AMS

Release 0.9.0

Jinning Wang

Feb 27, 2024

AMS MANUAL

Useful Links: Source Repository | Report Issues | Q&A | LTB Repository | ANDES Repository

LTB AMS is an open-source packages for dispatch modeling, serving as the market simulator for the CURENT Large scale Testbed (LTB).

AMS enables **flexible** dispatch modeling and **interoprability** with the in-house dynamic simulator ANDES.

Getting started

New to AMS? Check out the getting started guides.

To the getting started guides

Examples

The examples of using AMS for power system dispatch study.

To the examples

Model development guide

New dispatch modeling in AMS.

To the development guide

API reference

The API reference of AMS.

To the API reference

Using AMS for Research?

Please cite our paper [Cui2021] if AMS is used in your research for publication.

CHAPTER

ONE

GETTING STARTED

1.1 Package Overview

AMS is an open-source packages for flexible dispatch modeling and co-simulation with the in-house dynanic simulation engine ANDES.

AMS is currently under active development. To get involved,

- Report issues in the GitHub issues page
- Learn version control with the command-line git or GitHub Desktop

This work was supported in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under NSF Award Number EEC-1041877 and the CURENT Industry Partnership Program. AMS is made open source as part of the CURENT Large Scale Testbed project.

AMS is developed and actively maintained by Jinning Wang. See the GitHub repository for a full list of contributors.

1.2 Installation

1.2.1 New to Python

Setting Up Mambaforge

If you are new to Python and want to get started quickly, you can use Mambaforge, which is a conda-like package manager configured with conda-forge.

Step 1:

Downloaded the latest Mambaforge for your platform from https://github.com/conda-forge/miniforge#mambaforge. Most users will use `x86_64(amd64)` for Intel and AMD processors. Mac users with Apple Silicon should use `arm64(Apple Silicon)` for best performance.

Next, complete the Mambaforge installation on your system.

Note: Mambaforge is a drop-in replacement for conda. If you have an existing conda installation, you can replace all following `mamba` commands with `conda` and achieve the same functionality.

If you are using Anaconda or Miniconda on Windows, you should open `Anaconda Prompt` instead of `Miniforge Prompt`.

Step 2:

Open Terminal (on Linux or maxOS) or *Miniforge Prompt* (on Windows, **not cmd!!**). Make sure you are in a conda environment - you should see `(base)` prepended to the command-line prompt, such as `(base) C:\Users\username>`.

Create an environment for AMS (recommended)

```
mamba create --name ams python=3.8
```

Activate the new environment with

```
mamba activate ams
```

Note: You will need to activate the `ams` environment every time in a new Miniforge Prompt or shell.

If these steps complete without error, you now have a working Python environment. See the commands at the top to *Getting started* AMS.

1.2.2 Extra support package

Some AMS features require extra support packages, which are not installed by default. For example, to build the documentation, one will need to install development packages. Other packages will be required for interoperability.

The extra support packages are specified in groups. The following group names are supported, with descriptions given below:

- `dev`: packages to support development such as testing and documentation

Note: TODO: Extra support packages are not supported by conda/mamba installation. One needs to install AMS with `pip`.

To install packages in the `dev` when installing AMS, do:

```
pip install ltbams[dev]
```

To install all extra packages, do:

```
pip install ltbams[all]
```

One can also inspect the `requirements-extra.txt` to identify the packages for manual installation.

1.2.3 Develop Install

The development mode installation is for users who want to modify the code and, for example, develop new models or routines. The benefit of development mode installation is that changes to source code will be reflected immediately without re-installation.

Step 1: Get AMS source code

As a developer, you are strongly encouraged to clone the source code using `git` from either your fork or the original repository. Clone the repository with

```
git clone https://github.com/CURENT/ams
```

Note: Replace the URL with yours to use your fork. With `git`, you can later easily update the source code and perform version control.

Alternatively, you can download the AMS source code from https://github.com/CURENT/ams and extract all files to the path of your choice. Although works, this method is discouraged because tracking changes and pushing back code edits will require significant manual efforts.

Step 2: Install dependencies

In the Mambaforge environment, use `cd` to change directory to the AMS root folder. The folder should contain the `setup.py` file.

Install dependencies with

```
mamba install --file requirements.txt
mamba install --file requirements-extra.txt
```

Alternatively, you can install them with `pip`:

```
pip install -r requirements.txt
pip install -r requirements-extra.txt
```

Step 3: Install AMS in the development mode using

```
python3 -m pip install -e .
```

Note the dot at the end. Pip will take care of the rest.

Note: The AMS version number shown in `pip list` will stuck at the version that was intalled, unless AMS is develop-installed again. It will not update automatically with `git pull`.

To check the latest version number, check the preamble by running the `ams` command or chek the output of `python -c "import ams; print(ams.__version__)"`

Note: AMS updates may infrequently introduce new package requirements. If you see an `ImportError` after updating AMS, you can manually install the missing dependencies or redo *Step 2*.

Note: To install extra support packages, one can append `[NAME_OF_EXTRA]` to `pip install -e .`. For example, `pip install -e .[interop]` will install packages to support interoperability when installing AMS in the development, editable mode.

1.2.4 Updating AMS

Warning: If AMS has been installed in the development mode using source code, you will need to use `git` or the manual approach to update the source code. In this case, Do not proceed with the following steps, as they will install a separate site-package installation on top of the development one.

Regular AMS updates will be pushed to both `conda-forge` and Python package index. It is recommended to use the latest version for bug fixes and new features. We also recommended you to check the *Release notes* before updating to stay informed of changes that might break your downstream code.

Depending you how you installed AMS, you will use one of the following ways to upgrade.

If you installed it from mamba or conda, run

```
conda install -c conda-forge --yes ltbams
```

If you install it from PyPI (namely, through `pip`), run

```
python3 -m pip install --yes ltbams
```

1.2.5 Uninstall Multiple Copies

A common mistake new users make is to have multiple copies of AMS installed in the same environment. This can happen when one previously installed AMS in the development mode but later ran `conda install` or `python3 -m pip install` to install the latest version. As a result, only the most recently installed version will be accessible.

In this case, we recommend that you uninstall all version and reinstall only one copy using your preferred mode. Uninstalling all copies can be done by calling `conda remove ams` and `python3 -m pip uninstall ams`. The prompted path will indicate the copy to be removed. One may need to run the two commands for a couple of time until the package managers indicate that the `ams` package can no longer be found.

1.2.6 Troubleshooting

If you get an error message on Windows, reading

```
ImportError: DLL load failed: The specified module could not be found.
```

It is a path issue of your Python. In fact, Python on Windows is so broken that many people are resorting to WSL2 just for Python. Fixes can be convoluted, but the easiest one is to install AMS in a Conda/Mambaforge environment.

1.3 Input formats

AMS currently supports the following input formats:

- `.xlsx`: Excel spreadsheet file with AMS data
- `.raw`: PSS/E RAW format
- `.m`: MATPOWER format
- `.py`: PYPOWER format

1.3.1 AMS xlsx

The AMS xlsx format allows one to use Excel for convenient viewing and editing. If you do not use Excel, there are alternatives such as the free and open-source LibreOffice.

Format definition

The AMS xlsx format contains multiple workbooks (also known as "sheets") shown as tabs at the bottom. The name of a workbook is a *model* name, and each workbook contains the parameters of all *devices* that are *instances* of the model.

1.3.2 PSS/E RAW

The Siemens PSS/E data format is a widely used for power system simulation. PSS/E uses a variety of plain-text files to store data for different actions. The RAW format (with file extension `.raw`) is used to store the steady-state data for power flow analysis. Leveraging ANDES PSS/E parser, one can load PSS/E RAW files into AMS for power flow study.

RAW Compatibility

AMS supports PSS/E RAW in versions 32 and 33. Newer versions of `raw` files can store PSS/E settings along with the system data, but such feature is not yet supported in AMS. Also, manually edited `raw` files can confuse the parser in AMS. Following manual edits, it is strongly recommended to load the data into PSS/E and save the case as a v33 RAW file.

AMS supports most power flow models in PSS/E. It needs to be recognized that the power flow models in PSS/E is is a larger set compared with those in AMS. For example, switched shunts in PSS/E are converted to fixed ones, not all three-winding transformer flags are supported, and HVDC devices are not yet converted. This is not an exhaustive list, but all of them are advanced models.

We welcome contributions but please also reach out to us if you need to arrange the development of such models.

Loading files

In the command line, PSS/E files can be loaded with

```
ams run kundur.raw
```

Likewise, one can convert PSS/E files to AMS xlsx:

```
ams run kundur.raw -c
```

This will convert all models in the RAW files.

To load PSS/E files into a scripting environment, see Example - "Working with Data".

1.3.3 MATPOWER

The data file format of MATPOWER is excerpted below for quick reference. For more information, see the MATPOWER User's Manual.

Bus Data

name	column	description
BUS_I	1	bus number (positive integer)
BUS_TYPE	2	bus type (1 = PQ, 2 = PV, 3 = ref, 4 = isolated)
PD	3	real power demand (MW)
QD	4	reactive power demand (MVAr)
GS	5	shunt conductance (MW demanded at V = 1.0 p.u.)
BS	6	shunt susceptance (MVAr injected at V = 1.0 p.u.)
BUS AREA	7	area number (positive integer)
VM	8	voltage magnitude (p.u.)
VA	9	voltage angle (degrees)
BASE_KV	10	base voltage (kV)
ZONE	11	loss zone (positive integer)
VMAX	12	maximum voltage magnitude (p.u.)
VMIN	13	minimum voltage magnitude (p.u.)
LAM_P [1]	14	Lagrange multiplier on real power mismatch (u/MW)
LAM_Q [1]	15	Lagrange multiplier on reactive power mismatch (u/MVar)
MU_VMAX [1]	16	Kuhn-Tucker multiplier on upper voltage limit (u/p.u.)
MU_VMIN [1]	17	Kuhn-Tucker multiplier on lower voltage limit (u/p.u.)

1. Included in OPF output, typically not included (or ignored) in input matrix.Here we assume the objective function has units u.

Generator Data

name	column	description
GEN_BUS	1	bus number
PG	2	real power output (MW)
QG	3	reactive power output (MVAr)
QMAX	4	maximum reactive power output (MVAr)
QMIN	5	minimum reactive power output (MVAr)
VG [3]	6	voltage magnitude setpoint (p.u.)
MBASE	7	total MVA base of machine, defaults to baseMVA
GEN_STATUS	8	machine status, > 0 for in-service , <= 0 for out-of-service
PMAX	9	maximum real power output (MW)
PMIN	10	minimum real power output (MW)
PC1 [1]	11	lower real power output of PQ capability curve (MW)
PC2 [1]	12	upper real power output of PQ capability curve (MW)
QC1MIN [1]	13	minimum reactive power output at PC1 (MVAr)
QC1MAX [1]	14	maximum reactive power output at PC1 (MVAr)
QC2MIN [1]	15	minimum reactive power output at PC2 (MVAr)
QC2MAX [1]	16	maximum reactive power output at PC2 (MVAr)
RAMP_AGC [1]	17	ramp rate for load following/AGC (MW/min)
RAMP_10 [1]	18	ramp rate for 10 minute reserves (MW)
RAMP_30 [1]	19	ramp rate for 30 minute reserves (MW)
RAMP_Q [1]	20	ramp rate for reactive power (2 sec timescale) (MVAr/min)
APF [1]	21	area participation factor
MU_PMAX [2]	22	Kuhn-Tucker multiplier on upper Pg limit (u/MW)
MU_PMIN [2]	23	Kuhn-Tucker multiplier on lower Pg

1. Not included in version 1 case format.
2. Included in OPF output, typically not included (or ignored) in input matrix. Here we assume the objective function has units u.
3. Used to determine voltage setpoint for optimal power flow only if `opf.use_vg` option is non-zero (0 by default). Otherwise generator voltage range is determined by limits set for corresponding bus in bus matrix.

Branch Data

name	column	description
F_BUS	1	"from" bus number
T_BUS	2	"to" bus number
BR_R	3	resistance (p.u.)
BR_X	4	reactance (p.u.)
BR_B	5	total line charging susceptance (p.u.)
RATE_A [1]	6	MVA rating A (long term rating), set to 0 for unlimited
RATE_B [1]	7	MVA rating B (short term rating), set to 0 for unlimited
RATE_C [1]	8	MVA rating C (emergency rating), set to 0 for unlimited
TAP	9	transformer off nominal turns ratio
SHIFT	10	transformer phase shift angle (degrees), positive => delay
BR_STATUS	11	initial branch status, 1 = in-service, 0 = out-of-service
ANGMIN [2]	12	minimum angle difference, Of - Ot (degrees)
ANGMAX [2]	13	maximum angle difference, 0,-0 - (degrees)
PF [3]	14	real power injected at "from" bus end (MW)
QF [3]	15	reactive power injected at "from" bus end (MVAr)
PT [3]	16	real power injected at "to" bus end (MW)
QT [3]	17	reactive power injected at "to" bus end (MVAr)
MU_SF [4]	18	Kuhn-Tucker multiplier on MVA limit at "from" bus (u/MVA)
MU_ST [4]	19	Kuhn-Tucker multiplier on MVA limit at "to" bus (u/MVA)
MU_ANGMIN [4]	20	Kuhn-Tucker multiplier lower angle difference limit (u/degree)
MU_ANGMAX [4]	21	Kuhn

1. Used to specify branch flow limits. By default these are limits on apparent power with units in MVA. However, the 'opf.flow lim' option can be used to specify that the limits are active power or current, in which case the ratings are specified in MW or kAV_{basekV}, respectively. For current this is equivalent to an MVA value at a 1 p.u. voltage.
2. Not included in version 1 case format. The voltage angle difference is taken to be unbounded below if $ANGMIN360$ and unbounded above if $ANGMAX360$. If both parameters are zero, the voltage angle difference is unconstrained.
3. Included in power flow and OPF output, ignored on input.
4. Included in OPF output, typically not included (or ignored) in input matrix. Here we assume the objective function has units u.

Generator Cost Data

name	column	description
MODEL	1	cost model, 1 = piecewise linear, 2 = polynomial
STARTUP	2	startup cost in US dollars [1]
SHUTDOWN	3	shutdown cost in US dollars [1]
NCOST	4	number of points of an n-segment piecewise linear cost function or coefficients of an n-th order polynomial cost function
COST [2]	5	parameters defining total cost function $f(p)$

1. Not currently used by any Matpower functions.

2. MODEL = 1, $f(p)$ is defined by the coordinates $(p_1, f_1), (p_2, f_2), ..., (p_N, f_N)$; MODEL = 2, $f(p) = c_n p^n + ... + c_1 p^1 + c_0$.

1.3.4 PYPOWER

AMS includes PYPOWER cases in version 2 for dispatch modeling and analysis. PYPOWER cases follow the same format as MATPOWER.

The PYPOWER case is defined as a Python dictionary that includes `bus`, `gen`, `branch`, `areas`, and `gencost`. Defines the PYPOWER case file format.

A PYPOWER case file is a Python file or MAT-file that defines or returns a dict named `ppc`, referred to as a "PYPOWER case dict". The keys of this dict are `bus`, `gen`, `branch`, `areas`, and `gencost`. With the exception of C{baseMVA}, a scalar, each data variable is an array, where a row corresponds to a single bus, branch, gen, etc. The format of the data is similar to the PTI format described in PTI Load Flow Data Format.

Example Case9

```
ppc = {"version": '2'}

##-----  Power Flow Data  -----##
## system MVA base
ppc["baseMVA"] = 100.0

## bus data
# bus_i type Pd Qd Gs Bs area Vm Va baseKV zone Vmax Vmin
ppc["bus"] = array([
    [1, 3, 0,    0, 0, 0, 1, 1, 0, 345, 1, 1.1, 0.9],
    [2, 2, 0,    0, 0, 0, 1, 1, 0, 345, 1, 1.1, 0.9],
    [3, 2, 0,    0, 0, 0, 1, 1, 0, 345, 1, 1.1, 0.9],
    [4, 1, 0,    0, 0, 0, 1, 1, 0, 345, 1, 1.1, 0.9],
    [5, 1, 90,  30, 0, 0, 1, 1, 0, 345, 1, 1.1, 0.9],
    [6, 1, 0,    0, 0, 0, 1, 1, 0, 345, 1, 1.1, 0.9],
    [7, 1, 100, 35, 0, 0, 1, 1, 0, 345, 1, 1.1, 0.9],
    [8, 1, 0,    0, 0, 0, 1, 1, 0, 345, 1, 1.1, 0.9],
    [9, 1, 125, 50, 0, 0, 1, 1, 0, 345, 1, 1.1, 0.9]
])

## generator data
# bus, Pg, Qg, Qmax, Qmin, Vg, mBase, status, Pmax, Pmin, Pc1, Pc2,
# Qc1min, Qc1max, Qc2min, Qc2max, ramp_agc, ramp_10, ramp_30, ramp_q, apf
ppc["gen"] = array([
    [1, 0,   0, 300, -300, 1, 100, 1, 250, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [2, 163, 0, 300, -300, 1, 100, 1, 300, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [3, 85,  0, 300, -300, 1, 100, 1, 270, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
])

## branch data
# fbus, tbus, r, x, b, rateA, rateB, rateC, ratio, angle, status, angmin, angmax
ppc["branch"] = array([
    [1, 4, 0,      0.0576, 0,     250, 250, 250, 0, 0, 1, -360, 360],
    [4, 5, 0.017,  0.092,  0.158, 250, 250, 250, 0, 0, 1, -360, 360],
```

(continues on next page)

(continued from previous page)

```
    [5, 6, 0.039,  0.17,   0.358, 150, 150, 150, 0, 0, 1, -360, 360],
    [3, 6, 0,      0.0586, 0,     300, 300, 300, 0, 0, 1, -360, 360],
    [6, 7, 0.0119, 0.1008, 0.209, 150, 150, 150, 0, 0, 1, -360, 360],
    [7, 8, 0.0085, 0.072,  0.149, 250, 250, 250, 0, 0, 1, -360, 360],
    [8, 2, 0,      0.0625, 0,     250, 250, 250, 0, 0, 1, -360, 360],
    [8, 9, 0.032,  0.161,  0.306, 250, 250, 250, 0, 0, 1, -360, 360],
    [9, 4, 0.01,   0.085,  0.176, 250, 250, 250, 0, 0, 1, -360, 360]
])

##-----  OPF Data  -----##
## area data
# area refbus
ppc["areas"] = array([
    [1, 5]
])

## generator cost data
# 1 startup shutdown n x1 y1 ... xn yn
# 2 startup shutdown n c(n-1) ... c0
ppc["gencost"] = array([
    [2, 1500, 0, 3, 0.11,   5,   150],
    [2, 2000, 0, 3, 0.085,  1.2, 600],
    [2, 3000, 0, 3, 0.1225, 1,   335]
])
```

Version Information

There are two versions of the PYPOWER case file format. The current version of PYPOWER uses version 2 of the PYPOWER case format internally and includes a `version` field with a value of `2` to make the version explicit. Earlier versions of PYPOWER used the version 1 case format, which defined the data matrices as individual variables, as opposed to keys of a dict. Case files in version 1 format with OPF data also included an (unused) `areas` variable. While the version 1 format has now been deprecated, it is still handled automatically by `loadcase` and `savecase` which are able to load and save case files in both version 1 and version 2 formats.

See also doc for *idx_bus*, *idx_brch*, *idx_gen*, *idx_area* and *idx_cost* regarding constants which can be used as named column indices for the data matrices. Also described in the first three are additional results columns that are added to the bus, branch, and gen matrices by the power flow and OPF solvers.

The case dict also allows for additional fields to be included. The OPF is designed to recognize fields named `A`, `l`, `u`, `H`, `Cw`, `N`, `fparm`, `z0`, `zl`, and `zu` as parameters used to directly extend the OPF formulation (see doc for *opf* for details). Other user-defined fields may also be included and will be automatically loaded by the `loadcase` function and, given an appropriate 'savecase' callback function (see doc for *add_userfcn*), saved by the `savecase` function.

Bus

1. bus number (positive integer)
2. bus type - PQ bus = 1 - PV bus = 2 - reference bus = 3 - isolated bus = 4
3. `Pd`, real power demand (MW)
4. `Qd`, reactive power demand (MVAr)
5. `Gs`, shunt conductance (MW demanded at V = 1.0 p.u.)
6. `Bs`, shunt susceptance (MVAr injected at V = 1.0 p.u.)
7. area number (positive integer)
8. `Vm`, voltage magnitude (p.u.)
9. `Va`, voltage angle (degrees)
10. `baseKV`, base voltage (kV)
11. `zone`, loss zone (positive integer)
12. `maxVm`, maximum voltage magnitude (p.u.)
13. `minVm`, minimum voltage magnitude (p.u.)

