Calling nonlinear and MINLP solvers from Julia

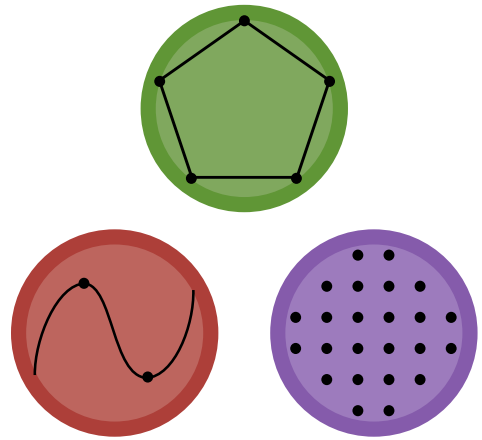
Interfaces, formats, expression trees and AD tools

Tony Kelman

University of California, Berkeley

kelman@berkeley.edu

https://github.com/tkelman



ISMP, Pittsburgh PA

July 16, 2015

In the spirit of reproducible research

You can get to solving a (small) MINLP before the end of this talk [1]

1. Install a binary of the latest Julia release at
   `http://julialang.org/downloads`
2. Open a Julia session and run:

```
Pkg.add("JuMP")
Pkg.add("CoinOptServices")
using JuMP, CoinOptServices, Compat
include(Pkg.dir("JuMP","test","solvers.jl"));
include(Pkg.dir("JuMP","test","nonlinear.jl"));
```

Will come back to an example problem later

[1] On Mac or Windows, where we download precompiled solver binaries
On Linux, has to compile solvers from source which takes a few minutes

Talk outline

- MINLP problem statement, standard form
- Expression trees
- Automatic differentiation
- Solver interfaces
  - Data formats, COIN OSiL and AMPL nl
  - APIs, connecting to high level languages
- Julia implementations
  - JuMP and MathProgBase
  - CoinOptServices and AmplNLWriter
- Example problems
- Performance evaluation

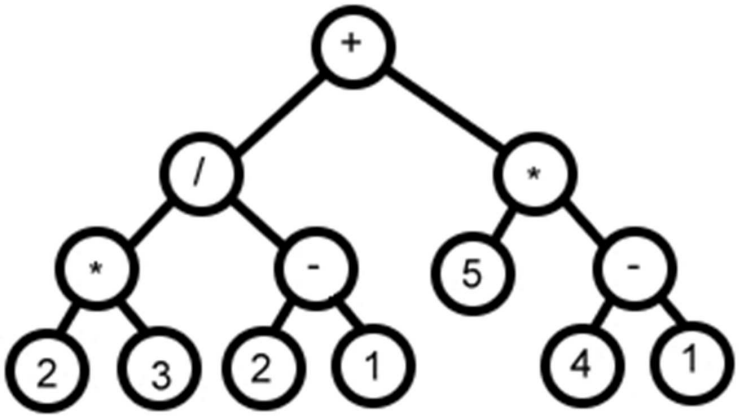
Mixed integer nonlinear programming

MINLP problem statement

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0 \\ & h(x) = 0 \\ & x \in \mathbb{R}^n \\ & x_i \in \mathbb{Z} \; \forall \; i \in \mathcal{I} \end{aligned}$$

- For this talk $f(x)$, $g(x)$ may be non-convex, $h(x)$ may be nonlinear
- Assume that $f(x), g(x), h(x)$ are algebraic expressions
- What is the "standard form" to encode this problem?

Expression trees



Expression tree for 2*3/(2-1)+5*(4-1)

Julia has a first-class expression type and quote operator, it already does exactly what we want here

Real macros and metaprogramming (`@addNLConstraint` in JuMP) take much of the work out of this

```
julia> dump(:( 2*3/(2-1)+5*(4-1) ), 6)
Expr
  head: Symbol call
  args: Array(Any,(3,))
    1: Symbol +
    2: Expr
      head: Symbol call
      args: Array(Any,(3,))
        1: Symbol /
        2: Expr
          head: Symbol call
          args: Array(Any,(3,))
            1: Symbol *
            2: Int64 2
            3: Int64 3
          typ: Any
        3: Expr
          head: Symbol call
          args: Array(Any,(3,))
            1: Symbol -
            2: Int64 2
            3: Int64 1
          typ: Any
      typ: Any
    3: Expr
      head: Symbol call
      args: Array(Any,(3,))
        1: Symbol *
        2: Int64 5
        3: Expr
          head: Symbol call
          args: Array(Any,(3,))
            1: Symbol -
            2: Int64 4
            3: Int64 1
          typ: Any
      typ: Any
  typ: Any
```

