Semidefinite programming via Projective Cutting-Planes for dense (easily-feasible) instances

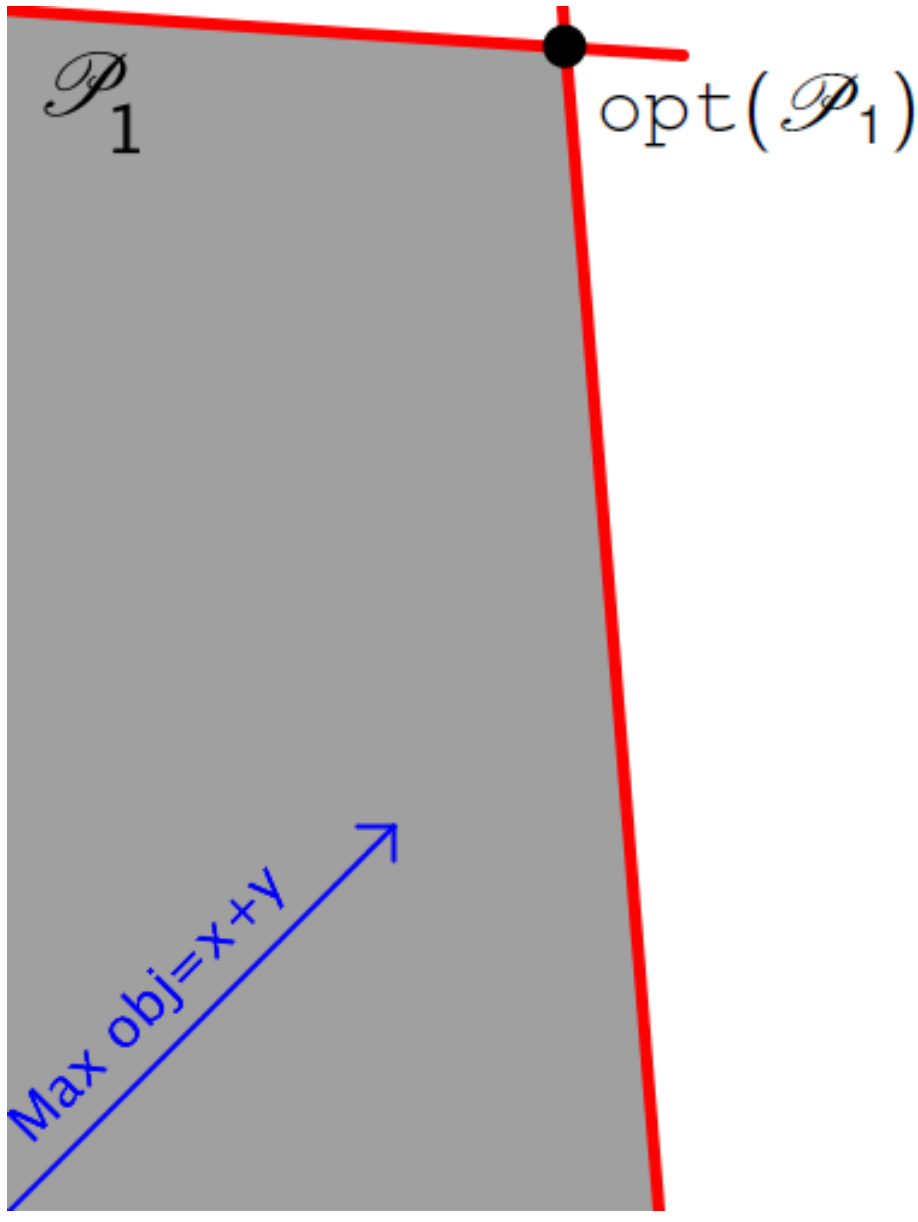
Input : a polytope $\mathscr{P}$ with prohibitively-many constraints

`Cutting-Planes` separates the outer solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at each iteration `it` $\Longrightarrow$ outer approximations that shrink :

$$\mathscr{P}_1 \supset \mathscr{P}_2 \supset \mathscr{P}_3 \supset \mathscr{P}_4 \supset \ldots$$

Goal : "upgrade" the separation sub-problem to a projection one

- Keep the sequence of outer optimal solutions $\texttt{opt}(\mathscr{P}_1) \geq \texttt{opt}(\mathscr{P}_2) \geq \texttt{opt}(\mathscr{P}_3) \geq \ldots \texttt{opt}(\mathscr{P})$
- The projection also generates interior points in $\mathscr{P}$



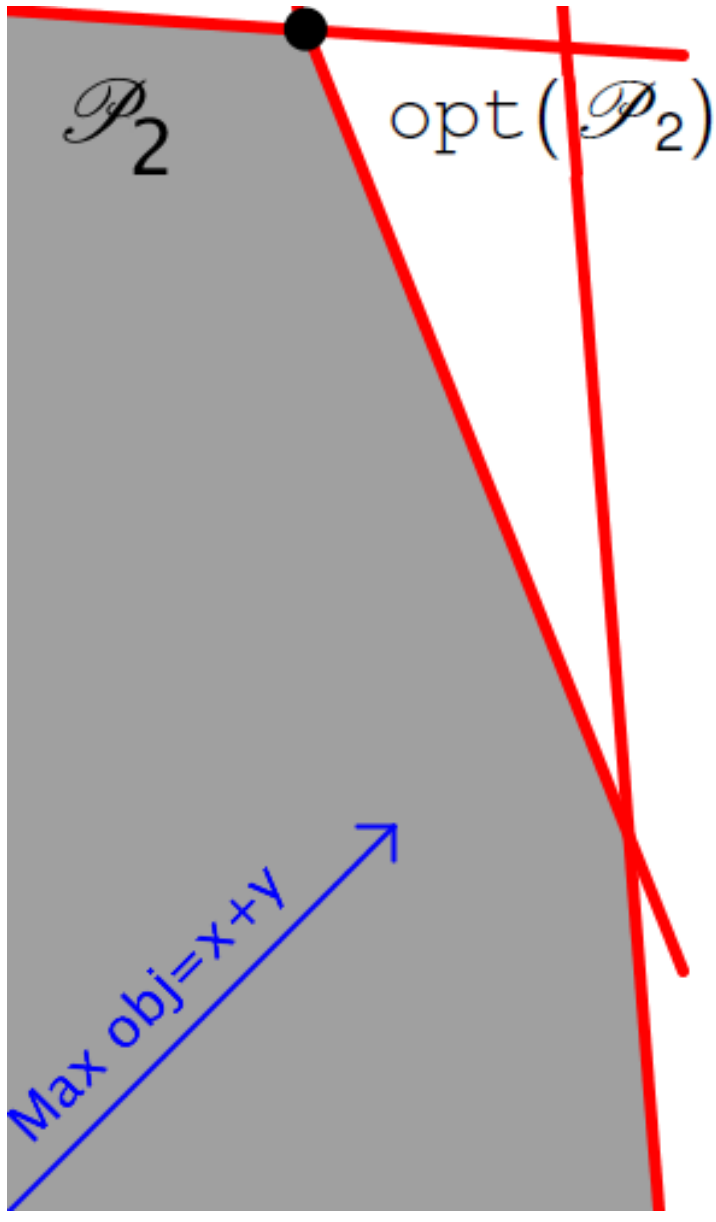
Input : a polytope $\mathscr{P}$ with prohibitively-many constraints

`Cutting-Planes` separates the outer solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at each iteration `it` $\Longrightarrow$ outer approximations that shrink :

$$\mathscr{P}_1 \supset \mathscr{P}_2 \supset \mathscr{P}_3 \supset \mathscr{P}_4 \supset \dots$$

Goal : "upgrade" the separation sub-problem to a projection one

- Keep the sequence of outer optimal solutions $\texttt{opt}(\mathscr{P}_1) \geq \texttt{opt}(\mathscr{P}_2) \geq \texttt{opt}(\mathscr{P}_3) \geq \dots \texttt{opt}(\mathscr{P})$
- The projection also generates interior points in $\mathscr{P}$



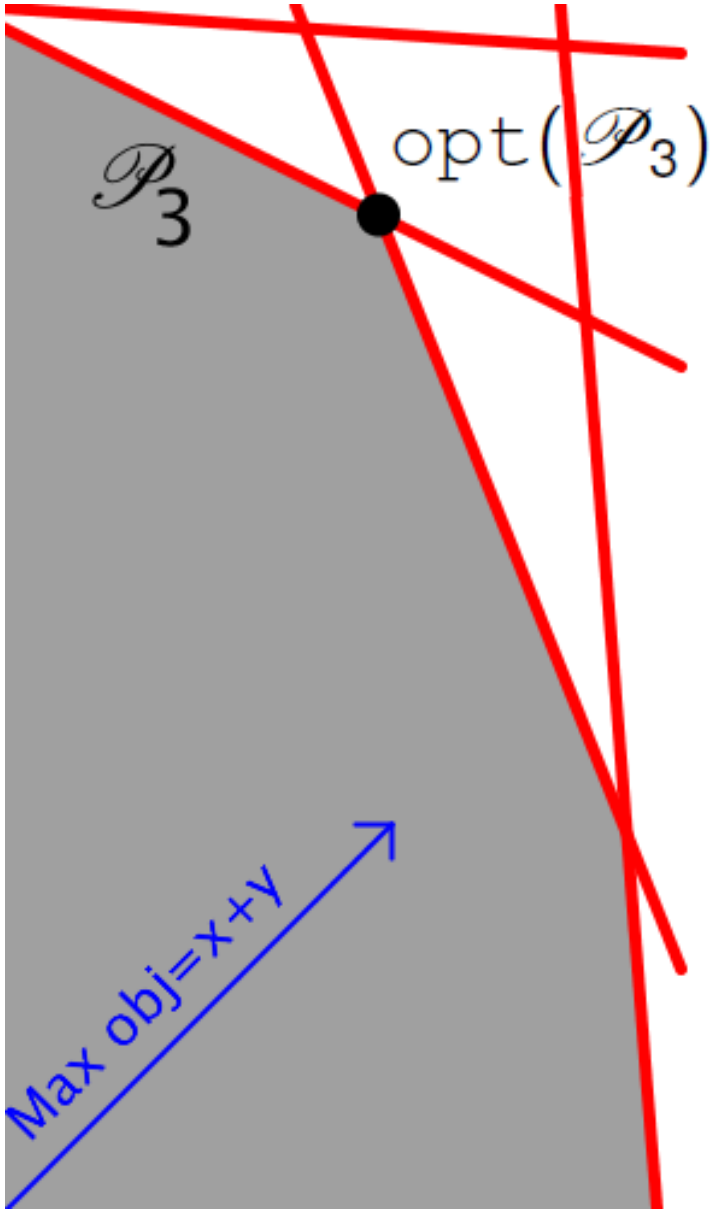
Input : a polytope $\mathscr{P}$ with prohibitively-many constraints

`Cutting-Planes` separates the outer solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at each iteration `it` $\Longrightarrow$ outer approximations that shrink :

$$\mathscr{P}_1 \supset \mathscr{P}_2 \supset \mathscr{P}_3 \supset \mathscr{P}_4 \supset \dots$$

Goal : "upgrade" the separation sub-problem to a projection one

- Keep the sequence of outer optimal solutions $\texttt{opt}(\mathscr{P}_1) \geq \texttt{opt}(\mathscr{P}_2) \geq \texttt{opt}(\mathscr{P}_3) \geq \dots \texttt{opt}(\mathscr{P})$
- The projection also generates interior points in $\mathscr{P}$



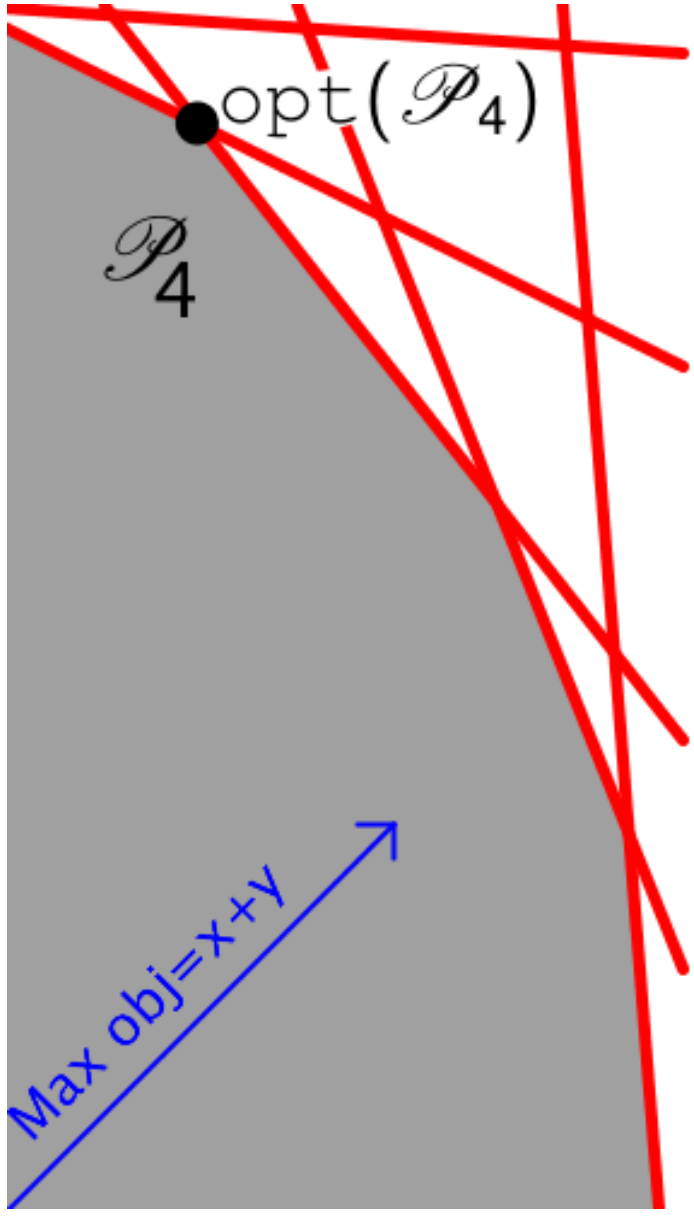
Input : a polytope $\mathscr{P}$ with prohibitively-many constraints

