EAGO

Easy Advanced Global Optimization

A Deterministic Nonconvex Optimization Package for Julia

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

Matthew Wilhelm, PhD Student
Matthew Stuber, Assistant Professor

Process Systems and Operations Research Laboratory
Department of Chemical and Biomolecular Engineering
UTC Institute for Advanced Systems Engineering
University of Connecticut

UCONN

First Things… Introductions!

- ❑ I’m Matt Wilhelm!
- ❑ I work at UCONN on robust and global nonconvex optimization
- ❑ Started working on EAGO, I’ve been in response to a couple things:

No low-level routines in global optimization.
- *Pyomo/JuMP:* great for higher level and meta-algorithms
- *Existing solvers:* Not open-source or aren’t meant to be extended.
- *Academic tools*: Exist as fragmented libraries across multiple languages.

→ **Julia**

- ***The real test is solver performance on standard benchmarks.***
- ***Want to be able to easily distribute work.***

→ **JuMP**

Integrate with AML

First Things… Introductions!

- ❑ I'm Matt Wilhelm!
- ❑ I work at UCONN on robust and global nonconvex optimization
- ❑ Started working o

No low-level
- *Pyomo/Ju*
- *Existing sc*
- *Academic*

- ***The real tes***
- ***Want to be able to easily distribute work.***

I. Relaxations Library
II. Modular Global Solve Algorithm
III. Standard Tools
IV. Access to Solver
V. Problem Formulation Tools

Julia

JuMP

Integrate with AML

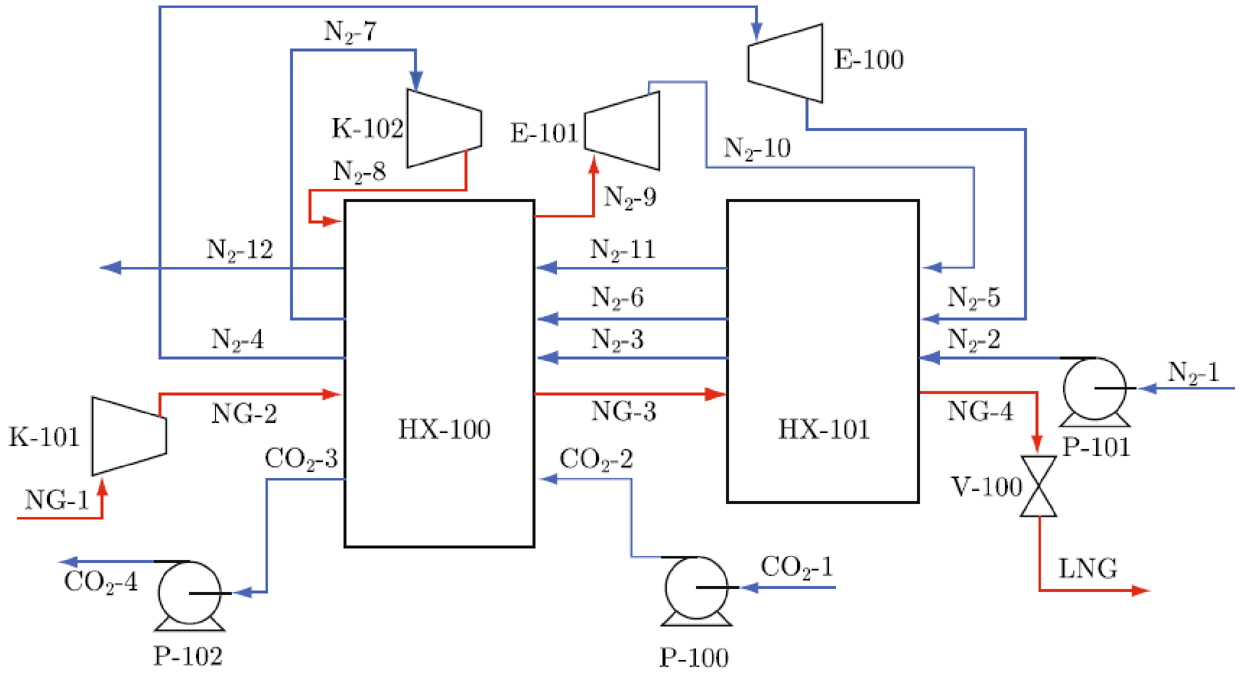
Simulation Motivation

- Nonconvex models from block diagrams are quite common
- Typical optimization problem formulate in state-control[1]

$$\min_{p \in P} f(x(p), p)$$
$$s.t. \quad f(x(p), p) = 0$$
$$g(x(p), p) \leq 0$$

p – Control variables
x – State variables

Offshore Liquefied Natural Gas Production[2]



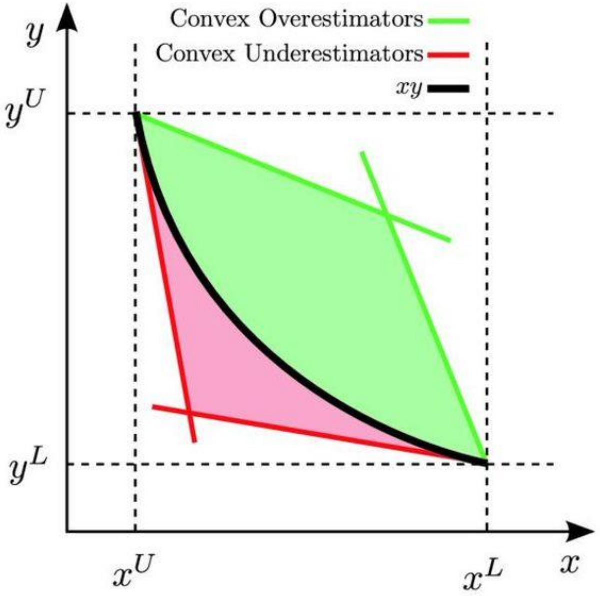
[1] **Optimum design of chemical plants with uncertain parameters.** I. E. Grossmann and R. W. H. Sargent (1978). AIChE Journal, 24(6):1021–1028.
[2] **Differentiable McCormick relaxations.** Khan, K. et al. (2017) *Journal Global Optimization*, 67(4), 687-729

McCormick Relaxations

Calculating convex bounds on bilinear term via a McCormick relaxations[3]

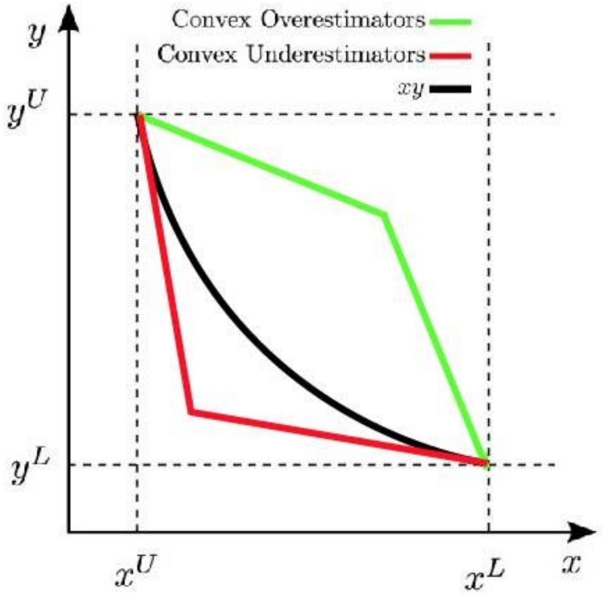
$$w = xy$$
$$x^L \le x \le x^U$$
$$y^L \le y \le y^U$$

Decomposition Method[4]



$$w \ge x^L y + x y^L - x^L y^L$$
$$w \ge x^U y + x y^U - x^U y^U$$
$$w \le x^U y + x y^L - x^U y^L$$
$$w \le x y^U + x^L y - x^L y^U$$

Composition method



$$w^{cv}(x,y) = \max(x^L y + x y^L - x^L y^L, x^U y + x y^U - x^U y^U)$$
$$w^{cc}(x,y) = \min(x^U y + x y^L - x^U y^L, x y^U + x^L y - x^L y^U)$$

[3] **Computability of global solutions to factorable nonconvex programs: Part I-Convex underestimating problems.** G. P. McCormick. Mathematical Programming, 10:147–175, 1976.