Automatic (algorithmic) differentiation

- Large body of literature and implementations (ADOL-C, CppAD, AMPL ASL, ReverseDiffSparse.jl, etc), see `http://www.mit.edu/~mlubin/informs2014-nlp.pdf` for more details
- Given expression trees of objective and constraint functions $f(x), g(x), h(x)$, AD provides efficient calculation method for sparse constraint Jacobian and Lagrangian Hessian
- For general nonlinear problems that don't fit in a special form, you should be using AD unless you have a good reason not to

Data formats for representing expression trees

- State of the art in practice: AMPL .nl format, see "Hooking Your Solver to AMPL" and "Writing .nl Files" technical reports by David M. Gay
- High level human-readable .mod file converted to low level expression tree .nl format by proprietary AMPL modeling layer
- Solvers link to open-source AMPL solver library (ASL) for function, Jacobian, Hessian evaluation from .nl file
- Now available in Julia thanks to Jack Dunn's AmplNLWriter.jl package, JuMP can create .nl files

Creating a .nl file from JuMP

Copy-paste from `http://bit.do/ismp-minlp`

```
Pkg.add("AmplNLWriter")
using JuMP, AmplNLWriter
# Solve test problem 1 (Synthesis of processing system) in
# M. Duran & I.E. Grossmann, "An outer approximation algorithm for
# a class of mixed integer nonlinear programs", Mathematical
# Programming 36, pp. 307-339, 1986.
m = Model(solver = BonminNLSolver())
x_U = [2,2,1]
@defVar(m, x_U[i] >= x[i=1:3] >= 0)
@defVar(m, y[4:6], Bin)
@setNLObjective(m, Min, 10 + 10*x[1] - 7*x[3] + 5*y[4] + 6*y[5] +
    8*y[6] - 18*log(x[2]+1) - 19.2*log(x[1]-x[2]+1))
@addNLConstraints(m, begin
    0.8*log(x[2] + 1) + 0.96*log(x[1] - x[2] + 1) - 0.8*x[3] >= 0
    log(x[2] + 1) + 1.2*log(x[1] - x[2] + 1) - x[3] - 2*y[6] >= -2
    x[2] - x[1] <= 0
    x[2] - 2*y[4] <= 0
    x[1] - x[2] - 2*y[5] <= 0
    y[4] + y[5] <= 1
end)
status = solve(m)
```

The resulting .nl file

```
g3 1 1 0
 6 6 1 6 0 0
 2 1
 0 0
 2 2 2
 0 0 0 0
 3 0 0 0 0
 16 6
 0 0
 0 0 0 0 0
C0
o0
o2
n0.8
o43
o0
v1
n1
o2
n0.96
o43
o0
o1
v0
v1
n1
C1
o0
o43
o0
v1
n1
o2
n1.2
o43
o0
o1
v0
v1
n1
C2
n0
C3
n0
C4
n0
C5
n0
O0 0
o0
o1
o16
o2
n18
o43
o0
v1
n1
o2
n19.2
o43
o0
o1
v0
v1
n1
n10
d6
0 0
1 0
2 0
3 0
4 0
5 0
x6
0 0.0
1 0.0
2 0.0
3 0.0
4 0.0
5 0.0
r
2 0.0
2 -2.0
1 0.0
1 0.0
1 0.0
1 1.0
b
0 0.0 2.0
0 0.0 2.0
0 0.0 1.0
0 0.0 1.0
0 0.0 1.0
0 0.0 1.0
k5
4
9
11
13
15
J0 3
0 0.0
1 0.0
2 -0.8
J1 4
0 0.0
1 0.0
2 -1.0
5 -2.0
J2 2
0 -1.0
1 1.0
J3 2
1 1.0
3 -2.0
J4 3
0 1.0
1 -1.0
4 -2.0
J5 2
3 1.0
4 1.0
G0 6
0 10.0
1 0.0
2 -7.0
3 5.0
4 6.0
5 8.0
```

