

# JADE Documentation

A Julia DOASA Environment

Anthony Downward, Andy Philpott



Electric Power Optimization Centre,  
University of Auckland.

# Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Introduction</b>                              | <b>1</b> |
| 1.1      | Overview . . . . .                               | 2        |
| 1.2      | JADE stage problem . . . . .                     | 2        |
| <b>2</b> | <b>Details of JADE implementation</b>            | <b>4</b> |
| 2.1      | Software environment . . . . .                   | 4        |
| 2.2      | Setting up the JADE package . . . . .            | 5        |
| 2.2.1    | Installing JADE . . . . .                        | 5        |
| 2.2.2    | Testing JADE . . . . .                           | 5        |
| 2.2.3    | JADE API Documentation . . . . .                 | 5        |
| 2.3      | Mathematical modelling in JADE . . . . .         | 6        |
| 2.3.1    | Water network . . . . .                          | 6        |
| 2.3.2    | Power transmission network . . . . .             | 7        |
| 2.3.3    | Contingent storage . . . . .                     | 7        |
| 2.3.4    | Decision rules . . . . .                         | 8        |
| 2.3.5    | Stage objective . . . . .                        | 8        |
| 2.3.6    | Dependent Inflow Adjustment . . . . .            | 8        |
| <b>3</b> | <b>JADE package specifications</b>               | <b>8</b> |
| 3.1      | Data structures . . . . .                        | 9        |
| 3.2      | Input files . . . . .                            | 9        |
| 3.3      | Training and Simulating JADE models . . . . .    | 17       |
| 3.3.1    | Initialising and training JADE a model . . . . . | 17       |
| 3.3.2    | Simulating a JADE policy . . . . .               | 18       |
| 3.3.3    | Cuts . . . . .                                   | 19       |
| 3.4      | Output . . . . .                                 | 20       |
| 3.4.1    | Policy Generation Output . . . . .               | 20       |

# 1 Introduction

This document describes a new implementation of the Dynamic Outer Approximation Sampling Algorithm (DOASA), named JADE. DOASA is an implementation of the SDDP algorithm for hydro-thermal scheduling. JADE has been developed in the programming language Julia; previous implementations of DOASA have been developed in C++ and AMPL.

## 1.1 Overview

Dynamic Outer Approximation Sampling Algorithm (DOASA) was developed for optimization in hydro-thermal scheduling and water valuation, in the New Zealand electricity system. DOASA has previously been implemented in AMPL, and later in C++. JADE is a flexible version of DOASA, developed in the scripting language Julia. The purpose of JADE is to provide a version of DOASA that is flexible enough to allow modelling changes to be made easily, but without compromising computational speed. The hydro-scheduling problem that is modelled in JADE is based entirely on the EMI-DOASA model developed by Stochastic Optimization Ltd, as described in [7].

The purpose of this document is to give an overview of how JADE can be used and how it was developed. The optimization problem in JADE is briefly outlined in the next section. Section 2 describes how JADE model was implemented in Julia. Section 3 describes package specifications and the inputs and outputs of JADE.

## 1.2 JADE stage problem

The optimization model in JADE has been designed to be equivalent to the EMI-DOASA model. The description of the medium-term hydro-scheduling model, as given in [7] is repeated here. Differences between this mathematical formulation and the computational model in JADE are highlighted in section 2.3.

The DOASA model seeks a policy of electricity generation that meets demand and minimizes the expected fuel cost of thermal generation. All data are deterministic except for weekly inflows that are assumed to be stagewise independent. This results in a large-scale stochastic dynamic programming model which is defined as follows. Let  $x_j(t)$  denote the storage in reservoir  $j$  at the *end* of week  $t$ , and let the Bellman function  $C_t(\bar{x}, \omega(t))$  be the minimum expected fuel cost to meet electricity demand in weeks  $t, t+1, \dots$ , when reservoir storage  $x_j(t-1)$  at the start of week  $t$  is equal to  $\bar{x}_j$  and the inflow to reservoir  $j$  in week  $t$  is known to be  $\omega_j(t)$ . We assume  $\mathbb{E}_{\omega(T+1)} C_{T+1}(\bar{x}, \omega(T+1))$  is a known function of  $\bar{x}$  that defines the expected future cost at the end of stage  $T$  when  $x(T) = \bar{x}$ . Then the Bellman function  $C_t(\bar{x}, \omega(t))$  is the optimal solution value of the mathematical program:

$$\begin{aligned}
P_t(\bar{x}, \omega(t)): \quad & \min \quad \sum_{i \in \mathcal{N}} \sum_{m \in \mathcal{F}(i)} \phi_m \sum_b T(b, t) f_m(b, t) + \mathbb{E}[C_{t+1}(x(t), \omega(t+1))] \\
\text{s.t.} \quad & g_i(y(b, t)) + \sum_{m \in \mathcal{F}(i)} f_m(b, t) + \sum_{m \in \mathcal{H}(i)} \gamma_m h_m(b, t) = D_i(b, t), \quad i \in \mathcal{N}, \\
& x(t) = \bar{x} - S \sum_b T(b, t) (A h(b, t) + A s(b, t) - \omega(t)), \\
& 0 \leq f_m(t) \leq a_m, \quad m \in \mathcal{F}(i), \quad i \in \mathcal{N}, \\
& 0 \leq h_m(t) \leq b_m, \quad 0 \leq s_m(t) \leq c_m, \quad m \in \mathcal{H}(i), \\
& 0 \leq x_j(t) \leq r_j, \quad j \in \mathcal{J}, \quad i \in \mathcal{N}, \quad y \in Y.
\end{aligned}$$

This description uses the following indices:

| Index            | Refers to                            |
|------------------|--------------------------------------|
| $t$              | index of week                        |
| $i$              | node in transmission network         |
| $b$              | index of load block                  |
| $m$              | index of plant                       |
| $j$              | index of reservoir                   |
| $\mathcal{N}$    | set of nodes in transmission network |
| $\mathcal{F}(i)$ | set of thermal plants at node $i$    |
| $\mathcal{H}(i)$ | set of hydro plants at node $i$      |
| $\mathcal{J}$    | set of reservoirs.                   |

