Open energy models: benchmarking, profiling and debugging tools for JuMP

Joaquim Dias Garcia (Soma Energy)

Energy

By "energy" we actually mean "energy systems" that typically include

- conversion
- transportation
- storage

These come in various flavours:

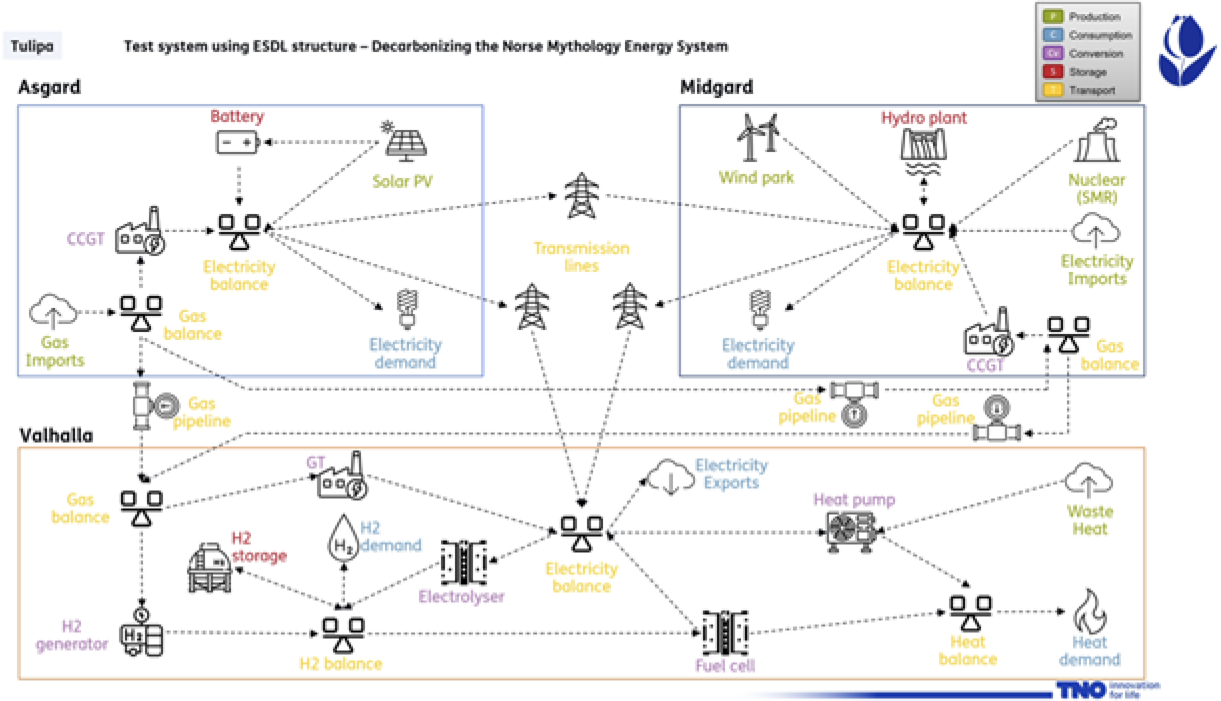
- Electricity / power systems (the most common case in this talk)
- Heat
- Natural Gas
- Hydrogen
- Combinations of the above and more…

Energy

Here is an example from:

"Tulipa Energy Model: Mathematical Formulation", Tejada-Arango e al., 2023

https://arxiv.org/pdf/2309.07711



Energy models

Here we mean:

Mathematical programming models of energy systems

- production costing models
- power flow models
- integrates resource planning models
- commodity flow models
- etc

$$
\begin{aligned}
\min_{x_1,\ldots,x_n} \quad & \sum_{j=1}^{n} a_j^T x_j + b_0 & \\
\text{s.t.} \quad & \sum_{j=1}^{n} A_{ij} x_j + b_i \in \mathcal{C}_i & i = 1 \ldots m \\
& x_j \in \mathcal{V}_j & j = 1 \ldots n
\end{aligned}
$$

Open energy models

Here we mean:

Open source implementations of mathematical programming models of energy systems

Here is a massive (non-exhaustive) list

https://wiki.openmod-initiative.org/wiki/Open_Models

from

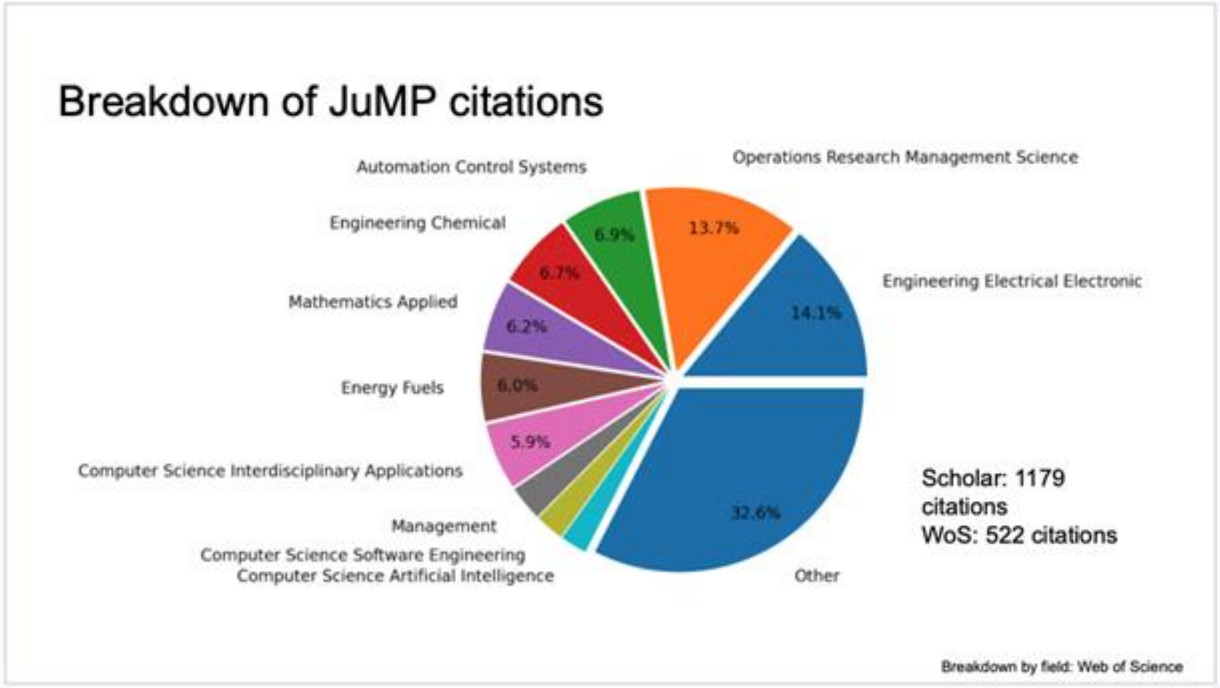
openmod open energy modelling initiative

