# Set Programming with JuMP

Benoît Legat*, *In collaboration with*: Raphaël M. Jungers*, Pablo A. Parrilo† and Paulo Tabuada‡

* UCLouvain, † MIT, ‡ UCLA

# Set program

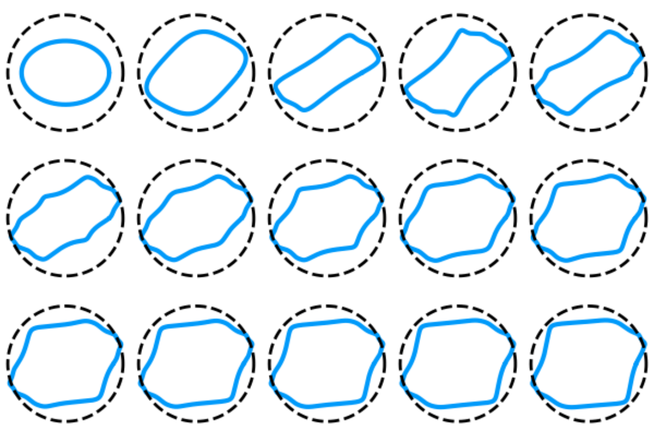
**Set inclusion**
Consider two sets $S \subseteq \mathbb{R}^n$, $T \subseteq \mathbb{R}^m$, matrices $A \in \mathbb{R}^{r \times n}$, $B \in \mathbb{R}^{r \times m}$:

$$AS \subseteq BT.$$

**Set program**
Given fixed sets $T_i$, find sets $S_i$:

$$\begin{aligned} \max_{S_i, x_i} \quad & f(\mathrm{vol}(S_1), \ldots, \mathrm{vol}(S_n)) \\ A_j S_{a_j} & \subseteq B_j S_{b_j} \\ S_i & \subseteq T_i \\ x_{c_j} & \in S_{d_j} \end{aligned}$$

# Stability of Hybrid Systems



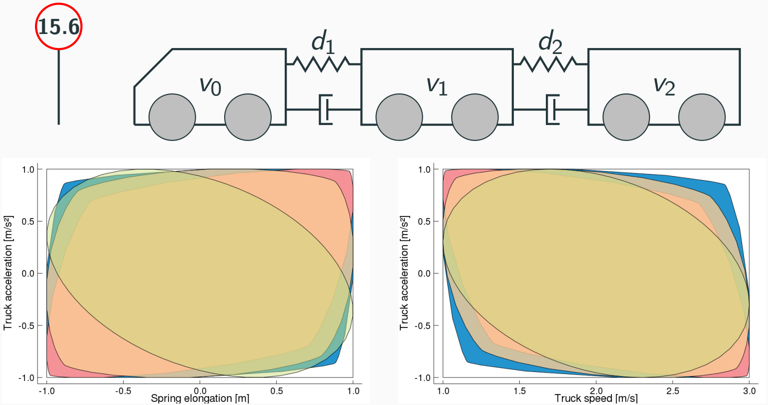
Instability may be certified using a low-rank infeasibility certificate.

See presentation at *19th ACM International Conference on Hybrid Systems: Computation and Control,* (HSCC), 2016.

# Information Theory

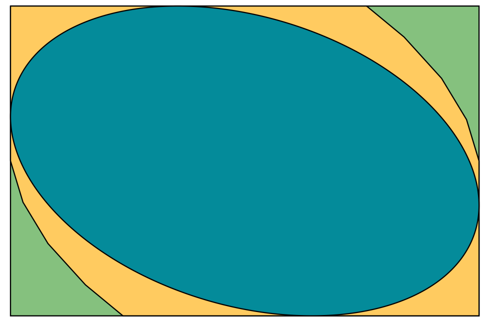
See presentation at *37rd Symposium on Information Theory in the Benelux*, 2016.

# Safe Model Predictive Control




See presentation at *IFAC Conference on Analysis and Design of Hybrid Systems* (ADHS), 2018.

# Safe Stochastic Programming



See presentation at *23rd International Symposium on Mathematical Programming* (ISMP), 2018.