The parameters are:

| Symbol      | Meaning  | Units              |
|-------------|--|--------------------|
| $\phi_m$    | short-run marginal cost of thermal plant $m$           | \$/MWh             |
| $\gamma_m$  | conversion factor for water flow into energy           | MWs/m <sup>3</sup> |
| $D_i(b, t)$ | electricity demand in node $i$ in block $b$ , week $t$ | MW                 |
| $T(b, t)$   | number of hours in load block $b$ in week $t$          | h                  |
| $S$         | number of seconds per hour (3600)                      |                    |
| $a_m$       | thermal plant capacity                                 | MW                 |
| $b_m$       | hydro plant capacity                                   | m <sup>3</sup> /s  |
| $c_m$       | spillway capacity                                      | m <sup>3</sup> /s  |
| $r_j$       | reservoir capacity                                     | m <sup>3</sup>     |
| $Y$         | feasible set of transmission flows                     |                    |
| $A$         | incidence matrix of river chain                        |                    |

The variables are:

| Symbol      | Meaning   | Units             |
|-------------|---|-------------------|
| $f_m(b, t)$ | generation of thermal plant $m$ in load block $b$ in week $t$ | MW                |
| $x_j(t)$    | storage in reservoir $j$ at end of week $t$                   | m <sup>3</sup>    |
| $\bar{x}_j$ | known storage in reservoir $j$ at start of week $t$           | m <sup>3</sup>    |
| $h(b, t)$   | vector of hydro releases in block $b$ , week $t$              | m <sup>3</sup> /s |
| $s(b, t)$   | vector of hydro spills in block $b$ , week $t$                | m <sup>3</sup> /s |
| $\omega(t)$ | inflow (assumed constant over the week)                       | m <sup>3</sup> /s |
| $y(b, t)$   | flow in transmission lines in load block $b$ in week $t$      | MW                |
| $g_i(y)$    | sum of flow into node $i$ when transmission flows are $y$     | MW.               |

Here the water-balance constraints in the storage reservoirs at the end of week  $t$  are represented by

$$x(t) = \bar{x} - S \sum_b T(b, t) (A h(b, t) + A s(b, t) - \omega(t))$$

where  $x_j(t)$  is the storage in reservoir  $j$  at the end of week  $t$ ,  $s_j(b, t)$  denotes the rate of spill (in  $\text{m}^3/\text{second}$ ) in load block  $b$  in week  $t$ , and  $\omega_j(t)$  is the uncontrolled rate of inflow into reservoir  $j$  in week  $t$ . We multiply all of these by  $S$  to convert to  $\text{m}^3/\text{hour}$ , and then by  $T(b, t)$  to give  $\text{m}^3$  in each load block. All these are subject to capacity constraints. (In some cases we also have minimum flow constraints that are imposed by environmental resource consents.) The parameter  $\gamma_m$ , which varies by generating station  $m$ , converts flows of water  $h_m(t)$  into electric power.

The node-arc incidence matrix  $A$  represents the river-valley network, and aggregates controlled flows that leave a reservoir by spilling or generating electricity and subtracts those that enter a reservoir from upstream. In other words row  $j$  of  $Ah(b, t) + As(b, t)$  gives the total controlled flow out of the reservoir (or river junction) represented by row  $j$ , this being the release and spill of reservoir  $j$  minus the sum of any immediately upstream releases and spill.

## 2 Details of JADE implementation

This section details the specifications for the JADE package including installation instructions.

### 2.1 Software environment

JADE 1.0.0 has been tested using Julia version 1.7.3.

An environment for implementing the SDDP algorithm in Julia was developed in a separate research project. This environment is a Julia package named SDDP.jl<sup>1</sup>. The New Zealand hydro-scheduling stage problem and the data / modelling interface is defined inside JADE, but the SDDP algorithm is carried out using the functionality within the SDDP.jl package.

JuMP is a mathematical modelling language embedded in Julia. The DOASA stage problem has been written for SDDP.jl, utilising the JuMP modelling language [2].

Several registered Julia packages are required to run JADE. These are:

|                  |               |                |
|------------------|---------------|----------------|
| DataFrames       | JuMP          | SDDP           |
| MathOptInterface | Random        | DelimitedFiles |
| Statistics       | LinearAlgebra | JSON           |

Registered packages will typically be automatically installed along with JADE, however they can also be individually installed using:

---

```
] add "<Package name>"
```

---



---

<sup>1</sup>SDDP.jl was developed by Dr. Oscar Dowson at the University of Auckland, see [3, 4].

## 2.2 Setting up the JADE package

In this section we detail how to set up JADE and test that it is working correctly on your system.

### 2.2.1 Installing JADE

In order to run JADE, the user requires:

- The Julia environment; this can be downloaded from <http://julialang.org/downloads>.
- An optimizer (solver); we recommend CPLEX or Gurobi, although GLPK also works for small problems.
- The registered Julia packages mentioned above.
- Input data in the format described in the next section.

**Installation:** JADE can be installed as a Julia package by opening a Julia session and running:

---

```
] add "https://github.com/EPOC-NZ/JADE.git"
```

---

Julia packages are typically stored in a directory such as:

---

```
C:/Users/<user>/.julia/packages/
```

---

Using the JADE package, we can create a model object that is defined by SDDP.jl. So, a user can use JADE to build the DOASA model, then use commands defined in SDDP.jl to solve the model with various options. The readme.md file in JADE and files in the JADE/examples folder show how to use functions from the JADE module.

### 2.2.2 Testing JADE

Output produced by JADE was tested by comparing policy generation results between JADE and EMI-DOASA. Automated tests are stored in the file `test/runtests.jl`. After installing JADE as a package, a user can run tests with:

---

```
] test JADE
```

---

Test cases include both deterministic and stochastic problems, and unusual scenarios (for example, where load shedding is required). The tests use the GLPK optimizer, so you will see some warnings as the tests are completed.

If JADE is modified, `runtests.jl` needs to be checked and updated.

### 2.2.3 JADE API Documentation

Full documentation of the functions available in JADE can be found here:

<https://epoc-nz.github.io/JADE/>

## 2.3 Mathematical modelling in JADE

The core mathematical model in JADE represents a DC load-flow model of the electricity system, with piecewise-linear losses, along with a hydrological model of rivers and lakes. The implementations for these are defined using the JuMP modelling framework in Julia, but the topography and parameters of the networks are defined by the user (this is explained in detail in section 3.2).

The JADE stage problem is defined in `model.jl`. This file contains the definition of the function `JADEmodel()`, which is used to construct an SDDP model.