Open energy models in JuMP

All that… but using JuMP

Why it matters?

- State of JuMP 2022



Open energy models in JuMP

All that… but using JuMP

Why it matters?

- Past presentations

https://jump.dev/open-energy-modeling/2024/09/19/open-energy-models/

Open Energy Modeling at JuMP-dev

open-energy-modeling · 19 Sep 2024

Author: Oscar Dowson

Contents

This post ended up being pretty long, so here is a table of contents if you want to JuMP (if you will) around:

1. [2024] Applied optimization with JuMP at SINTEF
2. [2024] Introduction to TulipaEnergyModel.jl
3. [2024] SpineOpt.jl: A highly adaptable modelling framework for multi-energy systems
4. [2024] Solving the Market-to-Market Problem in Large Scale Power Systems
5. [2024] PiecewiseAffineApprox.jl
6. [2023] How JuMP Enables Abstract Energy System Models
7. [2023] TimeStruct.jl: Multi Horizon Time Modelling in JuMP
8. [2023] Designing a Flexible Energy System Model Using Multiple Dispatch
9. [2022] UnitJuMP: Automatic Unit Handling in JuMP
10. [2022] SparseVariables.jl: Efficient Sparse Modelling with JuMP
11. [2022] Benchmarking Nonlinear Optimization with AC Optimal Power Flow
12. [2021] Modelling Australia's National Electricity Market with JuMP
13. [2021] AnyMOD.jl: A Julia package for creating energy system models
14. [2021] Power Market Tool (POMATO)
15. [2021] UnitCommitment.jl Security-Constrained Unit Commitment in JuMP
16. [2021] A Brief Introduction to InfrastructureModels
17. [2019] PowerSimulations.jl
18. [2017] Stochastic programming in energy systems
19. [2017] PowerModels.jl: a Brief Introduction

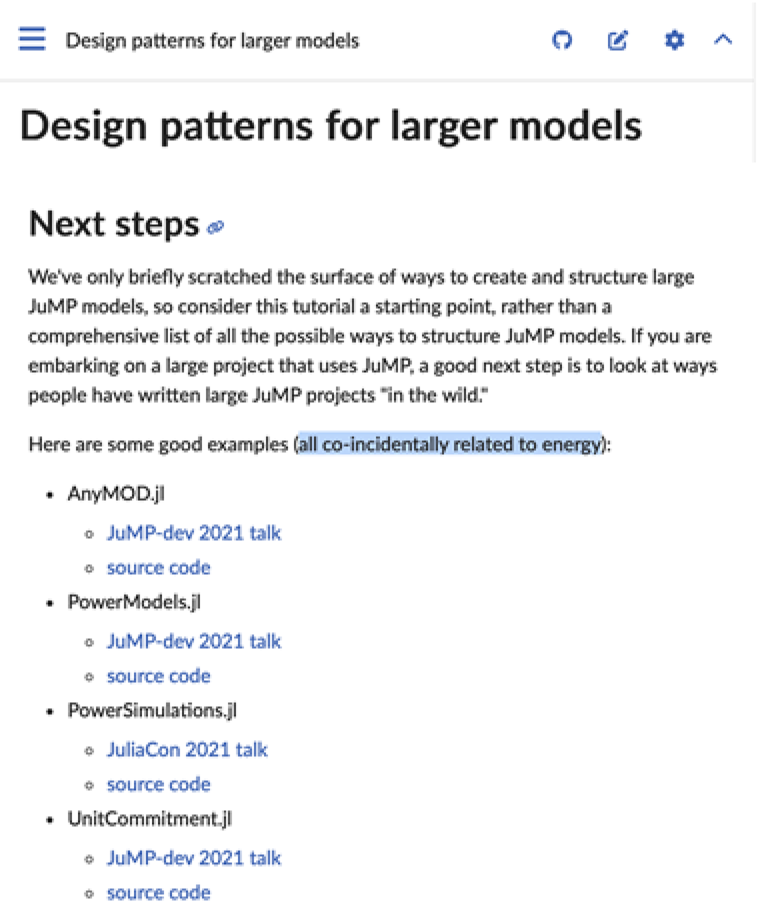
Open energy models in JuMP

All that… but using JuMP

Why it matters?

