Validating Numerical Semidefinite Programming Solvers for Polynomial Invariants

Pierre Roux[1]

[1]ONERA

Joint work with Yuen-Lam Voronin and Sriram Sankaranarayanan (UC Boulder) and Mohamed Iguernlala (Ocamlpro) and Sylvain Conchon (Université Paris Sud)

September 20th, 2016

Numerical Optimization

Powerful tool to infer numerical invariants

$(x1, x2) \in \left\{ x_1, x_2 \;\middle|\; x_1^2 + x_2^2 \leqslant 1.5^2 \right\}$

```
while (1) {
  // Find Inv. p(x1, x2) ⩾ 0
  x1 = x1 * x2;
  x2 = -x1;
}
```

Numerical Optimization

Powerful tool to infer numerical invariants

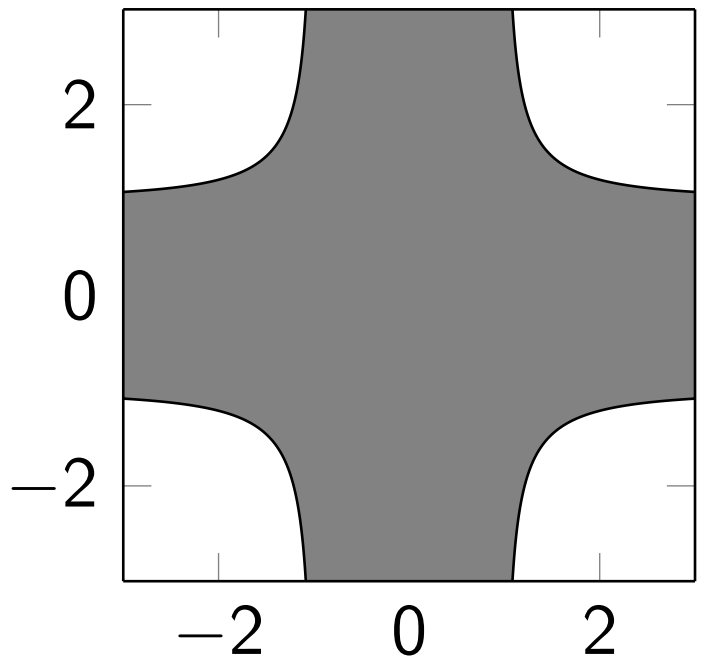
```
(x1, x2) ∈ {x1, x2 | x1^2 + x2^2 ⩽ 1.5^2}
while (1) {
  // Find Inv. p(x1, x2) ⩾ 0
  x1 = x1 * x2;
  x2 = -x1;
}
```

encoded as an optimization problem
"$polynomial\_expr(p) \geqslant 0$"

Numerical Optimization

Powerful tool to infer numerical invariants

```
(x1, x2) ∈ {x1, x2 | x1^2 + x2^2 ⩽ 1.5^2}
while (1) {
  // Find Inv. p(x1, x2) ⩾ 0
  x1 = x1 * x2;
  x2 = -x1;
}
```



encoded as an optimization problem
"*polynomial_expr*$(p) \geqslant 0$"

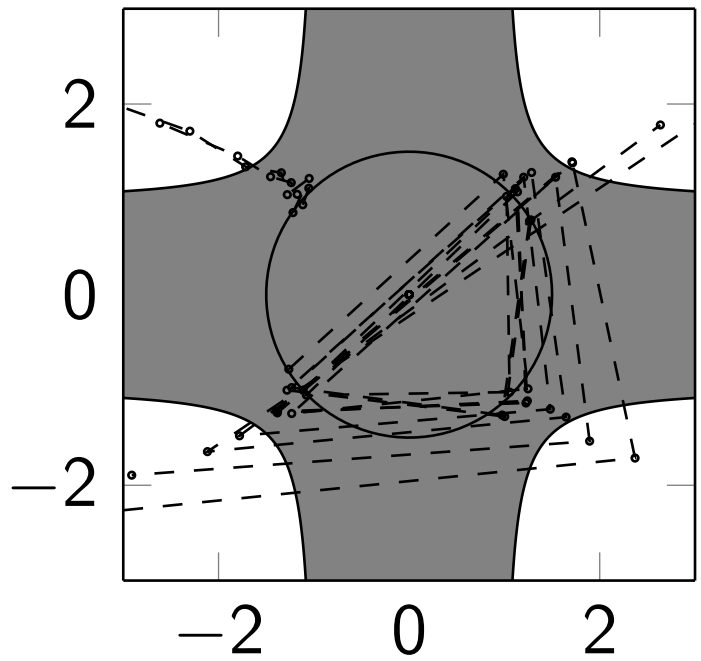
optimization procedure gives

$$p(x_1, x_2) = 1 + 2.46x_1^2 + 2.46x_2^2 - 5 \times 10^{-7}x_1^4 - 2.46x_1^2x_2^2 - 5 \times 10^{-7}x_2^4$$

Numerical Optimization

Powerful tool to infer numerical invariants

```
(x1, x2) ∈ {x1, x2 | x1^2 + x2^2 ⩽ 1.5^2}
while (1) {
  // Find Inv. p(x1, x2) ⩾ 0
  x1 = x1 * x2;
  x2 = -x1;
}
```



encoded as an optimization problem
"*polynomial_expr*$(p) \geqslant 0$"

optimization procedure gives

$$p(x_1, x_2) = 1 + 2.46x_1^2 + 2.46x_2^2 - 5 \times 10^{-7}x_1^4 - 2.46x_1^2x_2^2 - 5 \times 10^{-7}x_2^4$$

Can yield incorrect results without warning.

Polynomial Invariants

In a very nice SAS'15 paper, Adjé, Garoche and Magron offer for

```
(x1, x2) ∈ [0.9, 1.1] × [0, 0.2]
while (1) {
  pre_x1 = x1; pre_x2 = x2;
  if (x1^2 + x2^2 <= 1) {
    x1 = pre_x1^2 + pre_x2^3;
    x2 = pre_x1^3 + pre_x2^2;
  } else {
    x1 = 0.5 * pre_x1^3 + 0.4 * pre_x2^2;
    x2 = -0.6 * pre_x1^2 + 0.3 * pre_x2^2;
  }
}
```

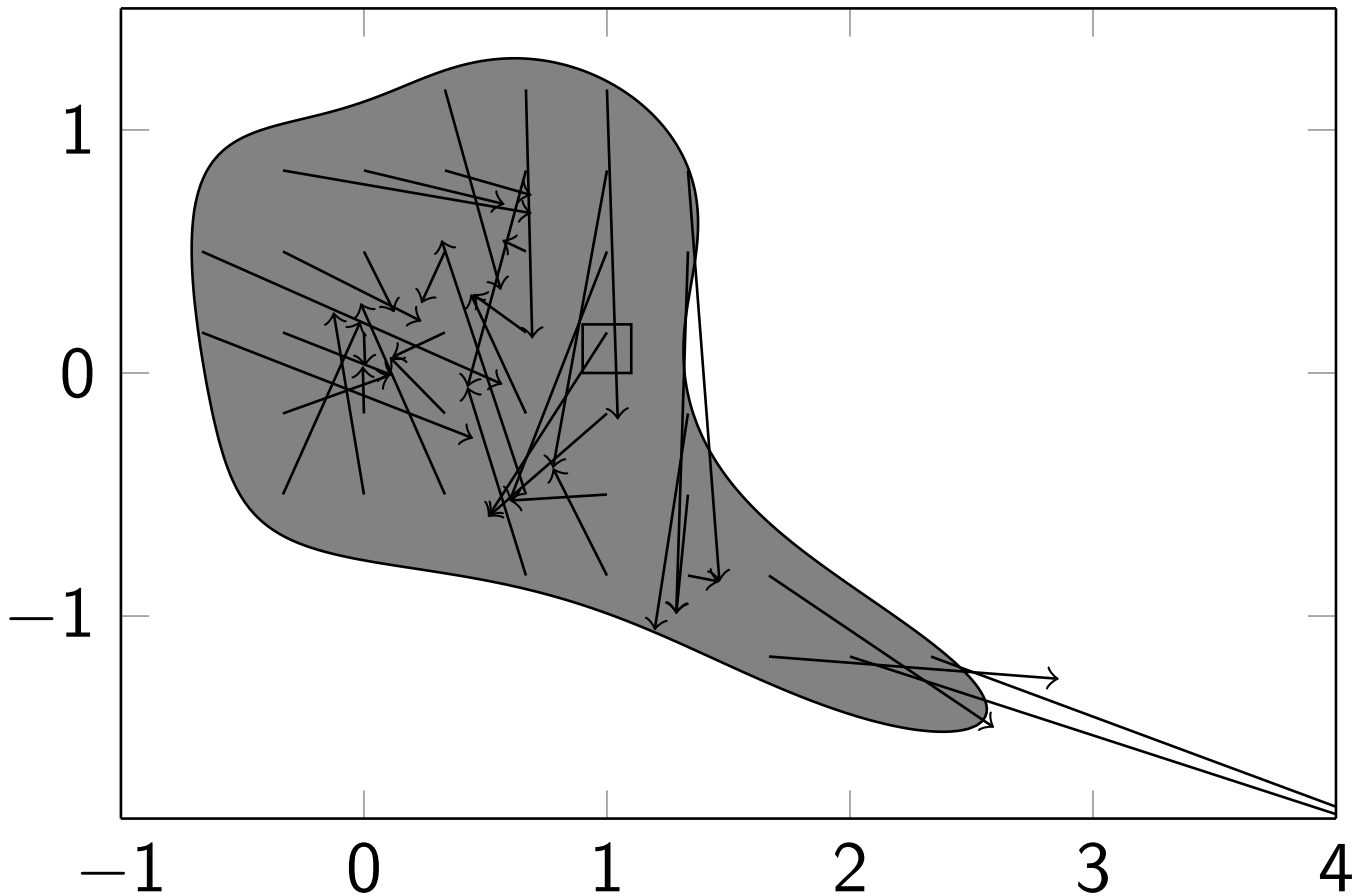
the inductive invariant $2.510902467 + 0.0050x_1 + 0.0148x_2 - 3.0998x_1^2 + 0.8037x_2^3 + 3.0297x_1^3 - 2.5924x_2^2 - 1.5266x_1x_2 + 1.9133x_1^2x_2 + 1.8122x_1x_2^2 - 1.6042x_1^4 - 0.0512x_1^3x_2 + 4.4430x_1^2x_2^2 + 1.8926x_1x_2^3 - 0.5464x_2^4 + 0.2084x_1^5 - 0.5866x_1^4x_2 - 2.2410x_1^3x_2^2 - 1.5714x_1^2x_2^3 + 0.0890x_1x_2^4 + 0.9656x_2^5 - 0.0098x_1^6 + 0.0320x_1^5x_2 + 0.0232x_1^4x_2^2 - 0.2660x_1^3x_2^3 - 0.7746x_1^2x_2^4 - 0.9200x_1x_2^5 - 0.6411x_2^6 \geqslant 0.$

Should we trust such results ?

