A Reflexive Tactic for Polynomial Positivity

using Numerical Solvers and Floating-Point Computations

Érik Martin-Dorel[1] Pierre Roux[2]

[1]IRIT, Université Paul Sabatier, Toulouse, France

[2]ONERA, Toulouse, France

January 16th, 2017
CPP 2017

Motivation

- Polynomial inequalities in the real field are decidable (Tarski)
- But exact algo. expensive

Motivation

- Polynomial inequalities in the real field are decidable (Tarski)
- But exact algo. expensive

$\Rightarrow$ Use incomplete numerical methods

  - off-the-shelf optimization solvers
  - a posteriori validation with exact rational arithmetic: state of the art (simple but costly)

Motivation

- Polynomial inequalities in the real field are decidable (Tarski)
- But exact algo. expensive

$\Rightarrow$ Use incomplete numerical methods

  - off-the-shelf optimization solvers
  - a posteriori validation with exact rational arithmetic: state of the art (simple but costly)
  - a posteriori validation with floating-point arithmetic (more efficient but non trivial)

  $\Rightarrow$ We'd like formal proofs

Motivation

- Polynomial inequalities in the real field are decidable (Tarski)
- But exact algo. expensive

$\Rightarrow$ Use incomplete numerical methods

  - off-the-shelf optimization solvers
  - a posteriori validation with exact rational arithmetic: state of the art (simple but costly)
  - a posteriori validation with floating-point arithmetic (more efficient but non trivial)

  $\Rightarrow$ We'd like formal proofs

demo.v

Agenda

Agenda

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 $p \geq 0$

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 $p \geq 0$
- 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 $x, x^T Q x \geq 0$) 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 $p(x, y) = \frac{1}{2}\left(2x^2 - 3y^2 + xy\right)^2 + \frac{1}{2}\left(y^2 + 3xy\right)^2.$

Agenda

1. Sum of Squares (SOS) Polynomials
2. Numerical Verification
3. Formalization & Reflexive Tactic
4. Benchmarks
5. Conclusion

SOS: Using approximate SDP solvers

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

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

SOS: Using approximate SDP solvers

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

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

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 error δ

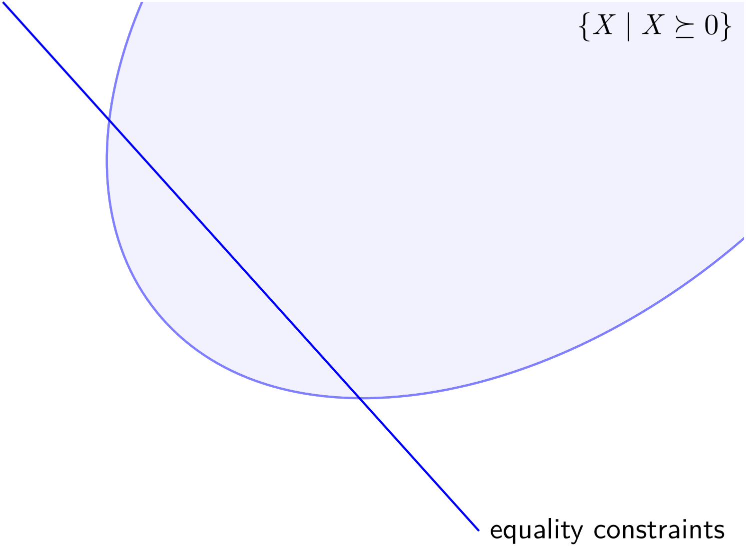
$$p = z^T Q\, z + z^T E\, z, \qquad\qquad \forall i\, j, |E_{i,j}| \leq \delta.$$