[4] **McCormick envelopes.** https://optimization.mccormick.northwestern.edu/index.php/McCormick_envelopes

McCormick Relaxations

McCormick Relaxations Via Method Overloading

```
# (Smooth) McCormick Object w/ Subgradient
struct SMCg{N,V<:AbstractInterval,T<:AbstractFloat} <: Real
  cc::T
  cv::T
  cc_grad::SVector{N,T}
  cv_grad::SVector{N,T}
  Intv::V
  cnst::Bool

  function SMCg{N,V,T}(cc1::T,cv1::T,cc_grad1::SVector{N,T},cv_grad1::SVector{N,T},
                       Intv1::V,cnst1::Bool) where {N,V<:AbstractInterval,T<:AbstractFloat}
    new(cc1,cv1,cc_grad1,cv_grad1,Intv1,cnst1)
  end
end

# Overload Addition Operator
function +(x::SMCg{N,V,T},y::SMCg{N,V,T}) where {N,V,T<:AbstractFloat}
    return SMCg{N,V,T}(x.cc+y.cc,
                       x.cv+y.cv,
                       x.cc_grad+y.cc_grad,
                       x.cv_grad+y.cv_grad,
                       (x.Intv+y.Intv),
                       (x.cnst && y.cnst))
end
```

McCormick Composition Rule[3]:
Consider the composite function $h = g \circ f$. Let $f^{cv}, f^{cc} : X \to \mathbb{R}$ be known convex and concave relaxations of f on X, respectively. Let $g^{cv}, g^{cc} : Z \to \mathbb{R}$ be known convex and concave relaxations of g on Z, respectively. Then,

$$\begin{aligned} h^{cv}(\mathbf{z}) &= g^{cv}(mid(f^{cv}(\mathbf{z}), f^{cc}(\mathbf{z}), x^{min})) \\ h^{cc}(\mathbf{z}) &= g^{cc}(mid(f^{cv}(\mathbf{z}), f^{cc}(\mathbf{z}), x^{max})) \end{aligned} \tag{4}$$

Where x^{min} is a point at which g^{cv} attains its infimum on X and x^{max} is a point at which g^{cc} attains its supremum on X.

Currently Supported Operators:

- ***Algebraic:*** +, -, /, *, ^, pow
- ***Trigonometric:*** sin, cos, tan, asin, acos, atan
- ***Hyperbolic:*** sinh, cosh, tanh, asinh, acosh, atanh
- ***Nonsmooth:*** min, max, abs, step, sign
- ***Others:*** exp, exp2, exp10, log, log2, log10, sqrt

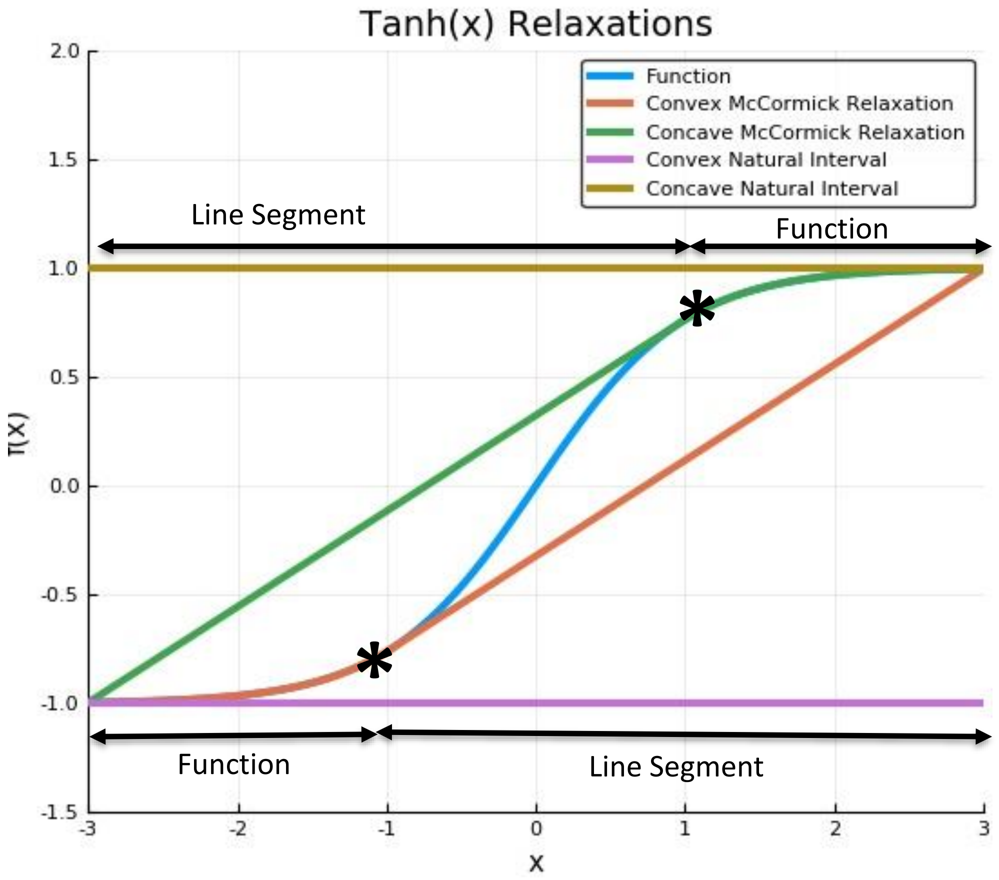
[5] McCormick-based relaxations of algorithms. Mitsos et al. (2009) *SIAM Journal on Optimization*, SIAM, 2009, 20, 73-601
[6] MC++: A versatile library for McCormick relaxations and Taylor models. B. Chachuat. http://www3.imperial.ac.uk/people/b.chachuat/research

Relaxation from Macros

List of convexity properties for standard functions, dictionary structure for relaxations, and registration/generation via macros.