In every stage problem, we compute the current year and week. The week and year are calculated from the stage number and the problem start time, so a fixed number of weeks per year is assumed (set to be 52). It is assumed that inflow data exists for this fixed number of weeks per year.

In the basic version of JADE, the state variables are the volume of water stored at the end of each week, in all reservoirs defined in `reservoirs.csv`.

We will now highlight some of the features in the JADE model that are not fully captured by the mathematical representation of the model in Section 1.2.

### 2.3.1 Water network

The specific power value of a reservoir is the sum of the specific power values of all hydro stations downstream. To calculate this, the hydro stations which are downstream from every reservoir are derived.

The water flow network can be thought of as a graph where vertices are reservoirs and junctions. For every hydro station, we construct one *spill arc*, and one *release arc*. The amount of electricity dispatched by hydro stations is set to the flow on the release arc times the specific power of that station.

The file `network.jl` contains the definition of the function used to traverse a flow network. A depth-first search is carried out from every reservoir to get a list of hydro stations beneath it.

**Hydro flows** The JADE model defines flow conservation constraints for every reservoir and junction. As mentioned, these are all treated as vertices in a flow network. The flows in the network are stage problem variables that are defined over all existing arcs and also load blocks. There are three types flow conservation constraints:

- conservation for reservoirs,
- conservation for junctions with inflow data,
- conservation for junctions with no inflow.

All flow conservation constraints take into account the flow in all arcs entering and leaving a vertex. If the vertex is a junction with inflow, then the inflow is also taken into account. If the vertex is a reservoir, then the inflow as well as the initial and final storage values are taken into account. Also, if the vertex is a reservoir, then the flows going into the reservoir and leaving the reservoir at different load blocks are aggregated in the flow conservation load constraint. For all other vertices, the flow going in must equal the flow going out at every load block.

**Inflows** In the basic version of JADE, inflows are stage-wise independent between scenarios and at any stage, all possible inflow scenarios have an equal probability of being realised. The scenarios considered are all the inflows between the sample start year and sample end year, inclusive.

In order for JADE to be compatible with EMI-DOASA, you must set `first_week_known = true` in `define_JADE_model`. With this setting, all inflows are taken as the inflow value observed in the corresponding week of the problem year, otherwise inflows are sampled randomly every week during training and simulation.

**Flow bounds** Flow can exceed the upper or lower bounds of an arc (without becoming negative), but there is an associated penalty cost. If  $z(b, t)$  is the amount in cumecs by which a flow is above or below some bound during block  $b$ , ( $z(b, t) \geq 0$ ), then the cost associated with this violation is calculated as:

$$z(b, t) \times T(b, t) \times SP_{\max} \times 3600 \times \text{penalty}.$$

The value of *penalty* is specified by the user in `run.csv`. Here  $SP_{\max}$  is the value of the maximum specific power of all reservoirs, in MWh / m<sup>3</sup>.

### 2.3.2 Power transmission network

JADE will formulate a model of the hydro scheduling problem with any radial transmission network defined in `transmission.csv`. Each node in JADE has a name that is used to specify locations of generators, demand, and demand-response in the appropriate file. It is possible to define a line for any pair of nodes, by default a symmetric line will be defined with identical properties in both directions.

**Losses** JADE supports piecewise-linear transmission losses. These are defined through loss tranches, as shown in `transmission.csv` in Section 3.2. The losses on a line are approximately applied on the nominal flow on the line. Mathematically, this is modelled by adding half of the losses to the load at each of the two nodes connected by the line.

If the nominal flow on the line is  $f$  and the losses on the line are  $l$ , then  $f + \frac{l}{2}$  flow leaves the sending node and  $f - \frac{l}{2}$  arrives at the receiving node.

### 2.3.3 Contingent storage

Storage in JADE is defined such that a storage level of 0 means that reservoir is at the bottom of its typical operating range. JADE supports the use of contingent storage, which is modelled as negative storage levels. For every week that the storage is negative, there is a user-defined penalty that is charged. This is defined through a set of tranches of increasing cost, based on the reservoir level. Note that it is important to understand that this penalty is applied to the level of the reservoir at the end of the week, not the amount of water used in the week.

One should be aware that due to the penalty being applied based on the water level (each week), it is possible that the marginal value of water to be significantly larger than the penalty, if the model anticipates that the contingent storage will be utilised for several weeks.



### 2.3.4 Decision rules

In JADE, generation and spill from hydro-stations can be restricted using decision rules. These are defined by linear constraints (typically setting upper or lower bounds) on generation and/or spill, based on the current storage in the corresponding reservoir. Suppose for station  $m$  we have a set of such decision rules  $d \in \mathcal{D}_m$ ; a decision rule  $d$  applies one of the following constraints:

$$\begin{aligned}\gamma_m \sum_b T(b, t)(h_m(b, t) + s_m(b, t)) &\leq \alpha_d x_j(t) + \beta_d, \\ \gamma_m \sum_b T(b, t)(h_m(b, t) + s_m(b, t)) &= \alpha_d x_j(t) + \beta_d, \\ \gamma_m \sum_b T(b, t)(h_m(b, t) + s_m(b, t)) &\geq \alpha_d x_j(t) + \beta_d,\end{aligned}$$

where  $\alpha_d$  and  $\beta_d$  define the slope and intercept of the linear constraint for decision rule  $d$ , and  $j$  is the appropriate reservoir.

In the JADE code, a vector of decision rules must be provided when the model is defined, since these form constraints that are included in the stage problems. For example the following decision rule sets an upper bound for the weekly hydro release for generation through the Manapouri station, as a function of the storage in the reservoir representing Lakes Manapouri and Te Anau. The decision rule is applied in calendar weeks 30–40 inclusive.

---

```
decision_rules = [ JADE.DecisionRule(97.54, 6180.97, :upper, :MANAPOURI,
                                     :LAKES_MANAPOURI_TE_ANAU, :GENERATION, 30:40) ]
```

---

In this case,  $\alpha_d = 97.54$  MWh / Mm<sup>3</sup> and  $\beta_d = 6180.97$  MWh.

### 2.3.5 Stage objective

At every stage, the objective is computed as:

$$\begin{aligned}\text{objective} = & \text{fuel costs} + \text{emissions costs} + \text{hydro variable O\&M costs} + \\ & \text{demand-response costs} + \text{contingent storage penalties} + \\ & \text{flow penalties} + \text{future cost}.\end{aligned}$$

The last stage's future cost in a finite-horizon model is set to the terminal cost, whereas in an steady-state model the model links back to stage 1, with a discount.