Generator

1. bus number
2. `Pg`, real power output (MW)
3. `Qg`, reactive power output (MVAr)
4. `Qmax`, maximum reactive power output (MVAr)
5. `Qmin`, minimum reactive power output (MVAr)
6. `Vg`, voltage magnitude setpoint (p.u.)
7. `mBase`, total MVA base of this machine, defaults to baseMVA
8. status - > `0` - machine in service - <= `0` - machine out of service
9. `Pmax`, maximum real power output (MW)
10. `Pmin`, minimum real power output (MW)
11. `Pc1`, lower real power output of PQ capability curve (MW)
12. `Pc2`, upper real power output of PQ capability curve (MW)
13. `Qc1min`, minimum reactive power output at Pc1 (MVAr)
14. `Qc1max`, maximum reactive power output at Pc1 (MVAr)
15. `Qc2min`, minimum reactive power output at Pc2 (MVAr)
16. `Qc2max`, maximum reactive power output at Pc2 (MVAr)
17. ramp rate for load following/AGC (MW/min)
18. ramp rate for 10-minute reserves (MW)
19. ramp rate for 30-minute reserves (MW)
20. ramp rate for reactive power (2-sec timescale) (MVAr/min)

21. APF, area participation factor

Branch

1. `f`, from bus number
2. `t`, to bus number
3. `r`, resistance (p.u.)
4. `x`, reactance (p.u.)
5. `b`, total line charging susceptance (p.u.)
6. `rateA`, MVA rating A (long-term rating)
7. `rateB`, MVA rating B (short-term rating)
8. `rateC`, MVA rating C (emergency rating)
9. `ratio`, transformer off nominal turns ratio (= `0` for lines)
10. `angle`, transformer phase shift angle (degrees), positive -> delay
    - (Gf, shunt conductance at from bus p.u.)
    - (Bf, shunt susceptance at from bus p.u.)
    - (Gt, shunt conductance at to bus p.u.)
    - (Bt, shunt susceptance at to bus p.u.)
11. initial branch status, 1 - in service, 0 - out of service
12. minimum angle difference, angle(Vf) - angle(Vt) (degrees)
13. maximum angle difference, angle(Vf) - angle(Vt) (degrees)

Generator Cost

Note: If `gen` has `ng` rows, then the first `ng` rows of `gencost` contain the cost for active power produced by the corresponding generators. If `gencost` has $2 \times ng$ rows then rows $ng + 1$ to $2 \times ng$ contain the reactive power costs in the same format.

1. `model`, 1 - piecewise linear, 2 - polynomial
2. `startup`, startup cost in US dollars
3. `shutdown`, shutdown cost in US dollars
4. `N`, number of cost coefficients to follow for polynomial cost function, or number of data points for piecewise linear. The following parameters define the total cost function `f(p)`, where units of `f` and `p` are $/hr and MW (or MVAr), respectively.
    - For MODEL = 1: `p0, f0, p1, f1, ..., pn, fn` where `p0 < p1 < ... < pn` and the cost `f(p)` is defined by the coordinates `(p0,f0), (p1,f1), ..., (pn,fn)` of the end/break-points of the piecewise linear cost function.
    - For MODEL = 2: `cn, ..., c1, c0` `n + 1` coefficients of an `n`-th order polynomial cost function, starting with the highest order, where cost is $f(p) = c_n \times p^n + \ldots + c_1 \times p + c_0$.

Area (deprecated)

Note: This data is not used by PYPOWER and is no longer necessary for version 2 case files with OPF data.

1. `i`, area number
2. `price_ref_bus`, reference bus for that area

1.4 Test cases

AMS ships with with test cases in the `ams/cases` folder. The cases can be found in the online repository.

1.4.1 Summary

Below is a summary of the folders and the corresponding test cases. Some folders contain a README file with notes. When viewing the case folder on GitHub, one can conveniently read the README file below the file listing.

- `5bus`: a small PJM 5-bus test case for power flow study [PJM5].
- `ieee14` and `ieee39`: the IEEE 14-bus and 39-bus test cases [IEEE].
- `ieee123`: the IEEE 123-bus test case [TSG].
- `matpower`: a subset of test cases from [MATPOWER].
- `npcc` and `wecc`: NPCC 140-bus and WECC 179-bus test cases [SciData].

1.4.2 How to contribute

We welcome the contribution of test cases! You can make a pull request to contribute new test cases. Please follow the structure in the `cases` folder and provide an example Jupyter notebook (see `examples/demonstration`) to showcase the results of your system.

1.5 Verification

This section presents the verification of AMS by comparing the DCOPF results with other tools.

1.5.1 DCOPF Verification

Prepared by Jinning Wang.

Conclusion

For test cases, DCOPF results from AMS are identical to that from MATPOWER.

```
import datetime

import numpy as np
import pandas as pd

import ams
```

```
print("Last run time:", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

print(f'ams: {ams.__version__}')
```

```
Last run time: 2024-01-16 16:03:19
ams: 0.8.0.post6+g32850c1
```

Using built-in MATPOWER cases as inputs.

```
cases = [
    ams.get_case('matpower/case14.m'),
    ams.get_case('matpower/case39.m'),
    ams.get_case('matpower/case118.m'),
    ams.get_case('npcc/npcc.m'),
    ams.get_case('wecc/wecc.m'),
    ams.get_case('matpower/case300.m'),]

case_names = [case.split('/')[-1].split('.')[0] for case in cases]
```

```
ams_obj = np.zeros(len(cases))

for i, case in enumerate(cases):
    sp = ams.load(case, setup=True)
    sp.DCOPF.init()
    sp.DCOPF.solve(solver='ECOS')
    ams_obj[i] = sp.DCOPF.obj.v
```

Following MATPOWER results are obtained using MATPOWER 8.0b1 and Matlab R2023b.

```
mp_obj = np.array([7642.59177699,  41263.94078588,
                   125947.8814179, 705667.88555058,
                   348228.35589771, 706292.32424361])
```

```
res = pd.DataFrame({'AMS': ams_obj, 'MATPOWER': mp_obj},
                   index=case_names)
res
```

```
                   AMS       MATPOWER
case14     7642.591752    7642.591777
case39    41263.940187   41263.940786
case118  125947.881253  125947.881418
```

(continues on next page)

(continued from previous page)

```
npcc     705667.885550  705667.885551
wecc     348228.355895  348228.355898
case300  706292.326604  706292.324244
```

1.6 License

1.6.1 GNU Public License v3

1.7 Quick install

Working with conda?

AMS will available on conda-forge and can be installed with Anaconda, Miniconda, and Mambaforge:

```
conda install -c conda-forge ltbams
```

Prefer pip?

AMS will be installed via pip from PyPI.

```
pip install ltbams
```

New to Python?

Set up a Mambaforge environment following *Setting Up Mambaforge*. We recommend Mambaforge on Windows and Apple Silicon for new users.

Are you a developer?

Installing from source? Looking to develop models? Check the guide in *Develop Install*.

CHAPTER

TWO

EXAMPLES

Refer to the development development demos for examples prior to preparing this section.

A collection of examples are presented to supplement the tutorial. The examples below are identical to the Jupyter Notebook in the `examples` folder of the repository here.

2.1 Simulate

This example gives a "hello world" example to use AMS.

2.1.1 Import and Setting the Verbosity Level

We first import the `ams` library.

```
import ams

import datetime
```

```
print("Last run time:", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

print(f'ams:{ams.__version__}')
```

```
Last run time: 2024-02-24 17:28:40
ams:0.8.5.post62.dev0+gf6ed683
```

We can configure the verbosity level for logging (output messages) by passing a verbosity level (10-DEBUG, 20-INFO, 30-WARNING, 40-ERROR, 50-CRITICAL) to the `stream_level` argument of `ams.main.config_logger()`. Verbose level 10 is useful for getting debug output.

The logging level can be altered by calling `config_logger` again with new `stream_level` and `file_level`.

```
ams.config_logger(stream_level=20)
```

Note that the above `ams.config_logger()` is a shorthand to `ams.main.config_logger()`.

If this step is omitted, the default `INFO` level (`stream_level=20`) will be used.

2.1.2 Run Simulations

Load Case

AMS support multiple input file formats, including AMS `.xlsx` file, MATPOWER `.m` file, PYPOWER `.py` file, and PSS/E `.raw` file.

Here we use the AMS `.xlsx` file as an example. The source file locates at `$HOME/ams/ams/cases/ieee39/ieee39_uced.xlsx`.

```
sp = ams.load(ams.get_case('5bus/pjm5bus_uced.xlsx'),
              setup=True,
              no_output=True,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_uced.
↪xlsx"...
Input file parsed in 0.2701 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0021 seconds.
```

Inspect Models and Routines

In AMS, `model` refers to the device model, and all models are registered to an OrderedDict `models`.

```
sp.models
```

```
OrderedDict([('Summary', Summary (3 devices) at 0x1102ec910),
             ('Bus', Bus (5 devices) at 0x11001f6a0),
             ('PQ', PQ (3 devices) at 0x14cf7a490),
             ('PV', PV (3 devices) at 0x137ae0100),
             ('Slack', Slack (1 device) at 0x14cf7d610),
             ('Shunt', Shunt (0 devices) at 0x14cfa14c0),
             ('Line', Line (7 devices) at 0x14cfa1970),
             ('PVD1', PVD1 (0 devices) at 0x14cfbb0a0),
             ('ESD1', ESD1 (0 devices) at 0x14cfbb6a0),
             ('REGCA1', REGCA1 (0 devices) at 0x14cfbbc10),
             ('REGCV1', REGCV1 (0 devices) at 0x14cfc5250),
             ('REGCV2', REGCV2 (0 devices) at 0x14cfc5a30),
             ('Area', Area (3 devices) at 0x14cfc5f70),
             ('Region', Region (2 devices) at 0x14cfd3730),
             ('SFR', SFR (2 devices) at 0x14cfd3ee0),
             ('SR', SR (2 devices) at 0x14cfde580),
             ('NSR', NSR (2 devices) at 0x14cfde9a0),
             ('VSGR', VSGR (0 devices) at 0x14cfdedc0),
             ('GCost', GCost (4 devices) at 0x15e5de250),
             ('SFRCost', SFRCost (4 devices) at 0x15e5de8e0),
             ('SRCost', SRCost (4 devices) at 0x15e5dee80),
             ('NSRCost', NSRCost (4 devices) at 0x15e5ec2e0),
             ('VSGCost', VSGCost (0 devices) at 0x15e5ec700),
             ('DCost', DCost (3 devices) at 0x15e5eca00),
             ('TimeSlot', TimeSlot (0 devices) at 0x15e5ecf70),
```

(continues on next page)

(continued from previous page)

```
            ('EDTSlot', EDTSlot (24 devices) at 0x15e5f5a30),
            ('UCTSlot', UCTSlot (24 devices) at 0x15e5f5e50)])
```

One can inspect the detailed model data by converting it to a pandas DataFrame.

```
sp.PQ.as_df()
```

```
      idx    u  name    bus     Vn   p0      q0  vmax  vmin owner  ctrl
uid
0    PQ_1  1.0  PQ 1  Bus_2  230.0  3.0  0.9861   1.1   0.9  None   1.0
1    PQ_2  1.0  PQ 2  Bus_3  230.0  3.0  0.9861   1.1   0.9  None   1.0
2    PQ_3  1.0  PQ 3  Bus_4  230.0  4.0  1.3147   1.1   0.9  None   1.0
```

In AMS, all supported routines are registered to an OrderedDict `routines`.

```
sp.routines
```

```
OrderedDict([('DCPF', DCPF at 0x1102eb970),
             ('PFlow', PFlow at 0x15e60d3a0),
             ('CPF', CPF at 0x15e60d910),
             ('ACOPF', ACOPF at 0x15e632a00),
             ('DCOPF', DCOPF at 0x15e64f160),
             ('ED', ED at 0x15e6901c0),
             ('EDDG', EDDG at 0x15e688040),
             ('EDES', EDES at 0x15e60ce50),
             ('RTED', RTED at 0x15e6b8460),
             ('RTEDDG', RTEDDG at 0x15e6b8520),
             ('RTEDES', RTEDES at 0x15e6dbbb0),
             ('RTEDVIS', RTEDVIS at 0x15e6ff940),
             ('UC', UC at 0x162648400),
             ('UCDG', UCDG at 0x16307a490),
             ('UCES', UCES at 0x16309f3d0),
             ('DOPF', DOPF at 0x1630c2dc0),
             ('DOPFVIS', DOPFVIS at 0x1630e60d0)])
```

Solve an Routine

Before solving an routine, we need to initialize it first. Here Real-time Economic Dispatch (RTED) is used as an example.

```
sp.RTED.init()
```

```
Routine <RTED> initialized in 0.0139 seconds.
```

```
True
```

Then, one can solve it by calling `run()`. Here, argument `solver` can be passed to specify the solver to use, such as `solver='ECOS'`.

Installed solvers can be listed by `ams.shared.INSTALLED_SOLVERS`, and more detailes of solver can be found at CVXPY-Choosing a solver.

```
ams.shared.INSTALLED_SOLVERS
```

```
['CLARABEL',
 'CVXOPT',
 'ECOS',
 'ECOS_BB',
 'GLPK',
 'GLPK_MI',
 'GUROBI',
 'MOSEK',
 'OSQP',
 'PIQP',
 'PROXQP',
 'SCIPY',
 'SCS']
```

```
sp.RTED.run(solver='ECOS')
```

```
RTED solved as optimal in 0.0207 seconds, converged after 9 iterations using solver ECOS.
```

```
True
```

The solved results are stored in each variable itself. For example, the solved power generation of ten generators are stored in `pg.v`.

```
sp.RTED.pg.v
```

```
array([2.1, 5.2, 0.7, 2. ])
```

Here, `get_idx()` can be used to get the index of a variable.

```
sp.RTED.pg.get_idx()
```

```
['PV_1', 'PV_3', 'PV_5', 'Slack_4']
```

Part of the solved results can be accessed with given indices.

```
sp.RTED.get(src='pg', attr='v', idx=['PV_1', 'PV_3'])
```

```
array([2.1, 5.2])
```

All Vars are listed in an OrderedDict `vars`.

```
sp.RTED.vars
```

```
OrderedDict([('pg', Var: StaticGen.pg),
             ('aBus', Var: Bus.aBus),
             ('plf', Var: Line.plf),
             ('pru', Var: StaticGen.pru),
             ('prd', Var: StaticGen.prd)])
```

The Objective value can be accessed with `obj.v`.

```
sp.RTED.obj.v
```

```
0.19537500005072062
```

Similarly, all Constrs are listed in an OrderedDict `constrs`, and the expression values can also be accessed.

```
sp.RTED.constrs
```

```
OrderedDict([('pglb', Constraint: pglb [ON]),
             ('pgub', Constraint: pgub [ON]),
             ('pb', Constraint: pb [ON]),
             ('plflb', Constraint: plflb [ON]),
             ('plfub', Constraint: plfub [ON]),
             ('alflb', Constraint: alflb [ON]),
             ('alfub', Constraint: alfub [ON]),
             ('rbu', Constraint: rbu [ON]),
             ('rbd', Constraint: rbd [ON]),
             ('rru', Constraint: rru [ON]),
             ('rrd', Constraint: rrd [ON]),
             ('rgu', Constraint: rgu [ON]),
             ('rgd', Constraint: rgd [ON])])
```

One can also inspect the Constr values.

```
sp.RTED.rgu.v
```

```
array([-996.9, -993.8, -998.3, -997. ])
```

2.2 Manipulate the Simulation

This example shows how to play with the simulation, such as contingency analysis and manipulate the constraints.

```
import ams

import datetime
```

```
print("Last run time:", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

print(f'ams:{ams.__version__}')
```

```
Last run time: 2024-02-24 17:28:53
ams:0.8.5.post62.dev0+gf6ed683
```

```
ams.config_logger(stream_level=20)
```

2.2.1 Manipulate the Simulation

Load Case

```
sp = ams.load(ams.get_case('5bus/pjm5bus_uced.xlsx'),
              setup=True,
              no_output=True,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_uced.
↪xlsx"...
Input file parsed in 0.1465 seconds.
Zero line rates detected in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0027 seconds.
```

The system load are defined in model PQ.

```
sp.PQ.as_df()
```

```
      idx    u  name    bus     Vn   p0      q0  vmax  vmin owner  ctrl
uid
0    PQ_1  1.0  PQ 1  Bus_2  230.0  3.0  0.9861   1.1   0.9  None   1.0
1    PQ_2  1.0  PQ 2  Bus_3  230.0  3.0  0.9861   1.1   0.9  None   1.0
2    PQ_3  1.0  PQ 3  Bus_4  230.0  4.0  1.3147   1.1   0.9  None   1.0
```

In RTED, system load is referred as pd.

```
sp.RTED.pd.v
```

```
array([3., 3., 4.])
```

Run Simulation

RTED can be solved and one can inspect the results as discussed in previous example.

```
sp.RTED.run(solver='ECOS')
```

```
Routine <RTED> initialized in 0.0166 seconds.
RTED solved as optimal in 0.0212 seconds, converged after 9 iterations using solver ECOS.
```

```
True
```

Power generation pg and line flow `plf` can be accessed as follows.

```
sp.RTED.pg.v
```

```
array([2.1, 5.2, 0.7, 2. ])
```

```
sp.RTED.plf.v
```

```
array([ 0.70595331,  0.68616798,  0.00192539, -1.58809337,  0.61190663,
       -0.70192539,  0.70595331])
```

Change Load

The load values can be manipulated in the model PQ.

```
sp.PQ.set(src='p0', attr='v', idx=['PQ_1', 'PQ_2'], value=[3.2, 3.2])
```

```
True
```

According parameters need to be updated to make the changes effective in the optimization model. If not sure which parameters need to be updated, one can use `update()` to update all parameters.

```
sp.RTED.update('pd')
```

```
True
```

After manipulation, the routined can be solved again.

```
sp.RTED.run(solver='ECOS')
```

```
RTED solved as optimal in 0.0031 seconds, converged after 9 iterations using solver ECOS.
```

```
True
```

```
sp.RTED.pg.v
```

```
array([2.1, 5.2, 1.1, 2. ])
```

An alternative way is to alter the load through RTED.

As pd has owner `StaticLoad` and soruce `p0`, the parameter update through RTED actually happens to `StaticLoad.p0`.

```
sp.RTED.pd.owner
```

```
StaticLoad (3 devices) at 0x15a4e41f0
```

```
sp.RTED.pd.src
```

```
'p0'
```

Similarly, the load can be changed using `set` method.

```
sp.RTED.set(src='pd', attr='v', idx=['PQ_1', 'PQ_2'], value=[3.8, 3.8])
```

```
True
```

Remember to update the optimization parameters after the change.

```
sp.RTED.update('pd')
```

```
True
```

We can see that the original load is also updated.

```
sp.PQ.as_df()
```

```
      idx    u  name    bus     Vn   p0      q0  vmax  vmin owner  ctrl
uid
0    PQ_1  1.0  PQ 1  Bus_2  230.0  3.8  0.9861   1.1   0.9  None   1.0
1    PQ_2  1.0  PQ 2  Bus_3  230.0  3.8  0.9861   1.1   0.9  None   1.0
2    PQ_3  1.0  PQ 3  Bus_4  230.0  4.0  1.3147   1.1   0.9  None   1.0
```

```
sp.RTED.run(solver='ECOS')
```

```
RTED solved as optimal in 0.0059 seconds, converged after 9 iterations using solver ECOS.
```

```
True
```

As expected, the power generation also changed.

```
sp.RTED.pg.v
```

```
array([2.1, 5.2, 2.3, 2. ])
```

Trip a Generator

We can see that there are three PV generators in the system.

```
sp.PV.as_df()
```

```
      idx    u      name     Sn     Vn    bus  busr      p0   q0  pmax  ...  \
uid                                                                    ...
0    PV_1  1.0      Alta  100.0  230.0  Bus_1  None  1.0000  0.0   2.1  ...
1    PV_3  1.0  Solitude  100.0  230.0  Bus_3  None  3.2349  0.0   5.2  ...
2    PV_5  1.0  Brighton  100.0  230.0  Bus_5  None  4.6651  0.0   6.0  ...

     Qc2min  Qc2max   Ragc    R10    R30     Rq  apf  pg0  td1  td2
uid
0       0.0     0.0  999.0  999.0  999.0  999.0  0.0  0.0  0.5  0.0
1       0.0     0.0  999.0  999.0  999.0  999.0  0.0  0.0  0.5  0.0
2       0.0     0.0  999.0  999.0  999.0  999.0  0.0  0.0  0.5  0.0

[3 rows x 33 columns]
```

PV_1 is tripped by setting its connection status u to 0.

```
sp.StaticGen.set(src='u', attr='v', idx='PV_1', value=0)
```

```
True
```

In AMS, some parameters are defiend as constants in the numerical optimization model to follow the CVXPY DCP and DPP rules. Once non-parametric parameters are changed, the optimization model will be re-initialized to make the changes effective.

