Abstract glue for optimization in Julia

Miles Lubin
with Iain Dunning, Joey Huchette, Tony Kelman,
Dominique Orban, and Madeleine Udell
ISMP – July 13, 2015

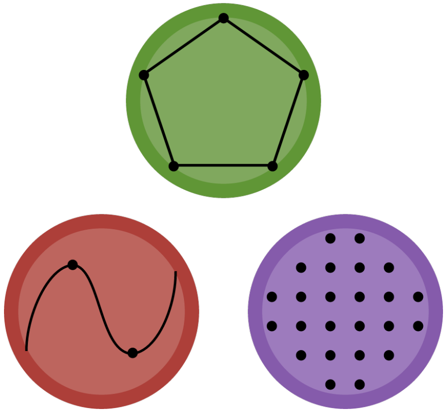
Why did we choose Julia?

Lubin and Dunning, “Computing in Operations Research using Julia”, IJOC, 2015

- “I want to model and solve a large LP/MIP within a programming language, but Python is too slow and C++ is too low level”
- “I want to implement optimization algorithms in a fast, high-level language designed for numerical computing”
- “I want to create an end-user-friendly interface for optimization without writing MEX files”

And so…

We (and many other contributors) developed a new set of tools to help us do our work in Julia.



JuliaOpt

JuMP | Convex.jl

MathProgBase.jl

Cbc.jl | Clp.jl | CPLEX.jl
ECOS.jl | GLPK.jl | Gurobi.jl
Ipopt.jl | KNITRO.jl | Mosek.jl
NLopt.jl | SCS.jl

CoinOptServices.jl

AmplNLWriter.jl

AmplNLReader.jl

Optim.jl

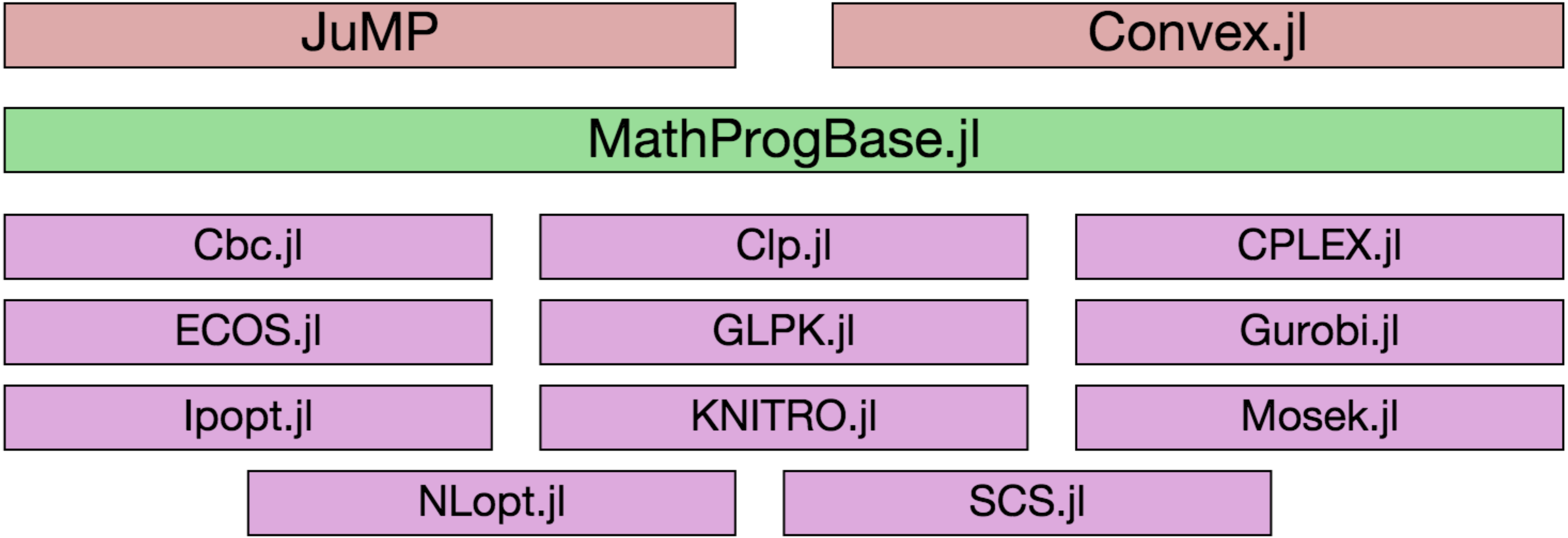
LsqFit.jl

Modeling languages in Julia

- JuMP
  - Linear, mixed-integer, conic, and nonlinear optimization
  - Like AMPL, GAMS, Pyomo
- Convex.jl (Udell, Thursday at 10:20am)
  - Disciplined convex programming
  - Like CVX, cvxpy
- Both use the same solver infrastructure

MathProgBase

- A standard interface which solver wrappers implement
    - Like COIN-OR/OSI


JuMP
Convex.jl
MathProgBase.jl
Cbc.jl
Clp.jl
CPLEX.jl
ECOS.jl
GLPK.jl
Gurobi.jl
Ipopt.jl
KNITRO.jl
Mosek.jl
NLopt.jl
SCS.jl

MathProgBase philosophy

- In a small package which wraps the solver's C API, implement a few additional methods to provide a standardized interface to the solver.
    - Clp.jl, Cbc.jl, Gurobi.jl, ECOS.jl, etc...

