Recent Advances in EAGO.jl and Its Use With JuMP.jl

Dimitri Alston, Ph.D. Student
Robert Gottlieb, Ph.D. Student
Matthew Stuber, P&W Associate Professor in Advanced Systems Engineering

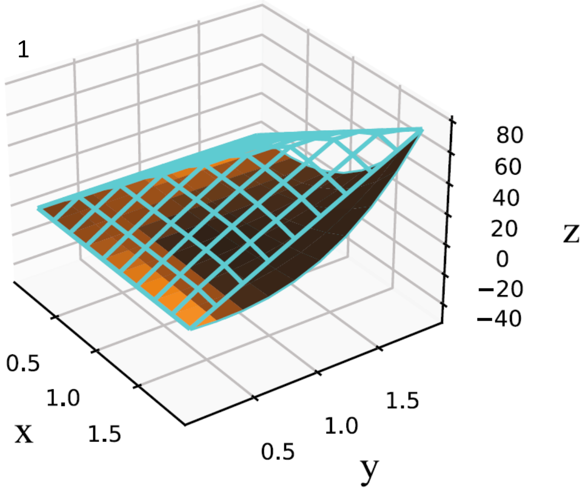
July 20th, 2024

UCONN
UNIVERSITY OF CONNECTICUT

Process Systems and Operations Research Laboratory

Easy Advanced Global Optimization

- Open-source deterministic global solver for nonconvex MINLPs
  - Semi-infinite programs (SIPs)
  - Dynamic optimization
  - User-defined functions
- Uses branch-and-bound (B&B) to guarantee global optimality or infeasibility
- Applies McCormick-based relaxations for convex lower-bounding problems
- Designed in conjunction with JuMP



[1] Wilhelm, M.E., and Stuber, M.D. Improved Convex and Concave Relaxations of Composite Bilinear Forms. *Journal of Optimization Theory and Applications*. 197, 174-204 (2023).

Parameter Estimation Example

$$\min_{\mathbf{p}} \phi(\mathbf{p},t) = \sum_{i=0}^{N} (I_i^{calc} - I_i^{exp})^2$$

$$\text{s.t. } \mathbf{p} \in [\mathbf{p}^L, \mathbf{p}^U]$$

$$I_i^{calc} = x_{A,i} + \frac{2}{21} x_{B,i} + \frac{2}{21} x_{D,i}$$

$$\frac{dx_A}{dt} = k_1 x_Z x_Y - c_{O_2}(k_{2f} + k_{3f})x_A + \frac{k_{2f}}{K_2} x_D + \frac{k_{3f}}{K_3} x_B - k_5 x_A^2$$

$$\frac{dx_B}{dt} = c_{O_2} k_{3f} x_A - \left(\frac{k_{3f}}{K_3} + k_4\right) x_B$$

$$\frac{dx_D}{dt} = c_{O_2} k_{2f} x_A - \frac{k_{2f}}{K_2} x_D$$

$$\frac{dx_Y}{dt} = -k_{1s} x_Z x_Y$$

$$\frac{dx_Z}{dt} = -k_1 x_Z x_Y$$

[2] Mitsos, A., Chachuat, B., and Barton, P.I. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, *SIAM*. 20(2), 573-601 (2009).

Parameter Estimation Example

$$\frac{dx_A}{dt} = k_1 x_Z x_Y - c_{O_2}(k_{2f} + k_{3f})x_A + \frac{k_{2f}}{K_2}x_D + \frac{k_{3f}}{K_3}x_B - k_5 x_A^2$$

$$\frac{dx_B}{dt} = c_{O_2} k_{3f} x_A - \left(\frac{k_{3f}}{K_3} + k_4\right)x_B$$

$$\frac{dx_D}{dt} = c_{O_2} k_{2f} x_A - \frac{k_{2f}}{K_2}x_D$$

$$\frac{dx_Y}{dt} = -k_{1s} x_Z x_Y$$

$$\frac{dx_Z}{dt} = -k_1 x_Z x_Y$$

[2] Mitsos, A., Chachuat, B., and Barton, P.I. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, *SIAM*. 20(2), 573-601 (2009).

Parameter Estimation Example

$$x_A^{i+1} = x_A^i + \Delta t\left(k_1 x_Z^i x_Y^i - c_{O_2}(k_{2f} + k_{3f})x_A^i + \frac{k_{2f}}{K_2}x_D^i + \frac{k_{3f}}{K_3}x_B^i - k_5 (x_A^i)^2 \right)$$

$$x_B^{i+1} = x_B^i + \Delta t\left(c_{O_2} k_{3f} x_A^i - \left(\frac{k_{3f}}{K_3} + k_4 \right) x_B^i \right)$$

