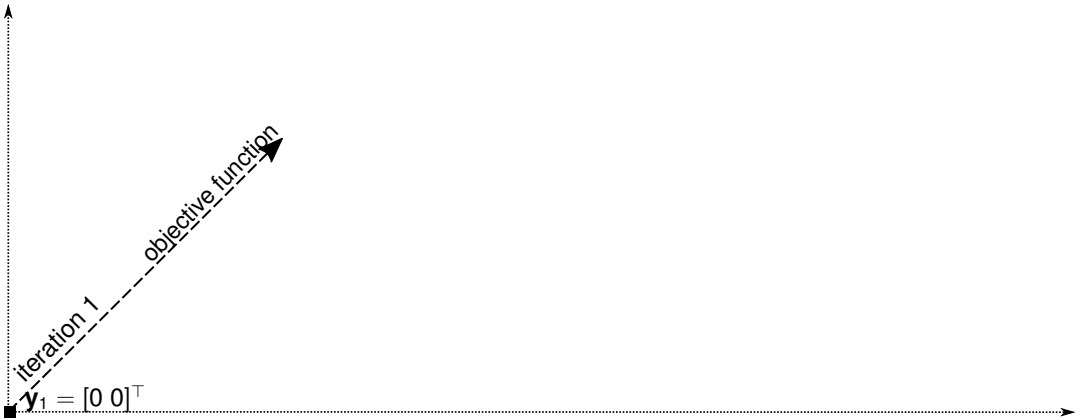
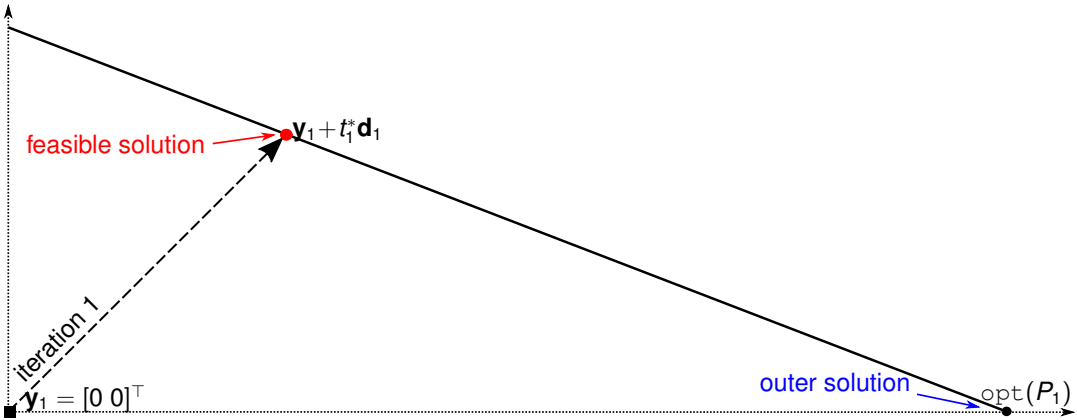
Dense Semidefinite Programming by Projective Cutting-Planes

Daniel Porumbel

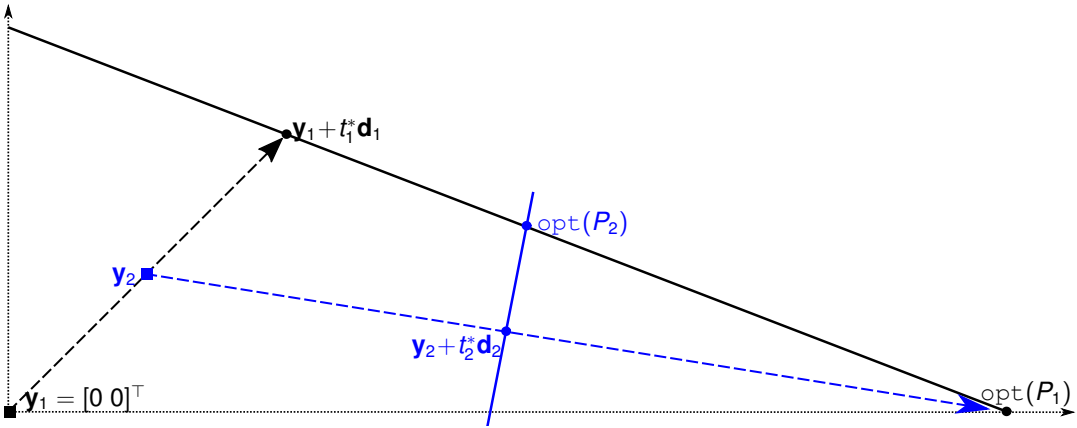
Conservatoire National des Arts et Métiers, Paris