More details can be found at CVXPY - Disciplined Convex Programming.

```
sp.RTED.update()
```

```
Re-init RTED OModel due to non-parametric change.
```

```
True
```

Then we can re-solve the model.

```
sp.RTED.run(solver='ECOS')
```

```
RTED solved as optimal in 0.0159 seconds, converged after 8 iterations using solver ECOS.
```

```
True
```

We can see that the tripped generator has no power generation.

```
sp.RTED.pg.v.round(2)
```

```
array([-0. ,  5.2,  4.4,  2. ])
```

Trip a Line

We can inspect the Line model to check the system topology.

```
sp.Line.as_df()
```

```
        idx    u      name   bus1   bus2     Sn    fn    Vn1    Vn2        r  \
uid
0    Line_0  1.0   Line AB  Bus_1  Bus_2  100.0  60.0  230.0  230.0  0.00281
1    Line_1  1.0   Line AD  Bus_1  Bus_4  100.0  60.0  230.0  230.0  0.00304
2    Line_2  1.0   Line AE  Bus_1  Bus_5  100.0  60.0  230.0  230.0  0.00064
3    Line_3  1.0   Line BC  Bus_2  Bus_3  100.0  60.0  230.0  230.0  0.00108
4    Line_4  1.0   Line CD  Bus_3  Bus_4  100.0  60.0  230.0  230.0  0.00297
5    Line_5  1.0   Line DE  Bus_4  Bus_5  100.0  60.0  230.0  230.0  0.00297
6    Line_6  1.0  Line AB2  Bus_1  Bus_2  100.0  60.0  230.0  230.0  0.00281

     ...  tap  phi  rate_a  rate_b  rate_c  owner  xcoord  ycoord      amin  \
uid  ...
0    ...  1.0  0.0     4.0   999.0   999.0   None    None    None -6.283185
1    ...  1.0  0.0   999.0   999.0   999.0   None    None    None -6.283185
2    ...  1.0  0.0   999.0   999.0   999.0   None    None    None -6.283185
3    ...  1.0  0.0   999.0   999.0   999.0   None    None    None -6.283185
4    ...  1.0  0.0   999.0   999.0   999.0   None    None    None -6.283185
```

(continues on next page)

(continued from previous page)

```
5    ...  1.0  0.0     2.4   999.0   999.0   None    None    None -6.283185
6    ...  1.0  0.0     4.0   999.0   999.0   None    None    None -6.283185

         amax
uid
0    6.283185
1    6.283185
2    6.283185
3    6.283185
4    6.283185
5    6.283185
6    6.283185

[7 rows x 28 columns]
```

Here line 2 is tripped by setting its connection status `u` to 0.

Note that in ANDES, dynamic simulation of *line tripping should use model* `Toggle`.

```
sp.Line.set(src='u', attr='v', idx='Line_1', value=0)
```

```
True
```

```
sp.RTED.update()
```

```
Re-init RTED OModel due to non-parametric change.
```

```
True
```

```
sp.RTED.run(solver='ECOS')
```

```
RTED solved as optimal in 0.0182 seconds, converged after 8 iterations using solver ECOS.
```

```
True
```

Here we can see the tripped line has no flow.

```
sp.RTED.plf.v.round(2)
```

```
array([ 1.34,  0.  , -2.68, -1.12,  0.28, -1.72,  1.34])
```

2.2.2 Disable the Constraints

In addition to the system parameters, the constraints can also be manipulated.

Here, we load the case to a new system.

```
spc = ams.load(ams.get_case('5bus/pjm5bus_uced.xlsx'),
               setup=True,
               no_output=True,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_uced.
↪xlsx"...
Input file parsed in 0.0413 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0021 seconds.
```

```
spc.RTED.init()
```

```
Routine <RTED> initialized in 0.0091 seconds.
```

```
True
```

```
spc.RTED.set(src='rate_a', attr='v', idx=['Line_2'], value=1.4)
```

```
True
```

```
spc.RTED.update('rate_a')
```

```
True
```

We can inspect the constraints status as follows. All constraints are turned on by default.

```
spc.RTED.constrs
```

```
OrderedDict([('pglb', Constraint: pglb [ON]),
             ('pgub', Constraint: pgub [ON]),
             ('pb', Constraint: pb [ON]),
             ('plflb', Constraint: plflb [ON]),
             ('plfub', Constraint: plfub [ON]),
             ('alflb', Constraint: alflb [ON]),
             ('alfub', Constraint: alfub [ON]),
             ('rbu', Constraint: rbu [ON]),
             ('rbd', Constraint: rbd [ON]),
             ('rru', Constraint: rru [ON]),
             ('rrd', Constraint: rrd [ON]),
             ('rgu', Constraint: rgu [ON]),
             ('rgd', Constraint: rgd [ON])])
```

Then, solve the dispatch and inspect the line flow.

```
spc.RTED.run(solver='ECOS')
```

```
RTED solved as optimal in 0.0143 seconds, converged after 10 iterations using solver␣
↪ECOS.
```

```
True
```

```
spc.RTED.plf.v.round(2)
```

```
array([ 0.71,  0.69,  0.  , -1.59,  0.61, -0.7 ,  0.71])
```

In the next, we can disable specific constraints, and the parameter name takes both single constraint name or a list of constraint names.

```
spc.RTED.disable(['plflb', 'plfub'])
```

```
Turn off constraints: plflb, plfub
```

```
True
```

Now, it can be seen that the two constraints are disabled.

```
spc.RTED.constrs
```

```
OrderedDict([('pglb', Constraint: pglb [ON]),
             ('pgub', Constraint: pgub [ON]),
             ('pb', Constraint: pb [ON]),
             ('plflb', Constraint: plflb [OFF]),
             ('plfub', Constraint: plfub [OFF]),
             ('alflb', Constraint: alflb [ON]),
             ('alfub', Constraint: alfub [ON]),
             ('rbu', Constraint: rbu [ON]),
             ('rbd', Constraint: rbd [ON]),
             ('rru', Constraint: rru [ON]),
             ('rrd', Constraint: rrd [ON]),
             ('rgu', Constraint: rgu [ON]),
             ('rgd', Constraint: rgd [ON])])
```

```
spc.RTED.run(solver='ECOS')
```

```
Disabled constraints: plflb, plfub
Routine <RTED> initialized in 0.0087 seconds.
RTED solved as optimal in 0.0146 seconds, converged after 9 iterations using solver ECOS.
```

```
True
```

We can see that the line flow limits are not in effect.

```
spc.RTED.plf.v.round(2)
```

```
array([ 0.71,  0.69,  0.  , -1.59,  0.61, -0.7 ,  0.71])
```

Similarly, you can also enable the constraints again.

```
spc.RTED.enable(['plflb', 'plfub'])
```

```
Turn on constraints: plflb, plfub
```

```
True
```

```
spc.RTED.constrs
```

```
OrderedDict([('pglb', Constraint: pglb [ON]),
             ('pgub', Constraint: pgub [ON]),
             ('pb', Constraint: pb [ON]),
             ('plflb', Constraint: plflb [ON]),
             ('plfub', Constraint: plfub [ON]),
             ('alflb', Constraint: alflb [ON]),
             ('alfub', Constraint: alfub [ON]),
             ('rbu', Constraint: rbu [ON]),
             ('rbd', Constraint: rbd [ON]),
             ('rru', Constraint: rru [ON]),
             ('rrd', Constraint: rrd [ON]),
             ('rgu', Constraint: rgu [ON]),
             ('rgd', Constraint: rgd [ON])])
```

```
spc.RTED.run(solver='ECOS')
```

```
Routine <RTED> initialized in 0.0094 seconds.
RTED solved as optimal in 0.0163 seconds, converged after 10 iterations using solver␣
↪ECOS.
```

```
True
```

```
spc.RTED.plf.v.round(2)
```

```
array([ 0.71,  0.69,  0.  , -1.59,  0.61, -0.7 ,  0.71])
```

Alternatively, you can also force init the dispatch to rebuild the system matrices, enable all constraints, and re-init the optimization models.

```
spc.RTED.disable(['plflb', 'plfub', 'rgu', 'rgd'])
```

```
Turn off constraints: plflb, plfub, rgu, rgd
```

```
True
```

```
spc.RTED.init(force=True)
```

```
Routine <RTED> initialized in 0.0094 seconds.
```

```
True
```

```
spc.RTED.constrs
```

```
OrderedDict([('pglb', Constraint: pglb [ON]),
             ('pgub', Constraint: pgub [ON]),
             ('pb', Constraint: pb [ON]),
             ('plflb', Constraint: plflb [ON]),
             ('plfub', Constraint: plfub [ON]),
             ('alflb', Constraint: alflb [ON]),
             ('alfub', Constraint: alfub [ON]),
             ('rbu', Constraint: rbu [ON]),
             ('rbd', Constraint: rbd [ON]),
             ('rru', Constraint: rru [ON]),
             ('rrd', Constraint: rrd [ON]),
             ('rgu', Constraint: rgu [ON]),
             ('rgd', Constraint: rgd [ON])])
```

2.2.3 Alter the Config

In AMS, routines have an `config` object as configuration settings.

```
spf = ams.load(ams.get_case('5bus/pjm5bus_uced.xlsx'),
               setup=True,
               no_output=True,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_uced.
↪xlsx"...
Input file parsed in 0.0951 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0024 seconds.
```

In RTED, the default interval is 5/60 [hour], and the formulations has been adjusted to fit the interval.

```
spf.RTED.config
```

```
OrderedDict([('t', 0.08333333333333333)])
```

```
spf.RTED.run(solver='ECOS')
```

```
Routine <RTED> initialized in 0.0113 seconds.
RTED solved as optimal in 0.0155 seconds, converged after 9 iterations using solver ECOS.
```

```
True
```

```
spf.RTED.obj.v
```

```
0.19537500005072062
```

We can update the interval to 1 [hour] and re-solve the dispatch.

Note that in this senario, compared to DCOPF, RTED has extra costs for `pru` and `prd`.

```
spf.RTED.config.t = 60/60
```

Remember to update the parameters after the change.

```
spf.RTED.update()
```

```
Re-init RTED OModel due to non-parametric change.
```

```
True
```

```
spf.RTED.run(solver='SCS')
```

```
RTED solved as optimal in 0.0181 seconds, converged after 325 iterations using solver␣
↪SCS.
```

```
True
```

We can then get the objective value.

```
spf.RTED.obj.v
```

```
2.3444999986498134
```

Note that in this build-in case, the `cru` and `crd` are defined as zero.

```
spf.RTED.cru.v
```

```
array([0., 0., 0., 0.])
```

```
spf.RTED.crd.v
```

```
array([0., 0., 0., 0.])
```

As benchmark, we can solve the DCOPF.

```
spf.DCOPF.run(solver='SCS')
```

```
Routine <DCOPF> initialized in 0.0048 seconds.
DCOPF solved as optimal in 0.0101 seconds, converged after 225 iterations using solver␣
↪SCS.
```

```
True
```

As expected, the DCOPF has a similar objective value.

```
spf.DCOPF.obj.v
```

```
2.3445094955490013
```

2.3 Inspecting Models

We first import the ams library and configure the logger level.

```
import ams

import datetime
```

```
print("Last run time:", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

print(f'ams:{ams.__version__}')
```

```
Last run time: 2024-02-24 17:29:03
ams:0.8.5.post62.dev0+gf6ed683
```

```
ams.config_logger(stream_level=20)
```

Load an example case.

```
sp = ams.load(ams.get_case('5bus/pjm5bus_uced.xlsx'),
              setup=True,
              no_output=True,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_uced.
↪xlsx"...
Input file parsed in 0.1415 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0022 seconds.
```

2.3.1 List all models and routines

```
print(sp.supported_models())
```

```
Supported Groups and Models

   Group     |              Models
-------------+------------------------------
 ACLine      | Line
 ACTopology  | Bus
 Collection  | Area, Region
 Cost        | GCost, SFRCost, VSGCost, DCost
 DG          | PVD1, ESD1
```

(continues on next page)

(continued from previous page)

```
 Horizon     | TimeSlot, EDTSlot, UCTSlot
 Information | Summary
 RenGen      | REGCA1
 Reserve     | SFR, SR, NSR, VSGR
 StaticGen   | PV, Slack
 StaticLoad  | PQ
 StaticShunt | Shunt
 Undefined   | SRCost, NSRCost
 VSG         | REGCV1, REGCV2
```

Similarly, all supported routiens can be listed.

```
print(sp.supported_routines())
```

```
Supported Types and Routines

Type  |                        Routines
------+-------------------------------------------------------
 ACED | ACOPF
 DCED | DCOPF, ED, EDDG, EDES, RTED, RTEDDG, RTEDES, RTEDVIS
 DCUC | UC, UCDG, UCES
 DED  | DOPF, DOPFVIS
 PF   | DCPF, PFlow, CPF
```

2.3.2 Check routine documentation

To check the documentation for the routine model, use its `doc()` method.

```
print(sp.RTED.doc())
```

```
Routine <RTED> in Type <DCED>
DC-based real-time economic dispatch (RTED).
RTED extends DCOPF with:

- Mapping dicts to interface with ANDES
- Function ``dc2ac`` to do the AC conversion
- Vars for SFR reserve: ``pru`` and ``prd``
- Param for linear SFR cost: ``cru`` and ``crd``
- Param for SFR requirement: ``du`` and ``dd``
- Param for ramping: start point ``pg0`` and ramping limit ``R10``
- Param ``pg0``, which can be retrieved from dynamic simulation results.

The function ``dc2ac`` sets the ``vBus`` value from solved ACOPF.
Without this conversion, dynamic simulation might fail due to the gap between
DC-based dispatch results and AC-based dynamic initialization.

Notes
-----
1. Formulations has been adjusted with interval ``config.t``, 5/60 [Hour] by default.

```

(continues on next page)

(continued from previous page)

```
2. The tie-line flow has not been implemented in formulations.

Objective

Name | Unit |            Description
-----+------+----------------------------------
 obj | $    | total generation and reserve cost

Constraints

 Name  |            Description
-------+----------------------------------
 pglb  | pg min
 pgub  | pg max
 pb    | power balance
 plflb | line flow lower bound
 plfub | line flow upper bound
 alflb | line angle difference lower bound
 alfub | line angle difference upper bound
 rbu   | RegUp reserve balance
 rbd   | RegDn reserve balance
 rru   | RegUp reserve source
 rrd   | RegDn reserve source
 rgu   | Gen ramping up
 rgd   | Gen ramping down

Vars

Name  |    Description    | Unit | Properties
------+-------------------+------+-----------
 pg   | Gen active power  | p.u. |
 aBus | Bus voltage angle | rad  |
 plf  | Line flow         | p.u. |
 pru  | RegUp reserve     | p.u. | nonneg
 prd  | RegDn reserve     | p.u. | nonneg

Services

 Name   |              Description              |   Type
--------+---------------------------------------+----------
 ctrle  | Effective Gen controllability         | NumOpDual
 nctrl  | Effective Gen uncontrollability       | NumOp
 nctrle | Effective Gen uncontrollability       | NumOpDual
 amax   | max line angle difference             | NumOp
 gs     | Sum Gen vars vector in shape of zone | ZonalSum
 ds     | Sum pd vector in shape of zone        | ZonalSum
 pdz    | zonal total load                      | NumOpDual
 dud    | zonal RegUp reserve requirement       | NumOpDual
 ddd    | zonal RegDn reserve requirement       | NumOpDual

Parameters
```

(continues on next page)

(continued from previous page)

```
 Name    |                Description                 |    Unit
---------+--------------------------------------------+-----------
 c2      | Gen cost coefficient 2                     | $/(p.u.^2)
 c1      | Gen cost coefficient 1                     | $/(p.u.)
 c0      | Gen cost coefficient 0                     | $
 ug      | Gen connection status                      |
 ctrl    | Gen controllability                        |
 pmax    | Gen maximum active power                   | p.u.
 pmin    | Gen minimum active power                   | p.u.
 p0      | Gen initial active power                   | p.u.
 pd      | active demand                              | p.u.
 rate_a  | long-term flow limit                       | MVA
 gsh     | shunt conductance                          |
 Cg      | Gen connection matrix                      |
 Cl      | Load connection matrix                     |
 CftT    | Transpose of line connection matrix        |
 Csh     | Shunt connection matrix                    |
 Bbus    | Bus admittance matrix                      |
 Bf      | Bf matrix                                  |
 Pbusinj | Bus power injection vector                 |
 Pfinj   | Line power injection vector                |
 zg      | Gen zone                                   |
 zd      | Load zone                                  |
 R10     | 10-min ramp rate                           | p.u./h
 cru     | RegUp reserve coefficient                  | $/(p.u.)
 crd     | RegDown reserve coefficient                | $/(p.u.)
 du      | RegUp reserve requirement in percentage    | %
 dd      | RegDown reserve requirement in percentage | %
```

2.3.3 Data Check

The `summary()` method gives a brief summary of the system and routiens that passed the data check.

```
sp.summary()
```

```
-> Systen size:
Base: 100 MVA; Frequency: 60 Hz
5 Buses; 7 Lines; 4 Static Generators
Active load: 10.00 p.u.; Reactive load: 3.29 p.u.
-> Data check results:
ACED: ACOPF
DCED: DCOPF, ED, RTED
DCUC: UC
DED: DOPF
PF: DCPF, PFlow, CPF
```

2.4 Case I/O

AMS supprots multiple case formats.

Still, first import the ams library and configure the logger level.

```
import os

import ams

import datetime
```

```
print("Last run time:", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

print(f'ams:{ams.__version__}')
```

```
Last run time: 2024-02-24 17:29:23
ams:0.8.5.post62.dev0+gf6ed683
```

```
ams.config_logger(stream_level=20)
```

2.4.1 Input

AMS Execel

```
sp_xlsx = ams.load(ams.get_case('ieee14/ieee14_uced.xlsx'),
                   setup=True,
                   no_output=True,)

sp_xlsx.summary()
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/ieee14/ieee14_uced.
↪xlsx"...
Input file parsed in 0.1440 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0043 seconds.
-> Systen size:
Base: 100 MVA; Frequency: 60 Hz
14 Buses; 20 Lines; 5 Static Generators
Active load: 2.24 p.u.; Reactive load: 0.95 p.u.
-> Data check results:
ACED: ACOPF
DCED: DCOPF, ED, RTED
DCUC: UC
DED: DOPF
PF: DCPF, PFlow, CPF
```

AMS JSON

```
sp_json = ams.load(ams.get_case('ieee14/ieee14.json'),
                   setup=True,
                   no_output=True,)

sp_json.summary()
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/ieee14/ieee14.json"..
↪.
Input file parsed in 0.0021 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
```

```
System set up in 0.0040 seconds.
-> Systen size:
Base: 100 MVA; Frequency: 60 Hz
14 Buses; 20 Lines; 5 Static Generators
Active load: 2.24 p.u.; Reactive load: 0.95 p.u.
-> Data check results:
PF: DCPF, PFlow, CPF
```

MATPOWER

```
sp_mp = ams.load(ams.get_case('matpower/case14.m'),
                 setup=True,
                 no_output=True,)

sp_mp.summary()
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/matpower/case14.m"...
Input file parsed in 0.0071 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0028 seconds.
-> Systen size:
Base: 100.0 MVA; Frequency: 60 Hz
14 Buses; 20 Lines; 5 Static Generators
Active load: 2.59 p.u.; Reactive load: 0.74 p.u.
-> Data check results:
ACED: ACOPF
DCED: DCOPF
DED: DOPF
PF: DCPF, PFlow, CPF
```

Note that AMS also supports PYPOWER format py-file.

PSS/E RAW

AMS also supports PSS/E RAW format for power flow analysis.

```
sp_raw = ams.load(ams.get_case('ieee14/ieee14.raw'),
                  setup=True,
                  no_output=True,)

sp_raw.summary()
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/ieee14/ieee14.raw"...
Input file parsed in 0.0102 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0030 seconds.
-> Systen size:
Base: 100.0 MVA; Frequency: 60.0 Hz
14 Buses; 20 Lines; 5 Static Generators
Active load: 2.24 p.u.; Reactive load: 0.95 p.u.
-> Data check results:
PF: DCPF, PFlow, CPF
```

2.4.2 Output

Vice versa, AMS supports multiple output formats.

```
ams.io.xlsx.write(system=sp_xlsx,
                  outfile='out.xlsx',)
```

```
xlsx file written to "out.xlsx"
```

```
True
```

```
os.remove('out.xlsx')
```

Similarly, JSON output formats can be achieved by using `ams.io.json.write`.

2.5 Interoperation with ANDES

One of the most interesting feature of AMS is its interoperation with dynamic simulator ANDES.

Interoperation includes compatible case conversion and data exchange, thus it facilitates dispatch-dynamic co-simulation using AMS and ANDES.

```
import numpy as np

import andes
import ams

import datetime
```

```
print("Last run time:", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

print(f'andes:{andes.__version__}')
print(f'ams:{ams.__version__}')
```

```
Last run time: 2024-02-24 17:29:32
andes:1.9.0
ams:0.8.5.post62.dev0+gf6ed683
```

```
ams.config_logger(stream_level=20)
```

2.5.1 Dispatch

```
sp = ams.load(ams.get_case('ieee14/ieee14_uced.xlsx'),
              setup=True,
              no_output=True,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/ieee14/ieee14_uced.
↪xlsx"...
Input file parsed in 0.1225 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0042 seconds.
```

```
sp.RTED.init()
```

```
Routine <RTED> initialized in 0.0171 seconds.
```

```
True
```

```
sp.RTED.run(solver='ECOS')
```

```
RTED solved as optimal in 0.0245 seconds, converged after 11 iterations using solver␣
↪ECOS.
```

```
True
```

2.5.2 Convert to ANDES

The built-in ANDES interface can convert an AMS case to ANDES case in memory.