COIN-OR solvers and Optimization Services

- Comprehensive stack of cutting-edge open source solvers and libraries
- CLP: continuous linear programming
- CBC: mixed-integer linear programming
- Ipopt: continuous nonlinear programming
- Bonmin: evaluation based mixed-integer nonlinear programming
- Couenne: expression tree based mixed-integer nonlinear programming
- CppAD: automatic differentiation
- Optimization Services: standardized interchange formats, remote solution protocols, solver-independent interfaces

Optimization Services OSiL format - 1

- XML based, human readable
- `OSSolverService` driver uses CppAD for Jacobian, Hessian
- Same Julia code, except for:

```
using CoinOptServices
m = Model(solver = OsilBonminSolver())
```

```
<?xml version="1.0" encoding="utf-8"?>
<osil xmlns="os.optimizationservices.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="os.optimizationservices.org
    http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
  <instanceHeader>
    <description>generated by CoinOptServices.jl</description>
  </instanceHeader>
  <instanceData>
    <variables numberOfVariables="6">
      <var lb="0.0" ub="2.0" type="C"/>
      <var lb="0.0" ub="2.0" type="C"/>
      <var lb="0.0" ub="1.0" type="C"/>
      <var lb="0.0" ub="1.0" type="B"/>
      <var lb="0.0" ub="1.0" type="B"/>
      <var lb="0.0" ub="1.0" type="B"/>
    </variables>
    <objectives numberOfObjectives="1">
      <obj maxOrMin="min" numberOfObjCoef="0"/>
    </objectives>
    <constraints numberOfConstraints="6">
      <con lb="0.0"/>
      <con lb="0.0"/>
      <con ub="0.0"/>
      <con ub="0.0"/>
      <con ub="0.0"/>
      <con ub="0.0"/>
    </constraints>
```

Optimization Services OSiL format - 2

```
<nonlinearExpressions numberOfNonlinearExpressions="7">
  <nl idx="-1">
    <minus>
      <minus>
        <sum>
          <minus>
            <plus>
              <number value="10"/>
              <variable idx="0" coef="10"/>
            </plus>
            <variable idx="2" coef="7"/>
          </minus>
          <variable idx="3" coef="5"/>
          <variable idx="4" coef="6"/>
          <variable idx="5" coef="8"/>
        </sum>
        <times>
          <number value="18"/>
          <ln>
            <plus>
              <variable idx="1"/>
              <number value="1"/>
            </plus>
          </ln>
        </times>
      </minus>
      <times>
        <number value="19.2"/>
        <ln>
          <plus>
            <minus>
              <variable idx="0"/>
              <variable idx="1"/>
            </minus>
            <number value="1"/>
          </plus>
        </ln>
      </times>
    </minus>
  </nl>
```

```
<nl idx="0">
  <minus>
    <minus>
      <plus>
        <times>
          <number value="0.8"/>
          <ln>
            <plus>
              <variable idx="1"/>
              <number value="1"/>
            </plus>
          </ln>
        </times>
        <times>
          <number value="0.96"/>
          <ln>
            <plus>
              <minus>
                <variable idx="0"/>
                <variable idx="1"/>
              </minus>
              <number value="1"/>
            </plus>
          </ln>
        </times>
      </plus>
      <variable idx="2" coef="0.8"/>
    </minus>
    <number value="0"/>
  </minus>
</nl>
```