- We even recommend them

https://jump.dev/JuMP.jl/stable/tutorials/getting_started/design_patterns_for_larger_models/



Design patterns for larger models

Design patterns for larger models

Next steps

We've only briefly scratched the surface of ways to create and structure large JuMP models, so consider this tutorial a starting point, rather than a comprehensive list of all the possible ways to structure JuMP models. If you are embarking on a large project that uses JuMP, a good next step is to look at ways people have written large JuMP projects "in the wild."

Here are some good examples (all co-incidentally related to energy):

- AnyMOD.jl
  - JuMP-dev 2021 talk
  - source code
- PowerModels.jl
  - JuMP-dev 2021 talk
  - source code
- PowerSimulations.jl
  - JuliaCon 2021 talk
  - source code
- UnitCommitment.jl
  - JuMP-dev 2021 talk
  - source code

Benchmark Instances

Open energy modeling benchmarks

https://github.com/jump-dev/open-energy-modeling-benchmarks

- Fully reproducible scripts to
  - Generate fixed instances
  - Test iterative solves
- 68 benchmark instances in MPS format
  - Range from easy to hard (Gurobi cannot solve in 2 hours)
  - Instances submitted for consideration in **MIPLIB 2024**
- From various energy models
  - **GenX** (MIT / Princeton)
  - **Sienna** (NREL / Berkeley)
  - **Tulipa** (TNO, Netherlands eScience center, TU Delft and Utrecht University)
  - **SpineOpt** (EU Funding including DTU, KTH, TNO, EPRI and others)
  - PowerModels (Los Alamos National Lab)
  - UnitCommitment.jl (Argonne National Lab)

What can we do with those

- Benchmark solvers
  - Compare solvers
  - Compare algorithms
  - Track performance improvements
- Benchmark the Julia code
  - `JuMP`
  - `MOI`
  - Solver wrappers
  - Energy models

Benchmarking Julia code

- Can be done with [jump-dev/open-energy-modeling-benchmarks]
  - `julia --project=XXX XXX/main.jl --case=XXX --run -profile`
  - `time_limit = 1.0` (there might be multiple solves)
- Sienna
  - `Sienna_modified_RTS_GMLC_DA_sys_NetCopperPlate_Horizon12_Day29`
  - `total_time: 1.994`
  - `JuMP:       1.329`
  - `HiGHS.jl:   1.329`
  - `Highs_run:  1.329`
- Tulipa
  - `TulipaEnergyModel_1_EU_investment_simple_168h`
  - `total_time: 2.829`
  - `JuMP:       0.446`
  - `HiGHS.jl:   0.446`
  - `Highs_run:  0.297`
- GenX
  - `GenX_1_three_zones`
  - `total_time: 1.600`
  - `JuMP:       1.600`
  - `HiGHS.jl:   0.985`
  - `Highs_run:  0.985`

Benchmarking Julia code: concrete results

- JuMP ecosystem
  - `MutableArithmetics.jl` (missing optimizations)
  - Faster MPS writer
  - Optimized adding bounds to variables (too many C API calls)
  - New macro timing function: `print_macro_timing_summary`
  - Better performance tips and code structuring suggestion
- Energy models
  - Better use of performance tips (such as `set_string_names_on_creation`)
  - Avoid repeated computations in macros
  - Avoid operations outside macros (or use `add_to_expression!` for in-place operations)
- Results from all the above
  - Tulipa builds models 30% faster
  - GenX.jl builds models 2x faster and allocations dropped 75%
  - Sienna builds models 2x faster with 15% fewer allocations

Comparing algorithms: Test heuristics in HiGHS

Table: The number of models in each benchmark set for which the new heuristics find a feasible solution. Energy models include some OET models from PyPSA (openenergybenchmark.org).

Heuristic	240 MIPLIB models	88 Energy models
Feasibility Jump	83 (35%)	7 (8%)
Local MIP	128 (53%)	4 (5%)

Interesting lesson:
Successful primal heuristics from the literature worked poorly for energy models

Analysing Instances

`MathOptAnalyzer`

- Generic interface for analysis of optimization problem instances
  - Extendable for multiple analysis types
- Flexible API
  - Can be used in summary mode
  - Can be integrated in larger code base
- Currently contains 3 analyzers
  - Numerical
  - Feasibility (and optimality)
  - Infeasibility

- Some analysis just require MPS files
  - Your model does not need to be written in `JuMP` / `MOI`

MathOptAnalyzer.Numerical

```
model = Model() # no solver needed
@variable(model, x >= 1e9);
@variable(model, y >= 0);
@constraint(model, x + y >= 10)
@objective(model, Min, 2 * x + y);
```

MathOptAnalyzer.Numerical

```julia
model = Model() # no solver needed
@variable(model, x >= 1e9);
@variable(model, y >= 0);
@constraint(model, x + y >= 10)
@objective(model, Min, 2 * x + y);

# analyse and cache information
data = MathOptAnalyzer.analyze(
    MathOptAnalyzer.Numerical.Analyzer(), model)
# print to screen
MathOptAnalyzer.summarize(data) # verbose = false
```

MathOptAnalyzer.Numerical

```julia
model = Model() # no solver needed
@variable(model, x >= 1e9);
@variable(model, y >= 0);
@constraint(model, x + y >= 10)
@objective(model, Min, 2 * x + y);

# analyse and cache information
data = MathOptAnalyzer.analyze(
    MathOptAnalyzer.Numerical.Analyzer(), model)
# print to screen
MathOptAnalyzer.summarize(data) # verbose = false
```

```
julia> MathOptAnalyzer.summarize(data)
## Numerical Analysis

## Configuration

  Dense fill-in threshold: 0.1
  Dense entries threshold: 1000
  Small coefficient threshold: 1.0e-5
  Large coefficient threshold: 100000.0

## Dimensions

  Number of variables: 2
  Number of constraints: 3
  Number of nonzeros in matrix: 2
  Constraint types:
    * MathOptInterface.ScalarAffineFunction{Float64}-MathOptInterface.GreaterThan
{Float64}: 1
    * MathOptInterface.VariableIndex-MathOptInterface.GreaterThan{Float64}: 2

## Coefficient ranges

  Matrix:     [1e+00, 1e+00]
  Objective:  [1e+00, 2e+00]
  Bounds:     [1e+00, 1e+09]
  RHS:        [1e+00, 1e+01]
```