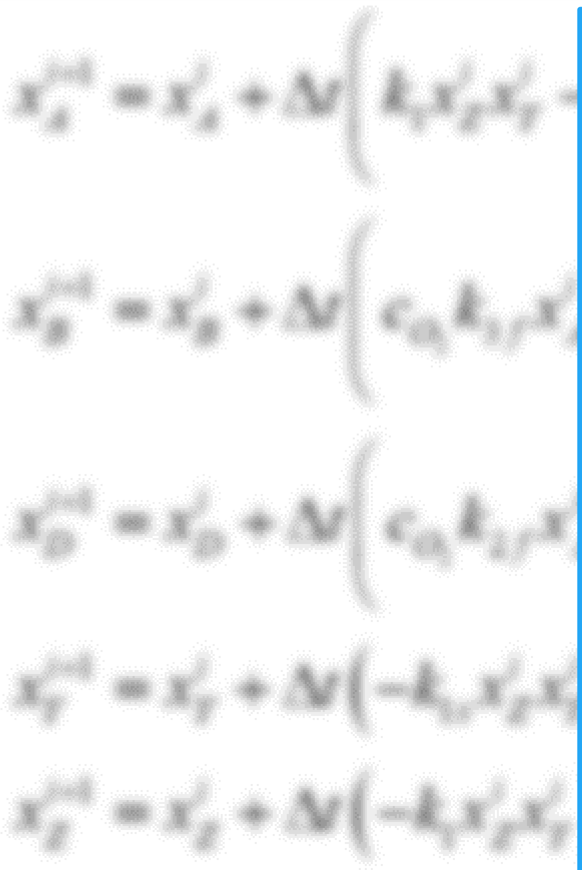
$$x_D^{i+1} = x_D^i + \Delta t\left(c_{O_2} k_{2f} x_A^i - \frac{k_{2f}}{K_2} x_D^i \right)$$

$$x_Y^{i+1} = x_Y^i + \Delta t\left(-k_{1s} x_Z^i x_Y^i \right)$$

$$x_Z^{i+1} = x_Z^i + \Delta t\left(-k_1 x_Z^i x_Y^i \right)$$

[2] Mitsos, A., Chachuat, B., and Barton, P.I. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, *SIAM*. 20(2), 573-601 (2009).

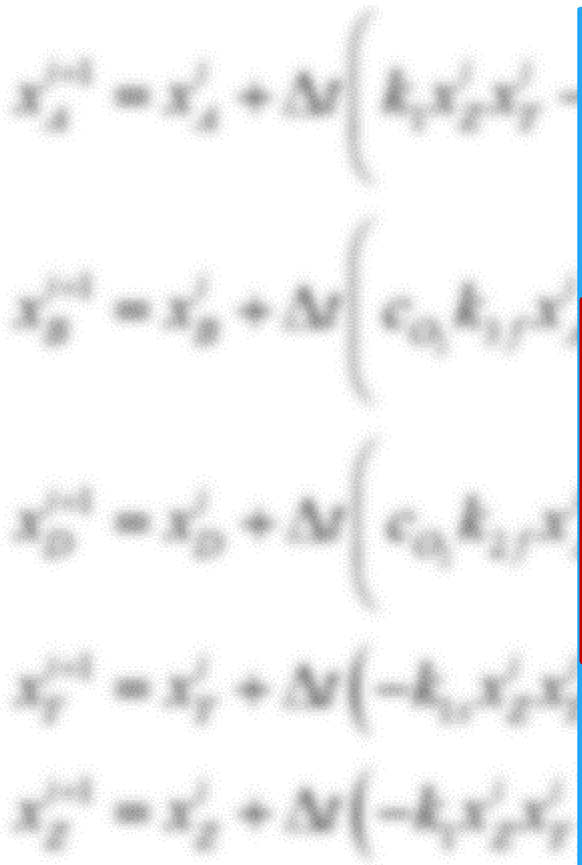
Parameter Estimation Example



```julia
using CSV, DataFrames, JuMP, EAGO, Gurobi

data = CSV.read("kinetic_intensity_data.csv", DataFrame)
bounds = CSV.read("implicit_variable_bounds.csv", DataFrame)

pL = [10.0, 10.0, 0.001]
pU = [1200.0, 1200.0, 40.0];

intensity(xA, xB, xD) = xA + (2/21)*xB + (2/21)*xD

function objective(p...)
    x = explicit_euler_integration(p)
    SSE = 0.0
    for i=1:200
        SSE += (intensity(x[5i-4],x[5i-3],x[5i-2]) - data[!, :intensity][i])^2
    end
    return SSE
end

factory = () -> EAGO.Optimizer(SubSolvers(; r = Gurobi.Optimizer()))
model = JuMP.Model(factory)
@variable(model, pL[i] <= p[i=1:3] <= pU[i])
fobj(p...) = objective(p...)
JuMP.register(model, :fobj, 3, fobj, autodiff=true)
@NLobjective(model, Min, fobj(p...))
JuMP.optimize!(model)
```

[2] Mitsos, A., Chachuat, B., and Barton, P.I. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, *SIAM*. 20(2), 573-601 (2009).

Parameter Estimation Example



```julia
using CSV, DataFrames, JuMP, EAGO, Gurobi

data = CSV.read("kinetic_intensity_data.csv", DataFrame)
bounds = CSV.read("implicit_variable_bounds.csv", DataFrame)

pL = [10.0, 10.0, 0.001]
pU = [1200.0, 1200.0, 40.0];

intensity(xA, xB, xD) = xA + (2/21)*xB + (2/21)*xD

function objective(p...)
    x = explicit_euler_integration(p)
    SSE = 0.0
    for i=1:200
        SSE += (intensity(x[5i-4],x[5i-3],x[5i-2]) - data[!, :intensity][i])^2
    end
    return SSE
end

factory = () -> EAGO.Optimizer(SubSolvers(; r = Gurobi.Optimizer()))
model = JuMP.Model(factory)
@variable(model, pL[i] <= p[i=1:3] <= pU[i])
fobj(p...) = objective(p...)
JuMP.register(model, :fobj, 3, fobj, autodiff=true)
@NLobjective(model, Min, fobj(p...))
JuMP.optimize!(model)
```