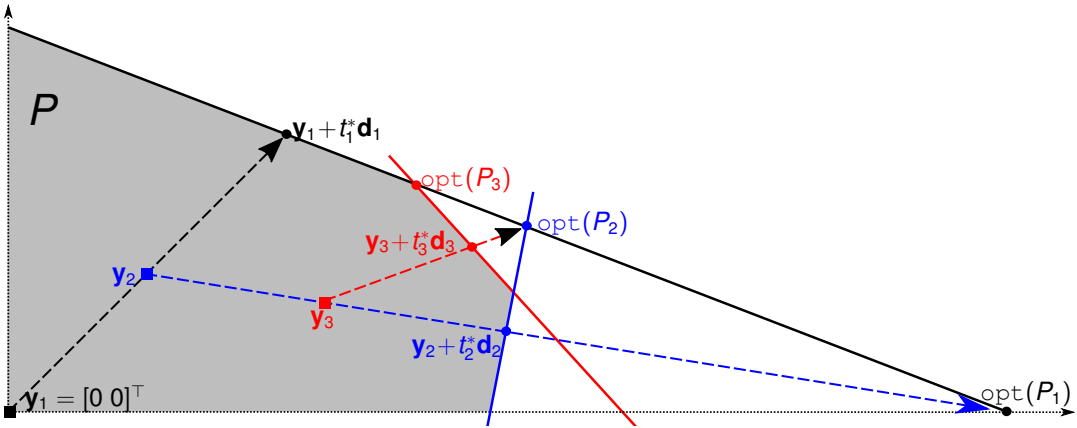
Iteration 1 : uncharted territory, follow objective function, i.e., advance along $\mathbf{y}_1 \rightarrow \mathbf{d}_1$ where $\mathbf{d}_1$ takes the value of the objective function



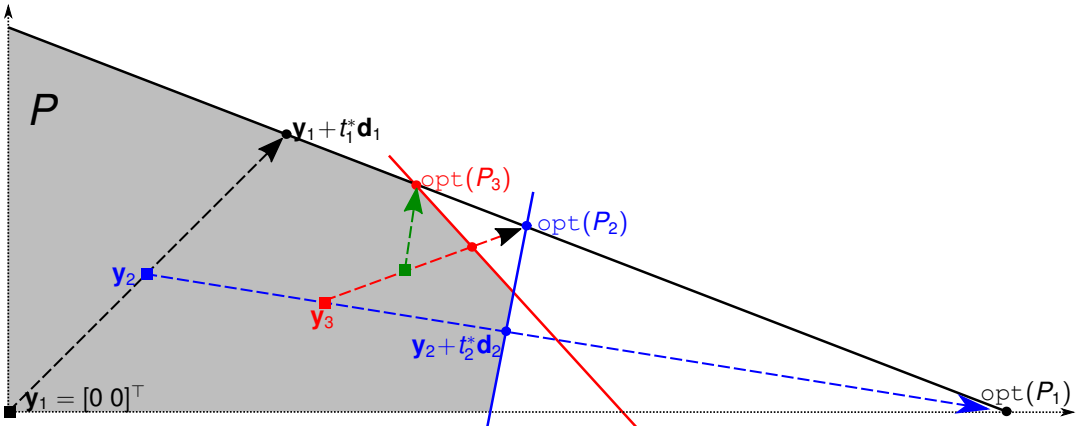
Iteration 1 : found a first outer solution $\mathtt{opt}(P_1)$ and a first inner solution (contact point) $\mathbf{y}_1 + t_1^* \mathbf{d}_1$



Iteration 2 : an inner feasible solution (contact point) $\mathbf{y}_2 + t_2^* \mathbf{d}_2$ and a new outer solution. We take $\mathbf{d}_2 = \mathtt{opt}(P_1) - \mathbf{y}_2$.