Optimization Services OSiL format - 3

```
<nl idx="1">
  <minus>
    <minus>
      <minus>
        <plus>
          <ln>
            <plus>
              <variable idx="1"/>
              <number value="1"/>
            </plus>
          </ln>
          <times>
            <number value="1.2"/>
            <ln>
              <plus>
                <minus>
                  <variable idx="0"/>
                  <variable idx="1"/>
                </minus>
                <number value="1"/>
              </plus>
            </ln>
          </times>
        </plus>
        <variable idx="2"/>
      </minus>
      <variable idx="5" coef="2"/>
    </minus>
    <number value="-2"/>
  </minus>
</nl>
<nl idx="2">
  <minus>
    <minus>
      <variable idx="1"/>
      <variable idx="0"/>
    </minus>
    <number value="0"/>
  </minus>
</nl>
<nl idx="3">
  <minus>
    <minus>
      <variable idx="1"/>
      <variable idx="3" coef="2"/>
    </minus>
    <number value="0"/>
  </minus>
</nl>
<nl idx="4">
  <minus>
    <minus>
      <minus>
        <variable idx="0"/>
        <variable idx="1"/>
      </minus>
      <variable idx="4" coef="2"/>
    </minus>
    <number value="0"/>
  </minus>
</nl>
<nl idx="5">
  <minus>
    <plus>
      <variable idx="3"/>
      <variable idx="4"/>
    </plus>
    <number value="1"/>
  </minus>
</nl>
  </nonlinearExpressions>
 </instanceData>
</osil>
```

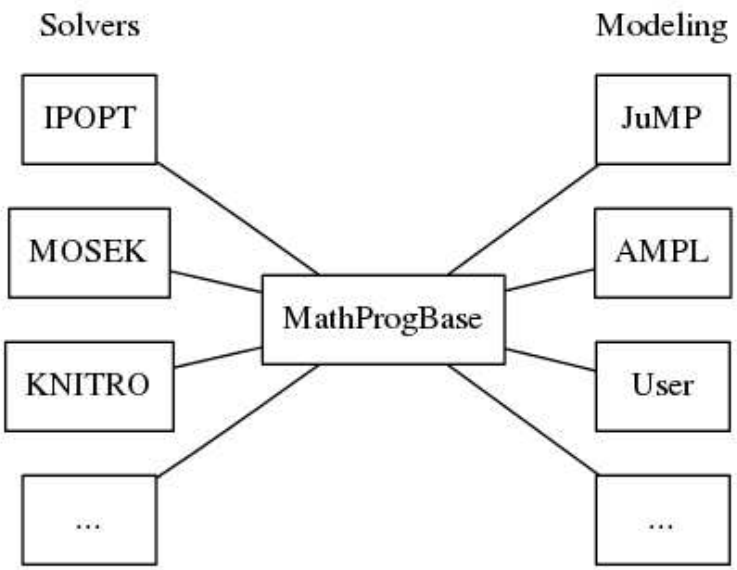
Solver APIs and connecting to high level languages

- Convenient for users to work in high level languages for model formulation, data processing, function evaluation (if fast enough)
- Modern solvers in C++ are hard to use from other languages unless they have a dedicated C API
  - e.g. Bonmin doesn't need expression trees, but only has a C++ API
  - Julia solver interfaces very easy when there is a C API: `ccall((function_name, library_name), return_type, (input_types,), inputs)`
  - Direct Julia to C++ interface being worked on, but still unstable
- Julia is both fast and high level, convenient for developers too
  - No need to prototype in Matlab/Python then rewrite in C/C++ for performance
  - Pure-Julia AD in ReverseDiffSparse.jl is fast enough that it's not the bottleneck
  - Can stay in memory, avoid .nl or .osil files, allow efficient re-solves
  - Solvers entirely in Julia? Very little code to connect to MathProgBase, JuMP, Convex.jl

Solver APIs and connecting to high level languages

- Convenient for users to work in high level languages for model formulation, data processing, function evaluation (if fast enough)
- Modern solvers in C++ are hard to use from other languages unless they have a dedicated C API
  - e.g. Bonmin doesn't need expression trees, but only has a C++ API
  - Julia solver interfaces very easy when there is a C API: `ccall((function_name, library_name), return_type, (input_types,), inputs)`
  - Direct Julia to C++ interface being worked on, but still unstable
- Julia is both fast and high level, convenient for developers too
  - No need to prototype in Matlab/Python then rewrite in C/C++ for performance
  - Pure-Julia AD in ReverseDiffSparse.jl is fast enough that it's not the bottleneck
  - Can stay in memory, avoid .nl or .osil files, allow efficient re-solves
  - Solvers entirely in Julia? Very little code to connect to MathProgBase, JuMP, Convex.jl