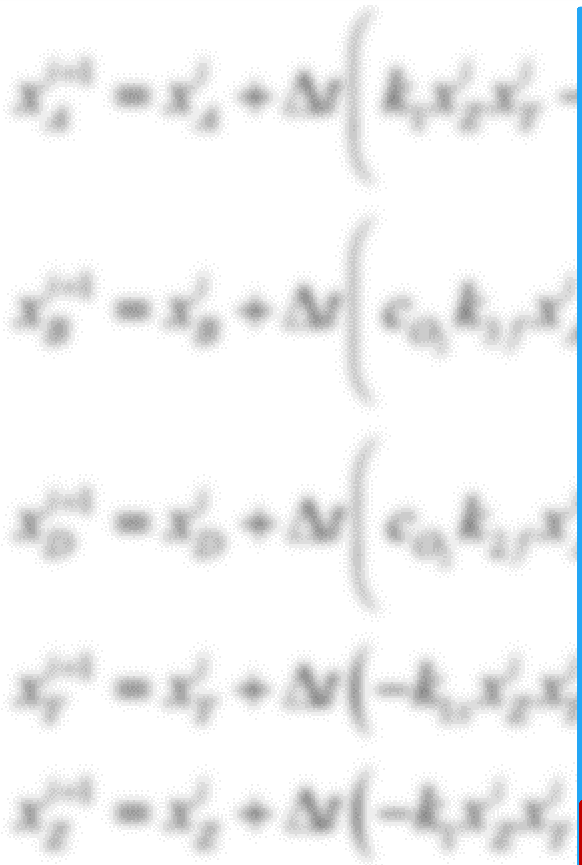
[2] Mitsos, A., Chachuat, B., and Barton, P.I. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, *SIAM*. 20(2), 573-601 (2009).

Parameter Estimation Example

```julia
using CSV, DataFrames, JuMP, EAGO, Gurobi

data = CSV.read("kinetic_intensity_data.csv", DataFrame)
bounds = CSV.read("implicit_variable_bounds.csv", DataFrame)

pL = [10.0, 10.0, 0.001]
pU = [1200.0, 1200.0, 40.0];

intensity(xA, xB, xD) = xA + (2/21)*xB + (2/21)*xD

function objective(p...)
    x = explicit_euler_integration(p)
    SSE = 0.0
    for i=1:200
        SSE += (intensity(x[5i-4],x[5i-3],x[5i-2]) - data[!, :intensity][i])^2
    end
    return SSE
end

factory = () -> EAGO.Optimizer(SubSolvers(; r = Gurobi.Optimizer()))
model = JuMP.Model(factory)
@variable(model, pL[i] <= p[i=1:3] <= pU[i])
fobj(p...) = objective(p...)
JuMP.register(model, :fobj, 3, fobj, autodiff=true)
@NLobjective(model, Min, fobj(p...))
JuMP.optimize!(model)
```

[2] Mitsos, A., Chachuat, B., and Barton, P.I. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, *SIAM*. 20(2), 573-601 (2009).

Parameter Estimation Example

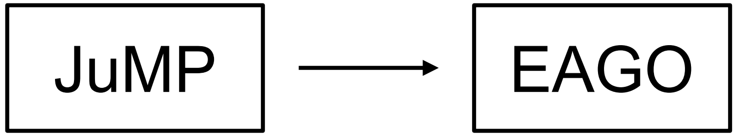


```julia
using CSV, DataFrames, JuMP, EAGO, Gurobi

data = CSV.read("kinetic_intensity_data.csv", DataFrame)
bounds = CSV.read("implicit_variable_bounds.csv", DataFrame)

pL = [10.0, 10.0, 0.001]
pU = [1200.0, 1200.0, 40.0];

intensity(xA, xB, xD) = xA + (2/21)*xB + (2/21)*xD

function objective(p...)
    x = explicit_euler_integration(p)
    SSE = 0.0
    for i=1:200
        SSE += (intensity(x[5i-4],x[5i-3],x[5i-2]) - data[!, :intensity][i])^2
    end
    return SSE
end

factory = () -> EAGO.Optimizer(SubSolvers(; r = Gurobi.Optimizer()))
model = JuMP.Model(factory)
@variable(model, pL[i] <= p[i=1:3] <= pU[i])
fobj(p...) = objective(p...)
JuMP.register(model, :fobj, 3, fobj, autodiff=true)
@NLobjective(model, Min, fobj(p...))
JuMP.optimize!(model)
```

[2] Mitsos, A., Chachuat, B., and Barton, P.I. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, *SIAM*. 20(2), 573-601 (2009).

