
JuMP
Developers
Workshop
June 12-16, 2017.
MIT MANAGEMENT
SLOAN SCHOOL
Speakers
Chris Coey, MIT • Carleton Coffrin, LANL • Steven Diamond, Stanford • Joaquim Dias Garcia, PSR & PUC-Rio • Oscar Dowson, U. of Auckland • Joey Huchette, MIT • Jordan Jalving, UW-Madison • Benoît Legat, UCL • Miles Lubin, MIT • Yee Sian Ng, MIT • Jarrett Revels, MIT • Nestor Sepulveda, MIT • Bartolomeo Stellato, U. of Oxford • Juan Pablo Vielma, MIT
Sponsored by:
MIT
MANAGEMENT
LATIN AMERICA OFFICE
www.juliaopt.org/developersmeetup

Why a meetup?

- Transfer of knowledge
- Community building
- Decisions on JuMP 1.0 and MathProgBase 1.0
- Accelerate JuMP-related projects

Schedule

http://www.juliaopt.org/developersmeetup/

WiFi: MIT GUEST

Streaming

The Design of JuMP and MathProgBase

Miles Lubin

JuMP developers meetup

June 12, 2017

To accomplish in this talk

- Explain **everything** about JuMP (maybe not everything)
  - How JuMP evolved
  - Design considerations
  - Which parts are there for a good reason
  - Which parts should be reconsidered
- This is a starting point for rest of discussions

What did we want in a modeling language?

- Modern, modular, easy to embed…
  - Within a simulation
  - Within an interactive visualization
- Interact with solvers while they are running
- Easy to extend to specialized problem classes
- Commercial tools are **none of these**
- Open source tools have been **slow**
  - Based on Python and MATLAB

Why We Created Julia

14 Feb 2012 | Jeff Bezanson, Stefan Karpinski, Viral Shah, Alan Edelman

In short, because we are greedy.

We are power Matlab users. Some of us are Lisp hackers. Some are Pythonistas, others Rubyists, still others Perl hackers. There are those of us who used Mathematica before we could grow facial hair. There are those who still can't grow facial hair. We've generated more R plots than any sane person should. C is our desert island programming language.

We love all of these languages; they are wonderful and powerful. For the work we do — scientific computing, machine learning, data mining, large-scale linear algebra, distributed and parallel computing — each one is perfect for some aspects of the work and terrible for others. Each one is a trade-off.

We are greedy: we want more.

October 2012

```
#############################################################
# Julp
# A MILP modelling langauge for Julia
#  Julia
# +  LP
# -----
# =Julp.
#
# By Iain Dunning and Miles Lubin
#############################################################
```

First implementation question

How do we capture linear expressions inside a programming language?

Modeling in Julia

http://github.com/IainNZ/Julp

- Julia replaces domain-specific language
- Use macros to avoid issues with operator overloading

```
m = Model("max")
x = [ Variable(m) for j = 1:N ]
profit = rand(N);  weight = rand(N);

setObjective(m,
    @sumExpr([ profit[j] * x[j] for j = 1:N ])
)

addConstraint(m,
    @sumExpr([ weight[j] * x[j] for j = 1:N ])
)
```

January 2013: https://youtu.be/O1icUP6sajU

Revisiting the performance trade-offs

https://github.com/mlubin/18.337/blob/master/operators.jl

- AffExpr operator overloading
- Graph operator overloading
- Manual expression construction

Consequences of choosing to use macros

- JuMP is fast
- Performance warnings on AffExpr operator overloads
- “variable*coefficient” not supported until JuMP 0.7.0

First benchmarks (May 2013)

Table 2 **Linear-quadratic control benchmark results. N=M is the grid size. Total time (in seconds) to process the model definition and produce the output file in LP and MPS formats (as available).**

	JuMP/Julia		AMPL	Gurobi/C++		Pulp/PyPy		Pyomo
N	LP	MPS	MPS	LP	MPS	LP	MPS	LP
250	0.5	0.9	0.8	1.2	1.1	8.3	7.2	13.3
500	2.0	3.6	3.0	4.5	4.4	27.6	24.4	53.4
750	5.0	8.4	6.7	10.2	10.1	61.0	54.5	121.0
1,000	9.2	15.5	11.6	17.6	17.3	108.2	97.5	214.7

M. Lubin & I. Dunning, “Computing in Operations Research using Julia”, *INFORMS Journal on Computing*, 2015

Note: Why don’t LP/MPS writers use names?