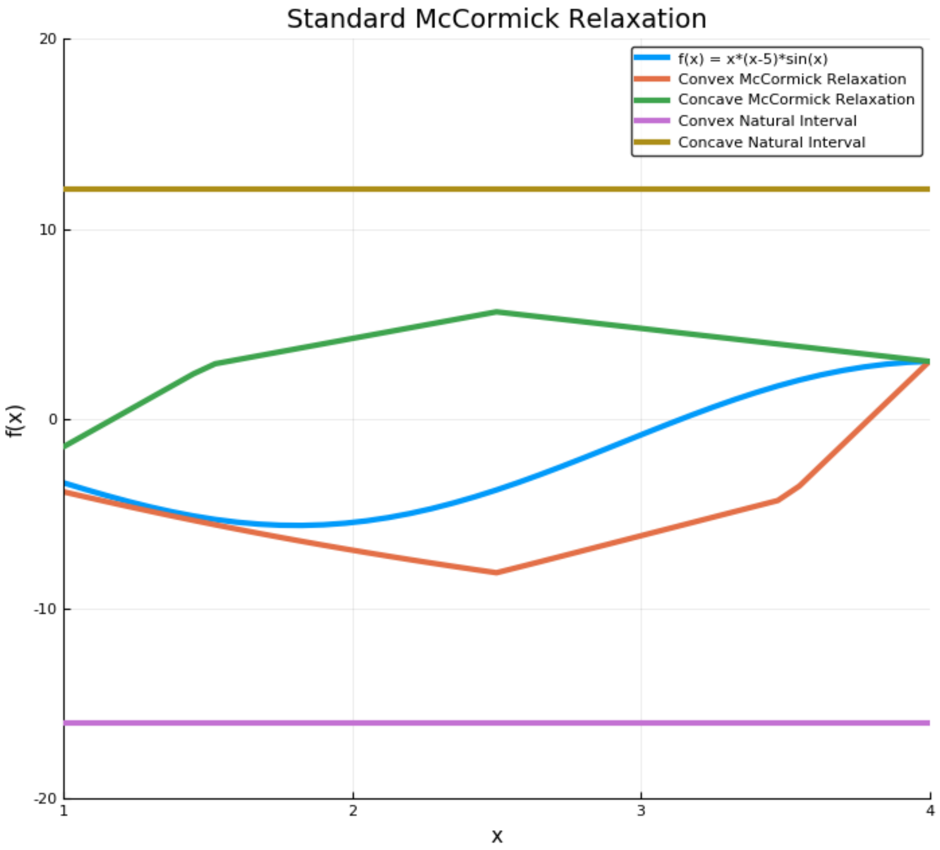
- :convex (exp, abs, sqr)
- :concave (log, sqrt)
- :convexoconcave (tanh)
- :concavoconvex (erfc)

```
@set_cvtrait Base.sinh(x)     :Concavoconvex  :Increasing  0.0  -Inf  Inf
@set_cvtrait Base.cosh(x)     :Convex         :DecrToIncr  0.0  -Inf  Inf
@set_cvtrait Base.tanh(x)     :Convexoconcave :Increasing  0.0  -Inf  Inf
@set_cvtrait Base.sech(x)     :Convexoconcave :IncrToDecr  Inf  -Inf  Inf

@set_cvtrait Base.asinh(x)    :Convexoconcave :Increasing  0.0  -Inf  Inf
@set_cvtrait Base.acosh(x)    :Convexoconcave :Increasing  0.0  -Inf  Inf
@set_cvtrait Base.atanh(x)    :Concave        :Increasing  Inf  1.0   Inf
@set_cvtrait Base.asech(x)    :Convexoconcave :Decreasing  0.5  0.0  1.0
```



McCormick Relaxations - Example

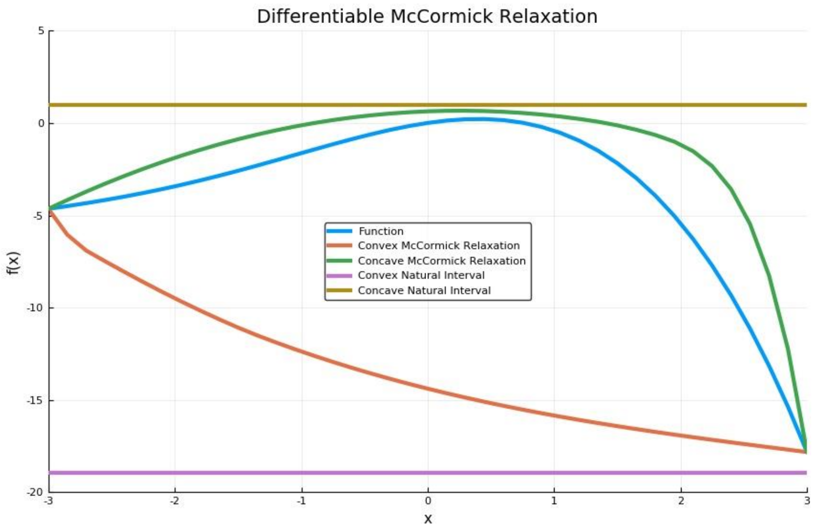
```julia
using EAGO

# create SmoothMcCormick seed object for x = 2.0 on [1.0,4.0] for relaxing
# a function f(x) on the interval box xIbox using mBox as a reference point

f(x) = x*(x-5.0)*sin(x)

x = 2.0                          # value of independent variable x
subx = seed_g(Float64,1,1)       # set initial subgradient of x to [1.0]
Intv = Interval(1.0,4.0)        # define interval to relax over

# create McCormick object
SMC = SMCg{1,Interval{Float64},Float64}(x,x,subx,subx,Intv,false)

fSMC = f(SMC)            # relax the function

cv = fSMC.cv              # convex relaxation
cc = fSMC.cc              # concave relaxation
cvgrad = fSMC.cv_grad     # subgradient/gradient of convex relaxation
ccgrad = fSMC.cc_grad     # subgradient/gradient of concave relaxation
Iv = fSMC.Intv            # retrieve interval bounds of f(x) on Intv
```

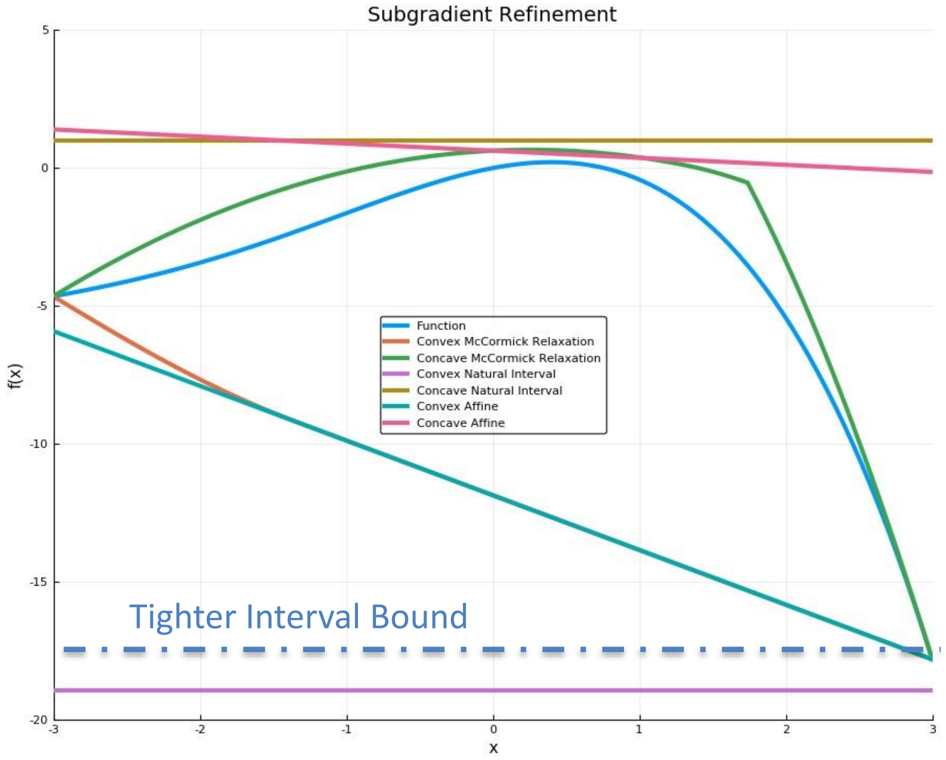


McCormick Relaxations (Extensions)

