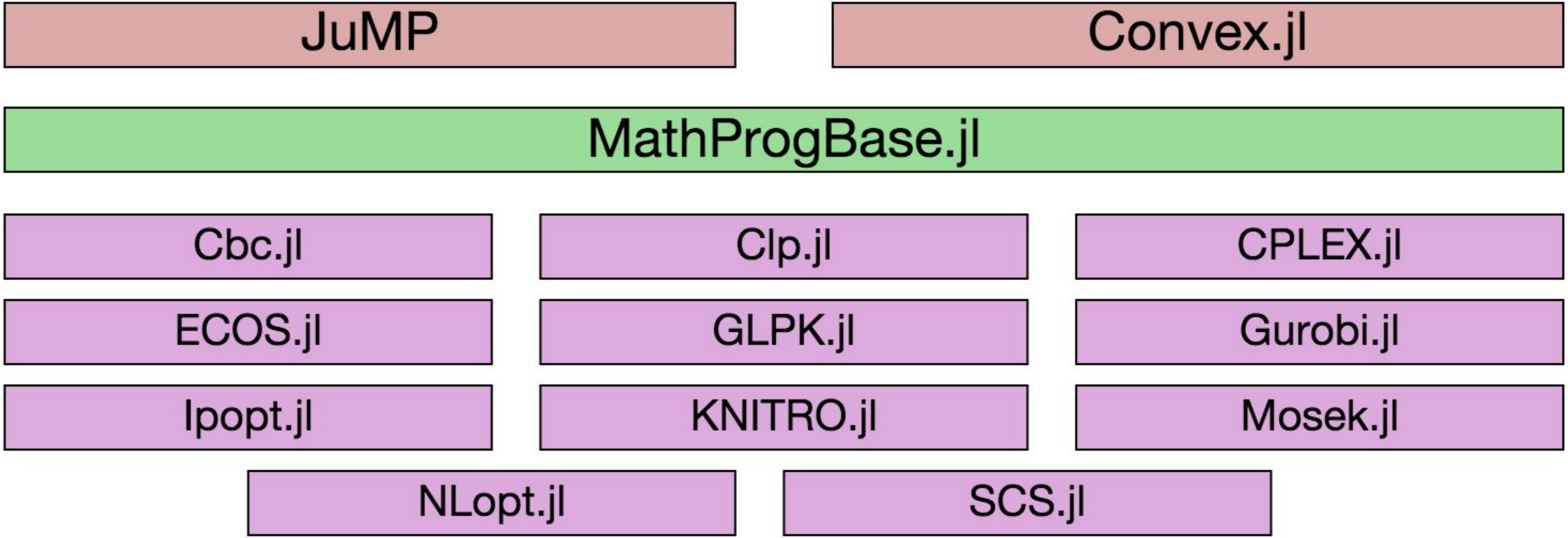
MathOptInterface and JuMP 0.19

Miles Lubin
Google
JuMP-dev 2018

JuMP is great, but how do I …

- add support for a new type of constraint?
- combine NLP constraints with conic constraints?
- delete a constraint or variable?
- test if a solution is feasible?
- modify coefficients in the constraint matrix?
- provide a dual warm-start?
- access the irreducible inconsistent subsystem (IIS) from Gurobi?
- distinguish between a solver that stopped because of the time limit 1) *with* a solution and 2) *without*?
- handle *CPXMIP_OPTIMAL_INFEAS*?

JuMP’s architecture (0.1 to 0.18)



~~**MathProgBase**~~

MathOptInterface (MOI)

- New problem representation
  - Extensible way to define categories of constraints and modifications
- New interface for attributes
  - Warm starts, IIS
- New status codes
- Nonconsecutive variable and constraint indices
  - For deletion
- Callbacks *not* defined; they become solver-specific
- Nonlinear optimization mostly unchanged

MOI definition of an optimization problem

The standard form problem is:

$$\begin{align}
\min_{x \in \mathbb{R}^n} \quad & f_0(x) \\
\text{s.t.} \quad & f_i(x) \in \mathcal{S}_i \qquad i = 1 \ldots m
\end{align}$$

where:

- the functions $f_0, f_1, \ldots, f_m$ are specified by `AbstractFunction` objects
- the sets $\mathcal{S}_1, \ldots, \mathcal{S}_m$ are specified by `AbstractSet` objects