The bridge between AMS and converted ANDES is the shared power flow devices, Bus, PQ, PV, Slack, Line, and Shunt.

```
sa = sp.to_andes(setup=True,
                 addfile=andes.get_case('ieee14/ieee14_full.xlsx'))
```

```
Parsing additional file "/Users/jinningwang/Documents/work/mambaforge/envs/ams/lib/
↪python3.9/site-packages/andes/cases/ieee14/ieee14_full.xlsx"...
Following PFlow models in addfile will be overwritten: <Bus>, <PQ>, <PV>, <Slack>,
↪<Shunt>, <Line>, <Area>
Addfile parsed in 0.0519 seconds.
System converted to ANDES in 0.1987 seconds.
AMS system 0x10a441af0 is linked to the ANDES system 0x17f0720a0.
```

If you wish to add devices to the converted ANDES system, set `setup=False` to skip the ANDES setup process.

As indicated by the output information, in the conversion process, ANDES power flow devices will be overwritten by AMS ones, if exists.

Upon a successful conversion, you are ready to enjoy full capability of ANDES.

`help` command can give a quick reference.

```
help(sp.to_andes)
```

```
Help on method to_andes in module ams.system:

to_andes(setup=True, addfile=None, **kwargs) method of ams.system.System instance
    Convert the AMS system to an ANDES system.

    A preferred dynamic system file to be added has following features:
    1. The file contains both power flow and dynamic models.
    2. The file can run in ANDES natively.
    3. Power flow models are in the same shape as the AMS system.
    4. Dynamic models, if any, are in the same shape as the AMS system.

    Parameters
    ----------
    setup : bool, optional
        Whether to call `setup()` after the conversion. Default is True.
    addfile : str, optional
        The additional file to be converted to ANDES dynamic mdoels.
    **kwargs : dict
        Keyword arguments to be passed to `andes.system.System`.

    Returns
    -------
    andes : andes.system.System
        The converted ANDES system.

    Examples
    --------
    >>> import ams
    >>> import andes
    >>> sp = ams.load(ams.get_case('ieee14/ieee14_rted.xlsx'), setup=True)
    >>> sa = sp.to_andes(setup=False,
    ...                  addfile=andes.get_case('ieee14/ieee14_wt3.xlsx'),
    ...                  overwrite=True, no_keep=True, no_output=True)
```

2.5.3 Interoperation with ANDES

In the interface class `dyn`, the link table is stored in `dyn.link`.

It describes the mapping relationships between power flow devices and dynamic devices.

```
sp.dyn.link
```

```
   stg_idx bus_idx   syg_idx  gov_idx dg_idx rg_idx  gammap  gammaq
0  Slack_1       1  GENROU_1  TGOV1_1    NaN    NaN     1.0     1.0
1     PV_5       8  GENROU_5  TGOV1_5    NaN    NaN     1.0     1.0
2     PV_4       6  GENROU_4  TGOV1_4    NaN    NaN     1.0     1.0
3     PV_3       3  GENROU_3  TGOV1_3    NaN    NaN     1.0     1.0
4     PV_2       2  GENROU_2  TGOV1_2    NaN    NaN     1.0     1.0
```

Send

As there is a gap between DC-based dispatch and AC-based TDS, a conversion is required to ensure the TDS initialization.

```
sp.RTED.dc2ac()
```

```
Routine <ACOPF> initialized in 0.0042 seconds.
ACOPF solved in 0.2784 seconds, converged after 12 iterations using solver PYPOWER-PIPS.
Attribute <aBus> already exists in <RTED>.
<RTED> is converted to AC.
```

```
True
```

In the RTED routine, there are two mapping dictionaries to define the data exchange, namely, `map1` for receiving data from ANDES and `map2` for sending data to ANDES.

```
sp.RTED.map2
```

```
OrderedDict([('vBus', ('Bus', 'v0')),
             ('ug', ('StaticGen', 'u')),
             ('pg', ('StaticGen', 'p0'))])
```

```
sp.dyn.send(adsys=sa, routine='RTED')
```

```
Send <RTED> results to ANDES <0x17f0720a0>...
Send <vBus> to Bus.v0
Send <ug> to StaticGen.u
Send <pg> to StaticGen.p0
```

```
True
```

Run ANDES

Sometimes, the ANDES TDS initialization may fail due to inapproriate limits.

Here, we alleviate the `TGOV1` limit issue by enlarging the `Pmax` and `Pmin` to the same value.

```
sa.TGOV1.set(src='VMAX', attr='v', idx=sa.TGOV1.idx.v, value=100*np.ones(sa.TGOV1.n))
sa.TGOV1.set(src='VMIN', attr='v', idx=sa.TGOV1.idx.v, value=np.zeros(sa.TGOV1.n))
```

```
True
```

Run power flow.

```
sa.PFlow.run()
```

```
True
```

Try to init TDS.

```
_ = sa.TDS.init()
```

Run TDS.

```
sa.TDS.config.no_tqdm = True # disable progress bar
sa.TDS.run()
```

```
True
```

Receive

```
sp.RTED.map1
```

```
OrderedDict([('ug', ('StaticGen', 'u')), ('pg0', ('StaticGen', 'p'))])
```

```
sp.dyn.receive(adsys=sa, routine='RTED')
```

```
Receive <ug> from SynGen.u
Receive <pg0> from SynGen.Pe
```

```
True
```

The RTED parameter `pg0`, is retrieved from ANDES as the corresponding generator output power.

```
sp.RTED.pg0.v
```

```
array([0.32260084, 0.01      , 0.02      , 0.01      , 1.97393997])
```

2.6 Multi-period Dispatch Simulation

Multi-period dispatch economic dispatch (ED) and unit commitment (UC) is also available.

In this case, we will show a 24-hour ED simulation.

```
import ams

import datetime
```

```
print("Last run time:", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

print(f'ams:{ams.__version__}')
```

```
Last run time: 2024-02-24 17:29:42
ams:0.8.5.post62.dev0+gf6ed683
```

```
ams.config_logger(stream_level=20)
```

2.6.1 Load Case

```
sp = ams.load(ams.get_case('5bus/pjm5bus_demo.xlsx'),
              setup=True,
              no_output=True,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_demo.
↪xlsx"...
Input file parsed in 0.1202 seconds.
Zero line rates detected in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0022 seconds.
```

2.6.2 Reginonal Design

The disaptch models in AMS has develoepd with regional structure, and it can be inspected in device `Region`.

```
sp.Region.as_df()
```

uid	idx	u	name
0	ZONE_1	1.0	ZONE 1
1	ZONE_2	1.0	ZONE 2

In device `Bus`, the Param `zone` indicates the zone of the bus. Correspondingly, the region of generator and load are determined by the bus they connected.

```
sp.Bus.as_df()
```

```
      idx    u name     Vn  vmax  vmin   v0   a0  xcoord  ycoord  area  \
uid
0    Bus_1  1.0    A  230.0   1.1   0.9  1.0  0.0       0       0     1
1    Bus_2  1.0    B  230.0   1.1   0.9  1.0  0.0       0       0     1
2    Bus_3  1.0    C  230.0   1.1   0.9  1.0  0.0       0       0     2
3    Bus_4  1.0    D  230.0   1.1   0.9  1.0  0.0       0       0     2
4    Bus_5  1.0    E  230.0   1.1   0.9  1.0  0.0       0       0     3

       zone owner
uid
0    ZONE_1  None
1    ZONE_1  None
2    ZONE_1  None
3    ZONE_1  None
4    ZONE_1  None
```

2.6.3 Multi-period Dispatch Base

In AMS, multi-period dispatch involves devices in group `Horizon`. This group is developed to provide time-series data for multi-period dispatch.

```
sp.Horizon.models
```

```
OrderedDict([('TimeSlot', TimeSlot (0 devices) at 0x15b19ee20),
             ('EDTSlot', EDTSlot (6 devices) at 0x15b1a58e0),
             ('UCTSlot', UCTSlot (6 devices) at 0x15b1a5d00)])
```

We can get the idx of StaticGens.

```
sp.StaticGen.get_idx()
```

```
['PV_1', 'PV_3', 'PV_5', 'Slack_4']
```

In `EDTSlot`, Param `sd` refers the load factors of each region in each time slot, and Param `ug` represents the generator commitment status in each time slot.

To be more specific, EDT1 has `sd=0.0793,0.0`, which means the load factor of region 1 is 0.0793 in the first time slot, and 0.0 in the second time slot.

Next, EDT1 has `ug=1,1,1,1`, and it means the commitment status of generator PV_1, PV_3, PV_5, and Slack_4 are all online.

```
sp.EDTSlot.as_df()
```

```
     idx    u  name         sd       ug
uid
0    EDT1  1.0  EDT1  0.793,0.0  1,1,1,1
1    EDT2  1.0  EDT2  0.756,0.0  1,1,1,1
2    EDT3  1.0  EDT3  0.723,0.0  1,1,1,1
3    EDT4  1.0  EDT4  0.708,0.0  1,1,1,1
4    EDT5  1.0  EDT5    0.7,0.0  1,1,1,1
5    EDT6  1.0  EDT6  0.706,0.0  1,1,1,1
```

2.6.4 Solve and Result

```
sp.ED.init()
```

```
Routine <ED> initialized in 0.0288 seconds.
```

```
True
```

```
sp.ED.run(solver='ECOS')
```

```
ED solved as optimal in 0.0305 seconds, converged after 9 iterations using solver ECOS.
```

```
True
```

All decision variables are collected in the dict `vars`.

```
sp.ED.vars
```

```
OrderedDict([('pg', Var: StaticGen.pg),
             ('aBus', Var: Bus.aBus),
             ('plf', Var: Line.plf),
             ('pru', Var: StaticGen.pru),
             ('prd', Var: StaticGen.prd),
             ('prs', Var: StaticGen.prs)])
```

As we can see, the generator output `pg` is a 2D array, and the first dimension is the generator index, and the second dimension is the time slot.

```
sp.ED.pg.v
```

```
array([[2.1 , 2.1 , 2.1 , 2.1 , 2.1 , 2.1 ],
       [3.23, 2.86, 2.53, 2.38, 2.3 , 2.36],
       [0.6 , 0.6 , 0.6 , 0.6 , 0.6 , 0.6 ],
       [2.  , 2.  , 2.  , 2.  , 2.  , 2.  ]])
```

Partial results can be accessed with desired time slot. In the retrieved result, the first dimension is the generator index, and the second dimension is the time slot.

```
sp.ED.get(src='pg', attr='v', idx='PV_1', horizon=['EDT1'])
```

```
array([2.1])
```

Or, get multiple variables in mutliple time slots.

```
sp.ED.get(src='pg', attr='v', idx=['PV_1', 'PV_3'], horizon=['EDT1', 'EDT2', 'EDT3'])
```

```
array([[2.1 , 2.1 , 2.1 ],
       [3.23, 2.86, 2.53]])
```

2.7 Output Simulation Results

In AMS, the results can be output in different formats.

One is the plain-text format, where it lists all solved dispatch requests. Another is the CSV format, where the dispatch results are exported to a CSV file.

```
import os

import ams

import datetime

import pandas as pd
```

```
print("Last run time:", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

print(f'ams:{ams.__version__}')
```

```
Last run time: 2024-02-24 17:29:49
ams:0.8.5.post62.dev0+gf6ed683
```

```
ams.config_logger(stream_level=20)
```

2.7.1 Import case and run simulation

```
sp = ams.load(ams.get_case('5bus/pjm5bus_demo.xlsx'),
              setup=True,
              no_output=False,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_demo.
↪xlsx"...
```

```
Input file parsed in 0.1234 seconds.
Zero line rates detected in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0024 seconds.
```

```
sp.DCOPF.run(solver='ECOS')
```

```
Routine <DCOPF> initialized in 0.0090 seconds.
DCOPF solved as optimal in 0.0105 seconds, converged after 9 iterations using solver␣
↪ECOS.
```

```
True
```

2.7.2 Report to plain text

Then, the system method `report()` can generated a plain-text report of the simulation results.

If multiple simulation runs are performed, the report will contain all of them.

```
sp.report()
```

```
Report saved to "pjm5bus_demo_out.txt" in 0.0013 seconds.
```

```
True
```

The report is like:

```
report_file = "pjm5bus_demo_out.txt"

with open(report_file, 'r') as file:
    report_content = file.read()

print(report_content)
```

```
AMS 0.8.5.post62.dev0+gf6ed683
Copyright (C) 2023-2024 Jinning Wang

AMS comes with ABSOLUTELY NO WARRANTY
Case file: /Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_demo.xlsx
Report time: 02/24/2024 05:29:49 PM


========== System Statistics ==========
Buses                              5
Generators                         4
Loads                              3
Shunts                             0
Lines                              7
Transformers                       0
Areas                              3
Regions                            2

============================== DCOPF ==============================
                            P (p.u.)

Generation                        10
Load                              10

Bus DATA:
                                Name        aBus (rad)

Bus_1                              A          0.006759
Bus_2                              B         -0.013078
Bus_3                              C          0.004073
Bus_4                              D           -0.0141
Bus_5                              E          0.006747
```

(continues on next page)

(continued from previous page)

```
Line DATA:
                              Name        plf (p.u.)

Line_0                     Line AB           0.70595
Line_1                     Line AD           0.68617
Line_2                     Line AE          0.001925
Line_3                     Line BC           -1.5881
Line_4                     Line CD           0.61191
Line_5                     Line DE          -0.70193
Line_6                    Line AB2           0.70595

StaticGen DATA:
                              Name         pg (p.u.)

PV_1                          Alta               2.1
PV_3                      Solitude               5.2
PV_5                      Brighton               0.7
Slack_4                   Sundance                 2
```

2.7.3 Export to CSV

The dispatch simulation can also be exported to a CSV file.

```
sp.ED.run(solver='ECOS')
```

```
Routine <ED> initialized in 0.0204 seconds.
ED solved as optimal in 0.0251 seconds, converged after 9 iterations using solver ECOS.
```

```
True
```

```
sp.ED.export_csv()
```

```
'pjm5bus_demo_ED.csv'
```

```
df = pd.read_csv('pjm5bus_demo_ED.csv')
```

In the exported CSV file, each row represents a timeslot, and each column represents a variable.

```
df.iloc[:, :10]
```

```
   Time  pg PV_1  pg PV_3  pg PV_5  pg Slack_4  aBus Bus_1  aBus Bus_2  \
0  EDT1      2.1     3.23      0.6         2.0    0.008402   -0.014116
1  EDT2      2.1     2.86      0.6         2.0    0.008755   -0.014395
2  EDT3      2.1     2.53      0.6         2.0    0.009070   -0.014644
3  EDT4      2.1     2.38      0.6         2.0    0.009214   -0.014757
4  EDT5      2.1     2.30      0.6         2.0    0.009290   -0.014817
5  EDT6      2.1     2.36      0.6         2.0    0.009232   -0.014772
```

(continues on next page)

(continued from previous page)

```
   aBus Bus_3  aBus Bus_4  aBus Bus_5
0   -0.005732   -0.007951    0.008662
1   -0.007696   -0.006855    0.009147
2   -0.009447   -0.005878    0.009580
3   -0.010243   -0.005434    0.009776
4   -0.010668   -0.005197    0.009881
5   -0.010350   -0.005375    0.009802
```

2.7.4 Cleanup

Remove the output files.

```
os.remove('pjm5bus_demo_out.txt')
os.remove('pjm5bus_demo_ED.csv')
```

2.8 Customzie Formulation

Encapsuled the optimization problem calss, AMS provides direct access to the optimization formulation, where users have the option to customize the formulation without playing with the source code.

In this example, we will walk through the optimization problem structure and show how to customize the formulation.

```
import numpy as np

import ams

import datetime
```

```
print("Last run time:", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

print(f'ams:{ams.__version__}')
```

```
Last run time: 2024-02-25 13:22:40
ams:0.8.5.post70.dev0+g008f91c
```

```
ams.config_logger(stream_level=20)
```

2.8.1 Inspect the Optimization Problem Structure

```
sp = ams.load(ams.get_case('5bus/pjm5bus_demo.xlsx'),
              setup=True,
              no_output=True,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_demo.
↪xlsx"...
Input file parsed in 0.1310 seconds.
```

(continues on next page)

(continued from previous page)

```
Zero line rates detected in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0031 seconds.
```

In AMS, a routine collects the descriptive dispatch formulations. `DCOPF`, `RTED`, etc, are the subclasses of `RoutineBase`.

```
sp.DCOPF.init()
```

```
Routine <DCOPF> initialized in 0.0117 seconds.
```

```
True
```

After successful initialization, the attribute `om` is populated with CVXPY-based optimization problem.

The user can even hack to the source `prob` attribute to customize it if necessary.

```
type(sp.DCOPF.om.prob)
```

```
cvxpy.problems.problem.Problem
```

2.8.2 Customize Built-in Formulation

To maintain the consistency, AMS provides a set of built-in formulation customization methods to streamline the customization process.

Here we extend DCOPF with consideration of CO2 emission, where the original formulation can be found in the documentation Routine Reference - DCOPF. To simplify the demonstration, following assumptions are made:

1. Variable $\boxed{e_g}$ is the CO2 emission of each generator. It is proportional to the generation, described by a parameter $\boxed{k_e}$ in the unit t/p.u..
2. Total CO2 emmission is limited by a constant cap $\boxed{t_e}$, in the unit t.
3. A tax $\boxed{c_e}$ is imposed on each unit of CO2 emission in the unit of $/p.u., and the tax is included in the objective function.

Thus, the revised formulation is as follows, where box indicates the revision:

min. $ \sum (c_{2} p_{g}^2 + c_{1} p_{g} + u_{g} c_{0} + \boxed{c_{e} e_{g}})$

s.t.

$\boxed{ e_{g} - k_{e} p_{g} = 0}$

$\boxed{ \sum e_{g} - t_{e} \leq 0}$

$ -p_g + c_{trl,ne}p_{g,0} + c_{trl,e}p_{g,\min} \leq 0 $

$ p_g - c_{trl,ne}p_{g,0} - c_{trl,e}p_{g,\max} \leq 0 $

$ B_{bus}\theta_{bus} + p^{inj}{bus} + C{lpd} + C_{sh}g_{sh} - C_{p}g_{p} = 0 $

$ -B_f\theta_{bus} - p^{inj}f - R{ATEA} \leq 0 $

$ B_f\theta_{bus} + p^{inj}f - R{ATEA} \leq 0 $

$ -C^T_f\theta_{bus} - \theta_{\max} \leq 0 $

$ C^T_f\theta_{bus} - \theta_{\max} \leq 0 $

Decision variables: $ p_g $, $\theta_{bus} $, $\boxed{e_g}$

Note that line flow p^{lf} is calculated as $ B_f\theta_{bus} + p^{inj}_f $ after solving the problem.

Add services

Services are used to store values or build matrix for easier formulation.

```
sp.DCOPF.addService(name='te', tex_name='t_e',
                    unit='t', info='emission cap',
                    value=12)
```

```
ValueService: DCOPF.te
```

Add parameters

We need the following parameters to be defined as `RParam`: `ke` and `ce`. They should be 1D array in the same length as the number of generators and `te` is a scalar.

For a general `RParam`, it has attributes `model`, `indexer`, and `imodel` to describe its source model and index model. The definition of `c2` in DCOPF source code is a good example. However, for ones defined through API, since there is no model containing it, all above attributes are not applicable, and the user should be aware of the sequence of the parameters.

Considering the sequence can be indexed by the generator index, it is used to reference the variables order. Assuming `ke` is reciprocal to the generator capacity, and `ce` is the same for each generator, we can define the parameters as follows:

```
# get the generator indices
stg_idx = sp.DCOPF.pg.get_idx()

# get the value of pmax
pmax = sp.DCOPF.get(src='pmax', attr='v', idx=stg_idx)

# assume the emission factor is 1 for all generators
ke = np.ones_like(pmax)

# assume tax is reciprocal of pmax
ce = np.reciprocal(pmax)
```

```
sp.DCOPF.addRParam(name='ke', tex_name='k_e',
                   info='gen emission factor',
                   model=None, src=None, unit=None,
                   v=ke)
```

```
RParam: DCOPF.ke
```

```
sp.DCOPF.addRParam(name='ce', tex_name='c_e',
                   info='gen emission tax',
                   model=None, src=None, unit=None,
                   v=ce)
```

```
RParam: DCOPF.ce
```

Add variables

The gerator emission eg is added as a new variable.

```
sp.DCOPF.addVars(name='eg', tex_name='e_g',
                 info='Gen emission', unit='t',
                 model='StaticGen', src=None)
```

```
Var: StaticGen.eg
```

Add constraints

The CO2 emission is an equality constraint, and the CO2 emission cap is a simple linear inequality constraint.

