Key Contributor

Matthew Wilhelm
PhD Candidate
PSOR Lab, Dept. of Chemical and Biomolecular Eng.
University of Connecticut

EAGO.jl developer
Outline

• Motivation
  – Why deterministic global optimization?
• Background
  – What is Julia and why’d we choose it?
• EAGO.jl: Deterministic global optimization in Julia
  – Architecture, core features/capabilities
  – Advanced optimization formulations
  – Examples
  – Performance
• Conclusions
Motivation

- Optimization problems (especially in OR) are often formulated for convexity/concavity
Motivation

- Optimization problems (especially in OR) are often formulated for convexity/concavity
  - May limit applications of optimal decision-making
Motivation

• Optimization problems (especially in OR) are often formulated for convexity/concavity
  – We don’t always need to find global optima, but when we do, we need fast, accessible, and flexible software
Background: Julia

What is Julia?
Background: Julia

What is Julia?

- New open-source programming language built specifically for scientific and high-performance computing
Background: Julia

What is Julia?

- New open-source programming language built specifically for scientific and high-performance computing
  - Combines ease of use of high-level languages (MATLAB) with speed of low-level languages (C, FORTRAN)
Background: Julia

What is Julia?

- New open-source programming language built specifically for scientific and high-performance computing
  - Combines ease of use of high-level languages (MATLAB) with speed of low-level languages (C, FORTRAN)
    - Can “feel” like a scripting language (dynamically-typed, but can also declare types) (“generic programming”)

EURO 2019 - June 24, 2019
Background: Julia

What is Julia?

• New open-source programming language built specifically for scientific and high-performance computing
  – Combines ease of use of high-level languages (MATLAB) with speed of low-level languages (C, FORTRAN)
    • Can “feel” like a scripting language (dynamically-typed, but can also declare types) (“generic programming”)
• Paradigm: multiple dispatch
  – define function behavior across argument types
Background: Julia
Background: Julia

What is Julia?

• Written in Julia (even primitives)
• Can natively call C and FORTRAN without wrapper code or APIs
• Automatic code generation
• Metaprogramming
What is Julia?

• Written in Julia (even primitives)
• Can natively call C and FORTRAN without wrapper code or APIs
• Automatic code generation
• Metaprogramming
  – Julia is represented as a data structure of the language itself
  – We can write a program to transform and generate its own code
Background: EAGO

Why did we choose Julia?
Background: EAGO

Why did we choose Julia?

• Global optimization algorithms must be very fast and utilize many complicated data types
  – E.g., derivatives, bounds, relaxations
Background: EAGO

Why did we choose Julia?

• Global optimization algorithms must be very fast and utilize many complicated data types
  – E.g., derivatives, bounds, relaxations
• For research and prototyping purposes, we want algorithms to be easy to implement and test
Why did we choose Julia?

• Global optimization algorithms must be very fast and utilize many complicated data types
  – E.g., derivatives, bounds, relaxations

• For research and prototyping purposes, we want algorithms to be easy to implement and test

• We often encounter optimization formulations which are difficult to represent in standard modeling languages (GAMS, AMPL)
  – E.g., embedded simulation
Background: EAGO

Why did we choose Julia?

• Global optimization algorithms must be very fast and utilize many complicated data types
  – E.g., derivatives, bounds, relaxations

• For research and prototyping purposes, we want algorithms to be easy to implement and test

• We often encounter optimization formulations which are difficult to represent in standard modeling languages (GAMS, AMPL)
  – E.g., embedded simulation

• We may want to invoke a global solver as part of another algorithm
  – E.g., semi-infinite programming
Why did we choose Julia?
  • It’s open-source and free for non-commercial use!
Why did we choose Julia?

- It’s open-source and free for non-commercial use!
- There exists an open-source advanced modeling language: JuMP.jl
Why did we choose Julia?

- It’s open-source and free for non-commercial use!
- There exists an open-source advanced modeling language: JuMP.jl

How do you get EAGO?

From Julia package manager:

```
(julia) pkg> add EAGO
```

```
julia> using Pkg
```

```
julia> Pkg.add("EAGO")
```
Why did we choose Julia?
• It’s open-source and free for non-commercial use!
• There exists an open-source advanced modeling language: JuMP.jl

How do you get EAGO?
From Julia package manager:
```julia
(v1.1) pkg> add EAGO
```
```julia
julia> using Pkg
julia> Pkg.add("EAGO")
```

From GitHub:
https://www.github.com/PSORLab/EAGO.jl
EAGO.jl: Architecture and Features

EAGO Formulation Tools

Input:
JuMP AML

JuMP Model Object

Expression Graph Builder

Expression Rewriter

Solver

Standard Form (LP, QP, ...)