Standard functions

The current function types are:

- `SingleVariable`: x_j, projection onto a single coordinate defined by a variable index j
- `VectorOfVariables`: projection onto multiple coordinates (i.e., extracting a subvector)
- `ScalarAffineFunction`: $a^T x + b$, where a is a vector and b scalar
- `VectorAffineFunction`: $Ax + b$, where A is a matrix and b is a vector
- `ScalarQuadraticFunction`: $\frac{1}{2}x^T Qx + a^T x + b$, where Q is a symmetric matrix, a is a vector, and b is a constant
- `VectorQuadraticFunction`: a vector of scalar-valued quadratic functions

And their definitions...

```
"""
    VariableIndex

A type-safe wrapper for `Int64` for use in referencing variables in a model.
To allow for deletion, indices need not be consecutive.
"""
struct VariableIndex
    value::Int64
end
```

```
struct SingleVariable <: AbstractScalarFunction
    variable::VariableIndex
end
```

```julia
struct ScalarAffineTerm{T}
    coefficient::T
    variable_index::VariableIndex
end
```

```julia
mutable struct ScalarAffineFunction{T} <: AbstractScalarFunction
    terms::Vector{ScalarAffineTerm{T}}
    constant::T
end
```

A few sets

- `LessThan(upper)`: $\{x \in \mathbb{R} : x \leq \text{upper}\}$
- `GreaterThan(lower)`: $\{x \in \mathbb{R} : x \geq \text{lower}\}$
- `EqualTo(value)`: $\{x \in \mathbb{R} : x = \text{value}\}$
- `Interval(lower, upper)`: $\{x \in \mathbb{R} : x \in [\text{lower}, \text{upper}]\}$
- `SecondOrderCone(dimension)`: $\{(t, x) \in \mathbb{R}^{\text{dimension}} : t \geq ||x||_2\}$
- `Integer()`: $\mathbb{Z}$
- `ZeroOne()`: $\{0, 1\}$

And their definitions…

```
struct LessThan{T <: Real} <: AbstractScalarSet
    upper::T
end
```

```
struct SecondOrderCone <: AbstractVectorSet
    dimension::Int
end
```

```
struct ZeroOne <: AbstractScalarSet end
```

Linear constraints

Mathematical Constraint	MOI Function	MOI Set
$a^T x \leq u$	`ScalarAffineFunction`	`LessThan`
$a^T x \geq l$	`ScalarAffineFunction`	`GreaterThan`
$a^T x = b$	`ScalarAffineFunction`	`EqualTo`
$l \leq a^T x \leq u$	`ScalarAffineFunction`	`Interval`
$x_i \leq u$	`SingleVariable`	`LessThan`
$x_i \geq l$	`SingleVariable`	`GreaterThan`
$x_i = b$	`SingleVariable`	`EqualTo`
$l \leq x_i \leq u$	`SingleVariable`	`Interval`
$Ax + b \in \mathbb{R}_+^n$	`VectorAffineFunction`	`Nonnegatives`
$Ax + b \in \mathbb{R}_-^n$	`VectorAffineFunction`	`Nonpositives`
$Ax + b = 0$	`VectorAffineFunction`	`Zeros`

Quadratic constraints

Mathematical Constraint	MOI Function	MOI Set
$x^T Qx + a^T x + b \geq 0$	`ScalarQuadraticFunction`	`GreaterThan`
$x^T Qx + a^T x + b \leq 0$	`ScalarQuadraticFunction`	`LessThan`
$x^T Qx + a^T x + b = 0$	`ScalarQuadraticFunction`	`EqualTo`
Bilinear matrix inequality	`VectorQuadraticFunction`	`PositiveSemidefiniteCone...`

Discrete and logical constraints

Mathematical Constraint	MOI Function	MOI Set
$x_i \in \mathbb{Z}$	`SingleVariable`	`Integer`
$x_i \in \{0, 1\}$	`SingleVariable`	`ZeroOne`
$x_i \in \{0\} \cup [l, u]$	`SingleVariable`	`Semicontinuous`
$x_i \in \{0\} \cup \{l, l+1, \ldots, u-1, u\}$	`SingleVariable`	`Semiinteger`
At most one component of x can be nonzero	`VectorOfVariables`	`SOS1`
At most two components of x can be nonzero, and if so they must be adjacent components	`VectorOfVariables`	`SOS2`