MathOptAnalyzer.Numerical

```julia
model = Model() # no solver needed
@variable(model, x >= 1e9);
@variable(model, y >= 0);
@constraint(model, x + y >= 10)
@objective(model, Min, 2 * x + y);

# analyse and cache information
data = MathOptAnalyzer.analyze(
    MathOptAnalyzer.Numerical.Analyzer(), model)
# print to screen
MathOptAnalyzer.summarize(data) # verbose = false
```

```
# `LargeBoundCoefficient`

## What

A `LargeBoundCoefficient` issue is identified when a variable has a
bound with a large absolute value.

## Why

Large bounds can lead to numerical instability in the solution process.

## How to fix

Check if the bound is correct. Check if the units of variables and
coefficients are correct. Check if the number makes is
reasonable given that solver have tolerances. Sometimes these
bounds can be replaced by zeros.

## More information

- https://jump.dev/JuMP.jl/stable/tutorials/getting_started/tolerances/

## Number of issues

Found 1 issues

## List of issues

 * Variable: MOI.VariableIndex(1) with bound 1.0e9
```

MathOptAnalyzer.Numerical

```julia
model = Model() # no solver needed
@variable(model, x >= 1e9);
@variable(model, y >= 0);
@constraint(model, x + y >= 10)
@objective(model, Min, 2 * x + y);

# analyse and cache information
data = MathOptAnalyzer.analyze(
    MathOptAnalyzer.Numerical.Analyzer(), model)
# print to screen
MathOptAnalyzer.summarize(data) # verbose = false
# or
# print to file
open("my_report.txt", "w") do io
    return MathOptAnalyzer.summarize(io, data)
end
```

MathOptAnalyzer.Numerical

```
julia> list = MathOptAnalyzer.list_of_issue_types(data)
1-element Vector{Type}:
 MathOptAnalyzer.Numerical.LargeBoundCoefficient

julia> issues = MathOptAnalyzer.list_of_issues(data, list[1])
1-element Vector{MathOptAnalyzer.Numerical.LargeBoundCoefficient}:
 MathOptAnalyzer.Numerical.LargeBoundCoefficient(MOI.VariableIndex(1), 1.0e9)

julia> issues = MathOptAnalyzer.list_of_issues(
           data,
           MathOptAnalyzer.Numerical.LargeBoundCoefficient,
       )
1-element Vector{MathOptAnalyzer.Numerical.LargeBoundCoefficient}:
 MathOptAnalyzer.Numerical.LargeBoundCoefficient(MOI.VariableIndex(1), 1.0e9)

julia> MathOptAnalyzer.variable(issues[1], model)
x

julia> MathOptAnalyzer.value(issues[1])
1.0e9
```

MathOptAnalyzer.Numerical

What if my code is not in JuMP?

```
# if you just have an mps file
filename = joinpath("model.mps")
model = read_from_file(filename)
data = MathOptAnalyzer.analyze(
    MathOptAnalyzer.Numerical.Analyzer(), model)
open("my_report.txt", "w") do io
    return MathOptAnalyzer.summarize(io, data)
end
```

MathOptAnalyzer.Numerical

```
MathOptAnalyzer.Numerical.VariableNotInConstraints
MathOptAnalyzer.Numerical.EmptyConstraint
MathOptAnalyzer.Numerical.VariableBoundAsConstraint
MathOptAnalyzer.Numerical.DenseConstraint
MathOptAnalyzer.Numerical.SmallMatrixCoefficient
MathOptAnalyzer.Numerical.LargeMatrixCoefficient
MathOptAnalyzer.Numerical.SmallBoundCoefficient
MathOptAnalyzer.Numerical.LargeBoundCoefficient
MathOptAnalyzer.Numerical.SmallRHSCoefficient
MathOptAnalyzer.Numerical.LargeRHSCoefficient
MathOptAnalyzer.Numerical.SmallObjectiveCoefficient
MathOptAnalyzer.Numerical.LargeObjectiveCoefficient
MathOptAnalyzer.Numerical.SmallObjectiveQuadraticCoefficient
MathOptAnalyzer.Numerical.LargeObjectiveQuadraticCoefficient
MathOptAnalyzer.Numerical.SmallMatrixQuadraticCoefficient
MathOptAnalyzer.Numerical.LargeMatrixQuadraticCoefficient
MathOptAnalyzer.Numerical.NonconvexQuadraticObjective
MathOptAnalyzer.Numerical.NonconvexQuadraticConstraint
MathOptAnalyzer.Numerical.LargeDynamicRangeConstraint
MathOptAnalyzer.Numerical.LargeDynamicRangeMatrix
MathOptAnalyzer.Numerical.LargeDynamicRangeObjective
MathOptAnalyzer.Numerical.LargeDynamicRangeRHS
MathOptAnalyzer.Numerical.LargeDynamicRangeVariable
MathOptAnalyzer.Numerical.LargeDynamicRangeBound
```