- Some are correct (we'll prove it formally).

Should we trust such results ?

- Some are correct (we'll prove it formally).
- Other aren't (previous degree 6 polynomial)



Polynomial Invariants

Sum of Squares (SOS) Polynomials

Numerical Verification of SOS

Experiments

Use in SMT solvers

Polynomial Invariants

Sum of Squares (SOS) Polynomials

Numerical Verification of SOS

Experiments

Use in SMT solvers

In Adjé et al. paper

Look for a polynomial p s.t.

$$p - \sigma\, q \geqslant 0, \quad \sigma \geqslant 0 \qquad \text{initial condition}$$
$$p \circ f - p \geqslant 0 \qquad \text{inductiveness}$$

Then $p \geqslant 0$ is an invariant for

```
x ∈ {x | q(x) ⩾ 0}
while (1) {
  x = f(x)
}
```

In Adjé et al. paper

Look for a polynomial p minimizing w s.t.

$p - \sigma\, q \geqslant 0, \quad \sigma \geqslant 0$	initial condition
$p \circ f - p \geqslant 0$	inductiveness
$w - \|x\|_2^2 - p \geqslant 0$	look for a "small" set

Then $p \geqslant 0$ is an invariant for

```
x ∈ {x | q(x) ⩾ 0}
while (1) {
   x = f(x)
}
```

In Adjé et al. paper

Look for a polynomial p minimizing w s.t.

$p - \sigma\, q \geqslant 0, \quad \sigma \geqslant 0$	initial condition
$p \circ f - p \geqslant 0$	inductiveness
$w - \|x\|_2^2 - p \geqslant 0$	look for a "small" set

Then $p \geqslant 0$ is an invariant for

```
x ∈ {x | q(x) ⩾ 0}
while (1) {
   x = f(x)
}
```

$q \geqslant 0$ relaxed as "q sum of squares" and solved with SDP solvers.

Polynomial Invariants

Sum of Squares (SOS) Polynomials

Numerical Verification of SOS

Experiments

Use in SMT solvers

Sum of Squares (SOS) Polynomials

Definition (SOS Polynomial)

A polynomial p is SOS if there are polynomials $q_1, \ldots, q_m$ s.t.

$$p = \sum_i q_i^2.$$

- If p SOS then $p \geqslant 0$

Sum of Squares (SOS) Polynomials

Definition (SOS Polynomial)

A polynomial p is SOS if there are polynomials $q_1, \ldots, q_m$ s.t.

$$p = \sum_i q_i^2.$$

- If p SOS then $p \geqslant 0$
- p SOS iff there exist $z := \left[1, x_0, x_1, x_0 x_1, \ldots, x_n^d\right]$ and $Q \succeq 0$ (i.e., for all $x, x^T Q x \geqslant 0$) s.t.

$$p = z^T Q z.$$

$\Rightarrow$ SOS can be encoded as semi-definite programming (SDP).

SOS: Example

Example

Is $p(x,y) := 2x^4 + 2x^3y - x^2y^2 + 5y^4$ SOS ?

$$p(x,y) = \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}$$

that is

$$p(x,y) = q_{11}x^4 + 2q_{13}x^3y + 2q_{23}xy^3 + (2q_{12} + q_{33})x^2y^2 + q_{22}y^4$$

SOS: Example

Example

Is $p(x,y) := 2x^4 + 2x^3y - x^2y^2 + 5y^4$ SOS ?

$$p(x,y) = \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}$$

that is

$$p(x,y) = q_{11}x^4 + 2q_{13}x^3y + 2q_{23}xy^3 + (2q_{12} + q_{33})x^2y^2 + q_{22}y^4$$

hence $q_{11} = 2$, $2q_{13} = 2$, $2q_{23} = 0$, $2q_{12} + q_{33} = -1$, $q_{22} = 5$.

SOS: Example

Example

Is $p(x,y) := 2x^4 + 2x^3y - x^2y^2 + 5y^4$ SOS ?

$$p(x,y) = \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}$$

that is

$$p(x,y) = q_{11}x^4 + 2q_{13}x^3y + 2q_{23}xy^3 + (2q_{12} + q_{33})x^2y^2 + q_{22}y^4$$

hence $q_{11} = 2$, $2q_{13} = 2$, $2q_{23} = 0$, $2q_{12} + q_{33} = -1$, $q_{22} = 5$.

For instance

$$Q = \begin{bmatrix} 2 & -3 & 1 \\ -3 & 5 & 0 \\ 1 & 0 & 5 \end{bmatrix} = L^T L \qquad L = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & -3 & 1 \\ 0 & 1 & 3 \end{bmatrix}$$

hence $p(x,y) = \frac{1}{2}\left(2x^2 - 3y^2 + xy\right)^2 + \frac{1}{2}\left(y^2 + 3xy\right)^2$.