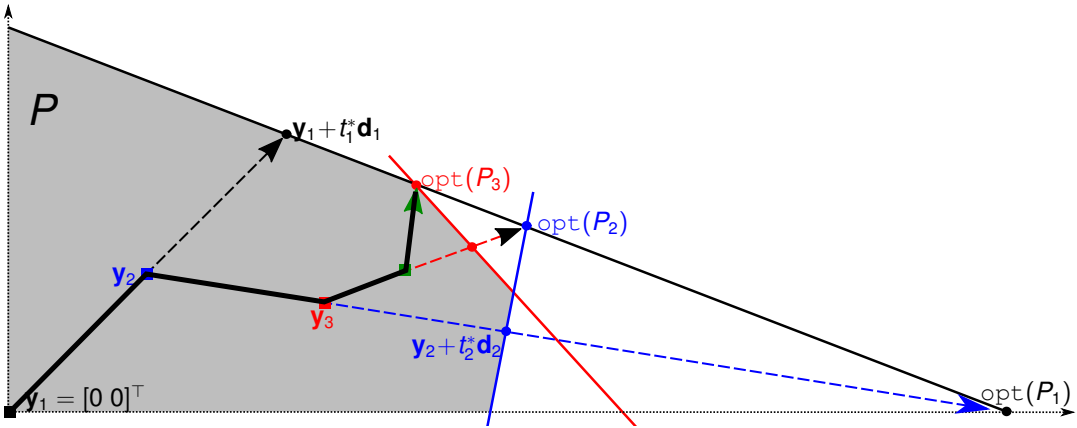


Iteration 3 : the feasible solution $\mathbf{y}_3 + t_3^* \mathbf{d}_3$ is almost optimal



Iteration 4 : optimality of $\mathtt{opt}(P_3)$ proved
You can see the proposed method is convergent because it solves a separation problem on $\mathtt{opt}(P_k)$ at each iteration k

- The convergence proof takes two lines, cool !



Building on existing work [1,2], the new method was deliberately designed to be more general and when possible simpler

[1] Daniel Porumbel. Ray projection for optimizing polytopes with prohibitively many constraints in set-covering column generation. *Mathematical Programming*, 155(1) :147–197, 2016.

[2] Daniel Porumbel. Projective Cutting-Planes, *SIAM Journal on Optimization*, 30(1) : 1007-1032, 2020

We now focus on the following standard (semidefinite programming) SDP problem in scalar variables $y_1, y_2, \dots y_k$, where $C, A_1, A_2, \dots A_k$ are symmetric $n \times n$ matrices.

$$(SDP)\begin{cases} \max\limits_{\mathbf{y}} & \mathbf{b}^\top \mathbf{y} = b_1 y_1 + b_2 y_2 + \dots b_k y_k \\ s.t & X = C - \sum_{i=1}^{k} A_i y_i \text{ is semidefinite positive (SDP)} \\ & \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ & T\mathbf{y} \le \mathbf{s} \qquad \rightarrow \text{variables } \mathbf{y} \text{ (also) live in a polytope} \end{cases}$$

We now focus on the following standard (semidefinite programming) SDP problem in scalar variables $y_1, y_2, \dots y_k$, where $C, A_1, A_2, \dots A_k$ are symmetric $n \times n$ matrices.

$$(SDP) \begin{cases} \max\limits_{\mathbf{y}} & \mathbf{b}^\top \mathbf{y} \\ s.t & X = C - \sum\limits_{i=1}^{k} A_i y_i \\ & X \succeq \mathbf{0} \iff X \bullet \mathbf{v}\mathbf{v}^\top \geq 0 \ \forall \mathbf{v} \in \mathbb{R}^n \\ & T\mathbf{y} \leq \mathbf{s} \qquad \rightarrow \text{variables } \mathbf{y} \text{ (also) live in a polytope} \end{cases}$$

using notation $\boxed{X \bullet \mathbf{v}\mathbf{v}^\top = \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} X_{ij} v_i v_j}$

