k, & \quad \forall k,
\end{array}
$$

where

- $c^k \in \mathbb{R}^{n^k}$,
- $A^k \in \mathbb{R}^{m \times n^k}$,
- $b \in \mathbb{R}^m$,
- $\mathcal{K}^k$ are convex cones.

Comments:

- Wikipedia reference:
  https://en.wikipedia.org/wiki/Convex_cone.
- Cone types:
  - Linear.
  - Quadratic.
  - Semi-definite.
  - Power cone.
  - Exponential.
- Most convex optimization models can be formulated with the **5** convex cone types.
- Evidence: Lubin [1] shows all convex instances (333) in the benchmark library MINLPLIB2 is conic representable using the 5 cones.

Challenge: Design an interface

Mosek version 9 optimizer API interface:

$$\begin{bmatrix} x_5 \\ x_6 \\ x_8 \end{bmatrix} \in \mathcal{K}^k$$

- Only vector of variables can belong to a cone.
- In principle general. (Use artificial variables to handle affine expressions).
- Cumbersome to use at least for some models. (One solution: Cvxpy or Fusion).
- Want to specify an affine expression belonging to a cone easily.

I.e.

$$F^k x + f^k \in \mathcal{K}^k + g^k$$

Challenge:

- How to build an interface for this type of constraint.
- That works in a low level language like C.
- Extensible to new cone types.
- Efficient i.e. low space and computational overhead.
- Why the g? Reason reuse of affine expression:

$$2 \leq x + y + 1 \text{ and } x + y + 1 \leq 6.$$

The affine expressions storage

Define

$$Fx + f$$

which is a store/dictionary of affine expressions.
Assumptions:

- F is a sparse matrix and f is a dense vector.
- Affine expressions can be appended but never deleted. (Important assumption!)
- Variables (x) can be appended and deleted.
- Affine expressions can be modified.

Hence

$$F^k x + f^k = F_{\mathcal{I},:} x + f_{\mathcal{I}}$$

where $\mathcal{I}$ is an ordered list of indexes.

Hence,

- F^k is not provided explicitly.
- Represented by a list of indexes into the affine expression store.

The domains storage

Introduce

$$\mathcal{D}$$

which is a list of domains. A domain

- has a dimension d.
- has type e.g. the exponential cone type.
- has potentially some associated parameters e.g. $\alpha's$ for the power cone.
- can never be deleted.

An affine conic constraint

An affine conic constraint consist of:

- An ordered list of affine expressions indexes.
- A domain index.
- A g vector.

and represents

$$F_{\mathcal{I},:}x + f_{\mathcal{I}} \in \mathcal{D}_k + g.$$

Comments:

- Conic constraints can be appended, deleted and modified.
- Dimension checking is simple.
- Everything is (easily) implementable in C.
- Extensible with new cone types. (Introduce new domains.)
- Can be implemented efficiently.

Section 4

Disjunctive constraints

Modeling alternatives via disjunctions

A disjunctive constraint is of the form

$$[D^1 x = b^1] \vee [D^2 x = b^2] \vee ... \vee [D^l x = b^l].$$

where

- $D^k \in \mathbb{R}^{m^k \times n}$,
- $b^k \in \mathbb{R}^{m^k}$.

- Inequalities are fine.
- Wiki reference: `https://optimization.mccormick.northwestern.edu/index.php/Disjunctive_inequalities`.

Applications of disjunctive constraints

- Simple OR conditions: two jobs i, j with start times s_i, s_j have to be executed on the same machine, but only one at a time. So which one first?

$$[s_j \geq s_i + duration_i] \vee [s_i \geq s_j + duration_j].$$

- Semi-continuous variables: $[x = 0] \vee [l \leq x \leq u]$.

- An indicator constraint $z = 1 \implies \left[d^T x \leq b\right]$ is the same as

$$[z = 0] \vee \left[d^T x \leq b\right].$$

Mosek's new `.mps` and `.lp` readers translate indicator constraints directly to disjunctions.

Applications of disjunctive constraints

- Only one among some variables maybe non-zero, i.e., SOS1$(x_1, \ldots, x_k)$:

$$[y_i = 1] \vee [x_i = 0],\ y_i \in \{0, 1\}\ \forall i,\ \sum_i y_i \leq 1.$$

- Complementarity constraints: $s \cdot t = 0 \Longleftrightarrow [s = 0] \vee [t = 0]$.

- Piecewise linear functions:

$$[f = x, 0 \leq x \leq 1] \vee [f = 1 - 2x, 1 \leq x \leq 2] \vee [f = -3, 2 \leq x].$$

Disjunctive and big-M constraints

- Often disjunctive constraints can be reformulated with the big-M principle to ordinary Mixed-integer models (and Mosek may do this automatically!).

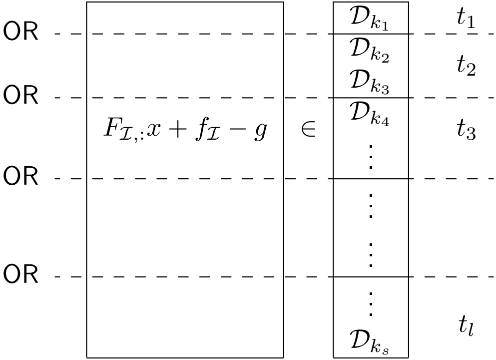
- For example the indicator constraint: $d^T x \leq b + M(1 - z)$.

- In some cases though, when (a valid) M is large or even infinite, this can lead to a certain loss in solution accuracy.

Use the affine expressions and domain storages!

A disjunctive constraint in Mosek 10 consists of:

- An ordered list of affine expression indexes.
- An ordered list of domain indexes (only linear ones for now!).
- A g vector.
- A list of term sizes $t_1, \ldots, t_l$:



Summary

- Described how to specify general affine conic constraints v10.
    - Efficient yet quite general when implemented in C.
    - Easy to extend to new cone types.
- Described how to specify disjunctive constraints in v10.
    - Generalize many special constructs such as semi-continuous variables, indicator constraints, SOS1 constraints etc.
    - Make it possible to get rid of big-Ms.
    - For now mainly syntactic sugar.

References I

[1] M. Lubin and E. Yamangil and R. Bent and J. P. Vielma.
Extended Formulations in Mixed-integer Convex Programming.
In Q. Louveaux and M. Skutella, editors, *Integer Programming and Combinatorial Optimization. IPCO 2016. Lecture Notes in Computer Science, Volume 9682*, pages 102–113. Springer, Cham, 2016.