MathProgBase extensibility and new solvers



- Unified Julia API for all solver packages and modeling tools to talk to
- Solvers call `eval_jac_g` to evaluate constraint Jacobian
- AmplNLWriter and CoinOptServices call `obj_expr`, `constr_expr` to get Julia expression trees for objective and constraint functions
- More solvers, more modeling environments (AmplNLReader.jl), extensions of JuMP and Convex.jl

Larger example problem

- Goddard rocket control problem from COPS3 (continuous NLP)
  `http://bit.do/rocket-jump`
- Comparing AD function evaluation performance between
  1. Pure-Julia ReverseDiffSparse.jl:
     `solver=IpoptSolver(print_timing_statistics="yes")`
  2. ASL via nl: `solver=IpoptNLSolver()`
  3. CppAD via osil:
     `solver=OsilSolver(OSOption("print_timing_statistics", "yes"))`

Pure-Julia AD

```
solver=IpoptSolver(print_timing_statistics="yes")

Total CPU secs in IPOPT (w/o function evaluations)   =      0.719
Total CPU secs in NLP function evaluations           =      0.412

Timing Statistics:

OverallAlgorithm.....................:      1.131 (sys:      0.000 wall:      1.131)
 PrintProblemStatistics..............:      0.021 (sys:      0.000 wall:      0.021)
 InitializeIterates..................:      0.101 (sys:      0.000 wall:      0.101)
 UpdateHessian.......................:      0.317 (sys:      0.000 wall:      0.317)
 OutputIteration.....................:      0.138 (sys:      0.000 wall:      0.138)
 UpdateBarrierParameter..............:      0.003 (sys:      0.000 wall:      0.003)
 ComputeSearchDirection..............:      0.418 (sys:      0.000 wall:      0.418)
 ComputeAcceptableTrialPoint.........:      0.040 (sys:      0.000 wall:      0.040)
 AcceptTrialPoint....................:      0.000 (sys:      0.000 wall:      0.000)
 CheckConvergence....................:      0.071 (sys:      0.000 wall:      0.071)
PDSystemSolverTotal..................:      0.417 (sys:      0.000 wall:      0.417)
 PDSystemSolverSolveOnce.............:      0.396 (sys:      0.000 wall:      0.396)
 ComputeResiduals....................:      0.018 (sys:      0.000 wall:      0.018)
 StdAugSystemSolverMultiSolve........:      0.445 (sys:      0.000 wall:      0.445)
 LinearSystemScaling.................:      0.000 (sys:      0.000 wall:      0.000)
 LinearSystemSymbolicFactorization...:      0.044 (sys:      0.000 wall:      0.044)
 LinearSystemFactorization...........:      0.000 (sys:      0.000 wall:      0.000)
 LinearSystemBackSolve...............:      0.058 (sys:      0.000 wall:      0.058)
 LinearSystemStructureConverter......:      0.000 (sys:      0.000 wall:      0.000)
  LinearSystemStructureConverterInit:      0.000 (sys:      0.000 wall:      0.000)
Function Evaluations.................:      0.412 (sys:      0.000 wall:      0.412)
 Objective function..................:      0.000 (sys:      0.000 wall:      0.000)
 Objective function gradient.........:      0.000 (sys:      0.000 wall:      0.000)
 Equality constraints................:      0.024 (sys:      0.000 wall:      0.024)
 Inequality constraints..............:      0.000 (sys:      0.000 wall:      0.000)
 Equality constraint Jacobian........:      0.072 (sys:      0.000 wall:      0.072)
 Inequality constraint Jacobian......:      0.000 (sys:      0.000 wall:      0.000)
 Lagrangian Hessian..................:      0.316 (sys:      0.000 wall:      0.316)
```