`Cutting-Planes` separates the outer solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at each iteration `it` $\Longrightarrow$ outer approximations that shrink :

$$\mathscr{P}_1 \supset \mathscr{P}_2 \supset \mathscr{P}_3 \supset \mathscr{P}_4 \supset \dots$$

Goal : "upgrade" the separation sub-problem to a projection one

- Keep the sequence of outer optimal solutions $\texttt{opt}(\mathscr{P}_1) \geq \texttt{opt}(\mathscr{P}_2) \geq \texttt{opt}(\mathscr{P}_3) \geq \dots \texttt{opt}(\mathscr{P})$
- The projection also generates interior points in $\mathscr{P}$



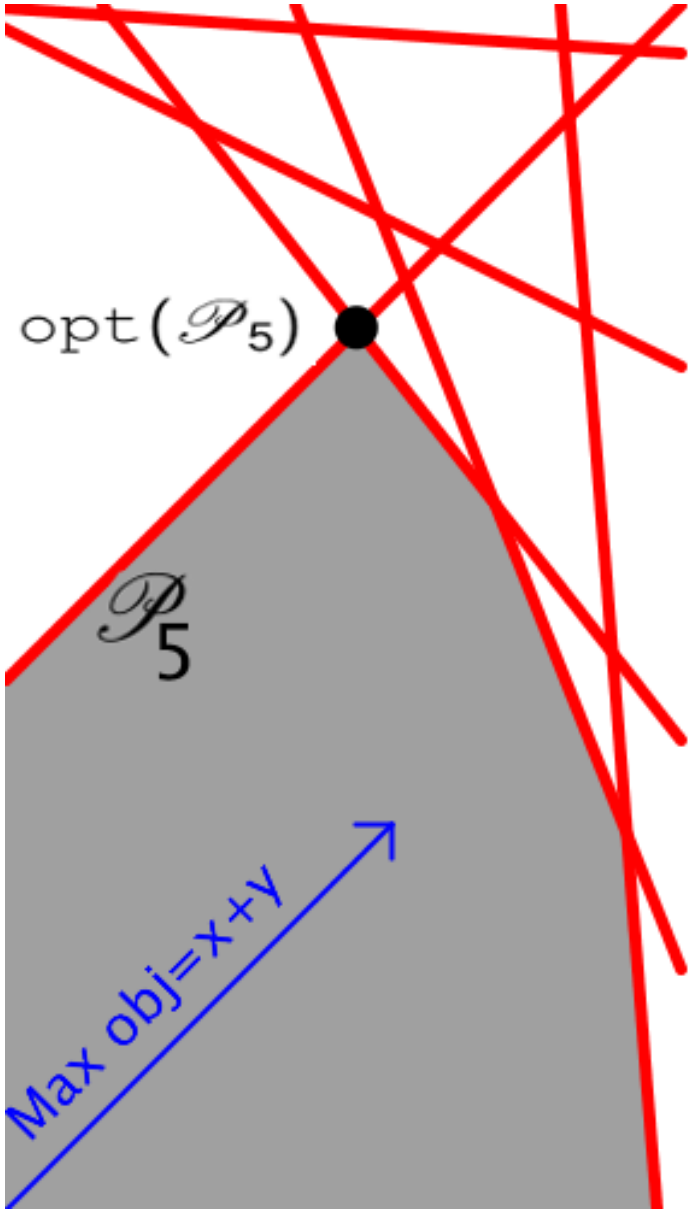
Input : a polytope $\mathscr{P}$ with prohibitively-many constraints

`Cutting-Planes` separates the outer solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at each iteration `it` $\Longrightarrow$ outer approximations that shrink :

$$\mathscr{P}_1 \supset \mathscr{P}_2 \supset \mathscr{P}_3 \supset \mathscr{P}_4 \supset \dots$$

Goal : "upgrade" the separation sub-problem to a projection one

- Keep the sequence of outer optimal solutions $\texttt{opt}(\mathscr{P}_1) \geq \texttt{opt}(\mathscr{P}_2) \geq \texttt{opt}(\mathscr{P}_3) \geq \dots \texttt{opt}(\mathscr{P})$
- The projection also generates interior points in $\mathscr{P}$



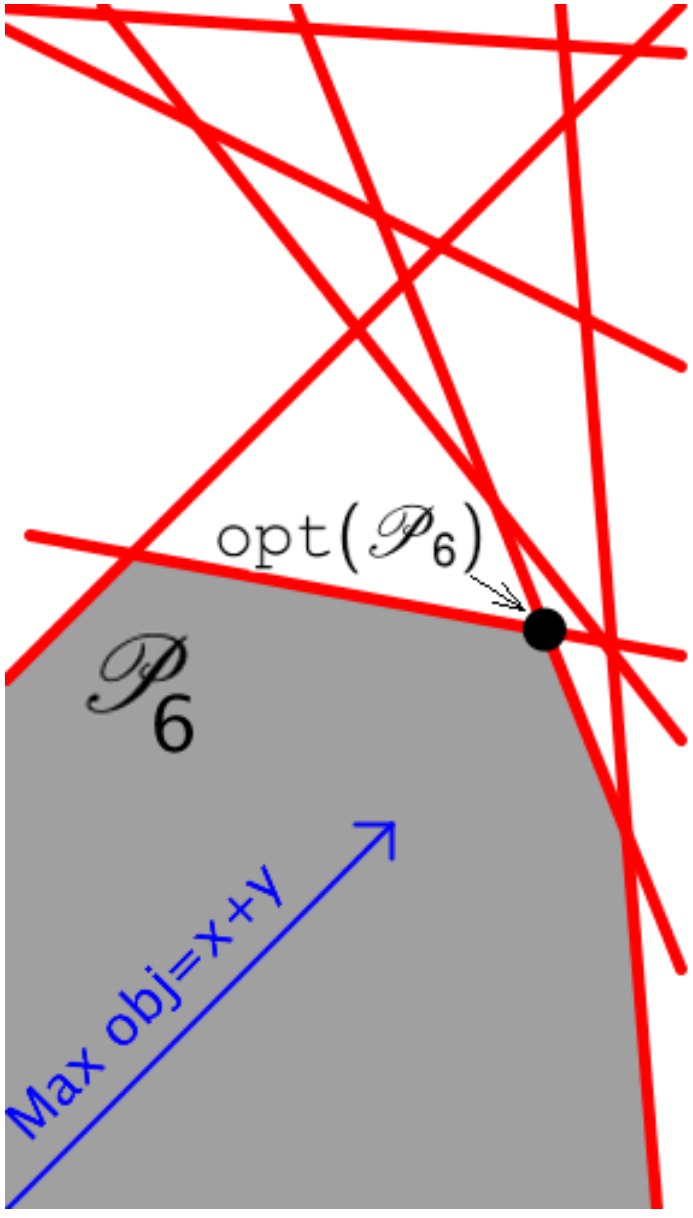
Input : a polytope $\mathscr{P}$ with prohibitively-many constraints

`Cutting-Planes` separates the outer solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at each iteration `it` $\Longrightarrow$ outer approximations that shrink :

$$\mathscr{P}_1 \supset \mathscr{P}_2 \supset \mathscr{P}_3 \supset \mathscr{P}_4 \supset \dots$$

Goal : "upgrade" the separation sub-problem to a projection one

- Keep the sequence of outer optimal solutions $\texttt{opt}(\mathscr{P}_1) \geq \texttt{opt}(\mathscr{P}_2) \geq \texttt{opt}(\mathscr{P}_3) \geq \dots \texttt{opt}(\mathscr{P})$
- The projection also generates interior points in $\mathscr{P}$



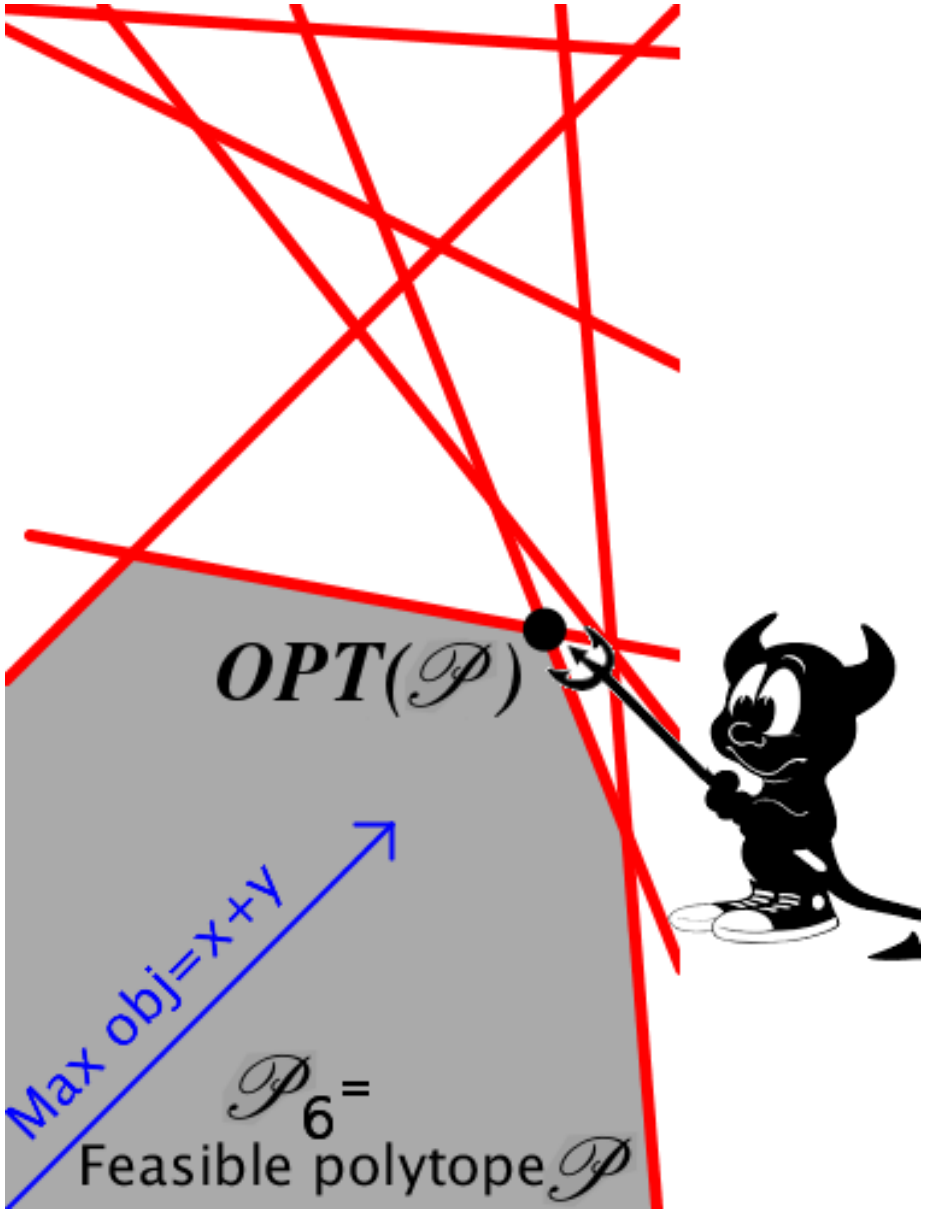
Input : a polytope $\mathscr{P}$ with prohibitively-many constraints

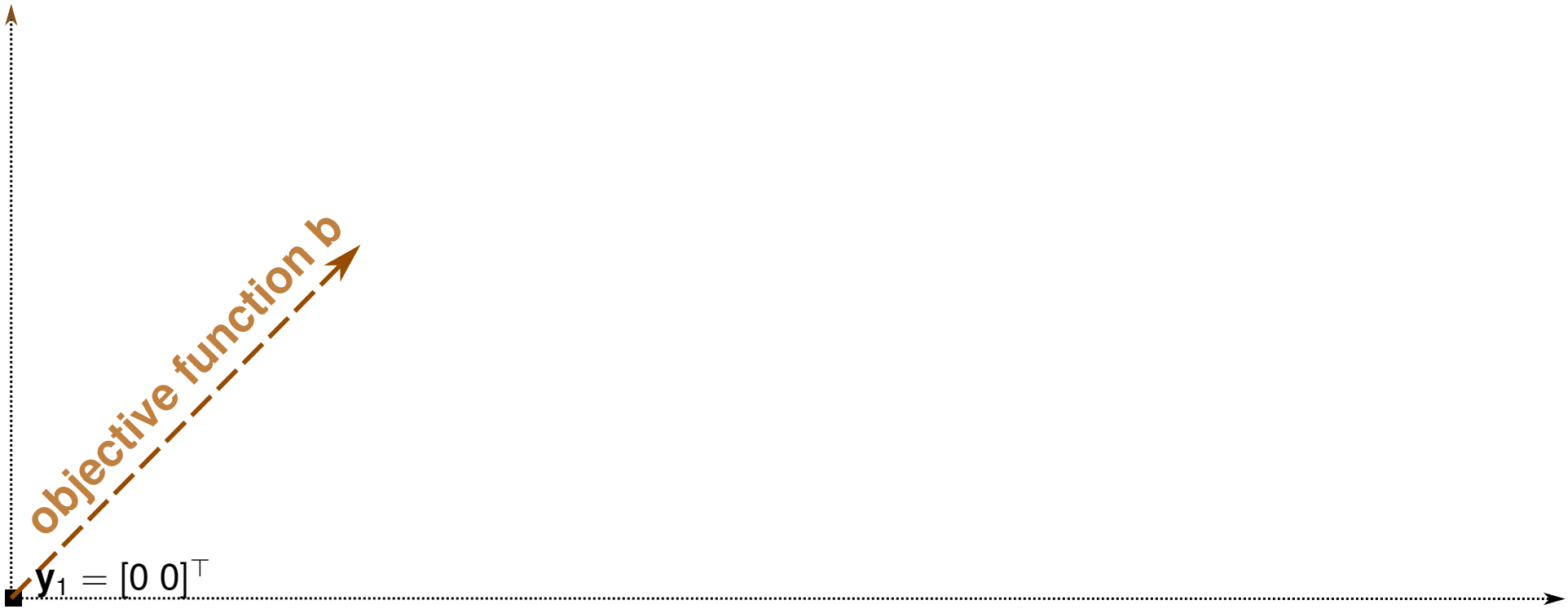
`Cutting-Planes` separates the outer solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at each iteration `it` $\Longrightarrow$ outer approximations that shrink :

$$\mathscr{P}_1 \supset \mathscr{P}_2 \supset \mathscr{P}_3 \supset \mathscr{P}_4 \supset \ldots$$