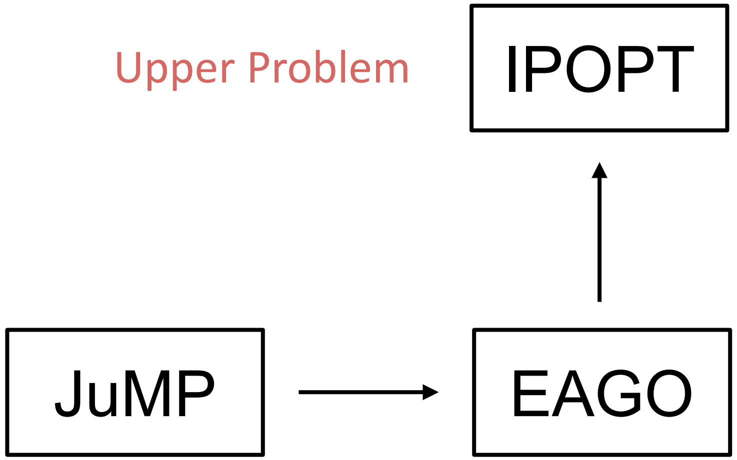
Integration with JuMP

- EAGO parses a JuMP model and sends information to bounding subroutines



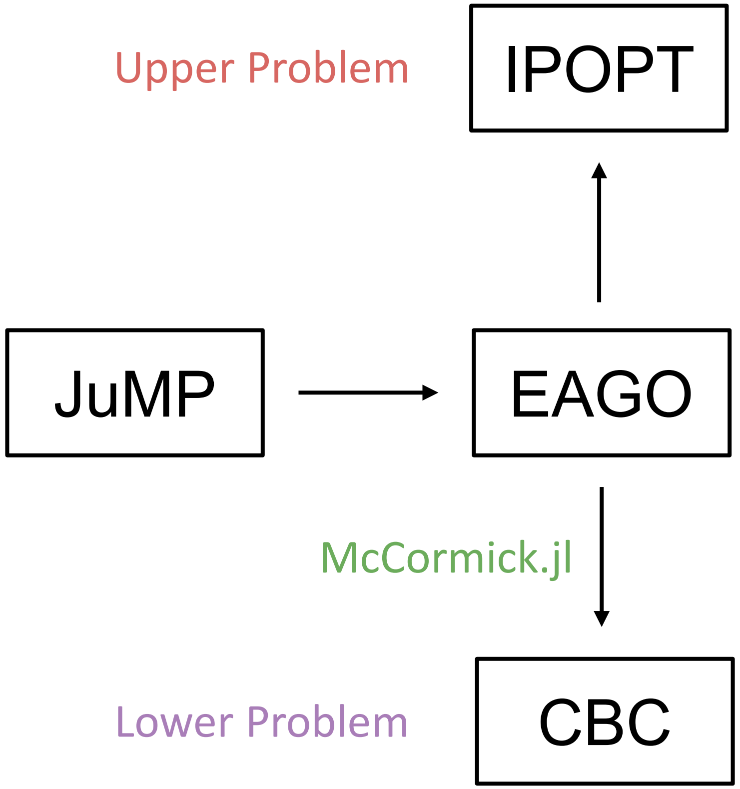
Integration with JuMP

- EAGO parses a JuMP model and sends information to bounding subroutines
  - Original problem is sent to IPOPT to generate an upper bound



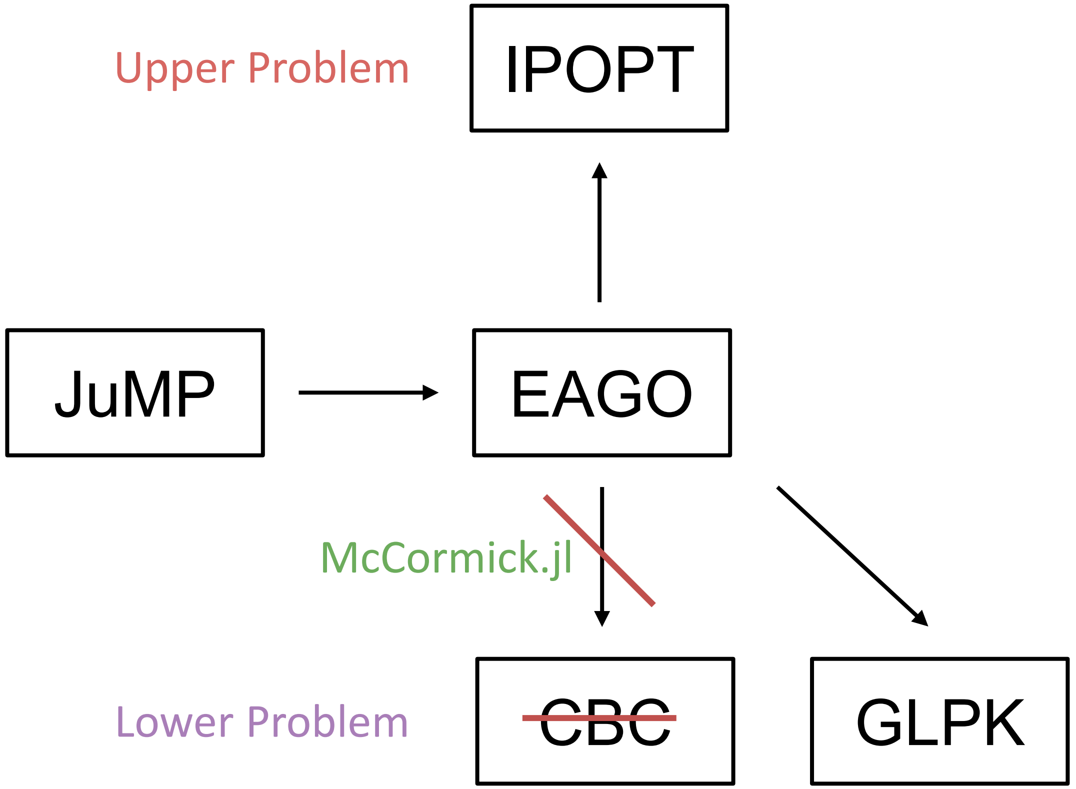
Integration with JuMP

- EAGO parses a JuMP model and sends information to bounding subroutines
  - Original problem is sent to IPOPT to generate an upper bound
  - Relaxed problem is sent to CBC to generate a lower bound



