Developing new optimization methods with packages from the JuliaSmoothOptimizers organization

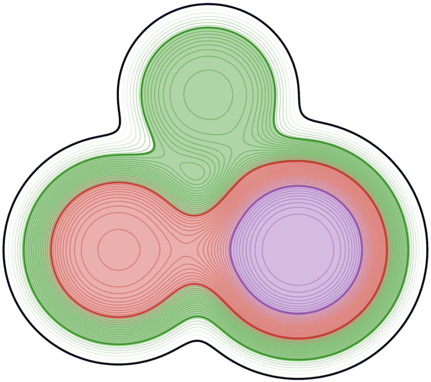
Second Annual JuMP-dev Workshop

Abel Soares Siqueira
Federal University of Paraná - Curitiba/PR - Brazil

Dominique Orban
GERAD/Polytechnique Montréal - Montreal - Canada

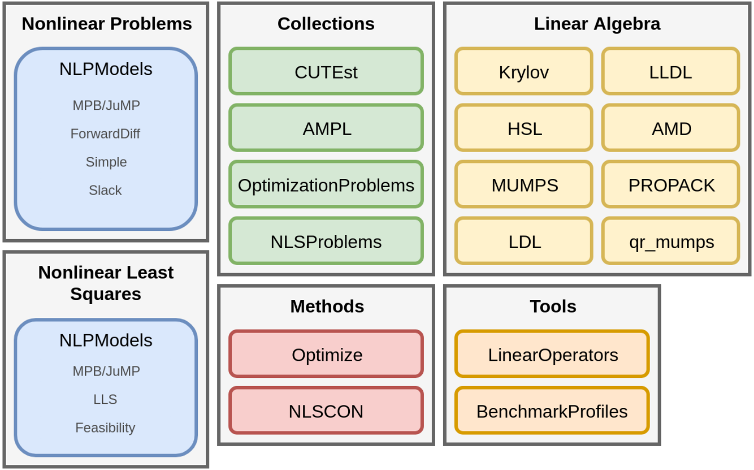
June 28, 2018

Julia Smooth Optimizers



- Linear Algebra and Optimization tools for developers/researchers/academics;
- Created from our demands;
- Integrates with MPB/JuMP ([1]);
- We also develop solvers, focusing on large-scale;
- Similar work done previously in PythonOptimizers.

Julia Smooth Optimizers



NLPModels

- Defines nonlinear programming models and unified API to access them;
- Some models provide sparse derivatives;
- Some models provide efficient matrix-free products (i.e. no explicit matrix used);
- Creating a new model type should be easy.

NLPModels

```julia
# Short
adnlp = ADNLPModel(x -> (x[1] - 1)^2 + 100 * (x[2] - x[1]^2)^2, [-1.2; 1.0])

# ROSENBR from the CUTEst list of problem. Also uses CUTEst.jl
ctnlp = CUTEstModel("ROSENBR")

# using JuMP -> sparse Hessian
m = Model()
@variable(m, x[1:2])
setvalue(x, [-1.2; 1.0])
@NLobjective(m, Min, (x[1] - 1)^2 + 100 * (x[2] - x[1]^2)^2)
mpnlp = MathProgNLPModel(m);
```

Unified API

```
for (name, nlp) in [("Autodiff", adnlp),
                    ("CUTEst", ctnlp),
                    ("JuMP", mpnlp)]
    x = nlp.meta.x0
    println("$name")
    print("fx = $(obj(nlp, x)), ")
    println("gx = $(grad(nlp, x))")
    println("Hx = $(hess(nlp, x))")
end
finalize(ctnlp)
```

```
Autodiff
fx = 24.199999999999996, gx = [-215.6, -88.0]
Hx = [1330.0 0.0; 480.0 200.0]
CUTEst
fx = 24.199999999999996, gx = [-215.6, -88.0]
Hx =
  [1, 1]  =  1330.0
  [2, 1]  =  480.0
  [2, 2]  =  200.0
JuMP
fx = 24.199999999999996, gx = [-215.6, -88.0]
Hx =
  [1, 1]  =  1330.0
  [2, 1]  =  480.0
  [2, 2]  =  200.0
```

