# Sum-of-Squares Programming in Julia with JuMP

Benoît Legat*, Chris Coey†, Robin Deits†, Joey Huchette† and Amelia Perry†

* UCLouvain, † MIT

## Sum-of-Squares (SOS) Programming

### Nonnegative quadratic forms into sum of squares

$(x_1, x_2, x_3)$ unique

$$p(x) = x^\top Q x$$

$$x_1^2 + 2x_1x_2 + 5x_2^2 + 4x_2x_3 + x_3^2 = x^\top \begin{pmatrix} 1 & 1 & 0 \\ 1 & 5 & 2 \\ 0 & 2 & 1 \end{pmatrix} x$$

$$p(x) \geq 0 \ \forall x \iff Q \succeq 0$$

cholesky

$$(x_1 + x_2)^2 + (2x_2 + x_3)^2 \longleftarrow x^\top \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix}^\top \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix} x$$

### Nonnegative polynomial into sum of squares

$(x_1, x_2, x_3)$ $(x_1, x_1x_2, x_2)$ *not* unique

$$p(x) = X^\top Q X$$

$$\begin{gathered} x_1^2 + 2x_1^2x_2 + 5x_1^2x_2^2 \\ +4x_1x_2^2 + x_2^2 \end{gathered} = X^\top \begin{pmatrix} 1 & 1 & 0 \\ 1 & 5 & 2 \\ 0 & 2 & 1 \end{pmatrix} X$$

$$p(x) \geq 0 \ \forall x \Longleftarrow Q \succeq 0$$

cholesky

$$\begin{gathered} (x_1 + x_1x_2)^2 + \\ (2x_1x_2 + x_2)^2 \end{gathered} \longleftarrow X^\top \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix}^\top \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix} X$$

**When is nonnegativity equivalent to sum of squares ?**

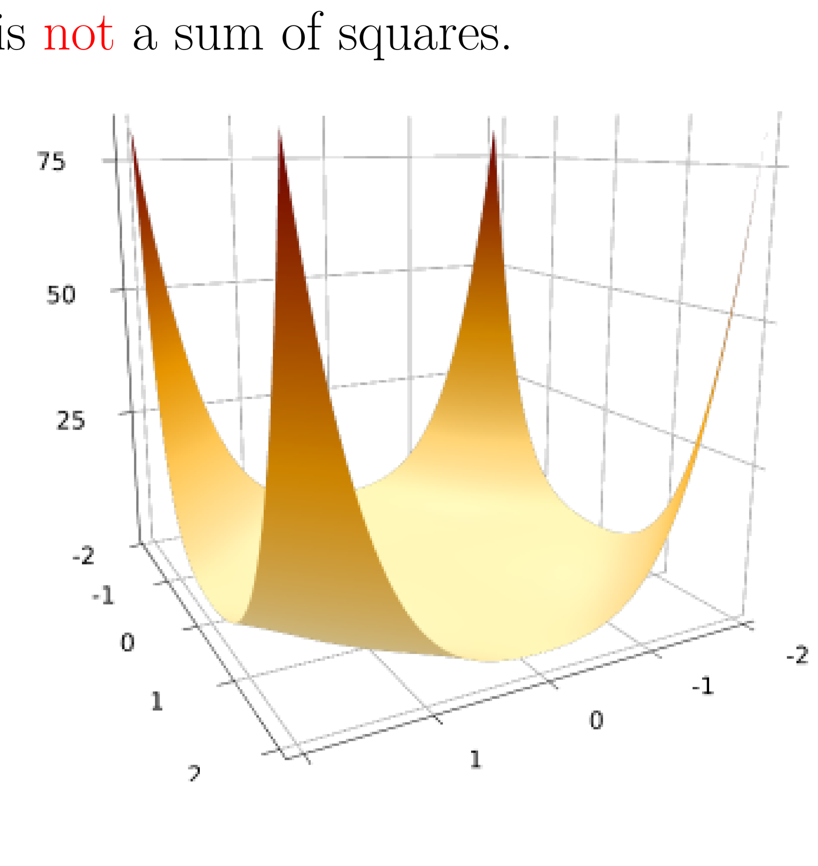
Determining whether a polynomial is nonnegative is NP-hard.