ASL via nl

```
solver=IpoptNLSolver()
```

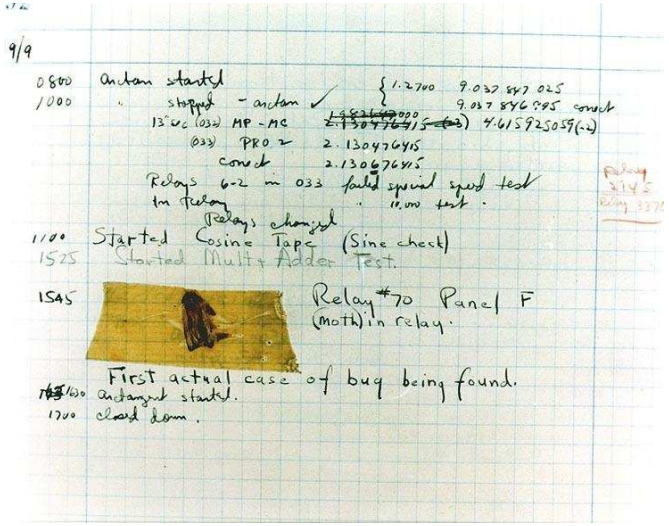
```
Total CPU secs in IPOPT (w/o function evaluations)   =      0.732
Total CPU secs in NLP function evaluations           =      0.244

Timing Statistics:

OverallAlgorithm.....................:      0.976 (sys:      0.000 wall:      0.976)
 PrintProblemStatistics..............:      0.021 (sys:      0.000 wall:      0.021)
 InitializeIterates..................:      0.055 (sys:      0.000 wall:      0.055)
 UpdateHessian.......................:      0.174 (sys:      0.000 wall:      0.175)
 OutputIteration.....................:      0.139 (sys:      0.000 wall:      0.139)
 UpdateBarrierParameter..............:      0.001 (sys:      0.000 wall:      0.001)
 ComputeSearchDirection..............:      0.470 (sys:      0.000 wall:      0.470)
 ComputeAcceptableTrialPoint.........:      0.057 (sys:      0.000 wall:      0.057)
 AcceptTrialPoint....................:      0.000 (sys:      0.000 wall:      0.000)
 CheckConvergence....................:      0.032 (sys:      0.000 wall:      0.032)
PDSystemSolverTotal..................:      0.470 (sys:      0.000 wall:      0.470)
 PDSystemSolverSolveOnce.............:      0.454 (sys:      0.000 wall:      0.454)
 ComputeResiduals....................:      0.011 (sys:      0.000 wall:      0.011)
 StdAugSystemSolverMultiSolve........:      0.483 (sys:      0.000 wall:      0.483)
 LinearSystemScaling.................:      0.000 (sys:      0.000 wall:      0.000)
 LinearSystemSymbolicFactorization...:      0.026 (sys:      0.000 wall:      0.026)
 LinearSystemFactorization...........:      0.000 (sys:      0.000 wall:      0.000)
 LinearSystemBackSolve...............:      0.058 (sys:      0.000 wall:      0.058)
 LinearSystemStructureConverter......:      0.000 (sys:      0.000 wall:      0.000)
  LinearSystemStructureConverterInit:      0.000 (sys:      0.000 wall:      0.000)
Function Evaluations.................:      0.244 (sys:      0.000 wall:      0.244)
 Objective function..................:      0.000 (sys:      0.000 wall:      0.000)
 Objective function gradient.........:      0.003 (sys:      0.000 wall:      0.003)
 Equality constraints................:      0.039 (sys:      0.000 wall:      0.039)
 Inequality constraints..............:      0.000 (sys:      0.000 wall:      0.000)
 Equality constraint Jacobian........:      0.028 (sys:      0.000 wall:      0.028)
 Inequality constraint Jacobian......:      0.000 (sys:      0.000 wall:      0.000)
 Lagrangian Hessian..................:      0.174 (sys:      0.000 wall:      0.174)
```

CppAD via osil

```
solver=OsilSolver(OSOption("print_timing_statistics", "yes"))
```



- Failed to converge, initial conditions not set right by `OSSolverService`
- Function evaluation time several hundred times slower per iteration
- Interesting disagreeement on "Number of nonzeros in Lagrangian Hessian" for same problem
  1. ReverseDiffSparse.jl: 48800
  2. ASL: 11214
  3. OSSolverService: 8811

Conclusions

- There are some bugs in Optimization Services, need to report them along with .nl and .osil test files, and spend some time profiling in C++
- Only a few hundred lines of Julia code to implement expression tree to .osil (or .nl) conversion
- For now use CoinOptServices to install Bonmin and Couenne solver binaries, AmplNLWriter for MINLP's, and either AmplNLWriter or in-memory interfaces for continuous NLP's
- Want advice on writing Julia bindings to your solver, or how to start prototyping algorithms in Julia? `julia-opt@googlegroups.com`