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

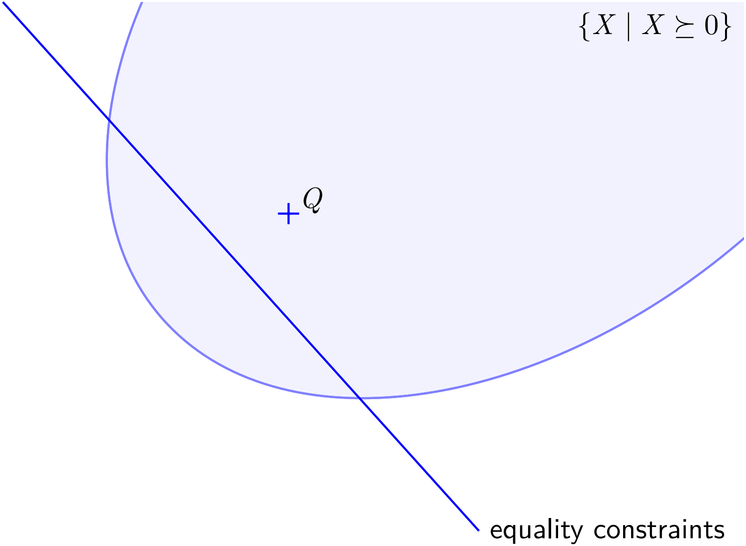
- Hence the validation method: given $p \simeq z^T Q\, z$
  1. Check that all monomials of p are in $z\, z^T$.
  2. Bound difference δ between coefficients of p and $z^T Q\, z$.
  3. If $Q - s\, \delta\, I \succeq 0$ ($s :=$ size of Q), then p is proved SOS.
- 2 can be done with interval arithmetic and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

$\Rightarrow$ Efficient validation method using just floats.