`MathOptAnalyzer.Numerical` in action

TulipaEnergyModel_2_EU_SectorCoupling_P2x_8760h

Problem size

- 2.2 M constraints
- 1.5 M variables
- 5.4 M nonzeros

MathOptAnalyzer identified…

- 1.1 M constraints that could be trivially removed
- 81 K variables that could be trivially removed
- 53 K constraints with bad numerics

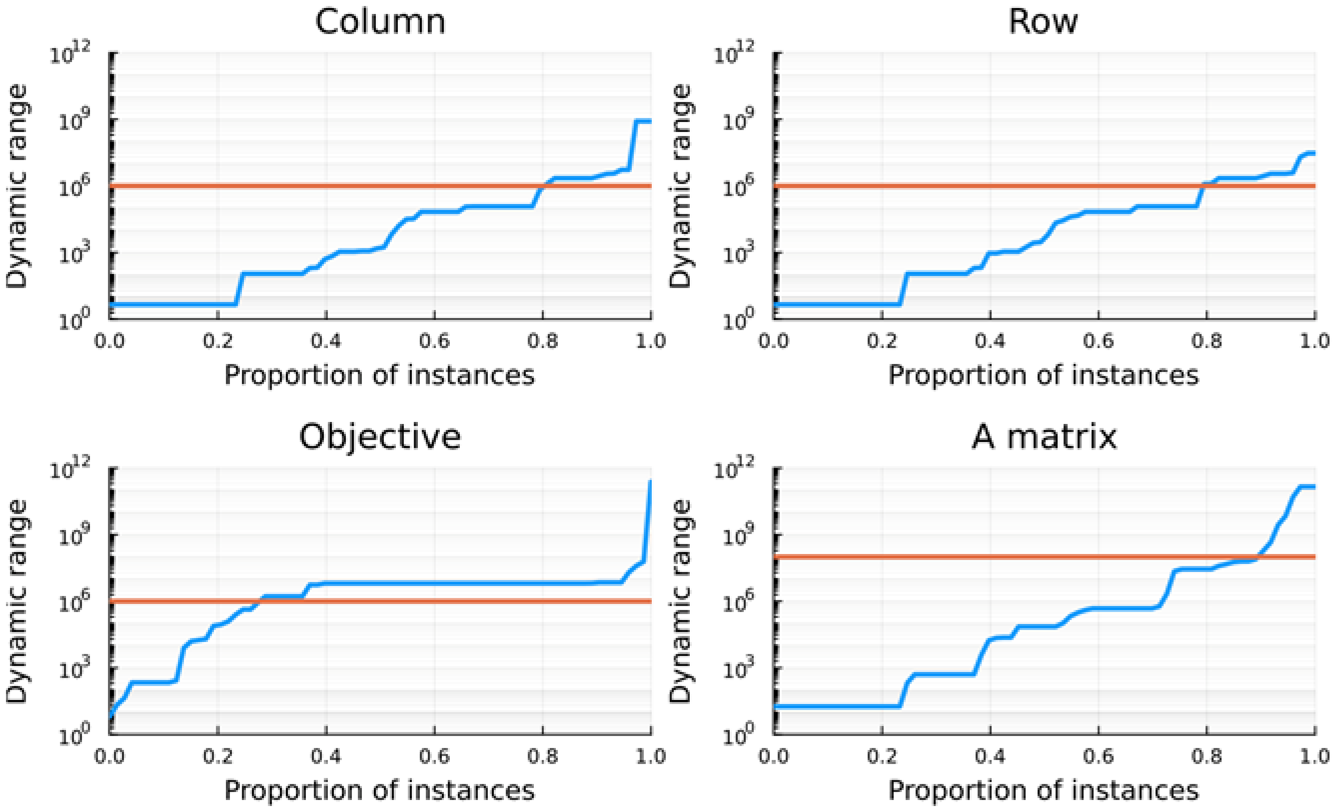
Consequences

HiGHS has a presolve to identify and remove trivial variables and constraints, but there are costs:

- Model is slower to build in JuMP
- More memory is required
- The problem size artificially inflated (size != difficulty)

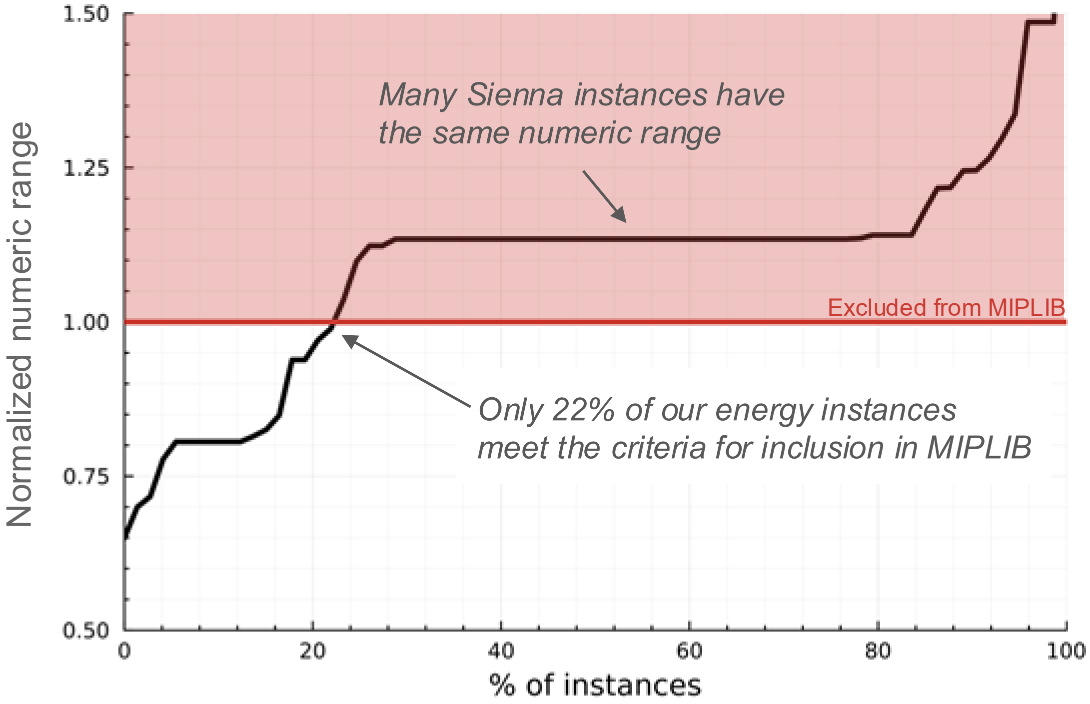
Big lesson: JuMP can do a better job at helping domain experts write high quality software