Unified API

```
function newton(nlp :: AbstractNLPModel)
  x = copy(nlp.meta.x0)
  fx = obj(nlp, x)
  gx = zeros(nlp.meta.nvar)
  grad!(nlp, x, gx)

  while norm(gx) > 1e-4
    Hx = Symmetric(hess(nlp, x), :L)
    d = -Hx \ gx
    if dot(d, gx) >= 0.0
      d = -gx
    end

    xt = x + d
    ft = obj(nlp, xt)
    slope = dot(d, gx)
```

```
    t = 1.0
    while !(ft < fx + 1e-2 * t * slope)
      t *= 0.25
      xt = x + t * d
      ft = obj(nlp, xt)
    end

    x .= xt
    fx = ft
    grad!(nlp, x, gx)
  end

  return x, fx, gx # Unified output also available in other packages
end

for nlp in [adnlp; ctnlp; mpnlp]
  x, fx, gx = newton(nlp)
  # ...
```

Unified API

$$\min \quad f(x) \qquad \text{s. to} \qquad c_L \leq c(x) \leq c_U, \quad \ell \leq x \leq u$$

$f(x)$	`obj`
$\nabla f(x)$	`grad, grad!`
$\nabla^2 f(x)$	`hess, hess_op, hess_op!, hess_coord, hprod, hprod!`
$f(x), \nabla f(x)$	`objgrad, objgrad!`
$c(x)$	`cons, cons!`
$f(x), c(x)$	`objcons, objcons!`
$J(x) = \nabla c(x)$	`jac, jac_op, jac_op!, jac_coord, jprod, jprod!, jtprod, jtprod!`
$\nabla^2_{xx} L(x, y)$	`hess, hess_op, hess_coord, hprod, hprod!`

Unified API

$$\min \quad f(x) \qquad \text{s. to} \qquad \begin{array}{l} c_L \leq c(x) \leq c_U \\ \ell \leq x \leq u \end{array}$$

```
nlp = ADNLPModel(x->sum(x.^4), [1.0; 2.0; 3.0; 4.0],
                lvar=[-1; -Inf; 0; 0], uvar=[1; 0; Inf; 0],
                c=x->[sum(x); prod(x)], lcon=[1.0; 1.0], ucon=[1.0; Inf])

meta = nlp.meta
x = meta.x0
l, u = meta.lvar, meta.uvar
cl, cu = meta.lcon, meta.ucon
vartypes = meta.ifix, meta.ifree, meta.ilow, meta.iupp, meta.irng
contypes = meta.jfix, meta.jfree, meta.jlow, meta.jupp, meta.jrng
```

MPB solvers integration

```julia
using Ipopt

nlp = CUTEstModel("ROSENBR")
model = NLPtoMPB(nlp, IpoptSolver(print_level=0))
MathProgBase.optimize!(model)
finalize(nlp)
println("#f = $(neval_obj(nlp))")
println("#g = $(neval_grad(nlp))")
println("#H = $(neval_hess(nlp))")
println("#Hp = $(neval_hprod(nlp))")
println("sum = $(sum_counters(nlp))")
```

NLPModels

- Specific models can be created by extending ‘AbstractNLPModel‘, and defining the specific API functions.
- Can create models on top of models, such as ‘SlackModel‘

$$
\begin{array}{rl} \min & f(x) \\ \text{s. to} & c(x) \geq 0 \end{array} \quad \Rightarrow \quad \begin{array}{rl} \min & f(x) \\ \text{s. to} & c(x) - s = 0 \\ & s \geq 0. \end{array}
$$

