# Nonlinear Optimization Modeling using JuMP and JuliaOpt

Miles Lubin

AIChE Webinar

April 5, 2016

# What we'll cover

- JuliaOpt organization
- JuMP, MathProgBase, Convex.jl, Pajarito
- Focus on infrastructure. Important if you:
  - are an advanced user
  - want to extend or build on top of our software
- Not a tutorial

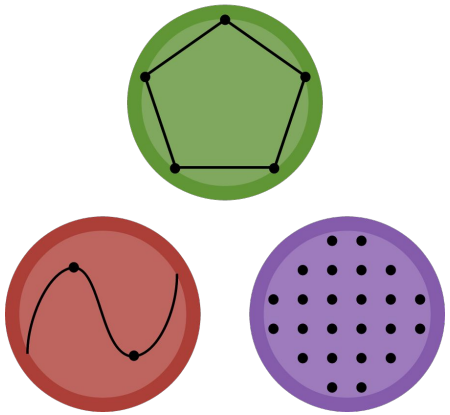
# Why choose Julia?

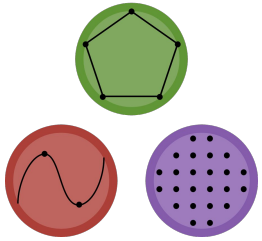
Lubin and Dunning, “Computing in Operations Research using Julia”, IJOC, 2015

- “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…

I (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

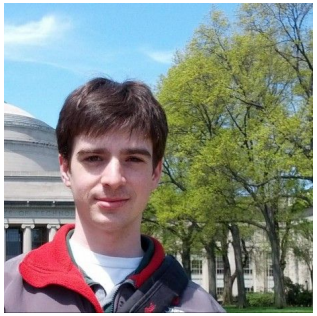
Optim.jl

LsqFit.jl

CoinOptServices.jl

AmplNLWriter.jl

# JuMP



- Modeling language for linear, mixed-integer, conic (SOCP, SDP), nonlinear
  - Like AMPL, GAMS, Pyomo (assume familiar)
- http://jump.readthedocs.org/
- JuliaOpt Notebooks
- Benchmarks: http://arxiv.org/abs/1508.01982
- Used for teaching in 10+ universities

# Automatic differentiation of user-defined functions

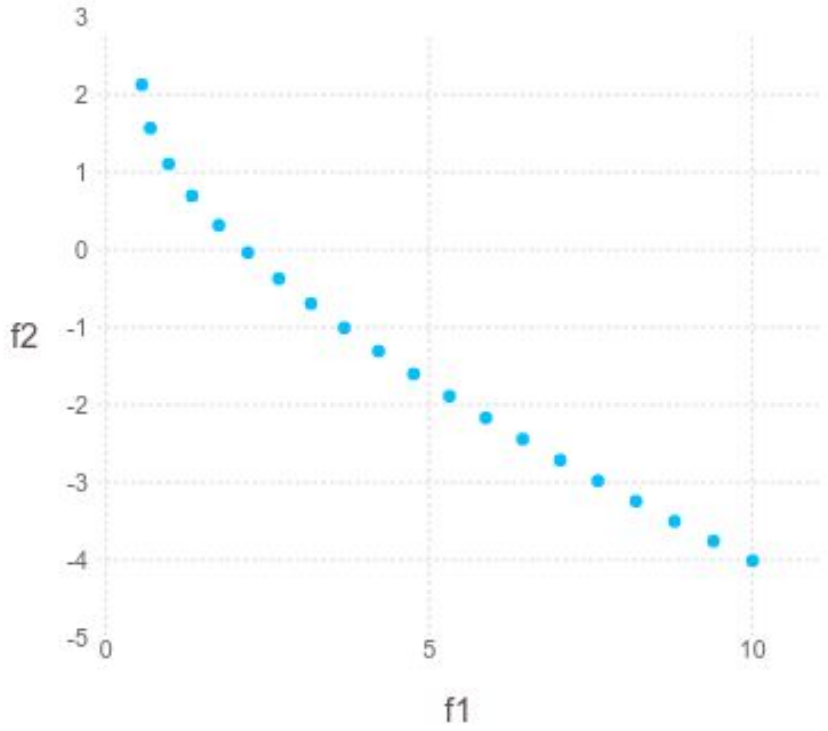
```
function squareroot(x)
    z = x # Initial starting point for Newton
    while abs(z*z - x) > 1e-13
        z = z - (z*z-x)/(2z)
    end
    return z
end

registerNLFunction(:squareroot, 1, squareroot,
                   autodiff=true)
```