If wish to revise an existing built-in constraint, you can redefine the constraint e_str attribute.

```
sp.DCOPF.addConstrs(name='egb', info='Gen emission balance',
                    e_str='eg - mul(ke, pg)', type='eq')
```

```
Constraint: egb [ON]
```

```
sp.DCOPF.addConstrs(name='eub', info='emission upper bound',
                    e_str='sum(eg) - te', type='uq',)
```

```
Constraint: eub [ON]
```

Revise the objective function

The e_str can be revised to include the CO2 emission tax. Here we only need to append the tax term to the original objective function.

```
sp.DCOPF.obj.e_str += '+ sum(mul(ce, pg))'
```

Finalize the Customization

After revising the problem, remember to initialize it before solving.

```
sp.DCOPF.init()
```

```
Routine <DCOPF> initialized in 0.0080 seconds.
```

```
True
```

Solve it and Check the Results

```
sp.DCOPF.run(solver='ECOS')
```

```
DCOPF solved as optimal in 0.0127 seconds, converged after 9 iterations using solver␣
↪ECOS.
```

```
True
```

Inspect the results.

```
sp.DCOPF.eg.v
```

```
array([0.2, 5.2, 4.4, 0.2])
```

```
sp.DCOPF.pg.v
```

```
array([0.2, 5.2, 4.4, 0.2])
```

```
sp.DCOPF.obj.v
```

```
5.297571428627203
```

Load the original problem as a baseline for comparison.

```
sp0 = ams.load(ams.get_case('5bus/pjm5bus_demo.xlsx'),
                 setup=True,
                 no_output=True,)
```

```
Parsing input file "/Users/jinningwang/Documents/work/ams/ams/cases/5bus/pjm5bus_demo.
↪xlsx"...
Input file parsed in 0.0439 seconds.
Zero line rates detacted in rate_a, rate_b, rate_c, adjusted to 999.
If expect a line outage, please set 'u' to 0.
System set up in 0.0020 seconds.
```

```
sp0.DCOPF.run(solver='ECOS')
```

```
Routine <DCOPF> initialized in 0.0067 seconds.
DCOPF solved as optimal in 0.0120 seconds, converged after 9 iterations using solver␣
↪ECOS.
```

```
True
```

From the comparasion, we can see that the generation schedule changes.

```
sp0.DCOPF.pg.v
```

```
array([2.1, 5.2, 0.7, 2. ])
```

```
sp0.DCOPF.obj.v
```

```
2.3445000004668826
```

CHAPTER

THREE

DEVELOPMENT

This chapter introduces advanced topics on modeling with AMS. It aims to give an in-depth explanation of flexible dispatch mdoeling framework and the interoperation with dynamic simulation.

3.1 System

3.1.1 Overview

System is the top-level class for organizing power system dispatch models and routines. The full API reference of System is found at `ams.system.System`.

Dynamic Imports

System dynamically imports groups, models, and routines at creation. To add new models, groups or routines, edit the corresponding file by adding entries following examples.

`ams.system.System.`**`import_models`**`(`*self*`)`

Import and instantiate models as System member attributes.

Models defined in `models/__init__.py` will be instantiated *sequentially* as attributes with the same name as the class name. In addition, all models will be stored in dictionary `System.models` with model names as keys and the corresponding instances as values.

Examples

`system.Bus` stores the *Bus* object, and `system.PV` stores the PV generator object.

`system.models['Bus']` points the same instance as `system.Bus`.

`ams.system.System.`**`import_groups`**`(`*self*`)`

Import all groups classes defined in `models/group.py`.

Groups will be stored as instances with the name as class names. All groups will be stored to dictionary `System.groups`.

`ams.system.System.`**`import_routines`**`(`*self*`)`

Import routines as defined in `routines/__init__.py`.

Routines will be stored as instances with the name as class names. All routines will be stored to dictionary `System.routines`.

Examples

`System.PFlow` is the power flow routine instance.

`ams.system.System.`**`import_types`**(*self*)

Import all types classes defined in `routines/type.py`.

Types will be stored as instances with the name as class names. All types will be stored to dictionary `System.types`.

3.1.2 Device-level Models

AMS follows a similar device-level model organization of ANDES with a few differences.

3.1.3 Routine-level Models

In AMS, routines are responsible for collecting data, defining optimization problems, and solving them.

3.1.4 Optimization

Within the `Routine`, the descriptive formulation are translated into CVXPY optimization problem with `Vars`, `Constraints`, and `Objective`. The full API reference of them can be found in *`ams.opt.Var`*, *`ams.opt.Constraint`*, and *`ams.opt.Objective`*.

class `ams.opt.omodel.`**`OModel`**(*routine*)

Base class for optimization models.

Parameters

routine: Routine
Routine that to be modeled.

Attributes

prob: cvxpy.Problem
Optimization model.

params: OrderedDict
Parameters.

vars: OrderedDict
Decision variables.

constrs: OrderedDict
Constraints.

obj: Objective
Objective function.

3.2 Device

This section introduces the modeling of power system devices. Here the term "model" refers to the descriptive model of a device, which is used to hold the device-level data and variables, such as `Bus`, `Line`, and `PQ`.

AMS employs a similar model organization manner as ANDES, where the class `model` is used to hold the device-level parameters and variables.

Note: One major difference here is, in ANDES, two classes, `ModelData` and `Model`, are used to hold the device-level parameters and equations.

3.2.1 Parameters

Parameter is an atom element in building a power system model. Most parameters are read from an input file, and other parameters are calculated from the existing parameters.

AMS leverages the parameter definition in ANDES, where four classes, `DataParam`, `IdxParam`, `NumParam`, and `ExtParam` are used. More details can be found in ANDES documentation Development - Parameters.

3.2.2 Variables

In AMS, the definition of variables `Algeb` is simplified from ANDES. The `Algeb` class is used to define algebraic variables in the model level, which are used to exchange data with dynamic simulator.

class ams.core.var.**Algeb**(*name: str | None = None*, *tex_name: str | None = None*, *info: str | None = None*, *unit: str | None = None*)

Algebraic variable class.

This class is simplified from `andes.core.var.Algeb`.

Note: The `Algeb` class here is not directly used for optimization purpose, we will discuss its role further in the Routine section.

3.2.3 Model

Encapsulating the parameters and variables, the `Model` class is used to define the device model.

class ams.core.model.**Model**(*system=None*, *config=None*)

Base class for power system dispatch models.

This class is revised from `andes.core.model.Model`.

3.2.4 Examples

The following two examples demonstrate how to define a device model in AMS.

PV model

In this example, we define a PV generator model PV in three steps, data definition, model definition, and manufacturing.

First, we need to define the parameters needed in a PV. not included in ANDES PVData. In this example, we hold the parameters in a separate class GenParam.

```
from andes.core.param import NumParam, ExtParam

class GenParam:
    def __init__(self) -> None:
        self.ctrl = NumParam(default=1,
                             info="generator controllability",
                             tex_name=r'c_{trl}',)
        self.Pc1 = NumParam(default=0.0,
                            info="lower real power output of PQ capability curve",
                            tex_name=r'P_{c1}',
                            unit='p.u.')
        self.Pc2 = NumParam(default=0.0,
                            info="upper real power output of PQ capability curve",
                            tex_name=r'P_{c2}',
                            unit='p.u.')

        ......

        self.pg0 = NumParam(default=0.0,
                            info='real power start point',
                            tex_name=r'p_{g0}',
                            unit='p.u.',
                            )
```

Second, we define the PVModel model with two algebraic variables and an external parameter.

```
from ams.core.model import Model
from ams.core.var import Algeb

class PVModel(Model):

    def __init__(self, system=None, config=None):
        super().__init__(system, config)
        self.group = 'StaticGen'

        self.zone = ExtParam(model='Bus', src='zone', indexer=self.bus, export=False,
                             info='Retrieved zone idx', vtype=str, default=None,
                             )
        self.p = Algeb(info='actual active power generation',
                       unit='p.u.',
                       tex_name='p',
                       name='p',
```

(continues on next page)

(continued from previous page)

```
                            )
        self.q = Algeb(info='actual reactive power generation',
                       unit='p.u.',
                       tex_name='q',
                       name='q',
                       )
```

Note: The external parameter zone is added here to enable the zonal reserve dispatch, but it is not included in ANDES PV.

Third, we manufacture these classes together as the PV model.

```
from andes.models.static.pv import PVData  # NOQA

class PV(PVData, GenParam, PVModel):
    """
    PV generator model.
    """

    def __init__(self, system, config):
        PVData.__init__(self)
        GenParam.__init__(self)
        PVModel.__init__(self, system, config)
```

Lastly, we need to finalize the model by adding the PV model to the model list in $HOME/ams/ams/models/__init__.py, where 'static' is the file name, and 'PV' is the model name.

```
ams_file_classes = list([
    ('info', ['Summary']),
    ('bus', ['Bus']),
    ('static', ['PQ', 'PV', 'Slack']),
    ... ...
])
```

Note: The device-level model development procedures is similar to ANDES. The only difference is that a device-level model for dispatch is much simpler than that for dynamic simulation. In AMS, we only defines the data and small amount of variables. In contrast, ANDES defines the data, variables, and equations for dynamic simulation. Mode details for device-level model development can be found in ANDES documentation Development - Examples.

Line model

In this example, we define a Line model, where the data is extended from existing ANDES LineData by including two extra parameters amin and amax.

```
from andes.models.line.line import LineData
from andes.core.param import NumParam
from andes.shared import deg2rad
```

(continues on next page)

(continued from previous page)

```
from ams.core.model import Model


class Line(LineData, Model):
    """
    AC transmission line model.

    The model is also used for two-winding transformer. Transformers can set the
    tap ratio in ``tap`` and/or phase shift angle ``phi``.

    Notes
    -----
    There is a known issue that adding Algeb ``ud`` will cause Line.algebs run into
    AttributeError: 'NoneType' object has no attribute 'n'. Not figured out why yet.
    """

    def __init__(self, system=None, config=None) -> None:
        LineData.__init__(self)
        Model.__init__(self, system, config)
        self.group = 'ACLine'

        self.amin = NumParam(default=-360 * deg2rad,
                             info="minimum angle difference, from bus - to bus",
                             unit='rad',
                             tex_name=r'a_{min}',
                             )
        self.amax = NumParam(default=360 * deg2rad,
                             info="maximum angle difference, from bus - to bus",
                             unit='rad',
                             tex_name=r'a_{max}',
                             )
```

3.3 Routine

Routine refers to dispatch-level model, and it includes two sectinos, namely, Data Section and Model Section.

3.3.1 Data Section

A simplified code snippet for RTED is shown below as an example.

```
class RTED:

    def __init__(self):
        ... ...
        self.R10 = RParam(info='10-min ramp rate',
                          name='R10', tex_name=r'R_{10}',
                          model='StaticGen', src='R10',
                          unit='p.u./h',)
        self.gs = ZonalSum(u=self.zg, zone='Region',
```

(continues on next page)

(continued from previous page)

```
                       name='gs', tex_name=r'S_{g}',
                       info='Sum Gen vars vector in shape of zone',
                       no_parse=True, sparse=True)
    ... ...
    self.rbu = Constraint(name='rbu', type='eq',
                          info='RegUp reserve balance',
                          e_str = 'gs @ mul(ug, pru) - dud')
    ... ...
```

Routine Parameter

As discussed in previous section, actual data parameters are stored in the device-level models. Thus, in routines, parameters are retrieved from target devices given the device name and the parameter name. In the example above, `R10` is a 10-min ramp rate parameter for the static generator. The parameter is retrieved from the devices `StaticGen` with the parameter name `R10`.

Service

Services are developed to assit the formulations. In the example above, `ZonalSum` is a service to sum the generator variables in a zone. Later, in the constraint, `gs` is multiplied to the reserve variable `pru`.

3.3.2 Model Section

Descriptive Formulation

Dispatch routine is the descriptive model of the optimization problem.

Further, to facilitate the routine definition, AMS developed a class `ams.core.param.RParam` to pass the model data to multiple routine modeling.

class `ams.core.param.`**`RParam`**(*name: str | None = None*, *tex_name: str | None = None*, *info: str | None = None*, *src: str | None = None*, *unit: str | None = None*, *model: str | None = None*, *v: ndarray | None = None*, *indexer: str | None = None*, *imodel: str | None = None*, *expand_dims: int | None = None*, *no_parse: bool | None = False*, *nonneg: bool | None = False*, *nonpos: bool | None = False*, *complex: bool | None = False*, *imag: bool | None = False*, *symmetric: bool | None = False*, *diag: bool | None = False*, *hermitian: bool | None = False*, *boolean: bool | None = False*, *integer: bool | None = False*, *pos: bool | None = False*, *neg: bool | None = False*, *sparse: list | None = None*)

Class for parameters used in a routine. This class is developed to simplify the routine definition.

RParm is further used to define *Parameter* in the optimization model.

no_parse is used to skip parsing the *RParam* in optimization model. It means that the *RParam* will not be added to the optimization model. This is useful when the RParam contains non-numeric values, or it is not necessary to be added to the optimization model.

Parameters

name
[str, optional] Name of this parameter. If not provided, *name* will be set to the attribute name.

tex_name
[str, optional] LaTeX-formatted parameter name. If not provided, *tex_name* will be assigned the same as *name*.

info
[str, optional] A description of this parameter

src
[str, optional] Source name of the parameter.

unit
[str, optional] Unit of the parameter.

model
[str, optional] Name of the owner model or group.

v
[np.ndarray, optional] External value of the parameter.

indexer
[str, optional] Indexer of the parameter.

imodel
[str, optional] Name of the owner model or group of the indexer.

no_parse: bool, optional
True to skip parsing the parameter.

nonneg: bool, optional
True to set the parameter as non-negative.

nonpos: bool, optional
True to set the parameter as non-positive.

complex: bool, optional
True to set the parameter as complex.

imag: bool, optional
True to set the parameter as imaginary.

symmetric: bool, optional
True to set the parameter as symmetric.

diag: bool, optional
True to set the parameter as diagonal.

hermitian: bool, optional
True to set the parameter as hermitian.

boolean: bool, optional
True to set the parameter as boolean.

integer: bool, optional
True to set the parameter as integer.

pos: bool, optional
True to set the parameter as positive.

neg: bool, optional
True to set the parameter as negative.

sparse: bool, optional
True to set the parameter as sparse.

Examples

Example 1: Define a routine parameter from a source model or group.

In this example, we define the parameter *cru* from the source model *SFRCost* with the parameter *cru*.

```
>>> self.cru = RParam(info='RegUp reserve coefficient',
>>>                  tex_name=r'c_{r,u}',
>>>                  unit=r'$/(p.u.)',
>>>                  name='cru',
>>>                  src='cru',
>>>                  model='SFRCost'
>>>                  )
```

Example 2: Define a routine parameter with a user-defined value.

In this example, we define the parameter with a user-defined value. TODO: Add example

RoutineBase([system, config])	Class to hold descriptive routine models and data mapping.

ams.routines.RoutineBase

class ams.routines.**RoutineBase**(*system=None*, *config=None*)

Class to hold descriptive routine models and data mapping.

__init__(*system=None*, *config=None*)

Methods

addConstrs(name, e_str[, info, type])	Add *Constraint* to the routine.
addRParam(name[, tex_name, info, src, unit, ...])	Add *RParam* to the routine.
addService(name, value[, tex_name, unit, ...])	Add *ValueService* to the routine.
addVars(name[, model, shape, tex_name, ...])	Add a variable to the routine.
dc2ac(**kwargs)	Convert the DC-based results with ACOPF.
disable(name)	Disable a constraint by name.
doc([max_width, export])	Retrieve routine documentation as a string.
enable(name)	Enable a constraint by name.
export_csv([path])	Export dispatch results to a csv file.
get(src, idx[, attr, horizon])	Get the value of a variable or parameter.
get_load(horizon, src[, attr, idx, model, ...])	Get the load value by applying zonal scaling factor defined in `Horizon`.
igmake([directed])	Build an igraph object from the system.
igraph([input, ytimes, decimal, directed, ...])	Plot a system uging *g.plot()* of *igraph*, with optional input.
init([force, no_code])	Initialize the routine.
prepare()	Prepare the routine.
run([force_init, no_code])	Run the routine.
set(src, idx[, attr, value])	Set the value of an attribute of a routine parameter.
solve(**kwargs)	Solve the routine optimization model.
summary(**kwargs)	Summary interface
unpack(**kwargs)	Unpack the results.
update([params, mat_make])	Update the values of Parameters in the optimization model.

RoutineBase.addConstrs

RoutineBase.**addConstrs**(*name: str*, *e_str: str*, *info: str | None = None*, *type: str | None = 'uq'*)

Add *Constraint* to the routine. to the routine.

Parameters

name
[str] Constraint name. One should typically assigning the name directly because it will be automatically assigned by the model. The value of `name` will be the symbol name to be used in expressions.

e_str
[str] Constraint expression string.

info
[str, optional] Descriptive information

type
[str, optional] Constraint type, `uq` for uncertain, `eq` for equality, `ineq` for inequality.

RoutineBase.addRParam

RoutineBase.**addRParam**(*name: str, tex_name: str | None = None, info: str | None = None, src: str | None = None, unit: str | None = None, model: str | None = None, v: ndarray | None = None, indexer: str | None = None, imodel: str | None = None*)

Add *RParam* to the routine.

Parameters

name
[str] Name of this parameter. If not provided, *name* will be set to the attribute name.

tex_name
[str, optional] LaTeX-formatted parameter name. If not provided, *tex_name* will be assigned the same as *name*.

info
[str, optional] A description of this parameter

src
[str, optional] Source name of the parameter.

unit
[str, optional] Unit of the parameter.

model
[str, optional] Name of the owner model or group.

v
[np.ndarray, optional] External value of the parameter.

indexer
[str, optional] Indexer of the parameter.

imodel
[str, optional] Name of the owner model or group of the indexer.

RoutineBase.addService

RoutineBase.**addService**(*name: str, value: ndarray, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, model: str = None*)

Add *ValueService* to the routine.

Parameters

name
[str] Instance name.

value
[np.ndarray] Value.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

vtype
[Type, optional] Variable type.

model
[str, optional] Model name.

RoutineBase.addVars

RoutineBase.**addVars**(*name: str*, *model: str | None = None*, *shape: tuple | int | None = None*, *tex_name: str | None = None*, *info: str | None = None*, *src: str | None = None*, *unit: str | None = None*, *horizon: RParam | None = None*, *nonneg: bool | None = False*, *nonpos: bool | None = False*, *complex: bool | None = False*, *imag: bool | None = False*, *symmetric: bool | None = False*, *diag: bool | None = False*, *psd: bool | None = False*, *nsd: bool | None = False*, *hermitian: bool | None = False*, *bool: bool | None = False*, *integer: bool | None = False*, *pos: bool | None = False*, *neg: bool | None = False*)

Add a variable to the routine.

Parameters

name
[str, optional] Variable name. One should typically assigning the name directly because it will be automatically assigned by the model. The value of `name` will be the symbol name to be used in expressions.

model
[str, optional] Name of the owner model or group.

shape
[int or tuple, optional] Shape of the variable. If is None, the shape of *model* will be used.

info
[str, optional] Descriptive information

unit
[str, optional] Unit

tex_name
[str] LaTeX-formatted variable symbol. If is None, the value of *name* will be used.

src
[str, optional] Source variable name. If is None, the value of *name* will be used.

lb
[str, optional] Lower bound

ub
[str, optional] Upper bound

horizon
[ams.routines.RParam, optional] Horizon idx.

nonneg
[bool, optional] Non-negative variable

nonpos
[bool, optional] Non-positive variable

complex
[bool, optional] Complex variable

imag
[bool, optional] Imaginary variable

symmetric
[bool, optional] Symmetric variable

diag
[bool, optional] Diagonal variable

psd
[bool, optional] Positive semi-definite variable

nsd
[bool, optional] Negative semi-definite variable

hermitian
[bool, optional] Hermitian variable

bool
[bool, optional] Boolean variable

integer
[bool, optional] Integer variable

pos
[bool, optional] Positive variable

neg
[bool, optional] Negative variable

RoutineBase.dc2ac

`RoutineBase.dc2ac(**kwargs)`

Convert the DC-based results with ACOPF.

RoutineBase.disable

`RoutineBase.disable(name)`

Disable a constraint by name.

Parameters

name: str or list
name of the constraint to be disabled

RoutineBase.doc

`RoutineBase.doc(max_width=78, export='plain')`

Retrieve routine documentation as a string.

RoutineBase.enable

`RoutineBase.enable(name)`

Enable a constraint by name.

Parameters

name: str or list
name of the constraint to be enabled

RoutineBase.export_csv

`RoutineBase.export_csv(path=None)`

Export dispatch results to a csv file. For multi-period routines, the column "Time" is the time index of `timeslot.v`, which usually comes from `EDTSlot` or `UCTSlot`. The rest columns are the variables registered in `vars`.

For single-period routines, the column "Time" have a pseduo value of "T1".

Parameters

path
[str] path of the csv file to save

Returns

str
The path of the exported csv file

RoutineBase.get

`RoutineBase.get(src: str, idx, attr: str = 'v', horizon: int | str | Iterable | None = None)`

Get the value of a variable or parameter.

Parameters

src: str
Name of the variable or parameter.

idx: int, str, or list
Index of the variable or parameter.

attr: str
Attribute name.

horizon: list, optional
Horizon index.

RoutineBase.get_load

RoutineBase.**get_load**(*horizon: int | str*, *src: str*, *attr: str = 'v'*, *idx=None*, *model: str = 'EDTSlot'*, *factor: str = 'sd'*)

Get the load value by applying zonal scaling factor defined in `Horizon`.

