HydroPowerModels.jl

Andrew W. Rosenberg

\(^1\) Pontifical Catholic University of Rio de Janeiro
\(^\dagger\) Work supported by CAPES Foundation

March 18, 2019
Agenda 1

1. Introduction
   - Author
   - Overview

2. Dependencies and Integration
   - PowerModels.jl
   - Hydro
   - SDDP.jl

3. Example
   - Specification and Model DC
   - HydroPowerModels.jl Usage

4. Bibliography
Introduction: Andrew Rosemberg

- Control Engineering at Pontifical Catholic University of Rio de Janeiro (PUC-RIO), Brazil.
- Double Degree General Engineering at École centrale de Marseille, France.
- Currently enrolled in the Operations Research Masters at PUC-RIO (Electrical Department).
- Researcher at Laboratory of Applied Mathematical Programming and Statistics (LAMPS).
Overview

- **HydroPowerModels.jl** is a Julia/JuMP package for Hydrothermal Multistage Steady-State Power Network Optimization solved by Stochastic Dual Dynamic Programming (SDDP) [Pereira and Pinto, 1991].

- Problem Specifications and Network Formulations are handled by the **PowerModels.jl** package [Coffrin et al., 2018].

- Solution method is handled by the **SDDP.jl** package [Dowson and Kapelevich, 2017].
HydroPowerModels.jl

Overview

HydroPowerModels.jl

Network Description
Case.m

AC, DC, SOC, ...

Network Optimization Model
\[
\min_{x \in X} \quad f(x)
\]

s.t. \quad g(x) \in \mathbb{R}

PF, OPL, ...

JuMP

Optimization Model
\[
\min_{x \in S} \quad f(x)
\]

s.t. \quad g(x) \in \mathbb{R}

SDDP.jl

Hydro

Network Optimization Model
JuMP

\[
\min_{x \in X} \quad f(x)
\]

s.t. \quad g(x) \in \mathbb{R}

Add Hydro Constraints and Variables

Hydro-thermal Dispatch Opt Model
JuMP

\[
\min_{x \in S} \quad f(x)
\]

s.t. \quad g(x) \in \mathbb{R}

Hydro Description
hydro.json
inflows.csv
scenarioprobability.csv

Andrew W. Rosenberg
PowerModels.jl

- Steady-State Power Network Optimization Framework.
- Provides utilities for parsing and modifying network data.
- Designed to enable computational evaluation of emerging power network formulations and algorithms in a common platform.
- The code is engineered to decouple Problem Specifications (e.g. Power Flow, Optimal Power Flow, ...) from Network Formulations (e.g. AC, DC-approximation, SOC-relaxation, ...).
Dependencies and Integration

PowerModels.jl

Network Description

Case.m

AC, DC, SOC, ...

PF, OPf, ...

Network Optimization Model
JuMP

\[
\min_{x \in X} \quad f(x) \\
\text{s.t.} \quad g(x) \in \mathbb{R}
\]
Hydro Power Models.jl

Network Optimization Model

\[ \min_{x \in X} f(x) \]
\[ s.t. \quad g(x) \in \mathcal{X} \]

Add Hydro Constraints and Variables

Hydro-Thermal Dispatch Opt Model

\[ \min_{x \in X} f(x) \]
\[ s.t. \quad g(x) \in \mathcal{X} \]

Hydro Description

hydro.json
inflows.csv
scenarioprobablity.csv
SDDP.jl

- Open source.
- Generic (Not domain specific).

Why SDDP.jl (Oscar Dowson)

- Easy to use.
- Easy to extend.
- Many features.
Optimization Model

\[ \min_{x \in X} f(x) \]
\[ s.t. \quad g(x) \in \mathbb{N} \]
Optimization Model

\[
\begin{align*}
\min_{x \in X} & \quad f(x) \\
\text{s.t.} & \quad g(x) \in \mathbb{N}
\end{align*}
\]

\[
Q_t(x_{t-1}, \omega_t) = \min_{x_t \in \mathcal{X}(x_{t-1})} \quad f(x) + E[Q_{t+1}(x_t)] \\
\text{s.t.} & \quad g(x) \in \mathcal{N}(\omega_t)
\]
Example Case 3: Simplified Hydrothermal Dispatch