Let us develop on the SDP constraint below. In some sense, it describes the SDP cone as a polytope with infinitely-many constraints, one for each $\mathbf{v} \in \mathbb{R}^n$.

$$X \succeq \mathbf{0} \iff X \bullet \mathbf{v}\mathbf{v}^\top \geq 0 \ \forall \mathbf{v} \in \mathbb{R}^n$$

Recall X is this linear combination of matrices defined by variables $y_1, y_2, \ldots, y_k$

$$X = C - \sum_{i=1}^{k} A_i y_i$$

Let us develop on the SDP constraint below. In some sense, it describes the SDP cone as a polytope with infinitely-many constraints, one for each $\mathbf{v} \in \mathbb{R}^n$.

$$X \succeq \mathbf{0} \iff X \bullet \mathbf{v}\mathbf{v}^\top \geq 0 \; \forall \mathbf{v} \in \mathbb{R}^n$$

Recall X is this linear combination of matrices defined by variables $y_1, y_2, \ldots, y_k$

$$X = C - \sum_{i=1}^{k} A_i y_i$$

Solution $\mathbf{y} = [y_1 y_2 \ldots y_k]^\top$ respects above SDP constraint if and only if

$$\left(C - \sum_{i=1}^{k} A_i y_i \right) \bullet \mathbf{v}\mathbf{v}^\top \geq 0, \quad \forall \mathbf{v} \in \mathbb{R}^n \quad \text{equiv to}$$

$$\left(\sum_{i=1}^{k} A_i y_i \right) \bullet \mathbf{v}\mathbf{v}^\top \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \mathbb{R}^n \quad \text{equiv to...}$$

Let us develop on the SDP constraint below. In some sense, it describes the SDP cone as a polytope with infinitely-many constraints, one for each $\mathbf{v} \in \mathbb{R}^n$.

$$X \succeq \mathbf{0} \iff X \bullet \mathbf{v}\mathbf{v}^\top \geq 0 \; \forall \mathbf{v} \in \mathbb{R}^n$$

Recall X is this linear combination of matrices defined by variables $y_1, y_2, \ldots, y_k$

$$X = C - \sum_{i=1}^{k} A_i y_i$$

Solution $\mathbf{y} = [y_1 y_2 \ldots y_k]^\top$ respects above SDP constraint if and only if

$$\left(C - \sum_{i=1}^{k} A_i y_i \right) \bullet \mathbf{v}\mathbf{v}^\top \geq 0, \quad \forall \mathbf{v} \in \mathbb{R}^n \quad \text{equiv to}$$

$$\sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top \right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \mathbb{R}^n$$

The eigenvector corresponding to the minimum eigenvalue of X gives the strongest constraint for a given $\mathbf{y}$.

$$(SDP)\begin{cases} \max\limits_{\mathbf{y}} & \mathbf{b}^\top \mathbf{y} \\ s.t & \sum\limits_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top \right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \mathbb{R}^n \\ & T\mathbf{y} \leq \mathbf{s} \qquad \rightarrow \text{variables } \mathbf{y} \text{ (also) live in a polytope} \end{cases}$$

We could address the problem by progressively separating infeasible solutions $\mathbf{y}_{\text{out}} \in \mathbb{R}^k$, generating at each iteration a $\mathbf{v}$ associated to the minimum eigenvalue of X. This standard `Cutting-Planes` is not considered very effective; work on this area remains a rare sight (a half-dozen of articles).