They were too slow in the benchmarks. (code)

Also, benchmarks led us to implement crazy lazy naming scheme:

`@variable(m, x[1:N])` only creates names for x when needed. (code)

JuliaOpt / **JuMP.jl**

Unwatch 52 | Unstar 330 | Fork 92

Code | Issues 52 | Pull requests 6 | Projects 0 | Pulse | Graphs | Settings

Quadratic constraints #34

Edit

Merged **mlubin** merged 12 commits into `JuliaOpt:master` from `unknown repository` on Sep 12, 2013

Conversation 9 | Commits 12 | Files changed 8 | +150 −20

joehuchette commented on Sep 12, 2013 — Member

Added (preliminary) quadratic constraint functionality for LP/MIP.

Reviewers

No reviews—request one

Assignees

Extending macros to quadratic expressions

- Quadratic expressions could only be created by overloading only for a long time
- Traces of this: Collections defined by `@expression` must be linear expressions (code)
- Extending macros to quadratic expressions was hard/impossible before generated functions introduced in Julia 0.4
  - At compile time, unknown what is a variable and what is a coefficient
- Implementation is still messy, use generated functions to reorder arguments (code)

Quadratic benchmarks (May 2015)

		Commercial			Open-source		
Instance	**JuMP**	GRB/C++	AMPL	GAMS	Pyomo	CVX	YALMIP
lqcp-500	8	2	2	2	55	6	8
lqcp-1000	11	6	6	13	232	48	25
lqcp-1500	15	14	13	41	530	135	52
lqcp-2000	22	26	24	101	>600	296	100
fac-25	7	0	0	0	14	>600	533
fac-50	9	2	2	3	114	>600	>600
fac-75	13	5	7	11	391	>600	>600
fac-100	24	12	18	29	>600	>600	>600

TABLE 1

Time (sec.) to generate each model and pass it to the solver, a comparison between JuMP and existing commercial and open-source modeling languages. The `lqcp` *instances have quadratic objectives and linear constraints. The* `fac` *instances have linear objectives and conic-quadratic constraints.*

What does JuMP really do?

- Provides a nice interface to MathProgBase
- Why? Keep JuMP lightweight, fast, no transformations

MathProgBase (MPB)

- Started out as an interface to LP solvers (March 2013)
- Before that, Julia had GLPK built-in! (for package management)
- Named MPB because JuMP was called MathProg for a while (Feb 2013)
- After LP, tacked on integers, SOS, callbacks, nonlinear, conic
- Problem: which solvers support what?
  - Led to splitting into LPQP, Conic, Nonlinear
- Major reorganization to be proposed this afternoon

The abstraction juggle

- Try to keep MathProgBase close to the solver APIs to make the Julia wrappers simple
- Try to keep the JuMP model close to the in-memory model so that we can easily modify it in place
- Compare with file formats: LP, MPS, NL, OSiL, CBF

Three problem classes

- LinearQuadratic
- Conic
- Nonlinear

LinearQuadratic abstraction

$$\min_x c^T x$$
$$s.t. a_i^T x \ \text{sense}_i \ b_i \forall \ i$$
$$l \leq x \leq u$$

- Plus integer variables, quadratic objective, quadratic constraints, SOCP
- LP hotstarts, branch & bound callbacks
- CPLEX, Gurobi, Cbc/Clp, GLPK, Mosek, SCIP

```
mcf = Model()

@variable(mcf, 0 <= flow[e in edges] <= e.capacity)

@constraint(mcf, sum(flow[e] for e in edges if e.to==5)== 1)

@constraint(mcf, flowcon[n=2:4],sum(flow[e] for e in edges
if e.to==node) == sum(flow[e] for e in edges if
e.from==node))

@objective(mcf, Min, sum(e.cost * flow[e] for e in edges))

solve(mcf)
```

Generates input data structures and calls `loadproblem!`, `optimize!`, etc.