Goal : "upgrade" the separation sub-problem to a projection one

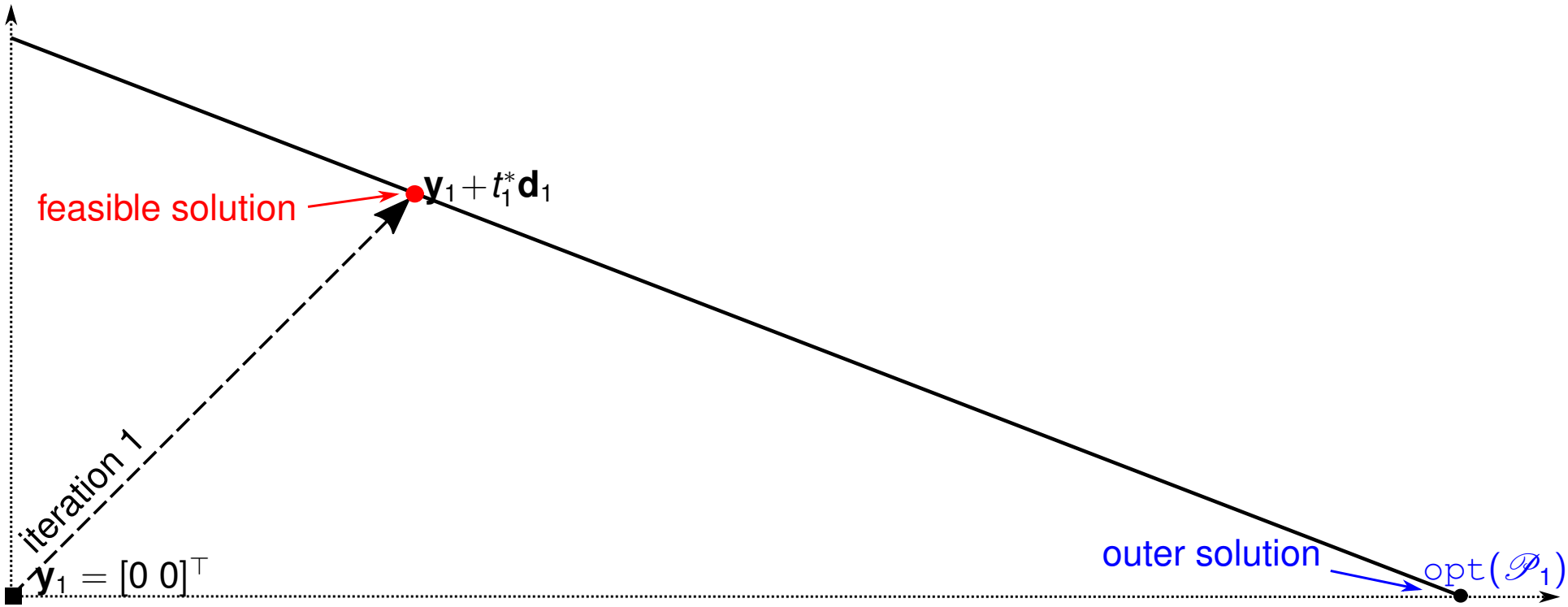
- Keep the sequence of outer optimal solutions $\texttt{opt}(\mathscr{P}_1) \geq \texttt{opt}(\mathscr{P}_2) \geq \texttt{opt}(\mathscr{P}_3) \geq \ldots \texttt{opt}(\mathscr{P})$
- The projection also generates interior points in $\mathscr{P}$





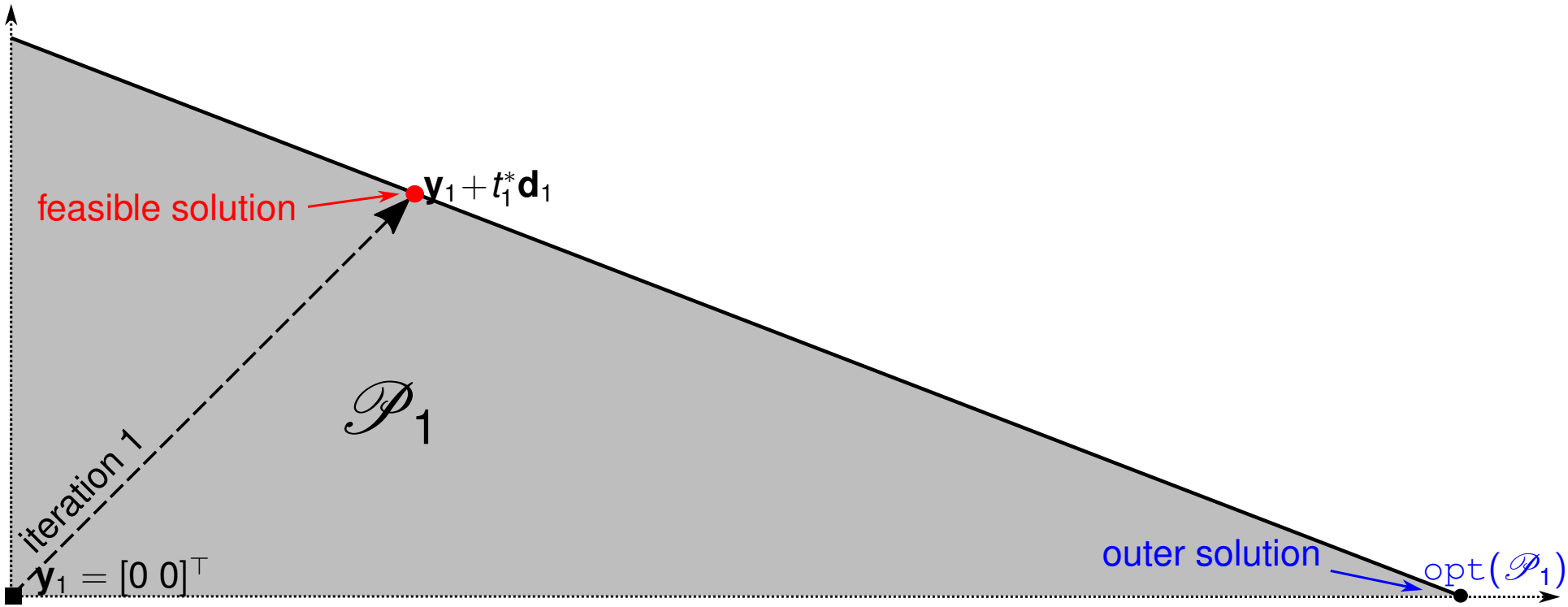
Iteration 1 : uncharted territory ; start from origin [0 0] and follow the objective function **b** :

Call this solution $\mathbf{y}_1$ $[0\ 0]^\top \rightarrow \mathbf{b}$ say this is direction $\mathbf{d}_1$



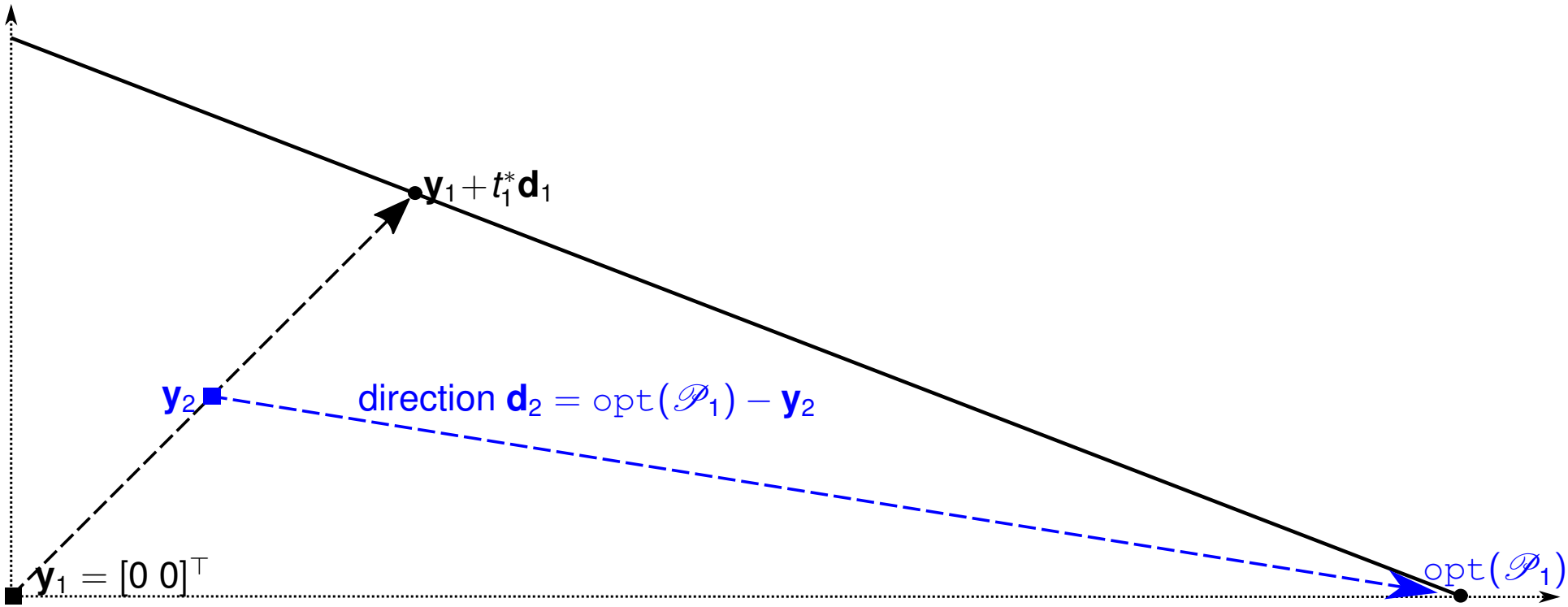
Iteration 1 : the projection sub-problem returns

- ⋆ a first-hit point $\mathbf{y}_1 + t_1^* \mathbf{d}_1$
- ⋆ a first-hit facet that . . . determines a first outer approximation $\mathscr{P}_1$; its optimum is outer solution $\mathtt{opt}(\mathscr{P}_1)$



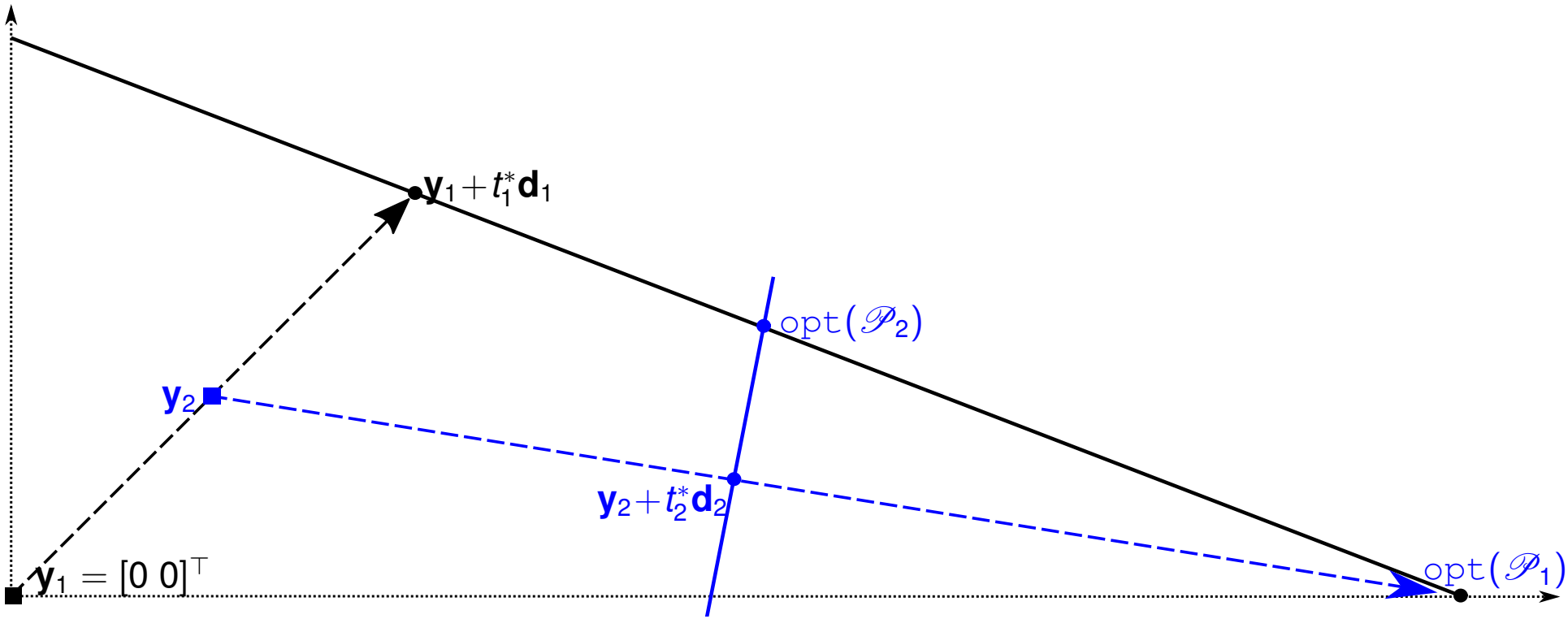
Iteration 1 : the projection sub-problem returns

- ⋆ a first-hit point $\mathbf{y}_1 + t_1^* \mathbf{d}_1$
- ⋆ a first-hit facet that determines a first outer approximation $\mathscr{P}_1$; its optimum is outer solution $\mathtt{opt}(\mathscr{P}_1)$