MathOptAnalyzer.Numerical in action



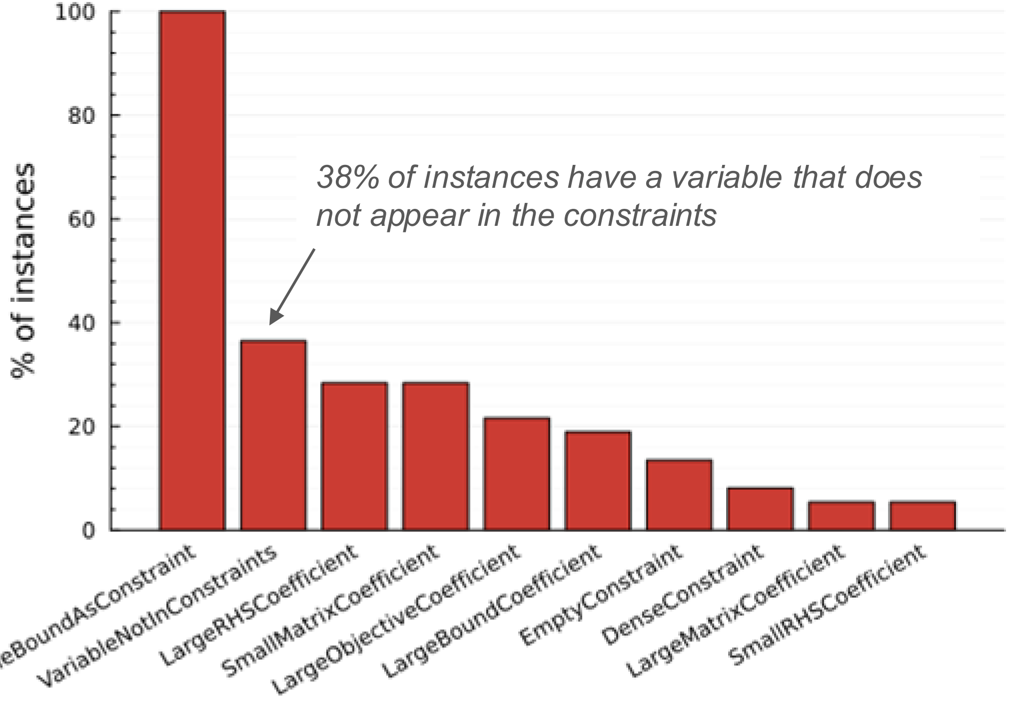
`MathOptAnalyzer.Numerical` in action

The numeric range is a measure of spread of the input data. Smaller is better



`MathOptAnalyzer.Numerical` in action

Percentage of benchmark instances by each common modeling issue detected by MathOptAnalyzer.



MathOptAnalyzer.Feasibility

```
model = Model()
@variable(model, x, Bin)
@constraint(model, c, x^4 >= 4) # nonlinear
```

MathOptAnalyzer.Feasibility

```julia
model = Model()
@variable(model, x, Bin)
@constraint(model, c, x^4 >= 4) # nonlinear
```

```
julia> data = MathOptAnalyzer.analyze(
           MathOptAnalyzer.Feasibility.Analyzer(),
           model,
           primal_point = Dict(JuMP.index(x) => 1.0),
       )
The model cannot be dualized. Automatically setting `dual_check = false`.
Feasibility analysis found 1 issues

julia> list = MathOptAnalyzer.list_of_issue_types(data)
1-element Vector{Type}:
 MathOptAnalyzer.Feasibility.PrimalViolation

julia> issues = MathOptAnalyzer.list_of_issues(data, list[1])
1-element Vector{MathOptAnalyzer.Feasibility.PrimalViolation}:
 MathOptAnalyzer.Feasibility.PrimalViolation(MathOptInterface.ConstraintIndex{MathOp
tInterface.ScalarNonlinearFunction, MathOptInterface.GreaterThan{Float64}}(1), 3.0)

julia> MathOptAnalyzer.constraint(issues[1], model)
c : (x ^ 4.0) - 4.0 ≥ 0

julia> MathOptAnalyzer.value(issues[1])
3.0
```

`MathOptAnalyzer.Feasibility`

- Can also test the available solutions after an `optimize!` call

```julia
model = Model(HiGHS.Optimizer)
set_silent(model)
@variable(model, x)
@constraint(model, c, x >= 0)
@objective(model, Min, x)
optimize!(model)
data = MathOptAnalyzer.analyze(
    MathOptAnalyzer.Feasibility.Analyzer(), model)
list = MathOptAnalyzer.list_of_issue_types(data)
```

MathOptAnalyzer.Infeasibility

```
MathOptAnalyzer.Infeasibility.InfeasibleBounds
MathOptAnalyzer.Infeasibility.InfeasibleIntegrality
MathOptAnalyzer.Infeasibility.InfeasibleConstraintRange
MathOptAnalyzer.Infeasibility.IrreducibleInfeasibleSubset
```