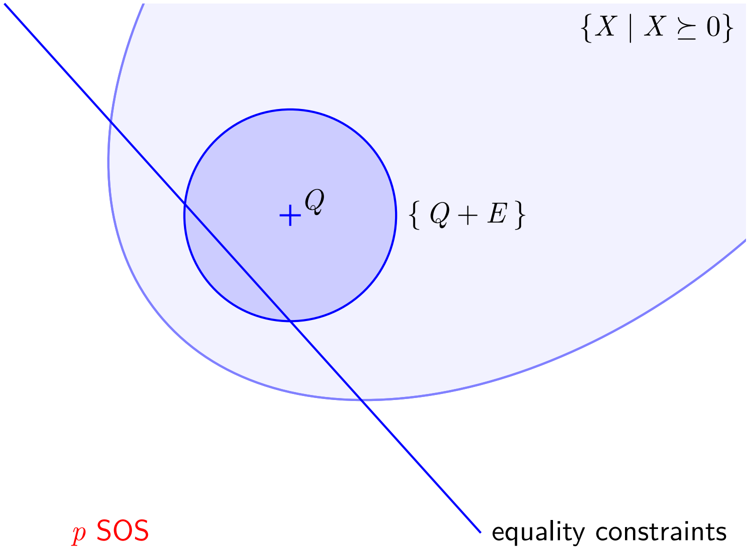
Intuitively



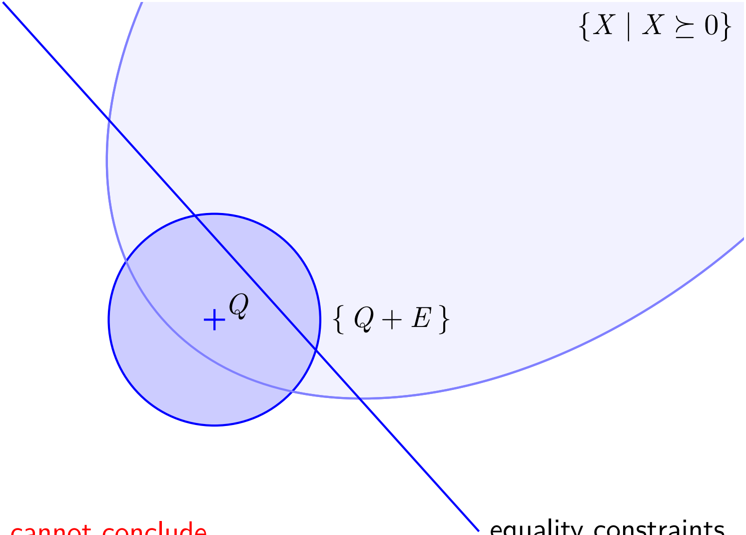
Intuitively



Intuitively

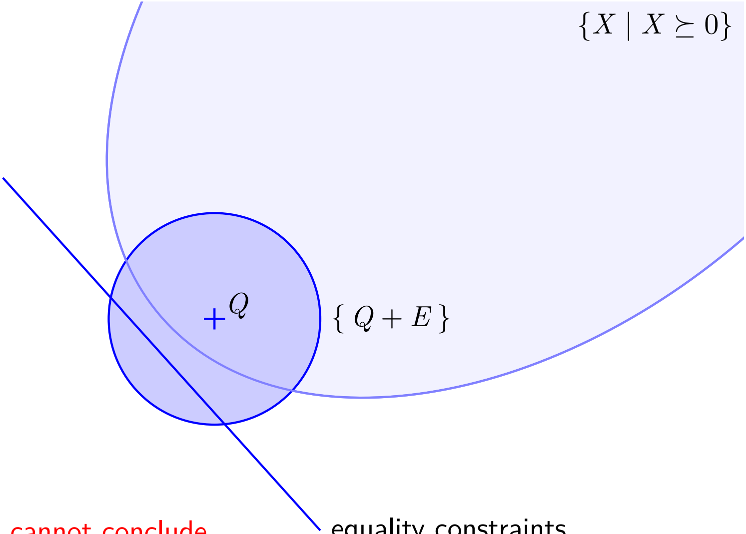


Intuitively



cannot conclude

Intuitively



cannot conclude

Cholesky Decomposition

- To prove that $a \in \mathbb{R}$ is non negative,
  we can exhibit r such that $a = r^2$ (typically $r = \sqrt{a}$).

Cholesky Decomposition

- To prove that $a \in \mathbb{R}$ is non negative, we can exhibit r such that $a = r^2$ (typically $r = \sqrt{a}$).
- To prove that a matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite we can similarly expose R such that $A = R^T R$ (since $x^T \left(R^T R\right) x = (Rx)^T (Rx) = \|Rx\|_2^2 \geq 0$).

Cholesky Decomposition

- To prove that $a \in \mathbb{R}$ is non negative, we can exhibit r such that $a = r^2$ (typically $r = \sqrt{a}$).
- To prove that a matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite we can similarly expose R such that $A = R^T R$ (since $x^T \left(R^T R\right) x = (Rx)^T (Rx) = \|Rx\|_2^2 \geq 0$).
- The Cholesky decomposition computes such a matrix R:

$$
\begin{array}{l}
R \;\mathbf{:=}\; 0; \\
\textbf{for } j \textbf{ from } 1 \textbf{ to } n \textbf{ do} \\
\quad \textbf{for } i \textbf{ from } 1 \textbf{ to } j-1 \textbf{ do} \\
\qquad R_{i,j} \;\mathbf{:=}\; \left(A_{i,j} - \sum_{k=1}^{i-1} R_{k,i} R_{k,j}\right) / R_{i,i}; \\
\quad \textbf{od} \\
\quad R_{j,j} \;\mathbf{:=}\; \sqrt{M_{j,j} - \sum_{k=1}^{j-1} {R_{k,j}}^2}; \\
\textbf{od}
\end{array}
$$

Cholesky Decomposition (end)

With rounding errors $A \neq R^T R$, Cholesky can succeed while $A \not\succeq 0$.

Cholesky Decomposition (end)

With rounding errors $A \neq R^T R$, Cholesky can succeed while $A \not\succeq 0$.

But error is bounded and for some (tiny) $c \in \mathbb{R}$:
if Cholesky succeeds on A then $A + c\,I \succeq 0$.

Hence:

Theorem

If Cholesky succeeds on $A - c\,I$ then $A \succeq 0$

holds for any $c \geq \frac{(s+1)\varepsilon}{1-(2s+2)\varepsilon}\operatorname{tr}(A) + 4(s+1)\left(2(s+2) + \max_i(A_{i,i})\right)\eta$

(ε and η relative and absolute precision of floating-point format).

Proved in Coq (paper proof: 6 pages, Coq: 5.1 kloc)

Agenda

Outline of the formalization

Outline of the formalization

1. Effective multivariate polynomials
   - CoqEAL [Cano, Cohen, Dénès, Mörtberg, Rouhling, Siles]

   $\leadsto$ uses SSReflect and MathComp [Gonthier et al.]
   - proof: SsrMultinomials [Strub]
   - implem.: FMapAVL from Coq stdlib
   - coefficients: $\mathbb{Q}$ as `bigQ` from Coq stdlib

Outline of the formalization

1. Effective multivariate polynomials
   - CoqEAL [Cano, Cohen, Dénès, Mörtberg, Rouhling, Siles]

   $\rightsquigarrow$ uses SSReflect and MathComp [Gonthier et al.]
   - proof: SsrMultinomials [Strub]
   - implem.: FMapAVL from Coq stdlib
   - coefficients: $\mathbb{Q}$ as `bigQ` from Coq stdlib
2. Effective check for positive definite matrices
   - CoqEAL
   - proof: previous work
   - implem.: lists of lists, CoqEAL
   - coefficients: floating-point numbers from CoqInterval [Melquiond]