Yet the cutting-planes framework has some advantages. Imagine that the tight saturated constraints generated at some iteration belong to some set $\texttt{VCUTS}_{\texttt{now}}$:

$$\begin{aligned}\mathbf{b}^\top \mathbf{y}_{\text{out}} = \max_{\mathbf{y}} \quad & \mathbf{b}^\top \mathbf{y} \\ \alpha: \quad & \sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top\right) y_i \le C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \texttt{VCUTS}_{\texttt{now}}\end{aligned}$$

The dual LP solution $\alpha \ge \mathbf{0}$ will satisfy

$$\sum_{\mathbf{v} \in \texttt{VCUTS}_{\texttt{now}}} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top\right) \alpha_{\mathbf{v}} = b_i, \ \forall i \in [1..k]$$

Yet the cutting-planes framework has some advantages. Imagine that the tight saturated constraints generated at some iteration belong to some set $\texttt{VCUTS}_{\texttt{now}}$:

$$\begin{aligned}\mathbf{b}^\top \mathbf{y}_{\text{out}} = \max_{\mathbf{y}} \quad & \mathbf{b}^\top \mathbf{y} \\ \alpha: \quad & \sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top\right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \texttt{VCUTS}_{\texttt{now}}\end{aligned}$$

The dual LP solution $\alpha \geq \mathbf{0}$ will satisfy

$$\sum_{\mathbf{v} \in \texttt{VCUTS}_{\texttt{now}}} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top\right) \alpha_{\mathbf{v}} = b_i, \ \forall i \in [1..k]$$

This is equivalent to

$$A_i \bullet \underbrace{\sum_{\mathbf{v} \in \texttt{VCUTS}_{\texttt{now}}} \mathbf{v}\mathbf{v}^\top \alpha_{\mathbf{v}}}_{Z \succeq \mathbf{0}} = b_i, \ \forall i \in [1..k]$$

which means Z is a feasible SDP solution in the dual SDP.

Yet the cutting-planes framework has some advantages. Imagine that the tight saturated constraints generated at some iteration belong to some set $\texttt{VCUTS}_{\texttt{now}}$:

$$\begin{aligned}\mathbf{b}^\top \mathbf{y}_{\text{out}} = \max_{\mathbf{y}} \quad & \mathbf{b}^\top \mathbf{y} \\ \alpha: \quad & \sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top\right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \texttt{VCUTS}_{\texttt{now}}\end{aligned}$$

The dual LP solution $\alpha \geq \mathbf{0}$ will satisfy

$$\sum_{\mathbf{v} \in \texttt{VCUTS}_{\texttt{now}}} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top\right) \alpha_{\mathbf{v}} = b_i, \ \forall i \in [1..k]$$

This is equivalent to

$$A_i \bullet \underbrace{\sum_{\mathbf{v} \in \texttt{VCUTS}_{\texttt{now}}} \mathbf{v}\mathbf{v}^\top \alpha_{\mathbf{v}}}_{Z \succeq \mathbf{0}} = b_i, \ \forall i \in [1..k]$$

which means Z is a feasible SDP solution in the dual SDP.
Projective-Cutting-Planes generates at each iteration a feasible primal solution $\mathbf{y} \in \mathbb{R}^k$ and a feasible dual $Z \in \mathbb{R}^{n \times n}$, something an Interior Point Method can not do (by default along the way).

An SDP heuristic

Projective-Cutting-Planes needs a feasible solution $\mathbf{y}$ to start. I found none in the literature, so I implemented one. Say $\mathbf{y}$ is not feasible, meaning there is a $\mathbf{v}$ associated to a negative eigenvalue λ_- of X such that

$$\underbrace{\left(C - \sum_{i=1}^{k} A_i y_i\right)}_{X} \bullet \mathbf{v}\mathbf{v}^\top = \lambda_- < 0$$

An SDP heuristic

Projective-Cutting-Planes needs a feasible solution $\mathbf{y}$ to start. I found none in the literature, so I implemented one. Say $\mathbf{y}$ is not feasible, meaning there is a $\mathbf{v}$ associated to a negative eigenvalue λ_- of X such that

$$\underbrace{\left(C - \sum_{i=1}^{k} A_i y_i\right)}_{X} \bullet \mathbf{v}\mathbf{v}^\top = \lambda_- < 0$$