$$f(x) = -x^2 \exp(x) + max(x-5, \sin(x))$$

- Relaxations for max and min[7]
- Differentiable McCormick relaxations.[2]





- Subgradient-based interval tightening method[8]
- Reverse McCormick contractors[9]

[7] **Multivariate McCormick relaxations.** Tsoukalas, A. and Mitsos, A. (2014) *Journal of Global Optimization,* 633-662
[2] **Differentiable McCormick relaxations.** Khan, K. et al. (2017) *Journal Global Optimization*, 67(4), 687-729
[8] **Tighter McCormick Relaxations through Subgradient Propagation.** Najman, J. and Mitsos, A. (2017) *https://arxiv.org/abs/1710.09188*
[9] **Reverse propagation of McCormick relaxations**. Wechsung, Achim, et al. (2015) *Journal of Global Optimization* 63(1) : 1-36.

McCormick Relaxation: Branch & Bound

Full Nonconvex NLP:

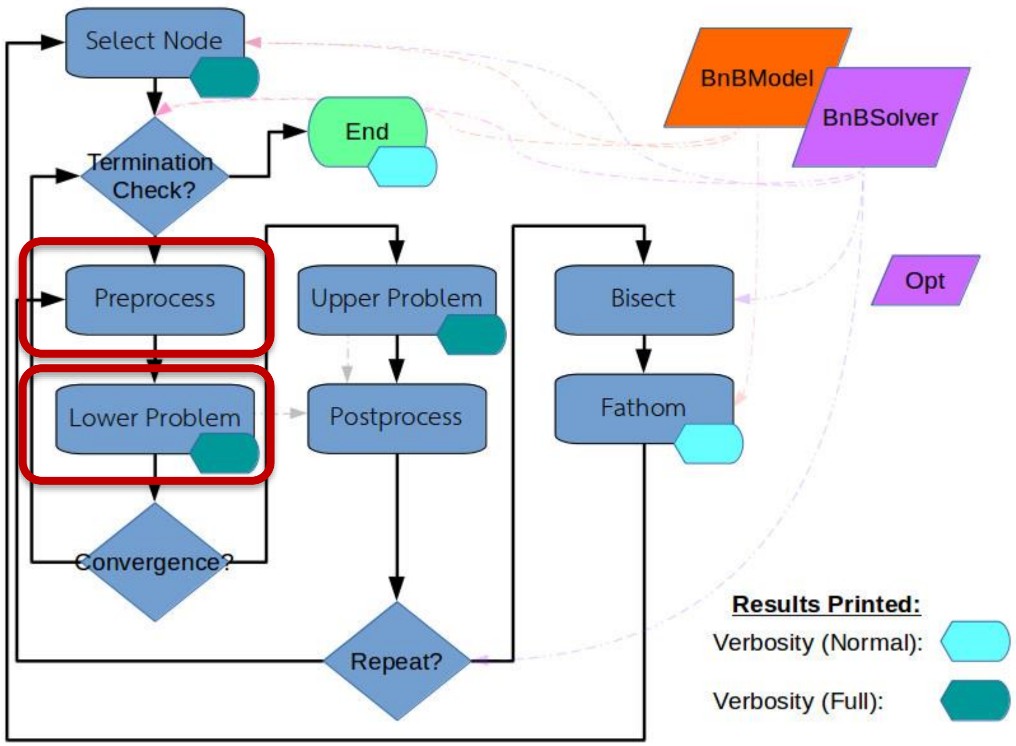
$$\min_{y \in Y} f(y) \quad s.t. \quad g(y) \leq 0$$

Lower Problem:

$$\min_{y \in Y} f^{cv}(y) \quad s.t. \quad g^{cv}(y) \leq 0$$

Preprocessing:

$$\min_{y \in Y} (\pm y) \quad s.t. \quad f^{cv}(y) = UBD, \quad g^{cv}(y) \leq 0$$



McCormick Relaxations:

- Nonsmooth NLP (Bundle Solver)
- Affine Relaxation (LP Solver)

Differentiable McCormick Relaxations:

- NLP solver

The standard EAGO solver

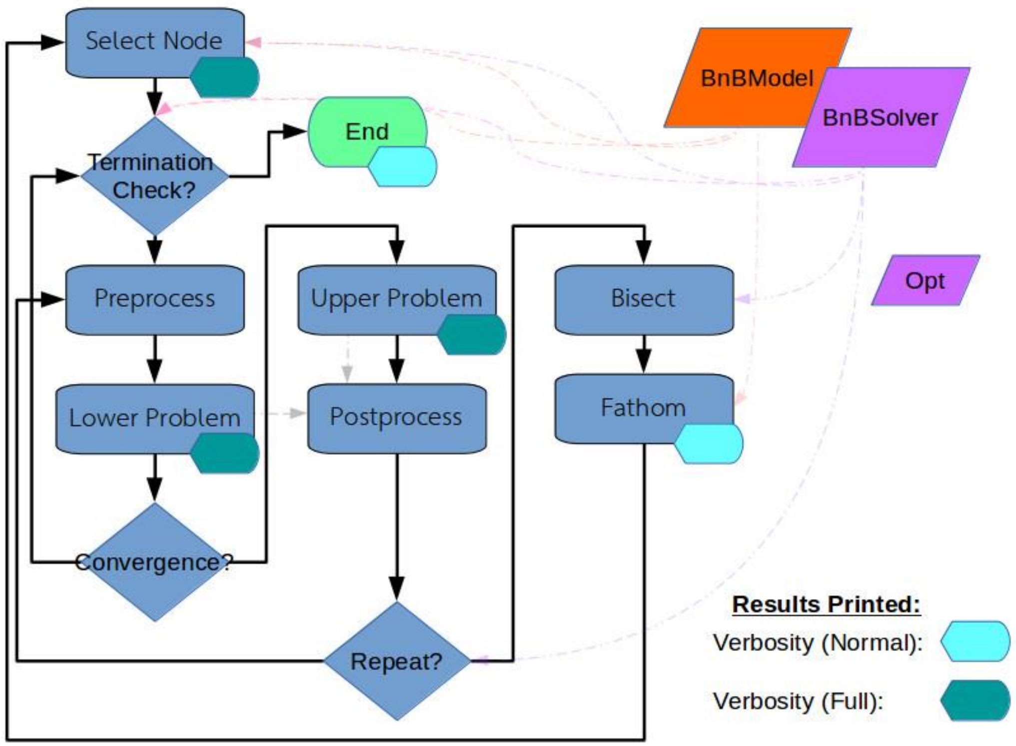
Search Routine:
- Best-first

Preprocessing:
- Interval constraint propagation[10]
- LP contractor[10] (NS McCormick Relaxations)

Lower Bounding Problem:
- NLP solve via Ipopt[11]
- LP solve with user selected solver

Upper Bounding Problem:
- Local NLP solve via Ipopt

Postprocessing:
- Duality-based bound tightening[10].



