Recent Advances in EAGO.jl: Easy Advanced Global Optimization in Julia

Matthew D. Stuber, Assistant Professor,
stuber@alum.mit.edu
Matthew E. Wilhelm, PhD Candidate

Process Systems and Operations Research Laboratory
EURO Athens 2021 Greece
Key Contributor

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

EAGO.jl developer
Outline

• Motivation
  – Reduced-space deterministic global optimization
• EAGO.jl: Deterministic global optimization in Julia
  – Core features
  – Main features for advanced formulations
  – New and near-future additions
• Conclusions
Motivation: Reduced-Space Optimization

Want to solve dynamic optimization problems to guaranteed global optimality:

\[
\phi^* = \min_{p \in P \subset \mathbb{R}^{n_p}} \phi(x(p, t_f), p) \\
\text{s.t. } \dot{x}(p, t) = f(x(p, t), p, t), \ \forall t \in I = [t_0, t_f] \\
x(p, t_0) = x_0(p) \\
g(x(p, t_f), p) \leq 0
\]
Motivation: Reduced-Space Optimization

Want to solve dynamic optimization problems to guaranteed global optimality:

\[ \phi^* = \min_{p \in P \subset \mathbb{R}^n_p} \phi(x(p, t_f), p) \]

s.t. \[ \dot{x}(p, t) = f(x(p, t), p, t), \forall t \in I = [t_0, t_f] \]

\[ x(p, t_0) = x_0(p) \]

\[ g(x(p, t_f), p) \leq 0 \]

Parametric ordinary differential equation initial value problem (ODE-IVP) constraints.

Arise from optimal control, parameter estimation, etc.
Motivation: Reduced-Space Optimization

Want to solve dynamic optimization problems to guaranteed global optimality:

\[ \begin{align*}
\dot{x}(p,t) &= f(x(p,t), p, t), \quad \forall t \in I = [t_0, t_f] \\
x(p, t_0) &= x_0(p) \\
\end{align*} \]

\[ z_0 = x_0(p) \]
\[ \hat{z}_1 - z_0 - hf(\hat{z}_1, p, t_1) = 0 \]
\[ \ldots \]
\[ \hat{z}_K - \hat{z}_{K-1} - hf(\hat{z}_K, p, t_K) = 0 \]

Discrete-time reformulation (implicit Euler)

Paradigm for dynamic optimization problems with terminal and intermediate (path) constraints.

Arise from optimal control, parameter estimation, etc.
Motivation: Reduced-Space Optimization

Want to solve dynamic optimization problems to guaranteed global optimality:

\[
\phi^* = \min_{p \in P \subseteq \mathbb{R}^n_p} \phi(x(p, t_f), p)
\]

s.t. \( \dot{x}(p, t) = f(x(p, t), p, t), \forall t \in I = [t_0, t_f] \)

\( x(p, t_0) = x_0(p) \)

\( g(x(p, t_f), p) \leq 0 \)

Dimensionality: \( n_p \)

\[
\phi^* = \min_{p \in P, \hat{z} \in Z} \phi(\hat{z}, p, t_f)
\]

s.t. \( z_0 = x_0(u, p) \)

\( \hat{z}_1 - z_0 - hf(\hat{z}_1, p, t_1) = 0 \)

\( \vdots \)

\( \hat{z}_K - \hat{z}_{K-1} - hf(\hat{z}_K, p, t_K) = 0 \)

\( g(\hat{z}_K, p) \leq 0 \)

Dimensionality: \( n_p \times K \)
Motivation: Reduced-Space Optimization

Want to solve dynamic optimization problems to guaranteed global optimality:

\[
\phi^* = \min_{p \in P} \phi(x(p,t_f), p) \\
\text{s.t. } \dot{x}(p,t) = f(x(p,t), p, t), \forall t \in I = [t_0, t_f] \\
x(p,t_0) = x_0(p) \\
g(x(p,t_f), p) \leq 0
\]

Dimensionality: \( n_p \)

\[
\phi^* = \min_{p \in P} \phi(z(p), p, t_f) \\
\text{s.t. } g(z_K(p), p) \leq 0
\]

Dimensionality: \( n_p \times K \)

\[
\phi^* = \min_{p \in P, z \in Z} \phi(\hat{z}, p, t_f) \\
\text{s.t. } z_0 = x_0(u, p) \\
\hat{z}_1 - z_0 - hf(\hat{z}_1, p, t_1) = 0 \\
\vdots \\
\hat{z}_K - \hat{z}_{K-1} - hf(\hat{z}_K, p, t_K) = 0 \\
g(\hat{z}_K, p) \leq 0
\]

\[
z(p) = (z_0(p), z_1(p), \ldots, z_K(p))
\]
Background: EAGO

How do you get EAGO?

From Julia package manager:

```
(@v1.6) pkg> add EAGO
```

From GitHub:

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

How do you get EAGO?

From Julia package manager:

```
(@v1.6) pkg> add EAGO
```

From GitHub:

[https://www.github.com/PSORLab/EAGO.jl](https://www.github.com/PSORLab/EAGO.jl)

How do you use EAGO?