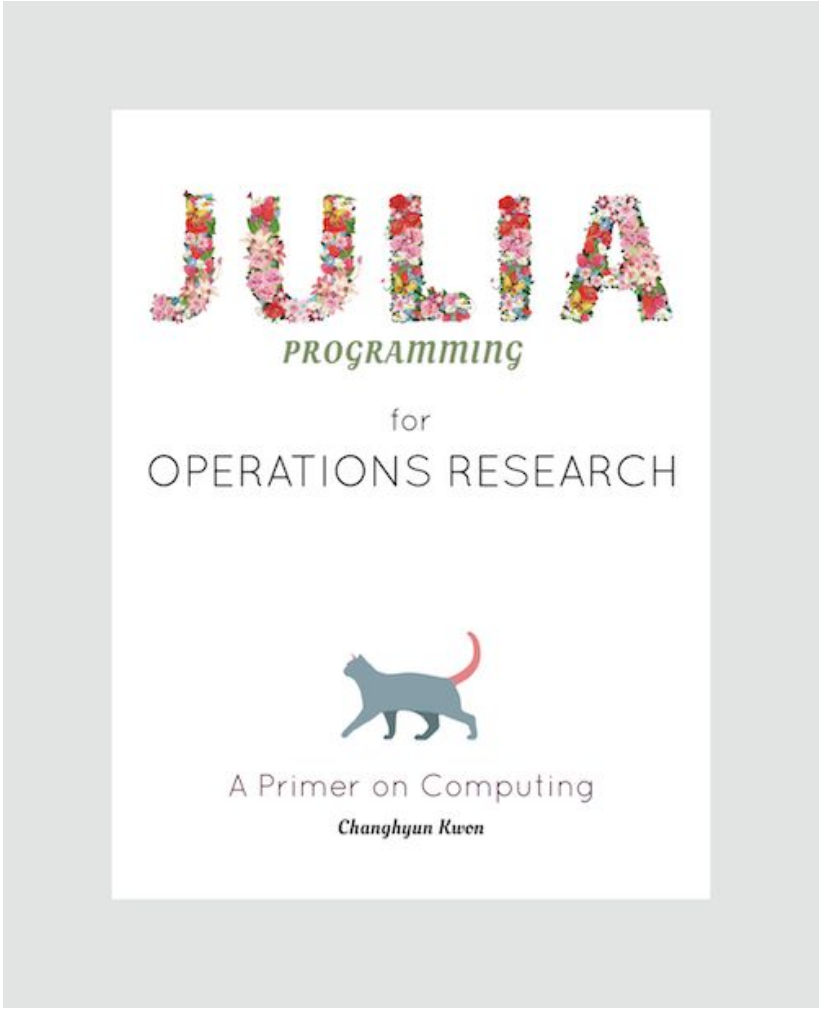
# Automatic differentiation of user-defined functions

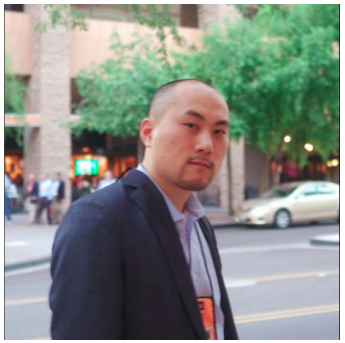
```
m = Model()
@defVar(m, x[1:2], start=0.5)
@setObjective(m, Max, sum(x))
@addNLConstraint(m,
      squareroot(x[1]^2+x[2]^2) <= 1)
solve(m)
```

