UCONN
UNIVERSITY OF CONNECTICUT

EAGO.jl: Easy Advanced Global Optimization in Julia

Matthew D. Stuber
Assistant Professor
stuber@alum.mit.edu

EURO 2019

Process Systems and Operations Research Laboratory

Key Contributor

Matthew Wilhelm

PhD Candidate

PSOR Lab, Dept. of Chemical and Biomolecular Eng.

University of Connecticut

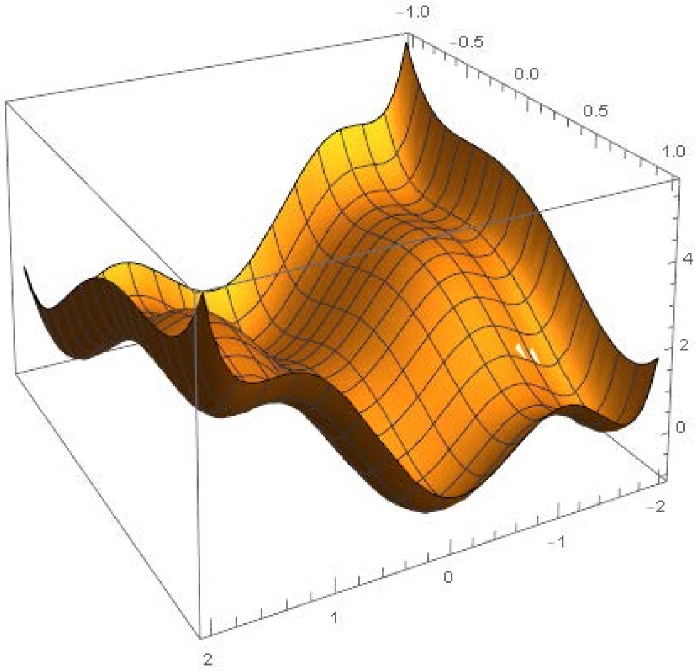
EAGO.jl developer

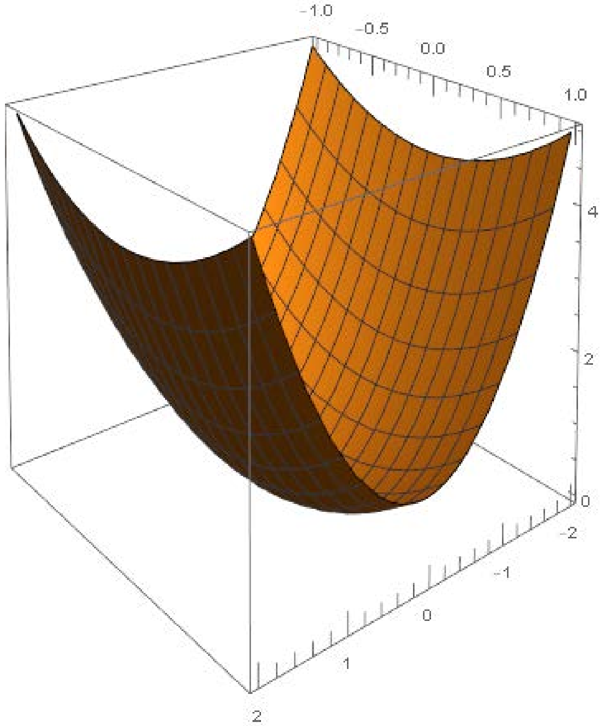
Outline

- 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

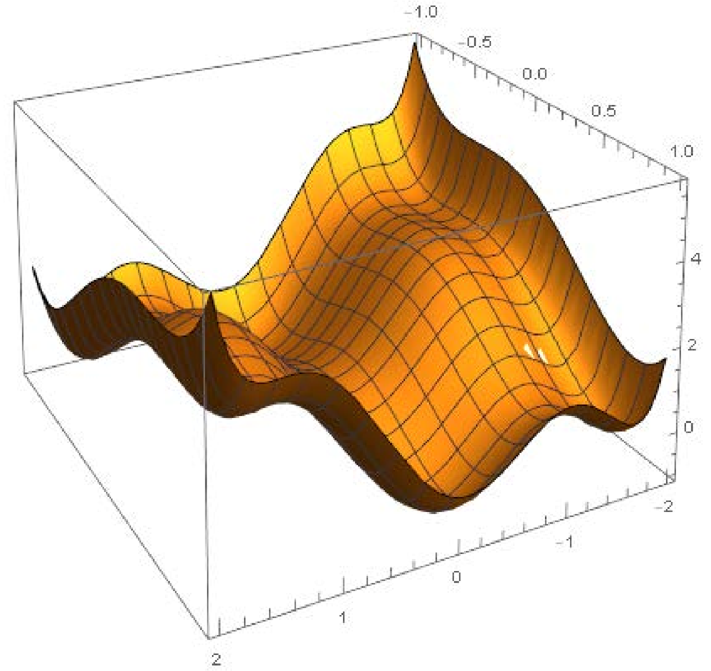
- Optimization problems (especially in OR) are often formulated for convexity/concavity