Nonlinear Least Squares

$$\min \quad f(x) = \|F(x)\|^2 \qquad \text{s. to} \qquad \begin{array}{l} c_L \leq c(x) \leq c_U \\ \ell \leq x \leq u \end{array}$$

- API for $F(x)$ and derivatives;
- Extensions of NLPModels;
- Main models:
  - `LLSModel(A, b)`: $F(x) = Ax - b$;
  - `ADNLSModel(F, x0)`: ForwardDiff models;
  - `FeasibilityResidual(nlp)`: $F(x)$ defined from constraints;
  - `MathProgNLSModel(model, vec_of_expr)`:

Nonlinear Least Squares

```
model = Model()
@variable(model, x[1:2])
setvalue(x, [-1.2; 1.0])
@NLexpression(model, F1, x[1] - 1)
@NLexpression(model, F2, x[2] - x[1]^2)
@NLconstraint(model, x[1]^2 + x[2]^2 == 1)
nls = MathProgNLSModel(model, [F1; F2])

x = nls.meta.x0
Fx = residual(nls, x)
Jx = jac_residual(nls, x)
```

Collections of problems

- `CUTEst.jl` provides access to all of 1305 CUTEst ([2]) problems, the CUTEst API, and NLPModels API. Contains a tool for selecting problems;
- `OptimizationProblems.jl` stores NLP problems in JuMP format. Some problems from CUTEst are implemented. More are welcome;
- `NLSProblems.jl` stores NLS problems. Moré-Garbow-Hillstrom ([3]) and some other models are implemented. More are welcome;
- No way to classify and select problems from these last two yet - `abelsiqueira/NLPClass.jl` was an attempt.

Linear Operators

- Provides matrix-like entities;
- Useful for factorization-free methods, wrapping Jacobian/Hessian-vector products;
- Can wrap around matrices, generalizing;
- Implements LBFGS and LSR1;
- Lazy: `(A * B) * v` is the same as `A * (B * v)`.

```
T = LinearOperator{Float64}(nlp.meta.ncon, nlp.meta.nvar,
                            false, false, # Symmetric? hermitian?
                            v->jprod(nlp, x, v),  # T * v    prod(v)
                            v->jtprod(nlp, x, v), # T.' * v  tprod(v)
                            v->jtprod(nlp, x, v)) # T' * v    ctprod(v)
```

Linear Operators

- `jac_op(nlp, x)` returns a `LinearOperator` with `jprod` and `jtprod`;
- `hess_op(nlp, x)` is similar for `hprod`;

Krylov

- Iterative methods for linear systems, least squares and least norm problems;
- Also accept trust-region constraint;
- Works for matrices and Linear Operators;

$$\nabla^2 f(x) d = -\nabla f(x)$$

```
Bx = hess_op(nlp, x)
gx = grad(nlp, x)
d, cg_stats = cg(Bx, -gx)
```

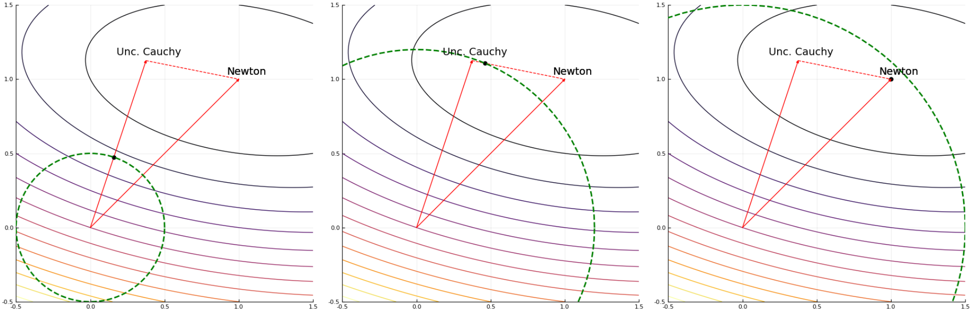
$$\min_y \|J(x)^T y - \nabla f(x)\|^2$$

```
Jx = jac_op(nlp, x)
gx = grad(nlp, x)
y, cgls_stats = cgls(Jx', gx)
```