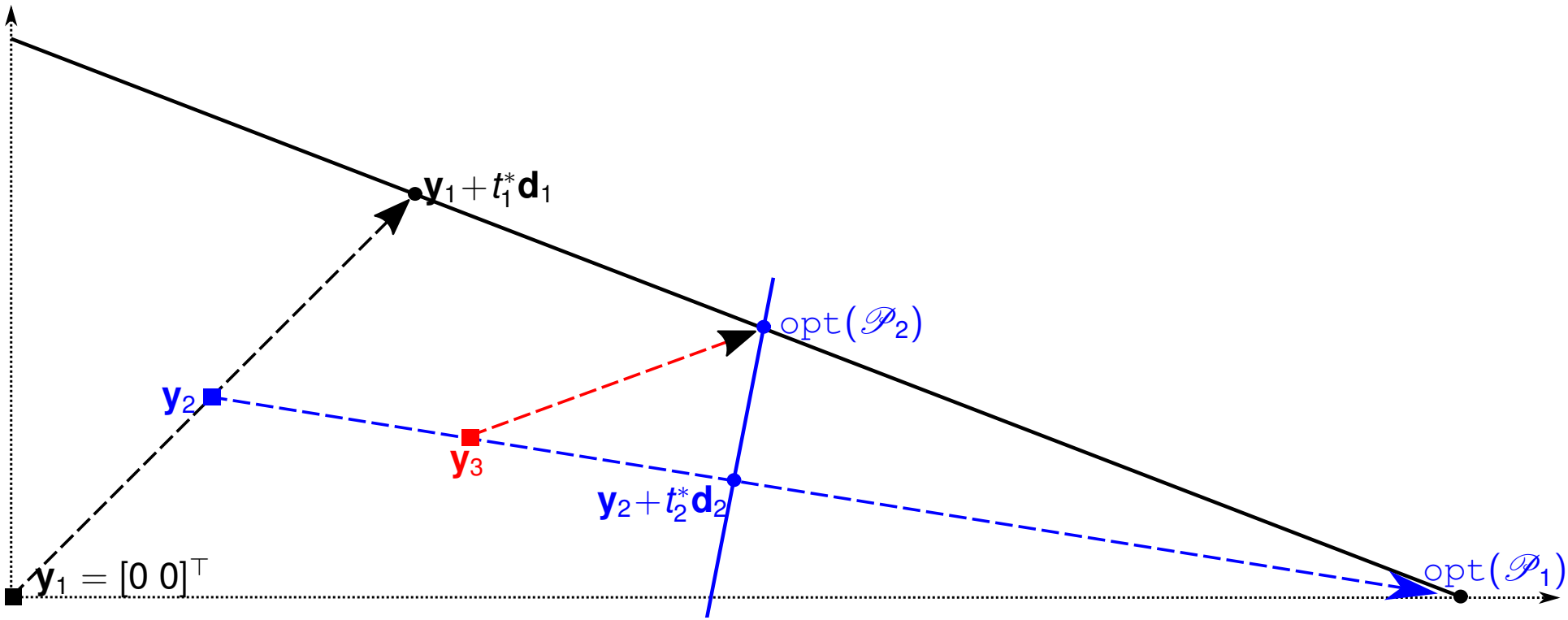
Iteration 2 :

- Choose $\mathbf{y}_2$ between $\mathbf{y}_1$ and $\mathbf{y}_1 + t_1^* \mathbf{d}_1$ and project towards $\mathbf{d}_2 = \mathtt{opt}(\mathscr{P}_1) - \mathbf{y}_2$, *i.e.*, towards the current outer solution
- The projection returns hit point $\mathbf{y}_2 + t_2^* \mathbf{d}_2$ and a new outer facet to construct $\mathscr{P}_2$

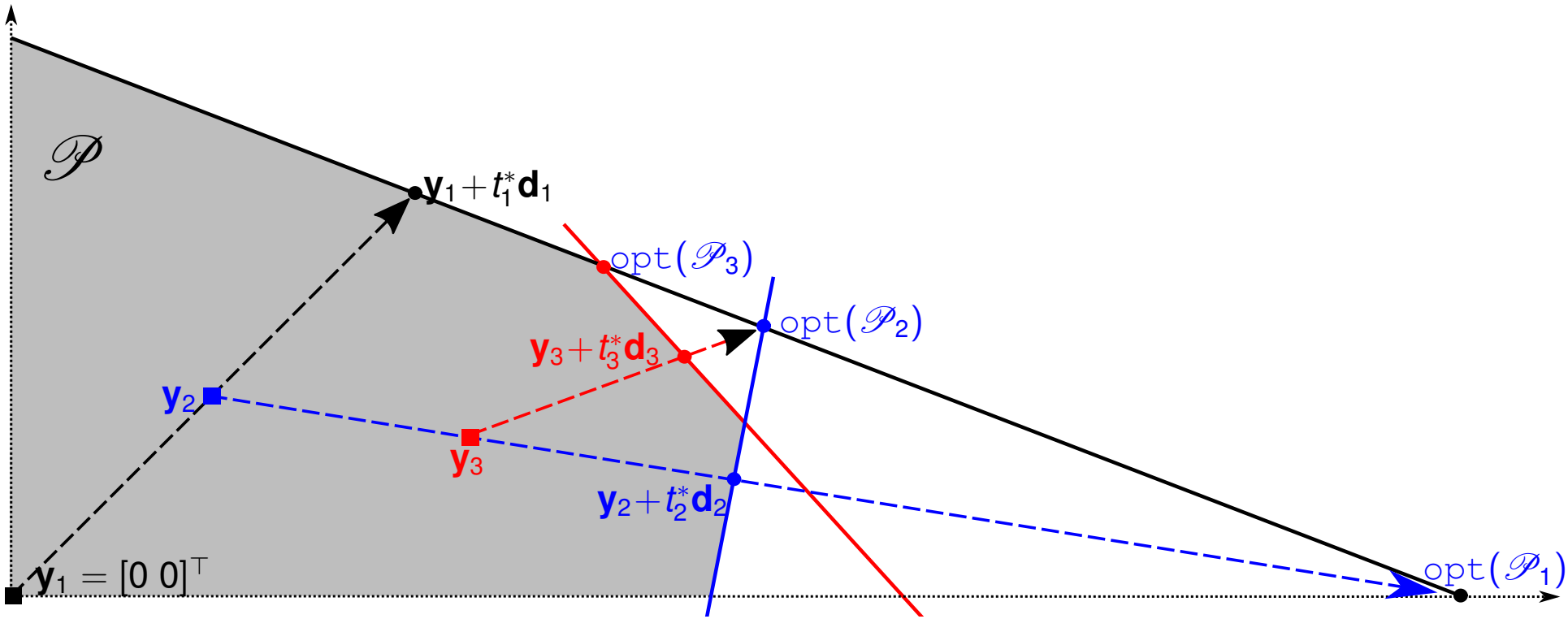


Iteration 2 :

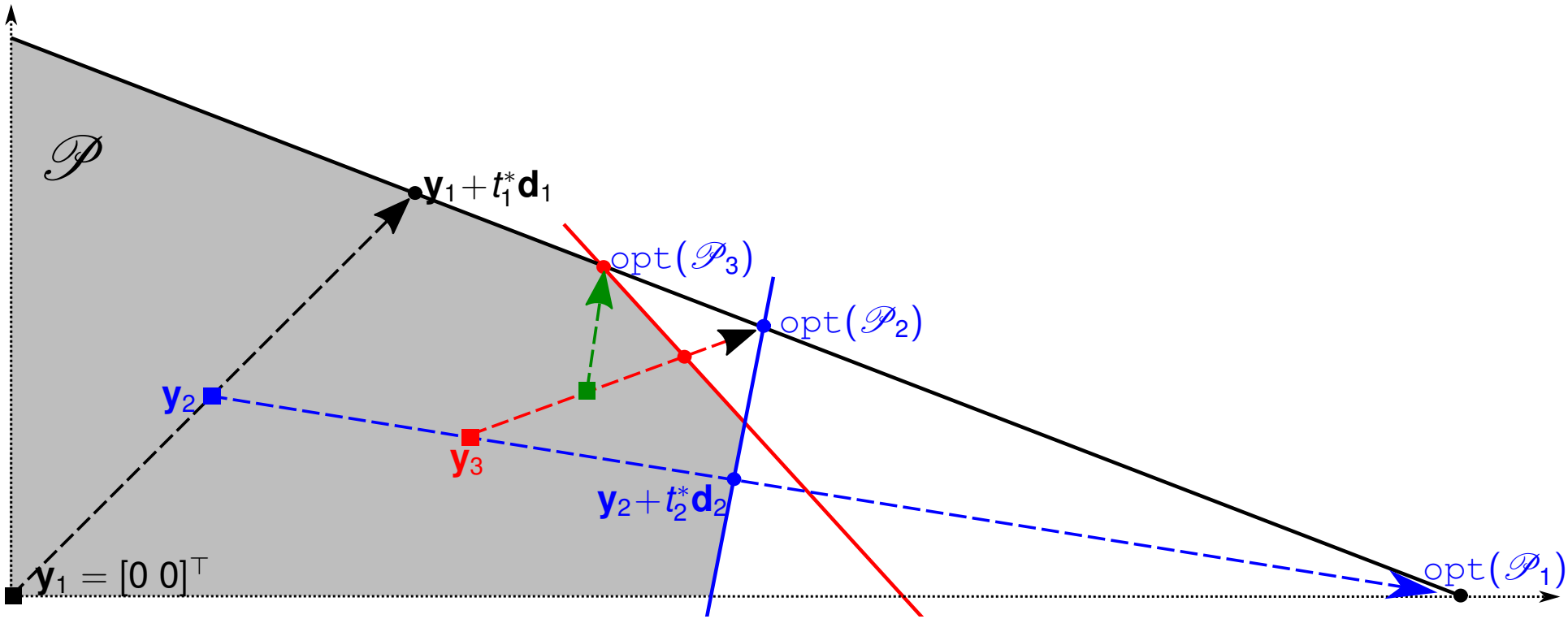
- Choose $\mathbf{y}_2$ between $\mathbf{y}_1$ and $\mathbf{y}_1 + t_1^* \mathbf{d}_1$ and project towards $\mathbf{d}_2 = \mathtt{opt}(\mathscr{P}_1) - \mathbf{y}_2$, *i.e.*, towards the current outer solution
- The projection returns hit point $\mathbf{y}_2 + t_2^* \mathbf{d}_2$ and a new outer facet to construct $\mathscr{P}_2$



Iteration 3 : Choose a new inner point $\mathbf{y}_3$ and project towards current outer optimal solution $\mathtt{opt}(\mathscr{P}_2)$

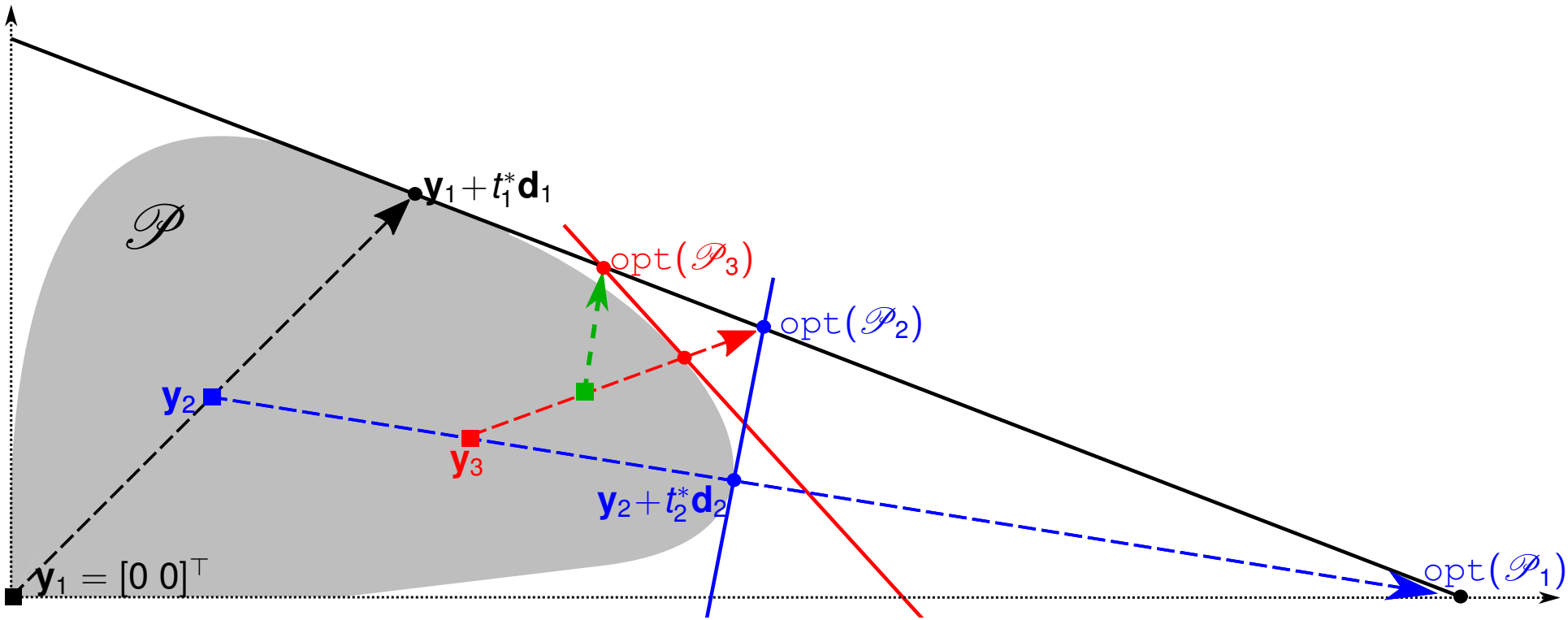


Iteration 3 : the feasible solution $\mathbf{y}_3 + t_3^* \mathbf{d}_3$ is almost optimal



