HydroPowerModels.jl

Andrew W. Rosemberg[1]

[1]Pontifical Catholic University of Rio de Janeiro
†Work supported by CAPES Foundation

March 18, 2019

Agenda I

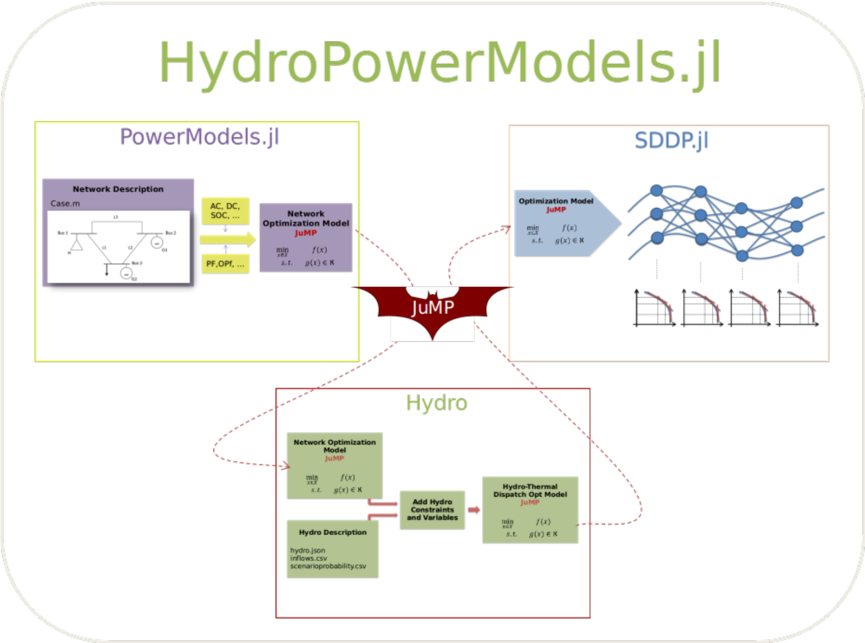
1. Introduction
   - Author
   - Overview
2. Dependencies and Integration
   - PowerModels.jl
   - Hydro
   - SDDP.jl
3. Example
   - Specification and Model DC
   - HydroPowerModels.jl Usage
4. Bibliography

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

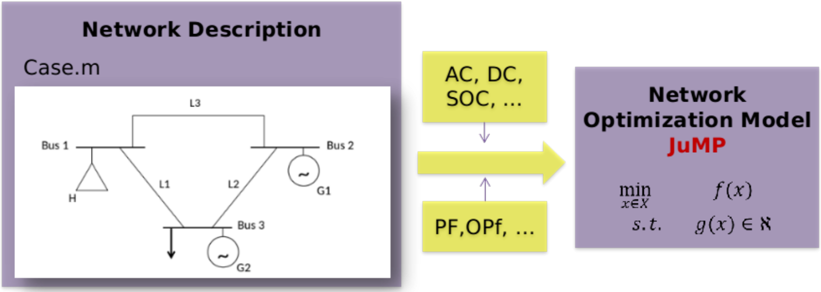
- **HydroPowerModels.jl** is a **Julia/JuMP** package for Hydrothermal Multistage Steady-State Power Network Optimization solved by Stochastic Dual Dynamic Programming (SDDP) [Pereira and Pinto, 1991].
- Problem Specifications and Network Formulations are handled by the **PowerModels.jl** package [Coffrin et al., 2018].
- Solution method is handled by the **SDDP.jl** package [Dowson and Kapelevich, 2017].