Outline of the formalization

1. Effective multivariate polynomials
   - CoqEAL [Cano, Cohen, Dénès, Mörtberg, Rouhling, Siles]

   $\rightsquigarrow$ uses SSReflect and MathComp [Gonthier et al.]
   - proof: SsrMultinomials [Strub]
   - implem.: FMapAVL from Coq stdlib
   - coefficients: $\mathbb{Q}$ as `bigQ` from Coq stdlib
2. Effective check for positive definite matrices
   - CoqEAL
   - proof: previous work
   - implem.: lists of lists, CoqEAL
   - coefficients: floating-point numbers from CoqInterval [Melquiond]
3. Reflexive tactic
   - OCaml code as a wrapper for SDP solvers
   - Some Ltac code

Refinement proofs: overview of CoqEAL's methodology

1. Implement algorithms in a general way
(polymorphic functions; type classes)

Refinement proofs: overview of CoqEAL's methodology

1. Implement algorithms in a general way
(polymorphic functions; type classes)
2. Specialize the algorithms with proof-oriented datatypes;
correctness proof w.r.t a specification

$$\begin{array}{c} x : A \\ \Big\downarrow {\scriptstyle g_A} \\ g_A\ x \end{array}$$

Refinement proofs: overview of CoqEAL's methodology

1. Implement algorithms in a general way
   (polymorphic functions; type classes)
2. Specialize the algorithms with proof-oriented datatypes;
   correctness proof w.r.t a specification
   (or w.r.t another algorithm $\rightsquigarrow$ program refinement)

$$\begin{tikzcd} x \arrow[d, "f"'] & = & x : A \arrow[d, "g_A"] \\ f\,x & = & g_A\,x \end{tikzcd}$$

Refinement proofs: overview of CoqEAL's methodology

1. Implement algorithms in a general way
   (polymorphic functions; type classes)
2. Specialize the algorithms with proof-oriented datatypes;
   correctness proof w.r.t a specification
   (or w.r.t another algorithm $\rightsquigarrow$ program refinement)
3. Specialize the algorithms with effective datatypes;

$$\begin{tikzcd} x \arrow[d, "f"'] & = & x : A \arrow[d, "g_A"] & & c : C \arrow[d, "g_C"] \\ f\,x & = & g_A\,x & & g_C\,c \end{tikzcd}$$

Refinement proofs: overview of CoqEAL's methodology

1. Implement algorithms in a general way
   (polymorphic functions; type classes)
2. Specialize the algorithms with proof-oriented datatypes;
   correctness proof w.r.t a specification
   (or w.r.t another algorithm $\rightsquigarrow$ program refinement)
3. Specialize the algorithms with effective datatypes;
   correctness proof w.r.t proof-oriented version ($\rightsquigarrow$ data refinement)

$$
\begin{tikzcd}
x \arrow[d, "f"'] & = & x : A \arrow[d, "g_A"] \arrow[r, leftrightarrow, "\text{refines}"] & c : C \arrow[d, "g_C"] \\
f\,x & = & g_A\,x \arrow[r, leftrightarrow, "\text{refines}"'] & g_C\,c
\end{tikzcd}
$$

Effective multivariate polynomials

- Implemented in a modular way:

```
Definition seqmultinom := list N.
Module MultinomOrd <: OrderedType.
  Definition t := seqmultinom. (*...*) End MultinomOrd.
Module FMapMultipoly (M : Sfun MultinomOrd).
  Definition effmpoly := M.t. (*...*) End FMapMultipoly.
Module M := FMapAVL.Make MultinomOrd.
Module PolyAVL := FMapMultipoly M.
```

Effective multivariate polynomials

- Implemented in a modular way:

```
Definition seqmultinom := list N.
Module MultinomOrd <: OrderedType.
  Definition t := seqmultinom. (*...*) End MultinomOrd.
Module FMapMultipoly (M : Sfun MultinomOrd).
  Definition effmpoly := M.t. (*...*) End FMapMultipoly.
Module M := FMapAVL.Make MultinomOrd.
Module PolyAVL := FMapMultipoly M.
```

- Main refinement predicates:

```
Rseqmultinom : ∀(n : nat), multinom n → seqmultinom → Type
Reffmpoly : ∀(T : ringType) (n : nat), mpoly n T → effmpoly T → Type
ReffmpolyC : ∀(A : ringType) (C : Type), (A → C → Type) →
    ∀(n : nat), mpoly n A → effmpoly C → Type
```