Polynomial Invariants

Sum of Squares (SOS) Polynomials

Numerical Verification of SOS

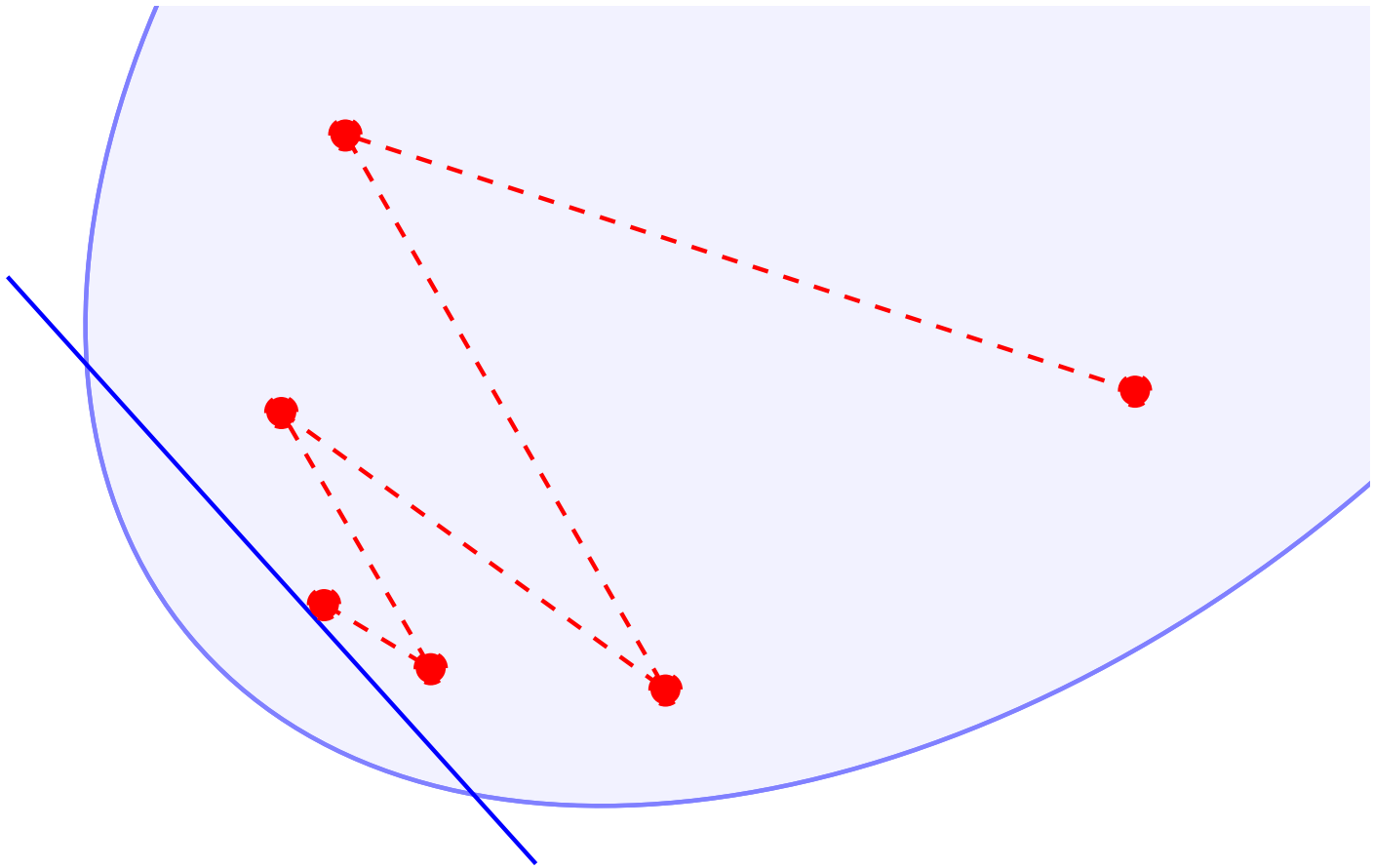
Experiments

Use in SMT solvers

Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

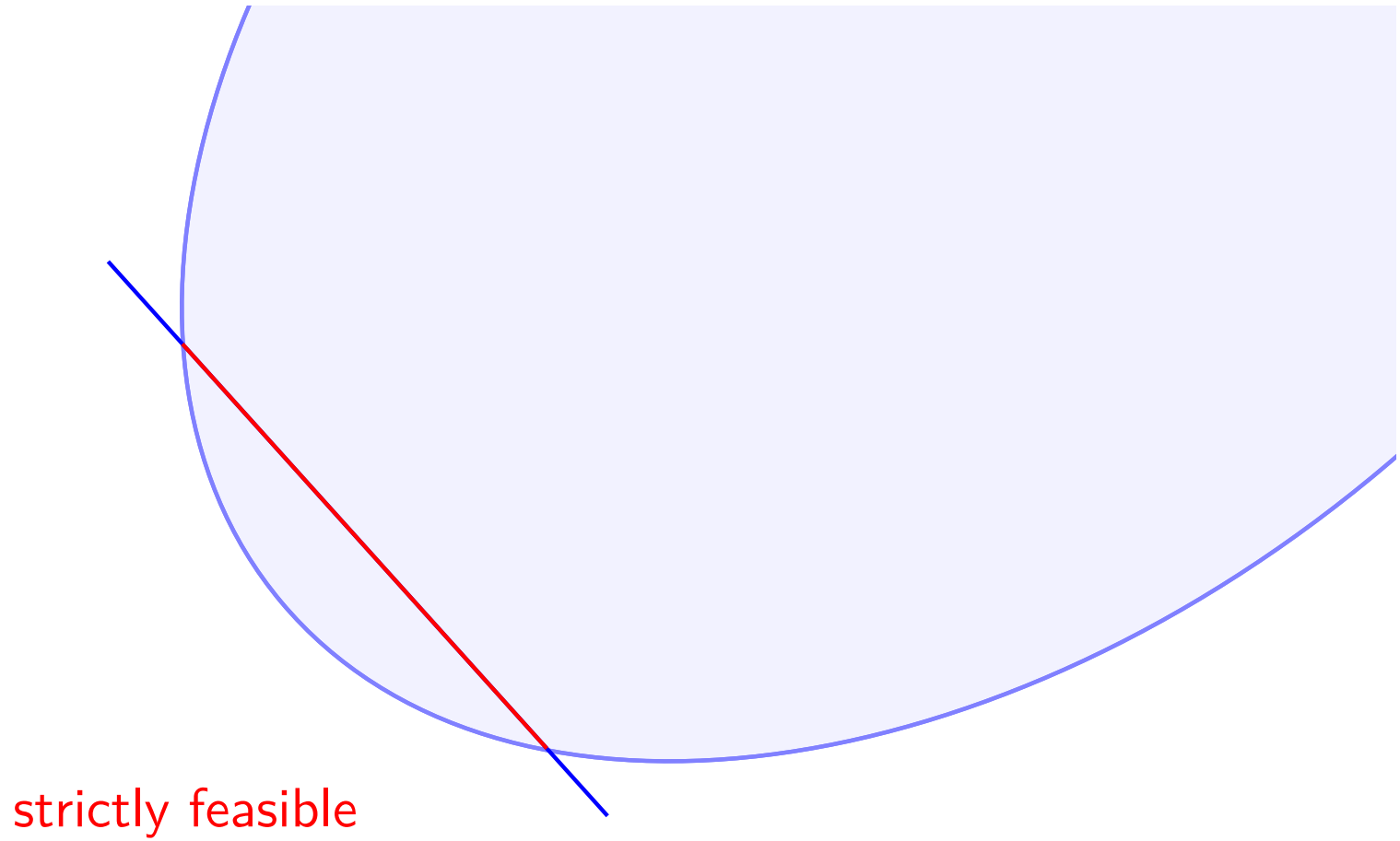
- inexact termination



Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

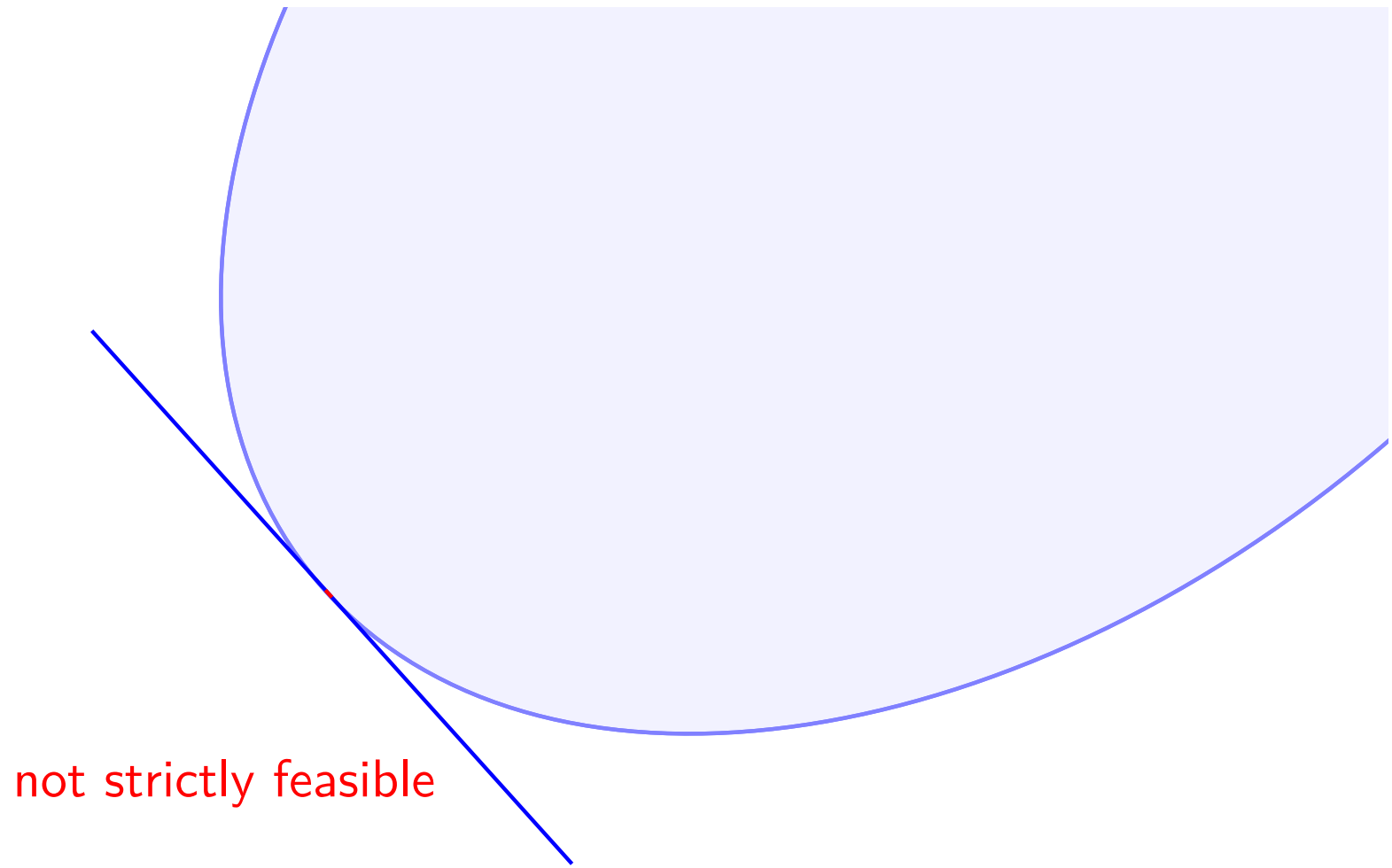
- inexact termination
- failure of strict feasibility



Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

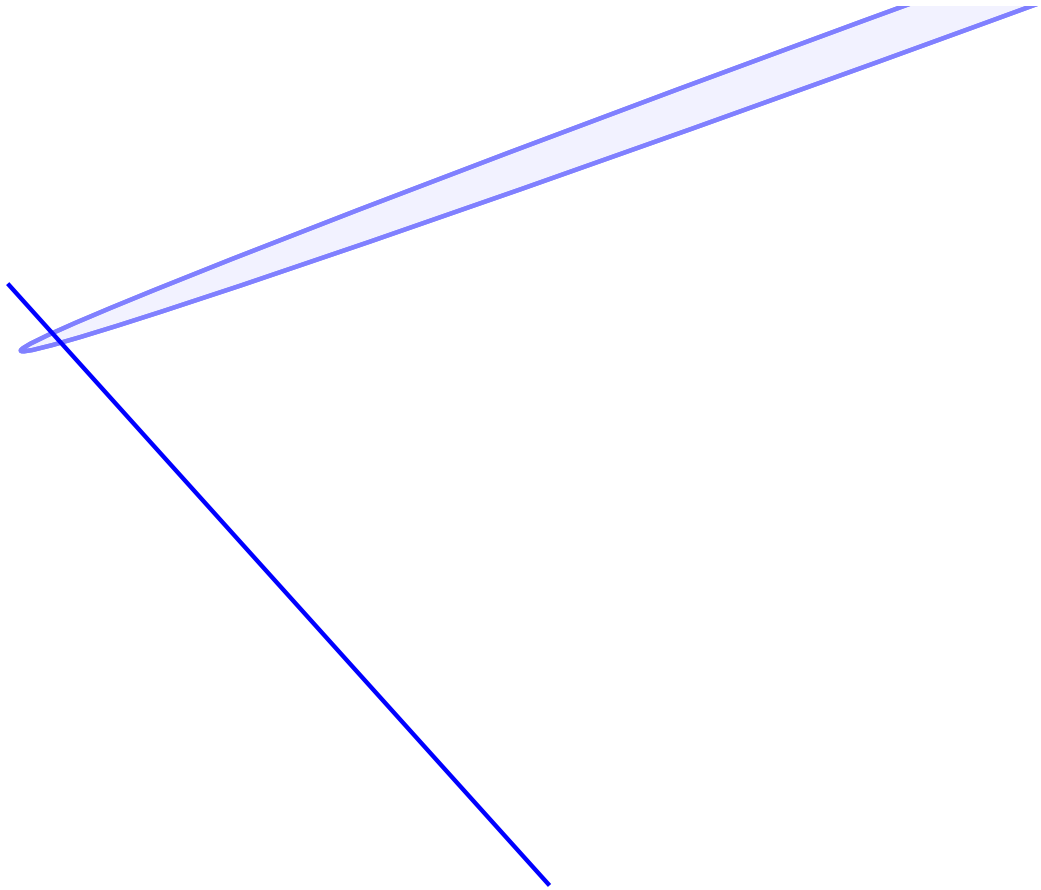
- inexact termination
- failure of strict feasibility



Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

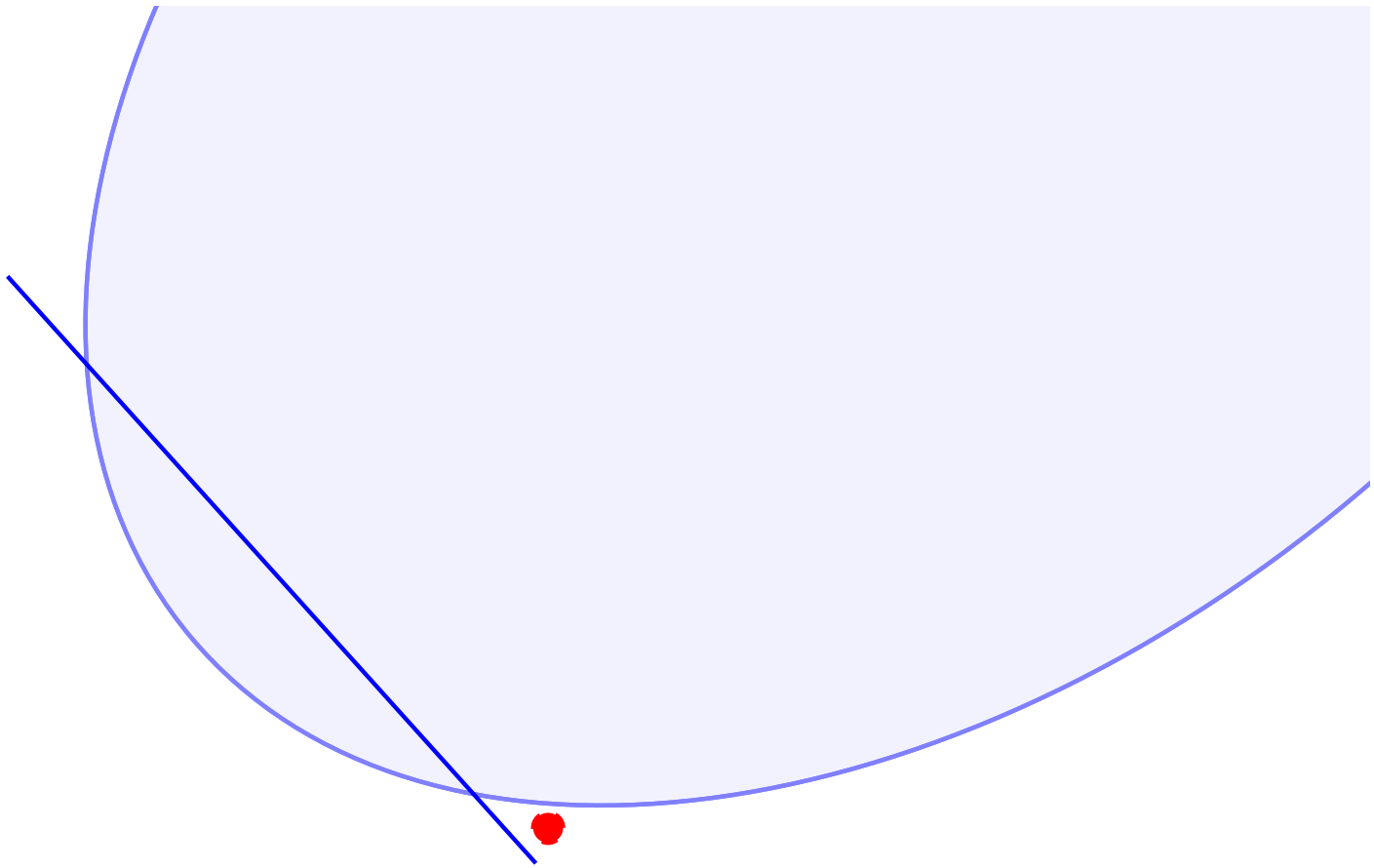
- inexact termination
- failure of strict feasibility
- ill conditioning



Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

- inexact termination
- failure of strict feasibility
- ill conditioning
- floating-point rounding errors



Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

- inexact termination
- failure of strict feasibility
- ill conditioning
- floating-point rounding errors

State of the art [Harrison, Peyrl and Parrilo, Monniaux and Corbineau, Kaltofen et al., Magron et al.]

- round to exact rational solution (heuristic)
- proofs in rational arithmetic (expensive).

SOS: Using approximate SDP solvers

Result Q from SDP solver will only satisfy equality constraints up to some ϵ

$$p = z^T Q z + z^T E z, \qquad \forall i\, j, |E_{i,j}| \leqslant \epsilon.$$

SOS: Using approximate SDP solvers

Result Q from SDP solver will only satisfy equality constraints up to some ϵ

$$p = z^T Q z + z^T E z, \qquad \forall i\, j, |E_{i,j}| \leqslant \epsilon.$$

If $Q + E \succeq 0$ then $p = z^T (Q + E) z$ is SOS.

SOS: Using approximate SDP solvers

Result Q from SDP solver will only satisfy equality constraints up to some ϵ

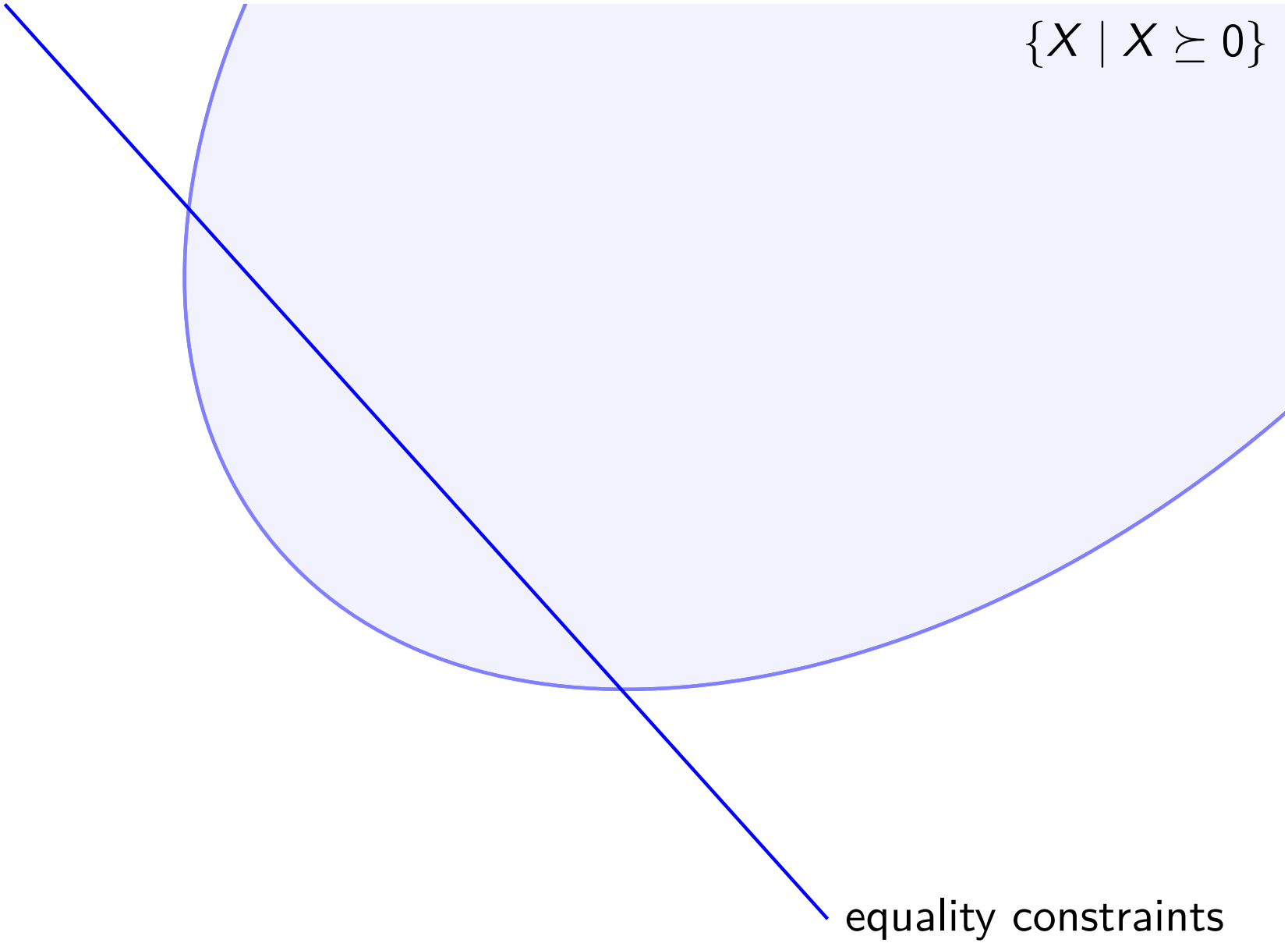
$$p = z^T Q z + z^T E z, \qquad \forall i\, j, |E_{i,j}| \leqslant \epsilon.$$

If $Q + E \succeq 0$ then $p = z^T (Q + E) z$ is SOS.

