

JuMP extension to Sum-of-Squares Programming

Benoît Legat (UCLouvain)

Joint work with:

Chris Coey, Robin Deits, Joey Huchette, Amelia Perry and Tillmann Weisser

29 octobre 2026

UCLouvain

Massachusetts Institute of Technology (MIT)

Rice University

Los Alamos National Laboratory (LANL)

Nonnegative quadratic forms into sum of squares

$$(x_1, x_2, x_3) \quad p(x) = x^T Q x \quad \text{unique}$$

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

$$p(x) \geq 0 \quad \forall x \iff Q \geq 0 \quad \downarrow \text{cholesky}$$

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

Nonnegative polynomial into sum of squares

$$(x_1, x_2, x_3) \quad (x_1, x_1 x_2, x_2) \quad \text{not unique}$$
$$p(x) = X^T Q X$$

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

$$p(x) \geq 0 \quad \forall x \iff Q \geq 0 \quad \downarrow \text{cholesky}$$

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

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 :

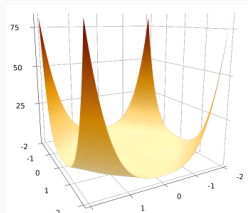
- $n = 1$: Univariate polynomials
- $2d = 2$: Quadratic polynomials
- $n = 2, 2d = 4$: Bivariate quartics

Motzkin 1967

First **explicit** example :

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

but is **not** a sum of squares.



Semidefinite programming

$$\underset{Q \in \mathcal{S}^n}{\text{minimize}} \quad \langle C, Q \rangle$$

$$\text{subject to} \quad \langle A_i, Q \rangle = b_i$$

$$Q \succeq 0$$

$$\underset{y \in \mathbb{R}^n}{\text{maximize}} \quad \langle b, y \rangle$$

$$\text{subject to} \quad \sum_i A_i y_i \preceq C$$

File format : SDPA

Solvers : CSDP, SDPA, DSDP, SDPLR, ...

Variables : Q block diagonal, nonnegative scalar variables (1×1 blocks) or SDP matrices.

Constraints : Affine equations.

Conic Modelling

```
using JuMP
model = Model(...)
@variable(model, -1 <= x <= 1)
@variable(model, y)
@variable(model, z <= 0)
@constraint(model, [x + y x
                    y      x - y] in PSDCone())
@constraint(model, [x + y, z, y] in SecondOrderCone())
@objective(model, x^2 - 2x*z + z^2)
```

The gap between models and solvers

The solver interface should only support structures and the algorithm **exploits** :

- n solvers and m structures $\rightarrow mn$ transformations $\rightarrow$ unscalable for large m, n .
- enables evaluation of formulation quality, e.g. automatic transformation and automatic dualization.

The model should

- be **independent** from solvers.
- represent the structure **exploitable** by algorithms.
- allow representable structure **unknown** to solvers, e.g. Sum-of-Squares variables/constraints.

Bridging the gap

$$x \in S_1 \Leftrightarrow Ax \in S_2 \quad AS_1 = S_2$$

$$A^*y \in S_1^* \Leftrightarrow y \in S_2^* \quad S_1^* = A^*S_2^*$$

In Lagrangian :

$$\langle Ax, y \rangle_2 = \langle x, A^*y \rangle_1$$

Transformation of variable-in- S_2 to variable-in- S_1 .

Primal Transform value v to Av .

Dual Transform dual y to $A^{-*}y$.

Transformation of f -in- S_1 constraint to Af -in- S_2 constraint.

Primal Transform value v of Af to $A^{-1}v$ of f .

Dual Transform dual y of A^*y .

Examples

FlipSignBridge

- **Variable** $x \geq l$ substituted by $x = -y$ where $y \leq -l$.
- **Constraint** $a^\top x \leq \beta$ transformed into $-a^\top x \geq -\beta$.

VectorizeBridge

- **Variable** $x \geq l$ substituted by $x = y + l$ where $y \in \mathbb{R}_+^1$.
- **Constraint** $a^\top x \leq \beta$ transformed into $[a^\top x - \beta] \in \mathbb{R}_-^1$.

FreeBridge

- **Variable** $x \in \mathbb{R}$ substituted by $x = y + z$ where $y \in \mathbb{R}_+$ and $z \in \mathbb{R}_-$.

SlackBridge

- **Constraint** $f \in S$ transformed into $f = x$ for variable $x \in S$.

Selection of bridges

How to select bridges **automatically**?

Example

Free variable for SDP solver :

- FreeBridge : $x \in \mathbb{R} \rightarrow y \in \mathbb{R}_+$ (supported) and $z \in \mathbb{R}_-$ (**not** supported)
- FlipSignBridge : $x \in \mathbb{R}_- \rightarrow y \in \mathbb{R}_+$.

Shortest path?

Shortest path in directed Hypergraph

Nodes

Node for each set S (variable-in- S).