\[
\begin{align*}
\min_{g,f,\theta,h,u,s} & \quad c_d g_{d,t} + \sum_{i \in \text{Gen}} g_{i,t} c_i \\
\text{s.t.} & \quad \sum_{l \in T(b)} f_{l,t} - \sum_{l \in F(b)} f_{l,t} + \sum_{i \in \text{Gen}(b)} g_{i,t} - \sum_{i \in D(b)} d_{i,t} = 0 \quad \forall b \\
& \quad f_{l,t} = \frac{1}{x_l} (\theta_{\text{From}(l)} - \theta_{\text{To}(l)}) \quad \forall l \\
& \quad v_{h,t} + u_{h,t} + s_{h,t} = v_{h,t-1} + a_{h,t} \quad \forall h \\
& \quad u_{h,t} = g_{\text{HGen}(h)} \\
& \quad 0 \leq g \leq \bar{g}, \quad 0 \leq f \leq \bar{f}, \quad 0 \leq \theta \leq \bar{\theta}, \quad 0 \leq v \leq \bar{v}
\end{align*}
\]

Kirchhoff's laws

Hydro Balance

Bounds
Example Case 3: Simplified Hydrothermal Dispatch

\[ \min_{g,f,\theta,v,u,s} \quad c_d g_{d,t} + \sum_{i \in \text{Gen}} g_{i,t} c_i \]

\[ \sum_{l \in T(h)} f_{l,t} - \sum_{l \in F(h)} f_{l,t} + \sum_{i \in \text{Gen}(h)} g_{i,t} - \sum_{i \in D(h)} d_{i,t} = 0 \quad \forall b \]

Kirchhoff's laws

\[ f_{l,t} = \frac{1}{\xi_l} (\theta_{\text{Prom}(l)} - \theta_{\text{To}(l)}) \quad \forall l \]

Hydro Balance

\[ v_{h,t} + u_{h,t} + s_{h,t} = v_{h,t-1} + a_{h,t} \quad \forall h \]

\[ u_{h,t} = g_{\text{Gen}(h)} \]

\[ 0 \leq g \leq \bar{g}, 0 \leq f \leq \bar{f}, 0 \leq \theta \leq \bar{\theta}, 0 \leq v \leq \bar{v} \]

Bounds
Example Case 3: Simplified Hydrothermal Dispatch

\[
\begin{align*}
\min_{g,f,\theta,v,u,s} & \quad c_d g_{d,t} + \sum_{i \in \text{Gen}} g_{i,t} c_i \\
\text{s.t.} & \quad \sum_{l \in \text{T}(b)} f_{l,t} - \sum_{l \in \text{F}(b)} f_{l,t} + \sum_{i \in \text{Gen}(b)} g_{i,t} - \sum_{i \in \text{D}(b)} d_{i,t} = 0 \quad \forall b \\
& \quad f_{i,t} = \frac{1}{x_l} (\theta_{\text{From}(l)} - \theta_{\text{To}(l)}) \quad \forall l \\
& \quad v_{h,t} + u_{h,t} + s_{h,t} = v_{h,t-1} + a_{h,t} \quad \forall h \\
& \quad u_{h,t} = g_{H\text{Gen}(h)} \\
& \quad 0 \leq g \leq \bar{g}, 0 \leq f \leq \bar{f}, 0 \leq \theta \leq \bar{\theta}, 0 \leq v \leq \bar{v}
\end{align*}
\]

\{Kirchhoff's laws\}
\{Hydro Balance\}
\{Bounds\}
Example Case 3: Simplified Hydrothermal Dispatch

\[
\begin{align*}
\min_{g,f,\theta,v,u,s} & \quad c_d g_d,t + \sum_{i \in \text{Gen}} g_{i,t} c_i \\
\text{s.t.} & \quad \sum_{l \in \text{T(b)}} f_{l,t} - \sum_{l \in \text{F(b)}} f_{l,t} + \sum_{i \in \text{Gen(b)}} g_{i,t} - \sum_{i \in \text{D(b)}} d_{i,t} = 0 \quad \forall b \\
& \quad f_{l,t} = \frac{1}{x_l} (\theta_{\text{From}(l)} - \theta_{\text{To}(l)}) \quad \forall l \\
& \quad v_{h,t} + u_{h,t} + s_{h,t} = v_{h,t-1} + a_{h,t} \quad \forall h \\
& \quad u_{h,t} = g_{\text{HGen}(h)} \quad \forall h \\
& \quad 0 \leq g \leq \bar{g}, \; 0 \leq f \leq \bar{f}, \; 0 \leq \theta \leq \bar{\theta}, \; 0 \leq v \leq \bar{v}
\end{align*}
\]