[10] **Domain reduction techniques for global NLP and MINLP optimization** Puranik, Y. and Sahinidas, N. (2017) *Constraints*, 22: 338-376.
[11] **On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming** A. Wächter and L. T. Biegler (2006), *Mathematical Programming* 106(1), 25-57.

McCormick versus Interval Methods

Three-hump camel function via Interval Method

```
m = Model(solver=EAGO_NLPSolver(LBD_func_relax = "Interval",
                                LBDsolvertype = "Interval",
                                UBDsolvertype = "Interval")
@variable(m, -5 <= x <= 5)
@variable(m, -5 <= y <= 5)
@NLobjective(m, Min, 2*x^2-1.05*x^4+(x^6)/6+x*y+y^2)
```

- ❑ **13442 Iterations**
- ❑ **3.09 Seconds**

Three-hump camel function via McCormick Approach

```
m = Model(solver=EAGO_NLPSolver(LBD_func_relax = "NS-STD",
                                LBDsolvertype = "LP",
                                UBDsolvertype = "Interval")
@variable(m, -5 <= x <= 5)
@variable(m, -5 <= y <= 5)
@NLobjective(m, Min, 2*x^2-1.05*x^4+(x^6)/6+x*y+y^2)
```

- ❑ **432 Iterations**
- ❑ **0.1 Seconds**

Swapping in a custom lower-bound

```

m = Model(solver = EAGO_NLPSolver())
@variable(m, -5 <= x <= 5)
@variable(m, -5 <= x <= 5)
@NLobjective(m, Min, 2*x^2-1.05*x^4+(x^6)/6+x*y+y^2)
JuMP.build()

function custom_LBD(x::Vector{Interval{Float64}},   # Box
                    k::Int,                          # Iteration number
                    p::Int,                          # Tree Depth
                    opt,                             # Data passed from solver
                    temp)                            # Data passed between problems
    pnt = mid.(X)   # Solution point
    val = (2*x[1]^2-1.05*x[1]^4+(x[1]^6)/6+x[1]*x[2]+x[2]^2).lo # Solution value
    # Return lower bound, solution value, feasibility, and duality info
    return [val,pnt,true,[]]
end
set_LBD!(custom_LBD,m)

solve(m)
```

Branch and Bound Implementation

- **Full Customizable**
  - All blocks are functions that can be reset.

- **Common Heuristics are available via API**:
  - **Search Methods:** Best-First Search, etc.
  - **Bisection Methods:** Relative Width, etc.

- **Typical output templates available**
  - Convergence plots
  - CSV data for batch runs
  - Solver progress output to console

Setting search to a breadth-first style

```
julia> EAGO.set_Branch_Scheme!(a,"breadth")
```

Console display with solver progress

```
Iteration   NodeID   Current_LBD   Global_LBD   Global_UBD   NodesLeft   Absolute_Gap   Absolute_Ratio
    60,      115,     3.0004273,    3.0000000,   3.0005493,      8,       0.0005493,      0.9998169
    61,      118,     3.0000000,    3.0000000,   3.0004883,      9,       0.0004883,      0.9998373
    62,      119,     3.0000610,    3.0000000,   3.0004883,     10,       0.0004883,      0.9998373
    63,      120,     3.0000000,    3.0000000,   3.0002747,     11,       0.0002747,      0.9999085
    64,      121,     3.0002136,    3.0000000,   3.0002747,      8,       0.0002747,      0.9999085
    65,      124,     3.0000000,    3.0000000,   3.0002747,      7,       0.0002747,      0.9999085
    66,      125,     3.0000305,    3.0000000,   3.0002747,      6,       0.0002747,      0.9999085
    67,      122,     3.0000610,    3.0000610,   3.0002747,      5,       0.0002136,      0.9999288
    68,      123,     3.0002747,    3.0000610,   3.0002747,      4,       0.0002136,      0.9999288
    69,      116,     3.0001221,    3.0001221,   3.0002747,      3,       0.0001526,      0.9999491
    70,      117,     3.0005493,    3.0001221,   3.0002747,      2,       0.0001526,      0.9999491
    71,      110,     3.0002441,    3.0002441,   3.0002747,      1,       0.0000305,      0.9999898
Convergence Tolerance Reached
First Solution Found at Node 64
UBD = 3.00027466105553004
Solution is :
    X[1] = -0.999969482421875
    X[2] = 2.0000267028808594
Total UBD problems solved = 62 in 0.022459687999999999 seconds.
Total LBD problems solved = 72 in 1.2186544090000009 seconds.
```

Transform MINLP to nonconvex NLP

Objective: $\min_{x} F_2$

Constraints:

Mass Balances + Process Models AND the below:

Polytropic Gas

$$(\kappa-1)\ln(P_1^s)+\kappa\ln(T_2)=(\kappa-1)\ln(P_2^s)+\kappa\ln(T_1)$$

Pinch-Point Energy Balances

$$\min_{p\in P}\{EBP_H^p-EBP_C^p\}=-Q_c$$

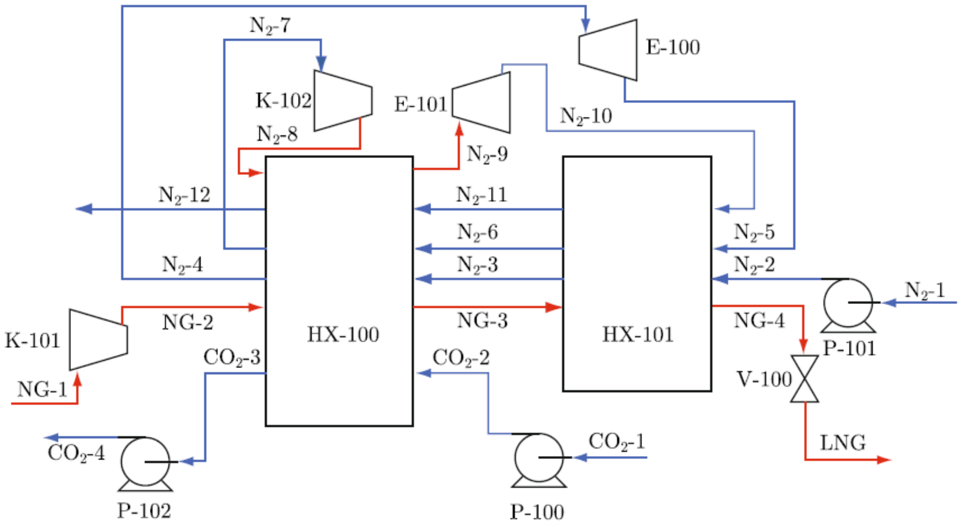
$$Q_H+\sum_{i\in H}Fc_{p,i}(T_i^{in}-T_i^{out})=Q_C+\sum_{j\in C}Fc_{p,j}(T_j^{in}-T_j^{out})$$