Adding a constraint at the MOI level

```
model = OSQPOptimizer()
var_index = MOI.addvariable!(model)
constr_index = MOI.addconstraint!(model,
                                  MOI.SingleVariable(var_index),
                                  MOI.LessThan(10.0))
```

`addconstraint!` returns an index

```julia
"""
    ConstraintIndex{F,S}

A type-safe wrapper for `Int64` for use in referencing `F`-in-`S` constraints in
a model.
The parameter `F` is the type of the function in the constraint, and the
parameter `S` is the type of set in the constraint. To allow for deletion,
indices need not be consecutive. Indices within a constraint type (i.e. `F`-in-`S`)
must be unique, but non-unique indices across different constraint types are allowed.
"""
struct ConstraintIndex{F,S}
    value::Int64
end
```

Deleting variables and constraints

```
MOI.isvalid(model, variable_index) # True
MOI.isvalid(model, constraint_index) # True

MOI.delete!(model, variable_index)
MOI.delete!(model, constraint_index)

MOI.isvalid(model, variable_index) # False
MOI.isvalid(model, constraint_index) # False
```

Adding a constraint at the JuMP level

```
m = JuMP.Model(optimizer = GurobiOptimizer())

@variable(m, x <= 10) # SingleVariable-LessThan
@variable(m, y)

@constraint(m, x + y >= 10) # ScalarAffineFunction-GreaterThan
@constraint(m, [x,y] in MOI.SecondOrderCone(2)) # VectorOfVariables-...

# Hypothetical
@constraint(m, [x,1-x] in MOI.Complements()) # VectorAffineFunction-...
```

`@constraint` returns a reference

```
struct ConstraintRef{M<:AbstractModel,C}
    m::M
    index::C
end
```

For example,

```
ConstraintRef{Model, MOI.ConstraintIndex{MOI.ScalarAffineFunction,
                                         MOI.LessThan}}
```

Discussion

There are multiple ways to write down the same constraint. Which should a solver support? *Who should do the work of transforming the problem?*

- `VectorAffineFunction`-in-`Zeros` vs. multiple `ScalarAffineFunction`-in-`EqualsTo`?
- `GemetricMeanCone` versus `PowerCone` versus `SecondOrderCone`?

MOI provides a single framework in which to experiment with different representations of a problem at *both the model and solver level*.

Attributes in MOI

```
model = OSQPOptimizer()
var_index = MOI.addvariable!(model)
constr_index = MOI.addconstraint!(model,
                                  MOI.SingleVariable(var_index),
                                  MOI.LessThan(10.0))

MOI.set!(model, MOI.ObjectiveSense(), MOI.MaxSense)

MOI.set!(model, MOI.ObjectiveFunction(), [...])

MOI.set!(model, var_index, MOI.VariablePrimalStart(), 5.0)

MOI.set!(model, constr_index, MOI.ConstraintDualStart(), 10.0)
```

Attributes in JuMP

```
model = JuMP.Model(optimizer = OSQPOptimizer())
@variable(model, x >= 10.0, start = 5.0)
@objective(model, Max, [...])

bound_ref = JuMP.LowerBoundRef(x)
MOI.set!(model, bound_ref, MOI.ConstraintDualStart(), 10.0)
```

From MOI to JuMP

```julia
"""
    VariableRef <: AbstractVariableRef

Holds a reference to the model and the corresponding MOI.VariableIndex.
"""
struct VariableRef <: AbstractVariableRef
    m::Model
    index::MOIVAR
end
```

```julia
function MOI.set!(m::Model, attr::MOI.AbstractVariableAttribute, v::VariableRef, value)
    @assert m === v.m
    MOI.set!(m.moibackend, attr, index(v), value)
end
```

```julia
startvalue(v::VariableRef) = MOI.get(v.m, MOI.VariablePrimalStart(), v)
setstartvalue(v::VariableRef, val::Number) = MOI.set!(v.m, MOI.VariablePrimalStart(), v, val)
```

Status codes

1. Why did the solver stop? `TerminationStatus()`
2. Does the solver have vectors to return? `ResultCount()`
3. What do the result vectors mean? `PrimalStatus()` and `DualStatus()`

Example situations with primal-dual solvers

What happened?	`TerminationStatus`	`ResultCount`	`PrimalStatus`	`DualStatus`
Proved optimality*	`Success`	1	`FeasiblePoint`	`FeasiblePoint`
Proved primal infeasible	`Success`	1	error	`InfeasibilityCertificate`
Optimal within relaxed tolerances	`AlmostSuccess`	1	`FeasiblePoint` or `AlmostFeasiblePoint`	`FeasiblePoint` or `AlmostFeasiblePoint`
Stall	`SlowProgress`	1	?	?