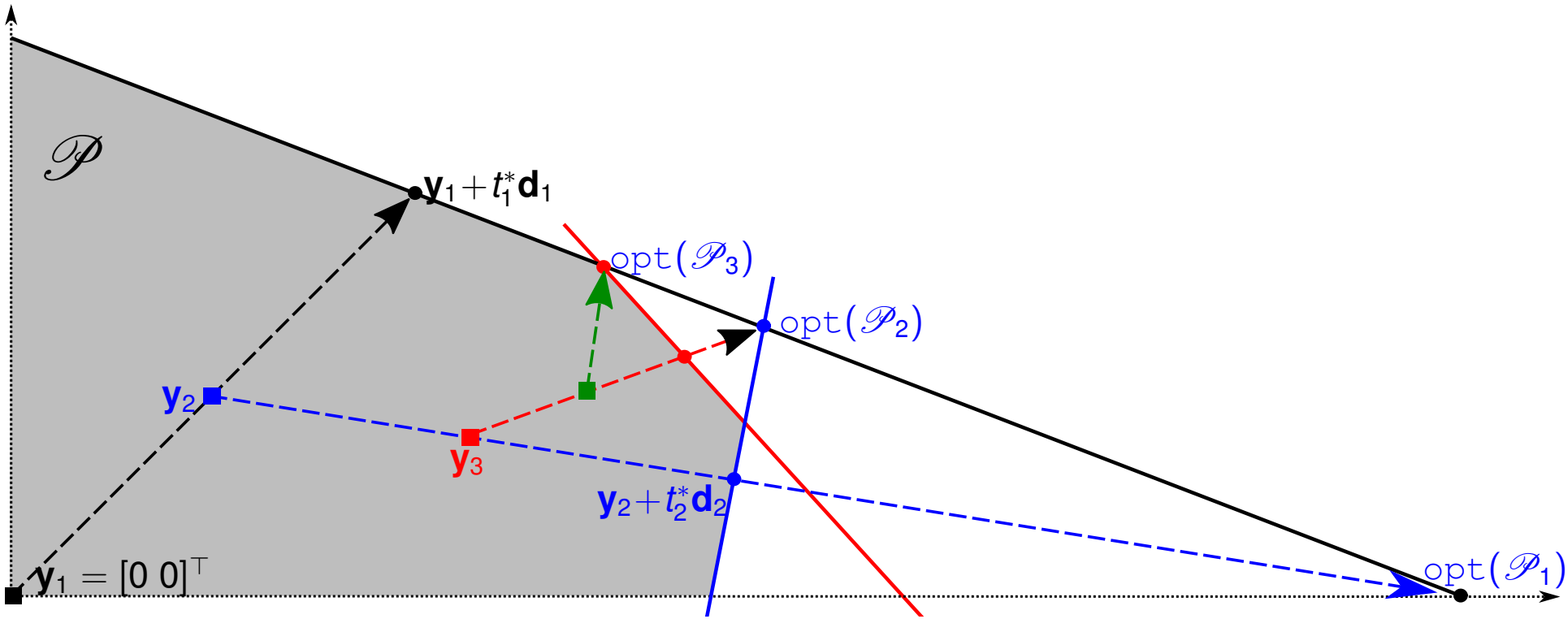
Iteration 4 : optimality of $\mathtt{opt}(\mathscr{P}_3)$ proved
You can see the proposed method is convergent because it solves a separation problem on $\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ at each iteration $\mathtt{it}$

- The convergence proof takes two lines, cool !



Iteration 4 : optimality of $\mathtt{opt}(\mathscr{P}_3)$ proved
You can see the proposed method is convergent because it solves a separation problem on $\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ at each iteration $\mathtt{it}$

- The convergence proof takes two lines, cool !



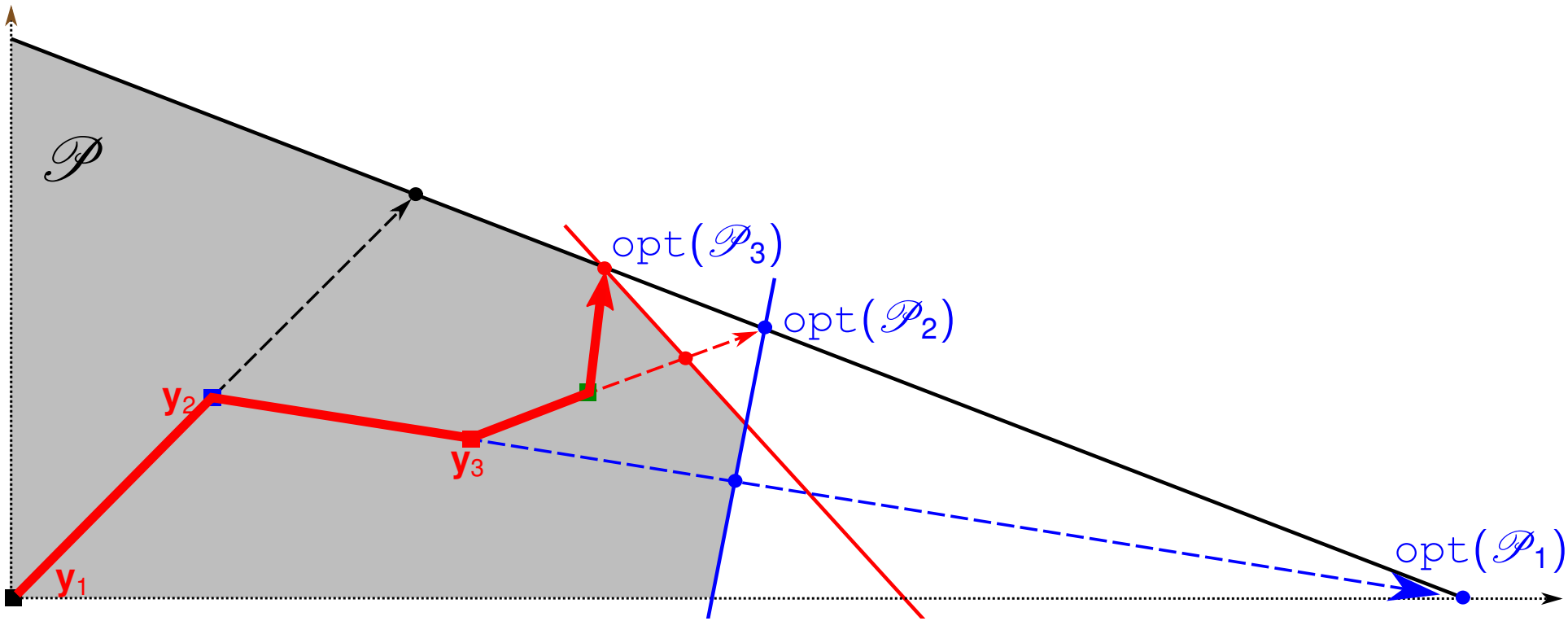
Convergence proof of 2 pages in previous work :

[Ray projection for optimizing polytopes with prohibitively many constraints in set-covering column generation. *Math Program* 2016]

The new method was designed to be more general&simpler

[Projective Cutting-Planes,
SIAM J Optim, 2020]

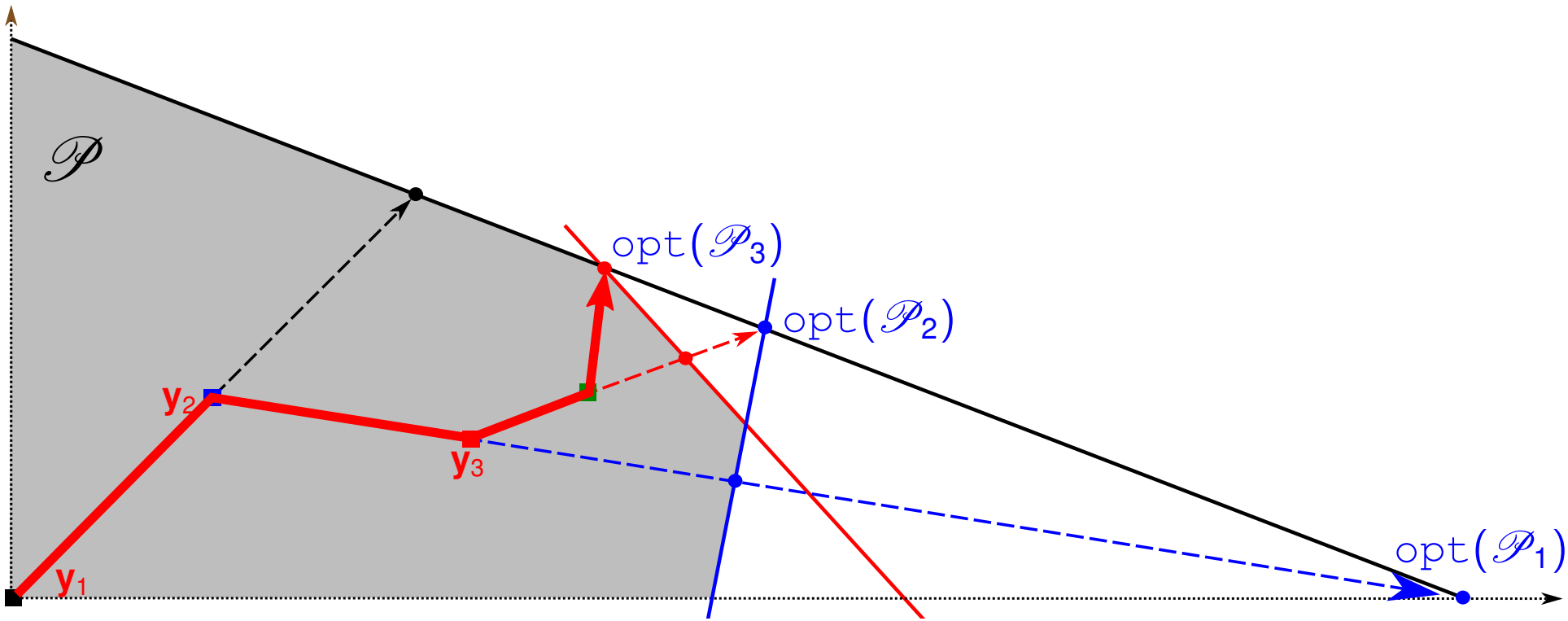
[Further experiments and insights on Projective Cutting-Planes,
INFORMS J Comput, 2022]



The trajectory of these inner points is a driving-force of the algorithm, which does not exist in related `Cutting-Planes`

---

everything was like a movie until here : let's see the difficult part



The trajectory of these inner points is a driving-force of the algorithm, which does not exist in related `Cutting-Planes`

---

everything was like a movie until here : let's see the difficult part

Standard SDP problem below in scalar variables $y_1, y_2, \ldots y_k$, uses $k+1$ symmetric $n \times n$ matrices : $A_1, A_2, \ldots A_k, C$

$$\max_{\mathbf{y}} \ \mathbf{b}^\top \mathbf{y} = b_1 y_1 + b_2 y_2 + \ldots b_k y_k$$

$$\mathscr{P} \begin{cases} X = C - \sum_{i=1}^{k} A_i y_i \text{ is SDP} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \mathbf{a}^\top \mathbf{y} \leq c_a \ \forall (\mathbf{a}, c_a) \in \texttt{Constr} \quad (\mathbf{y} \text{ also lives in a polytope}) \end{cases}$$

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

Standard SDP problem below in scalar variables $y_1, y_2, \ldots y_k$, uses $k+1$ symmetric $n \times n$ matrices : $A_1, A_2, \ldots A_k, C$

$$\max_{\mathbf{y}} \quad \mathbf{b}^\top \mathbf{y} = b_1 y_1 + b_2 y_2 + \ldots b_k y_k$$

$$\mathscr{P} \begin{cases} X = C - \sum_{i=1}^{k} A_i y_i \succeq \mathbf{0} \\ X \succeq \mathbf{0} \iff X \bullet \mathbf{v}\mathbf{v}^\top \geq 0 \ \forall \mathbf{v} \in \mathbb{R}^n \\ \mathbf{a}^\top \mathbf{y} \leq c_a \ \forall (\mathbf{a}, c_a) \in \texttt{Constr} \quad (\mathbf{y} \text{ also lives in a polytope}) \end{cases}$$

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

SDP constraint below 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 $X = C - \sum_{i=1}^{k} A_i y_i$ and $X \bullet \mathbf{v}\mathbf{v}^\top = \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} v_i v_j$

This SDP constraint in gray is satisfied if and only if

SDP constraint below 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 $X = C - \sum_{i=1}^{k} A_i y_i$ and $X \bullet \mathbf{v}\mathbf{v}^\top = \sum_{i=1}^{n}\sum_{j=1}^{n} X_{ij} v_i v_j$

This SDP constraint in gray is satisfied if and only if

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

$$\iff$$

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

SDP constraint below 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 $X = C - \sum_{i=1}^{k} A_i y_i$ and $X \bullet \mathbf{v}\mathbf{v}^\top = \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} v_i v_j$

This SDP constraint in gray is satisfied if and only if this eingencut holds

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

$$\max_{\mathbf{y}} \ \mathbf{b}^\top \mathbf{y} = b_1 y_1 + b_2 y_2 + \dots b_k y_k$$

$$\mathscr{P} \begin{cases} s.t \ \sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top \right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \mathbb{R}^n \\ \mathbf{a}^\top \mathbf{y} \leq c_a \ \forall (\mathbf{a}, c_a) \in \texttt{Constr} \quad (\mathbf{y} \text{ also lives in a polytope}) \end{cases}$$

We could address the problem by separating at each iteration an infeasible $\mathbf{y}^{\text{out}} \in \mathbb{R}^k$; we separate such $\mathbf{y}^{\text{out}}$ by finding one $\mathbf{v}$ associated to the minimum eigenvalue of $X^{\text{out}} = C - \sum_{i=1}^{k} A_i y_i^{\text{out}}$.

This standard `Cutting-Planes` is not considered very effective ; work on this area remains a rare sight (a half-dozen of articles).

$$\max_{\mathbf{y}} \ \mathbf{b}^\top \mathbf{y} = b_1 y_1 + b_2 y_2 + \dots b_k y_k$$

$$\mathscr{P} \begin{cases} s.t \ \sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top \right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \mathbb{R}^n \\ \mathbf{a}^\top \mathbf{y} \leq c_a \ \forall (\mathbf{a}, c_a) \in \texttt{Constr} \quad (\mathbf{y} \text{ also lives in a polytope}) \end{cases}$$

We could address the problem by separating at each iteration an infeasible $\mathbf{y}^{\text{out}} \in \mathbb{R}^k$; we separate such $\mathbf{y}^{\text{out}}$ by finding one $\mathbf{v}$ associated to the minimum eigenvalue of $X^{\text{out}} = C - \sum_{i=1}^{k} A_i y_i^{\text{out}}$.