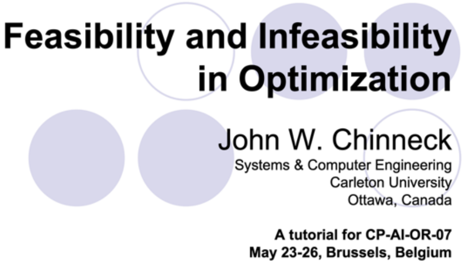
Search in the above order, in IIS do:

1 – Elastic Filter

2 – Deletion Filter

See:

www.sce.carleton.ca/faculty/chinneck/
docs/CPAIOR07InfeasibilityTutorial.pdf



MathOptAnalyzer.Infeasibility

```
model = Model()
@variable(model, 0 <= x <= 1)
@variable(model, 2 <= y <= 1)
@constraint(model, x + y <= 1)
@objective(model, Max, x + y)
```

MathOptAnalyzer.Infeasibility

```
model = Model()
@variable(model, 0 <= x <= 1)
@variable(model, 2 <= y <= 1)
@constraint(model, x + y <= 1)
@objective(model, Max, x + y)
```

```
julia> data = MathOptAnalyzer.analyze(
           MathOptAnalyzer.Infeasibility.Analyzer(), model)
Infeasibility analysis found 1 issues

julia> list = MathOptAnalyzer.list_of_issue_types(data)
1-element Vector{Type}:
 MathOptAnalyzer.Infeasibility.InfeasibleBounds

julia> ret = MathOptAnalyzer.list_of_issues(data, list[1])
1-element Vector{MathOptAnalyzer.Infeasibility.InfeasibleBounds}:
 MathOptAnalyzer.Infeasibility.InfeasibleBounds{Float64}(MOI.VariableIndex(2), 2.0, 1.0)
```

Note that no solver was used

Spin-off: `MathOptIIS`

Does Julia's package manager allow weird acronyms?

1. Irreducible Infeasible Set
2. Irreducibly Inconsistent Set
3. Irreducible Infeasible Subsystem
4. Infeasible Irreducible System
5. Irreducible Inconsistent Subsystem
6. Irreducibly Inconsistent System

Spin-off: `MathOptIIS`

- Backend of `MathOptAnalyzer.Infeasibility`
- Was the first IIS used by the (wrapper) `HiGHS.jl`
  - Automatically made available `compute_conflicts`
- Can be used independently
- Easy to plug in other solvers:
  - Ipopt?
    - Needs some extra work on NLP generalization

Debugging

performance and errors

Debugging performance

Premature optimization is the root of all evil

- Tutorials are your best friends!
  - jump.dev/JuMP.jl/stable/tutorials/getting_started/**design_patterns_for_larger_models**/
  - jump.dev/JuMP.jl/stable/tutorials/getting_started/**performance_tips**/
  - jump.dev/JuMP.jl/stable/tutorials/getting_started/**sum_if**/

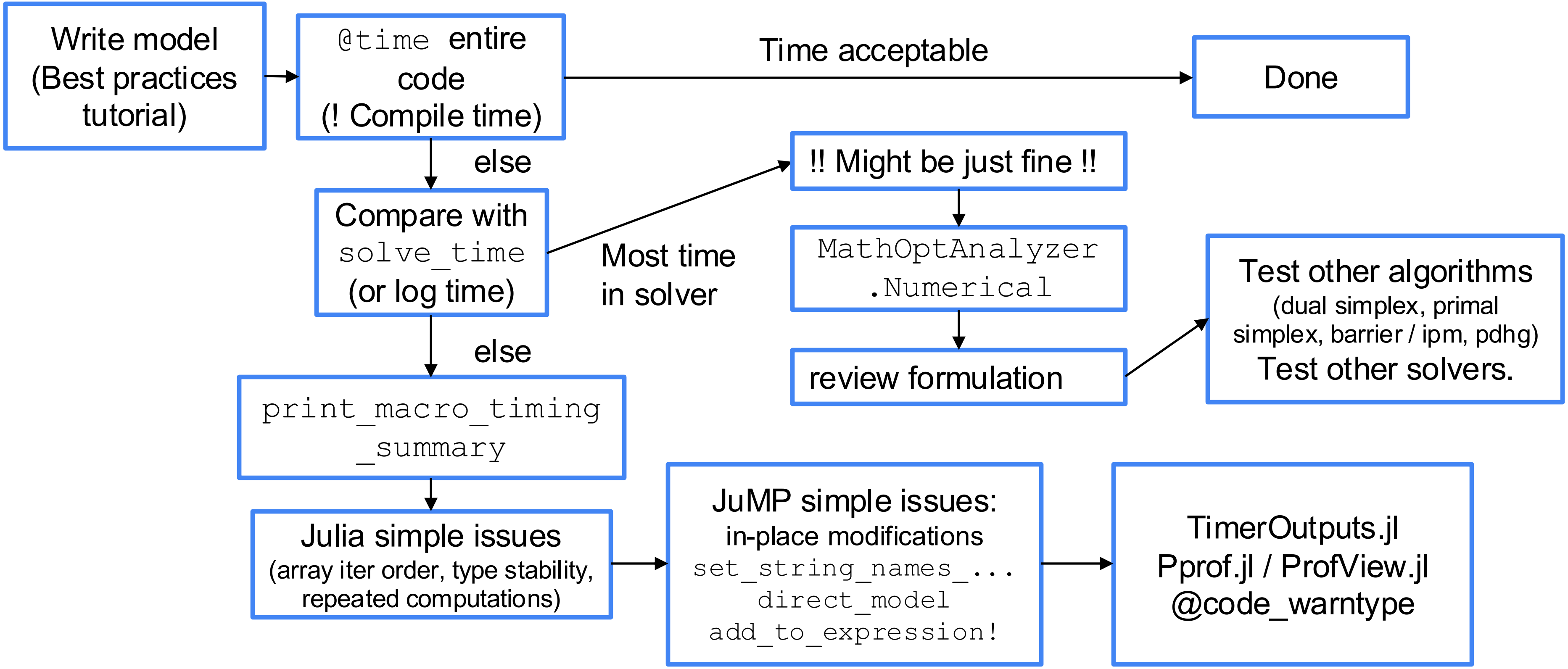
- Structure your code
  - If you your are doing something big: design patterns for larger models (above!)
  - Otherwise just JuMP-it