As a solver in the open-source algebraic modeling language JuMP.
As a stand-alone solver.
Published Results

- EAGO exhibits competitive performance on benchmarking set
EAGO.jl: Core Optimizer

Start -> Presolve

Presolve -> Check Termination

Check Termination -> Select Node

Select Node -> Process Node

Process Node -> Branch

Branch -> Presolve

Presolve -> Infeasible

Infeasible -> End

End -> Preprocess

Preprocess -> Lower Problem

Lower Problem -> Upper Problem

Upper Problem -> Postprocess

Postprocess -> Repeat Check

Repeat Check -> Preprocess
Key Improvements to Global Optimization Routine:

- Heuristics to ensure numerically safe affine relaxations for lower-bounding problems
- More computationally efficient approach to optimization-based bounds tightening
- No-overhead user-defined subroutines (lower-bounding problem, etc.)
- Improved parameter tuning

Other High-Level Improvements to EAGO’s Global Optimizer:

- Bridging + configuration for a large variety of subsolvers
- Preliminary support for integer-variables (MINLP problem forms)
- Detection of specialized problem forms (LP, MILP, convex)
- Support for additional semi-infinite programming (SIP) routines

Improvements to MINLP solution algorithm currently under development.
EAGO.jl: McCormick Relaxations

- EAGO generates relaxations of complicated nonlinear expressions using a McCormick relaxation methodology

- Improvements EAGO’s intrinsic library of relaxations which include:
  - Improved relaxations (tighter) of composite bilinear and trilinear terms*
  - Commonly encountered subexpressions: \( x \cdot \log(x), x \cdot \exp(x), \text{erf}(x) \), and more...
  - Envelopes for common activation functions**

\[
y = f(g(x), ..., h(x))
\]

Apply \(f\) composite relaxation rules

*In preparation  ** Under Review
Improved relaxation subroutine performance due to intrinsic relaxation library upgrade:

- Relaxation of implicit functions
- Relaxations of ODEs
- Reverse propagation of relaxations

Simple ODE Relaxation

\[
\frac{dx}{dt} = \exp(p) \sin(x)(2 - x),
\]

\(x(0) = 1, \quad p \in [0.01, 1], \quad t \in [0,5]\)

---

EAGO.jl: Main Features

- *Embedded Machine Learning (ML) Models*
- Semi-infinite Programming
- Dynamic Optimization
  - ... Composability thereof
EAGO.jl: Embedded ML-models

- Support for embedded general feedforward neural networks
  Multilayer Perceptron, Deep Residual Networks, etc.

- Incorporate many common Flux.jl trained models

Multilayer Perceptron

Envelopes of Relaxation Functions

EURO 2021 - July 13, 2021
EAGO.jl: Embedded ML-models

- Future support for specialized surrogate models
- Implicit function centered machine learning architectures
  
  i. Deep equilibrium networks
  ii. Implicit deep learning
      \[ \hat{y}(u) = Cx + Du \]
      \[ x = \phi(Ax + Bu) \]
  iii. Fixed-point equation-based networks

Enabled by core optimizer development. Expected by end of August 2021.

EAGO.jl: Main Features

- Embedded Machine Learning (ML) Models
- *Semi-infinite Programming*
- Dynamic Optimization

... Composability thereof
Solving Nonconvex SIPs

- EAGO.jl supports general nonconvex SIPs:
  \[ f^* = \min_{x \in X} f(x) \]
  \[ \text{s.t. } g(x, p) \leq 0, \forall (p) \in P \]

- Composable with ML/dynamic relaxations.

**New Features:**

- Added new hybrid-oracle SIP routine\[^5\].
- Automatic subproblem tolerance specification (SIPResRev SIPHybrid algorithms)\[^5\].
- User-extendable SIP subproblems.

---

EAGO.jl: Main Features

- Embedded Machine Learning (ML) Models
- Semi-infinite Programming
- Dynamic Optimization

... Composability thereof
Dynamics, Relaxation

- New support included for general nonlinear parametric ordinary differential equations.

- Incorporation into global optimizer:
  - Relaxations & Domain Reduction:
    - Interval bounds
    - Relaxations & (sub)gradients
  - Local NLP solver:
    - Automatic differentiation for upper-bounding problem

**Continuous-Time Relaxations**\([10,11,12]\)

\[
\begin{align*}
\dot{x}^c(t,p) &= f^c(t,p,x^c(t,p),x'^c(t,p)), \\
\dot{x}'^c(t,p) &= f'^c(t,p,x^c(t,p),x'^c(t,p)), \\
x^c(t_0,p) &= x_0^c(p)
\end{align*}
\]

**Discrete-Time Relaxations**\([13,14,15]\)

\[
x(\tau_{q+1},p) \in x(\tau_q,p) + \sum_{j=1}^{p} \frac{h^j}{j!} f^{(j)}(x(\tau_q,p),p) + \frac{h^{p+1}}{(p+1)!} f^{(p+1)}(X(\tau_q),P)
\]