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. Let $\texttt{CUTS}_{\texttt{now}}$ be the tight saturated constraints generated at some iteration ; consider this LP :

$$\max_{\mathbf{y}} \ \mathbf{b}^\top \mathbf{y}$$

$$\alpha_{\mathbf{v}} : \ \sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top \right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \texttt{CUTS}_{\texttt{now}}$$

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

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

$$Z \succeq \mathbf{0}$$

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

Yet the cutting-planes framework has some advantages. Let $\texttt{CUTS}_{\texttt{now}}$ be the tight saturated constraints generated at some iteration ; consider this LP :

$$\max_{\mathbf{y}} \ \mathbf{b}^\top \mathbf{y}$$

$$\alpha_{\mathbf{v}} : \ \sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top \right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \texttt{CUTS}_{\texttt{now}}$$

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

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

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

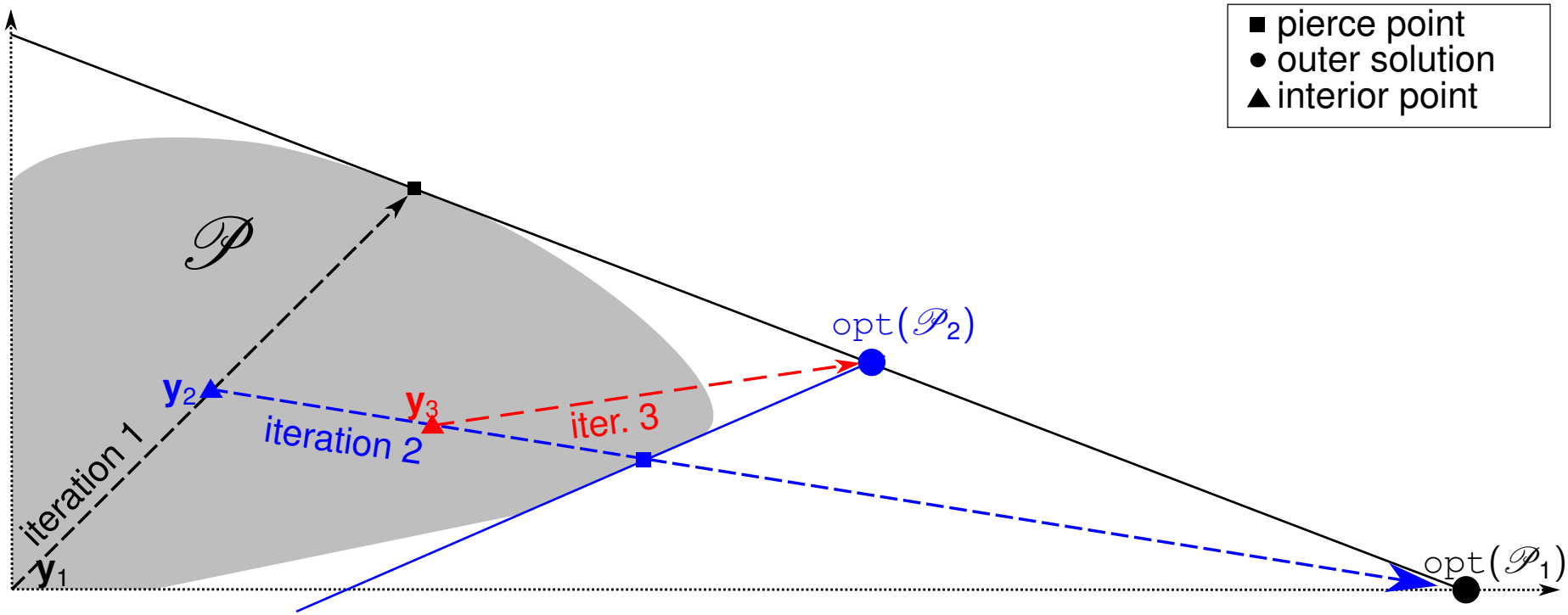
We generate at each iteration a feasible primal solution $\mathbf{y} \in \mathbb{R}^k$ and a feasible dual $Z \succeq \mathbf{0}$, going beyond Interior Point Methods

We generate at each iteration a feasible primal solution $\mathbf{y} \in \mathbb{R}^k$ and a feasible dual $Z \succeq \mathbf{0}$, going beyond Interior Point Methods

- Primal solutions : $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots \quad \oplus$ hit points
- Dual solutions : constructed as in the previous slide from the tight constraints around $\texttt{opt}(\mathscr{P}_1), \texttt{opt}(\mathscr{P}_2), \texttt{opt}(\mathscr{P}_3), \ldots$

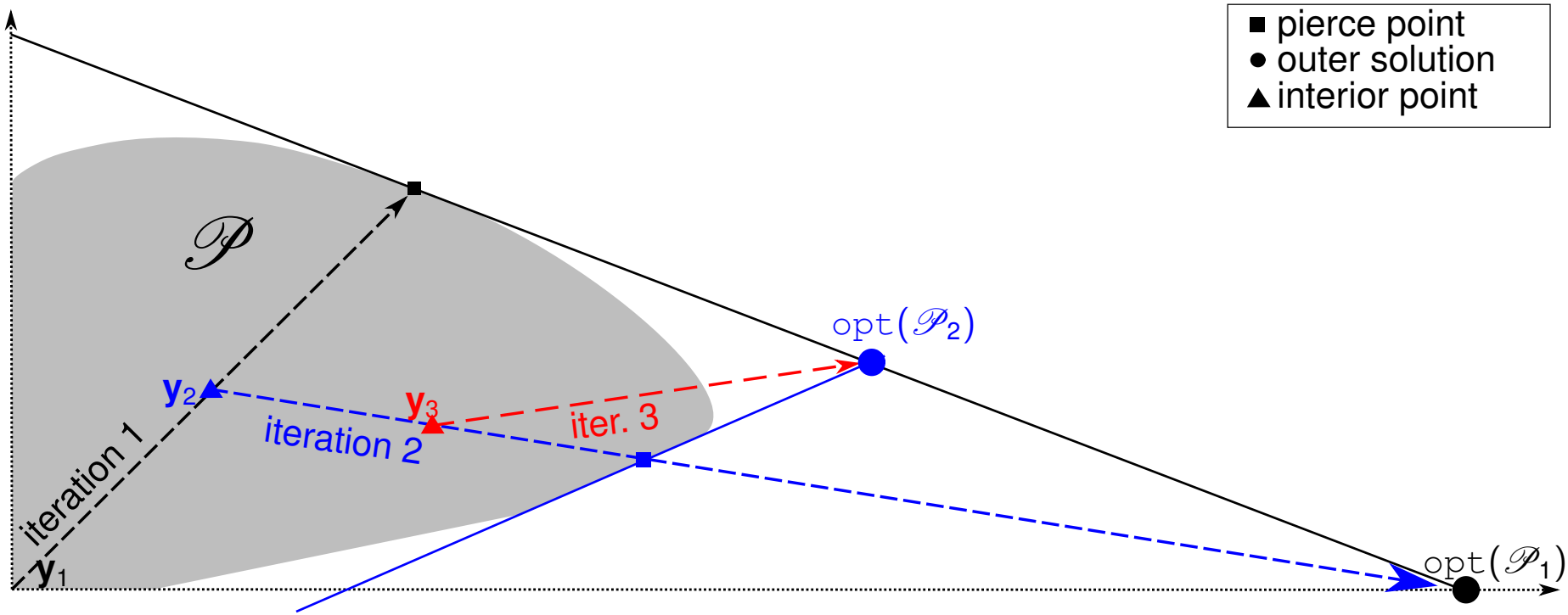
We generate at each iteration a feasible primal solution $\mathbf{y} \in \mathbb{R}^k$ and a feasible dual $Z \succeq \mathbf{0}$, going beyond Interior Point Methods

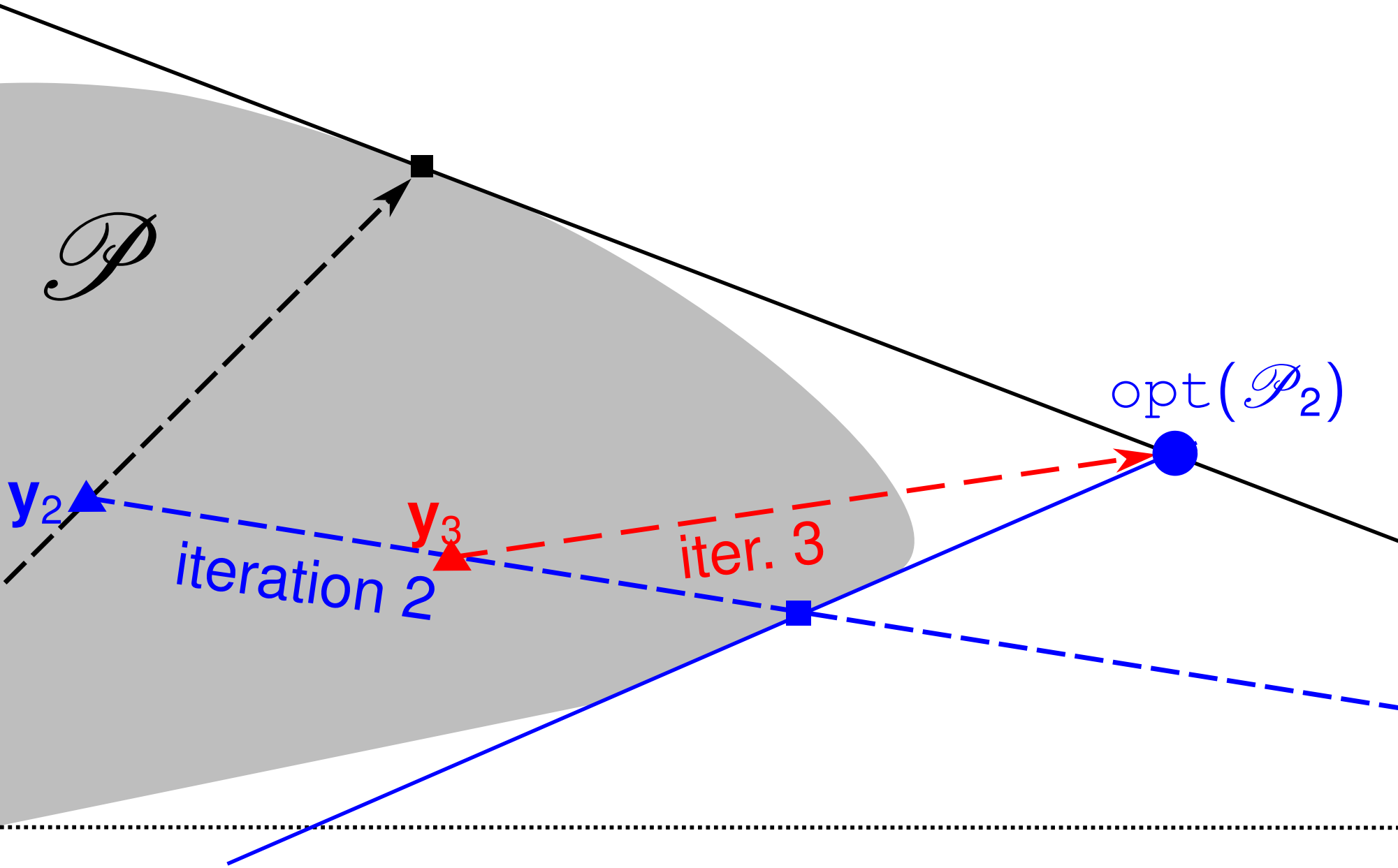
- Primal solutions : $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots$ $\oplus$ hit points
- Dual solutions : constructed as in the previous slide from the tight constraints around $\texttt{opt}(\mathscr{P}_1), \texttt{opt}(\mathscr{P}_2), \texttt{opt}(\mathscr{P}_3), \ldots$

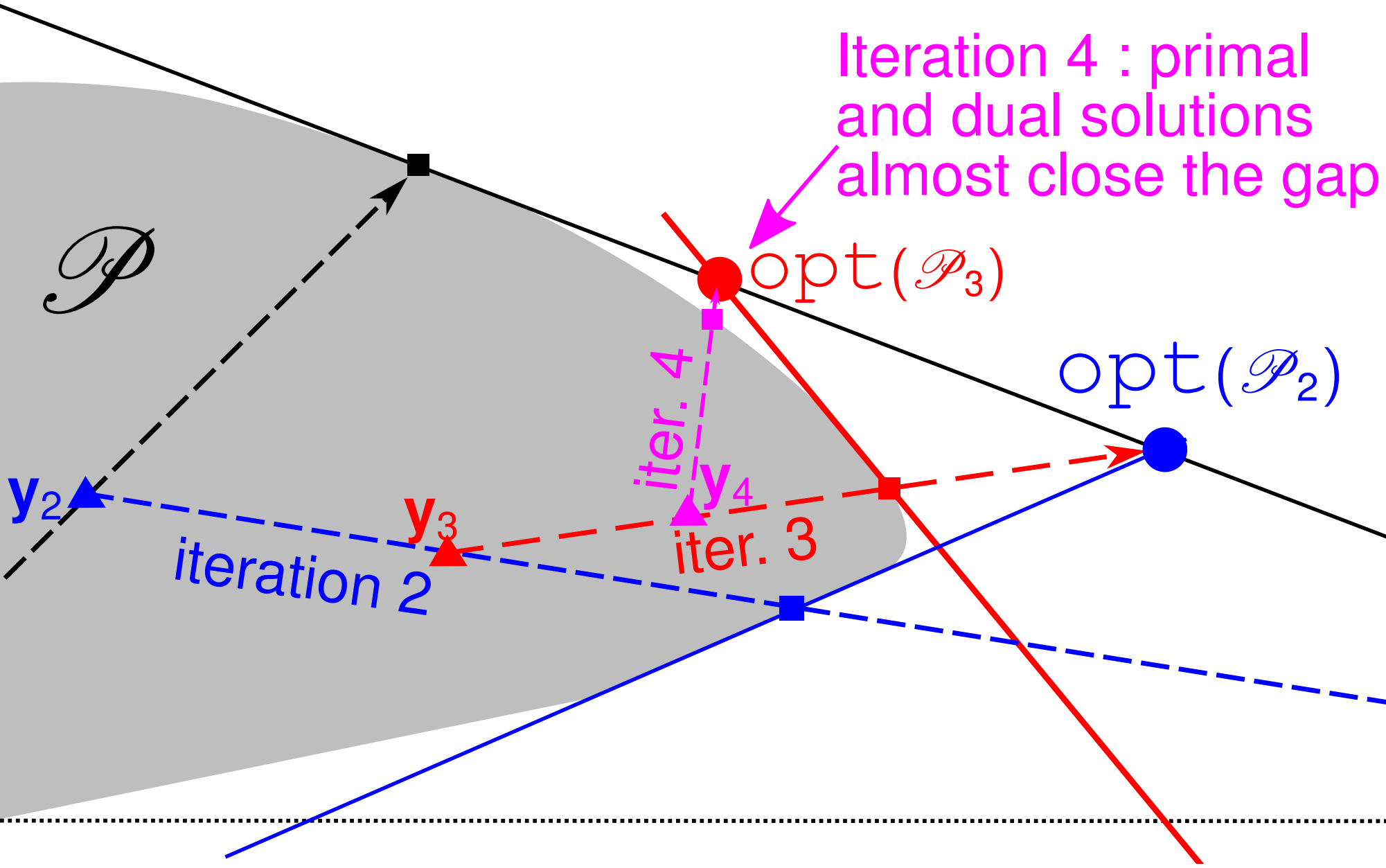


We generate at each iteration a feasible primal solution $\mathbf{y} \in \mathbb{R}^k$ and a feasible dual $Z \succeq \mathbf{0}$, going beyond Interior Point Methods