Integration with JuMP

- EAGO parses a JuMP model and sends information to bounding subroutines
  - Original problem is sent to IPOPT to generate an upper bound
  - Relaxed problem is sent to CBC to generate a lower bound
  - Subsolvers can be any appropriate MOI.AbstractOptimizer




```julia
using JuMP, EAGO, GLPK

factory = () -> EAGO.Optimizer(SubSolvers(; r = GLPK.Optimizer()))
model = JuMP.Model(optimizer_with_attributes(factory))
```

Timeline

Apr 2018 – Initial EAGO Release

Jun 2018 – JuMP-dev 2018

[3]


[3] https://jump.dev/meetings/bordeaux2018/

Timeline

Apr 2018 – Initial EAGO Release

Jun 2018 – JuMP-dev 2018

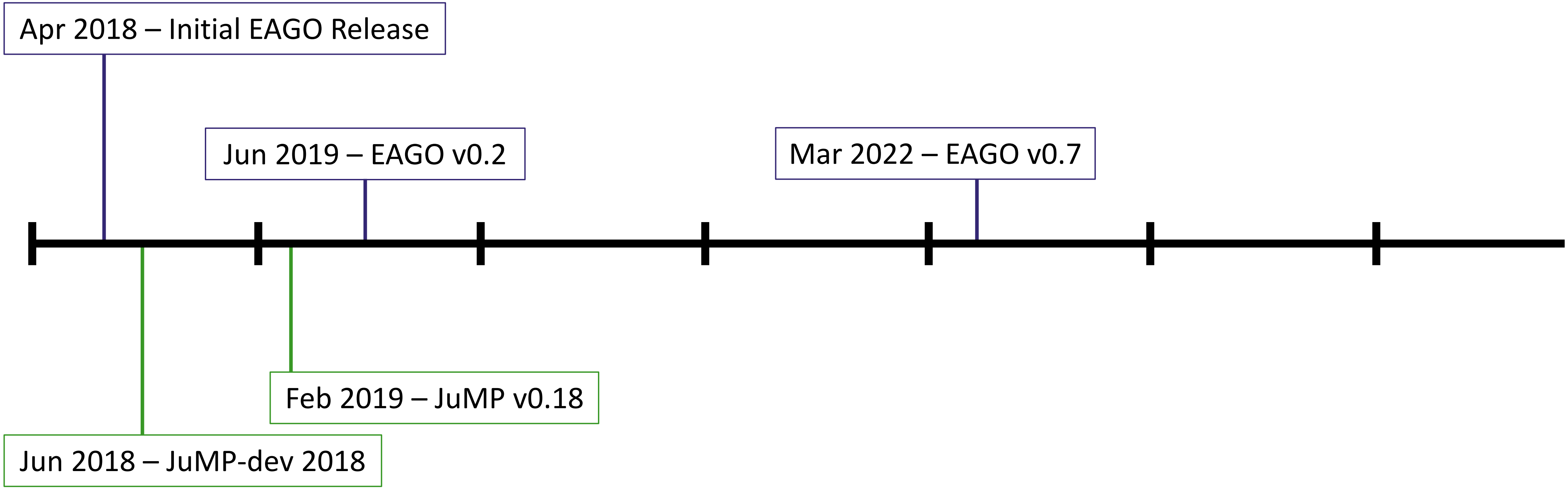
Feb 2019 – JuMP v0.18

Jun 2019 – EAGO v0.2

[3]