### 2.3.6 Dependent Inflow Adjustment

Inflows are assumed to be stagewise independent in DOASA. Real inflows are not independent between weeks. *Dependent Inflow Adjustment* (DIA) is a technique that modifies the variation in inflow sequences, so that the variance in the amount of accumulated inflow over some fixed window of weeks will be increased in comparison with the variance arising from independent inflows. This is intended to have the effect of improving the approximation of total inflow over a user-defined window of weeks. The DIA implementation in JADE follows the description of the technique in [7]. If the user selects an input correlation length of 1 or less, inflows will remain unadjusted.

Note that if we have set `first_week_known = true`, then the historical inflow of that week is always realised during training (and simulation), even if we adjust inflows using DIA.

## 3 JADE package specifications

In this section we will detail the input file specifications for JADE, and explain how these are read into the JADE data structures.

### 3.1 Data structures

In order to create and run a complete JADE model and simulation, we need to define several JADE objects. These are:

- **RunData**: this contains the settings required in order to create the SDDP model;
- **JADEData**: this contains all the data for the SDDP model;
- **JADEModel**: this contains the SDDP model and the associated data;
- **JADESolveOptions**: this contains the solve options that are used when training the model;
- **JADESimulation**: this contains the settings that define the simulation.

JADE can use the same input data files as EMI-DOASA. These are csv files, contained in the directory `Input/<data-dir>`. The contents of input files are explained here.

### 3.2 Input files

The file `data.jl` contains the definition of the **JADEData** object, and functions used for constructing a **JADEData** object. A **JADEData** object holds all the parameters required in the JADE stage problem.

JADE is written using the JuMP package, which defines variable indices using definitions of sets. To input set names, JADE defines these as row and/or column headers in its input files. JADE Table 3.1 lists the input files from which all sets in JADE are derived.

Table 3.1: Input files for JADE sets

| Set               | File                       |
|-------------------|----------------------------|
| Reservoirs        | reservoirs.csv             |
| Junctions         | hydro_junctions.csv        |
| Hydro stations    | hydro_stations.csv         |
| Station arcs      | hydro_stations.csv         |
| Natural arcs      | hydro_arcs.csv             |
| Thermal stations  | thermal_stations.csv       |
| Load blocks       | demand.csv (column names)  |
| Demand nodes      | demand.csv                 |
| Inflow locations  | inflows.csv (column names) |
| Transmission arcs | transmission.csv           |

Letter case is discarded in names of reservoirs, hydro stations, hydro junctions, and thermal stations. For example `Stratford_220kV` = `Stratford_220KV` = `STRATFORD_220KV`.

The column names in `inflows.csv` are used to define a set of inflow locations. It is required that all reservoirs are in this set. It is also assumed that all data are in the format described in this section and all time-related data are in chronological order. The sets for all attributes of

the model are inferred from the data files. Table 3.1 outlines which input file is used to define each set of attributes.

## run.csv

A key file in a DOASA input directory is `run.csv`. This file contains all the parameters for the current run of the model.

JADE comes with an Excel workbook that can be used to create run files for JADE. This workbook is called `runJADE.xlsx` in the directory `runJADE`. (You will need to enable macros when the workbook loads.)

In order to use this workbook, you must first download JADE input files from the EA’s EMI website: <https://www.emi.ea.govt.nz/Wholesale/Tools/Jade>. Once these have been placed into an input directory, you can either use the included `run.csv` or create your own using this workbook as follows.

The first step is to click the “Find JADE Input Files” button and select the `demand.csv` file within the input directory for which you are creating the run file. Based on data files in the corresponding directory, the workbook will enable and restrict certain choices.

The user then chooses options for the model, training, and simulation. These can then be saved as a run file, by clicking the “Create JADE Run File” button. The user interface of this workbook is shown below.

### Model Settings


|                              |             |
|------------------------------|-------------|
| Policy name                  | policy1     |
| Scenario directory           |             |
| Problem start year           |             |
| Problem start week           | 1           |
| Number of weeks              | 52          |
| Steady state discount factor | 0.92 (8.7%) |
| Sample start year            |             |
| Sample end year              |             |
| Inflow correlation length    | 3           |
| LB flow penalty              | 500         |
| UB flow penalty              | 50          |
| Scale objective              | 1.00E+06    |
| Scale reservoirs             | 1           |

### Training Options

|                      |       |
|----------------------|-------|
| Iterations           | 5000  |
| Load saved cuts from |       |
| Write EOH cuts file  | FALSE |
| Load EOH cuts file   | N/A   |
| Risk lambda          | 0.5   |
| Risk beta            | 1.0%  |
| Cut selection        | 1     |

### Simulation Settings

|                              |             |
|------------------------------|-------------|
| Simulation name              | sim1        |
| Simulation type              | Monte Carlo |
| Reset initial storage levels | TRUE        |
| Replications                 | 100         |
| Simulation start year        | N/A         |
| Simulation end year          | N/A         |
| Random seed                  |             |



First: Find JADE Input Files

Last: Create JADE Run File

© Electric Power Optimization Centre

This run file can be loaded by modifying the `data_dir` and `run_file` variables in the included Julia script: `runJADE.jl`.

The table below lists the parameters able to be used within the run files, and gives a brief description of each.