* within numerical tolerances or given optimality gap

Example situations with MIP solvers

What happened?	`TerminationStatus`	`ResultCount`	`PrimalStatus`	`DualStatus`
Proved optimality	`Success`	1	`FeasiblePoint`	error
Proved infeasible or unbounded	`InfeasibleOrUnbounded`	0	error	error
Proved infeasible	`InfeasibleNoResult`	0	error	error
Timed out (no solution)	`TimeLimit`	0	error	error
Timed out (with solution)	`TimeLimit`	1	`FeasiblePoint`	error
`CPXMIP_OPTIMAL_INFEAS`	`Success`	1	`InfeasiblePoint`	error

Example situations with NLP solvers

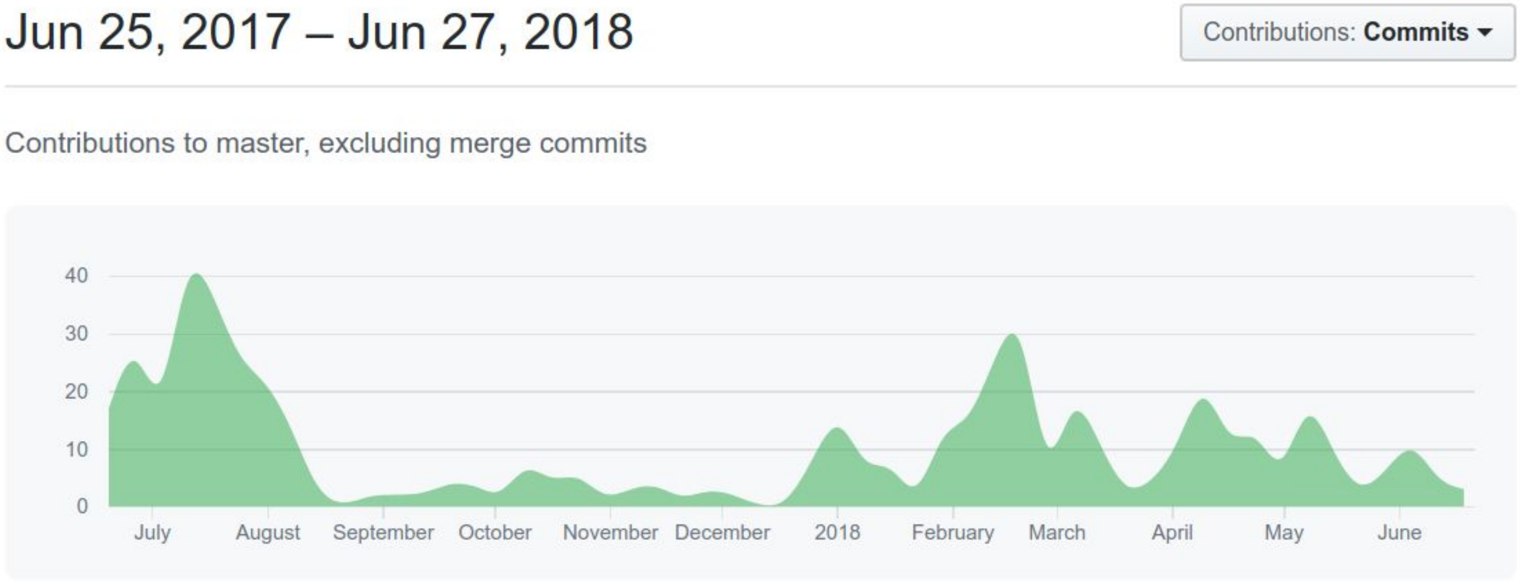
What happened?	`TerminationStatus`	`Result Count`	`PrimalStatus`	`DualStatus`
Converged to feasible point	`Success`	1	`FeasiblePoint`	`FeasiblePoint`
Converged to infeasible point	`Success`	1	`InfeasiblePoint`	`FeasiblePoint` ?
Iteration limit	`IterationLimit`	1	?	?
Diverging	`NormLimit` or `ObjectiveLimit`	1	?	?