[3] https://jump.dev/meetings/bordeaux2018/

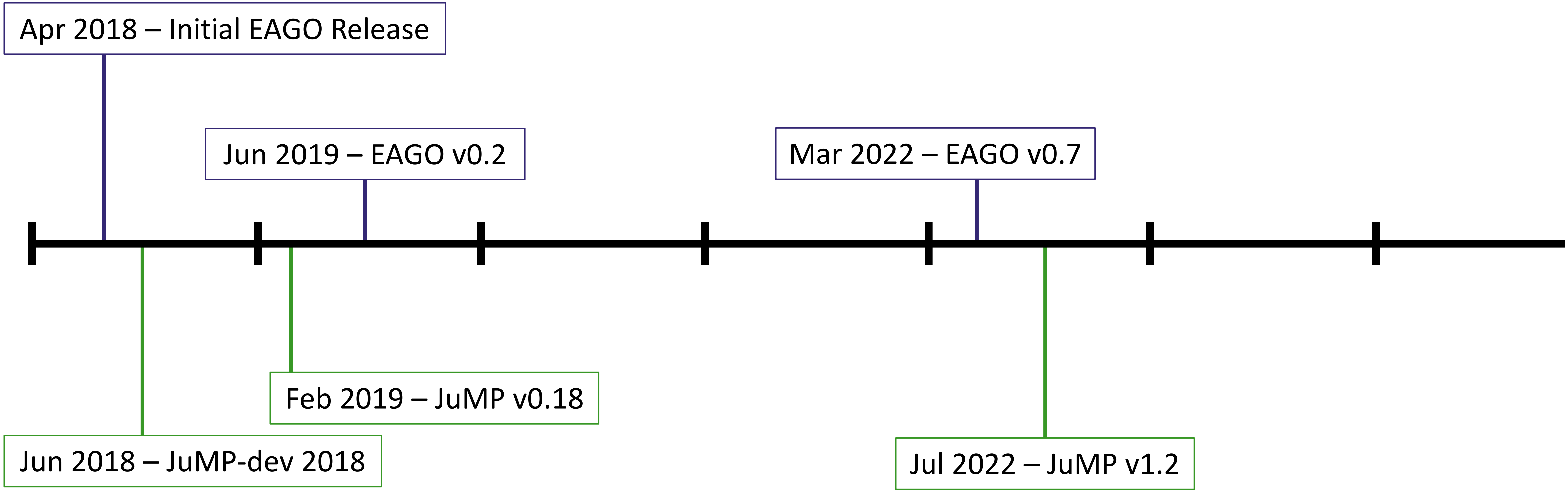
Timeline



3


[3] https://jump.dev/meetings/bordeaux2018/

Timeline



[3] 

[3] https://jump.dev/meetings/bordeaux2018/

Timeline

Apr 2018 – Initial EAGO Release

Jun 2018 – JuMP-dev 2018

Feb 2019 – JuMP v0.18

Jun 2019 – EAGO v0.2

Mar 2022 – EAGO v0.7

Jul 2022 – JuMP v1.2

Nov 2022 – Matthew Wilhelm Successfully Defends Ph.D. Dissertation

[3]

[3] https://jump.dev/meetings/bordeaux2018/

Timeline

Apr 2018 – Initial EAGO Release

Jun 2018 – JuMP-dev 2018

Feb 2019 – JuMP v0.18

Jun 2019 – EAGO v0.2

Mar 2022 – EAGO v0.7

Jul 2022 – JuMP v1.2

Nov 2022 – Matthew Wilhelm Successfully Defends Ph.D. Dissertation

Jan 2023 – Dimitri Joins PSOR

Jun 2023 – EAGO v0.8

[3]

[3] https://jump.dev/meetings/bordeaux2018/

Timeline

Apr 2018 – Initial EAGO Release

Jun 2018 – JuMP-dev 2018

[3]

Feb 2019 – JuMP v0.18

Jun 2019 – EAGO v0.2

Mar 2022 – EAGO v0.7

Jul 2022 – JuMP v1.2

Nov 2022 – Matthew Wilhelm Successfully Defends Ph.D. Dissertation

Jan 2023 – Dimitri Joins PSOR

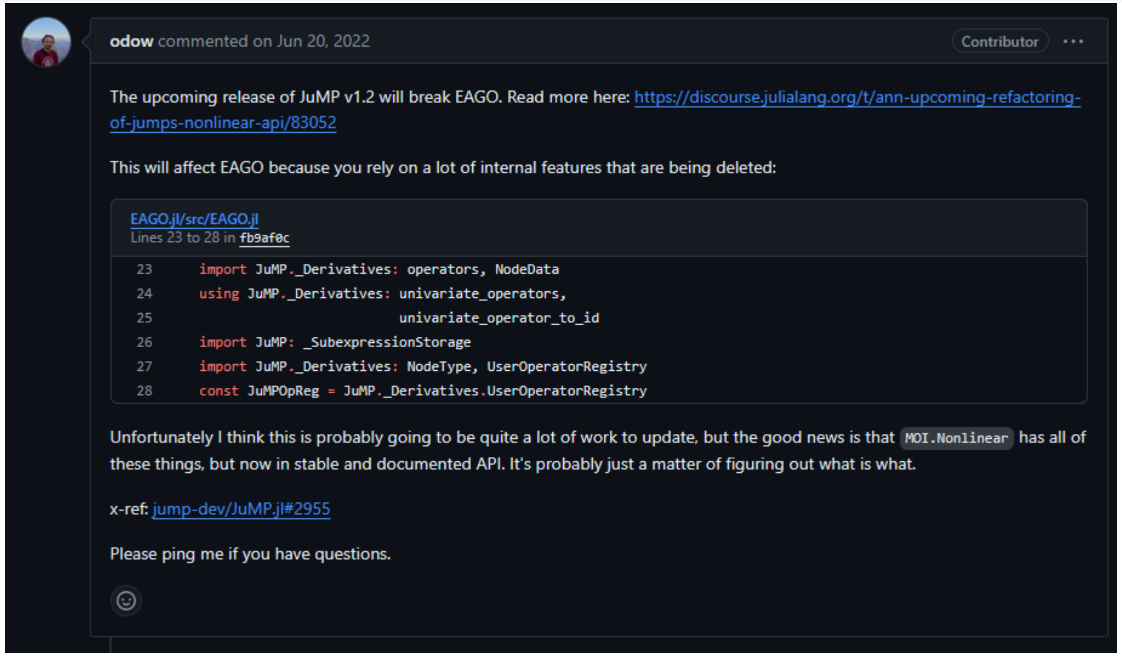
Jun 2023 – EAGO v0.8

Jun 2023 – MOI v1.17

[3] https://jump.dev/meetings/bordeaux2018/

Nonlinear Refactor

- Nonlinearity moved from JuMP to MOI
- Relatively low impact for EAGO
  - JuMP._Derivates → MOI.Nonlinear
  - Other minor name changes
  - Changes to EAGO's internal operator registry



Nonlinear Refactor

- Nonlinearity moved from JuMP to MOI
- Relatively low impact for EAGO
  - JuMP._Derivates → MOI.Nonlinear
  - Other minor name changes
  - Changes to EAGO's internal operator registry

odow commented on Jun 20, 2022 Contributor

The upcoming release of JuMP v1.2 will break EAGO. Read more here: https://discourse.julialang.org/t/ann-upcoming-refactoring-of-jumps-nonlinear-api/83052