- Primal solutions : $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots$ $\oplus$ hit points
- Dual solutions : constructed as in the previous slide from the tight constraints around $\texttt{opt}(\mathscr{P}_1), \texttt{opt}(\mathscr{P}_2), \texttt{opt}(\mathscr{P}_3), \ldots$




$\mathscr{P}$
$\mathtt{opt}(\mathscr{P}_2)$
y_2
y_3
iteration 2
iter. 3


Iteration 4 : primal
and dual solutions
almost close the gap
$\mathscr{P}$
$\mathtt{opt}(\mathscr{P}_3)$
$\mathtt{opt}(\mathscr{P}_2)$
y_2
y_3
y_4
iteration 2
iter. 3
iter. 4

Solving the SDP projection

$$\max_{\mathbf{y}} \ \mathbf{b}^\top \mathbf{y}$$

$$\mathscr{P} \begin{cases} s.t \ \sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top \right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \mathbb{R}^n \\ \mathbf{a}^\top \mathbf{y} \leq c_a \ \forall (\mathbf{a}, c_a) \in \texttt{Constr} \quad (\mathbf{y} \text{ also lives in a polytope}) \end{cases}$$

Given feasible $\mathbf{y}$ and an arbitrary direction $\mathbf{d}$, what is the maximum step-length t^* so that $\mathbf{y} + t^*\mathbf{d} \in \mathscr{P}$?

We have to solve $t^* = \max\{t : X + tD \succeq \mathbf{0}\}$ for :

- $X = C - \sum_{i=1}^{k} A_i y_i$ that is SDP when $\mathbf{y}$ is feasible
- $D = C - \sum_{i=1}^{k} A_i d_i$ indefinite.

Solving the SDP projection

$$\max_{\mathbf{y}} \ \mathbf{b}^\top \mathbf{y}$$

$$\mathscr{P} \begin{cases} s.t \ \sum_{i=1}^{k} \left(A_i \bullet \mathbf{v}\mathbf{v}^\top \right) y_i \leq C \bullet \mathbf{v}\mathbf{v}^\top, \quad \forall \mathbf{v} \in \mathbb{R}^n \\ \mathbf{a}^\top \mathbf{y} \leq c_a \ \forall (\mathbf{a}, c_a) \in \texttt{Constr} \quad (\mathbf{y} \text{ also lives in a polytope}) \end{cases}$$

Given feasible $\mathbf{y}$ and an arbitrary direction $\mathbf{d}$, what is the maximum step-length t^* so that $\mathbf{y} + t^*\mathbf{d} \in \mathscr{P}$?
We have to solve $t^* = \max\{t : X + tD \succeq \mathbf{0}\}$ for :

- $X = C - \sum_{i=1}^{k} A_i y_i$ that is SDP when $\mathbf{y}$ is feasible
- $D = C - \sum_{i=1}^{k} A_i d_i$ indefinite.

Projecting $X \to D$ in the SDP cone :

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

This is easy if $X \succ \mathbf{0}$. In this case, there is a unique Cholesky decomposition $X = KK^\top$ and K is non-singular and triangular.

- Solve $D = KD'K^\top$ in variables D'

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

- Determine $\max\{t : I_n + tD' \succeq \mathbf{0}\}$ by computing $\lambda_{\min}(D')$

Total projection cost :

- one Cholesky and a few back-substitutions $O(n^2)$
- one minimum eigenvalue $O(n^3)$
- a fast calculation of products $\mathbf{v}\mathbf{v}^\top$ $O(n^2)$
- $k+1$ dot products of the form $A_i \bullet \mathbf{v}\mathbf{v}^\top$ $O(kn^2)$

Projecting $X \to D$ in the SDP cone :

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

This is easy if $X \succ \mathbf{0}$. In this case, there is a unique Cholesky decomposition $X = KK^\top$ and K is non-singular and triangular.

- Solve $D = KD'K^\top$ in variables D'

A few back-substitutions cause K is triangular

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

$$t^* = \max\{t : KK^\top + tKD'K^\top \succeq \mathbf{0}\}$$

$$t^* = \max\{t : I_n + tD' \succeq \mathbf{0}\}$$

- Determine $\max\{t : I_n + tD' \succeq \mathbf{0}\}$ by computing $\lambda_{\min}(D')$

Total projection cost :

- one Cholesky and a few back-substitutions $O(n^2)$
- one minimum eigenvalue $O(n^3)$
- a fast calculation of products $\mathbf{v}\mathbf{v}^\top$ $O(n^2)$
- $k+1$ dot products of the form $A_i \bullet \mathbf{v}\mathbf{v}^\top$ $O(kn^2)$

Projecting $X \to D$ in the SDP cone :

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

This is easy if $X \succ \mathbf{0}$. In this case, there is a unique Cholesky decomposition $X = KK^\top$ and K is non-singular and triangular.

- Solve $D = KD'K^\top$ in variables D'
- The following are equivalent :

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

- Determine $\max\{t : I_n + tD' \succeq \mathbf{0}\}$ by computing $\lambda_{\min}(D')$

Total projection cost :

- one Cholesky and a few back-substitutions $O(n^2)$
- one minimum eigenvalue $O(n^3)$
- a fast calculation of products $\mathbf{v}\mathbf{v}^\top$ $O(n^2)$
- $k+1$ dot products of the form $A_i \bullet \mathbf{v}\mathbf{v}^\top$ $O(kn^2)$

Projecting $X \to D$ in the SDP cone :

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

This is easy if $X \succ \mathbf{0}$. In this case, there is a unique Cholesky decomposition $X = KK^\top$ and K is non-singular and triangular.

- Solve $D = KD'K^\top$ in variables D'
- The following are equivalent :

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

- Determine $\max\{t : I_n + tD' \succeq \mathbf{0}\}$ by computing $\lambda_{\min}(D')$

Total projection cost :

- one Cholesky and a few back-substitutions $O(n^2)$
- one minimum eigenvalue $O(n^3)$
- a fast calculation of products $\mathbf{v}\mathbf{v}^\top$ $O(n^2)$
- $k+1$ dot products of the form $A_i \bullet \mathbf{v}\mathbf{v}^\top$ $O(kn^2)$

Projecting $X \to D$ in the SDP cone :

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

This is easy if $X \succ \mathbf{0}$. In this case, there is a unique Cholesky decomposition $X = KK^\top$ and K is non-singular and triangular.

- Solve $D = KD'K^\top$ in variables D'
- The following are equivalent :

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

- Determine $\max\{t : I_n + tD' \succeq \mathbf{0}\}$ by computing $\lambda_{\min}(D')$

Total projection cost :

- one Cholesky and a few back-substitutions $O(n^2)$
- one minimum eigenvalue $O(n^3)$
- a fast calculation of products $\mathbf{v}\mathbf{v}^\top$ $O(n^2)$
- $k+1$ dot products of the form $A_i \bullet \mathbf{v}\mathbf{v}^\top$ $O(kn^2)$

Projecting $X \to D$ in the SDP cone :

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

This is easy if $X \succ \mathbf{0}$. In this case, there is a unique Cholesky decomposition $X = KK^\top$ and K is non-singular and triangular.

- Solve $D = KD'K^\top$ in variables D'

> An Interior Point Method needs to compute at each iteration a Schur matrix that costs $O(k^2n^2 + kn^3)$, but involves operations much more complex than the below ones.

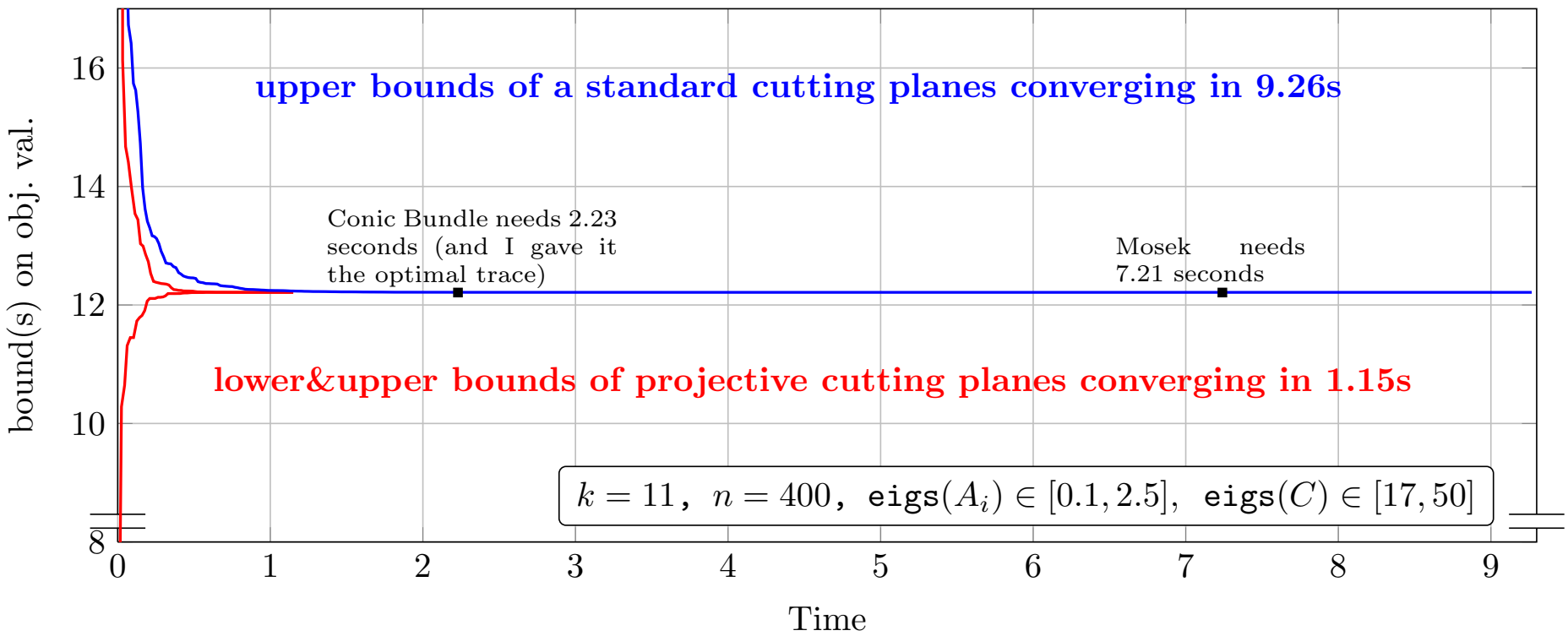
- Determine $\max\{t : I_n + tD' \succeq \mathbf{0}\}$ by computing $\lambda_{\min}(D')$

Total projection cost :

- one Cholesky and a few back-substitutions $O(n^2)$
- one minimum eigenvalue $O(n^3)$
- a fast calculation of products $\mathbf{v}\mathbf{v}^\top$ $O(n^2)$
- $k+1$ dot products of the form $A_i \bullet \mathbf{v}\mathbf{v}^\top$ $O(kn^2)$

This particular case was enough to solve an instance in which all matrices have strictly positive eigenvalues.