Parameters

idx: int, str, or list
Index of the desired load.

attr: str
Attribute name.

model: str
Scaling factor owner, `EDTSlot` or `UCTSlot`.

factor: str
Scaling factor name, usually `sd`.

horizon: int or str
Horizon single index.

RoutineBase.igmake

RoutineBase.**igmake**(*directed=True*)

Build an igraph object from the system.

Parameters

directed: bool
Whether the graph is directed.

Returns

igraph.Graph
An igraph object.

RoutineBase.igraph

RoutineBase.**igraph**(*input: RParam | Var | None = None*, *ytimes: float | None = None*, *decimal: int | None = 6*, *directed: bool | None = True*, *dpi: int | None = 100*, *figsize: tuple | None = None*, *adjust_bus: bool | None = False*, *gen_color: str | None = 'red'*, *rest_color: str | None = 'black'*, *vertex_shape: str | None = 'circle'*, *vertex_font: str | None = None*, *no_vertex_label: bool | None = False*, *vertex_label: str | list | None = None*, *vertex_size: float | None = None*, *vertex_label_size: float | None = None*, *vertex_label_dist: float | None = 1.5*, *vertex_label_angle: float | None = 10.2*, *edge_arrow_size: float | None = None*, *edge_arrow_width: float | None = None*, *edge_width: float | None = None*, *edge_align_label: bool | None = True*, *edge_background: str | None = None*, *edge_color: str | None = None*, *edge_curved: bool | None = False*, *edge_font: str | None = None*, *edge_label: str | list | None = None*, *layout: str | None = 'rt'*, *autocurve: bool | None = True*, *ax: Axes | None = None*, *title: str | None = None*, *title_loc: str | None = None*, ***visual_style*)

Plot a system uging *g.plot()* of *igraph*, with optional input. For now, only support plotting of Bus and Line elements as input.

Parameters

input: RParam or Var, optional
The variable or parameter to be plotted.

ytimes: float, optional
The scaling factor of the values.

directed: bool, optional
Whether the graph is directed.

dpi: int, optional
Dots per inch.

figsize: tuple, optional
Figure size.

adjust_bus: bool, optional
Whether to adjust the bus size.

gen_color: str, optional
Color of the generator bus.

rest_color: str, optional
Color of the rest buses.

no_vertex_label: bool, optional
Whether to show vertex labels.

vertex_shape: str, optional
Shape of the vertices.

vertex_font: str, optional
Font of the vertices.

vertex_size: float, optional
Size of the vertices.

vertex_label_size: float, optional
Size of the vertex labels.

vertex_label_dist: float, optional
Distance of the vertex labels.

vertex_label_angle: float, optional
Angle of the vertex labels.

edge_arrow_size: float, optional
Size of the edge arrows.

edge_arrow_width: float, optional
Width of the edge arrows.

edge_width: float, optional
Width of the edges.

edge_align_label: bool, optional
Whether to align the edge labels.

edge_background: str, optional
RGB colored rectangle background of the edge labels.

layout: str, optional
Layout of the graph, ['rt', 'kk', 'fr', 'drl', 'lgl', 'circle', 'grid_fr'].

autocurve: bool, optional
Whether to use autocurve.

ax: plt.Axes, optional
Matplotlib axes.

visual_style: dict, optional
Visual style, see `igraph.plot` for details.

Returns

plt.Axes
Matplotlib axes.

igraph.Graph
An igraph object.

Examples

```
>>> import ams
>>> sp = ams.load(ams.get_case('5bus/pjm5bus_uced.xlsx'))
>>> sp.DCOPF.run()
>>> sp.DCOPF.plot(input=sp.DCOPF.pn,
>>>               ytimes=10,
>>>               adjust_bus=True,
>>>               vertex_size=10,
>>>               vertex_label_size=15,
>>>               vertex_label_dist=2,
>>>               vertex_label_angle=90,
>>>               show=False,
>>>               edge_align_label=True,
>>>               autocurve=True,)
```

RoutineBase.init

`RoutineBase.init`(*force=False*, *no_code=True*, ***kwargs*)

Initialize the routine.

Force initialization (*force=True*) will do the following: - Rebuild the system matrices - Enable all constraints - Reinitialize the optimization model

Parameters

force: bool
Whether to force initialization.

no_code: bool
Whether to show generated code.

RoutineBase.prepare

RoutineBase.**prepare**()

Prepare the routine.

RoutineBase.run

RoutineBase.**run**(*force_init=False*, *no_code=True*, ***kwargs*)

Run the routine.

Force initialization (*force_init=True*) will do the following: - Rebuild the system matrices - Enable all constraints - Reinitialize the optimization model

Parameters

force_init: bool
Whether to force initialization.

no_code: bool
Whether to show generated code.

RoutineBase.set

RoutineBase.**set**(*src: str*, *idx*, *attr: str = 'v'*, *value=0.0*)

Set the value of an attribute of a routine parameter.

RoutineBase.solve

RoutineBase.**solve**(***kwargs*)

Solve the routine optimization model.

RoutineBase.summary

RoutineBase.**summary**(***kwargs*)

Summary interface

RoutineBase.unpack

RoutineBase.**unpack**(***kwargs*)

Unpack the results.

RoutineBase.update

`RoutineBase.update(params=None, mat_make=True)`

Update the values of Parameters in the optimization model.

This method is particularly important when some *RParams* are linked with system matrices. In such cases, setting *mat_make=True* is necessary to rebuild these matrices for the changes to take effect. This is common in scenarios involving topology changes, connection statuses, or load value modifications. If unsure, it is advisable to use *mat_make=True* as a precautionary measure.

Parameters

params: Parameter, str, or list
Parameter, Parameter name, or a list of parameter names to be updated. If None, all parameters will be updated.

mat_make: bool
True to rebuild the system matrices. Set to False to speed up the process if no system matrices are changed.

Attributes

class_name

RoutineBase.class_name

`property RoutineBase.class_name`

Numerical Optimization

Optimization model is the optimization problem. `Var`, `Constraint`, and `Objective` are the basic building blocks of the optimization model. `OModel` is the container of the optimization model. A summary table is shown below.

`Var`([name, tex_name, info, src, unit, ...])	Base class for variables used in a routine.
`Constraint`([name, e_str, info, type])	Base class for constraints.
`Objective`([name, e_str, info, unit, sense])	Base class for objective functions.
`OModel`(routine)	Base class for optimization models.

ams.opt.Var

`class ams.opt.Var(name: str | None = None, tex_name: str | None = None, info: str | None = None, src: str | None = None, unit: str | None = None, model: str | None = None, shape: tuple | int | None = None, v0: str | None = None, horizon=None, nonneg: bool | None = False, nonpos: bool | None = False, complex: bool | None = False, imag: bool | None = False, symmetric: bool | None = False, diag: bool | None = False, psd: bool | None = False, nsd: bool | None = False, hermitian: bool | None = False, boolean: bool | None = False, integer: bool | None = False, pos: bool | None = False, neg: bool | None = False)`

Base class for variables used in a routine.

When *horizon* is provided, the variable will be expanded to a matrix, where rows are indexed by the source variable index and columns are indexed by the horizon index.

Parameters

info
[str, optional] Descriptive information

unit
[str, optional] Unit

tex_name
[str] LaTeX-formatted variable symbol. Defaults to the value of `name`.

name
[str, optional] Variable name. One should typically assigning the name directly because it will be automatically assigned by the model. The value of `name` will be the symbol name to be used in expressions.

src
[str, optional] Source variable name. Defaults to the value of `name`.

model
[str, optional] Name of the owner model or group.

horizon
[ams.routines.RParam, optional] Horizon idx.

nonneg
[bool, optional] Non-negative variable

nonpos
[bool, optional] Non-positive variable

complex
[bool, optional] Complex variable

imag
[bool, optional] Imaginary variable

symmetric
[bool, optional] Symmetric variable

diag
[bool, optional] Diagonal variable

psd
[bool, optional] Positive semi-definite variable

nsd
[bool, optional] Negative semi-definite variable

hermitian
[bool, optional] Hermitian variable

boolean
[bool, optional] Boolean variable

integer
[bool, optional] Integer variable

pos
[bool, optional] Positive variable

neg
[bool, optional] Negative variable

Attributes

a
[np.ndarray] Variable address.

_v
[np.ndarray] Local-storage of the variable value.

rtn
[ams.routines.Routine] The owner routine instance.

__init__(*name: str | None = None, tex_name: str | None = None, info: str | None = None, src: str | None = None, unit: str | None = None, model: str | None = None, shape: tuple | int | None = None, v0: str | None = None, horizon=None, nonneg: bool | None = False, nonpos: bool | None = False, complex: bool | None = False, imag: bool | None = False, symmetric: bool | None = False, diag: bool | None = False, psd: bool | None = False, nsd: bool | None = False, hermitian: bool | None = False, boolean: bool | None = False, integer: bool | None = False, pos: bool | None = False, neg: bool | None = False*)

Methods

get_idx()	
parse()	Parse the variable.

Var.get_idx

`Var.get_idx()`

Var.parse

`Var.parse()`

Parse the variable.

Attributes

class_name	Return the class name
n	Return the number of elements.
shape	Return the shape.
size	Return the size.
v	Return the CVXPY variable value.

Var.class_name

property Var.**class_name**

Return the class name

Var.n

property Var.**n**

Return the number of elements.

Var.shape

property Var.**shape**

Return the shape.

Var.size

property Var.**size**

Return the size.

Var.v

property Var.**v**

Return the CVXPY variable value.

ams.opt.Constraint

class ams.opt.**Constraint**(*name: str | None = None, e_str: str | None = None, info: str | None = None, type: str | None = 'uq'*)

Base class for constraints.

This class is used as a template for defining constraints. Each instance of this class represents a single constraint.

Parameters

name
[str, optional] A user-defined name for the constraint.

e_str
[str, optional] A mathematical expression representing the constraint.

info
[str, optional] Additional informational text about the constraint.

type
[str, optional] The type of constraint, which determines the mathematical relationship. Possible values include 'uq' (inequality, default) and 'eq' (equality).

Attributes

is_disabled
[bool] Flag indicating if the constraint is disabled, False by default.

rtn
[ams.routines.Routine] The owner routine instance.

__init__(*name: str | None = None*, *e_str: str | None = None*, *info: str | None = None*, *type: str | None = 'uq'*)

Methods

parse([no_code])	Parse the constraint.

Constraint.parse

`Constraint.parse(no_code=True)`

Parse the constraint.

Parameters

no_code
[bool, optional] Flag indicating if the code should be shown, True by default.

Attributes

class_name	Return the class name
n	Return the number of elements.
shape	Return the shape.
size	Return the size.
v	Return the CVXPY constraint LHS value.
v2	Return the calculated constraint LHS value.

Constraint.class_name

`property Constraint.class_name`

Return the class name

Constraint.n

`property Constraint.n`

Return the number of elements.

Constraint.shape

property Constraint.**shape**

Return the shape.

Constraint.size

property Constraint.**size**

Return the size.

Constraint.v

property Constraint.**v**

Return the CVXPY constraint LHS value.

Constraint.v2

property Constraint.**v2**

Return the calculated constraint LHS value. Note that `v` should be used primarily as it is obtained from the solver directly. `v2` is for debugging purpose, and should be consistent with `v`.

ams.opt.Objective

class ams.opt.**Objective**(*name: str | None = None*, *e_str: str | None = None*, *info: str | None = None*, *unit: str | None = None*, *sense: str | None = 'min'*)

Base class for objective functions.

This class serves as a template for defining objective functions. Each instance of this class represents a single objective function that can be minimized or maximized depending on the sense ('min' or 'max').

Parameters

name
[str, optional] A user-defined name for the objective function.

e_str
[str, optional] A mathematical expression representing the objective function.

info
[str, optional] Additional informational text about the objective function.

sense
[str, optional] The sense of the objective function, default to 'min'. *min* for minimization and *max* for maximization.

Attributes

v
[NoneType] Return the CVXPY objective value.

rtn
[ams.routines.Routine] The owner routine instance.

__init__(*name: str | None = None*, *e_str: str | None = None*, *info: str | None = None*, *unit: str | None = None*, *sense: str | None = 'min'*)

Methods

parse([no_code])	Parse the objective function.

Objective.parse

`Objective.parse(no_code=True)`

Parse the objective function.

Parameters

no_code
[bool, optional] Flag indicating if the code should be shown, True by default.

Attributes

class_name	Return the class name
n	Return the number of elements.
shape	Return the shape.
size	Return the size.
v	Return the CVXPY objective value.
v2	Return the calculated objective value.

Objective.class_name

`property Objective.class_name`

Return the class name

Objective.n

`property Objective.n`

Return the number of elements.

Objective.shape

`property Objective.shape`

Return the shape.

Objective.size

property Objective.**size**

Return the size.

Objective.v

property Objective.**v**

Return the CVXPY objective value.

Objective.v2

property Objective.**v2**

Return the calculated objective value. Note that `v` should be used primarily as it is obtained from the solver directly. `v2` is for debugging purpose, and should be consistent with `v`.

ams.opt.OModel

class ams.opt.**OModel**(*routine*)

Base class for optimization models.

Parameters

routine: Routine
Routine that to be modeled.

Attributes

prob: cvxpy.Problem
Optimization model.

params: OrderedDict
Parameters.

vars: OrderedDict
Decision variables.

constrs: OrderedDict
Constraints.

obj: Objective
Objective function.

__init__(*routine*)

Methods

init([no_code])	Set up the optimization model from the symbolic description.
update(params)	Update the Parameter values.

OModel.init

OModel.**init**(*no_code=True*)

Set up the optimization model from the symbolic description.

This method initializes the optimization model by parsing decision variables, constraints, and the objective function from the associated routine.

Parameters

no_code
[bool, optional] Flag indicating if the parsing code should be displayed, True by default.

Returns

bool
Returns True if the setup is successful, False otherwise.

OModel.update

OModel.**update**(*params*)

Update the Parameter values.

Parameters

params: list
List of parameters to be updated.

Attributes

class_name	Return the class name

OModel.class_name

property OModel.**class_name**

Return the class name

3.3.3 Interoperation with ANDES

The interoperation with dynamic simulator invovles both file conversion and data exchange. In AMS, the built-in interface with ANDES is implemented in `ams.interop.andes`.

File Format Converter

Power flow data is the bridge between dispatch study and dynamic study, where it defines grid topology and power flow. An AMS case can be converted to an ANDES case, with the option to supply additional dynamic data.

`ams.interop.andes.`**`to_andes`**(*system*, *setup=False*, *addfile=None*, ***kwargs*)

Convert the AMS system to an ANDES system.

A preferred dynamic system file to be added has following features: 1. The file contains both power flow and dynamic models. 2. The file can run in ANDES natively. 3. Power flow models are in the same shape as the AMS system. 4. Dynamic models, if any, are in the same shape as the AMS system.

This function is wrapped as the `System` class method `to_andes()`. Using the file conversion `to_andes()` will automatically link the AMS system instance to the converted ANDES system instance in the AMS system attribute `dyn`.

It should be noted that detailed dynamic simualtion requires extra dynamic models to be added to the ANDES system, which can be passed through the `addfile` argument.

Parameters

system
[System] The AMS system to be converted to ANDES format.

setup
[bool, optional] Whether to call *setup()* after the conversion. Default is True.

addfile
[str, optional] The additional file to be converted to ANDES dynamic mdoels.

****kwargs**
[dict] Keyword arguments to be passed to *andes.system.System*.

Returns

adsys
[andes.system.System] The converted ANDES system.

Notes

1. Power flow models in the addfile will be skipped and only dynamic models will be used.
2. The addfile format is guessed based on the file extension. Currently only `xlsx` is supported.
3. Index in the addfile is automatically adjusted when necessary.

Examples

```
>>> import ams
>>> import andes
>>> sp = ams.load(ams.get_case('ieee14/ieee14_uced.xlsx'), setup=True)
>>> sa = sp.to_andes(setup=False,
...                  addfile=andes.get_case('ieee14/ieee14_full.xlsx'),
...                  overwrite=True, no_output=True)
```

Data Exchange in Simulation

To achieve dispatch-dynamic cosimulation, it requires bi-directional data exchange between dispatch and dynamic study. From the perspective of AMS, two functions, `send` and `receive`, are developed. The maping relationship for a specific routine is defined in the routine class as `map1` and `map2`. Additionally, a link table for the ANDES case is used for the controller connections.

Module *ams.interop.andes.Dynamic*, contains the necessary functions and classes for file conversion and data exchange.

class ams.interop.andes.**Dynamic**(*amsys=None*, *adsys=None*)

ANDES interface class.

Parameters

amsys
[AMS.system.System] The AMS system.

adsys
[ANDES.system.System] The ANDES system.

Notes

1. Using the file conversion `to_andes()` will automatically link the AMS system to the converted ANDES system in the attribute `dyn`.

Examples

```
>>> import ams
>>> import andes
>>> sp = ams.load(ams.get_case('ieee14/ieee14_rted.xlsx'), setup=True)
>>> sa = sp.to_andes(setup=True,
...                  addfile=andes.get_case('ieee14/ieee14_wt3.xlsx'),
...                  overwrite=True, keep=False, no_output=True)
>>> sp.RTED.run()
>>> sp.RTED.dc2ac()
>>> sp.dyn.send()  # send RTED results to ANDES system
>>> sa.PFlow.run()
>>> sp.TDS.run()
>>> sp.dyn.receive()  # receive TDS results from ANDES system
```

Attributes

link
[pandas.DataFrame] The ANDES system link table.

`receive`(*adsys=None*, *routine=None*, *no_update=False*)

Receive ANDES system results to AMS devices.

Parameters

adsys
[adsys.System.system, optional] The target ANDES dynamic system instance. If not provided, use the linked ANDES system isntance (`sp.dyn.adsys`).

routine
[str, optional] The routine to be received from ANDES. If None, `recent` will be used.

no_update
[bool, optional] True to skip update the AMS routine parameters after sync. Default is False.

`send`(*adsys=None*, *routine=None*)

Send results of the recent sovled AMS dispatch (`sp.recent`) to the target ANDES system.

Note that converged AC conversion DOES NOT guarantee successful dynamic initialization `TDS.init()`. Failed initialization is usually caused by limiter violation.

Parameters

adsys
[adsys.System.system, optional] The target ANDES dynamic system instance. If not provided, use the linked ANDES system isntance (`sp.dyn.adsys`).

routine
[str, optional] The routine to be sent to ANDES. If None, `recent` will be used.

When you use this interface, it automatically picks either the dynamic or static model based on the TDS initialization status. If the TDS is running, it selects the dynamic model; otherwise, it goes for the static model. For more details, check out the full API reference or take a look at the source code.

Note: Check ANDES documentation StaticGen for more details about substituting static generators with dynamic generators.

3.4 Examples

One example is provided to demonstrate descriptive dispatch modeling.

3.4.1 DCOPF

DC optimal power flow (DCOPF) is a fundamental routine used in power system analysis. In this example, we demonstrate how to implement a DCOPF routine in a descriptive manner using AMS. Below is a simplified DCOPF code snippet. The full code can be found in `ams.routines.dcopf.DCOPF`.

Data Section

```
1  class DCOPF(RoutineBase):
2  
3      def __init__(self, system, config):
4          RoutineBase.__init__(self, system, config)
5          self.info = 'DC Optimal Power Flow'
6          self.type = 'DCED'
7          # --- Data Section ---
8          # --- generator cost ---
9          self.c1 = RParam(info='Gen cost coefficient 1',
10                          name='c1', tex_name=r'c_{1}', unit=r'$/(p.u.)',
11                          model='GCost', src='c1',
12                          indexer='gen', imodel='StaticGen',)
13         # --- generator ---
14         self.ug = RParam(info='Gen connection status',
15                          name='ug', tex_name=r'u_{g}',
16                          model='StaticGen', src='u',
17                          no_parse=True)
18         self.ctrl = RParam(info='Gen controllability',
19                            name='ctrl', tex_name=r'c_{trl}',
20                            model='StaticGen', src='ctrl',
21                            no_parse=True)
22         self.ctrle = NumOpDual(info='Effective Gen controllability',
23                                name='ctrle', tex_name=r'c_{trl, e}',
24                                u=self.ctrl, u2=self.ug,
25                                fun=np.multiply, no_parse=True)
26         # --- load ---
27         self.pd = RParam(info='active demand',
28                          name='pd', tex_name=r'p_{d}',
29                          model='StaticLoad', src='p0',
30                          unit='p.u.',)
31         # --- line ---
32         self.rate_a = RParam(info='long-term flow limit',
33                              name='rate_a', tex_name=r'R_{ATEA}',
34                              unit='MVA', model='Line',)
35         # --- line angle difference ---
36         self.amax = NumOp(u=self.rate_a, fun=np.ones_like,
37                           rfun=np.dot, rargs=dict(b=np.pi),
38                           name='amax', tex_name=r'\theta_{max}',
39                           info='max line angle difference',
40                           no_parse=True,)
41         # --- connection matrix ---
42         self.Cg = RParam(info='Gen connection matrix',
43                          name='Cg', tex_name=r'C_{g}',
44                          model='mats', src='Cg',
45                          no_parse=True, sparse=True,)
46         # --- system matrix ---
47         self.Bbus = RParam(info='Bus admittance matrix',
48                            name='Bbus', tex_name=r'B_{bus}',
49                            model='mats', src='Bbus',
50                            no_parse=True, sparse=True,)
```

Lines 1-4: Derive subclass DCOPF from RoutineBase.