Input:
Julia

Script Bridge

Reformulation Rules

Relaxation Library

Search Heuristics

User-Defined or From Standard Library
EAGO.jl: Advanced Formulations

- User-defined functions
EAGO.jl: Architecture and Features

- Core solver: branch-and-bound
EAGO.jl: Architecture and Features

• Bounds and Relaxations
  – Interval arithmetic
  – McCormick-based relaxations
    • Multivariate, generalized, and differentiable
    • Implicit functions
  – αBB & Auxiliary variables coming soon to latest version
EAGO.jl: Architecture and Features

- Constraint propagation on directed graphs
- Optimization-based bound tightening
  - Aggressive bound tightening
  - Greedy ordering for solutions
  - Readily extendable to non-affine relaxations
- Interval Newton & Parametric Interval
  Newton Contractors in software library
- Specialized contractors for linear and quadratic forms
EAGO.jl: Ex. CST Hybridization

- Custom bounding routines
  - User-defined convex relaxation provides convex hull of nonconvex objective

EAGO.jl: Ex. CST Hybridization

- Custom bounding routines
  - User-defined convex relaxation provides convex hull of nonconvex objective
  - Specify user-defined lower-bounding problem instead of invoking full-space relaxation procedure

Suppose we have experimental heat capacity data of a two-component nonideal mixture and we wish to estimate the temperature-dependent parameters of a fundamental Gibbs free energy model.

\[
\min_{\mathbf{p} \in \mathcal{P}} \sum_{i,j} (c^\text{mod}_{i,j}(T_i, x_j, \mathbf{p}) - c^\text{exp}_{i,j}(T_i, x_j))^2 \\
\text{s.t. } c^\text{mod}_{i,j}(T_i, x_j, \mathbf{p}) = -T_i \frac{\partial^2 G(T_i, x_j, \mathbf{p})}{\partial T^2} \bigg|_{p_i}, \ \forall (i, j)
\]
Suppose we have experimental heat capacity data of a two-component nonideal mixture and we wish to estimate the temperature-dependent parameters of a fundamental Gibbs free energy model.

```julia
using EAGO, JuMP, ForwardDiff
R=8.314
CpA = 1.4*44.05
CpW = 4.184*18.02
T0=293.15
exGibbs(T,x1,p) = R*T*(x1*(1-x1)*2*(p[1]*T+p[2]*T^2+5*p[3]*log(T)) + (1-x1)*x1*2*(p[1]*T+p[2]*T^2+5*p[3]*log(T)))
GibbsA(T) = CpA*(T-T0)-T*CpA*log(T/T0)
GibbsW(T) = CpW*(T-T0)-T*CpW*log(T/T0)
Gibbs(T,x1,p) = x1*GibbsA(T)+(1-x1)*GibbsW(T)+R*T*(x1*log(x1)+(1-x1)*log(1-x1))+exGibbs(T,x1,p)
Cp(T,x1,p) = -T*ForwardDiff.derivative(T->ForwardDiff.derivative(T->Gibbs(T,x1,p),T),T)

function objective(T::Vector,x1::Vector,Cp_exp::Matrix,p...)
    SSE = 0.0
    for i=1:length(T)
        for j=1:length(x1)
            SSE += (Cp(T[i],x1[j],p)-Cp_exp[i,j])^2
        end
    end
    return SSE
end

fobj(p...) = objective(Tdata,x1data,Cp_exp,p...)
Suppose we have experimental heat capacity data of a two-component nonideal mixture and we wish to estimate the temperature-dependent parameters of a fundamental Gibbs free energy model.
EAGO.jl: Ex. Dynamic Optimization

- EAGO allows a large degree of functionality with a user-defined relaxation evaluator.
- Global optimization with differential equation constraints (supported by future EAGO_Differential.jl extensions):

Parameter Estimation for 1D Kinetic Problem

\[
\min_{p \in P} \sum_{i=1}^{n} \sum_{j=1}^{2} (x_j(p, t_i) - d_j(t_i))^2
\]

s.t. \[ \frac{dx}{dt}(p, t) = \left( \begin{array}{c} k_2 x_2 - k_1 x_1 \\ k_1 x_1 - k_2 x_2 \end{array} \right), t \in [0, 1] \]

\[ x(p, 0) = (p, 0) \]

\[ P = [0.8, 1] \]

EAGO.jl: Semi-Infinite Programming

- Support for nonconvex semi-infinite programming (design centering problems, etc.):

G.A. Watson (1983) DOI: 10.1007/978-3-642-46477-5_13
A. Mitsos (2009) DOI: 10.1080/02331934.2010.527970

\[
\begin{align*}
\min_{\mathbf{x}} f(\mathbf{x}) &= \frac{x_1^2}{3} + x_2^2 + \frac{x_1}{2} \\
\text{s.t. } (1 - x_1^2y^2)^2 - x_1y^2 - x_2^2 + x_2 &\leq 0, \quad \forall y \in [0,1] \\
\mathbf{x} &\in [-1000, 1000]^2
\end{align*}
\]

EAGO solves in ~2.5 seconds