| Parameter                 | Description   |
|---------------------------|---|
| Policy name               | text giving the name of the directory for output  |
| Scenario                  | text which can (optionally) be used to specify a scenario within the input  |
| Simulation name           | text giving the name of the directory for simulation output   |
| Problem start year        | integer giving the year of the study, e.g. 2005   |
| Problem start week        | integer giving the first week of the study (usually 1)  |
| Number of weeks           | integer giving the number of weeks in the study   |
| Use saved cuts from       | text in double quotes giving the path to cuts files or "" if no previously computed cuts available  |
| Maximum iterations        | integer giving how many new cuts to compute per stage   |
| Sample start year         | integer giving the first year from which inflows will be sampled  |
| Sample end year           | integer giving the last year from which inflows will be sampled   |
| Inflow correlation length | integer giving the number of weeks to use in the DIA procedure for inflows  |
| Simulation sample size    | number of samples to simulate in a Monte Carlo simulation   |
| Random seed               | integer giving a seed used when generating random number for the simulation   |
| LB                        | flow penalty penalty in \$/MWh on river flows going below their lower bounds  |
| UB                        | flow penalty penalty in \$/MWh on river flows going above their upper bounds  |
| Cut selection             | integer specifying the minimum number of cuts cached, before SDDP.jl runs its cut-selection algorithm   |
| Risk lambda               | $\lambda$ -weight for the nested CVaR (defaults to 0 = risk neutral)  |
| Risk beta                 | $\beta$ -weight for the nested CVaR (defaults to 1 = risk neutral)  |
| Scale reservoirs          | factor to scale the volume in a reservoir to change the scale of state variables  |
| Scale objective           | factor to scale the objective in SDDP.jl  |
| Write EOH cuts file       | boolean set to <b>true</b> if a steady-state model should output end-of-horizon cuts  |
| Load EOH cuts file        | string giving the name of the end-of-horizon cut file to be loaded  |
| Steady state              | discount factor to use if you have a steady-state model; terminal water values are ignored for a steady-state model; set to 0 to disable steady-state |

## demand.csv

Demand is represented by load blocks in each week. The demand file must have comma-separated columns for node, year and week, and then a column for each load block (with headings that must match other input files). The value in each column is the load in MW for that block, i.e.,  $D_i(b, t)$ . The node refers to the transmission system, and in the default input files is either NI, HAY or SI.

| NODE | YEAR | WEEK | peak    | shoulder | offpeak |
|------|------|------|---------|----------|---------|
| NI   | 2005 | 1    | 2132.15 | 1616.51  | 1116.58 |
| NI   | 2005 | 2    | 2205.11 | 1763.09  | 1249.8  |
| ...  |      |      |         |          |         |

## fixed\_stations.csv

Some stations have fixed generation in MW. These are listed in this file as shown. This generation is subtracted from demand in the corresponding node.

| STATION   | NODE | YEAR | WEEK | peak | shoulder | offpeak |
|-----------|------|------|------|------|----------|---------|
| Aniwhenua | NI   | all  | all  | 15   | 15       | 15      |
| Patea     | NI   | all  | all  | 14   | 14       | 14      |
| Waipori   | SI   | all  | all  | 11   | 11       | 11      |
| ...       |      |      |      |      |          |         |

In the **YEAR** and **WEEK** columns, you can specify various subsets of the data. Individual values e.g. the year 2020 can be specified, ranges can be defined using a hyphen 2010–2020 and these can be combined using a semicolon delimiter. The keyword **all** specifies all the values, and values can be excluded by prepending a **!**.

E.g. **all;!4–10** in the **WEEK** column will specify weeks: 1,2,3,11,...,52.

## hours\_per\_block.csv

In each week  $t$ , each demand block  $b$  has a certain number of hours  $T(b, t)$  (usually adding up to 168)<sup>1</sup> These are listed in this file:

| YEAR | WEEK | peak | shoulder | offpeak |
|------|------|------|----------|---------|
| 2005 | 1    | 43   | 74       | 51      |
| 2005 | 1    | 69   | 54       | 45      |
| ...  |      |      |          |         |

## inflows.csv

This file describes the inflows to the various points of the system. Inflows are defined in cumecs for 27 locations in the system. Each location is either a hydro junction (implicitly defined above) or is a storage reservoir. Inflows are defined as a rate of inflow in cumecs assumed constant over each week. The model assumes that the inflows are observed at the beginning of the week and the total is made available upfront. These data are taken directly from the inflow sequences provided in the EA-EMI data set. Currently, the inflow files provided with JADE run from 1932 to 2020, but these can be updated with new information as it becomes available.

| YEAR | WEEK | Lake_Aviemore | Lake_Benmore | Lake_Hawea | ... |
|------|------|---------------|--------------|------------|-----|
| 1970 | 1    | 27            | 67           | 71         |     |
| 1970 | 2    | 16            | 41           | 62         |     |
| ...  |      |               |              |            |     |
| 2019 | 51   | 11            | 29           | 32         |     |
| 2019 | 52   | 9             | 23           | 24         |     |

<sup>1</sup>The final week of the year has 24 more hours in it (or 48 if it is a leap year).

## hydro\_arcs.csv

This file gives river reaches that are used for transferring water and do not have stations on them. Minimum flow rates and maximum flow rates in cumecs are given.

| ORIG        | DEST          | MIN_FLOW | MAX_FLOW |
|-------------|---------------|----------|----------|
| Lake_Wanaka | Lake_Dunstan  | 0        | NA       |
| Lake_Hawea  | Lake_Roxburgh | 0        | NA       |
| Clyde_tail  | Lake_Dunstan  | 50       | 1000     |
| Lake_Hawea  | Lake_Dunstan  | 0        | NA       |
| ...         |               |          |          |

## hydro\_stations.csv

This file describes the hydro stations and how they are linked together by arcs in the river chains. Each station has a capacity in MW and a specific power parameter that defines the conversion rate of cumecs to MW. Bounds can also be specified to limit the spill.

| GENERATOR | HEAD_WATER     | TAIL_WATER    | NODE | CAPACITY | SPECIFIC_POWER | MAX_SPILL_FLOW | SRMC |
|-----------|----------------|---------------|------|----------|----------------|----------------|------|
| Arapuni   | Lake_Arapuni   | Lake_Karapiro | NI   | 196.7    | 0.439847649    | NA             | 10   |
| Aratiatia | Lake_Aratiatia | Lake_Ohakuri  | NI   | 78       | 0.273437947    | NA             | 10   |
| Karapiro  | Lake_Karapiro  | Karapiro_tail | NI   | 100      | 0.266255522    | NA             | 10   |
| ...       |                |               |      |          |                |                |      |

Note that the last column (**SRMC**) is optional, but should be used if the hydro plant has variable operations and maintenance costs; the units are \$ / MWh.