Krylov

$$\min_d \frac{1}{2} d^T B d + d^T g \quad \text{s.to} \quad \|d\| \leq \Delta$$

```
# Given B, g, radius
d, cg_stats = cg(B, -g, radius=radius)
```



Krylov + LBFGS

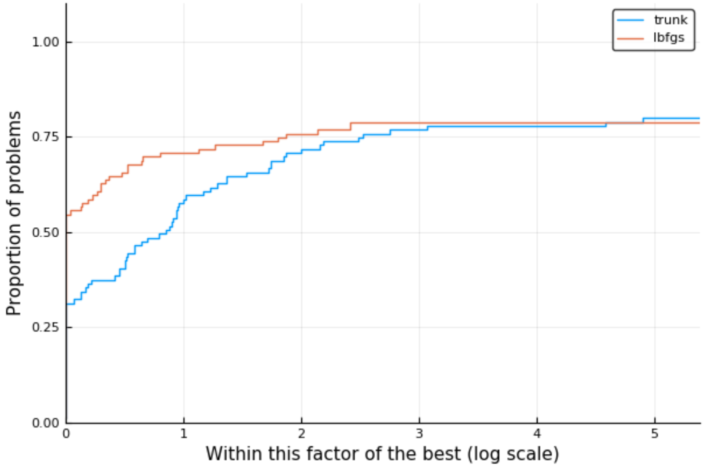
```
B = LBFGSOperator(n, scaling=false)
H = InverseLBFGSOperator(n, scaling=false)

s = rand(n)
y = rand(n)
push!(B, s, y)
push!(H, s, y)

v = ones(n)
x, _ = cg(B, v)
Hv = H * v
x - Hv # Almost zero
```

Optimize

- Tools for line search and trust region methods;
- Implementations of optimization methods, focusing on large-scale;
- Currently `lbfgs` and `trunk` for unconstrained problems, and `tron` for bound constrained problems.
- Tools for benchmarking (together with BenchmarkProfiles);

```
pnames = CUTEst.select(min_var=100, max_var=10_000, contype=:unc)
problems = (CUTEstModel(p) for p in pnames)
solvers = Dict{Symbol,Function}(:lbfgs => lbfgs, :trunk => trunk)
bmark_args = Dict(:max_f => 10_000, :max_time => 30.0)
stats, p = bmark_and_profile(solvers, problems, bmark_args=bmark_args)
png(p, "perfprof")
```

Optimize


trunk
lbfgs
1.00
0.75
0.50
0.25
0.00
Proportion of problems
0
1
2
3
4
5
Within this factor of the best (log scale)

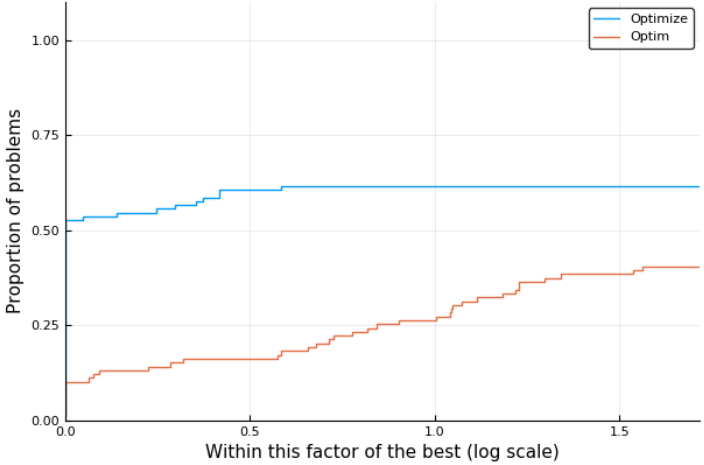
Optimize vs. Optim

```
function optim_method(nlp :: AbstractNLPModel; kwargs...)
  f(x) = obj(nlp, x)
  g!(storage, x) = grad!(nlp, x, storage)
  Dt = time()
  output = optimize(f, g!, nlp.meta.x0, LBFGS(m = 5),
                    Optim.Options(g_tol = 1e-8,
                                  iterations = 10_000_000,
                                  f_calls_limit = 10_000))
  Dt = time() - Dt
  status = output.g_converged ? :first_order : :unknown
  return GenericExecutionStats(status, nlp, solution=output.minimizer,
                               objective=output.minimum, dual_feas=output.g_residual,
                               iter=output.iterations, elapsed_time=Dt)
end
```