Effective multivariate polynomials

- Implemented in a modular way:

```
Definition seqmultinom := list N.
Module MultinomOrd <: OrderedType.
  Definition t := seqmultinom. (*...*) End MultinomOrd.
Module FMapMultipoly (M : Sfun MultinomOrd).
  Definition effmpoly := M.t. (*...*) End FMapMultipoly.
Module M := FMapAVL.Make MultinomOrd.
Module PolyAVL := FMapMultipoly M.
```

- Main refinement predicates:

```
Rseqmultinom : ∀(n : nat), multinom n → seqmultinom → Type
Reffmpoly : ∀(T : ringType) (n : nat), mpoly n T → effmpoly T → Type
ReffmpolyC : ∀(A : ringType) (C : Type), (A → C → Type) →
    ∀(n : nat), mpoly n A → effmpoly C → Type
```

- Proof-oriented type for coefficients: needs a `ringType` structure; instantiated with MathComp's `rat`. Effective counterpart: `bigQ`.

Positive definiteness check for floating-point matrices

```
Definition posdef (n : nat) (A : 'M[R]_n) :=
  ∀(x : 'cV[R]_n), x ≠ 0 → 0 < xᵀ×A×x. (* pointwise < *)
```

Positive definiteness check for floating-point matrices

```
Definition posdef (n : nat) (A : 'M[R]_n) :=
  ∀(x : 'cV[R]_n), x ≠ 0 → 0 < xᵀ×A×x. (* pointwise < *)

posdef_check : ∀(mx : Type → nat → nat → Type)
                (T : Type) (n : nat),
                (*...type classes...→ *)
                 mx T n n → bool
```

Positive definiteness check for floating-point matrices

```
Definition posdef (n : nat) (A : 'M[R]_n) :=
  ∀(x : 'cV[R]_n), x ≠ 0 → 0 < xᵀ×A×x. (* pointwise < *)

posdef_check : ∀(mx : Type → nat → nat → Type)
                (T : Type) (n : nat),
                (*...type classes...→ *)
                 mx T n n → bool
```

- Correctness: use formal proof of Cholesky algo over $\mathbb{R}$ (previous work)

Positive definiteness check for floating-point matrices

```
Definition posdef (n : nat) (A : 'M[R]_n) :=
  ∀(x : 'cV[R]_n), x ≠ 0 → 0 < x^T×A×x. (* pointwise < *)

posdef_check : ∀(mx : Type → nat → nat → Type)
                (T : Type) (n : nat),
                (*...type classes...→ *)
                 mx T n n → bool
```

- Correctness: use formal proof of Cholesky algo over $\mathbb{R}$ (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones

Positive definiteness check for floating-point matrices

```
Definition posdef (n : nat) (A : 'M[R]_n) :=
  ∀(x : 'cV[R]_n), x ≠ 0 → 0 < x^T×A×x. (* pointwise < *)

posdef_check : ∀(mx : Type → nat → nat → Type)
                (T : Type) (n : nat),
                (*...type classes...→ *)
                 mx T n n → bool
```

- Correctness: use formal proof of Cholesky algo over $\mathbb{R}$ (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones
- Refinement 2: refine real coefficients with floating-point ones:

Positive definiteness check for floating-point matrices

```
Definition posdef (n : nat) (A : 'M[R]_n) :=
  ∀ (x : 'cV[R]_n), x ≠ 0 → 0 < xᵀ×A×x. (* pointwise < *)

posdef_check : ∀ (mx : Type → nat → nat → Type)
                 (T : Type) (n : nat),
                 (*...type classes...→ *)
                  mx T n n → bool
```

- Correctness: use formal proof of Cholesky algo over $\mathbb{R}$ (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones
- Refinement 2: refine real coefficients with floating-point ones:
  - Rely on `Float_infnan_spec`

Positive definiteness check for floating-point matrices

```
Definition posdef (n : nat) (A : 'M[R]_n) :=
  ∀ (x : 'cV[R]_n), x ≠ 0 → 0 < xᵀ×A×x. (* pointwise < *)

posdef_check : ∀ (mx : Type → nat → nat → Type)
                 (T : Type) (n : nat),
                 (*...type classes...→ *)
                  mx T n n → bool
```

- Correctness: use formal proof of Cholesky algo over $\mathbb{R}$ (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones
- Refinement 2: refine real coefficients with floating-point ones:
  - Rely on `Float_infnan_spec`
  - = formalization of the floating-point “standard model” (previous work)