**Hilbert 1888** Nonnegativity of $p(x)$ of $n$ variables and degree $2d$ is equivalent to sum of squares in the following three cases:
- $n = 1$ : Univariate polynomials
- $2d = 2$ : Quadratic polynomials
- $n = 2, 2d = 4$ : Bivariate quartics

**Motzkin 1967** First explicit example:

$$x_1^4x_2^2 + x_1^2x_2^4 + 1 - 3x_1^2x_2^2 \geq 0 \quad \forall x$$

but is not a sum of squares.




## Manipulating Polynomials

Two implementations: `TypedPolynomials.jl` and `DynamicPolynomials.jl`.
One common independent interface: `MultivariatePolynomials.jl`.

```
@polyvar y # one variable
@polyvar x[1:2] # tuple/vector
```

Build a vector of monomials:
- $(x_1^2, x_1x_2, x_2^2)$:
  `X = monomials(x, 2)`
- $(x_1^2, x_1x_2, x_2^2, x_1, x_2, 1)$:
  `X = monomials(x, 0:2)`

## Polynomial variables

By hand, with an integer decision variable `a` and real decision variable `b`:

```
@variable(model, a, Int)
@variable(model, b)
p = a*x^2 + (a+b)*y^2*x + b*y^3
```

From a polynomial basis, e.g. the *scaled monomial* basis, with integer decision variables as coefficients:

```
@variable(model,
          Poly(ScaledMonomialBasis(X)),
          Int)
```

## Polynomial constraints

Constrain $p(x, y) \geq q(x, y)\ \forall x, y$ such that $x \geq 0, y \geq 0, x + y \geq 1$ using the scaled monomial basis:

```
S = @set x >= 0 && y >= 0 && x + y >= 1
@constraint(model, p >= q, domain = S,
            basis = ScaledMonomialBasis)
```

Interpreted as:

```
@constraint(model, p - q in SOSCone(),
            domain = S,
            basis = ScaledMonomialBasis)
```

To use DSOS or SDSOS (Ahmadi, Majumdar 2017):

```
@constraint(model, p - q in DSOSCone())
@constraint(model, p - q in SDSOSCone())
```

### SOS on algebraic domain

The domain $S$ is defined by equalities forming an *algebraic variety* $V$ and inequalities $q_i$. We search for Sum-of-Squares polynomials $s_i$ such that

$$p(x) - q(x) \equiv s_0(x) + s_1(x)q_1(x) + \cdots \pmod{V}$$

The Gröbner basis of $V$ is computed the equation is reduced modulo $V$.

## Dual value

The dual of the constraint is a positive semidefinite (PSD) matrix of moments $\mu$. The `extractatoms` function attempts to find an *atomic* measure with these moments by solving an algebraic system.