$$T^p=\begin{cases}T_i^{in} & \forall p=i\in H\\ T_j^{in}+\Delta T_{\min} & \forall p=j\in C\end{cases}$$

$$EBP_H^p=\sum_{i\in H}Fc_{p,i}\left[\max\left\{0,T^p-T_i^{out}\right\}-\max\left\{0,T^p-T_i^{in}\right\}-\max\left\{0,T^{\min}-T_i^p\right\}+\max\left\{0,T^p-T_i^{\max}\right\}\right] \qquad \forall p\in P$$

$$EBP_C^p=\sum_{j\in C}Fc_{p,j}\left[\max\left\{0,(T^p-\Delta T_{\min})-T_j^{in}\right\}-\max\left\{0,(T^p-\Delta T_{\min})-T_j^{out}\right\}-\max\left\{0,(T^p-\Delta T_{\min})-t^{\max}\right\}+\max\left\{0,t^{\min}-(T^p-\Delta T_{\min})\right\}\right] \qquad \forall p\in P$$

Offshore Liquidfied Natural Gas Production[2]



[2] **Differentiable McCormick relaxations.** Khan, K. et al. (2017) *Journal Global Optimization*, 67(4), 687-729

Transform MINLP to nonconvex NLP

Objective: $\min_{x} F_2$

Constraints:

Mass

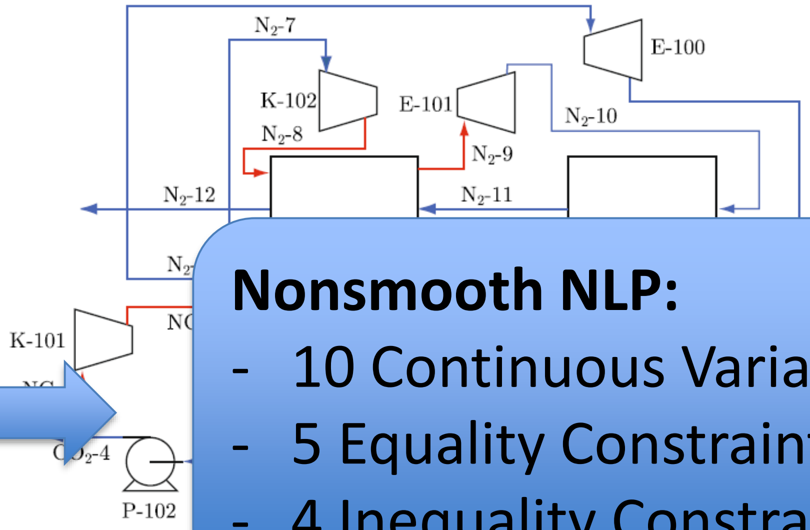
Polytrop

$(\kappa$

Pinch-P

MINLP:
- 173 Continuous Variables
- 363 Binary Variables
- 143 Equality Constraints
- 1101 Inequality Constraints

***Offshore Liquidfied Natural Gas Production*[1]**



Nonsmooth NLP:
- 10 Continuous Variables
- 5 Equality Constraints
- 4 Inequality Constraints

$$Q_H + \sum_{i \in H} Fc_{p,i}(T_i^{in} - T_i^{out}) = Q_C + \sum_{j \in C} Fc_{p,j}(T_j^{in} - T_j^{out})$$

$$T^p = \begin{cases} T_i^{in} & \forall p = i \in H \\ T_j^{in} + \Delta T_{\min} & \forall p = j \in C \end{cases}$$

$$EBP_H^p = \sum_{i \in H} Fc_{p,i}\left[\max\left\{0, T^p - T_i^{out}\right\} - \max\left\{0, T^p - T_i^{in}\right\} - \max\left\{0, T^{\min} - T_i^p\right\} + \max\left\{0, T^p - T_i^{\max}\right\}\right] \quad \forall p \in P$$

$$EBP_C^p = \sum_{j \in C} Fc_{p,j}\left[\max\left\{0, (T^p - \Delta T_{\min}) - T_j^{in}\right\} - \max\left\{0, (T^p - \Delta T_{\min}) - T_j^{out}\right\} - \max\left\{0, (T^p - \Delta T_{\min}) - t^{\max}\right\} + \max\left\{0, t^{\min} - (T^p - \Delta T_{\min})\right\}\right] \quad \forall p \in P$$

[2] **Differentiable McCormick relaxations.** Khan, K. et al. (2017) *Journal Global Optimization*, 67(4), 687-729

Branching on Implicit Functions

McCormick relaxations can bound functions via fixed-point methods:

- Theoretically, well-behaved relaxations of state variables that aren't factorable.
- Applicable to process units containing nonlinear equations that require nonlinear solve.
- Assumes the existence of an implicit function y = x(p).

Linear Parametric System:

$$A(p)x = b(p)$$

General Nonlinear Systems:

$$\boldsymbol{h}(\boldsymbol{z},\boldsymbol{p}) = \begin{cases} \boldsymbol{c}_{\boldsymbol{A}}^{i-1} - \boldsymbol{c}_{\boldsymbol{A}}^{i} + \Delta t\left(k_1 \boldsymbol{c}_{\boldsymbol{Y}}^{i}\boldsymbol{c}_{\boldsymbol{Z}}^{i} - c_{O_2}(k_{2f}+k_{3f})\boldsymbol{c}_{\boldsymbol{A}}^{i} + \frac{k_{2f}}{K_2}\boldsymbol{c}_{\boldsymbol{D}}^{i} + \frac{k_{3f}}{K_3}\boldsymbol{c}_{\boldsymbol{B}}^{i} - k_5 {\boldsymbol{c}_{\boldsymbol{A}}^{i}}^{2}\right) = 0 \\ \boldsymbol{c}_{\boldsymbol{B}}^{i-1} - \boldsymbol{c}_{\boldsymbol{B}}^{i} + \Delta t\left(k_{3f}\boldsymbol{c}_{\boldsymbol{A}}^{i} c_{O_2} - \left(\frac{k_{3f}}{K_3} + k_4\right)\boldsymbol{c}_{\boldsymbol{B}}^{i}\right) = 0 \\ \boldsymbol{c}_{\boldsymbol{D}}^{i-1} - \boldsymbol{c}_{\boldsymbol{D}}^{i} + \Delta t\left(k_{2f}\boldsymbol{c}_{\boldsymbol{A}}^{i} c_{O_2} - \frac{k_{2f}}{K_2}\boldsymbol{c}_{\boldsymbol{D}}^{i}\right) = 0 \\ \boldsymbol{c}_{\boldsymbol{Y}}^{i-1} - \boldsymbol{c}_{\boldsymbol{Y}}^{i} + \Delta t\left(-k_{1s}\boldsymbol{c}_{\boldsymbol{Y}}^{i}\boldsymbol{c}_{\boldsymbol{Z}}^{i}\right) = 0 \\ \boldsymbol{c}_{\boldsymbol{Z}}^{i-1} - \boldsymbol{c}_{\boldsymbol{Z}}^{i} + \Delta t\left(-k_1\boldsymbol{c}_{\boldsymbol{Y}}^{i}\boldsymbol{c}_{\boldsymbol{Z}}^{i}\right) = 0 \end{cases}$$

[12] **Convex and concave relaxations of implicit functions.** Stuber, M.D. et al. (2015) *Optimization Methods and Software*, 30, 424-460
[5] **McCormick-based relaxations of algorithms.** Mitsos et al. (2009) *SIAM Journal on Optimization*, SIAM, 2009, 20, 73-601
[13] **Direct measurement of the fast, reversible addition of oxygen to cyclohexadienyl radicals in nonpolar solvents**, J. W. Taylor, et al. *Phys. Chem. A*, 108 (2004), pp. 7193–7203.