Positive definiteness check for floating-point matrices

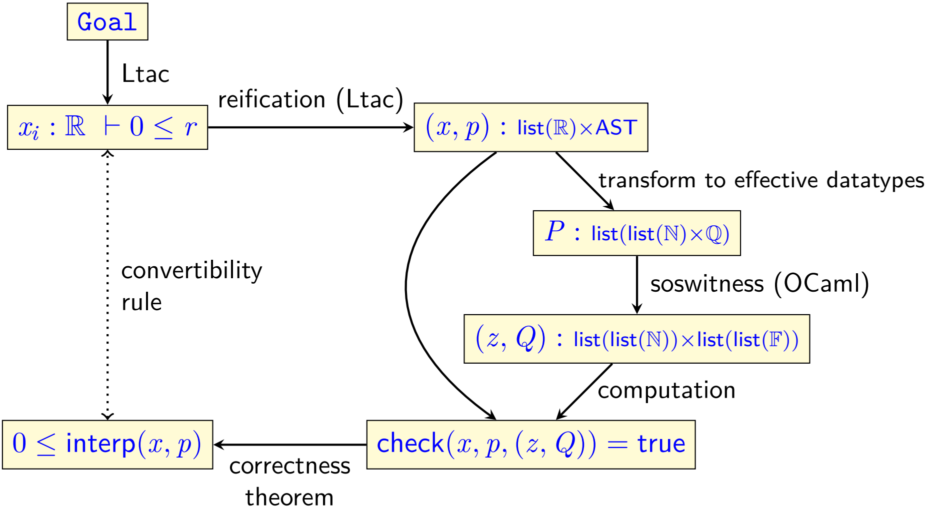
```
Definition posdef (n : nat) (A : 'M[R]_n) :=
  ∀ (x : 'cV[R]_n), x ≠ 0 → 0 < xᵀ×A×x. (* pointwise < *)

posdef_check : ∀ (mx : Type → nat → nat → Type)
                 (T : Type) (n : nat),
                 (*...type classes...→ *)
                  mx T n n → bool
```

- Correctness: use formal proof of Cholesky algo over $\mathbb{R}$ (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones
- Refinement 2: refine real coefficients with floating-point ones:
  - Rely on `Float_infnan_spec`

  = formalization of the floating-point "standard model" (previous work)

  → instantiated with CoqInterval's floating-point implementation, restricted to 53 bits.

The validsdp tactic (1/3) – the big picture



The validsdp tactic (2/3) – OCaml code

- Rely on the OSDP lib. (OCaml interface for off-the-shelf SDP solvers)
- Implement a Coq plugin (the `ValidSDP.soswitness` OCaml module provides a `soswitness` tactic that consists of a wrapper for OSDP)

OSDP library: 6.2 kloc of OCaml code + 1.2 kloc of C code.

`ValidSDP.soswitness` plugin: 0.3 kloc of OCaml code.

The validsdp tactic (3/3) – correctness theorem

```
Theorem soscheck_eff_wrapup_correct :
  ∀ (x : list R) (p : p_abstr_poly)
    (zQ : list (list N) * list (list (s_float bigZ bigZ))),
  soscheck_eff_wrapup x p zQ = true →
  (0 ≤ interp_p_abstr_poly x p)%R.
```

Coq: 2.0 kloc for the main tactic and proofs + 6.5 kloc of refinement proofs
(Cholesky: 3.0 kloc; FP arith: 1.3 kloc; multipoly: 2.2 kloc)

Agenda

Benchmarks (1/3)

- In the paper : OSDP 0.4.5 & ValidSDP 0.3
- In this presentation : OSDP 0.5.2 & ValidSDP version d60c663

- In both configurations: Coq 8.5.2, MathComp 1.6, Flocq 2.5.1, Coquelicot 2.1.1, CoqInterval 3.1.0, and dev. version of other libs

- Setup:
  - A desktop PC under Debian GNU/Linux Jessie
  - Core i5-4460S CPU clocked at 2.9 GHz
  - All timings are total elapsed time (in seconds)
  - Timeout of 900s

Benchmarks (2/3)