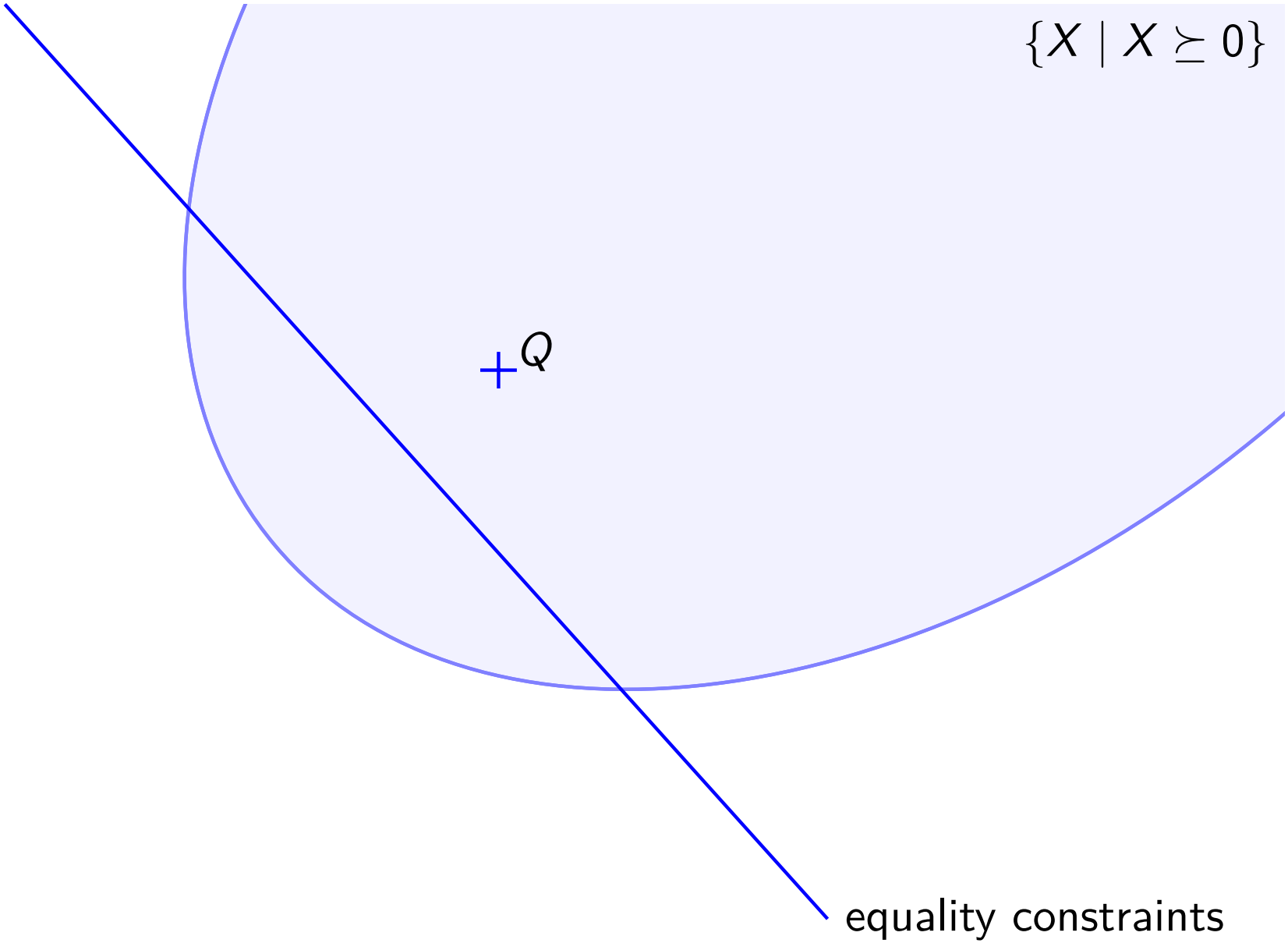
- Hence the validation method: given $p \simeq z^T Q z$
  1. Check that all monomials of p are in $z z^T$.
  2. Bound difference ϵ between coefficients of p and $z^T Q z$.
  3. If $Q - s \epsilon I \succeq 0$ ($s :=$ size of Q), then p is proved SOS.
- 2 can be done with interval arithmetic
  and 3 with a Cholesky decomposition ($\Theta(s^3)$ flops).

$\Rightarrow$ Efficient validation method using just floats.