Special Forms from General Functions

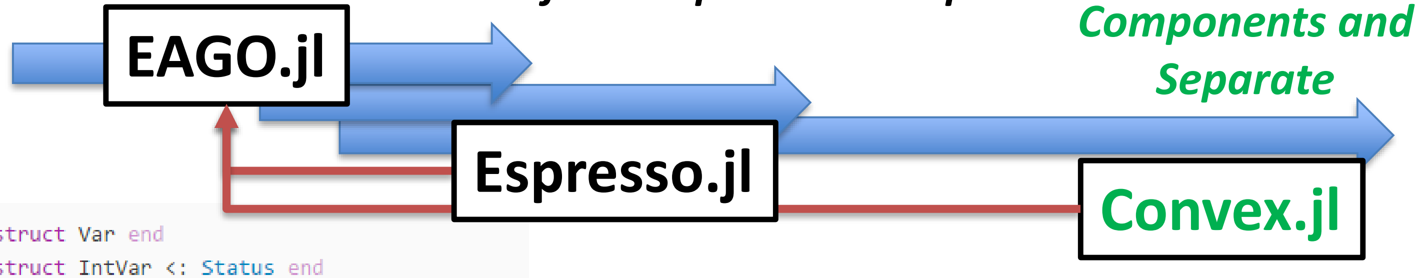
Can we formulate a global solver with as low a barrier to entry as low solvers with AD methods?

```julia
# Globally optimize script
#(Nonlinear Regression for Indentation)
function f(x)
    A = 0.0
    SSE = 0.0
    for j=1:100
        A = 0.0
        for i=1:2
           A += x[i]*exp(-x[2*i]*j)
        end
        SSE += (A - data[j])^2
    end
    return A
end

Optimize_Script(f,xL,xU, g = ginput,  h = hinput,
                solver = EAGO_NLPSolver())
```

Find Standard Forms
LP, QP, MILP, MIQP

Separate Standard Forms from Expression Graph

Identify Convex Components and Separate



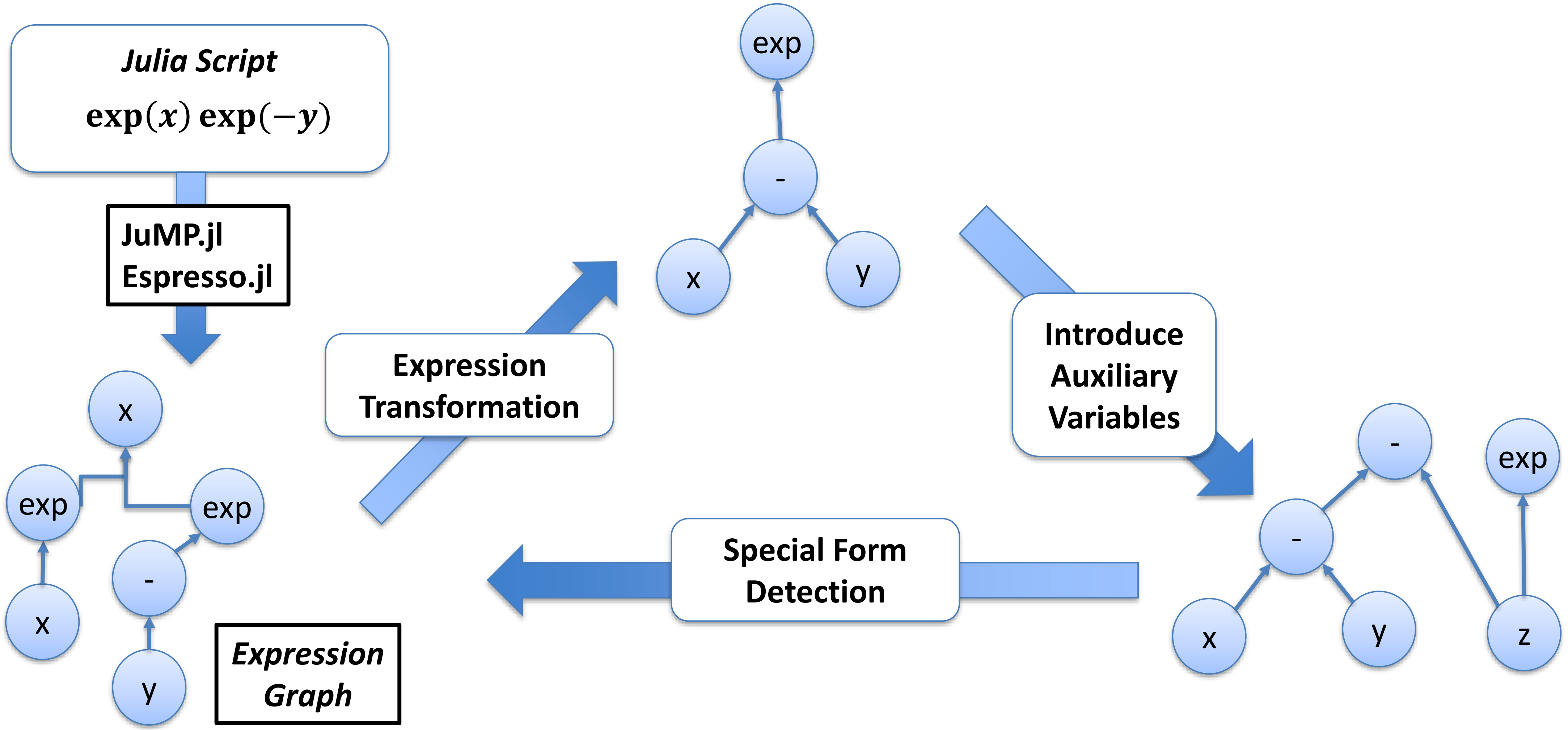

```julia
struct Var end
struct IntVar <: Status end
struct ContVar <: Status end

struct Status end
struct Linear <: Status end
struct Quadratic <: Status end
struct Nonlinear <: Status end

struct LQF{S<:Var,T<:Eqn} <: Real
    lp::SparseVector{Tv,Int}
    qp::Union{SparseMatrix{Tv,Int},Void}
    milp::Union{SparseVector{Tv,Int},Void}
    miqp::Union{SparseMatrix{Tv,Int},Void}
end
```

$$f(x) = f^{nl}(x) + \sum_i a_i x_i + f^{cv}(x) + \sum_i \sum_j b_{ij} x_i x_j$$

$$g_k(x) = g_k^{nl}(x) + \sum_i a_i x_i + g_k^{cv}(x) + \sum_i \sum_j b_{ij} x_i x_j$$

...and Mixed-Integer Forms (Coming Soon)

Reformulation Architecture



Further Software Development

Current Limitations	Main Development Schemes
▪ Weaker bounds than AVM methods[7]. ▪ Implicit function bounding is complex.	▪ Auxiliary Variable Oracle. ▪ Add Implicit Function Detection.
▪ Upper bounding NLP solvers may not support non-smooth problems.	▪ Add Lexicographic AD scheme[14].
▪ Currently can't handle general MINLP problems.	▪ Add a Branch-And-Cut Library.