Lines 5-6: Define routine information and type.

Lines 9-12: Define linear generator cost coefficients `c1`, where it sources from `GCost.c1` and sorted by `StaticGen.gen`.

Lines 22-24: Define effective controllability as service `ctrle`, where it multiplies `ctrl` and `ug`.

Lines 42-45: Define generator connection matrix `Cg`, where it sources from matrix processor `mats.Cg` and is sparse.

Model Section

```
51          # --- Model Section ---
52          # --- generation ---
53          self.pg = Var(info='Gen active power',
54                        unit='p.u.',
55                        name='pg', tex_name=r'p_g',
56                        model='StaticGen', src='p',
57                        v0=self.pg0)
58          # --- bus ---
59          self.aBus = Var(info='Bus voltage angle',
60                          unit='rad',
61                          name='aBus', tex_name=r'\theta_{bus}',
62                          model='Bus', src='a',)
63          # --- power balance ---
64          pb = 'Bbus@aBus + Pbusinj + Cl@pd + Csh@gsh - Cg@pg'
65          self.pb = Constraint(name='pb', info='power balance',
66                               e_str=pb, type='eq',)
67          # --- line flow ---
68          self.plf = Var(info='Line flow',
69                         unit='p.u.',
70                         name='plf', tex_name=r'p_{lf}',
71                         model='Line',)
72          self.plflb = Constraint(info='line flow lower bound',
73                                  name='plflb', type='uq',
74                                  e_str='-Bf@aBus - Pfinj - rate_a',)
75          self.plfub = Constraint(info='line flow upper bound',
76                                  name='plfub', type='uq',
77                                  e_str='Bf@aBus + Pfinj - rate_a',)
78          # --- objective ---
79          obj = 'sum(mul(c2, power(pg, 2)))'
80          obj += '+ sum(mul(c1, pg))'
81          obj += '+ sum(mul(ug, c0))'
82          self.obj = Objective(name='obj',
83                               info='total cost', unit='$',
84                               sense='min', e_str=obj,)
```

Continued from the above code.

Lines 53-57: Define variable `pg`, where it links to `StaticGen.p` and initial value `pg0`.

Lines 68-71: Define variable `plf`, where it links to `Line` with no target source variable nor initial value.

Lines 72-77: Define inequality constraints `plflb` and `plfub` for line flow limits.

Lines 79-84: Define objective function `obj` for minimizing total cost.

Finalize

Lastly, similar to finalize a device model, we need to finalize the routine by adding the `RTED` to the routine list in `$HOME/ams/ams/routines/__init__.py`, where 'rted' is the file name, and 'RTED' is the routine name.

```
all_routines = OrderedDict([
      ... ...
      ('dcopf', ['DCOPF']),
      ('ed', ['ED', 'EDDG', 'EDES']),
      ('rted', ['RTED', 'RTEDDG', 'RTEDES', 'RTEDVIS']),
      ... ...
])
```

Note: Refer to the documentation "Example - Customize Formulation" for API customization that does not require modification of the source code.

CHAPTER

FOUR

RELEASE NOTES

The APIs before v3.0.0 are in beta and may change without prior notice.

4.1 Pre-v1.0.0

4.1.1 v09.0 (2024-02-27)

- Add ex8 to demonstrate customize existing formulations via API
- Improve Development documentation
- Fix `addService`, `addVars`
- Rename `RoutineModel` to `RoutineBase` for better naming
- Fix ANDES file converter issue
- Initial release to conda-forge

4.1.2 v0.8.5 (2024-01-31)

- Improve quality of coverage and format
- Fix dependency issue

4.1.3 v0.8.4 (2024-01-30)

- Version cleanup

4.1.4 v0.8.3 (2024-01-30)

- Initial release to PyPI

4.1.5 v0.8.2 (2024-01-30)

- Improve examples
- Add report module and export_csv for results export

4.1.6 v0.8.1 (2024-01-20)

- Improve `MatProcessor`
- Add more examples
- Improve ANDES interface

4.1.7 v0.8.0 (2024-01-09)

- Refactor `DCED` routines to improve performance

4.1.8 v0.7.5 (2023-12-28)

- Refactor `MatProcessor` and `DCED` routines to improve performance
- Integrate sparsity pattern in `RParam`
- Rename energy storage routines `RTED2`, `ED2` and `UC2` to `RTEDES`, `EDES` and `UCES`

4.1.9 v0.7.4 (2023-11-29)

- Refactor routins and optimization models to improve performance
- Fix routines modeling
- Add examples
- Fix built-in cases

4.1.10 v0.7.3 (2023-11-03)

- Add tests

4.1.11 v0.7.2 (2023-10-26)

- Add routines `ED2` and `UC2`
- Minor fix on `SymProcessor` and `Documenter`

4.1.12 v0.7.1 (2023-10-12)

- Add function `_initial_guess` to routine `UC`
- Refactor PYPOWER

4.1.13 v0.7.0 (2023-09-22)

- Add interfaces for customizing optimization
- Add models `REGCV1` and `REGCV1Cost` for virtual inertia scheduling
- Add cost models: `SRCost`, `NSRCost`, `DCost`
- Add reserve models: `SR`, `NSR`
- Add routine `UC`
- Add routine `RTED2` to include energy storage model

4.1.14 v0.6.7 (2023-08-02)

- Version cleanup

4.1.15 v0.6.6 (2023-07-27)

- Improve routine reference
- Add routine ED, LDOPF

4.1.16 v0.6.5 (2023-06-27)

- Update documentation with auto-generated model and routine reference
- Add interface with ANDES `ams.interop.andes`
- Add routine RTED and example of RTED-TDS co-simulation
- Draft development documentation

4.1.17 v0.6.4 (2023-05-23)

- Setup PFlow and DCPF using PYPOWER

4.1.18 v0.6.3 (2023-05-22)

- Using CVXPY for draft implementation
- Improve `model`, `group`, `param` and `var` in `core`
- Refactor `routines` and `opt`
- Improve PYPOWER interface `io.pypower.system2ppc`
- Fix PYPOWER function `solver.pypower.makePTDF`

4.1.19 v0.6.2 (2023-04-23)

- Enhance docstring
- Remove unused module `utils.LazyImport`
- Remove unused module `shared`

4.1.20 v0.6.1 (2023-03-05)

- Fix incompatiability of NumPy attribute `object` in `io.matpower._get_bus_id_caller`
- Add file parser `io.pypower` for PYPOWER case file
- Deprecate PYPOWER interface `solvers.ipp`

4.1.21 v0.6.0 (2023-03-04)

- Set up PYPOWER for power flow calculation
- Add PYPOWER interface `solvers.ipp`
- Develop module `routines` for routine analysis
- Revise module `system`, `core.var`, `core.model` for routine analysis
- Set up routine `PFlow` for power flow calculation
- Add file parser `io.matpower` and `io.raw` for MATPOWER file and RAW file
- Documentation of APIs

4.1.22 v0.5 (2023-02-17)

- Develop module `system`, `main`, `cli`
- Development preparation: versioneer, documentation, etc.

4.1.23 v0.4 (2023-01)

This release outlines the package.

CHAPTER

FIVE

ROUTINE REFERENCE

Use the left navigation pane to locate the group and model and view details.

Supported Types and Routines

Type	Routines
ACED	*ACOPF*
DCED	*DCOPF*, *ED*, *EDDG*, *EDES*, *RTED*, *RTEDDG*, *RTEDES*, *RTEDVIS*
DCUC	*UC*, *UCDG*, *UCES*
DED	*DOPF*, *DOPFVIS*
PF	*DCPF*, *PFlow*, *CPF*

5.1 ACED

Type for AC-based economic dispatch.

Common Parameters: c2, c1, c0, pmax, pmin, pd, ptdf, rate_a, qd

Common Vars: pg, aBus, vBus, qg

Common Constraints: pb, lub, llb

Available routines: *ACOPF*

5.1.1 ACOPF

Standard AC optimal power flow.

Notes

1. ACOPF is solved with PYPOWER `runopf` function.
2. ACOPF formulation in AMS style is NOT DONE YET, but this does not affect the results because the data are passed to PYPOWER for solving.

Objective

Name	Description	Unit	Expression
obj	total cost		$min. \sum(c_2 p_g^2 + c_1 p_g + c_0)$

Constraints

Name	Description	Expression
pb	power balance	$\sum(p_d) - \sum(p_g) = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
aBus	a_{Bus}	Bus voltage angle	*rad*	Bus.a	
vBus	v_{Bus}	Bus voltage magnitude	*p.u.*	Bus.v	
pg	p_g	Gen active power	*p.u.*	StaticGen.p	
qg	q_g	Gen reactive power	*p.u.*	StaticGen.q	

Parameters

Name	Symbol	Description	Unit	Source
x	x	line reactance	*p.u.*	Line.x
tap	t_{ap}	transformer branch tap ratio	*float*	Line.tap
phi	ϕ	transformer branch phase shift in rad	*radian*	Line.phi
pd	p_d	active deman	*p.u.*	StaticLoad.p0
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
qd	q_d	reactive demand	*p.u.*	StaticLoad.q0

5.2 DCED

Type for DC-based economic dispatch.

Common Parameters: c2, c1, c0, pmax, pmin, pd, ptdf, rate_a

Common Vars: pg

Common Constraints: pb, lub, llb

Available routines: *DCOPF*, *ED*, *EDDG*, *EDES*, *RTED*, *RTEDDG*, *RTEDES*, *RTEDVIS*

5.2.1 DCOPF

DC optimal power flow (DCOPF).

Line flow variable *plf* is calculated as `Bf@aBus + Pfinj` after solving the problem in `_post_solve()` .

Objective

Name	Description	Unit	Expression
obj	total cost	$	$min. \sum(c_2 power(p_g, 2)) + \sum(c_1 p_g) + \sum(u_g c_0)$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e} p_{g,0} + c_{trl,e} p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e} p_{g,0} - c_{trl,e} p_{g,max} <= 0$
pb	power balance	$B_{bus} \theta_{bus} + P^{inj}_{bus} + C_l p_d + C_{sh} g_{sh} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft} \theta_{bus} - \theta_{max} <= 0$
alfub	line angle difference upper bound	$C^T_{ft} \theta_{bus} - \theta_{max} <= 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	Gen active power	$p.u.$	StaticGen.p	
aBus	θ_{bus}	Bus voltage angle	rad	Bus.a	
plf	p_{lf}	Line flow	$p.u.$		

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj

5.2.2 ED

DC-based multi-period economic dispatch (ED). Dispath interval `config.t` (T_{cfg}) is introduced, 1 [Hour] by default.

ED extends DCOPF as follows:

- Vars `pg`, `pru`, `prd` are extended to 2D
- 2D Vars `rgu` and `rgd` are introduced
- Param `ug` is sourced from `EDTSlot.ug` as commitment decisions

Notes

1. Formulations has been adjusted with interval `config.t`
2. The tie-line flow is not implemented in this model.

Objective

Name	Description	Unit	Expression
obj	total generation and reserve cost	*$*	$min. \sum(c_2(T_{cfg}p_g)^2 + c_1(T_{cfg}p_g)) + \sum(u_g c_0 1_{tl}) + \sum(c_{sr}(T_{cfg}p_{r,s}))$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e} p_{g,0} 1_{tl} + c_{trl,e} 1_{tl} p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e} p_{g,0} 1_{tl} - c_{trl,e} 1_{tl} p_{g,max} <= 0$
pb	power balance	$B_{bus} \theta_{bus} + P^{inj}_{bus} 1_{tl} + C_l p_{d,s} + C_{sh} g_{sh} 1_{tl} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f 1_{tl} - R_{ATEA} 1_{tl} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f 1_{tl} - R_{ATEA} 1_{tl} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft} \theta_{bus} - \theta_{max} 1_{tl} <= 0$
alfub	line angle difference upper bound	$C^T_{ft} \theta_{bus} - \theta_{max} 1_{tl} <= 0$
rbu	RegUp reserve balance	$S_g u_g p_{r,u} - d_{u,d} 1_{tl} = 0$
rbd	RegDn reserve balance	$S_g u_g p_{r,d} - d_{d,d} 1_{tl} = 0$
rru	RegUp reserve source	$u_g p_g + p_{r,u} - p_{g,max} 1_{tl} <= 0$
rrd	RegDn reserve source	$u_g - p_g + p_{r,d} + p_{g,min} 1_{tl} <= 0$
rgu	Gen ramping up	$p_g M_r - T_{cfg} R_{30,R} <= 0$
rgd	Gen ramping down	$-p_g M_r - T_{cfg} R_{30,R} <= 0$
prsb	spinning reserve balance	$u_g p_{g,max} 1_{tl} - p_g - p_{r,s} = 0$
rsr	spinning reserve requirement	$-S_g p_{r,s} + d_{s,r,z} <= 0$
rgu0	Initial gen ramping up	$p_g[:, 0] - p_{g,0}[:, 0] - R_{30} <= 0$
rgd0	Initial gen ramping down	$-p_g[:, 0] + p_{g,0}[:, 0] - R_{30} <= 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	2D Gen power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	2D Bus angle	*rad*	Bus.a	
plf	p_{lf}	2D Line flow	*p.u.*		
pru	$p_{r,u}$	2D RegUp power	*p.u.*		nonneg
prd	$p_{r,d}$	2D RegDn power	*p.u.*		nonneg
prs	$p_{r,s}$	spinning reserve	*p.u.*		nonneg

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
dud	$d_{u,d}$	zonal RegUp reserve requirement	NumOpDual
ddd	$d_{d,d}$	zonal RegDn reserve requirement	NumOpDual
tlv	1_{tl}	time length vector	NumOp
pds	$p_{d,s}$	Scaled load	LoadScale
Mr	M_r	Subtraction matrix for ramping	RampSub
RR30	$R_{30,R}$	Repeated ramp rate	NumHstack
dsrpz	$d_{s,r,p,z}$	zonal spinning reserve requirement in percentage	NumOpDual
dsr	$d_{s,r,z}$	zonal spinning reserve requirement	NumOpDual
ugt	u_g	input ug transpose	NumOp

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	unit commitment decisions		EDTSlot.ug
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10
cru	$c_{r,u}$	RegUp reserve coefficient	*$/(p.u.)*	SFRCost.cru
crd	$c_{r,d}$	RegDown reserve coefficient	*$/(p.u.)*	SFRCost.crd
du	d_u	RegUp reserve requirement in percentage	*%*	SFR.du
dd	d_d	RegDown reserve requirement in percentage	*%*	SFR.dd

continues on next page

Table 1 – continued from previous page

Name	Symbol	Description	Unit	Source
sd	s_d	zonal load factor for ED		EDTSlot.sd
timeslot	$t_{s,idx}$	Time slot for multi-period ED		EDTSlot.idx
R30	R_{30}	30-min ramp rate	*p.u./h*	StaticGen.R30
dsr	d_{sr}	spinning reserve requirement in percentage	*%*	SR.demand
csr	c_{sr}	cost for spinning reserve	*$/(p.u.*h)*	SRCost.csr

5.2.3 EDDG

ED with distributed generation *DG*.

Note that EDDG only inlcudes DG output power. If ESD1 is included, EDES should be used instead, otherwise there is no SOC.

Objective

Name	Description	Unit	Expression
obj	total generation and reserve cost	*$*	$min. \sum(c_2(T_{cfg}p_g)^2 + c_1(T_{cfg}p_g)) + \sum(u_g c_0 1_{tl}) + \sum(c_{sr}(T_{cfg}p_{r,s}))$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e}p_{g,0}1_{tl} + c_{trl,e}1_{tl}p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e}p_{g,0}1_{tl} - c_{trl,e}1_{tl}p_{g,max} <= 0$
pb	power balance	$B_{bus}\theta_{bus} + P^{inj}_{bus}1_{tl} + C_l p_{d,s} + C_{sh}g_{sh}1_{tl} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f\theta_{bus} - P^{inj}_f 1_{tl} - R_{ATEA}1_{tl} <= 0$
plfub	line flow upper bound	$B_f\theta_{bus} + P^{inj}_f 1_{tl} - R_{ATEA}1_{tl} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft}\theta_{bus} - \theta_{max}1_{tl} <= 0$
alfub	line angle difference upper bound	$C^T_{ft}\theta_{bus} - \theta_{max}1_{tl} <= 0$
rbu	RegUp reserve balance	$S_g u_g p_{r,u} - d_{u,d}1_{tl} = 0$
rbd	RegDn reserve balance	$S_g u_g p_{r,d} - d_{d,d}1_{tl} = 0$
rru	RegUp reserve source	$u_g p_g + p_{r,u} - p_{g,max}1_{tl} <= 0$
rrd	RegDn reserve source	$u_g - p_g + p_{r,d} + p_{g,min}1_{tl} <= 0$
rgu	Gen ramping up	$p_g M_r - T_{cfg}R_{30,R} <= 0$
rgd	Gen ramping down	$-p_g M_r - T_{cfg}R_{30,R} <= 0$
prsb	spinning reserve balance	$u_g p_{g,max}1_{tl} - p_g - p_{r,s} = 0$
rsr	spinning reserve requirement	$-S_g p_{r,s} + d_{s,r,z} <= 0$
rgu0	Initial gen ramping up	$p_g[:,0] - p_{g,0}[:,0] - R_{30} <= 0$
rgd0	Initial gen ramping down	$-p_g[:,0] + p_{g,0}[:,0] - R_{30} <= 0$
cdgb	Select DG power from pg	$C_{DG}p_g - p_{g,DG} = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	2D Gen power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	2D Bus angle	*rad*	Bus.a	
plf	p_{lf}	2D Line flow	*p.u.*		
pru	$p_{r,u}$	2D RegUp power	*p.u.*		nonneg
prd	$p_{r,d}$	2D RegDn power	*p.u.*		nonneg
prs	$p_{r,s}$	spinning reserve	*p.u.*		nonneg
pgdg	$p_{g,DG}$	DG output power	*p.u.*		

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
dud	$d_{u,d}$	zonal RegUp reserve requirement	NumOpDual
ddd	$d_{d,d}$	zonal RegDn reserve requirement	NumOpDual
tlv	1_{tl}	time length vector	NumOp
pds	$p_{d,s}$	Scaled load	LoadScale
Mr	M_r	Subtraction matrix for ramping	RampSub
RR30	$R_{30,R}$	Repeated ramp rate	NumHstack
dsrpz	$d_{s,r,p,z}$	zonal spinning reserve requirement in percentage	NumOpDual
dsr	$d_{s,r,z}$	zonal spinning reserve requirement	NumOpDual
ugt	u_g	input ug transpose	NumOp
cd	C_{DG}	Select DG power from pg	VarSelect

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	unit commitment decisions		EDTSlot.ug
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g

continues on next page

Table 2 – continued from previous page

Name	Symbol	Description	Unit	Source
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10
cru	$c_{r,u}$	RegUp reserve coefficient	*$/(p.u.)*	SFRCost.cru
crd	$c_{r,d}$	RegDown reserve coefficient	*$/(p.u.)*	SFRCost.crd
du	d_u	RegUp reserve requirement in percentage	*%*	SFR.du
dd	d_d	RegDown reserve requirement in percentage	*%*	SFR.dd
sd	s_d	zonal load factor for ED		EDTSlot.sd
timeslot	$t_{s,idx}$	Time slot for multi-period ED		EDTSlot.idx
R30	R_{30}	30-min ramp rate	*p.u./h*	StaticGen.R30
dsr	d_{sr}	spinning reserve requirement in percentage	*%*	SR.demand
csr	c_{sr}	cost for spinning reserve	*$/(p.u.*h)*	SRCost.csr
gendg	g_{DG}	gen of DG		DG.gen
gammapd	$\gamma_{p,DG}$	('Ratio of DG.pge w.r.t to that of static generator',)		DG.gammap

5.2.4 EDES

ED with energy storage *ESD1*. The bilinear term in the formulation is linearized with big-M method.