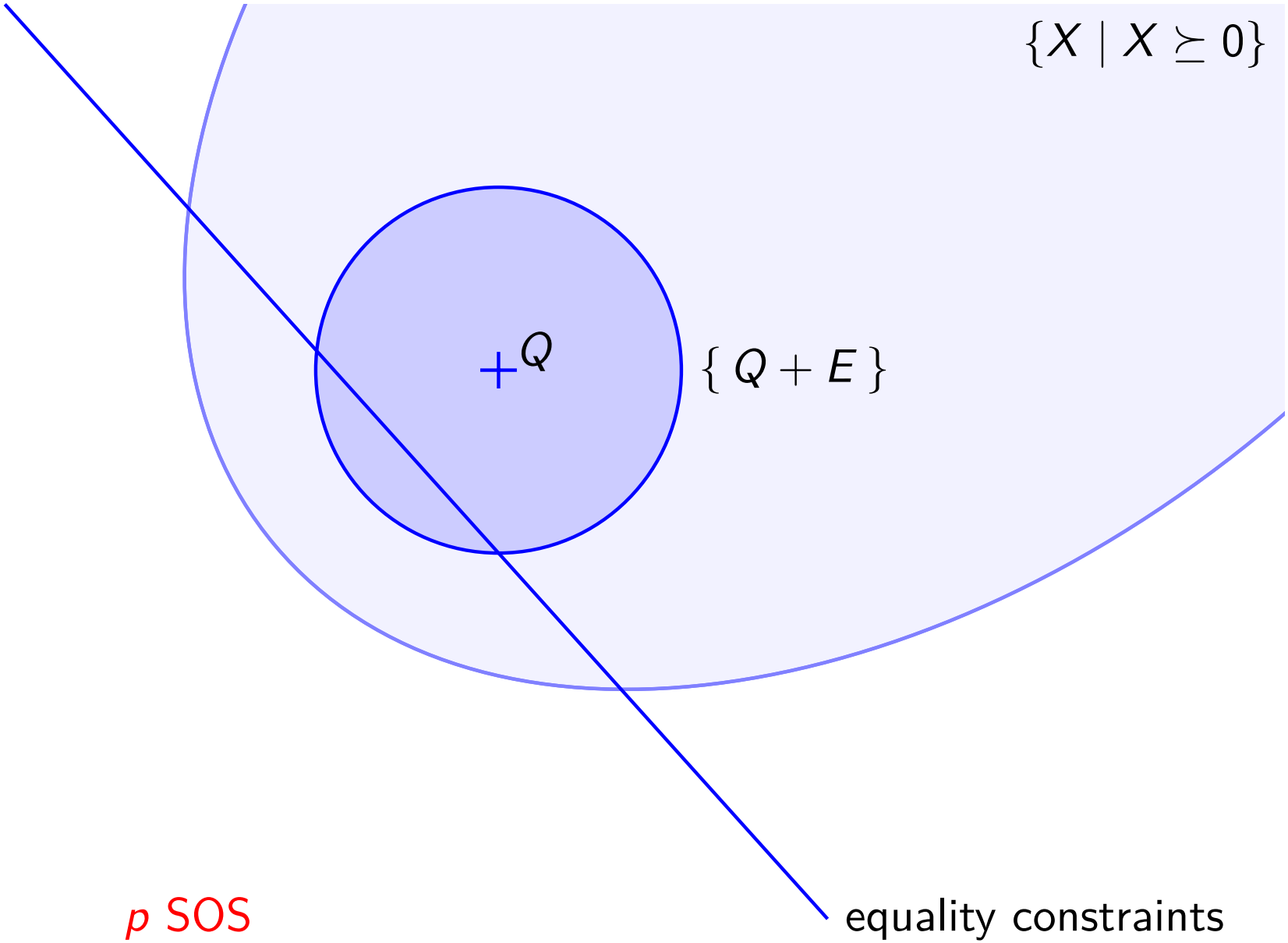
Intuitively



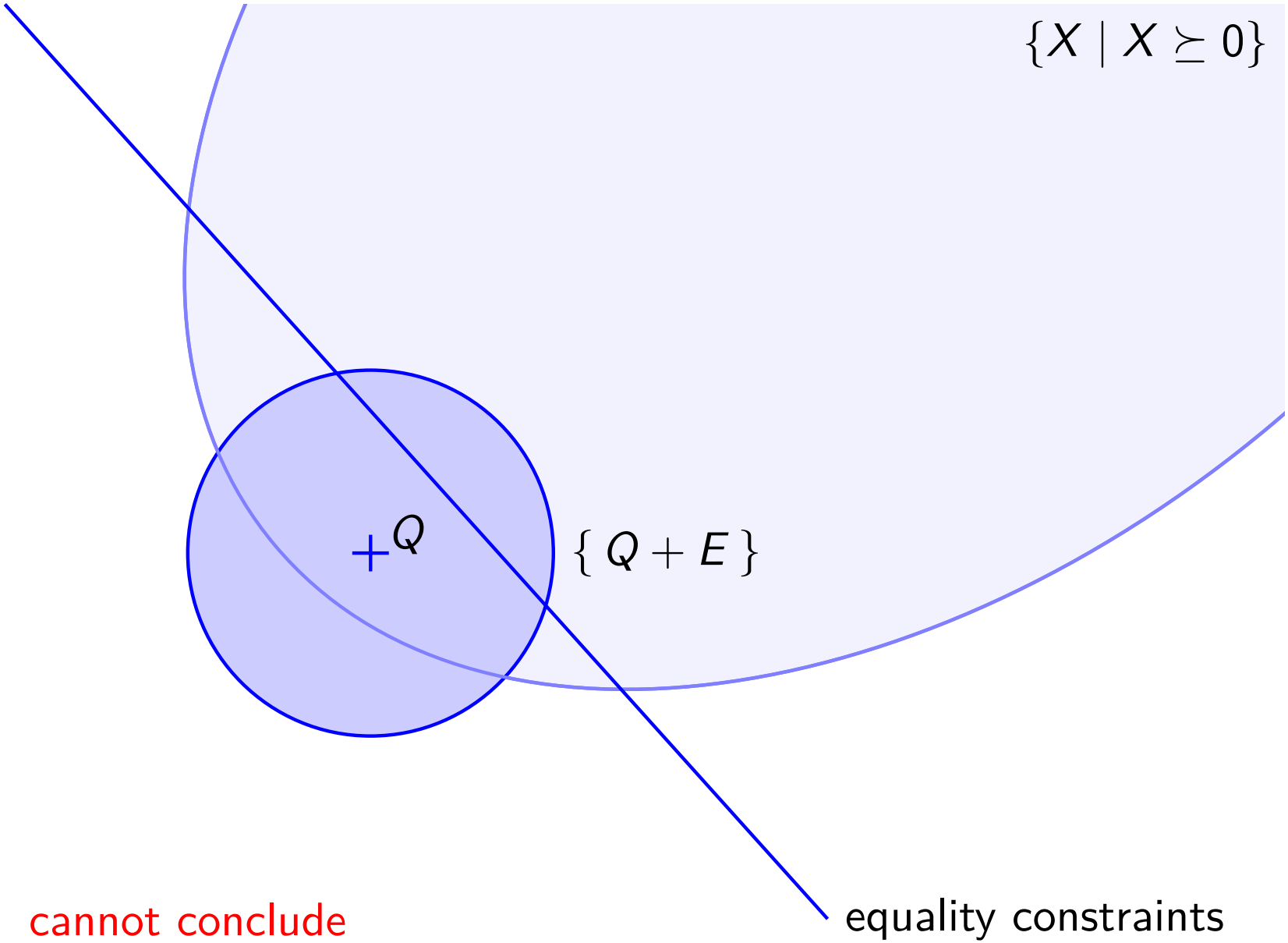
Intuitively



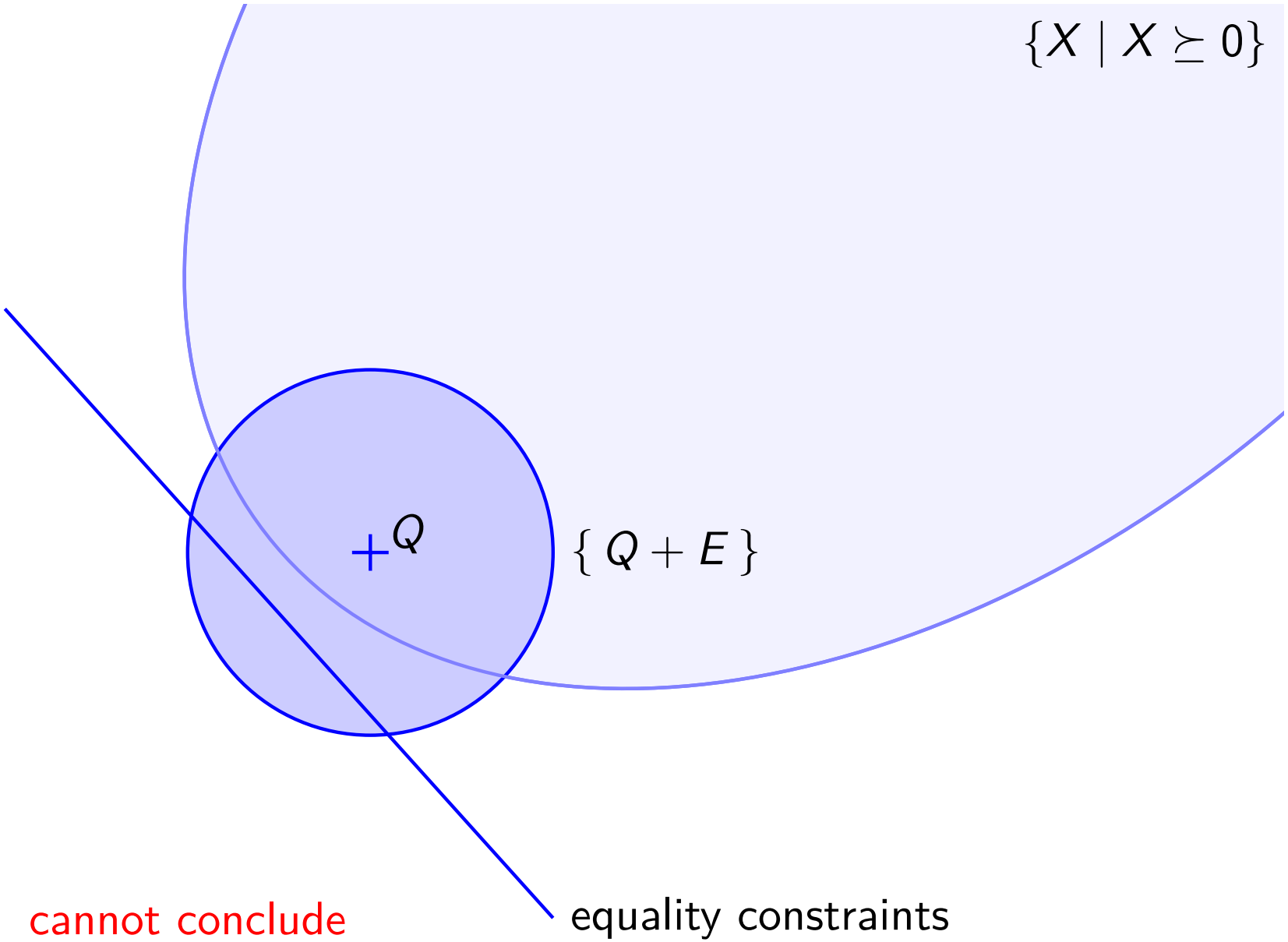
Intuitively



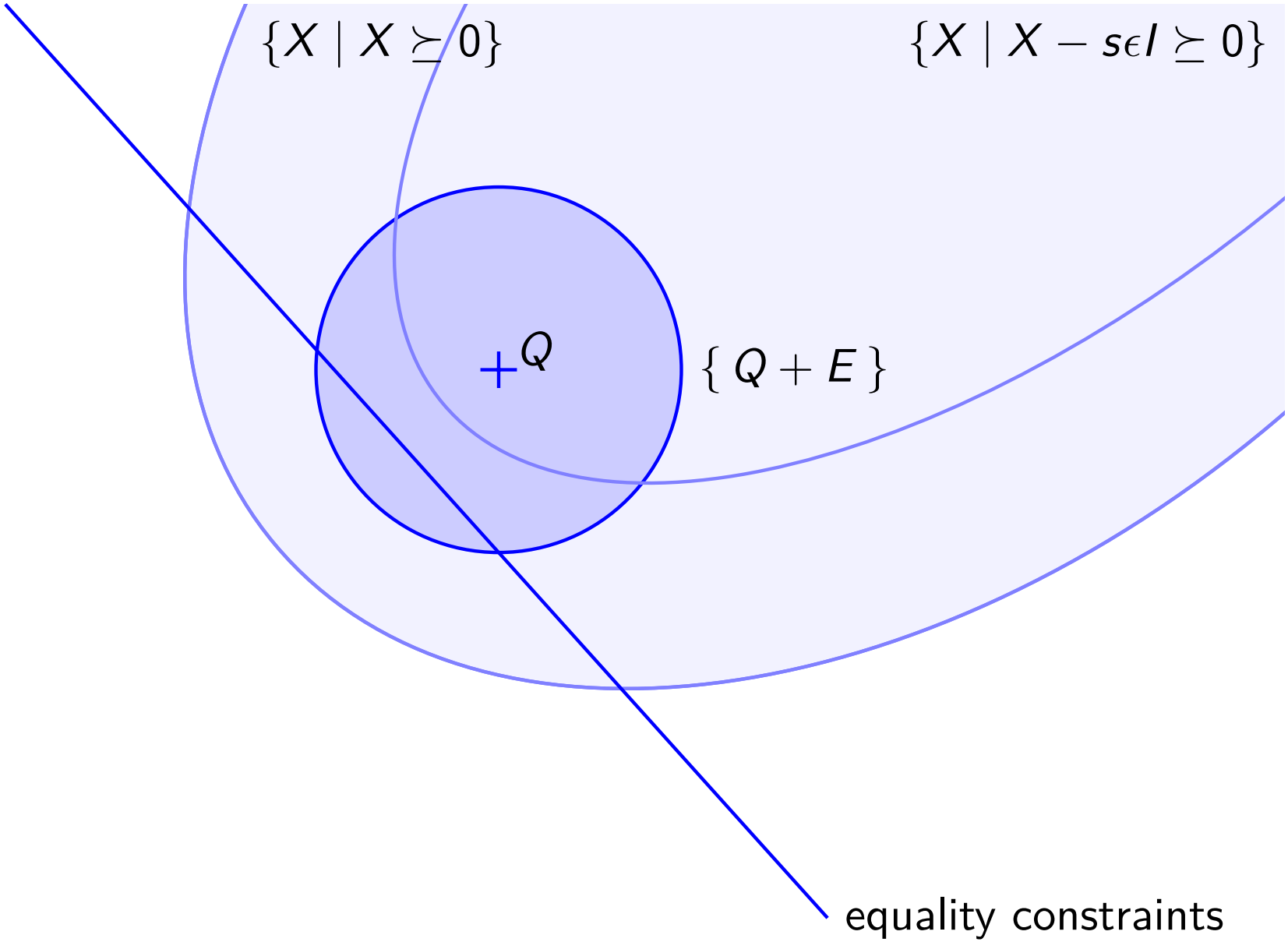
Intuitively



Intuitively



Padding


$\{X \mid X \succeq 0\}$
$\{X \mid X - s\epsilon I \succeq 0\}$
Q
$\{ Q + E \}$
equality constraints

Choose ϵ

In practice ϵ is a stopping criterion of SDP solvers.