# MultiJuMP

https://github.com/anriseth/MultiJuMP.jl



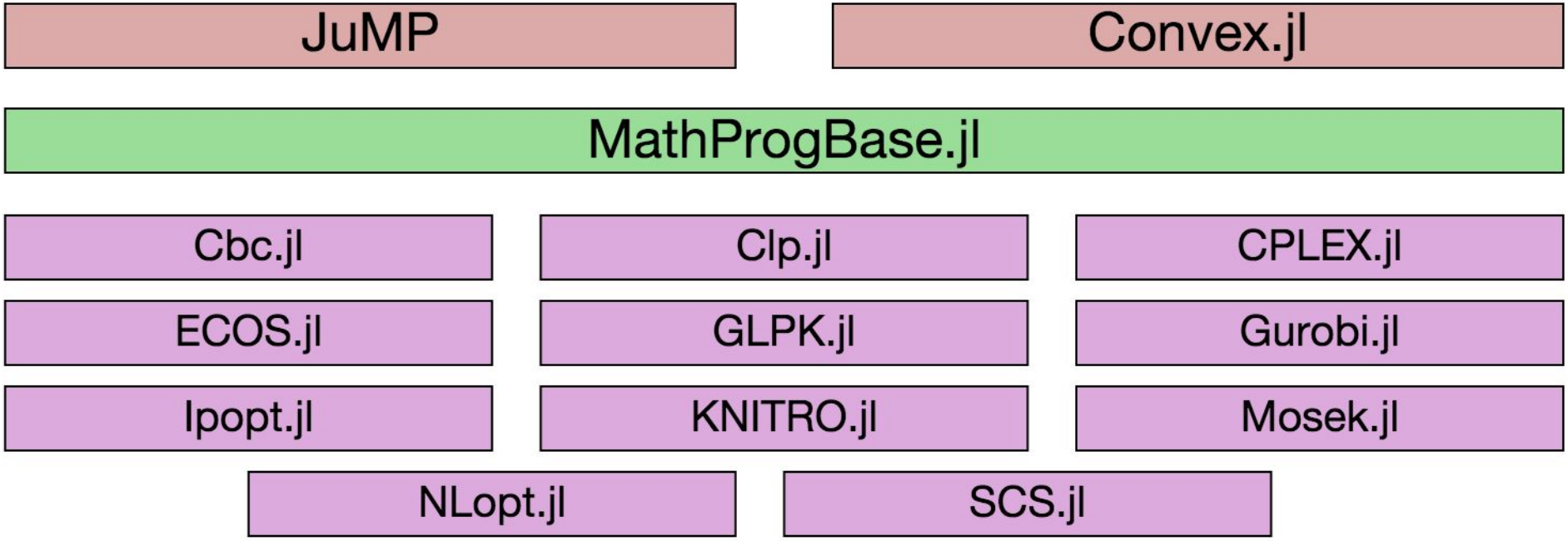


http://www.chkwon.net/julia/

# MathProgBase

- A standard interface which solver wrappers implement
  - Like COIN-OR/OSI
- You should care if you want to...
  - access properties of a solver not exposed by JuMP or Convex.jl (e.g., LP basis matrix)
  - query derivatives of a JuMP model
  - create a Julia wrapper for an existing solver
  - write a solver in Julia
  - create a modeling interface in Julia
  - access a Julia solver from another language


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, Ipopt.jl, etc...