Node for each constraint F -in- S .

Types F and S are **not limited** to those defined in MOI.

Infinitely many nodes, we need to be **lazy**.

Edges

Each bridge defined possible **infinitely** many edges.

For each edge and ingoing node : outgoing nodes are

- variable-in- S created.
- constraints F -in- S created.

Solved by a modified **Bellman-Ford** algorithm.

Sum-of-Squares bridges

Polynomial $p \in \Sigma$ (p is SOS) iff $p = X^T Q x$ with $Q \in \mathbb{S}_+$ (Q is PSD). Hence $\Sigma = A\mathbb{S}_+$.

SOSPolynomialBridge : Transformation of variable-in- Σ to variable-in- $\mathbb{S}_+$.

Transformation of constraint F -in- Σ : SlackBridge + SOSPolynomialBridge.

Constraint Attribute

Examples : ConstraintPrimal, ConstraintDual, ConstraintFunction, ConstraintSet, ...

Redirected to bridge when constraint is bridged.

New attributes :

- GramMatrixAttribute : Gram matrix Q indexed by X .
- MomentMatrixAttribute : Moment matrix index by X , dual of constraint $Q \in \mathbb{S}_+$.
- MomentsAttribute : Vector of moments, dual of constraint $p = X^\top QX$.

Extending JuMP macros

```
@constraint(model, [x + 1, x - y] in MOI.Zeros())
```

Implementation :

```
function build_constraint(  
    _error::Function,  
    func::Vector{<:AbstractJuMPScalar},  
    set::MOI.AbstractVectorSet)  
    return VectorConstraint(x, set)  
end
```

Extending JuMP macros : Custom set

```
@constraint(model, [x + 1, x - y] in SecondOrderCone())
```

Implementation :

```
function build_constraint(_error::Function,  
                        f::AbstractVector,  
                        s::AbstractVectorSet)  
    set = moi_set(s, length(f))  
    return build_constraint(_error, f, set)  
end  
function moi_set(::SecondOrderCone, dim::Int)  
    return MOI.SecondOrderCone(dim)  
end
```

Extending JuMP macros : PSD cone

```
using LinearAlgebra # For Symmetric
@constraint(model, Symmetric([x + 1 x - y
                             x - y y]) in PSDCone())
```

Implementation :

```
function build_constraint(_error::Function,
                        Q::Symmetric,
                        ::PSDCone)
    n = LinearAlgebra.checksquare(Q)
    func = [Q[i, j] for j in 1:n for i in 1:j]
    set = MOI.PositiveSemidefiniteConeTriangle(n)
    VectorConstraint(func, set,
                    SymmetricMatrixShape(n))
end
```

Reshaping results

```
function reshape_vector(vectorized_form::Vector{T},
    shape::SymmetricMatrixShape) where T
    matrix = Matrix{T}(undef, shape.side_dimension,
        shape.side_dimension)

    k = 0
    for j in 1:shape.side_dimension
        for i in 1:j
            k += 1
            matrix[j, i] = matrix[i, j] =
                vectorized_form[k]
        end
    end
    return Symmetric(matrix)
end
```

Reshaping sets

```
function reshape_set(set::MOI.AbstractScalarSet,  
                    ::ScalarShape)  
    return set  
end  
function reshape_set(  
    ::MOI.PositiveSemidefiniteConeTriangle,  
    ::SymmetricMatrixShape  
)  
    return PSDCone()  
end
```

Sum-of-Squares constraint macro

```
@constraint(model, p in SOS Cone())
```

Implementation :

```
function JuMP.build_constraint(_error::Function, p,  
                              cone::SOS Cone; kws...)
    coefs = coefficients(p)
    monos = monomials(p)
    set = JuMP.moi_set(cone, monos; kws...)
    shape = PolyJuMP.PolynomialShape(monos)
    return PolyJuMP.bridgeable(  
        JuMP.VectorConstraint(coefs, set, shape),  
        JuMP.moi_function_type(typeof(coefs)),  
        typeof(set)  
    )
end
```

Reshaping polynomial results

```
function JuMP.reshape_set(set::SOSPolynomialSet,  
                           ::PolyJuMP.PolynomialShape)  
    return set.cone  
end  
function JuMP.reshape_vector(x::Vector,  
                             shape::PolynomialShape)  
    return polynomial(x, shape.monomials)  
end  
function JuMP.reshape_vector(x::Vector,  
                             shape::MomentsShape)  
    return measure(x, shape.monomials)  
end  
function JuMP.dual_shape(shape::PolynomialShape)  
    return MomentsShape(shape.monomials)  
end
```

- Change of polynomial basis.
- Symmetry reduction.
- Chordal sparsity.
- Non-commutative (done), hermitian, orthogonal, idempotent variables.
- Generalized Problem of Moments :
<https://github.com/lanl-ansi/MomentOpt.jl>.