MathProgBase philosophy

- Make it easy to access low-level features.
  - Don’t get in the user’s way

MathProgBase philosophy

- If the solver's interface doesn't quite match the abstraction, either:
    - perform some transformations within the solver wrapper, or
    - if the above is too hard, update the abstraction

Diverse classes of solvers

- LP++
- Conic
- Nonlinear

LP++

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

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

Conic

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

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

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

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

Nonlinear

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

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

How it looks for users:

```julia
using JuMP, Clp
m = Model(solver=ClpSolver())
@defVar(m, x[1:2] >= 0)
@setObjective(m, Max, sum(x))
@addConstraint(m,
     x[1]+2*x[2] <= 1)
status = solve(m)
```

```julia
using Convex, Clp
x = Variable(2)
problem = maximize(sum(x),
 [x >= 0, x[1]+2*x[2] <= 1])
solve!(problem, ClpSolver())
```

```julia
using MathProgBase, Clp
sol = linprog([-1.0, -1.0], [1.0 2.0], '<', 1.0, ClpSolver())
```

Wait, how do I set solver options?

```
ClpSolver(PrimalTolerance=1e-5)
GurobiSolver(Method=2,Crossover=0)
CplexSolver(CPX_PARAM_TILIM=100)
MosekSolver(LOG=0)
```

- We don’t abstract over parameters

Wait, how do I do this thing which I can only do from the solver API?

```
# With JuMP model object m
grb = MathProgBase.getrawsolver(getInternalModel(m))
Gurobi.computeIIS(grb)
iisconstr = Gurobi.get_intarray(grb, “IISConstr”, 1, n_constr)
```

Example to compute IIS (Irreducible Inconsistent Subsystem) with Gurobi

Branch & bound callbacks!

```
m = Model(solver=GurobiSolver())
function lazyCallback(cb)
    … # e.g., TSP subtour elimination
end
addLazyCallback(m, lazyCallback)
solve(m)
```

Branch & bound callbacks!

- Lazy constraints, user cuts, user heuristics
- Currently supported by Gurobi, CPLEX, and GLPK