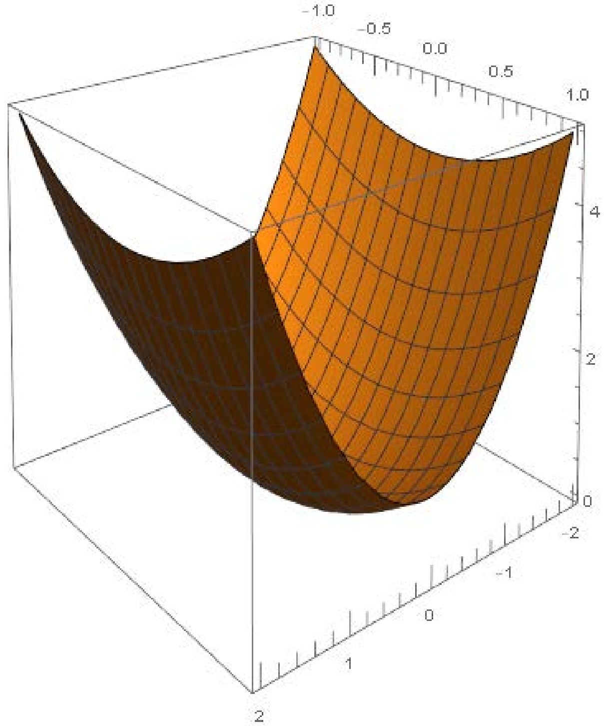




Motivation

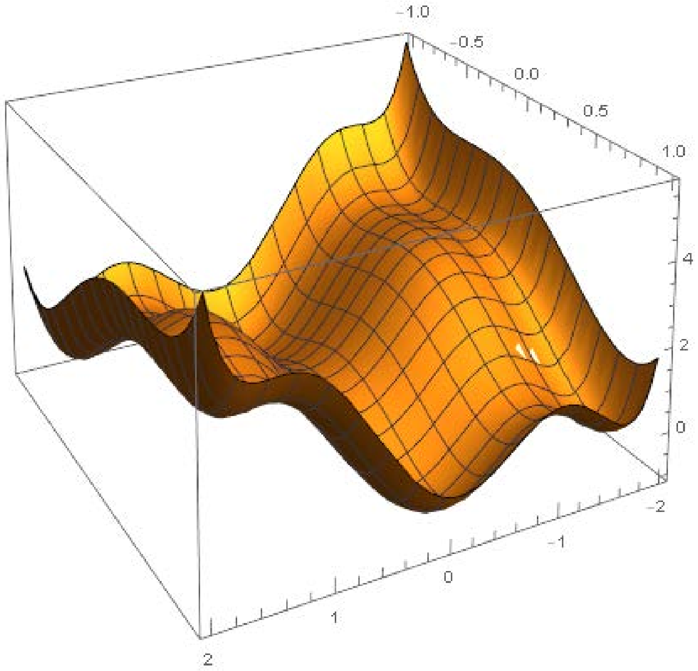
- Optimization problems (especially in OR) are often formulated for convexity/concavity
  - May limit applications of optimal decision-making

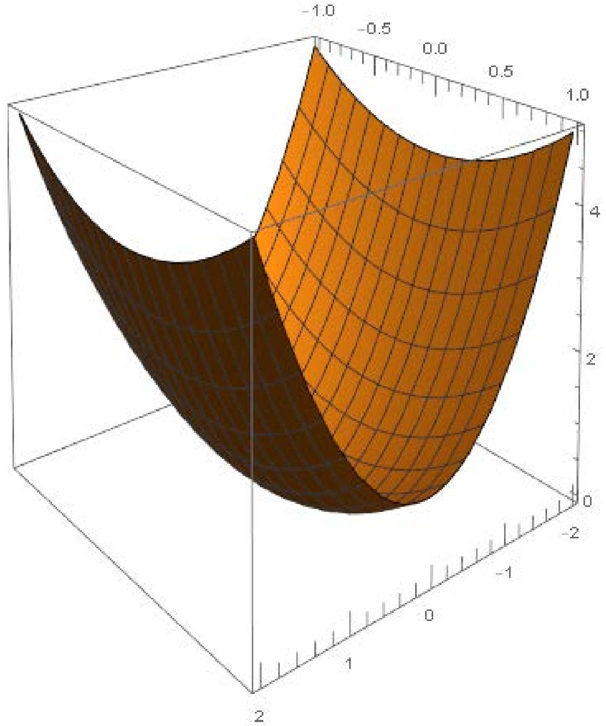




Motivation

- 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?

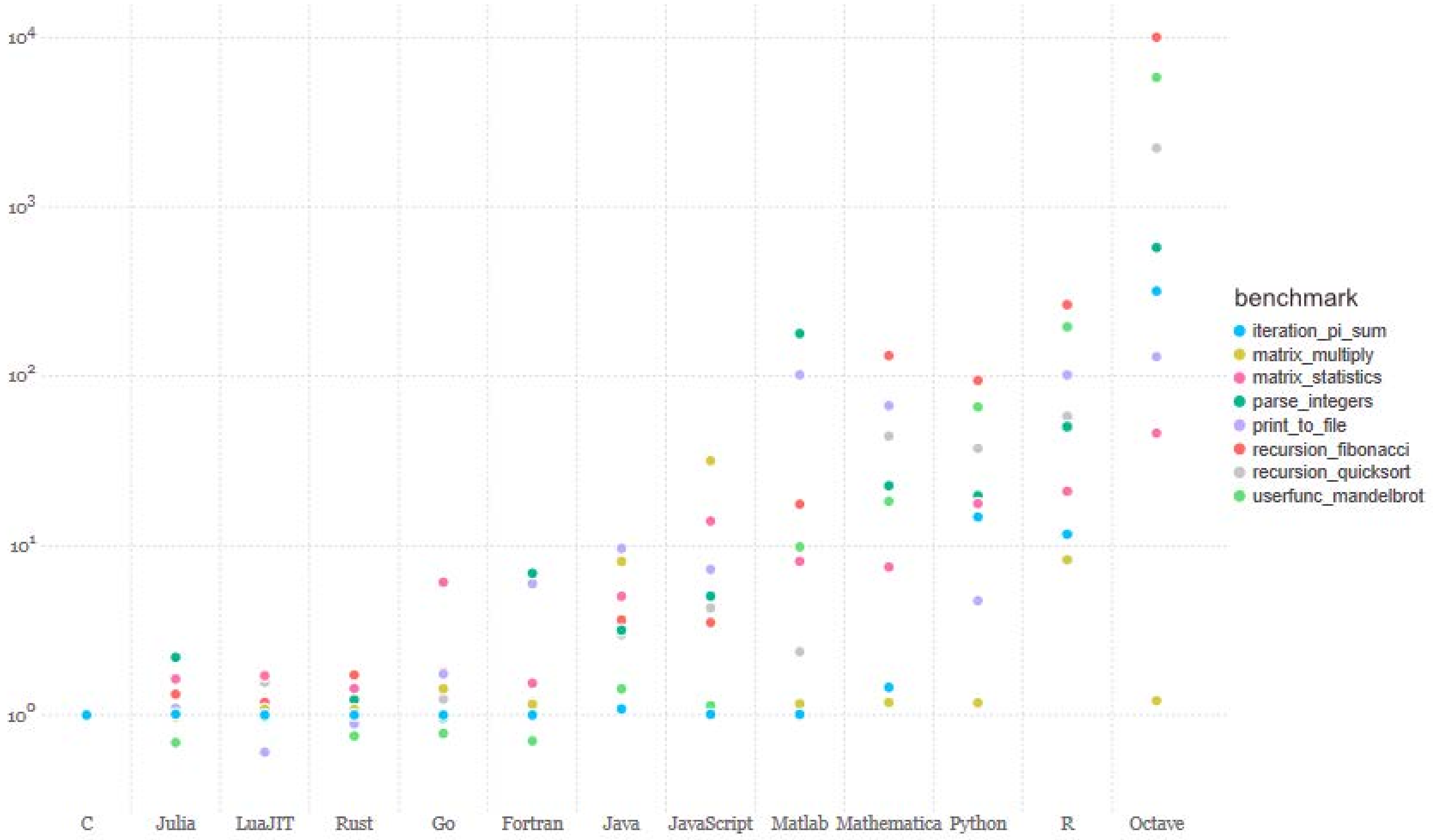
- New open-source programming language built specifically for scientific and high-performance computing
  - Combines ease of use of high-level languages (MATLAB) with speed of low-level languages (C, FORTRAN)
    - Can “feel” like a scripting language (dynamically-typed, but can also declare types) (“generic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