Problem	n	d	OSDP (not verified)	MonniauxC11 (not verified)	NLCertify (not verified)	ValidSDP	PVS/Bernstein	NLCertify	HOL Light/ Taylor
adaptativeLV	4	4	**0.75**	2.67	1.12	5.16	14.93	**2.61**	12.31
butcher	6	4	1.58	—	**1.05**	9.40	48.44	**8.36**	15.62
caprasse	4	4	**0.41**	1.82	0.88	**5.19**	25.89	2.63	17.68
heart	8	4	**3.18**	268.75	—	**16.67**	131.13	—	26.15
magnetism	7	2	**1.11**	2.04	1.64	**5.18**	245.52	14.50	16.07
reaction	3	2	**0.81**	1.56	0.24	4.33	11.48	**1.96**	12.41
schwefel	3	4	**0.95**	2.45	2.76	**3.70**	14.72	56.13	17.46
fs260	6	4	**1.25**	—	—	**5.99**	—	—	—
fs461	6	4	**0.70**	11.18	0.87	**5.18**	621.06	7.46	22.70
fs491	6	4	**0.54**	21.81	—	**5.38**	—	—	—
fs745	6	4	0.98	11.74	**0.94**	**5.55**	623.17	6.90	22.48
fs752	6	2	**0.35**	1.81	0.90	**3.80**	54.52	7.88	13.34
fs8	6	2	**0.43**	1.53	1.48	**3.93**	52.63	6.62	13.40
fs859	6	8	—	—	—	—	—	—	—
fs860	6	4	**1.21**	10.53	1.11	**6.08**	73.65	7.34	14.28
fs861	6	4	**1.09**	10.48	1.20	**5.15**	69.74	7.87	14.28
fs862	6	4	1.27	79.25	**1.25**	**5.37**	73.54	7.58	14.14
fs863	6	2	**0.94**	1.50	—	**3.85**	—	—	13.85
fs864	6	2	**0.56**	2.05	—	**4.05**	—	—	13.28
fs865	6	2	**0.76**	2.11	—	**3.68**	—	—	13.76
fs867	6	2	**0.21**	2.09	1.74	**4.22**	—	8.04	—

Benchmarks (3/3)

Problem	n	d	OSDP (not verified)	MonniauxC11 (not verified)	NLCertify (not verified)	ValidSDP	PVS/Bernstein	NLCertify	HOL Light/Taylor
fs868	6	4	**0.94**	—	—	**6.05**	—	—	—
fs884	6	4	—	—	—	—	—	—	—
fs890	6	4	—	**7.78**	—	—	—	—	—
ex4_d4	2	12	—	—	—	—	—	—	—
ex4_d6	2	18	—	—	—	—	—	—	—
ex4_d8	2	24	**16.99**	—	—	**82.89**	—	—	—
ex4_d10	2	30	—	—	—	—	—	—	—
ex5_d4	3	8	**1.67**	—	—	**13.63**	—	—	—
ex5_d6	3	12	**16.10**	—	—	**66.82**	—	—	—
ex5_d8	3	16	**203.06**	—	—	**353.70**	—	—	—
ex5_d10	3	20	—	—	—	—	—	—	—
ex6_d4	4	8	**16.82**	—	—	**44.99**	—	—	—
ex6_d6	4	12	—	—	—	—	—	—	—
ex7_d4	2	12	—	—	—	—	—	—	—
ex7_d6	2	18	**1.50**	—	—	**26.78**	—	—	—
ex7_d8	2	24	**15.38**	—	—	**83.47**	—	—	—
ex7_d10	2	30	—	—	—	—	—	—	—
ex8_d4	2	8	**0.87**	15.72	—	**7.52**	—	—	—
ex8_d6	2	12	—	—	—	—	—	—	—
ex8_d8	2	16	—	—	—	—	—	—	—
ex8_d10	2	20	—	—	—	—	—	—	—

Agenda

Conclusion

- Context: formal proof of multivariate polynomial positivity
- A Coq reflexive tactic
  - Input: polynomial ineq. goals with real variables and rational constants
  - Use off-the-shelf SDP solvers as untrusted oracles
  - Numerical approach with formal floating-point arithmetic
  - Algorithm involving matrices (Cholesky)

Discussion and open questions

- Bottleneck 1:
  emulate floating-point arithmetic in Coq (1000x overhead)
- Bottleneck 2: we currently use `vm_compute`.
  `native_compute` issue with large terms (witnesses for large problems)
- Coq 8.6 comes with LtacProf $\rightsquigarrow$ feasibility of a '`vm_compute` profiler'?
- Possible extension of ValidSDP: add support of elementary functions

Questions

Thanks for your attention!