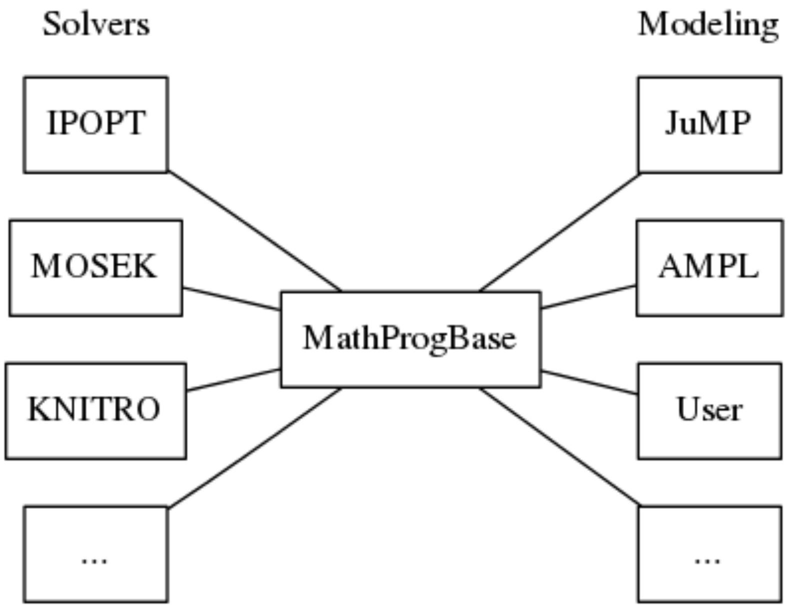
Conic interface

$$\min_x c^T x \quad \text{s.t.}\ b - Ax \in K_1,\quad x \in K_2$$

$$\max_y -b^T y \quad \text{s.t.}\ c + A^T y \in K_2^*,\quad y \in K_1^*$$

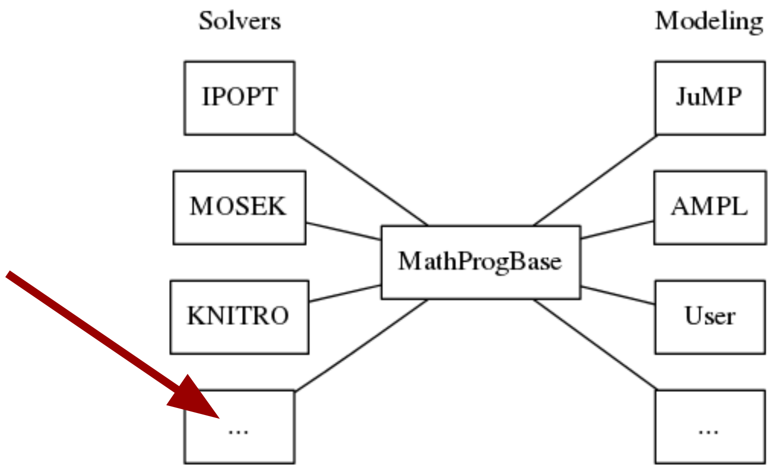
- Input format is sparse matrix A and list of cones (inspired by CBLIB format)
- We have an LP++ ⟷ Conic translation layer

Nonlinear



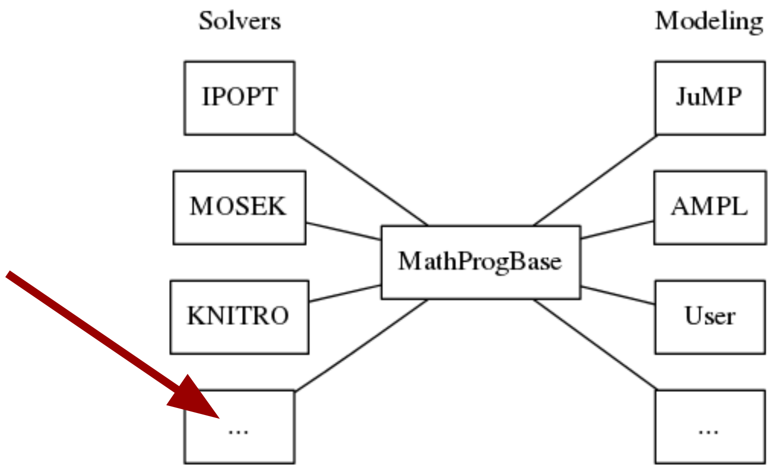
- JuMP implements automatic sparse Hessian computations (preprint)

Nonlinear



- If you write a solver in Julia accepting MathProgBase input, you can call it from both AMPL and JuMP!

Nonlinear

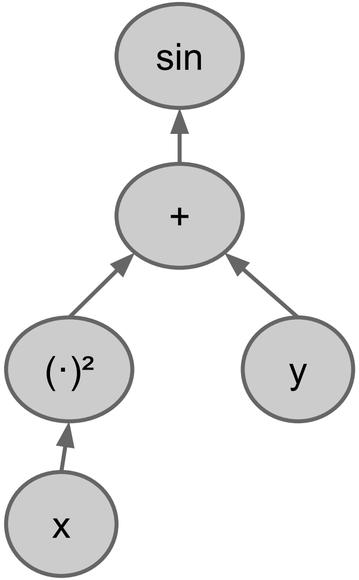


[Demo]

Expression graphs

- Nonlinear interface also allows access to *expression graphs*

$$\sin(x^2 + y)$$



Expression graphs

- Nonlinear interface also allows access to *expression graphs*
- Allows us to write “solvers” which write instances to OSiL and NL file formats
  - CoinOptServices.jl, AmplNLWriter.jl
    - Tony Kelman, Thursday at 10:20am
- DReal.jl – interface to a nonlinear satisfiability solver

Julia is not an island

- Embeddable C API
- Pyjulia

From the Clp mailing list (July 8)

I am in somewhat disbelief that I can't do this:

[xopt,fmin] = linprog(c, Acon, rhsvec) ;

to solve min c' * x given Acon * x <= rhsvec .

The above is the one line matlab interface to linprog.
There should be something similar in Python in support of CLP using it's primary matrix array interface, numpy/ndarrays.

From the Clp mailing list (July 8)

```
I am in somewhat disbelief that I can't do this:
[xopt,fmin] = linprog(c, Acon, rhsvec) ;

to solve min c' * x  given Acon * x <= rhsvec .

The above is the one line matlab interface to linprog.
There should be something similar in Python in support of
CLP using it's primary matrix array interface,
numpy/ndarrays.
```

We can do that: pylinprog

In conclusion

MathProgBase makes it easier than ever before to:

- Write fast, solver-independent code.
    - **There is no loss of performance**
- Write solvers and hook them into open-source and commercial modeling languages.

What’s next

- SCIP
- Constraint programming?

Thanks to

- David Anthoff, Carlo Baldassi, Oscar Dowson, Jack Dunn, Jenny Hong, Steven G. Johnson, Dahua Lin, Karanveer Mohan, Yee Sian Ng, Brendan O’Donoghue, Leonardo Taccari, Elliot Saba, João Felipe Santos, Abel Siqueira, Ulf Worsøe, David Zeng
- Julia developers