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



https://julialang.org

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

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
  - 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

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

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

Background: EAGO

Why did we choose Julia?

- It’s open-source and free for non-commercial use!

Background: 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

Background: 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:

```
(v1.1) pkg> add EAGO
```

```
julia> using Pkg

julia> Pkg.add("EAGO")
```

Background: 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:

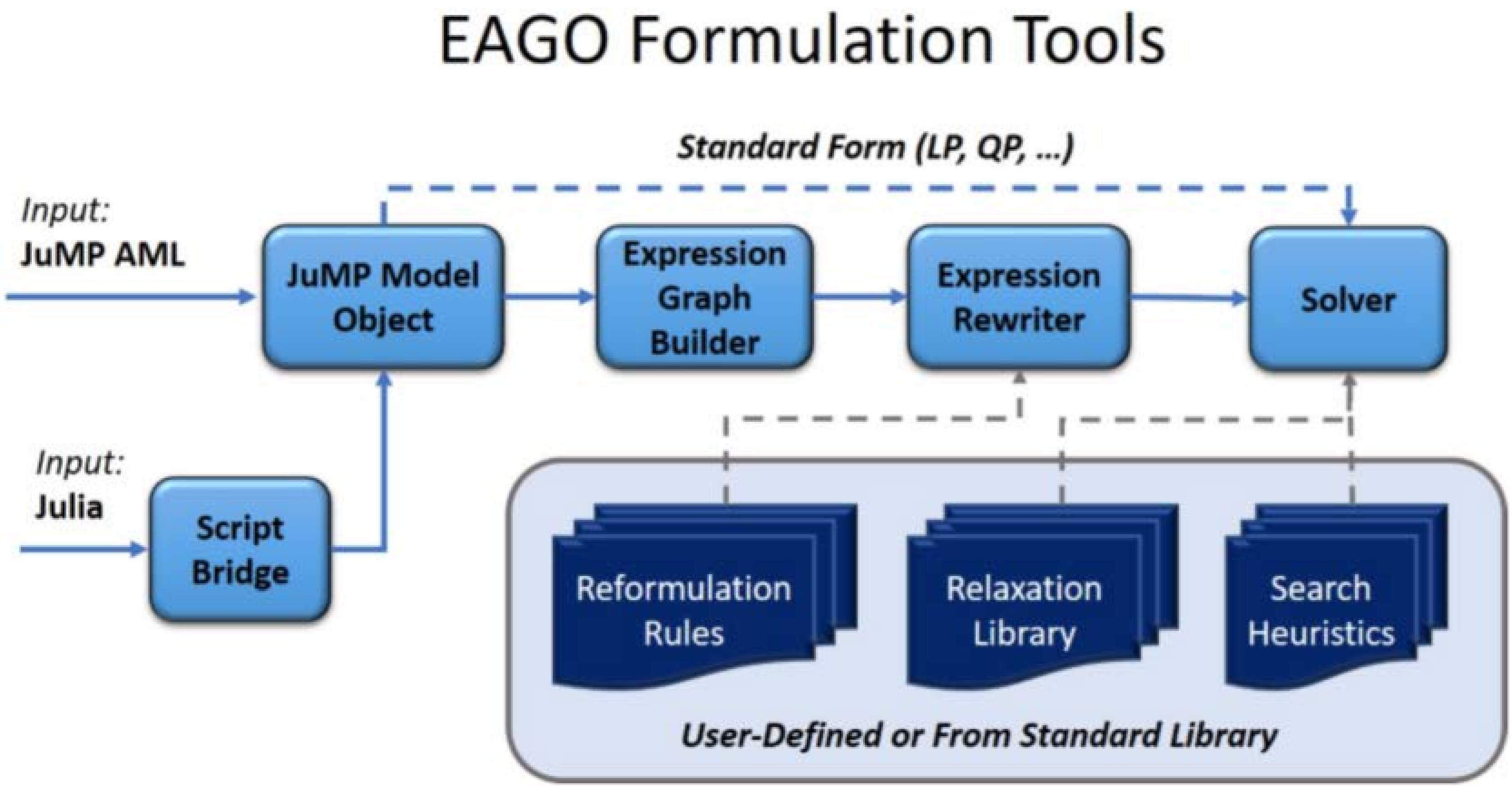
```
(v1.1) pkg> add EAGO
```

```
julia> using Pkg

julia> Pkg.add("EAGO")
```

From GitHub:

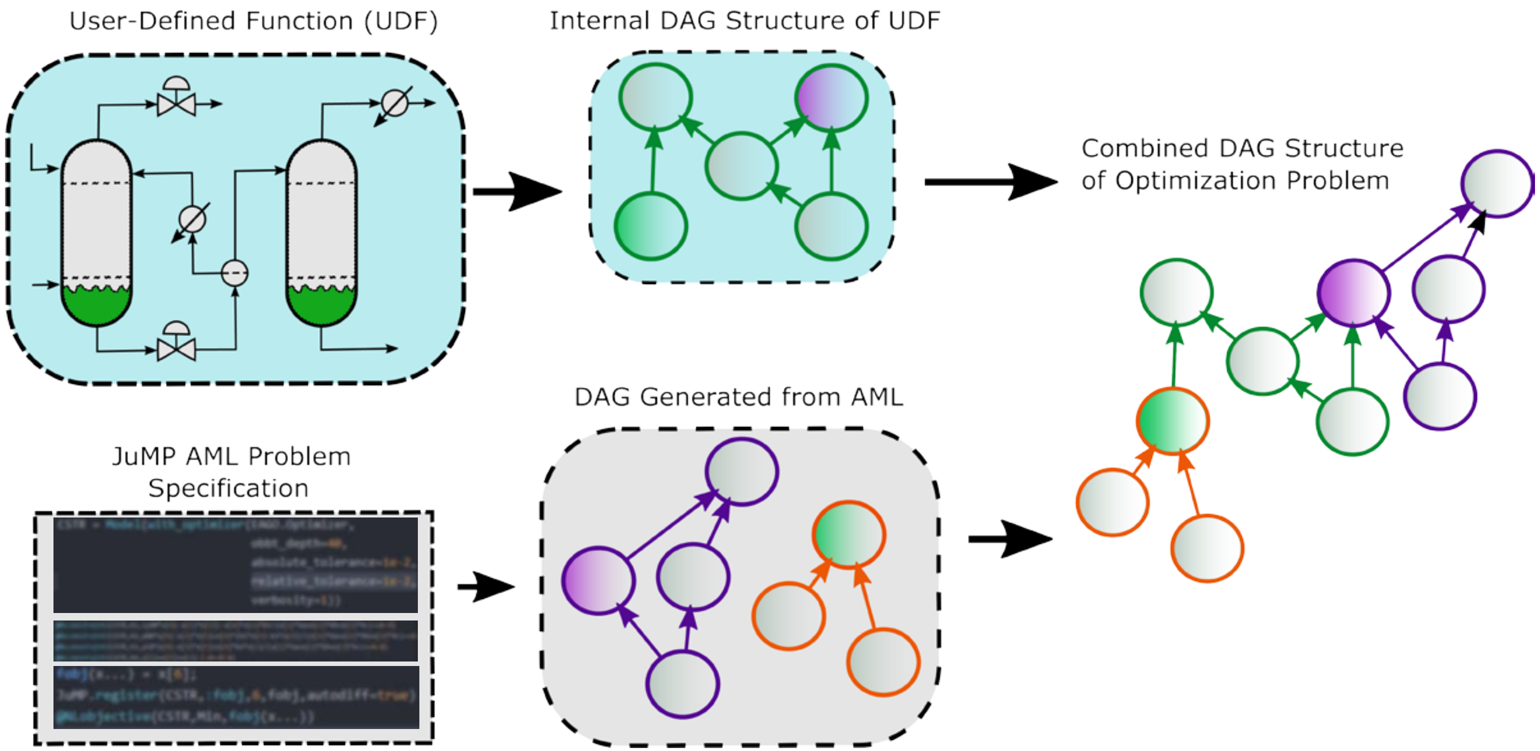
https://www.github.com/PSORLab/EAGO.jl

EAGO.jl: Architecture and Features



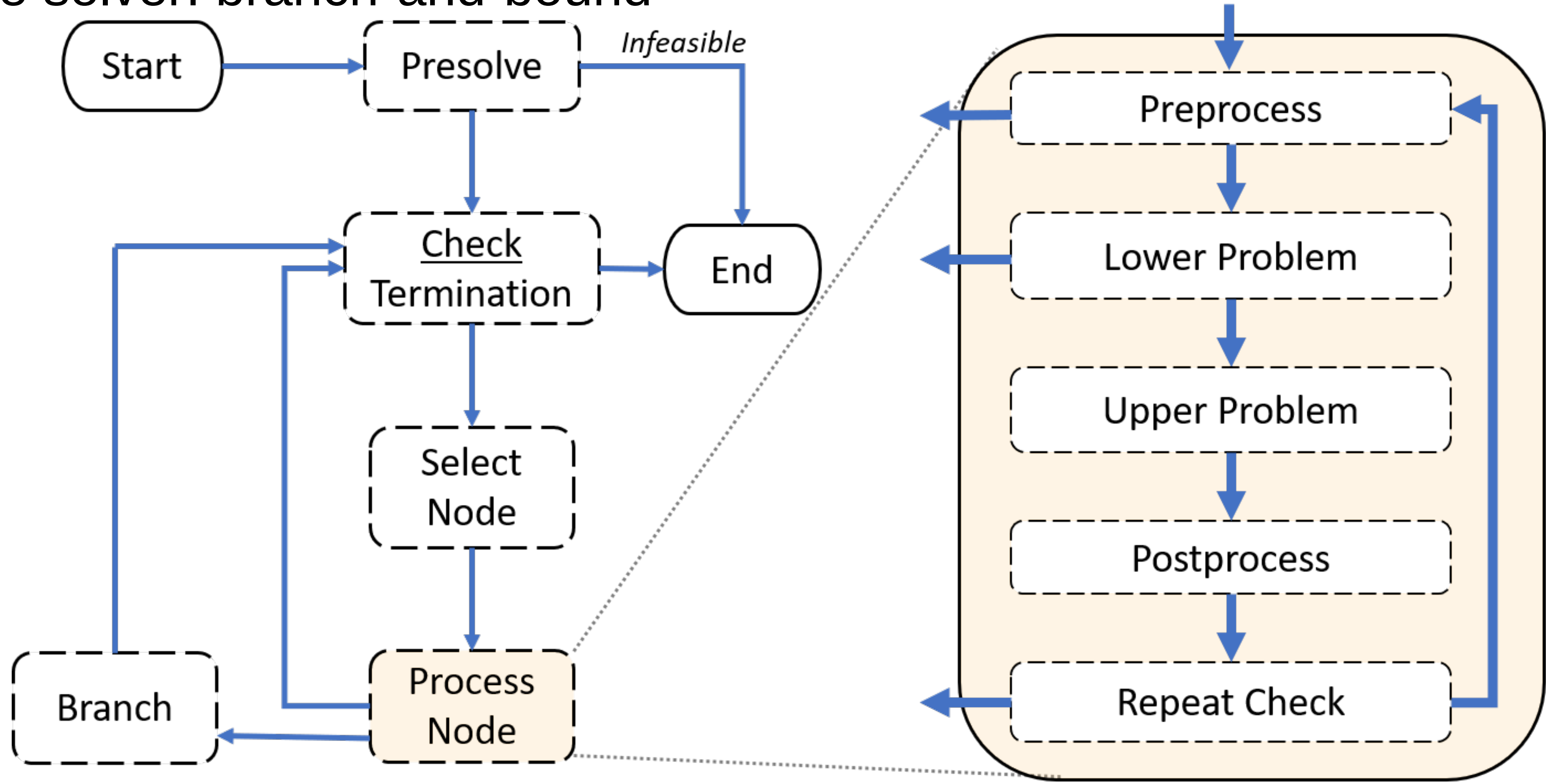
EAGO.jl: Advanced Formulations

- User-defined functions



EAGO.jl: Architecture and Features

- Core solver: branch-and-bound

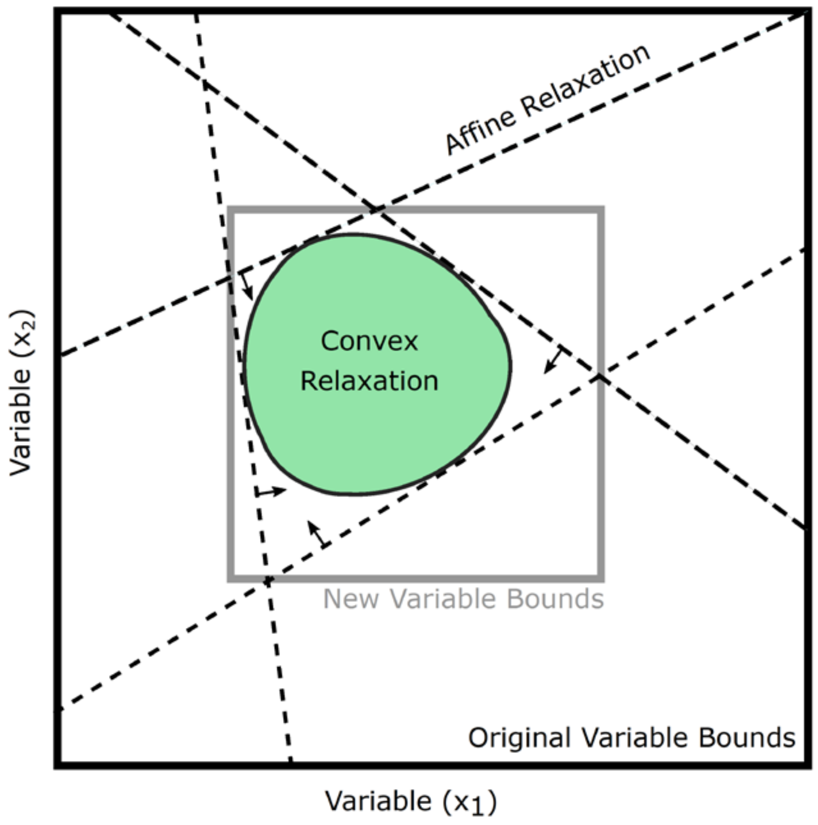