- Works well for solvers with nice stopping criteria (e.g., CSDP, SDPA).
- Less when they are not even documented (e.g., MOSEK).
- Idea: use a "nice" solver as postprocessing for a "bad" one.

About strict feasibility

Recall the inductivity constraint $p - p \circ f \geqslant 0$,
for any fixpoint $x_0 = f(x_0)$,

$$(p - p \circ f)(x_0) = p(x_0) - p(x_0) = 0.$$

$\Rightarrow$ failure of strict feasibility.

(x_0 exists when f continuous, (Brouwer fixpoint theorem)).

About strict feasibility

Recall the inductivity constraint $p - p \circ f \geqslant 0$,
for any fixpoint $x_0 = f(x_0)$,

$$(p - p \circ f)(x_0) = p(x_0) - p(x_0) = 0.$$

$\Rightarrow$ failure of strict feasibility.

(x_0 exists when f continuous, (Brouwer fixpoint theorem)).

p s.t. $p - p \circ f \geqslant 0$ is a *Lyapunov function* for control theorists.
This is a very strong requirement, indeed we only need

$$\forall x,\ p(x) \leqslant 0 \Rightarrow p \circ f(x) \leqslant 0$$

About strict feasibility

Recall the inductivity constraint $p - p \circ f \geqslant 0$,
for any fixpoint $x_0 = f(x_0)$,

$$(p - p \circ f)(x_0) = p(x_0) - p(x_0) = 0.$$

$\Rightarrow$ failure of strict feasibility.

(x_0 exists when f continuous, (Brouwer fixpoint theorem)).

p s.t. $p - p \circ f \geqslant 0$ is a *Lyapunov function* for control theorists.
This is a very strong requirement, indeed we only need

$$\forall x,\ p(x) \leqslant 0 \Rightarrow p \circ f(x) \leqslant 0$$

- First approximately solve $p - p \circ f \geqslant 0$;
- Then, for the obtained p, look for $\sigma \geqslant 0$ s.t. $\sigma p - p \circ f \geqslant 0$.

Polynomial Invariants

Sum of Squares (SOS) Polynomials

Numerical Verification of SOS

Experiments

Use in SMT solvers

Experiments

- Reuse benchmarks from Adjé et al.
- Synthesize Invariants using their method
- Attempt proofs using rational arithmetic
- Then floating-point arithmetic

OCaml library OSDP: `http://cavale.enseeiht.fr/osdp/`

Experiments: Invariant Synthesis

Table: Time to synthesize candidate invariants for benchmarks of Adjé et al. n is the number of variables. All times are in seconds, TO means timeout (900s) and MO out of memory (4GB).

	$d = 4$	$d = 6$	$d = 8$	$d = 10$
Ex. 4 ($n = 2$)	0.25	0.95	3.31	8.85
Ex. 5 ($n = 3$)	0.48	2.83	22.75	112.37
Ex. 6 ($n = 4$)	2.12	64.07	TO	MO
Ex. 7 ($n = 2$)	0.25	0.96	3.15	10.45
Ex. 8 ($n = 2$)	0.17	0.34	0.74	1.93

Experiments: Proof using Exact Rational Arithmetic

Table: Checking the candidate invariants with the implementation of Monniaux and Corbineau. All times in seconds, NS means no proof is found, TO means timeout (900s) and MO out of memory (4GB).

	$d = 4$		$d = 6$		$d = 8$		$d = 10$	
	init	ind.	init	ind.	init	ind.	init	ind.
Ex. 4 ($n = 2$)	1.43	NS	3.35	TO	19.80	MO	142.33	MO
Ex. 5 ($n = 3$)	3.82	TO	142.49	MO	TO	MO	TO	MO
Ex. 6 ($n = 4$)	32.20	TO	TO	MO	—	—	—	—
Ex. 7 ($n = 2$)	1.48	NS	3.36	TO	18.36	MO	120.40	MO
Ex. 8 ($n = 2$)	1.93	12.81	3.78	NS	26.29	TO	193.79	TO

Experiments: Proof using Floating-Point Arithmetic

Table: Checking the candidate invariants with our method.
Counter-examples are easily found for Ex. 4, $d = 4, 6$ and Ex. 7, $d = 4$.
Whether other unproved "invariants" are inductive remains unknown.

	$d = 4$		$d = 6$		$d = 8$		$d = 10$	
	init	ind.	init	ind.	init	ind.	init	ind.
Ex. 4 ($n = 2$)	0.05	NS	0.07	NS	0.19	3.03	0.17	NS
Ex. 5 ($n = 3$)	0.08	0.33	0.23	2.20	0.74	14.55	2.50	92.15
Ex. 6 ($n = 4$)	0.22	1.52	1.26	38.94	—	—	—	—
Ex. 7 ($n = 2$)	0.05	NS	0.07	0.85	0.19	3.32	0.17	NS
Ex. 8 ($n = 2$)	0.05	0.13	0.07	NS	0.09	NS	0.15	NS

Future work

- Coq automatic tactic
  (with E. Martin-Dorel, almost done)
- Use of facial reduction to alleviate strict feasibility constraint
- Applications to hybrid systems

Polynomial Invariants

Sum of Squares (SOS) Polynomials

Numerical Verification of SOS

Experiments

Use in SMT solvers

Example

SMT solvers have a hard time with non-linear numerical problems.

Demo

```
typedef struct { double x0, x1, x2; } state;

/*@ predicate inv(state *s) =
  @   6.04 * s->x0 * s->x0 - 9.65 * s->x0 * s->x1
  @   - 2.26 * s->x0 * s->x2 + 11.36 * s->x1 * s->x1
  @   + 2.67 * s->x1 * s->x2 + 3.76 * s->x2 * s->x2 <= 1; */

/*@ requires \valid(s) && inv(s) && -1 <= in0 <= 1;
  @ ensures inv(s); */
void step(state *s, double in0) {
  double pre_x0 = s->x0, pre_x1 = s->x1, pre_x2 = s->x2;

  s->x0 = 0.9379*pre_x0 - 0.0381*pre_x1 - 0.0414*pre_x2 + 0.0237*in0;
  s->x1 = -0.0404*pre_x0 + 0.968*pre_x1 - 0.0179*pre_x2 + 0.0143*in0;
  s->x2 = 0.0142*pre_x0 - 0.0197*pre_x1 + 0.9823*pre_x2 + 0.0077*in0;
}
```

Using Numerical Solvers

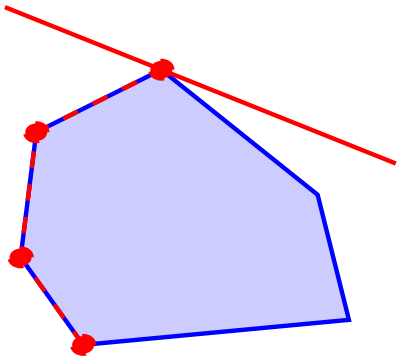
- First order theory of real numbers is decidable (Tarski).
- But complexity remains high.
- State of the art for non linear SMT:
  - Cylindrical Algebraic Decomposition (e.g., Z3);
  - Interval arithmetic with branch & bound (e.g., dReal).

$\Rightarrow$ We offer to use numerical optimization solvers: semi-definite programming (SDP) solvers.