# MathProgBase philosophy

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

# Diverse classes of solvers

- LinearQuadratic
- Conic
- Nonlinear

# LinearQuadratic

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

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

# Conic

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

- Linear, SOC, SDP, exponential, 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:

```
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)
```

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

```
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 get the best bound found during branch & bound?

MathProgBase docs

```
# With JuMP model object m, minimization problem
lowerbound = MathProgBase.getobjbound(getInternalModel(m))
```

# Wait, how do I get the best bound found during branch & bound?

MathProgBase docs

```
# With JuMP model object m, minimization problem
lowerbound = MathProgBase.getobjbound(getInternalModel(m))

# This is annoying, why not just have:
lowerbound = getobjbound(m)
```

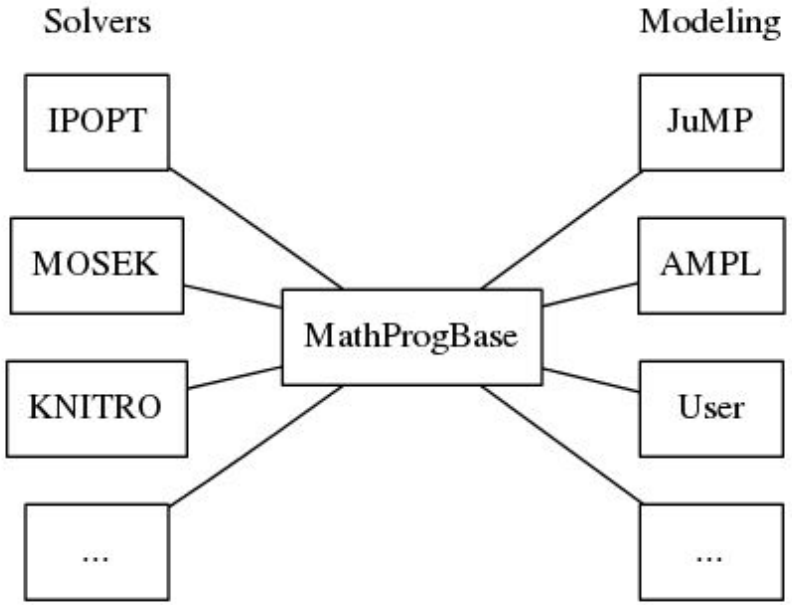
# Wait, how do I get the best bound found during branch & bound?

MathProgBase docs

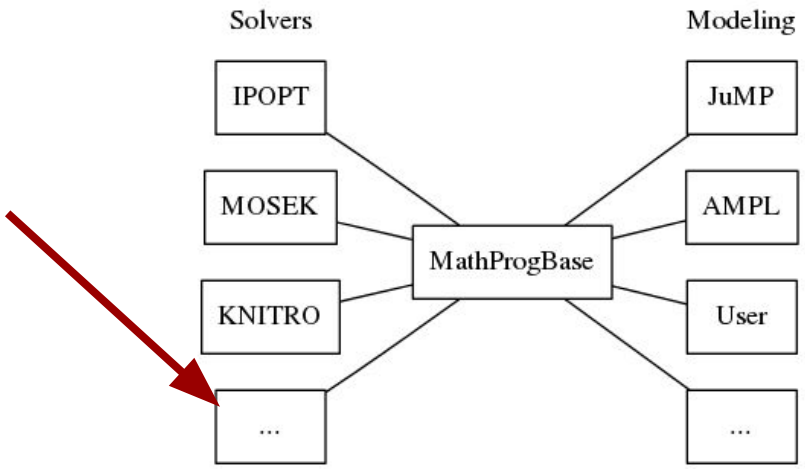
```
# With JuMP model object m, minimization problem
lowerbound = MathProgBase.getobjbound(getInternalModel(m))