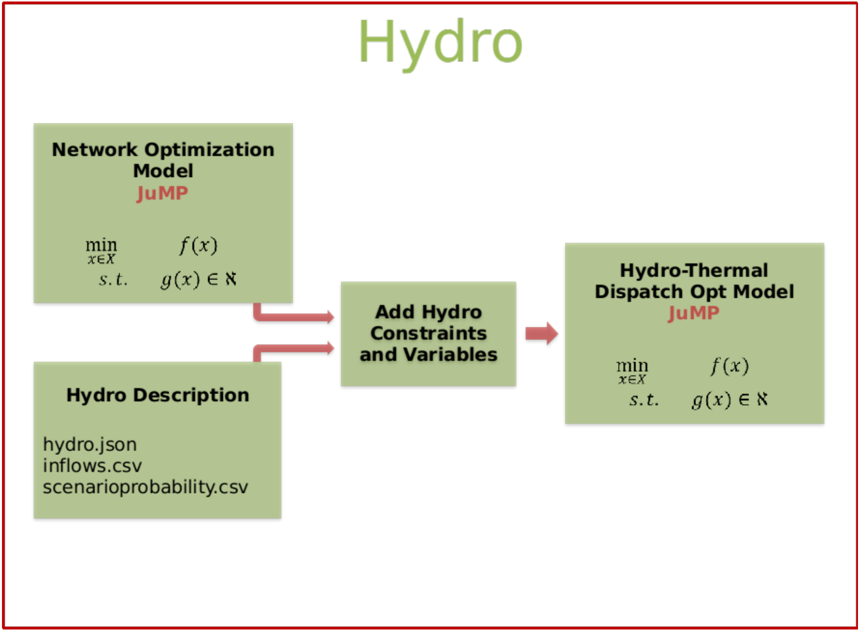

HydroPowerModels.jl
PowerModels.jl
Network Description
Case.m
AC, DC, SOC, ...
PF,OPf, ...
Network Optimization Model
JuMP
SDDP.jl
Optimization Model
JuMP
JuMP
Hydro
Network Optimization Model
JuMP
Add Hydro Constraints and Variables
Hydro-Thermal Dispatch Opt Model
JuMP
Hydro Description
hydro.json
inflows.csv
scenarioprobability.csv

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, ...).

PowerModels.jl



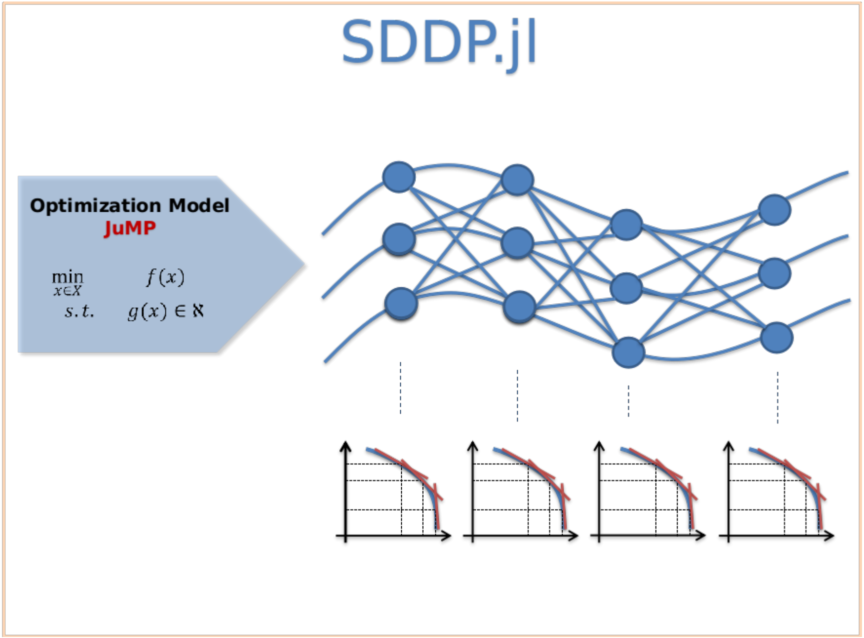

Hydro
Network Optimization Model
JuMP
$\min_{x \in X} \quad f(x)$
$s.t. \quad g(x) \in \aleph$
Hydro Description
hydro.json
inflows.csv
scenarioprobability.csv
Add Hydro Constraints and Variables
Hydro-Thermal Dispatch Opt Model
JuMP
$\min_{x \in X} \quad f(x)$
$s.t. \quad g(x) \in \aleph$