Modifications

- `x ≤ 1 ⇒ x ≤ 2`: set `ConstraintSet` attribute
- `x ≤ 1 ⇒ x ≥ 2`: call `MOI.transform!`
- `2x + y ≤ 10` ⇒ `3x + y ≤ 10`: set `ConstraintFunction` attribute or call `MOI.modify!` with `ScalarCoefficientChange`

The state of MOI

MOI 0.4 released this week. 700+ commits, 400 issues/PRs.



Status of solver wrappers

Released version supports MOI:

- CSDP
- ECOS
- Ipopt
- Mosek
- OSQP

MOI support in PR or master branch:

- Cbc
- Clp
- Gurobi
- GLPK
- Xpress
- SDPA
- SCS

Up for grabs:

- AmplNLWriter
- CPLEX
- Knitro
- NLopt
- Pajarito
- SCIP

SemidefiniteOptInterface and LinQuadOptInterface are optional helper layers for implementing MOI wrappers.

JuMP/MOI

What *hasn't* changed:

- Macro syntax
  - Except for `norm()`
- Variable construction syntax
- Automatic differentiation (ReverseDiffSparse moved into JuMP repo.)

What *has* changed

- Containers
- Names
- Data structures for AffExpr and QuadExpr
- Interacting with solvers
- Software engineering improvements
- Documenter and docstrings for documentation

JuMP containers

- `JuMPDict` replaced by `Base.Dict`
- `JuMPArray` rewritten, inspired by `AxisArrays`

Review of JuMP containers

```
@variable(m, [1:5, 1:5]) # Array
set_1 = Base.OneTo(5)
@variable(m, [set_1, 1:5]) # Array
set_2 = 1:5
@variable(m, [1:5, set_2]) # JuMPArray
set_3 = [:a, :b, :c]
@variable(m, [set_2, set_3]) # JuMPArray
@variable(m, [i=set_2, 1:i]) # Dict
@variable(m, [i = 1:5; isodd(i)]) # Dict
```

Same applies for `@constraint`, `@expression`, `@NLconstraint`, `@NLexpression`

Now possible to request a container type

```julia
@variable(m, [1:5, 1:5], container=JuMPArray)
set_1 = 1:5
@variable(m, [set_1, 1:5], container=Array)
set_2 = 2:3
@variable(m, [set_2, 1:5], container=Array) # Error
```

New JuMPArrays

```julia
set_1 = [:a, :b, :c]
set_2 = [:x, :y, :z]
```

```julia
# 0.18
julia> @variable(m, x[set_1, set_2])
x[i,j] ∀ i ∈ {a,b,c}, j ∈ {x,y,z}

julia> x[:, :z]
ERROR: Failed attempt to index JuMPArray ...
       along dimension 1: Colon() ∉ Symbol[:a, :b, :c]
```

```julia
# 0.19
julia> @variable(m, x[set_1, set_2])
2-dimensional JuMPArray{JuMP.VariableRef,2,...} with index sets:
    Dimension 1, Symbol[:a, :b, :c]
    Dimension 2, Symbol[:x, :y, :z]
And data, a 3×3 Array{JuMP.VariableRef,2}:
 x[a,x]  x[a,y]  x[a,z]
 x[b,x]  x[b,y]  x[b,z]
 x[c,x]  x[c,y]  x[c,z]

julia> x[:, :z]
1-dimensional JuMPArray{JuMP.VariableRef,1,...} with index sets:
    Dimension 1, Symbol[:a, :b, :c]
And data, a 3-element Array{JuMP.VariableRef,1}:
 x[a,z]
 x[b,z]
 x[c,z]
```

Names

- Variables and constraints now have string names. The set of nonempty names is unique. You can lookup by name.
- Model scope (`model[:x]`) still exists, useful for non-scalar variables.