# This is annoying, why not just have:
lowerbound = getobjbound(m)
```

# Live fix!

# Nonlinear

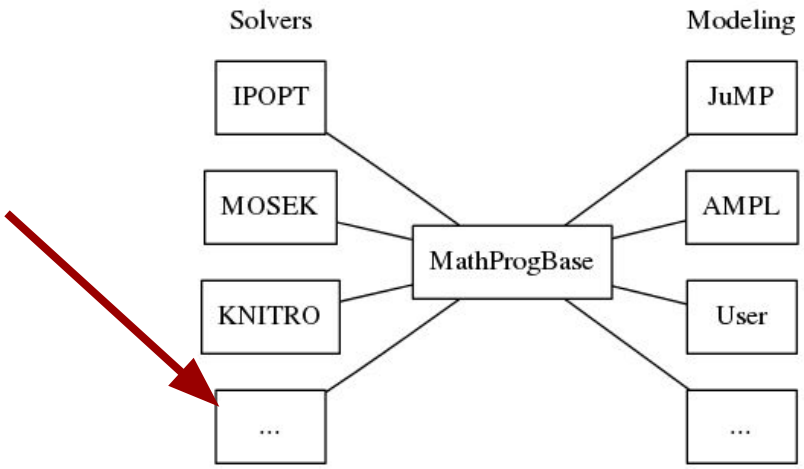

Solvers
Modeling
IPOPT
MOSEK
KNITRO
...
MathProgBase
JuMP
AMPL
User
...

# Nonlinear




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

# Nonlinear




Demo

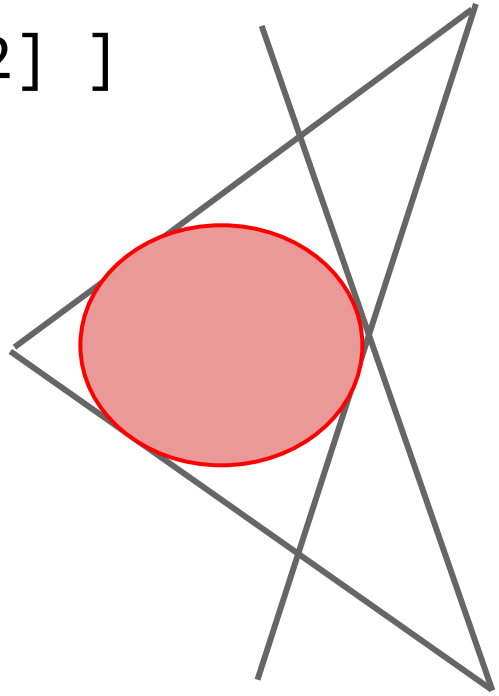
# Convex.jl



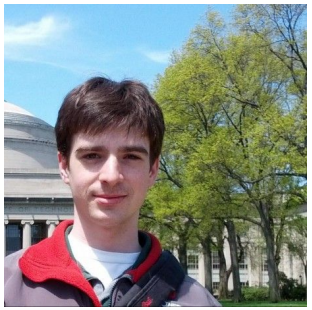

- Disciplined convex programming
  - Like CVX, CVXPY
- Translates convex problems into conic form, accessing advanced conic solvers
- http://dx.doi.org/10.1109/HPTCDL.2014.5

# Max Volume Inscribed Ellipsoid

```
using Convex

a = [ [ 2, 1]; [ 2,-1]; [-1, 2]; [-1,-2] ]
B = Variable(2,2)
d = Variable(2)
p = maximize(logdet(B))
for i in 1:4
  p.constraints += norm(B*a[i]) +
                        dot(a[i],d) <= 1
end
solve!(p)
println(B.value)
println(d.value)
```



# Pajarito



- New pure-Julia solver for mixed-integer convex optimization
- https://github.com/mlubin/Pajarito.jl
- Given nonlinear input, replaces Bonmin's outer approximation and branch-and-cut algorithms
- Given **conic** input, implements new conic outer approximation algorithm

# Pajarito

- Fastest mixed-integer convex solver on benchmark instances *when called from Convex.jl*
- http://arxiv.org/abs/1511.06710

# Pajarito and MathProgBase

- Conic algorithm
- Nonlinear algorithm
- Subproblem solvers
  - Plug in any MathProgBase-compatible MILP, NLP, and conic solvers
    - Bonmin supports only Ipopt + Cbc/CPLEX
  - Critical for fast development

# Thanks to

- David Anthoff, Carlo Baldassi, Chris Coey, Oscar Dowson, Jack Dunn, Steven G. Johnson, Tony Kelman, Dahua Lin, Yee Sian Ng, Brendan O'Donoghue, Leonardo Taccari, Elliot Saba, João Felipe Santos, Abel Siqueira, Ulf Worsøe
- Julia developers

- http://www.juliaopt.org/
- julia-opt google group