# Example with ellipsoids: Model

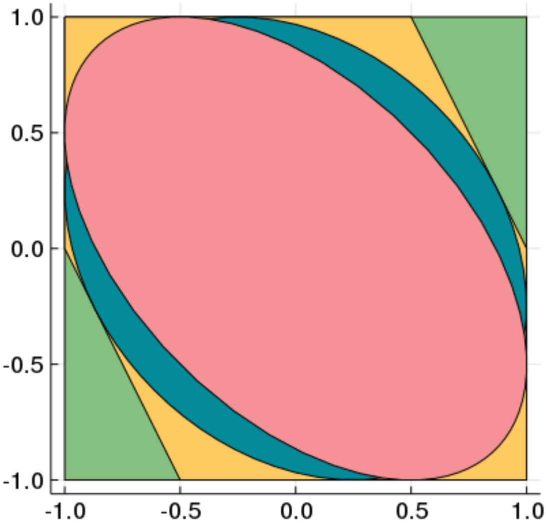
Maximal volume ellipsoid (determinant):

```
model = Model(...)
@variable(model, S, Ellipsoid(symmetric=true))
@constraint(model, S ⊆ □)
@constraint(model, A * S ⊆ E * S)
@objective(model, Max, nth_root(volume(S)))
@time JuMP.optimize!(model)
ell = JuMP.value(S)
```

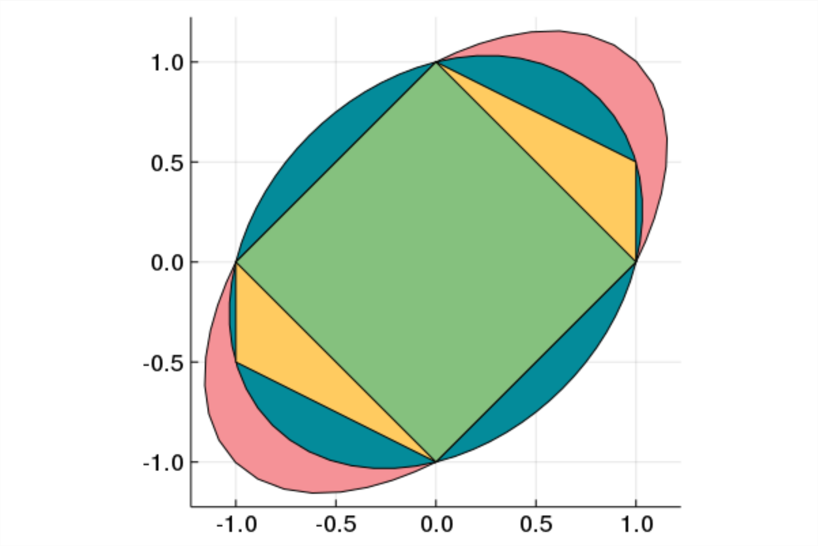
Maximal sum of the squares of the semi-axes of the ellipsoid (trace):

```
@objective(model, Max,
           L1_heuristic(volume(S), [1.0, 1.0]))
```

# Example with ellipsoids: Primal Solution



# Example with ellipsoids: Polar Solution

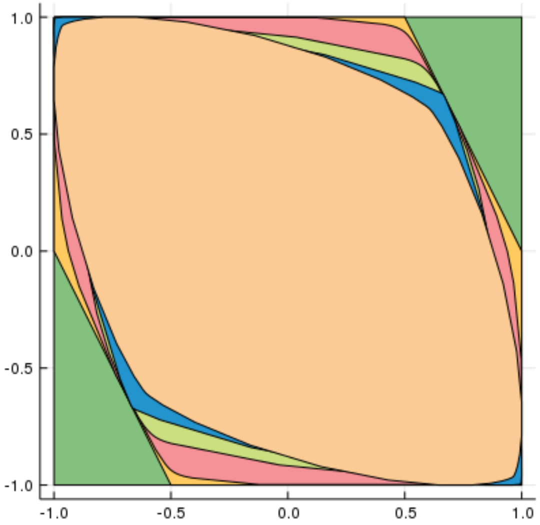


# Example with polynomial sublevel set: Model

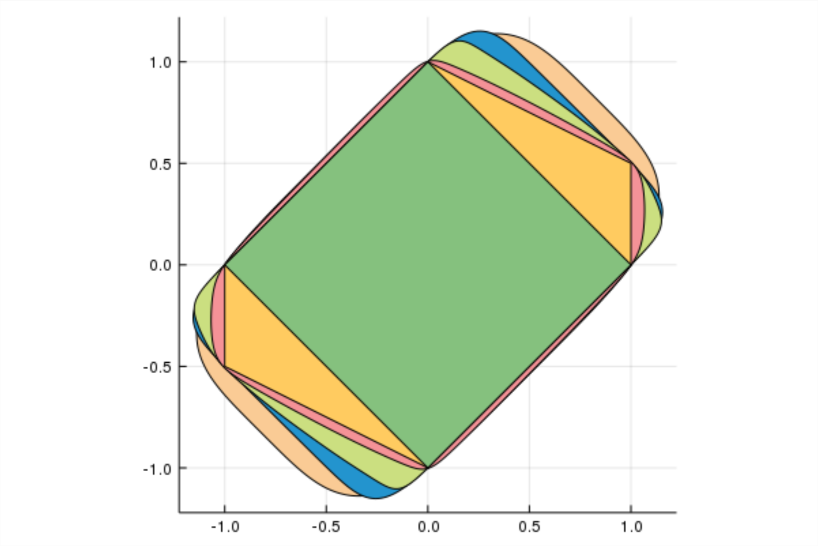
```
@variable(model, S, PolySet(degree=d, convex=true,
                            symmetric=true))
```

Use L1 norm as volume heuristic.