EAGO.jl: Architecture and Features

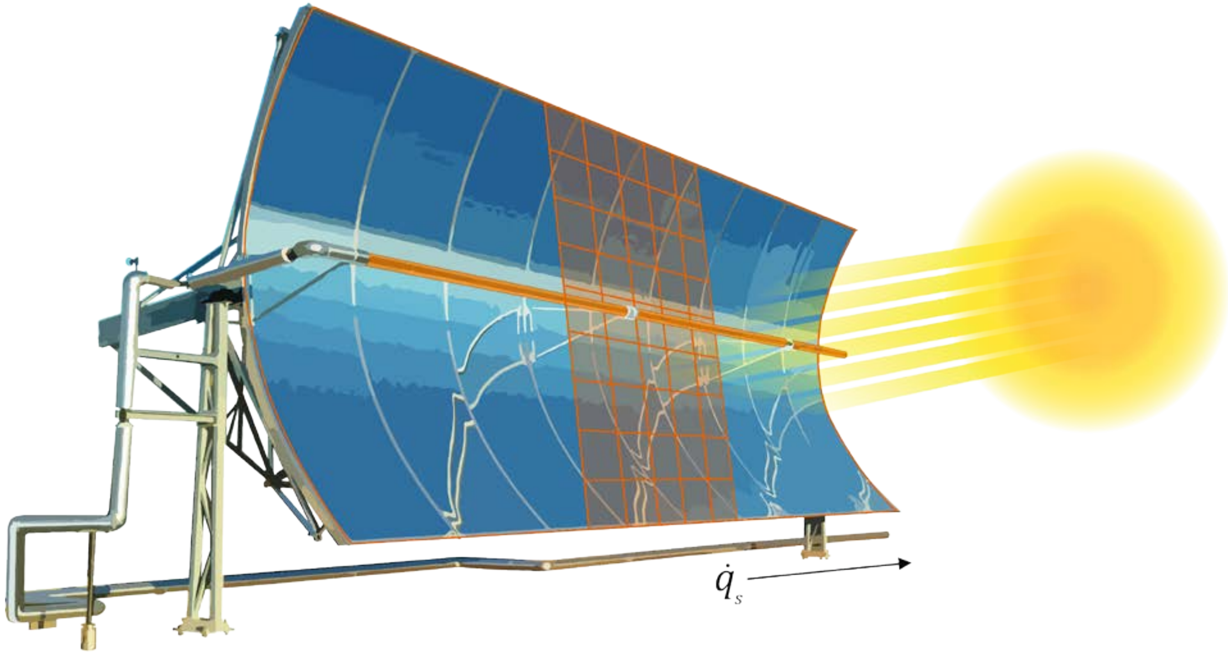
- Bounds and Relaxations
  - Interval arithmetic
  - McCormick-based relaxations
    - Multivariate, generalized, and differentiable
    - Implicit functions
  - αBB & Auxiliary variables coming soon to latest version