But notice this f is linear and that $f(\mathbf{0}) = \lambda_-$

$$f(\mathbf{d}) = \left(C - \sum_{i=1}^{k} A_i (y_i + d_i)\right) \bullet \mathbf{v}\mathbf{v}^\top$$

We can maximize f over all $\mathbf{d}$ in a small hypercube. If that is above λ_-, we can make a step $\mathbf{y} = \mathbf{y} + \mathbf{d}$ and improve the minimum eigenvalue.
If there is a feasible solution $\mathbf{y}^*$, we have $f(\mathbf{y}^* - \mathbf{y}) > 0$. But since the minimum eigenvalue function is concave, this means $f(\epsilon(\mathbf{y}^* - \mathbf{y})) > 0$ for any $\epsilon \in (0, 1]$, so we can surely find an improving step $\mathbf{d}$ in the hypercube!

Upgrading separation $\rightarrow$ projection

$$(SDP)\begin{cases} \max\limits_{\mathbf{y}} \quad \mathbf{b}^\top \mathbf{y} \\ \quad s.t \quad \sum\limits_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top\right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \mathbb{R}^n \\ \qquad T\mathbf{y} \leq \mathbf{s} \qquad\qquad \rightarrow \text{variables } \mathbf{y} \text{ (also) live in a polytope} \end{cases}$$

In *Projective-Cutting-Planes*, we upgrade the separation sub-problem to the projection sub-problem: given feasible $\mathbf{y}$ in a feasible area (SDP) and an arbitrary direction $\mathbf{d} = \mathbf{y}_{\text{out}} - \mathbf{y}$, what is the maximum step-length t^* so that $\mathbf{y} + t^*\mathbf{d} \in (SDP)$?

Upgrading separation $\rightarrow$ projection

$$(SDP)\begin{cases}\max\limits_{\mathbf{y}} & \mathbf{b}^\top\mathbf{y} \\ s.t & \sum\limits_{i=1}^{k}\left(A_i \bullet \mathbf{v}\mathbf{v}^\top\right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \mathbb{R}^n \\ & T\mathbf{y} \leq \mathbf{s} \qquad \rightarrow \text{variables } \mathbf{y} \text{ (also) live in a polytope}\end{cases}$$

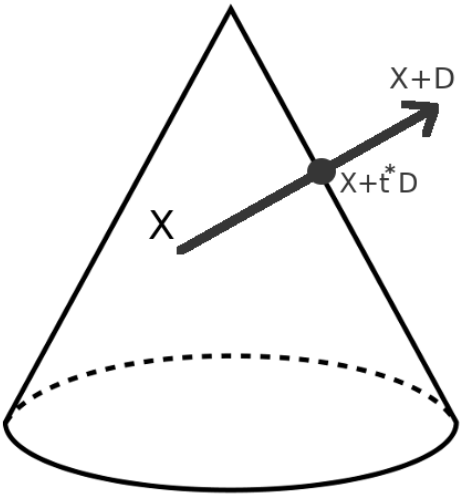
In *Projective-Cutting-Planes*, we upgrade the separation sub-problem to the projection sub-problem: given feasible $\mathbf{y}$ in a feasible area (SDP) and an arbitrary direction $\mathbf{d} = \mathbf{y}_{\text{out}} - \mathbf{y}$, what is the maximum step-length t^* so that $\mathbf{y} + t^*\mathbf{d} \in (SDP)$?

In SDP programming, projecting $\mathbf{y} \rightarrow \mathbf{d}$ requires solving $t^* = \max\{t : X + tD \succeq \mathbf{0}\}$ for this $X \succeq \mathbf{0}$ and D:

- $X = C - \sum_{i=1}^{k} A_i y_i$ is SDP when $\mathbf{y}$ is feasible
- $D = C - \sum_{i=1}^{k} A_i d_i$ may be SDP or not.