This will affect EAGO because you rely on a lot of internal features that are being deleted:

EAGO.jl/src/EAGO.jl
Lines 23 to 28 in fb9af0c

```julia
import JuMP._Derivatives: operators, NodeData
using JuMP._Derivatives: univariate_operators,
                         univariate_operator_to_id
import JuMP: _SubexpressionStorage
import JuMP._Derivatives: NodeType, UserOperatorRegistry
const JuMPOpReg = JuMP._Derivatives.UserOperatorRegistry
```

Unfortunately I think this is probably going to be quite a lot of work to update, but the good news is that `MOI.Nonlinear` has all of these things, but now in stable and documented API. It's probably just a matter of figuring out what is what.

x-ref: jump-dev/JuMP.jl#2955

Please ping me if you have questions.

```diff
-    for n in d.nd
-        if n.nodetype == JuMP._Derivatives.VARIABLE
             if !haskey(variable_dict, n.index)
                 variable_dict[n.index] = true
             end
         end
```

```diff
+    for n in d.nodes
+        if n.type == MOINL.NODE_VARIABLE
             if !haskey(variable_dict, n.index)
                 variable_dict[n.index] = true
             end
         end
```

EAGO's Directed Acyclic Graph

- MOI nodes converted to EAGO nodes
  - Forward pass: relaxations
  - Reverse pass: constraint propagation

EAGO's Directed Acyclic Graph

- MOI nodes converted to EAGO nodes
  - Forward pass: relaxations
  - Reverse pass: constraint propagation

$$f(x) = \sin(x)^2 + x$$

```
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 1, -1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 4, 1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_UNIVARIATE, 15, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 3)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VALUE, 1, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 1)
```

→

```
EAGO.Node(EAGO.EXPRESSION, EAGO.PLUS, 0, 0, 2, [2, 6])
EAGO.Node(EAGO.EXPRESSION, EAGO.POW, 0, 0, 2, [3, 5])
EAGO.Node(EAGO.EXPRESSION, EAGO.SIN, 0, 0, 1, [4])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.CONSTANT, EAGO.CONST_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
```

EAGO's Directed Acyclic Graph

- MOI nodes converted to EAGO nodes
  - Forward pass: relaxations
  - Reverse pass: constraint propagation

$$f(x) = \sin(x)^2 + x$$

```
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 1, -1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 4, 1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_UNIVARIATE, 15, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 3)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VALUE, 1, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 1)
```

→

```
EAGO.Node(EAGO.EXPRESSION, EAGO.PLUS, 0, 0, 2, [2, 6])
EAGO.Node(EAGO.EXPRESSION, EAGO.POW, 0, 0, 2, [3, 5])
EAGO.Node(EAGO.EXPRESSION, EAGO.SIN, 0, 0, 1, [4])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.CONSTANT, EAGO.CONST_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
```

EAGO’s Directed Acyclic Graph

- MOI nodes converted to EAGO nodes
  - Forward pass: relaxations
  - Reverse pass: constraint propagation

$$f(x) = \sin(x)^2 + x$$

```
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 1, -1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 4, 1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_UNIVARIATE, 15, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 3)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VALUE, 1, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 1)
```

→

```
EAGO.Node(EAGO.EXPRESSION, EAGO.PLUS, 0, 0, 2, [2, 6])
EAGO.Node(EAGO.EXPRESSION, EAGO.POW, 0, 0, 2, [3, 5])
EAGO.Node(EAGO.EXPRESSION, EAGO.SIN, 0, 0, 1, [4])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.CONSTANT, EAGO.CONST_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
```

- Issue: MOI.ScalarNonlinearFunction uses a symbolic tree…

EAGO's Directed Acyclic Graph

- MOI nodes converted to EAGO nodes
  - Forward pass: relaxations
  - Reverse pass: constraint propagation

$$f(x) = \sin(x)^2 + x$$

```
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 1, -1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 4, 1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_UNIVARIATE, 15, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 3)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VALUE, 1, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 1)
```

→

```
EAGO.Node(EAGO.EXPRESSION, EAGO.PLUS, 0, 0, 2, [2, 6])
EAGO.Node(EAGO.EXPRESSION, EAGO.POW, 0, 0, 2, [3, 5])
EAGO.Node(EAGO.EXPRESSION, EAGO.SIN, 0, 0, 1, [4])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.CONSTANT, EAGO.CONST_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
```

- Issue: MOI.ScalarNonlinearFunction uses a symbolic tree…

```
+(^(sin(MOI.VariableIndex(1)), 2.0), MOI.VariableIndex(1))
```

EAGO's Directed Acyclic Graph

- MOI nodes converted to EAGO nodes
  - Forward pass: relaxations
  - Reverse pass: constraint propagation

$$f(x) = \sin(x)^2 + x$$

```
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 1, -1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_MULTIVARIATE, 4, 1)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_CALL_UNIVARIATE, 15, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 3)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VALUE, 1, 2)
MathOptInterface.Nonlinear.Node(MathOptInterface.Nonlinear.NODE_VARIABLE, 1, 1)
```