AffExpr and QuadExpr

0.18

```
mutable struct GenericAffExpr{CoefType,VarType}
    vars::Vector{VarType}
    coeffs::Vector{CoefType}
    constant::CoefType
end
```

```
mutable struct GenericQuadExpr{CoefType,VarType}
    qvars1::Vector{VarType}
    qvars2::Vector{VarType}
    qcoeffs::Vector{CoefType}
    aff::GenericAffExpr{CoefType,VarType}
end
```

0.19

```
mutable struct GenericAffExpr{CoefType,VarType}
    constant::CoefType
    terms::OrderedDict{VarType,CoefType}
end
```

```
mutable struct GenericQuadExpr{CoefType,VarType}
    aff::GenericAffExpr{CoefType,VarType}
    terms::OrderedDict{UnorderedPair{VarType}, CoefType}
end
```

- No duplicates by construction
- Faster on 0.7

The multi-backend problem

JuMP now supports many kinds of problem modifications. Solvers support a subset of these. We want to keep the solver in memory and pass modifications efficiently when possible.

JuMP stores the problem data only in MOI

```julia
"""
    addconstraint(m::Model, c::AbstractConstraint, name::String="")

Add a constraint `c` to `Model m` and sets its name.
"""
function addconstraint(m::Model, c::AbstractConstraint, name::String="")
    cindex = MOI.addconstraint!(m.moibackend, moi_function_and_set(c)...)
    cref = ConstraintRef(m, cindex)
    if !isempty(name)
        setname(cref, name)
    end
    return cref
end
```

JuMP solver modes

- Direct: the `moibackend` field is a solver (e.g., Gurobi)
  - This is the mode for using callbacks.
- Manual: the `moibackend` field is a `CachingOptimizer` in Manual mode
  - A solver is “attached” or “empty”. When the solver is attached, error if user attempts to make a modification that the solver doesn’t support.
- Automatic: the `moibackend` field is a `CachingOptimizer` in Automatic mode
  - Solver is attached and emptied when needed without notice.

Are my incremental modifications efficiently passed to the solver?

Direct: Yes, you’ll always get an error when you make a change that the solver doesn’t support.

Manual: Yes, you control when the model is reloaded into the solver.

Automatic: Maybe, this will happen silently.

Software engineering improvements

JuMP's tests no longer depend on a solver!

- MOI lightweight text format for testing model generation
- Mock solvers for testing communication with a solver

```julia
@testset "Generation and solve with fake solver" begin
    @testset "LP" begin
        m = Model()
        @variable(m, x <= 2.0)
        @variable(m, y >= 0.0)
        @objective(m, Min, -x)

        c = @constraint(m, x + y <= 1)

        JuMP.setname(JuMP.UpperBoundRef(x), "xub")
        JuMP.setname(JuMP.LowerBoundRef(y), "ylb")
        JuMP.setname(c, "c")

        modelstring = """
        variables: x, y
        minobjective: -1.0*x
        xub: x <= 2.0
        ylb: y >= 0.0
        c: x + y <= 1.0
        """

        model = JuMP.JuMPMOIModel{Float64}()
        MOIU.loadfromstring!(model, modelstring)
        MOIU.test_models_equal(m.moibackend.model_cache, model, ["x","y"], ["c", "xub", "ylb"])
```

Style guide and design principles

Style guide

TODO: A style guide for JuMP, JuMP models and surrounding Julia code. Formatting, naming, use of macros, comments, TODOs, docstrings, etc.

Design principles

TODO: How to structure and test large JuMP models, libraries that use JuMP.

For how to write a solver, see MOI.

Remaining JuMP TODOs for 0.19

- Model printing
- Clean up API
  - Modifications not exposed
  - Names not well exposed
  - Getting/setting the solver
- Documentation
  - Guide for updating from 0.18
  - Update examples
- Callbacks
- Support 0.7

Maintenance plans for JuMP/MPB

MPB and MOI wrappers will co-exist for some time.

We would like to release a version of JuMP 0.18 that's compatible with Julia 0.7.

Yesterday's announcement

JuMP will be a NumFOCUS Sponsored Project!

NUMFOCUS

OPEN CODE = BETTER SCIENCE

JuliaOpt

JUMP

NumFOCUS will help with:

- Receiving Donations
- Receiving Grants
- Google Summer of Code
- Holding funds for the JuMP-dev workshop
- Facilitating contracts for open-source work

New JuMP branding

- JuMP needs a (new) website
- Low priority: New name for the GitHub organization?

Thanks!