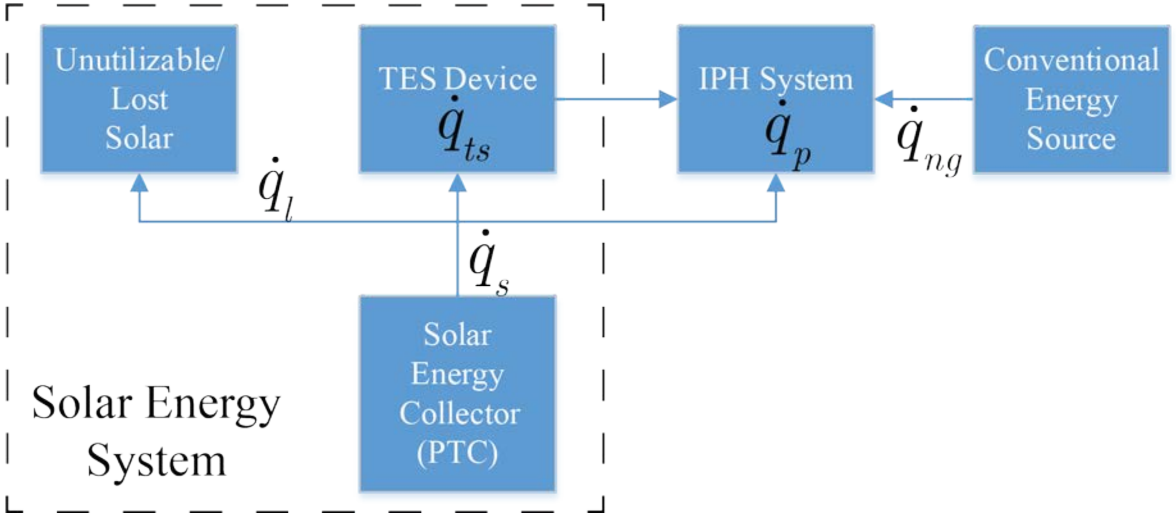
EAGO.jl: Architecture and Features

- 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



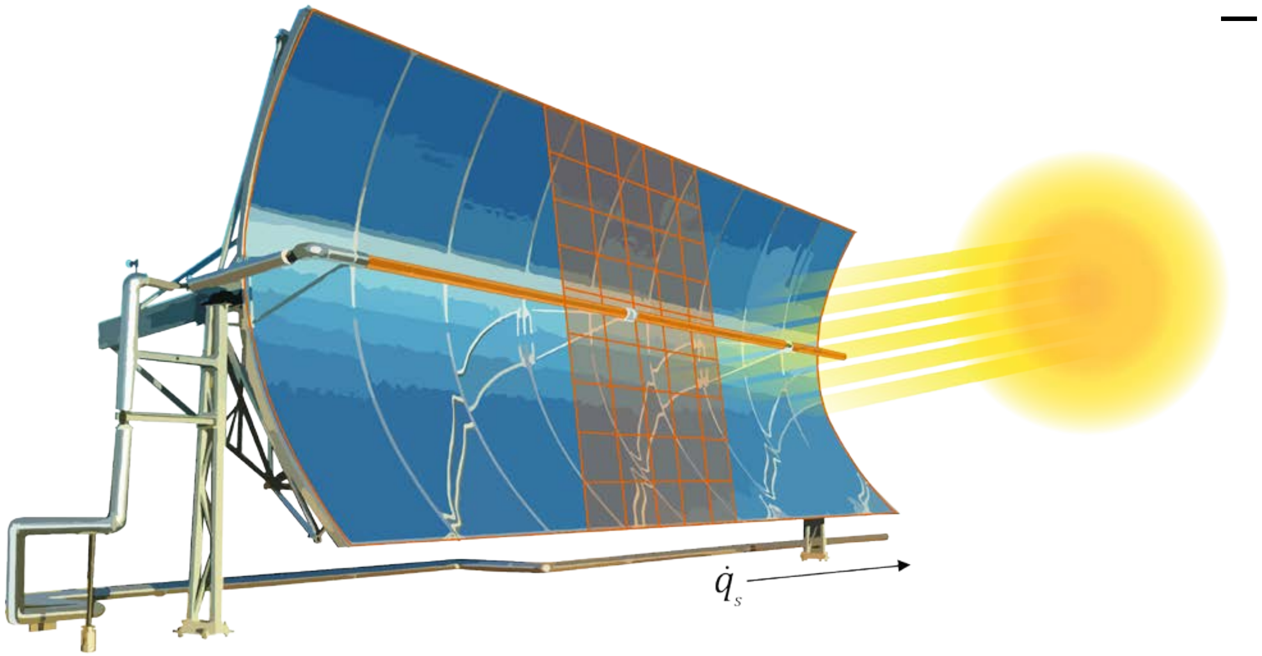
EAGO.jl: Ex. CST Hybridization





M.D. Stuber. A differentiable model for optimizing hybridization of industrial process heat systems with concentrating solar thermal power. *Processes*. 6(7), 76 (2018)