## Sum-of-Squares extension

### `MathOptInterface.jl` (MOI)

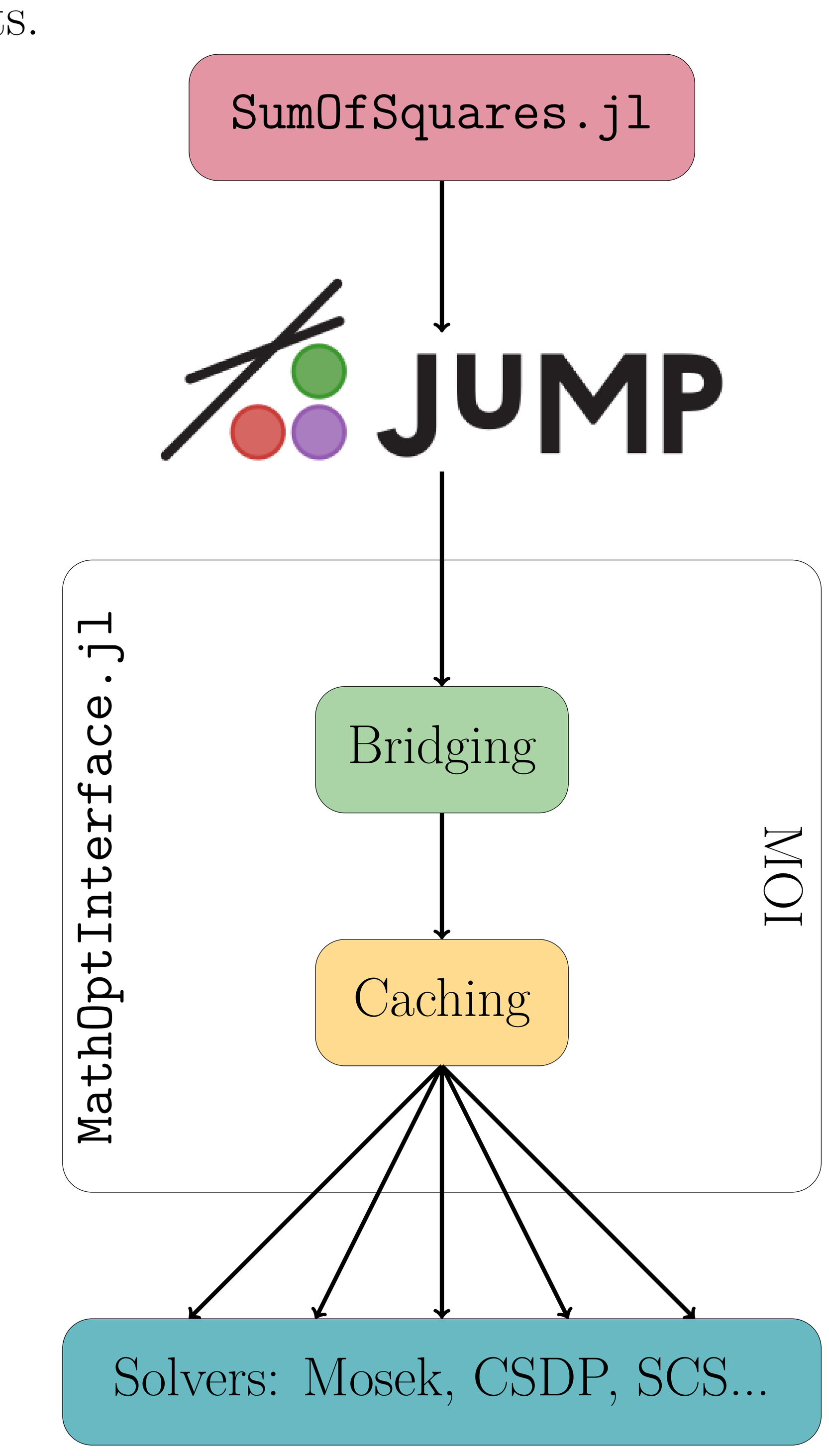
MOI is an abstraction layer for mathematical optimization solvers. A constraint is defined by a "function" ∈ "set" pair.

MOI extension: `AbstractVectorFunction` ∈ `SOS(X)` (resp. `WSOS(X)`): SOS constraint without (resp. with) `domain` equipped with a bridge to `AbstractVectorFunction` ∈ `PSD` (resp. `SOS(X)`).

### JuMP

JuMP is a domain-specific modeling language for mathematical optimization. It stores the problem directly (a cache can optionally be used) in the solver using MOI.

JuMP extension: $p(x) \geq q(x)$ and $p(x) \in$ `SOS()` are rewritten into MOI SOS or WSOS constraints, e.g. $x^2 + y^2 \geq 2xy$ is rewritten into $[1, -2, 1] \in \mathrm{SOS}(x^2, xy, y^2)$. $p(x) \in$ `DSOS()` (resp. `SDSOS()`) is rewritten into linear (resp. second-order cone) constraints.




**Bridging** Automatic reformulation of a constraint into an equivalent form supported by the solver, e.g. quadratic constraint into second-order cone constraint. In particular, reformulates SOS/WSOS constraints into PSD constraints. An interior-point solver that natively supports SOS and WSOS without reformulation to SDP using the approach of (Papp, Yıldız 2017) is under development.

**Caching** Cache of the problem data in case the solver do not support a modification (can be disabled). For instance, Mosek provides many modification capabilities in the API but CSDP only support pre-allocating and then loading the whole problem at once.