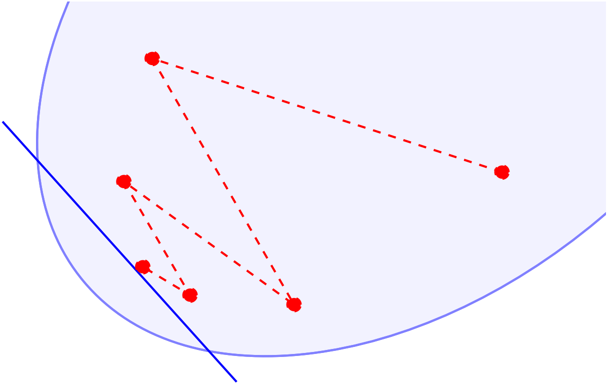
?

https://sourcesup.renater.fr/validsdp/

Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

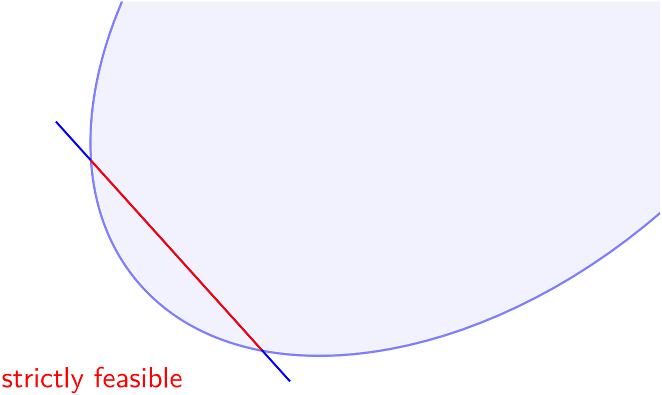
- inexact termination



Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

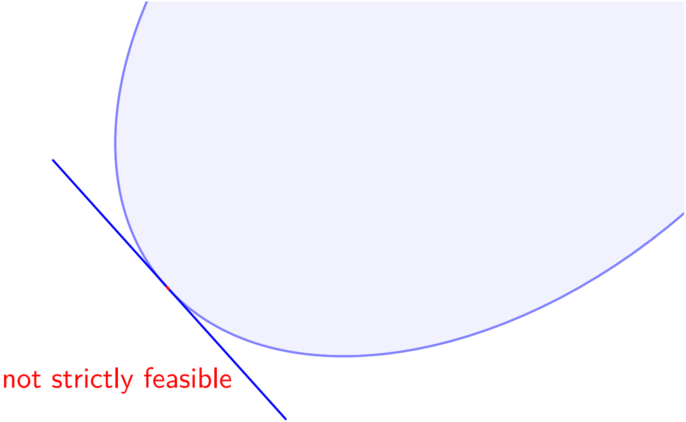
- inexact termination
- failure of strict feasibility



Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

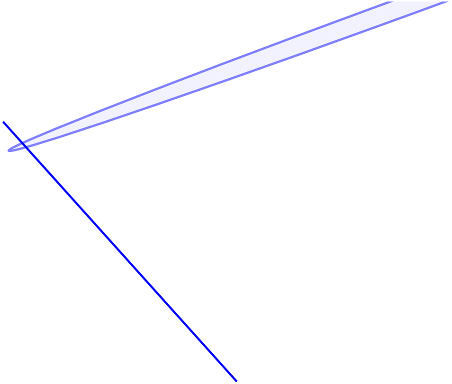
- inexact termination
- failure of strict feasibility



Inaccuracy in Solving SDPs

SDP solvers only yield approximate solutions due to

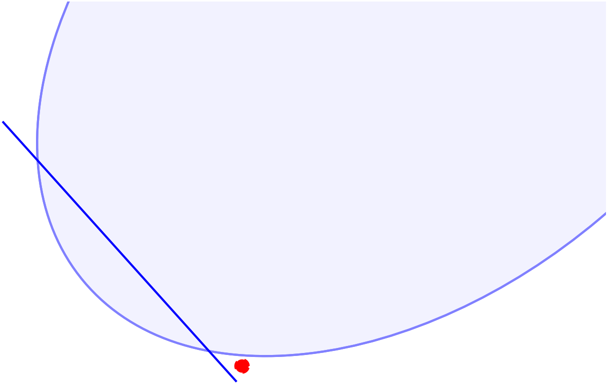
- inexact termination
- failure of strict feasibility
- ill conditioning



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).

Positivstellensatz

We want to prove that

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

is not satisfiable.

Positivstellensatz

We want to prove that

$$p_1(x_1, \dots, x_n) \geq 0 \wedge \dots \wedge p_m(x_1, \dots, x_n) \geq 0$$

is not satisfiable.

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

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

Positivstellensatz

We want to prove that

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

is not satisfiable.

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

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

- equivalence under hypotheses (Putinar's Positivstellensatz)
- no practical bound on degrees of $r_i \Rightarrow$ will be arbitrarily fixed