## reservoirs.csv

This file defines the reservoirs; the default JADE input files specify 5 reservoirs. The initial state is measured in  $\text{Mm}^3$ , and represents the initial volume above minimum operating level (not including contingent storage).

| RESERVOIR               | INITIAL_STATE |
|-------------------------|---------------|
| Lake_Hawea              | 332.074       |
| Lakes_Manapouri_Te_Anau | 688.25        |
| Lake_Pukaki             | 2099.866      |
| Lake-Taupo              | 750.275       |
| Lake_Tekapo             | 397.029       |

## reservoir\_limits.csv

This file defines the reservoir capacities (in  $\text{Mm}^3$ ), as well as contingent storage, and associated penalties. The penalties are applied based on the volume of contingent storage utilised each week. The contingent storage capacities are given as negative values, reflecting a minimum storage level. The reservoir levels, contingent storage levels and the penalties can change over time, but the total contingent storage must be fixed for each reservoir.

| YEAR | WEEK | Lake_Hawea MAX_LEVEL | Lake_Hawea MIN_1_LEVEL | Lake_Hawea MIN_1_PENALTY | ... |
|------|------|----------------------|------------------------|--------------------------|-----|
| 2010 | 1    | 1141.95              | -263.87                | 5000                     |     |
| 2010 | 2    | 1141.95              | -263.87                | 5000                     |     |
| 2010 | 3    | 1141.95              | -263.87                | 5000                     |     |
| ...  |      |                      |                        |                          |     |

## demand\_response.csv

The load shedding model is intended to reduce energy consumption in dry periods rather than responding to excess demand in peak periods. An excerpt from the table is given below.

| DEMAND     | TRANCHE | NODE | WEEK | LOADBLOCK | MODE   | TYPE         | BOUND | BID_PRICE |
|------------|---------|------|------|-----------|--------|--------------|-------|-----------|
| industrial | low     | NI   | all  | all       | power  | proportional | 0.017 | 1000      |
| industrial | medium  | NI   | all  | all       | power  | proportional | 0.017 | 2000      |
| industrial | high    | NI   | all  | all       | power  | proportional | 0.306 | 10000     |
| Tiwai      | power   | SI   | all  | all       | power  | absolute     | 600   | 0         |
| Tiwai      | energy  | SI   | all  | all       | energy | absolute     | 50    | 500       |
| ...        |         |      |      |           |        |              |       |           |

It is possible to shed load in any load block and demand node. This incurs penalties that are given in `demand_response.csv`. The amount of load shedding / demand-response is defined as the difference between demand and supply at every demand node and block (and must be non-negative). Demand-response is specified in terms of tranches, each with a price and quantity. These tranches represent a convex, non-decreasing, cost of load shedding.

`demand_response.csv` can be defined with the `MODE` set to either `power` or `energy`. `power` corresponds to a reduction in power usage (MW) in each given `LOADBLOCK`, whereas `energy` corresponds to a reduction in energy consumption over a set of loadblocks (GWh, within a single week). Note that in order to use an energy tranche, there must be a corresponding power tranche.

If `TYPE` is set to `absolute` for a `power` tranche, then the quantity for the tranche is set to the `BOUND` specified. On the other hand, if `TYPE` is set to `absolute` for an `energy` tranche, then the total energy reduction possible over a set of loadblocks (GWh, within a single week) is set to the `BOUND` specified.

If `TYPE` is set to `proportional` for a `power` tranche, then the quantity for the tranche is set to the `BOUND` specified, multiplied by the demand at the `NODE` in the periods specified by `LOADBLOCK` for each `WEEK` (or 0, if the demand is negative). On the other hand, if `TYPE` is set to `proportional` for an `energy` tranche, then the quantity for the tranche is set to the `BOUND` specified, multiplied by the total GWh of load at the `NODE` in the set of periods specified (by `LOADBLOCK`) each `WEEK`.

## thermal\_fuel\_costs.csv

Data for thermal stations can be obtained from various sources<sup>2</sup>. These are specified in a table as shown below.

|             |      | coal   | diesel | gas    | CO2 |
|-------------|------|--------|--------|--------|-----|
| CO2_CONTENT |      | 0.0912 | 0.0741 | 0.0528 |     |
| YEAR        | WEEK |        |        |        |     |
| 2005        | 1    | 4      | 18.68  | 4.49   | 0   |
| 2005        | 2    | 4      | 18.68  | 4.49   | 0   |
| 2005        | 3    | 4      | 18.68  | 4.49   | 0   |
| ...         |      |        |        |        |     |

The CO2\_CONTENT is given is TCO<sub>2</sub>/GJ. The prices of fuels are all \$/GJ, and the CO<sub>2</sub> price is \$/TCO<sub>2</sub>

## thermal\_stations.csv

Data for thermal stations (apart from fuel cost) are defined in the file **thermal\_stations.csv**. This file defines the fuel used, the heat-rate of the station and its capacity in MW. Commissioning and decommissioning dates can also be included if these are material. If the station has yet to be commissioned, or has been decommissioned its capacity is set to 0MW.

| GENERATOR  | NODE | FUEL | HEAT_RATE | CAPACITY | START_YEAR | START_WEEK | END_YEAR | END_WEEK |
|------------|------|------|-----------|----------|------------|------------|----------|----------|
| Stratford  | NI   | gas  | 7.3       | 387      | 0          | 0          | 0        | 0        |
| Huntly_e3p | NI   | gas  | 6.8       | 403      | 2007       | 23         | 0        | 0        |
| Huntly_g1  | NI   | coal | 10.5      | 260      | 0          | 0          | 0        | 0        |
| ...        |      |      |           |          |            |            |          |          |

## station\_outages.csv

JADE models outages for planned maintenance (as well as possibly random shutdowns) as known reductions in capacity in MW as shown in the table **station\_outages.csv**. Note that these outages are assumed to occur for the entire week.

| YEAR | WEEK | Stratford | Huntly_e3p | Huntly_g1 | ... |
|------|------|-----------|------------|-----------|-----|
| 2005 | 1    | 0         | 0          | 0         |     |
| 2005 | 2    | 0         | 0          | 0         |     |
| 2005 | 3    | 0         | 0          | 0         |     |
| 2005 | 4    | 0         | 0          | 0         |     |
| 2005 | 5    | 0         | 0          | 64.2679   |     |
| ...  |      |           |            |           |     |

<sup>2</sup>These data are available from [5, 1, 6]



## generation\_outages.csv

An alternative way of defining station outages uses the following file format. This format has more flexibility, and allows outages to vary by load block. `generation_outages.csv` will take precedence over `station_outages.csv` if both are present.

| YEAR | WEEK | LOADBLOCK | STATION        | OUTAGE |
|------|------|-----------|----------------|--------|
| all  | all  | all       | Huntly_main_g3 | 250    |
| all  | all  | all       | Huntly_main_g4 | 250    |
| ...  |      |           |                |        |

## transmission.csv

JADE requires a file containing transmission lines and their capacities in MW. If only one line direction is specified, this will be taken as a single capacity that applies in both directions. However, if both directions are specified they will have their own capacity and losses.