Conic abstraction

$$\min_x c^T x \qquad \max_y -b^T y$$

$$s.t.\, b - Ax \in K_1 \qquad s.t.\, c + A^T y \in K_2^*$$

$$x \in K_2 \qquad y \in K_1^*$$

- Linear, SOC, SDP, exponential cones
- Mosek, ECOS, SCS, CSDP

Nonlinear abstraction

$$\min_x f(x)$$
$$s.t.\, lb \leq g(x) \leq ub$$
$$l \leq x \leq u$$

- Gradient, Jacobian, Hessian oracles, expression graphs
- Ipopt, Mosek, KNITRO, NLopt

Callbacks

- Envisioned to be solver independent, only support a callback when it exists in more than one solver (e.g., lazy constraints)
- Problematic (what is a lazy constraint?), maybe too much abstraction

Proposed reorganization

- Solver statuses (issue)
- Generalizing conic form to “quadratic expression in set” (issue)
  - Subsumes both LinearQuadratic and Conic
- More discussion this afternoon

JuMP syntax

- How we started
- What we learned

February 2013: beginning of “modern” syntax

Old

```
s = [ Variable(m,0,1,0,"s$i") for i=1:numLocation ]
...
```

New

```
@defVar(m, 0 <= s[1:numLocation] <= 1)
@addConstraint(m, sum{s[i],i=1:numLocation} == numFacility )
```

Recent changes

- Renamed everything from camelCase (JuMP 0.13, April 2016)
- `sum{}` becomes `sum()` (JuMP 0.15, December 2016)

Defining variables

- Syntax has gotten simpler
  - Disallow `@variable(m, x[1][1:N])`
  - Disallow multiple variables with the same name
  - Anonymous variables
- Remove the magic

`@variable(m, x[1:N])` is a shorthand for

```
# error if m[:x] is already defined
x = m[:x] = @variable(m, [1:N], basename="x")
```

JuMP’s containers

- Inspired by AMPL, allow indexing over arbitrary sets
  - ```
    s = ["Green", "Blue"]
    @variable(m, x[-10:10,s], Int )
    ```
- Introduced triangular indexing in JuMP 0.6
  - ```
    @variable(m, x[i=1:10,j=i:10])
    ```
- Introduced conditional indexing in JuMP 0.10
  - ```
    @variable(m, x[i=1:10; iseven(i)])
    ```
    ```

JuMP’s containers

- If all index sets are ranges of the form 1:N, return a Julia **`Array`**
- If all axes are independent, return a **`JuMPArray`**
  - AxisArrays?
- Otherwise, return a **`JuMPDict`**
- **`JuMPDict`** is “just” a wrapper around a Julia Dict that translates **`x[a,b,c]`** to **`x.tupledict[(a,b,c)]`**

Semidefinite programming

- Required introduction of vector-based syntax, essentially using AffExpr operator overloading even inside macros
- `@SDconstraint(m, X >= 0)`
  - `@constraint(m, X in PSD())` soon?
- Symmetry is problematic (issue)
  - `A'XA` may not be exactly symmetric because of order of operations

Nonlinear programming, expression graphs

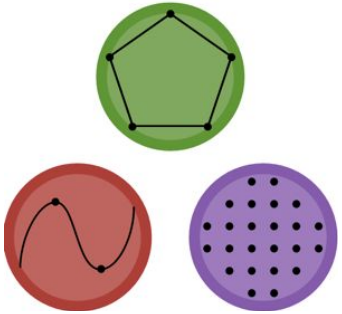
- Introduced in JuMP 0.5, rewritten in JuMP 0.12
- `@NLconstraint` stores a scalar-based expression graph, vector expressions not supported
- Key points already explained: https://youtu.be/xtfNug-htcs
- Future: ReverseDiff?

JuMP internals for extensions

- The `ext` dict (code)
- `solvehook` and `printhook` (code)
- Generic affine expressions (code)
- Generic norms (code)
- Re-using JuMP containers (example)
- Recent generalization of `@variable` (PR)
- Syntax for set inclusion constraints (PR)

JuliaOpt

October/November 2013



What is JuliaOpt?

JuliaOpt is an umbrella group for Julia-based optimization-related projects. All our code is available from our GitHub page, as well as through the Julia package manager. The current JuliaOpt projects are:

- **JuMP** - An algebraic modeling language for optimization problems embedded in Julia that generates models as quick as commercial modeling tools. (Documentation, GitHub)
- **Optim** - Implementations of standard optimization algorithms in pure Julia for unconstrained or box constrained problems. (Documentation, GitHub)
- **MathProgBase** - A standardized interface implemented by solvers that allows code to remain solver-agnostic. Used by JuMP. (Documentation, GitHub)

The new JuliaOpt

- GitHub organizations were designed to facilitate managing permissions and tasks across multiple repositories, not to be a public brand
- New JuliaOpt is organized around collaboration and interest in a common set of packages
- Hence “Optim family” moved to JuliaNLSolvers organization
- Peer review of optimization software?

Thoughts on open-source development