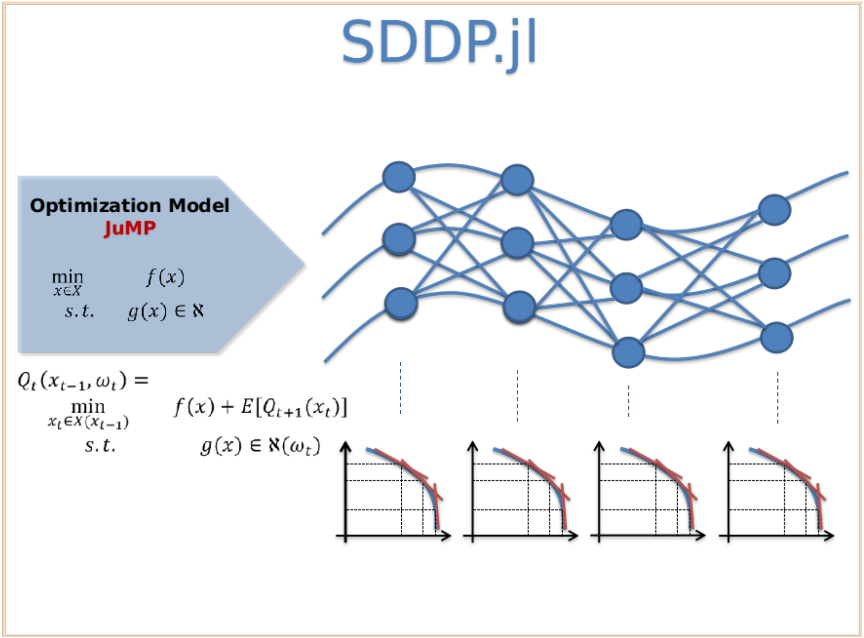
SDDP.jl

- Julia/JuMP Package for solving large multistage convex stochastic optimization problems using Stochastic Dual Dynamic Programming.
- Open source.
- Generic (Not domain specific).

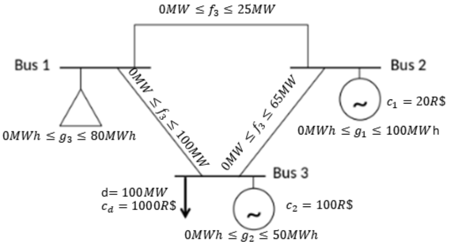
Why SDDP.jl (Oscar Dowson)

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


SDDP.jl
Optimization Model
JuMP
$\min_{x \in X} \quad f(x)$
$s.t. \quad g(x) \in \aleph$


SDDP.jl
Optimization Model
JuMP
$\min_{x \in X} \quad f(x)$
$s.t. \quad g(x) \in \aleph$
$Q_t(x_{t-1}, \omega_t) =$
$\min_{x_t \in X(x_{t-1})} \quad f(x) + E[Q_{t+1}(x_t)]$
$s.t. \quad g(x) \in \aleph(\omega_t)$

Example Case 3: Simplified Hydrothermal Dispatch



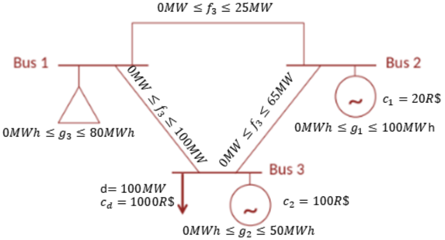
$$\min_{g,f,\theta,v,u,s} \quad c_d g_{d,t} + \sum_{i\in \mathrm{Gen}} g_{i,t}\, c_i$$