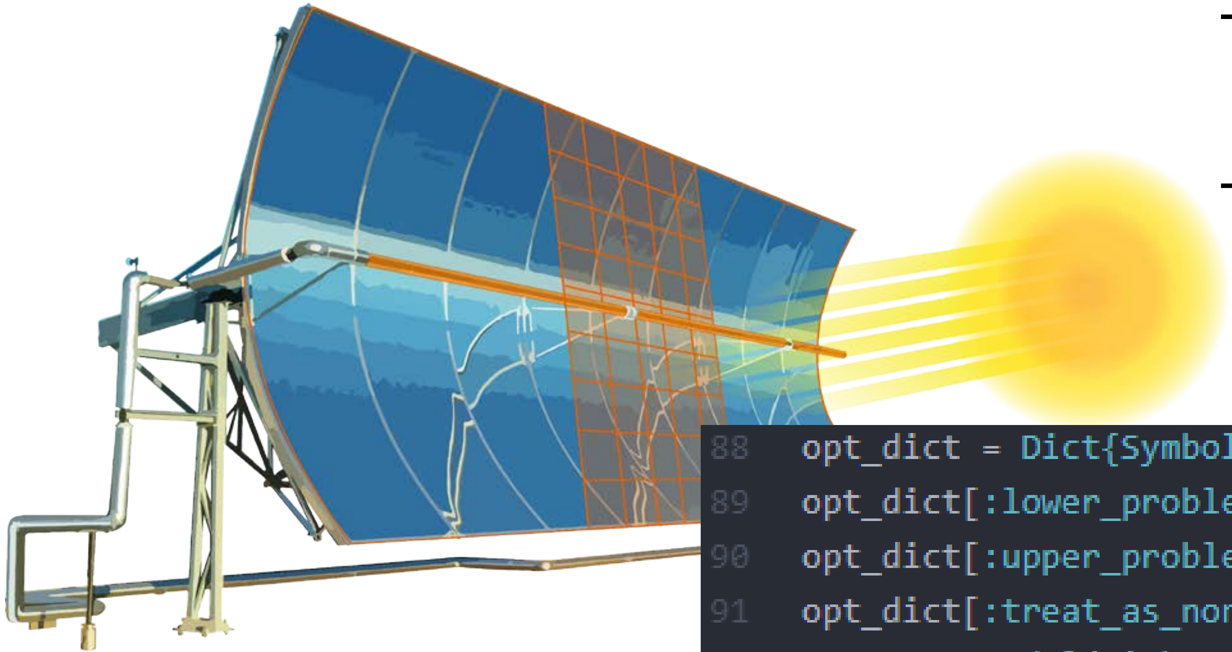
EAGO.jl: Ex. CST Hybridization



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

M.D. Stuber. A differentiable model for optimizing hybridization of industrial process heat systems with concentrating solar thermal power. *Processes*. 6(7), 76 (2018)

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

```
opt_dict = Dict{Symbol, Any}()
opt_dict[:lower_problem!] = LowerProblem!
opt_dict[:upper_problem!] = UpperProblem!
opt_dict[:treat_as_nonlinear] = [1; 2]
m = JuMP.Model(with_optimizer(EAGO.Optimizer,relative_tolerance=1e-2; opt_dict...))
```

M.D. Stuber. A differentiable model for optimizing hybridization of industrial process heat systems with concentrating solar thermal power. *Processes*. 6(7), 76 (2018)

EAGO.jl: Ex. Parameter Estimation

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.

$$\min_{\mathbf{p}\in\Pi} \sum_{i,j} \left(c_p^{\text{mod}}(T_i, x_j, \mathbf{p}) - c_p^{\text{exp}}(T_i, x_j)\right)^2$$

$$\text{s.t.} \;\; c_p^{\text{mod}}(T_i, x_j, \mathbf{p}) = -T_i \left.\frac{\partial^2 G(T_i, x_j, \mathbf{p})}{\partial T^2}\right|_P, \;\; \forall (i,j)$$

EAGO.jl: Ex. Parameter Estimation

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.

```
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+p[3]*log(T))+
                       (1-x1)*x1^2*(p[1]*T+p[2]*T^2+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)
```

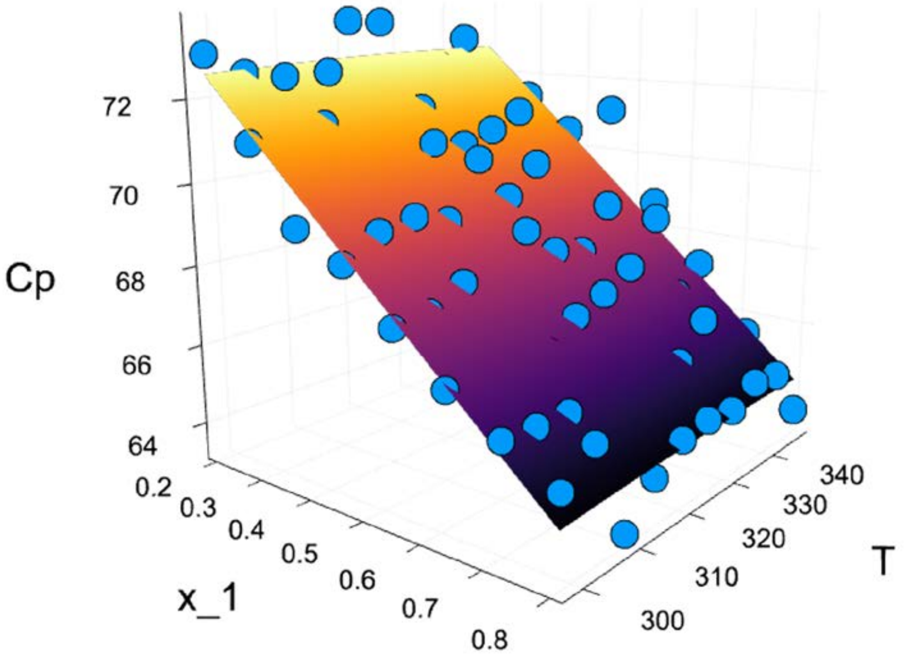
```
function objective(T::Vector,x1::Vector,Cp_exp::Matrix,p...)
    SSE = 0.0
    for i = 1:length(T)
        for j = 1:length(x1)
            SSE += (Cp(T[i],x1[j],p)-Cp_exp[i,j])^2
        end
    end
    return SSE
end
```

```
fobj(p...) = objective(Tdata,x1data,Cp_exp,p...)
```

EAGO.jl: Ex. Parameter Estimation

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]
                       (1-x1)*x1^2*(p[1]*T+p[2]
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-x
Cp(T,x1,p) = -T*ForwardDiff.derivative(T->Forwa
```




```julia
tive(T::Vector,x1::Vector,Cp_exp::Matrix,p...)