Kirchhoff's laws

Hydro Balance

Bounds
HydroPowerModels.jl Usage

- HydroPowerModels.jl gives you an interface to easily implement the described model.
- As in PowerModels, once the case has been specified in the respective files (PowerModels.m, hydro.json, inflows.csv, scenarioprobability.csv) inside a case folder, the SDDP may be executed:

First import the necessary packages:

```julia
using HydroPowerModels
using Clp
```

Load Case by passing the respective folder:

```julia
data = HydroPowerModels.parse_folder("case3_folderpath")
```

```julia
Dict{Any,Any} with 2 entries:
"powersystem" => Dict{String,Any}(Pair{String,Any}("bus", Dict{String,Any}(Pa...  
"hydro" => Dict{String,Any}(Pair{String,Any}("scenario_probabilities", ..., ...
```
Set Parameters to run, for example, an DC Economic Hydrothermal Dispatch:

```julia
params = set_param(
    stages = 12,
    model_constructor_grid = DCPPowerModel,
    post_method = PowerModels.post_opf,
    solver = ClpSolver(),

Dict{Any,Any} with 5 entries:
"stages" => 12
"post_method" => PowerModels.post_opf
"solver" => Clp.ClpMathProgSolverInterface.ClpSolver(Any[])
"setting" => Dict("output"=>Dict("branch_flows"=>true))
"model_constructor_grid" => PowerModels.GenericPowerModel{PowerModels.DCPloss...
```

Build the Model and execute the SDDP method:

```julia
m = hydrothermaloperation(data, params)
status = solve(m, iteration_limit = 60)
```
Simulate 100 Instances of the problem:

```julia
results = simulate_model(m, 100)
```

```
Dict{Any,Any} with 5 entries:
  "simulations" => Dict{Any,Any}
  "data"      => Dict{Any,Any}
  "params"    => Dict{Any,Any}
  "machine"   => Dict{Any,Any}
  "solve_time" => 4.31247
```

Simulation results are found in the simulations array inside the dictionary.

```julia
results["simulations"][10]
```

```
Dict{Any,Any} with 6 entries:
  "obj"     => [11296.7, 10749.7, 9498.59, 8249.62, 7000.87, 6052.11, 5193.71...  
  "markov"  => [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  "objective" => 12852.6
  "solution" => Dict{String,Any}
  "stageobjective" => [1248.76, 1248.76, 1248.76, 1248.76, 1248.76, 858.398, 1196.13...  
  "noise"   => [2, 3, 2, 2, 2, 3, 2, 2, 3, 2, 3, 2, 1]
```
Results Case 3

Plotting results is easy! The function `plotresults()` receives a results dictionary and generates the most common plots for a hydrothermal dispatch:

```
plotresults(results)
```

**Figure:** Case 3 Results
Documentation and More Examples

- Detailed Documentation about installation, usage and testing of the package can be found at: Docs HydroPowerModels.jl

- Under Examples in the documentation there are a few Jupyter like cases and results to help discussions and learning.

Other Packages

- This is one of the many open-source projects develop by LAMPS: LAMPSPUC Github


# Model Definition

```julia
m = SDDPModel(
    sense = Min,
    stages = params["stages"],
    solver = params["solver"],
    objective_bound = 0.0
)
```

# build electric grid model using PowerModels

```julia
pm = PowerModels.build_generic_model(data["powersystem"], params["model_constructor_grid"],
    params["post_method"], jump_model=sp, setting = params["setting"])
```

# create reference to variables

```
createvarrefs(sp, pm)
```

# save GenericPowerModel

```
sp.ext[pm] = pm
```

# reservoir variables

```
variable_volume(sp, data)
```

# outflow and spillage variables

```
variable_outflow(sp, data)
variable_spillage(sp, data)
```

# hydro balance

```
variable_inflow(sp, data)
```

```
rainfall_noises(sp, data, cidx(t, data["hydro"]["size_inflow"][1]))
```

```
setnoise probabilty!(sp, data["hydro"]["scenario probabilities"])[cidx(t, data["hydro"]["size_inflow"][1]), :]
```

```
constraint_hydro_balance(sp, data)
```

# hydro generation

```
constraint_hydro_generation(sp, data, pm)
```

# Stage objective

```
@stageobjective(sp, sp.obj + sum(data["hydro"]["Hydrogenerators"][i]["spill_cost"]*sp["spill"][i] for i=1: data["hydro"]["nHyd"])
```