$$\text{s.t.} \quad \left.\begin{array}{l} \displaystyle\sum_{l\in T(b)} f_{l,t} - \sum_{l\in F(b)} f_{l,t} + \sum_{i\in \mathrm{Gen}(b)} g_{i,t} - \sum_{i\in D(b)} d_{i,t} = 0 \quad \forall b \\ f_{l,t} = \dfrac{1}{x_l}\left(\theta_{\mathrm{From}(l)} - \theta_{\mathrm{To}(l)}\right) \quad \forall l \end{array}\right\} \text{Kirchhoff's laws}$$

$$\left.\begin{array}{l} v_{h,t} + u_{h,t} + s_{h,t} = v_{h,t-1} + a_{h,t} \\ u_{h,t} = g_{\mathrm{HGen}(h)} \end{array} \quad \forall h \right\} \text{Hydro Balance}$$

$$0 \leq g \leq \bar{g}, 0 \leq f \leq \bar{f}, 0 \leq \theta \leq \bar{\theta}, 0 \leq v \leq \bar{v} \quad \}\text{Bounds}$$

Example Case 3: Simplified Hydrothermal Dispatch



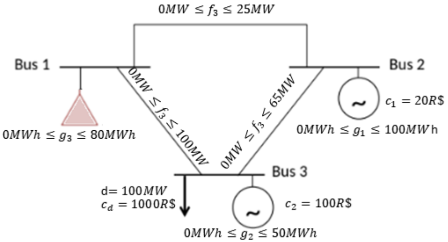
$$\min_{g,f,\theta,v,u,s} \quad c_d g_{d,t} + \sum_{i \in Gen} g_{i,t} c_i$$

$$\text{s.t.} \quad \left.\begin{array}{l} \sum_{l \in T(b)} f_{l,t} - \sum_{l \in F(b)} f_{l,t} + \sum_{i \in Gen(b)} g_{i,t} - \sum_{i \in D(b)} d_{i,t} = 0 \quad \forall b \\ f_{l,t} = \frac{1}{x_l}(\theta_{From(l)} - \theta_{To(l)}) \quad \forall l \end{array}\right\} \text{Kirchhoff's laws}$$

$$\left.\begin{array}{l} v_{h,t} + u_{h,t} + s_{h,t} = v_{h,t-1} + a_{h,t} \\ u_{h,t} = g_{HGen(h)} \end{array} \quad \forall h \right\} \text{Hydro Balance}$$

$$0 \le g \le \bar{g}, 0 \le f \le \bar{f}, 0 \le \theta \le \bar{\theta}, 0 \le v \le \bar{v} \quad \}\text{Bounds}$$

Example Case 3: Simplified Hydrothermal Dispatch



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

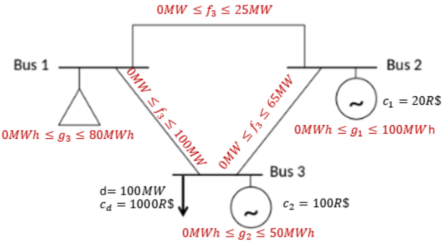
s.t.

$$\left.\begin{array}{l} \sum_{l\in T(b)} f_{l,t} - \sum_{l\in F(b)} f_{l,t} + \sum_{i\in Gen(b)} g_{i,t} - \sum_{i\in D(b)} d_{i,t} = 0 \quad \forall b \\ f_{l,t} = \frac{1}{x_l}(\theta_{From(l)} - \theta_{To(l)}) \quad \forall l \end{array}\right\} \text{Kirchhoff's laws}$$

$$\left.\begin{array}{l} v_{h,t} + u_{h,t} + s_{h,t} = v_{h,t-1} + a_{h,t} \\ u_{h,t} = g_{HGen(h)} \end{array} \quad \forall h \right\} \text{Hydro Balance}$$

$$0 \le g \le \bar{g}, 0 \le f \le \bar{f}, 0 \le \theta \le \bar{\theta}, 0 \le v \le \bar{v} \quad \}\text{Bounds}$$