→

```
EAGO.Node(EAGO.EXPRESSION, EAGO.PLUS, 0, 0, 2, [2, 6])
EAGO.Node(EAGO.EXPRESSION, EAGO.POW, 0, 0, 2, [3, 5])
EAGO.Node(EAGO.EXPRESSION, EAGO.SIN, 0, 0, 1, [4])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.CONSTANT, EAGO.CONST_ATOM, 1, 0, 0, Int64[])
EAGO.Node(EAGO.VARIABLE, EAGO.VAR_ATOM, 1, 0, 0, Int64[])
```

- Issue: MOI.ScalarNonlinearFunction uses a symbolic tree…
  - Rewrite EAGO's internal routines?
  - Convert symbolic tree into a node tree?

```
+(^(sin(MOI.VariableIndex(1)), 2.0), MOI.VariableIndex(1))
```

Future Work

- Add support for MOI.ScalarNonlinearFunction

Future Work

- Add support for MOI.ScalarNonlinearFunction
- Update EAGO's use of JuMP for advanced problem formulations
  - SIPs
  - Implicit functions

Future Work

- Add support for MOI.ScalarNonlinearFunction
- Update EAGO's use of JuMP for advanced problem formulations
  - **SIPs**
  - Implicit functions

$$
\begin{aligned}
f^* &= \min_{\mathbf{x}} f(\mathbf{x}) \\
&\text{s.t. } 0 \geq \max g(\mathbf{x}, \tilde{\mathbf{y}}, \mathbf{p}) \\
&\text{s.t. } \mathbf{h}(\mathbf{x}, \tilde{\mathbf{y}}, \mathbf{p}) = \mathbf{0} \\
&\mathbf{x} \in X = \{\mathbf{x} \in \mathbb{R}^{n_x} : \mathbf{x}^L \leq \mathbf{x} \leq \mathbf{x}^U\} \\
&\mathbf{p} \in \mathbf{P} = \{\mathbf{p} \in \mathbb{R}^{n_p} : \mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U\} \\
&\tilde{\mathbf{y}} \in D_y \subset \mathbb{R}^{n_y}
\end{aligned}
$$

$$\rightarrow$$

$$
\begin{aligned}
f^* &= \min_{\mathbf{x}} f(\mathbf{x}) \\
&\text{s.t. } g(\mathbf{x}, \mathbf{y}(\mathbf{x}, \mathbf{p}), \mathbf{p}) \leq 0, \quad \forall \mathbf{p} \in P \\
&\mathbf{x} \in X = \{\mathbf{x} \in \mathbb{R}^{n_x} : \mathbf{x}^L \leq \mathbf{x} \leq \mathbf{x}^U\} \\
&P = \{\mathbf{p} \in \mathbb{R}^{n_p} : \mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U\}
\end{aligned}
$$

Future Work

- Add support for MOI.ScalarNonlinearFunction
- Update EAGO's use of JuMP for advanced problem formulations
  - **SIPs**
  - Implicit functions

```julia
using EAGO

f(x) = (1/3)*x[1]^2 + x[2]^2 + x[1]/2
gSIP(x, p) = (1.0 - (x[1]^2)*(p[1]^2))^2 - x[1]*p[1]^2 - x[2]^2 + x[2]
x_l = Float64[-1000.0, -1000.0]
x_u = Float64[1000.0, 1000.0]
p_l = Float64[0.0]
p_u = Float64[1.0]
EAGO.sip_solve(EAGO.SIPRes(), x_l, x_u, p_l, p_u, f, [gSIP], res_sip_absolute_tolerance = 1E-3, verbosity = 3);
```

$$\mathbf{p} \in \mathbf{P} = \{\mathbf{p} \in \mathbb{R}^{n_p} : \mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U\}$$

$$\tilde{\mathbf{y}} \in D_y \subset \mathbb{R}^{n_y}$$

Future Work

- Add support for MOI.ScalarNonlinearFunction
- Update EAGO's use of JuMP for advanced problem formulations
  - **SIPs**
  - Implicit functions

```julia
using JuMP, EAGO

factory = () -> EAGO.Optimizer(SubSolvers(; t = SIPOptimizer(SIPRes(), 2, 1)))
model = JuMP.Model(optimizer_with_attributes(factory))
@variable(model, -1000.0 <= x[i=1:2] <= 1000.0)
@variable(model, 0.0 <= p <= 1.0)
@constraint(model, (1.0 - (x[1]^2)*(p^2))^2 - x[1]*p^2 - x[2]^2 + x[2] <= 0.0)
@objective(model, Min, (1/3)*x[1]^2 + x[2]^2 + x[1]/2)
JuMP.optimize!(model)
```

$$\mathbf{p} \in \mathbf{P} = \{\mathbf{p} \in \mathbb{R}^{n_p} : \mathbf{p}^L \le \mathbf{p} \le \mathbf{p}^U\}$$

$$\tilde{\mathbf{y}} \in D_y \subset \mathbb{R}^{n_y}$$

Acknowledgements

Members of the PSOR Laboratory at the University of Connecticut

The JuMP community

Process Systems and Operations Research Laboratory

UCONN UNIVERSITY OF CONNECTICUT

JUMP

Funding:

National Science Foundation, Award No.: 1932723

National Science Foundation, Award No.: 2330054

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 or the United States Government.

Questions?

{ ISMP 2024 }

Process Systems and Operations Research Laboratory

EAGO

GPU-Accelerated Deterministic Global Optimization
Robert Gottlieb
Tuesday, July 23rd, 2024, 8:30 AM

https://www.psor.uconn.edu

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