We have to project $X \to D$ in the SDP cone:

$$t^* = \max\{t : X + tD \succeq \mathbf{0}\}$$



We have to project $X \to D$ in the SDP cone:

$$t^* = \max\{t : X + tD \succeq \mathbf{0}\}$$

An easy-to-implement approach: notice t^* is the generalized eigenvalue of X and $-D$. The corresponding generalized eigenvector $\mathbf{v}$ satisfies $X\mathbf{v} = -t^* D\mathbf{v}$.

This is far too slow: we need a very particular generalized eigenvalue, namely, *the lowest real eigenvalue above 0*. With existing software, computing all eigenvalues or only the eigenvalues close to zero seems much too slow.

The main challenge is the speed of the projection algorithm. It should be closer to computing Cholesky or the smallest eigenvalue of matrix. If the speed is closer to computing the whole eigendecomposition, all seems lost.

```
And speed it's all you'll ever need.
All you'll ever need to know.
You and me we're going nowhere slowly
You go down on the pedal and you're ready to roll.
              Meat Loaf - Nowhere slowly
```

Recall we have to project $X \to D$ in the SDP cone:

$$t^* = \max\{t : X + tD \succeq \mathbf{0}\}$$

Th projection sub-problem is quite simple if $X \succ \mathbf{0}$. In this case, there is a unique Cholesky decomposition $X = KK^\top$ and K is non-singular and triangular. We can determine by simple and fast back-substitution D' as the unique solution of $D = KD'K^\top$.

Recall we have to project $X \to D$ in the SDP cone:

$$t^* = \max\{t : X + tD \succeq \mathbf{0}\}$$

Th projection sub-problem is quite simple if $X \succ \mathbf{0}$. In this case, there is a unique Cholesky decomposition $X = KK^\top$ and K is non-singular and triangular. We can determine by simple and fast back-substitution D' as the unique solution of $D = KD'K^\top$.
The following are equivalent:

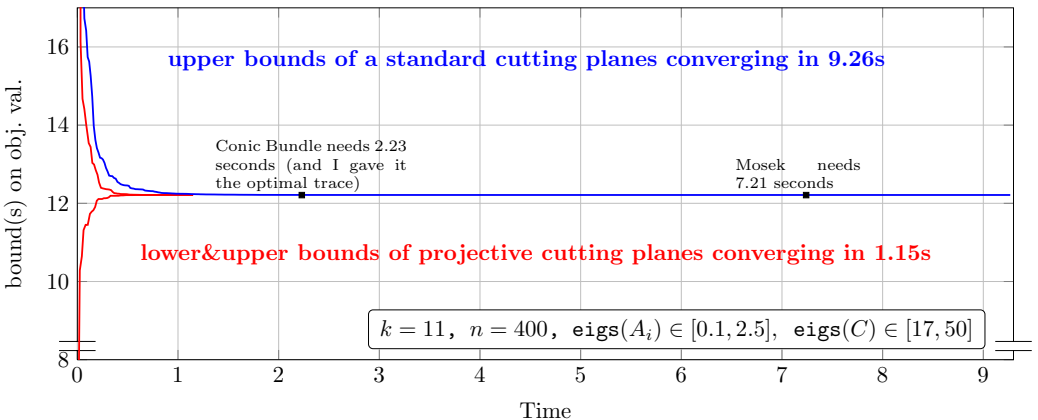
$$\begin{aligned} t^* &= \max\{t : X &&+ tD \succeq \mathbf{0}\} \\ t^* &= \max\{t : KK^\top &&+ tKD'K^\top \succeq \mathbf{0}\} \\ t^* &= \max\{t : I_n &&+ tD' \succeq \mathbf{0}\} \end{aligned}$$

We can determine $\max\{t : I_n + tD' \succeq \mathbf{0}\}$ by computing $\lambda_{\min}(D')$ Total projection cost:

- one Cholesky
- a few back-substitutions
- one minimum eigenvalue
- a fast calculation of products $\mathbf{v}\mathbf{v}^\top$
- $k + 1$ dot products of the form $A_i \bullet \mathbf{v}\mathbf{v}^\top$, recording A_i as an array

This projection is more difficult if X is not strictly SDP. Yet, the simplified pure SDP case enabled us to solve some particular instances very rapidly.