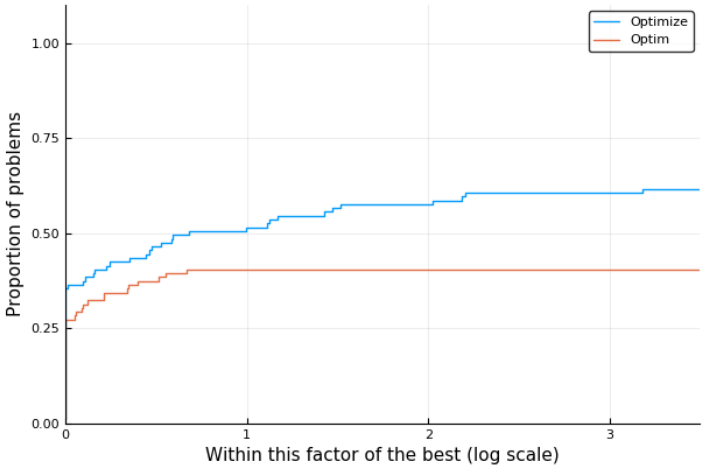
Optimize vs. Optim

```
solvers = Dict{Symbol,Function}(:Optim => optim_method, :Optimize => lbfgs)
pnames = sort(CUTEst.select(min_var=100, max_var=10_000, contype=:unc))
bmark_args = Dict(:atol => 1e-8, rtol => 0.0, :max_f => 10_000, :max_time => 30.0)

problems = (CUTEstModel(p) for p in pnames)
stats, p = bmark_and_profile(solvers, problems)
png(p, "vs-optim-sum-counters")
stats, p = bmark_and_profile(solvers, problems, cost=stat->stat.elapsed_time)
png(p, "vs-optim-time")
```

Optimize vs. Optim - Functions evaluations, from 100 to 10000 variables



Optimize vs. Optim - Elapsed time, from 100 to 10000 variables



Summary

- NLPModels for easy model creation and access;
- CUTEst + others for easy access to problems;
- LinearOperators + Krylov for factorization free methods;
- Optimize for benchmarking and subproblem solvers.

Future Work

- NLSCON: Constrained Nonlinear Least Squares Solver
- Code updates: Julia 0.7/1.0, MOI interface, and type stability;
- General constraints solver and stable version of Optimize;
- Parameter optimization;
- More problems natively in Julia, and problem classification;
- CUDA.

References

I. Dunning, J. Huchette, and M. Lubin, "JuMP: A modeling language for mathematical optimization", *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017. DOI: 10.1137/15M1020575.

N. I. Gould, D. Orban, and P. L. Toint, "CUTEst: A constrained and unconstrained testing environment with safe threads for mathematical optimization", *Comput. Optim. Appl.*, vol. 60, no. 3, pp. 545–557, 2015. DOI: 10.1007/s10589-014-9687-3.

J. J. Moré, B. S. Garbow, and K. E. Hillstrom, "Testing unconstrained optimization software", *ACM Trans. Math. Softw.*, vol. 7, no. 1, pp. 17–41, 1981. DOI: 10.1145/355934.355936.

Thank you