- Performance tips
  - Above!

- Also
  - Julia performance tips: docs.julialang.org/en/v1/manual/performance-tips/

Debugging performance

Premature optimization is the root of all evil

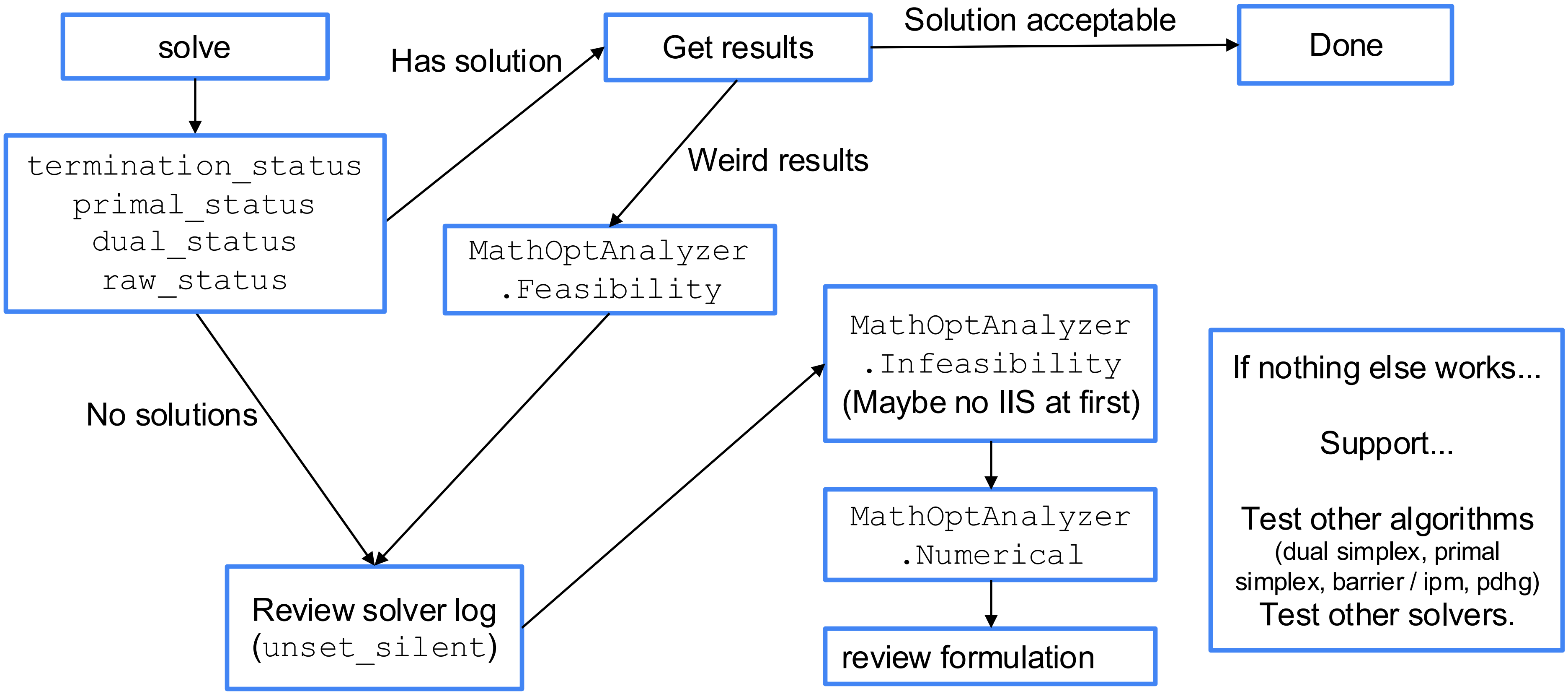


* There is also paralellism, see tutorial and be careful!

Debugging errors

- Tutorials are your best friends!
  - jump.dev/JuMP.jl/stable/tutorials/getting_started/**debugging**/
  - jump.dev/JuMP.jl/stable/tutorials/getting_started/**tolerances**/
- Structure your code
  - If you your are doing something big: design patterns for larger models
    - jump.dev/JuMP.jl/stable/tutorials/getting_started/**design_patterns_for_larger_models**/
- Test you code
  - CI and Code coverage are there for you

Debugging errors



Thanks!

And checkout:

1. JuMP tutorials
2. `]add MathOptAnalyzer`
3. jump-dev/open-energy-modeling-benchmarks

Joaquim Dias Garcia with help from:
Breakthrough Energy, Oscar Dowson, HiGHS Team (Julian Hall, Ivet Galabova, Mark Turner), José Daniel Lara (Sienna - NREL), Diego Tejada (Tulipa - TNO), Abel Siqueira (Tulipa - eScience NE), Luca Bonaldo (GenX - Princeton)