Instance below was solved by advancing to the optimum trough a sequence of strictly SDP matrices (strictly interior points): $X_0 = \epsilon I_n, X_1, X_2, X_3 \ldots$



Results on sparse and densified instances from the literature

Most instances are very very sparse and IPM algorithms really exploit this. In such cases, Proj-Cut-Planes is not competitive with mosek. But what if we densify the instances by adding some noise over all zeros?

	original sparse instance				densified instance		
	k	n	LP-cuts	mosek	mosek	Proj-Cut-Planes	
buck1	36	49	36	0.2	0.01	0.9	speed gain
buck2	144	193	144	1	6	4.2	42%
buck3	544	641	544	19	1320	828	60%
buck4	1200	1345	1200	163	38696	15931	142%

Results on highly feasible dense instances

Software	Instance						
	`highFsb5`	`highFsb10`	`highFsb15`	`highFsb20`	`highFsb25`	`highFsb30`	`highFsb50`
	$n, k = 50, 5$	$n, k = 100, 10$	$n, k = 150, 15$	$n, k = 200, 20$	$n, k = 250, 25$	$n, k = 300, 30$	$n, k = 500, 50$
Proj-cut-Planes	0.26	0.37	0.88	0.91	1.06	1.09	2.51
`Mosek`	0.04	0.17	0.77	2.39	5.34	11.7	79
`SeDuMi`	0.1	0.36	1.40	2.21	4.66	10.1	61
`ConicBundle`	0.06	0.62	0.97	1.71	—	—	—
`CSDP`	32	132	441	2215	—	—	—
`Clarabel`	5.26	339	—	—	—	—	—
`Loraine`	0.03	—	—	—	—	—	—

The polyhedral nature of the new method enables it to easily adapt to change, like extending the linear part $T\mathbf{y} \leq \mathbf{s}$ to follow a branch-and-bound decision or some robustness.

Property 1. We will see that the projection $X \to D$ can be calculated more rapidly if D belongs to the image of X. This means that each column (and row) of D can be written as a linear combination of the columns (or rows, resp.) of X. We can equivalently say that the null space of X is included in the null space of D; thus, $X\mathbf{d} = 0 \implies D\mathbf{d} = 0 \; \forall \mathbf{d} \in \mathbb{R}^n$. We will show below in cases A) and B) how it is easier to project when this property holds; if possible, `Projective Cutting-Planes` should thus adapt its own evolution to seek this property.

A) This case is characterized by $X \succ \mathbf{0}$, *i.e.*, X is non-singular; Prop 1 surely holds because the image of a non-singular X is $\mathbb{R}^n$. We apply the Cholesky decomposition to determine the unique non-singular K such that $X = KK^\top$. We then solve $D = KD'K^\top$ in variables D' by back substitution; this may require $O(n^3)$ in theory, but `Matlab` is able to compute it much more rapidly in practice because K is triangular. Let us re-write (3) as:

$$\max \left\{ t : KI_nK^\top + tKD'K^\top \succeq \mathbf{0} \right\}. \tag{4}$$

This is equivalent (by congruence according to Prop 2) to

$$\max \left\{ t : I_n + tD' \succeq \mathbf{0} \right\}. \tag{5}$$

The sought step length is $t^* = -\frac{1}{\lambda_{\min}(D')}$, or $t^* = \infty$ if $\lambda_{\min}(D') \geq 0$.

B) In this case Prop 1 is still satisfied, but X has rank $c < n$. This means X contains c independent rows (and columns by symmetry), referred to as *core* rows (or columns); the other dependent rows (or columns) are *non-core* positions. Using the LDL decomposition of X, we will factorize $X = K_{nc}K_{nc}^\top$, where $K_{nc} \in \mathbb{R}^{n\times c}$. The image of K_{nc} is equal to the image of X. Since Prop 1 is satisfied, we will see we can still solve $D = K_{nc}D'K_{nc}^\top$ in variables D'. A first intuition is to notice that we can project $X \to D$ only over the core rows and columns, because the non-core positions are dependent on the core ones.