Remainder Bound

---

Dynamics, Implementation

Core Algorithms

- `DynamicBoundspODEsDiscrete.jl`
  - Discrete time approaches

- `DynamicBoundspODEsIneq.jl`
  -- Continuous time approaches

Abstraction Layer

- `DynamicBounds.jl`

- `DynamicBoundsBase.jl`

Extendable Global Optimizer\(^{35}\)

- `EAGODynamicOptimizer.jl\(^{36}\)`

Future Work:
Integrate with JuMP-based frontends (e.g., InfiniteOpt.jl\(^{37}\))

---

EAGO.jl: Main Features

- Embedded Machine Learning (ML) Models
- Semi-infinite Programming
- Dynamic Programming

…Composability thereof
Dynamic SIP Formulation

\[
\Phi^* = \min_u \Phi(u)\\
\text{s.t. } g(x(u, p, t_f), u, p) \leq 0\\
\dot{x} = f(x(u, p, t), u, p)\\
x(u, p, t_0) = x_0(u, p)\\
t \in I = [t_0, t_f], \forall p \in P
\]

Objective

Performance Constraint(s)

Parametric ODEs

Initial Condition

- Design under worst-case realization of uncertainty.
- Safety-critical systems and high-impact defect elimination.

Robust Dynamic Optimization

Batch MMA Polymerization Reaction

Adequate cooling at maximum temperature to withstand sensor fault?

Robust Operation SIP

\[ \gamma^* = \max_{p \in P, \gamma \in \Gamma} \gamma \]

s.t. \( \gamma \leq T(t_f, u) - p, \forall u \in U \)

- Nonconvex semi-infinite program
- Embedded dynamic system
- Complex chemical kinetics (hybrid model desirable)
EAGO.jl: Distributed Computing

**Overall Framework:**

- Support for distributed computing via ClusterManager abstraction
- User-specified entry point for parallelism
  - Parallel evaluation of relaxations
  - Parameter setting in optimizers used by subproblem
  - Lower bounding, upper bounding subproblems

*Currently, under development. Expected by end of 2021.*
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

• Now and Near Future:
  – Exhaustive library of relaxation envelopes
  – Additional relaxations ($\alpha$BB 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
- EURO 2021 Organizers
- Funding: National Science Foundation

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

This material is based upon work supported by the National Science Foundation under Grant No. 1932723. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.
Motivation: Reduced-Space Optimization

\[
\begin{align*}
\min_{z, p} \quad & z - p \\
\text{s.t.} \quad & z^2 - p = 0 \\
& z \in [0, 2] \\
& p \in [0.01, 2]
\end{align*}
\]

\[
f(z, p) = \frac{z}{2} - p
\]

\[
h(z, p) = z^2 - p = 0
\]
Motivation: Reduced-Space Optimization

$$z^2 - p = 0$$

$$\Rightarrow x(p) = \sqrt{p}$$

$$\min_{p} \frac{x(p)}{2} - p$$

s.t. \( p \in [0.01, 2] \)

$$f(x(p), p) = \frac{\sqrt{p}}{2} - p$$
Robust Dynamic Optimization

Robust Operation SIP

\[ \gamma^* = \max_{p \in \gamma \in \Gamma} \gamma \]
\[ \text{s.t. } \gamma \leq T(t_f, u) - p, \ \forall u \in U \]

Dynamical System (Mass & Energy Balance)

\[ \frac{dC_m}{dt} = (1 + \epsilon C_m/C_{m0}) R_m, \]
\[ \frac{dC_i}{dt} = R_i + \epsilon C_i/C_{i0} R_m, \]
\[ \frac{dT}{dt} = \frac{\alpha_0 k P \xi_0 C_m}{1 + \epsilon C_m/C_{m0}} + \alpha_1 (T_f - T) \]

Rate constants \((R_m, R_i)\) from pseudo-empirical models

Rate Expression \((Greatly\ Simplified\ldots)\)

\[ R_m = -C_m \xi_0 (k_P + k_f), \]
\[ R_i = -k_i C_i, \]
\[ \xi_0^2 k_i (\xi_0, C_m, T) - 2 \xi k_i C_i = 0. \]

Relaxations of Dynamic System

Use 3-layer GeLU ANN as in place of solving nonlinear equation from quasi-steady state assumption

Able to solve SIP in 57.8 s using a modified SIPres algorithm.
EAGO.jl: Core Additions

New multi-graph representation:
- Introduce support for multiple-output subexpressions.
- Separate caches of information from graph structure.
- Preliminary implementations for graph-walking convexity detection.

Roadmap to Improved Composability of Nonlinear Functions (via JuMP extension):
- Introduce support for multiple-output subexpressions.
- Introduce auxiliary variables.
- Facilitates chaining blocks of general nonlinear expressions (such as implicitly defined functions).

Currently, under development. Expected by end of August 2021.

Graph representation:
- $h_1(y, x) = 0$
- $h_2(z, y) = 0$
- $f(z)$