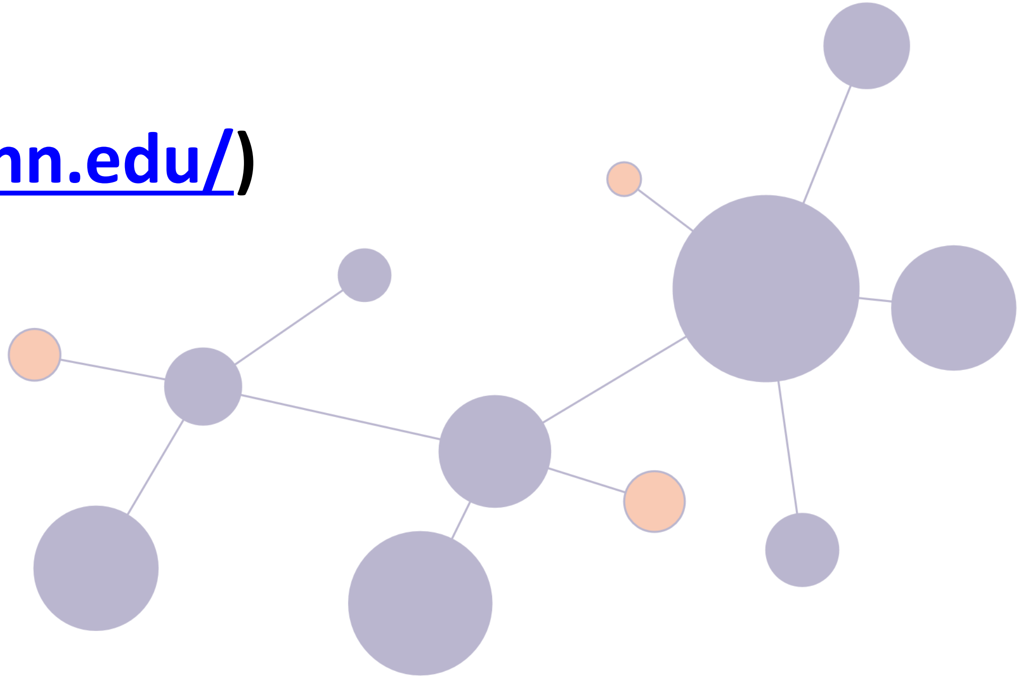
[7] **Multivariate McCormick relaxations.** Tsoukalas, A. and Mitsos, A. (2014) *Journal of Global Optimization,* 633-662
[14] **Branch-locking AD techniques for nonsmooth composite functions and nonsmooth implicit functions** Khan, K. et al. (2017) *Optimization Methods and Software*, 0, 1-29

Acknowledgements

PSOR lab at UCONN (https://psor.uconn.edu/)

- Professor Matthew Stuber
- Chenyu Wang
- William Hale

University of Connecticut



Fin…

SIP Solvers

- **EAGOSemiInfinite** implements implicit and explicit meta-algorithms

Explicit SIP

$$f^* = \min_{\mathbf{x}\in X} f(\mathbf{x})$$
$$\text{s.t.} \quad g(\hat{\mathbf{y}},\mathbf{x},\mathbf{p}) \le 0, \forall(\mathbf{y},\mathbf{p}) \in Y\times P$$
$$\mathbf{h}(\hat{\mathbf{y}},\mathbf{x},\mathbf{p}) = \mathbf{0}, \quad \forall(\mathbf{y},\mathbf{p}) \in Y\times P$$

Implicit SIP

$$f^* = \min_{\mathbf{x}\in X} f(\mathbf{x})$$
$$\text{s.t.} \quad g(\mathbf{y}(\mathbf{x},\mathbf{p}),\mathbf{x},\mathbf{p}) \le 0, \quad \forall \mathbf{p} \in P$$

- Solved generally via restriction of right-hand side method[9,4]
  - *Discretization of uncertainty set for upper/lower bounds.*
  - *Finite convergence (if Slater point arbitrarily near minimizer).*

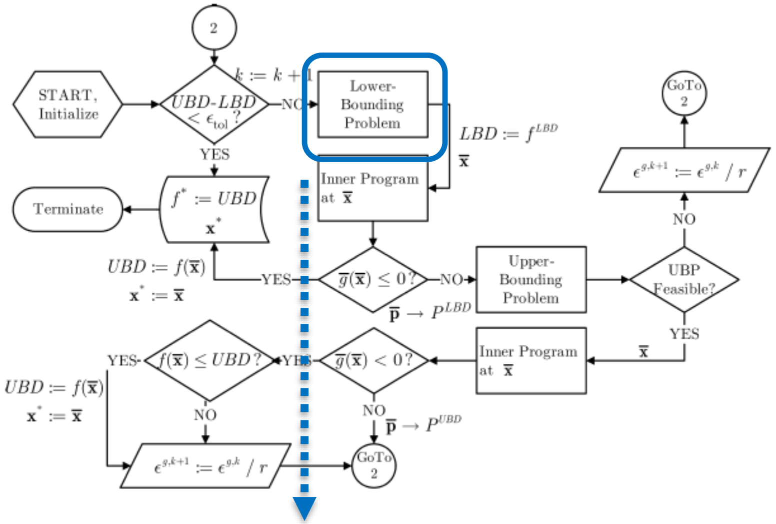
- Problem size and complexity dramatically increased with equality constraints[4,9,10]

[9] **Global optimization of semi-infinite programs via restriction of the right-hand side**. Mitsos, A. (2011). *Optimization* 60:10-11, 1291-1308

[4] **Semi-Infinite Optimization with Implicit Functions.** Stuber, M.D., Barton, P.I. (2015) *Industrial & Engineering Chemistry Research*, 54, 307-317

[10] **Evaluation of process systems operating envelopes.** Stuber, M.D., (2013) Ph.D Thesis.

Flowchart for restriction of RHS method[4]



Lower Bounding Problem (explicit)
for restriction of RHS method[4]

$$f^{LBD} = \min_{\mathbf{x}\in X} f(\mathbf{x})$$
$$\text{s.t.} \quad g_{SIP}(\hat{\mathbf{y}},\mathbf{x},\mathbf{p}) \le 0 \qquad \forall(\mathbf{y},\mathbf{p}) \in Y^{LBD}\times P^{LBD}$$
$$\mathbf{h}(\hat{\mathbf{y}},\mathbf{x},\mathbf{p}) = \mathbf{0} \qquad \forall(\mathbf{y},\mathbf{p}) \in Y^{LBD}\times P^{LBD}$$