Power is assumed to be transmitted at a constant rate in each demand block. Dynamic losses in transmission lines are modelled as piecewise linear functions defined by an increasing sequence of loss breakpoints<sup>3</sup> (e.g., `LOSSTRANCHE1`) and matching slopes (e.g., `LOSSFRACTION1`) up to each breakpoint.

| FROM_NODE | TO_NODE | CAPACITY | LOSSTRANCHE1 | LOSSFRACTION1 | LOSSTRANCHE2 | LOSSFRACTION2 | ... |
|-----------|---------|----------|--------------|---------------|--------------|---------------|-----|
| NI        | HAY     | 2343     | 100          | 0.005         | 200          | 0.015         |     |
| HAY       | NI      | 1581     | 100          | 0.005         | 200          | 0.015         |     |
| HAY       | SI      | 1446     | 100          | 0.005         | 200          | 0.015         |     |
| SI        | HAY     | 1480     | 100          | 0.005         | 200          | 0.015         |     |
| ...       |         |          |              |               |              |               |     |

## transmission-outages.csv

The transmission outages file should be set up with a row for each week / year, matching the demand data. The remaining columns detail direction-specific outages (MW) for each line. All lines / directions must be specified here. Transmission outages affect the entire week.

| YEAR | WEEK | HAY_to_SI | SI_to_HAY | HAY_to_NI | NI_to_HAY | ... |
|------|------|-----------|-----------|-----------|-----------|-----|
| 2019 | 1    | 0         | 0         | 0         | 0         |     |
| 2019 | 2    | 0         | 0         | 0         | 0         |     |
| 2019 | 3    | 0         | 0         | 0         | 0         |     |
| 2019 | 4    | 0         | 0         | 0         | 0         |     |
| 2019 | 5    | 0         | 0         | 0         | 0         |     |
| ...  |      |           |           |           |           |     |

<sup>3</sup>The breakpoints are the flow quantities where the loss slopes change.

## line-outages.csv

There is a more compact way of defining transmission outages using the following file format. `line-outages.csv` will take precedence over `transmission-outages.csv` if both are present.

| YEAR | WEEK  | LOADBLOCK | LINE      | REDUCTION |
|------|-------|-----------|-----------|-----------|
| all  | 42    | peak      | NI_TO_HAY | 500       |
| all  | 42    | peak      | HAY_TO_NI | 500       |
| all  | 20-24 | all       | HAY_TO_SI | 500       |
| all  | 20-24 | all       | SI_TO_HAY | 500       |
| ...  |       |           |           |           |

## 3.3 Training and Simulating JADE models

As previously mentioned, JADE can be run either using the DOASA run files, or by coding the parameters of the JADE model directly.

Either way, there are two steps in running a JADE model: training the model, which determines water value surfaces, which define marginal water value, as a function of storage; and simulating the model, which implements the policy for a number of random or supplied inflow sequences. JADE contains several files in the `examples` folder that implement the training and simulation for several JADE models.

You first must install a solver, commercial solvers such as CPLEX and Gurobi are more reliable and less prone to numerical errors. For example, if you are using Gurobi, you must load both the JADE and Gurobi packages with the command:

---

```
using JADE, JuMP, Gurobi

# Set up a solver (in this case, Gurobi) ensuring that no output will be displayed
env = Gurobi.Env()
optimizer = optimizer_with_attributes(() -> Gurobi.Optimizer(env), "OutputFlag" => 0)

# Set the JADE path to be this file's folder
ENV["JADE_DIR"] = @__DIR__
```

---

### 3.3.1 Initialising and training JADE a model

JADE supports loading the parts of run files for the set up and training of a JADE model into `RunData` objects through the two commands:

---

```
# Create JADE model from <JADE_DIR>/Input/default using run.csv
rundata = define_JADE_model("default", run_file = "run")
model = create_JADE_model(rundata, optimizer)

# Define JADE training options from <JADE_DIR>/Input/default/run.csv
solve_options = define_JADE_solve_options("default", run_file = "run")
```

---

In this case the training data will be read from the run file named `"run.csv"` in the `"default"` input data folder.

Alternatively, JADE allows the user to directly define the model settings in code, for example:

---

```
rundata = define_JADE_model(data_dir = "default",
                           policy_dir = "policy",
                           start_year = 2008,
                           hydro_sample_range = 2005:2008,
                           discount = 0.9,
                           DIA = 5,
                           flow_penalties = (under = 500, over = 50))
model = create_JADE_model(rundata, optimizer)
solve_options = define_JADE_solve_options(iterations = 2000)
```

---

Note that the `hydro_sample_range` attribute can take a general array: `[2001,2003,2005:2008]`. When computing a policy, JADE samples inflows uniformly from years in `hydro_sample_range`. More details about these functions can be found in the API manual: [define\\_JADE\\_model\(\)](#), [define\\_JADE\\_solve\\_options\(\)](#).

The model is trained using the `JADE.optimize_policy!()` function, which runs the SDDP training and produces water values that define a hydro-release policy.

---

```
JADE.optimize_policy!(model, solve_options, optimizer)
```

---

### 3.3.2 Simulating a JADE policy

Once a model is trained, the next step is to simulate the policy, given various inflow sequences. It is important to bear in mind that DOASA `run.csv` files specify parameters for both the training and simulation. In JADE, a simulation can set up from the same `run.csv` file:

---

```
simulation_settings = define_JADE_simulation("default", run_file = "run")
```

---

However, JADE is flexible, and allows different types of simulations to be defined explicitly; for example a Monte-Carlo simulation with 100 replications can be set up as follows:

---

```
simulation_settings = define_JADE_simulation(sim_dir = "montecarlo",
                                           sim_type = :monte_carlo,
                                           replications = 100)
```

---

"montecarlo" is the subdirectory where the output files will be stored. The full path to these files is: `Output/<data-dir>/<policy-dir>/montecarlo`.