- All generated interior points strictly SDP, starting at ϵI_n



Projecting from X of rank $c < n$

1. The LDL decomposition gives $X = \sum_{i=1}^{c} p_i L_i L_i^\top = K_{nc} K_{nc}^\top$ where K_{nc} has n rows and c columns; L_i is a column vector
2. If system $D = K_{nc} D' K_{nc}^\top$ has a solution in variables $D' \in \mathbb{R}^{c \times c}$, the problem reduces to $t^* = \max\{t : I_c + tD' \succeq \mathbf{0}\}$
3. If no solution D' above, some components of D' outside the image of $X^\top$;

Projecting from X of rank $c < n$

1. The LDL decomposition gives $X = \sum_{i=1}^{c} p_i L_i L_i^\top = K_{nc} K_{nc}^\top$ where K_{nc} has n rows and c columns; L_i is a column vector
2. If system $D = K_{nc} D' K_{nc}^\top$ has a solution in variables $D' \in \mathbb{R}^{c \times c}$, the problem reduces to $t^* = \max\{t : \ I_c + tD' \succeq \mathbf{0}\}$
3. This means D is in the image of X; we can ignore the null space of X and D
4. If no solution D' above, some components of D' are outside the image of X;

Projecting from X of rank $c < n$

1. The LDL decomposition gives $X = \sum_{i=1}^{c} p_i L_i L_i^\top = K_{nc} K_{nc}^\top$ where K_{nc} has n rows and c columns; L_i is a column vector
2. If system $D = K_{nc} D' K_{nc}^\top$ has a solution in variables $D' \in \mathbb{R}^{c \times c}$, the problem reduces to $t^* = \max\{t : \ I_c + tD' \succeq \mathbf{0}\}$
3. This means D is in the image of X; we can ignore the null space of X and D
4. If no solution D' above, some components of D' outside the image of X:

Projecting from X of rank $c < n$

1. The LDL decomposition gives $X = \sum_{i=1}^{c} p_i L_i L_i^\top = K_{nc} K_{nc}^\top$ where K_{nc} has n rows and c columns ; L_i is a column vector
2. If system $D = K_{nc} D' K_{nc}^\top$ has a solution in variables $D' \in \mathbb{R}^{c \times c}$, the problem reduces to $t^* = \max\{t : \ I_c + tD' \succeq \mathbf{0}\}$

   This means D is in the image of X ; we can ignore the null space of X and D
3. If no solution D' above, some components of D' outside the image of X :
   - either a few pages of other decomposions (QR)
   - (try to) find $\varepsilon > 0$ such that $X + \varepsilon D \succeq 0$ and D is in the image of $X + \varepsilon D$. If possible, we recover a problem as in 2.

Projecting from X of rank $c < n$

1. The LDL decomposition gives $X = \sum_{i=1}^{c} p_i L_i L_i^\top = K_{nc} K_{nc}^\top$ where K_{nc} has n rows and c columns ; L_i is a column vector
2. If system $D = K_{nc} D' K_{nc}^\top$ has a solution in variables $D' \in \mathbb{R}^{c \times c}$, the problem reduces to $t^* = \max\{t : \ I_c + tD' \succeq \mathbf{0}\}$

   This means D is in the image of X ; we can ignore the null space of X and D
3. If no solution D' above, some components of D' outside the image of X :
   - either a few pages of other decomposions (QR)
   - (try to) find $\epsilon > 0$ such that $X + \epsilon D \succeq \mathbf{0}$ and D is in the image of $X + \epsilon D$. If possible, we reach t^* in two steps :

$$X \to \underbrace{X + \epsilon D \to X + t^* D}_{\text{Projection solved with previous approaches}}$$

Projecting from X of rank $c < n$

1. The LDL decomposition gives $X = \sum_{i=1}^{c} p_i L_i L_i^\top = K_{nc} K_{nc}^\top$ where K_{nc} has n rows and c columns ; L_i is a column vector
2. If system $D = K_{nc} D' K_{nc}^\top$ has a solution in variables $D' \in \mathbb{R}^{c \times c}$, the problem reduces to $t^* = \max\{t : \ I_c + tD' \succeq \mathbf{0}\}$
   This means D is in the image of X ; we can ignore the null space of X and D
3. If no solution D' above, some components of D' outside the image of X :
   - either a few pages of other decomposions (QR)
   - (try to) find $\epsilon > 0$ such that $X + \epsilon D \succeq \mathbf{0}$ and D is in the image of $X + \epsilon D$. If possible, we reach t^* in two steps :

$$X \to \underbrace{X + \epsilon D \to X + t^* D}_{\text{Projection solved with previous approaches}}$$

Projecting from X of rank $c < n$

1. The LDL decomposition gives $X = \sum_{i=1}^{c} p_i L_i L_i^\top = K_{nc} K_{nc}^\top$ where K_{nc} has n rows and c columns ; L_i is a column vector
2. If system $D = K_{nc} D' K_{nc}^\top$ has a solution in variables $D' \in \mathbb{R}^{c \times c}$, the problem reduces to $t^* = \max\{t : \ I_c + tD' \succeq \mathbf{0}\}$
   This means D is in the image of X ; we can ignore the null space of X and D
3. If no solution D' above, some components of D' outside the image of X :
   - either a few pages of other decomposions (QR)
   - (try to) find $\epsilon > 0$ such that $X + \epsilon D \succeq \mathbf{0}$ and D is in the image of $X + \epsilon D$. If possible, we reach t^* in two steps :

$$X \to \underbrace{X + \epsilon D \to X + t^* D}_{\text{Projection solved with previous approaches}}$$

Sparse and densified instances

Can't compete with `Mosek` on sparse instances from public benchmarks.

Let's densify an instance : if $M \in \mathbb{R}^{n \times n}$ is ortho-normal and we replace $A_i \leftarrow M^\top A_i M \; \forall i \in [1..k]$ and $C \leftarrow M^\top CM$, the resulting instance is dense but has exactly the same optimal solution !

Instance	Original sparse instance				New method [seconds]
	k	n	|`Constr`|	`Mosek` [sec]	
`vibra1`	36	49	36	0.2	1.7
`buck1`	36	49	36	0.2	1.9
`vibra2`	144	193	144	0.8	14
`buck2`	144	193	144	1	17
`vibra3`	544	641	544	20	> 600
`buck3`	544	641	544	19	> 600
`vibra4`	1200	1345	1200	181	> 600
`buck4`	1200	1345	1200	163	> 600

Sparse and densified instances

Can't compete with `Mosek` on sparse instances from public benchmarks.
Let's densify an instance : if $M \in \mathbb{R}^{n \times n}$ is ortho-normal and we replace $A_i \leftarrow M^\top A_i M \; \forall i \in [1..k]$ and $C \leftarrow M^\top CM$, the resulting instance is dense but has exactly the same optimal solution !

Instance	Original sparse instance				New densified instance	
	k	n	|Constr|	Mosek [sec]	Mosek [sec]	New method [sec]
vibra1	36	49	36	0.2	0.3	2.2
buck1	36	49	36	0.2	0.01	0.9
vibra2	144	193	144	0.8	7.5	11
buck2	144	193	144	1	6	14
vibra3	544	641	544	20	1325	561
buck3	544	641	544	19	1320	828
vibra4	1200	1345	1200	181	36555	24450
buck4	1200	1345	1200	163	38696	15931

New easily-feasible instances

The new method has some heuristics to find an initial solution, but it remains most effective if the instance is easily feasible, a property not very often found in existing benchmarks.

- We generated 7 highly-feasible instances `highFsb-`k`-lp` with $n = 10 \cdot k$.

Instance	Projective Cut-Planes				Mosek	SeDuMi
	Iters	Time[s]	Proj	Sep	LP solver	Time[s]	Time[s]
`highFsb5-lp`	2	**0.15**	17%	1%	2%	**0.04**	**0.11**
`highFsb10-lp`	2	**0.19**	14%	1%	2%	**0.17**	**0.19**
`highFsb15-lp`	24	**0.81**	49%	7%	5%	**0.70**	**0.93**
`highFsb20-lp`	17	**0.63**	39%	6%	5%	**1.52**	**2.32**
`highFsb25-lp`	20	**0.85**	41%	9%	5%	**3.65**	**5.35**
`highFsb30-lp`	14	**0.84**	39%	7%	4%	**6.37**	**9.40**
`highFsb50-lp`	8	**1.61**	55%	4%	1%	**48.24**	**61.63**
($n = 500, k = 50$)							

Re-optimizing SDP programs ?

The iterative `Cutting-Planes` nature of the new method enables it to easily adapt to change and follow either :

- branch-and-bound decisions
- robustness over the LP part $\mathbf{a}^\top \mathbf{y} \leq c_a \; \forall (\mathbf{a}, c_a) \in$ `Constr`

Instance	Nominal	Robust	Projective Cut-Planes			Mosek		SeDuMi	
	Opt	Opt	Time (secs)	Iters	robust cuts	Time (sec)	solver calls	Time (secs)	solver calls
highFsb5-lp	2.22	2.01	0.19	4	3	0.06	2	0.13	2
highFsb10-lp	5.15	4.68	0.26	5	4	0.39	3	0.5	3
highFsb15-lp	5.45	5.35	0.80	19	6	2.61	4	4.01	4
highFsb20-lp	5.56	5.51	0.69	17	5	4.25	3	7.59	3
highFsb25-lp	6.84	6.79	0.95	17	5	9.9	3	16.4	3
highFsb30-lp	4.60	4.56	0.88	14	7	18	3	27	3
highFsb50-lp	7.38	7.32	1.6	8	5	145	3	276	3

Re-optimizing SDP programs ?

The iterative `Cutting-Planes` nature of the new method enables it to easily adapt to change and follow either :

- branch-and-bound decisions
- robustness over the LP part $\mathbf{a}^\top \mathbf{y} \leq c_a \; \forall (\mathbf{a}, c_a) \in$ `Constr`

Instance	Nominal	Robust	Projective Cut-Planes			Mosek		SeDuMi	
	Opt	Opt	Time (secs)	Iters	robust cuts	Time (sec)	solver calls	Time (secs)	solver calls
highFsb5-lp	2.22	2.01	0.19	4	3	0.06	2	0.13	2
highFsb10-lp	5.15	4.68	0.26	5	4	0.39	3	0.5	3
highFsb15-lp	5.45	5.35	0.80	19	6	2.61	4	4.01	4
highFsb20-lp	5.56	5.51	0.69	17	5	4.25	3	7.59	3
highFsb25-lp	6.84	6.79	0.95	17	5	9.9	3	16.4	3
highFsb30-lp	4.60	4.56	0.88	14	7	18	3	27	3
highFsb50-lp	7.38	7.32	1.6	8	5	145	3	276	3

Run-time depends on many code surprises

Computing $\sum_{p=1}^{k} A_p y_p$ in a schoolbook manner more expensive than all projections in my first `Matlab` implementation.

```
n = 1000
k = 90
matrices = rand(n,n,k)#var matrices
y        = rand(k)    #stands for A
function main()
  sum_matrix = zeros(n, n)
  for i in 1:n
    for j in 1:n
      s = 0.0
      for p in 1:k
        s+=matrices[i,j,p]*y[p]
      end
      sum_matrix[i,j] = s
    end
  end
end
main()
```

25 seconds

```
n_g = 1000    #_g stands for global
k_g = 90      #    variables
matrices_g = rand(k,n,n)
y_g        = rand(k)
function main(matrices,y,k,n)
  sum_matrix = zeros(n, n)
  for j in 1:n
    for i in 1:n
      s = 0.0
      for p in 1:k
        s+=matrices[p,i,j]*y[p]
      end
      sum_matrix[i,j] = s
    end
  end
end
main(matrices_g,y_g,k_g,n_g)
```

0.12 seconds

Run-time depends on many code surprises

I now explore `Julia`. I noticed enormous speed surprises even for reasons like replacing `for i for j` with `for j for i`

```
n = 1000
k = 90
matrices = rand(n,n,k)#var matrices
y        = rand(k)    #stands for A
function main()
  sum_matrix = zeros(n, n)
  for i in 1:n
     for j in 1:n
        s = 0.0
        for p in 1:k
          s+=matrices[i,j,p]*y[p]
        end
        sum_matrix[i,j] = s
     end
  end
end
main()
```

This computes $\sum_{p=1}^{k} A_p y_p$

25 seconds

Run-time depends on many code surprises

I now explore `Julia`. I noticed enormous speed surprises even for reasons like replacing `for i for j` with `for j for i`

```
n = 1000
k = 90
matrices = rand(n,n,k)#var matrices
y        = rand(k)    #stands for A
function main()
  sum_matrix = zeros(n, n)
  for i in 1:n
     for j in 1:n
        s = 0.0
        for p in 1:k
          s+=matrices[i,j,p]*y[p]
        end
        sum_matrix[i,j] = s
     end
  end
end
main()
```

This computes $\sum_{p=1}^{k} A_p y_p$

25 seconds

Run-time depends on many code surprises

I now explore `Julia`. I noticed enormous speed surprises even for reasons like replacing `for i for j` with `for j for i`

```
n = 1000
k = 90
matrices = rand(n,n,k)#var matrices
y        = rand(k)    #stands for A
function main()
  sum_matrix = zeros(n, n)
  for i in 1:n
     for j in 1:n
        s = 0.0
        for p in 1:k
          s+=matrices[i,j,p]*y[p]
        end
        sum_matrix[i,j] = s
     end
  end
end
main()
```

```
n_g = 1000    #_g stands for global
k_g = 90      #    variables
matrices_g = rand(k,n,n)
y_g        = rand(k)
function main(matrices,y,k,n)
  sum_matrix = zeros(n, n)
  for j in 1:n
     for i in 1:n
        s = 0.0
        for p in 1:k
          s+=matrices[p,i,j]*y[p]
        end
        sum_matrix[i,j] = s
     end
  end
end
main(matrices_g,y_g,k_g,n_g)
```

25 seconds

0.12 seconds

Run-time depends on many code surprises

This induces a speed-up factor of almost 2, changing the way we access the elements in variable `matrices` !

```
n = 1000
k = 90
matrices = rand(n,n,k)#var matrices
y        = rand(k)    #stands for A
function main()
  sum_matrix = zeros(n, n)
  for i in 1:n
     for j in 1:n
        s = 0.0
        for p in 1:k
          s+=matrices[i,j,p]*y[p]
        end
        sum_matrix[i,j] = s
     end
  end
end
main()
```

```
n_g = 1000    #_g stands for global
k_g = 90      #    variables
matrices_g = rand(k,n,n)
y_g        = rand(k)
function main(matrices,y,k,n)
  sum_matrix = zeros(n, n)
  for j in 1:n
     for i in 1:n
        s = 0.0
        for p in 1:k
          s+=matrices[p,i,j]*y[p]
        end
        sum_matrix[i,j] = s
     end
  end
end
main(matrices_g,y_g,k_g,n_g)
```

25 seconds

0.12 seconds

Conclusion

The new method may work over a large class of polytopes, but also over the SDP feasible area

- ⋆ A driving force is a sequence of inner solutions that do not exist in standard `Cutting-Planes`
  - It can generate a primal and a dual SDP solution at each iteration, which an Interior Point Method (IPM) can not do
  - For dense instances with $n \geq 2000$ or sparse instances with $n \geq 15000$, IPMs tend to become really slow and memory-intensive.