# Example with polynomial sublevel set: Primal Solution



# Example with polynomial sublevel set: Polar Solution



# Set variables

```
function JuMP.build_variable(
    _error, info::JuMP.VariableInfo, set::AbstractSetVariabl
    if !info.has_lb && !info.has_ub && !info.has_fix &&
       !info.binary && !info.integer && !info.has_start
        _error(...)
    end
    return set
end
function JuMP.add_variable(
    model::JuMP.AbstractModel, set::AbstractSetVariable,
    name::String)
    vref = SetVariableRef(...)
    push!(data(model).variables, vref)
    return vref
end
```

# Parsing set constraints

```
function JuMP.parse_one_operator_constraint(_error::Function, vectorized::Bool,
                                            ::Val{:⊂}, lhs, rhs)
    _error("Unrecognized symbol ⊂ you mean ⊆ ?")
end
function JuMP.parse_one_operator_constraint(_error::Function, vectorized::Bool,
                                            ::Val{:⊆}, lhs, rhs)
    parse_code = :()
    if vectorized
        build_call = :(JuMP.build_constraint.($_error, $(esc(lhs)), $(esc(:(SetProg.PowerSet.($rhs))))))
    else
        build_call = :(JuMP.build_constraint($_error, $(esc(lhs)), $(esc(:(SetProg.PowerSet($rhs))))))
    end
    return parse_code, build_call
end
function JuMP.parse_one_operator_constraint(_error::Function, vectorized::Bool,
                                            ::Val{:⊃}, lhs, rhs)
    _error("Unrecognized symbol ⊃, did you mean ⊇ ?")
end
function JuMP.parse_one_operator_constraint(_error::Function, vectorized::Bool,
                                            ::Val{:⊇}, lhs, rhs)
    parse_one_operator_constraint(_error, vectorized, Val(:⊆), rhs, lhs)
end
```

# Store set constraints

```
function JuMP.build_constraint(
    _error::Function, subset, sup_powerset::PowerSet;
    kws...)
    InclusionConstraint(subset, sup_powerset.set, kws)
end
function JuMP.add_constraint(
    model::JuMP.Model, constraint::SetConstraint,
    name::String="")
    d = data(model)
    index = ConstraintIndex(d.last_index += 1)
    d.constraints[index] = constraint
    d.names[index] = name
    return JuMP.ConstraintRef(model, index, SetShape())
end
```

# Optimize hook

`create_spaces`: Find dimensions and representation (e.g. polar/dual or not)

```
function optimize_hook(model::JuMP.AbstractModel)
    d = data(model)
    clear_spaces(d)
    create_spaces(d)
    load(model, d)
    JuMP.optimize!(model, ignore_optimize_hook = true)
end
```

# Load set constraints

S-procedure:

$$Q \subseteq P$$

$$x^\top Q x \le 1 \Rightarrow x^\top P x \le 1$$

$$x^\top P x \le x^\top Q x \quad \forall x$$

$$Q - P \text{ is PSD}$$

```
function JuMP.build_constraint(
    _error::Function, subset::Ellipsoid,
    sup_powerset::PowerSet{<:Ellipsoid})
    Q = subset.Q
    P = sup_powerset.set.Q
    JuMP.build_constraint(_error, Symmetric(Q - P),
                          PSDCone())
end
```

# Polar inclusion

```
function JuMP.build_constraint(
    _error::Function, subset::Sets.Polar,
    sup_powerset::PowerSet{<:Sets.Polar})
    S = subset
    T = sup_powerset.set
    JuMP.build_constraint(
        _error, Sets.polar(T), PowerSet(Sets.polar(S)))
end

function JuMP.build_constraint(
    _error::Function, subset::Sets.Polar,
    sup_powerset::PowerSet{<:Polyhedra.HalfSpace})
    point = sup_powerset.set.a / sup_powerset.set.b
    JuMP.build_constraint(_error, point, Sets.polar(subset))
end
```

# Future work

- Rely on `MathematicalSets` to represent sets.
- Polyhedra solver (CDD, LazySets, ...)
- Direction objective: Ellipsoid (SDP), polynomial (SOS), polyhedra (SDDP).