length(T)
= 1:length(x1)
E += (Cp(T[i],x1[j],p)-Cp_exp[i,j])^2
```

```julia
...)
```

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

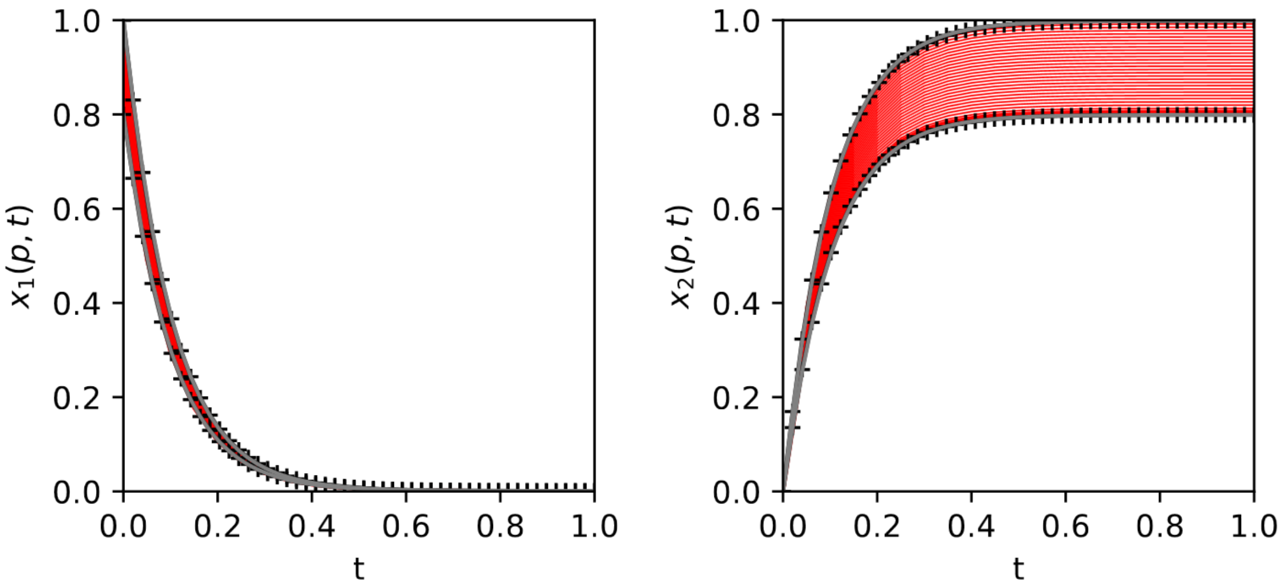
$$\min_{p\in P} \sum_{i=1}^{n}\sum_{j=1}^{2}\left(x_j(p,t_i) - d_j(t_i)\right)^2$$

$$\text{s.t. } \frac{d\mathbf{x}}{dt}(p,t) = \begin{pmatrix} k_2 x_2 - k_1 x_1 \\ k_1 x_1 - k_2 x_2 \end{pmatrix}, t \in [0,1]$$

$$\mathbf{x}(p,0) = (p,0)$$

$$P = [0.8, 1]$$

Relaxation Bounds for the ODE System



Wilhelm, Le, and Stuber (2019) *Under Review*

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

$$\min_{\mathbf{x}} f(\mathbf{x}) = \frac{x_1^2}{3} + x_2^2 + \frac{x_1}{2}$$

$$\text{s.t.} \quad (1 - x_1^2 y^2)^2 - x_1 y^2 - x_2^2 + x_2 \leq 0, \ \ \forall y \in [0,1]$$

$$\mathbf{x} \in [-1000, 1000]^2$$

EAGO solves in ~2.5 seconds

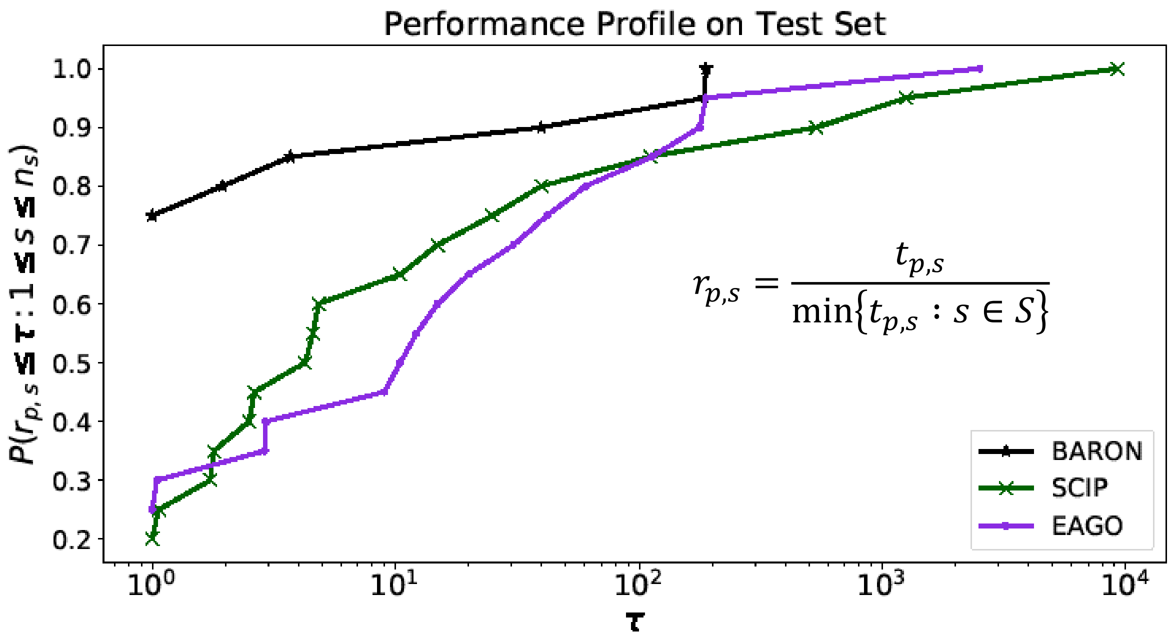
```
using JuMP, EAGO
# Defines objective and semi-infinite constraint
f(x) = (x[1]^2)/3.0 + x[2]^2 + x[1]/2.0
gSIP(x,p) = (1.0 - (x[1]^2)*(p[1]^2))^2 - x[1]*p[1]^2 - x[2]^2 + x[2]

# 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)

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

Table 2 Descriptive Statistics for Problems Selected for Benchmarking



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

UCONN
UNIVERSITY OF CONNECTICUT

Process Systems and Operations Research Laboratory