It's also possible to simulate a sequence of historical years' inflows:

---

```
simulation_settings = define_JADE_simulation(sim_dir = "hist-seq",
                                           sim_type = :historical,
                                           sim_years = 1970:2009)
```

---

Note the `sim_years` parameter can accept a vector of years (or `UnitRange{Int}`) in any order.

Finally, a random sample of historical years can also be used for simulation. In the code below, we generate 100 replications using inflows from historical years, but the order of years is randomly selected from 1970 to 2009, with replacement.

---

```
simulation_settings = define_JADE_simulation(sim_dir = "hist-rand",
                                           sim_type = :historical,
                                           replications = 100,
                                           sim_years = 1970:2009,
                                           randomize_years = true)
```

---

For each of the randomised simulations, it is also possible to set the random seed using the additional argument: `random_seed = <seed>`. For additional details about setting up simulations, see the full definition of `define_JADE_simulation()` in the API manual. Once a simulation has been defined, a policy can be simulated using the `JADE.simulate()` function

---

```
results = JADE.simulate(policy, simulation_settings)
```

---

It is then simple to visualise the results of the simulation using

---

```
JADE.plot_storage(results, "default")  
JADE.plot_prices(results, "default")
```

---

This will generate an interactive web-page of the storage trajectories of all the reservoirs or prices for each load block, for each replication of the simulation. The web-page generated is saved with a filename incorporating the string given – in this case "default".

### 3.3.3 Cuts

In JADE, due to the separation between the `RunData` (detailing the model training settings), and the simulation settings, it's possible to first compute a policy, saving the cuts, and then load these at a later time to simulate the policy. Note that all simulations begin in the start week of the start year defined in the model settings.

A user can also warm-start JADE training with cuts prepared earlier; this can be done by specifying an existing policy directory and setting `warmstart_cuts` to be `true`, when defining the JADE solver options.

---

```
define_JADE_solve_options(iterations = 2000,  
                           warmstart_cuts = true)
```

---

By default JADE will look in the policy directory of the model for the cuts. In order to specify a different cut file, one can add this additional solver option:

---

```
define_JADE_solve_options(iterations = 2000,  
                           warmstart_cuts = true,  
                           saved_cuts = "<cut-dir>")
```

---

There is some error-protection in JADE to ensure that model settings are compatible with previous cut files; e.g. you may be loading cut files for a simulation. However, ultimately it's up to the user to understand how cut files are being applied when warm-starting models for training or simulation.

Cuts are computed and written to a JSON file by the `SDDP.jl` package. If we use `run.jl` to generate a policy, cuts will be stored in a file named `cuts.json` in the appropriate policy directory.

The steady-state JADE models generate cuts for the final week as it cycles back to the first week, no such cuts exist for the finite-horizon case.

## 3.4 Output

All output from JADE is stored in a folder named **output**, in the user's working directory. There are two types of output. The first type relates to policy generation and is only produced if JADE performs at least one SDDP iteration. The second type of output is only created after policy simulation.

### 3.4.1 Policy Generation Output

Three `.json` files are stored in the policy directory. These specify the model settings that were being used when that policy was trained along with the constructed cuts. In the same folder `adjusted_inflows.csv` gives the inflows after modification using DIA. This file is created every time a `JADEData` object is built.

### Policy Simulation Output

In the descriptions below, “replication” refers to a sequence of inflows from a simulation.

| Output file / folder       | Description  |
|----------------------------|--|
| <b>FlowLBCost.csv</b>      | Cost (\$) incurred from flows going below their lower bound in each stage, in every replication.   |
| <b>FlowUBCost.csv</b>      | Cost (\$) incurred from flows going above their upper bound in each stage, in every replication.   |
| <b>FutureCost.csv</b>      | Approximated expected future cost (\$) at the end of every stage, in every replication. This is the value of the future cost function, as approximated by cuts.  |
| <b>InflowsOutput.csv</b>   | All inflow values observed in every stage, in every replication in cumecs.   |
| <b>LostLoad.csv</b>        | Total cost (\$) incurred from load shedding in every stage, in every replication.  |
| <b>PresentCost.csv</b>     | Total present cost (\$) in each stage, in every replication. This includes the thermal cost, flow penalties, and load shedding costs.  |
| <b>SpilledEnergy_x.csv</b> | Set of files containing the spilled energy in MWh in load block “x” in each stage, in every replication.   |
| <b>StoredEnergy.csv</b>    | The total energy in MWh, as a total over all reservoirs, at the end of each stage, in every replication.   |
| <b>SummedCosts.csv</b>     | The value of the present cost plus the future cost in each stage, in every replication. Here, “future cost” refers to the same cost reported in <code>FutureCost.csv</code> (it is derived from the approximation of the future cost, given by the cuts in the model). |
| <b>TotalCost.csv</b>       | The objective value for every replication. This is derived from the summation of all present costs over all stages in a simulation.  |
| <b>TidyResults.csv</b>     | This is a CSV file containing all the key outputs. This file may be very large, and require processing in statistical software such as R.  |

# Bibliography

- [1] Denne, T., Bond-Smith, S., Hennessy, W., 2009. Coal prices in New Zealand markets, COVEC report for New Zealand Ministry of Economic Development, (downloadable from <http://www.med.govt.nz/upload/68784/coal-price-report.pdf>).
- [2] Dunning, I., Huchette, J., Lubin, M., 2015. JuMP: A modeling language for mathematical optimization. *arXiv:1508.01982 [math.OC]*. Available online: <http://arxiv.org/abs/1508.01982>
- [3] Dowson, O., Kapelevich, L., 2021. SDDP.jl: a Julia package for stochastic dual dynamic programming, *INFORMS Journal on Computing*. 33, 1, pp 27-33.
- [4] Dowson, O., 2020. The policy graph decomposition of multistage stochastic optimization problems, *Networks*. 76, 1. pp 3–23.
- [5] New Zealand Electricity Commission database (downloadable from <http://www.electricitycommission.govt.nz/opdev/modelling/gpas/May2007/-Generation>).
- [6] New Zealand Ministry of Economic Development - Energy Data, June 2009, (downloadable from [http://www.med.govt.nz/templates/MultipageDocumentTOC\\_\\_\\_\\_21660.aspx](http://www.med.govt.nz/templates/MultipageDocumentTOC____21660.aspx)).
- [7] Philpott, A., Pritchard, G., 2018. EMI-DOASA4.0 (downloadable from <https://www.epoc.org.nz/doasa.html>).