```julia
using JuMP, EAGO
# Defines objective and semi-infinite constraint
f(x) = (x[1]^2)/3.0 + x[2]^2 + x[1]/2.0
# Defines bounds
xl = [-1000.0; -1000.0]; xu = [1000.0; 1000.0]
pL = [0.0]; pU = [1.0]
# Model for solving lower level problem
m = Model(with_optimizer(EAGO.Optimizer, verbosity = 0))
# Solve the semi-infinite program
output = explicit_sip_solve(f, gsIP, xl, xu, pL, pU, m)
```
EAGO.jl: Performance

- EAGO exhibits competitive performance on small benchmarking problem set
- Ubuntu 18.04LTS, LPsolver = CPLEX, NLPsolver = Ipopt, atol = 1E-3, rtol = 1E-3
- Xeon E3-1270v5 3.6GHz/4GHz (base/boost)

<table>
<thead>
<tr>
<th>Name</th>
<th>Variables</th>
<th>Inequalities</th>
<th>Equalities</th>
<th>Nonlinear Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>alkyl</td>
<td>15</td>
<td>0</td>
<td>7</td>
<td>$x, (\cdot)^2$, $\log, \log_{10}$, $\times, (\cdot)^2, (\cdot)^3, (\cdot)^4, (\cdot)^6$</td>
</tr>
<tr>
<td>bearing</td>
<td>14</td>
<td>0</td>
<td>12</td>
<td>$\sqrt{\cdot}$</td>
</tr>
<tr>
<td>BeckerLago</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>$\times$</td>
</tr>
<tr>
<td>ex3_1_1</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>$1, (\cdot)^2, (\cdot)^4$</td>
</tr>
<tr>
<td>ex4_1_9</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>$\times, (\cdot)^2, (\cdot)^4$</td>
</tr>
<tr>
<td>ex5_4_3</td>
<td>16</td>
<td>13</td>
<td>0</td>
<td>$\times, (\cdot)^3, (\cdot)^4, (\cdot)^6$</td>
</tr>
<tr>
<td>ex6_2_10</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>$\times, \log, \log_{10}$</td>
</tr>
<tr>
<td>ex6_2_11</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>$\times, \log, \log_{10}$</td>
</tr>
<tr>
<td>ex6_2_13</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>$\times, \log, \log_{10}$</td>
</tr>
<tr>
<td>ex6_2_14</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>$\times, \log, \log_{10}$</td>
</tr>
<tr>
<td>ex7_2_1</td>
<td>7</td>
<td>14</td>
<td>0</td>
<td>$\times, (\cdot)^2, (\cdot)^4$</td>
</tr>
<tr>
<td>ex7_2_3</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>$\times, (\cdot)^2, (\cdot)^4$</td>
</tr>
<tr>
<td>ex7_2_4</td>
<td>8</td>
<td>0</td>
<td>7</td>
<td>$\times, (\cdot)^2$</td>
</tr>
<tr>
<td>ex8_4_1</td>
<td>22</td>
<td>0</td>
<td>10</td>
<td>$\exp, \times, (\cdot)^2$</td>
</tr>
<tr>
<td>ex8_4_2</td>
<td>24</td>
<td>0</td>
<td>10</td>
<td>$\exp, \times, (\cdot)^2$</td>
</tr>
<tr>
<td>gold</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>$\times, (\cdot)^2$</td>
</tr>
<tr>
<td>hart6</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>$\exp, \times, (\cdot)^2$</td>
</tr>
<tr>
<td>meanvar</td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>$\exp, \times, (\cdot)^2$</td>
</tr>
<tr>
<td>Model13</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>$\times, (\cdot)^2, (\cdot)^4$</td>
</tr>
<tr>
<td>process</td>
<td>10</td>
<td>0</td>
<td>7</td>
<td>$\times, (\cdot)^2, (\cdot)^4$</td>
</tr>
</tbody>
</table>

Table 2: Descriptive Statistics for Problems Selected for Benchmarking

Performance Profile on Test Set

$$r_{p,s} = \min \left\{ \frac{t_{p,s}}{\tau} : s \in S \right\}$$
Conclusions

• EAGO is an extensible deterministic global optimization solver
  – Architected specifically for user-defined functions and routines
  – Performance comparable with state-of-the-art solvers
  – Open-source and free for non-commercial use

• Future:
  – Additional relaxations (αBB and AVM)
  – Release of dynamic optimization (optimal control) package
  – Implicit SIP algorithm (for simulation-based problems)
  – Integer variables

• Feature requests welcome on our GitHub!
Thank You – Any Questions?

- PSORLab@UCONN
- Debuggers: Prof. Kamil Khan and Student Huiyi Cao @ McMaster
- EURO 2019 Organizers
- Funding: University of Connecticut

https://www.psor.uconn.edu
https://www.github.com/PSORLab/EAGO.jl