Example Case 3: Simplified Hydrothermal Dispatch



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

s.t.

$$\left.\begin{array}{l} \sum_{l \in T(b)} f_{l,t} - \sum_{l \in F(b)} f_{l,t} + \sum_{i \in Gen(b)} g_{i,t} - \sum_{i \in D(b)} d_{i,t} = 0 \quad \forall b \\ f_{l,t} = \frac{1}{x_l}(\theta_{From(l)} - \theta_{To(l)}) \quad \forall l \end{array}\right\} \text{Kirchhoff's laws}$$

$$\left.\begin{array}{l} v_{h,t} + u_{h,t} + s_{h,t} = v_{h,t-1} + a_{h,t} \\ u_{h,t} = g_{HGen(h)} \end{array} \quad \forall h \right\} \text{Hydro Balance}$$

$$0 \le g \le \bar{g}, 0 \le f \le \bar{f}, 0 \le \theta \le \bar{\theta}, 0 \le v \le \bar{v} \quad \} \text{Bounds}$$

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:

```
using HydroPowerModels
using Clp
```

Load Case by passing the respective folder:

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

```
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:

```
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:

```
m = hydrothermaloperation(data, params)

status = solve(m, iteration_limit = 60)
```

- Simulate 100 Instances of the problem:

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

```
Dict{Any,Any} with 5 entries:
"simulations" => Dict[Dict{Any,Any}(Pair{Any,Any}("obj", [10496.1, 10500. 8, 9...
"data"        => Dict{Any,Any}[Dict{Any,Any}(Pair{Any,Any}("powersystem", Dic...
"params"      => Dict{Any,Any}(Pair{Any,Any}("stages", 12),Pair{Any,Any}("pos..."
"machine"     => Dict("cpu"=>"Intel(R) Xeon(R) CPU @ 2.30GHz","memory"=>"7.30..."
"solve_time"  => 4.31247
```

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

```
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[Dict{String,Any}(Pair{String,Any}("baseMVA", 100),Pair{St...
"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, 3, 2, 1]
```

Results Case 3

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

```
plotresults(results)
```

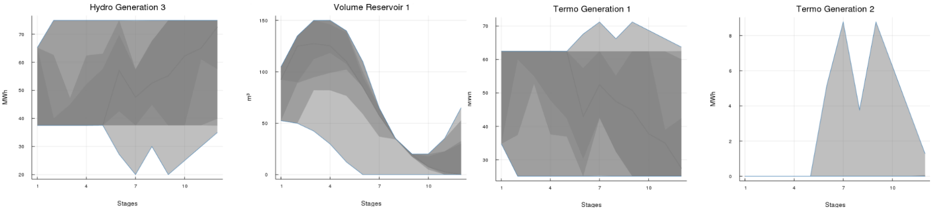


Figure: Case 3 Results

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**

Bibliography

Mario VF Pereira and Leontina MVG Pinto. Multi-stage stochastic optimization applied to energy planning. *Mathematical programming*, 52(1-3):359–375, 1991.

Carleton Coffrin, Russell Bent, Kaarthik Sundar, Yeesian Ng, and Miles Lubin. Powermodels.jl: An open-source framework for exploring power flow formulations. In *2018 Power Systems Computation Conference (PSCC)*, pages 1–8, June 2018. doi: 10.23919/PSCC.2018.8442948.

Oscar Dowson and Lea Kapelevich. SDDP.jl: a Julia package for Stochastic Dual Dynamic Programming. *Optimization Online*, 2017. URL `http://www.optimization-online.org/DB_HTML/2017/12/6388.html`.

SDDP.jl

PowerModels.jl

HydroPowerModels.jl

```
# Model Definition
m = SDDPModel(
            sense   = :Min,
            stages  = params["stages"],
            solver  = params["solver"],
    objective_bound = 0.0
                                ) do sp,t

    # build eletric grid model using PowerModels
    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

    # resevoir 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]))
    setnoiseprobability!(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"]))
end
```