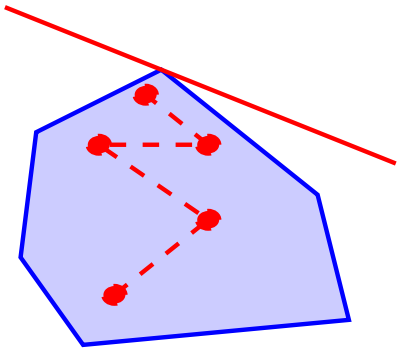
SDP solvers yield approximate solutions

- Linear programming

simplex: exact solution



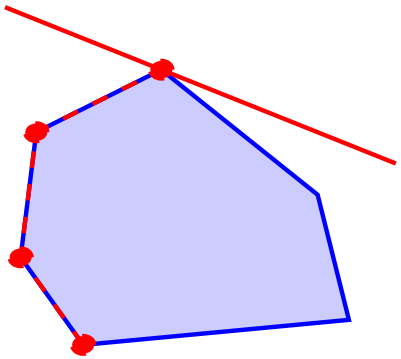
interior-point: approximate solution



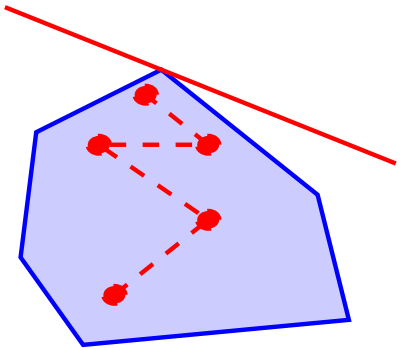
SDP solvers yield approximate solutions

- Linear programming

simplex: exact solution

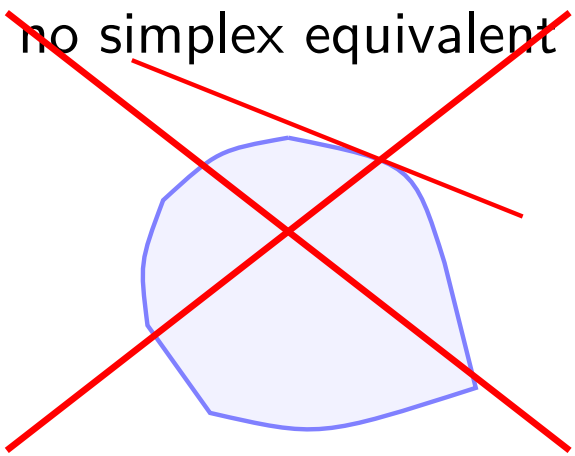


interior-point: approximate solution

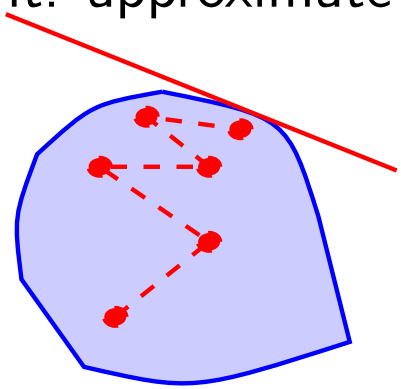


- Semi-definite programming

no simplex equivalent



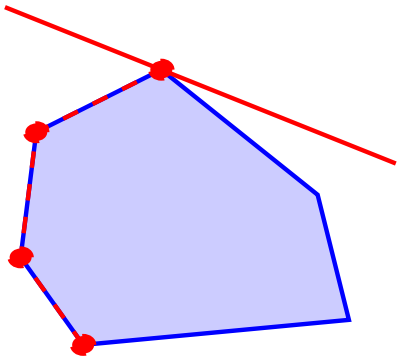
interior-point: approximate solution



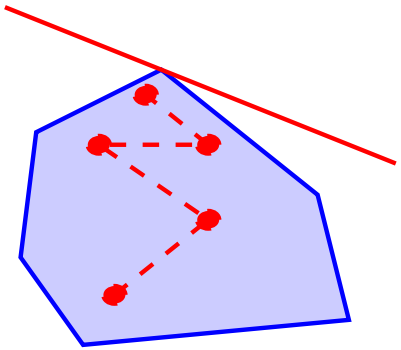
SDP solvers yield approximate solutions

- Linear programming

simplex: exact solution

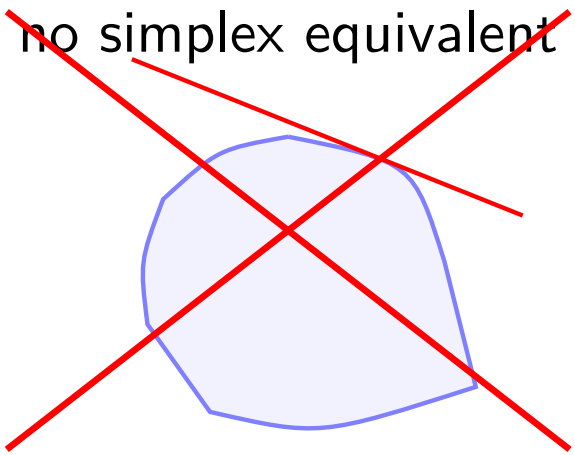


interior-point: approximate solution

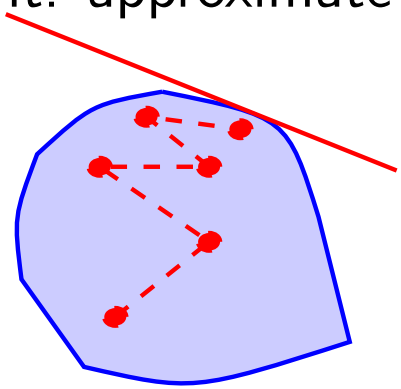


- Semi-definite programming

no simplex equivalent



interior-point: approximate solution



$\Rightarrow$ incompleteness, soundness requires care

Positivstellensatz

We want to prove that

$$p_1(x_1, \ldots, x_n) \geqslant 0 \wedge \ldots \wedge p_m(x_1, \ldots, x_n) \geqslant 0$$

is not satisfiable.

Positivstellensatz

We want to prove that

$$p_1(x_1, \ldots, x_n) \geqslant 0 \wedge \ldots \wedge p_m(x_1, \ldots, x_n) \geqslant 0$$

is not satisfiable.

Sufficient condition: there exist $r_i \in \mathbb{R}[x]$ s.t.

$$-\sum_i r_i \, p_i > 0 \quad \text{and} \quad \forall i, r_i \geqslant 0$$

Positivstellensatz

We want to prove that

$$p_1(x_1, \ldots, x_n) \geqslant 0 \wedge \ldots \wedge p_m(x_1, \ldots, x_n) \geqslant 0$$

is not satisfiable.

Sufficient condition: there exist $r_i \in \mathbb{R}[x]$ s.t.

$$-\sum_i r_i \, p_i > 0 \quad \text{and} \quad \forall i, r_i \geqslant 0$$

- equivalence under hypotheses (Putinar's Positivstellensatz)
- no practical bound on degrees of $r_i \Rightarrow$ will be arbitrarily fixed
- we are back to first part of this talk

Soundness Verification for SOS: Conclusion

efficiency	▶ use off-the-shelf SDP solvers ▶ Cholesky in exact rational arithmetic: would be exponential in worst case ▶ Cholesky in floating-point arithmetic: very small overhead for verif w.r.t. SDP solver
soundness	▶ very small trusted code base ▶ core algorithm proved in Coq
completeness	▶ incomplete method

The OSDP Library

OCaml library OSDP:

- simple interface to SOS programming
- interfaces SDP solvers
  - Csdp
  - Mosek
  - SDPA
- available at `https://cavale.enseeiht.fr/osdp/`

Integration in Alt-Ergo

- Alt-Ergo maintains a map: polynomial $p_i \rightarrow$ interval $[a_i, b_i]$.
- The constraints

$$-\sum_i r_i \left(p_i - a_i\right) + r'_i \left(b_i - p_i\right) > 0 \quad \text{and} \quad \forall i, r_i \geqslant 0 \wedge r'_i \geqslant 0$$

  are provided to OSDP.
- If OSDP returns a valid solution, $\bigwedge_i p_i \in [a_i, b_i]$ is `unsat`
- otherwise: `unknown`
- Future work:
  unsat-cores if `unsat`, narrowing intervals if `unknown`

Experimental Results

Benchmarks NIsat (QF_NRA in SMT-LIB).

	`metit` (8276)		`keymera` (421)		`zankl` (166)		`hong` (20)		all (8883)	
solver	unsat	time	unsat	time	unsat	time	unsat	time	unsat	time
AESDP	2138	66.36	352	45.41	14	1.06	**20**	**0.60**	2524	113.43
dReal	2282	45.33	216	1.08	4	0.02	**20**	**0.10**	2522	46.53
Z3	**2836**	**240.34**	**420**	**4.32**	**27**	**0.46**	9	13.19	**3292**	**258.31**

On Intel Xeon 2.3 GHz, time limits 900 s and memory limits 2 GB.
All times are in seconds.

Experimental Results

Benchmarks NIsat (QF_NRA in SMT-LIB).

	metit (8276)		keymera (421)		zankl (166)		hong (20)		all (8883)	
solver	unsat	time	unsat	time	unsat	time	unsat	time	unsat	time
AESDP	2138	66.36	352	45.41	14	1.06	**20**	**0.60**	2524	113.43
dReal	2282	45.33	216	1.08	4	0.02	**20**	**0.10**	2522	46.53
Z3	**2836**	**240.34**	**420**	**4.32**	**27**	**0.46**	9	13.19	**3292**	**258.31**

More numerical benchmarks.

	C (67)		quadr (67)		flyspek (20)		global (14)		all (168)	
solver	unsat	time	unsat	time	unsat	time	unsat	time	unsat	time
AESDP	**62**	**15.08**	**67**	**4.70**	**19**	**4.32**	**14**	**4.71**	**162**	**28.81**
dReal	0	0.00	13	23.36	16	11.77	9	0.05	38	35.18
Z3	0	0.00	29	185.95	10	0.05	13	37.90	52	223.90

On Intel Xeon 2.3 GHz, time limits 900 s and memory limits 2 GB.
All times are in seconds.

Conclusion

- Does not outperform state-of-the-art symbolic methods.
- But enables to solve problems out of reach for such methods.
- In particular, numerical problems arising in verification of functional properties of control-command programs.

Conclusion

- Does not outperform state-of-the-art symbolic methods.
- But enables to solve problems out of reach for such methods.
- In particular, numerical problems arising in verification of functional properties of control-command programs.

Future work

- Better integration in Alt-Ergo:
  - production of unsat-cores;
  - narrowing of interval bounds;
  - handling of unions of intervals.
- Combination with symbolic (or other numerical) methods.

Questions

Thanks for your attention!