Objective

Name	Description	Unit	Expression
obj	total generation and reserve cost	*$*	$min. \sum(c_2(T_{cfg}p_g)^2 + c_1(T_{cfg}p_g)) + \sum(u_g c_0 1_{tl}) + \sum(c_{sr}(T_{cfg}p_{r,s}))$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e}p_{g,0}1_{tl} + c_{trl,e}1_{tl}p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e}p_{g,0}1_{tl} - c_{trl,e}1_{tl}p_{g,max} <= 0$
pb	power balance	$B_{bus}\theta_{bus} + P_{bus}^{inj}1_{tl} + C_l p_{d,s} + C_{sh}g_{sh}1_{tl} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f\theta_{bus} - P_f^{inj}1_{tl} - R_{ATEA}1_{tl} <= 0$
plfub	line flow upper bound	$B_f\theta_{bus} + P_f^{inj}1_{tl} - R_{ATEA}1_{tl} <= 0$
alflb	line angle difference lower bound	$-C_{ft}^T\theta_{bus} - \theta_{max}1_{tl} <= 0$
alfub	line angle difference upper bound	$C_{ft}^T\theta_{bus} - \theta_{max}1_{tl} <= 0$

continues on next page

Table 3 – continued from previous page

Name	Description	Expression
rbu	RegUp reserve balance	$S_g u_g p_{r,u} - d_{u,d} 1_{tl} = 0$
rbd	RegDn reserve balance	$S_g u_g p_{r,d} - d_{d,d} 1_{tl} = 0$
rru	RegUp reserve source	$u_g p_g + p_{r,u} - p_{g,max} 1_{tl} <= 0$
rrd	RegDn reserve source	$u_g - p_g + p_{r,d} + p_{g,min} 1_{tl} <= 0$
rgu	Gen ramping up	$p_g M_r - T_{cfg} R_{30,R} <= 0$
rgd	Gen ramping down	$-p_g M_r - T_{cfg} R_{30,R} <= 0$
prsb	spinning reserve balance	$u_g p_{g,max} 1_{tl} - p_g - p_{r,s} = 0$
rsr	spinning reserve requirement	$-S_g p_{r,s} + d_{s,r,z} <= 0$
rgu0	Initial gen ramping up	$p_g[:,0] - p_{g,0}[:,0] - R_{30} <= 0$
rgd0	Initial gen ramping down	$-p_g[:,0] + p_{g,0}[:,0] - R_{30} <= 0$
cdgb	Select DG power from pg	$C_{DG} p_g - p_{g,DG} = 0$
SOClb	SOC lower bound	$-SOC + SOC_{min} <= 0$
SOCub	SOC upper bound	$SOC - SOC_{max} <= 0$
cdb	Charging decision bound	$u_{c,ESD} + u_{d,ESD} - 1 = 0$
cesb	Select ESD1 power from pg	$C_{ESD} p_g + z_{c,ESD} - z_{d,ESD} = 0$
zce1	zce bound 1	$-z_{c,ESD} + p_{c,ESD} <= 0$
zce2	zce bound 2	$z_{c,ESD} - p_{c,ESD} - M_{big}(1 - u_{c,ESD}) <= 0$
zce3	zce bound 3	$z_{c,ESD} - M_{big} u_{c,ESD} <= 0$
zde1	zde bound 1	$-z_{d,ESD} + p_{d,ESD} <= 0$
zde2	zde bound 2	$z_{d,ESD} - p_{d,ESD} - M_{big}(1 - u_{d,ESD}) <= 0$
zde3	zde bound 3	$z_{d,ESD} - M_{big} u_{d,ESD} <= 0$
SOCb	ESD1 SOC balance	$E_{n,R} SOC M_{r,ES} - T_{cfg} \eta_{c,R} z_{c,ESD}[:,1:] + T_{cfg} R_{\eta_d,R} z_{d,ESD}[:,1:] = 0$
SOCb0	ESD1 SOC initial balance	$E_n SOC[:,0] - SOC_{init} - T_{cfg} \eta_c z_{c,ESD}[:,0] + T_{cfg} \frac{1}{\eta_d} z_{d,ESD}[:,0] = 0$
SOCr	SOC requirement	$SOC[:,-1] - SOC_{init} = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	2D Gen power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	2D Bus angle	*rad*	Bus.a	
plf	p_{lf}	2D Line flow	*p.u.*		
pru	$p_{r,u}$	2D RegUp power	*p.u.*		nonneg
prd	$p_{r,d}$	2D RegDn power	*p.u.*		nonneg
prs	$p_{r,s}$	spinning reserve	*p.u.*		nonneg
pgdg	$p_{g,DG}$	DG output power	*p.u.*		
SOC	SOC	ESD1 State of Charge	%		pos
pce	$p_{c,ESD}$	ESD1 charging power	*p.u.*		nonneg
pde	$p_{d,ESD}$	ESD1 discharging power	*p.u.*		nonneg
uce	$u_{c,ESD}$	ESD1 charging decision			boolean
ude	$u_{d,ESD}$	ESD1 discharging decision			boolean
zce	$z_{c,ESD}$	Aux var for charging, $z_{c,ESD} = u_{c,ESD} * p_{c,ESD}$			nonneg
zde	$z_{d,ESD}$	Aux var for discharging, $z_{d,ESD} = u_{d,ESD} * p_{d,ESD}$			nonneg

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
dud	$d_{u,d}$	zonal RegUp reserve requirement	NumOpDual
ddd	$d_{d,d}$	zonal RegDn reserve requirement	NumOpDual
tlv	1_{tl}	time length vector	NumOp
pds	$p_{d,s}$	Scaled load	LoadScale
Mr	M_r	Subtraction matrix for ramping	RampSub
RR30	$R_{30,R}$	Repeated ramp rate	NumHstack
dsrpz	$d_{s,r,p,z}$	zonal spinning reserve requirement in percentage	NumOpDual
dsr	$d_{s,r,z}$	zonal spinning reserve requirement	NumOpDual
ugt	u_g	input ug transpose	NumOp
cd	C_{DG}	Select DG power from pg	VarSelect
REtaD	$\frac{1}{\eta_d}$		NumOp
Mb	M_{big}	10 times of max of pmax as big M	NumOp
ce	C_{ESD}	Select zue from pg	VarSelect
Mre	$M_{r,ES}$	Subtraction matrix for SOC	RampSub
EnR	$E_{n,R}$	Repeated En as 2D matrix, (ng, ng-1)	NumHstack
EtaCR	$\eta_{c,R}$	Repeated Etac as 2D matrix, (ng, ng-1)	NumHstack
REtaDR	$R_{\eta_d,R}$	Repeated REtaD as 2D matrix, (ng, ng-1)	NumHstack

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	unit commitment decisions		EDTSlot.ug
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj

continues on next page

Table 4 – continued from previous page

Name	Symbol	Description	Unit	Source
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10
cru	$c_{r,u}$	RegUp reserve coefficient	*$/(p.u.)*	SFRCost.cru
crd	$c_{r,d}$	RegDown reserve coefficient	*$/(p.u.)*	SFRCost.crd
du	d_u	RegUp reserve requirement in percentage	*%*	SFR.du
dd	d_d	RegDown reserve requirement in percentage	*%*	SFR.dd
sd	s_d	zonal load factor for ED		EDTSlot.sd
timeslot	$t_{s,idx}$	Time slot for multi-period ED		EDTSlot.idx
R30	R_{30}	30-min ramp rate	*p.u./h*	StaticGen.R30
dsr	d_{sr}	spinning reserve requirement in percentage	*%*	SR.demand
csr	c_{sr}	cost for spinning reserve	*$/(p.u.*h)*	SRCost.csr
gendg	g_{DG}	gen of DG		DG.gen
gammapd	$\gamma_{p,DG}$	('Ratio of DG.pge w.r.t to that of static generator',)		DG.gammap
En	E_n	Rated energy capacity	*MWh*	ESD1.En
SOCmax	SOC_{max}	Maximum allowed value for SOC in limiter	*%*	ESD1.SOCmax
SOCmin	SOC_{min}	Minimum required value for SOC in limiter	*%*	ESD1.SOCmin
SOCinit	SOC_{init}	Initial SOC	*%*	ESD1.SOCinit
EtaC	η_c	Efficiency during charging	*%*	ESD1.EtaC
EtaD	η_d	Efficiency during discharging	*%*	ESD1.EtaD
genesd	g_{ESD}	gen of ESD1		ESD1.gen
gammapesd	$\gamma_{p,ESD}$	Ratio of ESD1.pge w.r.t to that of static generator		ESD1.gammap

5.2.5 RTED

DC-based real-time economic dispatch (RTED). RTED extends DCOPF with:

- Mapping dicts to interface with ANDES
- Function `dc2ac` to do the AC conversion
- Vars for SFR reserve: `pru` and `prd`
- Param for linear SFR cost: `cru` and `crd`
- Param for SFR requirement: `du` and `dd`
- Param for ramping: start point `pg0` and ramping limit `R10`
- Param `pg0`, which can be retrieved from dynamic simulation results.

The function `dc2ac` sets the `vBus` value from solved ACOPF. Without this conversion, dynamic simulation might fail due to the gap between DC-based dispatch results and AC-based dynamic initialization.

Notes

1. Formulations has been adjusted with interval `config.t`, 5/60 [Hour] by default.
2. The tie-line flow has not been implemented in formulations.

Objective

Name	Description	Unit	Expression
obj	total generation and reserve cost	$	$min. \sum(c_2 power(T_{cfg} p_g, 2)) + \sum(c_1 (T_{cfg} p_g)) + u_g c_0 + \sum(c_{r,u} p_{r,u} + c_{r,d} p_{r,d})$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e} p_{g,0} + c_{trl,e} p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e} p_{g,0} - c_{trl,e} p_{g,max} <= 0$
pb	power balance	$B_{bus} \theta_{bus} + P^{inj}_{bus} + C_l p_d + C_{sh} g_{sh} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft} \theta_{bus} - \theta_{max} <= 0$
alfub	line angle difference upper bound	$C^T_{ft} \theta_{bus} - \theta_{max} <= 0$
rbu	RegUp reserve balance	$S_g u_g p_{r,u} - d_{u,d} = 0$
rbd	RegDn reserve balance	$S_g u_g p_{r,d} - d_{d,d} = 0$
rru	RegUp reserve source	$u_g p_g + p_{r,u} - u_g p_{g,max} <= 0$
rrd	RegDn reserve source	$u_g - p_g + p_{r,d} + u_g p_{g,min} <= 0$
rgu	Gen ramping up	$u_g p_g - p_{g,0} - R_{10} <= 0$
rgd	Gen ramping down	$u_g - p_g + p_{g,0} - R_{10} <= 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	Gen active power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	Bus voltage angle	*rad*	Bus.a	
plf	p_{lf}	Line flow	*p.u.*		
pru	$p_{r,u}$	RegUp reserve	*p.u.*		nonneg
prd	$p_{r,d}$	RegDn reserve	*p.u.*		nonneg

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
dud	$d_{u,d}$	zonal RegUp reserve requirement	NumOpDual
ddd	$d_{d,d}$	zonal RegDn reserve requirement	NumOpDual

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10
cru	$c_{r,u}$	RegUp reserve coefficient	*$/(p.u.)*	SFRCost.cru
crd	$c_{r,d}$	RegDown reserve coefficient	*$/(p.u.)*	SFRCost.crd
du	d_u	RegUp reserve requirement in percentage	*%*	SFR.du
dd	d_d	RegDown reserve requirement in percentage	*%*	SFR.dd

5.2.6 RTEDDG

RTED with distributed generator *DG*.

Note that RTEDDG only inlcudes DG output power. If ESD1 is included, RTEDES should be used instead, otherwise there is no SOC.

Objective

Name	Description	Unit	Expression
obj	total generation and reserve cost	$	$min. \sum(c_2 power(T_{cfg} p_g, 2)) + \sum(c_1(T_{cfg} p_g)) + u_g c_0 + \sum(c_{r,u} p_{r,u} + c_{r,d} p_{r,d})$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e} p_{g,0} + c_{trl,e} p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e} p_{g,0} - c_{trl,e} p_{g,max} <= 0$
pb	power balance	$B_{bus} \theta_{bus} + P^{inj}_{bus} + C_l p_d + C_{sh} g_{sh} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft} \theta_{bus} - \theta_{max} <= 0$
alfub	line angle difference upper bound	$C^T_{ft} \theta_{bus} - \theta_{max} <= 0$
rbu	RegUp reserve balance	$S_g u_g p_{r,u} - d_{u,d} = 0$
rbd	RegDn reserve balance	$S_g u_g p_{r,d} - d_{d,d} = 0$
rru	RegUp reserve source	$u_g p_g + p_{r,u} - u_g p_{g,max} <= 0$
rrd	RegDn reserve source	$u_g - p_g + p_{r,d} + u_g p_{g,min} <= 0$
rgu	Gen ramping up	$u_g p_g - p_{g,0} - R_{10} <= 0$
rgd	Gen ramping down	$u_g - p_g + p_{g,0} - R_{10} <= 0$
cdgb	Select DG power from pg	$C_{DG} p_g - p_{g,DG} = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	Gen active power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	Bus voltage angle	*rad*	Bus.a	
plf	p_{lf}	Line flow	*p.u.*		
pru	$p_{r,u}$	RegUp reserve	*p.u.*		nonneg
prd	$p_{r,d}$	RegDn reserve	*p.u.*		nonneg
pgdg	$p_{g,DG}$	DG output power	*p.u.*		

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
dud	$d_{u,d}$	zonal RegUp reserve requirement	NumOpDual
ddd	$d_{d,d}$	zonal RegDn reserve requirement	NumOpDual
cd	C_{DG}	Select DG power from pg	VarSelect

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10
cru	$c_{r,u}$	RegUp reserve coefficient	*$/(p.u.)*	SFRCost.cru
crd	$c_{r,d}$	RegDown reserve coefficient	*$/(p.u.)*	SFRCost.crd
du	d_u	RegUp reserve requirement in percentage	*%*	SFR.du
dd	d_d	RegDown reserve requirement in percentage	*%*	SFR.dd
gendg	g_{DG}	gen of DG		DG.gen
gammapd	$\gamma_{p,DG}$	('Ratio of DG.pge w.r.t to that of static generator',)		DG.gammap

5.2.7 RTEDES

RTED with energy storage *ESD1*. The bilinear term in the formulation is linearized with big-M method.

Objective

Name	Description	Unit	Expression
obj	total generation and reserve cost	$	$min. \sum(c_2 power(T_{cfg} p_g, 2)) + \sum(c_1 (T_{cfg} p_g)) + u_g c_0 + \sum(c_{r,u} p_{r,u} + c_{r,d} p_{r,d})$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e} p_{g,0} + c_{trl,e} p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e} p_{g,0} - c_{trl,e} p_{g,max} <= 0$
pb	power balance	$B_{bus} \theta_{bus} + P^{inj}_{bus} + C_l p_d + C_{sh} g_{sh} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft} \theta_{bus} - \theta_{max} <= 0$
alfub	line angle difference upper bound	$C^T_{ft} \theta_{bus} - \theta_{max} <= 0$
rbu	RegUp reserve balance	$S_g u_g p_{r,u} - d_{u,d} = 0$
rbd	RegDn reserve balance	$S_g u_g p_{r,d} - d_{d,d} = 0$
rru	RegUp reserve source	$u_g p_g + p_{r,u} - u_g p_{g,max} <= 0$
rrd	RegDn reserve source	$u_g - p_g + p_{r,d} + u_g p_{g,min} <= 0$
rgu	Gen ramping up	$u_g p_g - p_{g,0} - R_{10} <= 0$
rgd	Gen ramping down	$u_g - p_g + p_{g,0} - R_{10} <= 0$
cdgb	Select DG power from pg	$C_{DG} p_g - p_{g,DG} = 0$
SOClb	SOC lower bound	$-SOC + SOC_{min} <= 0$
SOCub	SOC upper bound	$SOC - SOC_{max} <= 0$
cdb	Charging decision bound	$u_{c,ESD} + u_{d,ESD} - 1 = 0$
cesb	Select ESD1 power from pg	$C_{ESD} p_g + z_{c,ESD} - z_{d,ESD} = 0$
zce1	zce bound 1	$-z_{c,ESD} + p_{c,ESD} <= 0$
zce2	zce bound 2	$z_{c,ESD} - p_{c,ESD} - M_{big}(1 - u_{c,ESD}) <= 0$
zce3	zce bound 3	$z_{c,ESD} - M_{big} u_{c,ESD} <= 0$
zde1	zde bound 1	$-z_{d,ESD} + p_{d,ESD} <= 0$
zde2	zde bound 2	$z_{d,ESD} - p_{d,ESD} - M_{big}(1 - u_{d,ESD}) <= 0$
zde3	zde bound 3	$z_{d,ESD} - M_{big} u_{d,ESD} <= 0$
SOCb	ESD1 SOC balance	$E_n(SOC - SOC_{init}) - T_{cfg} \eta_c z_{c,ESD} + T_{cfg} \frac{1}{\eta_d} z_{d,ESD} = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	Gen active power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	Bus voltage angle	*rad*	Bus.a	
plf	p_{lf}	Line flow	*p.u.*		
pru	$p_{r,u}$	RegUp reserve	*p.u.*		nonneg
prd	$p_{r,d}$	RegDn reserve	*p.u.*		nonneg
pgdg	$p_{g,DG}$	DG output power	*p.u.*		
SOC	SOC	ESD1 State of Charge	*%*		pos
pce	$p_{c,ESD}$	ESD1 charging power	*p.u.*		nonneg
pde	$p_{d,ESD}$	ESD1 discharging power	*p.u.*		nonneg
uce	$u_{c,ESD}$	ESD1 charging decision			boolean
ude	$u_{d,ESD}$	ESD1 discharging decision			boolean
zce	$z_{c,ESD}$	Aux var for charging, $z_{c,ESD} = u_{c,ESD} * p_{c,ESD}$			nonneg
zde	$z_{d,ESD}$	Aux var for discharging, $z_{d,ESD} = u_{d,ESD} * p_{d,ESD}$			nonneg

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
dud	$d_{u,d}$	zonal RegUp reserve requirement	NumOpDual
ddd	$d_{d,d}$	zonal RegDn reserve requirement	NumOpDual
cd	C_{DG}	Select DG power from pg	VarSelect
REtaD	$\frac{1}{\eta_d}$		NumOp
Mb	M_{big}	10 times of max of pmax as big M	NumOp
ce	C_{ESD}	Select zue from pg	VarSelect

Parameters

Name	Symbol	Description	Unit	Source
c2	c_{2}	Gen cost coefficient 2	*\$/(p.u.^2)*	GCost.c2
c1	c_{1}	Gen cost coefficient 1	*\$/(p.u.)*	GCost.c1
c0	c_{0}	Gen cost coefficient 0	*\$*	GCost.c0
ug	u_{g}	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_{d}	active demand	*p.u.*	StaticLoad.p0

continues on next page

Table 5 – continued from previous page

Name	Symbol	Description	Unit	Source
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10
cru	$c_{r,u}$	RegUp reserve coefficient	*$/(p.u.)*	SFRCost.cru
crd	$c_{r,d}$	RegDown reserve coefficient	*$/(p.u.)*	SFRCost.crd
du	d_u	RegUp reserve requirement in percentage	*%*	SFR.du
dd	d_d	RegDown reserve requirement in percentage	*%*	SFR.dd
gendg	g_{DG}	gen of DG		DG.gen
gammapd	$\gamma_{p,DG}$	('Ratio of DG.pge w.r.t to that of static generator',)		DG.gammap
En	E_n	Rated energy capacity	*MWh*	ESD1.En
SOCmax	SOC_{max}	Maximum allowed value for SOC in limiter	*%*	ESD1.SOCmax
SOCmin	SOC_{min}	Minimum required value for SOC in limiter	*%*	ESD1.SOCmin
SOCinit	SOC_{init}	Initial SOC	*%*	ESD1.SOCinit
EtaC	η_c	Efficiency during charging	*%*	ESD1.EtaC
EtaD	η_d	Efficiency during discharging	*%*	ESD1.EtaD
genesd	g_{ESD}	gen of ESD1		ESD1.gen
gammapesd	$\gamma_{p,ESD}$	Ratio of ESD1.pge w.r.t to that of static generator		ESD1.gammap

5.2.8 RTEDVIS

RTED with virtual inertia scheduling.

Reference:

[1] B. She, F. Li, H. Cui, J. Wang, Q. Zhang and R. Bo, "Virtual Inertia Scheduling (VIS) for Real-time Economic Dispatch of IBRs-penetrated Power Systems," in IEEE Transactions on Sustainable Energy, doi: 10.1109/TSTE.2023.3319307.

Objective

Name	Description	Unit	Expression
obj	total generation and reserve cost	*$*	$min. \sum(c_2 power(p_g, 2)) + \sum(c_1(T_{cfg} p_g)) + u_g c_0 + \sum(c_{r,u}(T_{cfg} p_{r,u}) + c_{r,d}(T_{cfg} p_{r,d})) + \sum(c_m M + c_d D)$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e} p_{g,0} + c_{trl,e} p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e} p_{g,0} - c_{trl,e} p_{g,max} <= 0$
pb	power balance	$B_{bus}\theta_{bus} + P^{inj}_{bus} + C_l p_d + C_{sh} g_{sh} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft}\theta_{bus} - \theta_{max} <= 0$
alfub	line angle difference upper bound	$C^T_{ft}\theta_{bus} - \theta_{max} <= 0$
rbu	RegUp reserve balance	$S_g u_g p_{r,u} - d_{u,d} = 0$
rbd	RegDn reserve balance	$S_g u_g p_{r,d} - d_{d,d} = 0$
rru	RegUp reserve source	$u_g p_g + p_{r,u} - u_g p_{g,max} <= 0$
rrd	RegDn reserve source	$u_g - p_g + p_{r,d} + u_g p_{g,min} <= 0$
rgu	Gen ramping up	$u_g p_g - p_{g,0} - R_{10} <= 0$
rgd	Gen ramping down	$u_g - p_g + p_{g,0} - R_{10} <= 0$
Mub	M upper bound	$M - M_{max} <= 0$
Dub	D upper bound	$D - D_{max} <= 0$
Mreq	Emulated inertia requirement	$-S_g M + d_{v,m} = 0$
Dreq	Emulated damping requirement	$-S_g D + d_{v,d} = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	Gen active power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	Bus voltage angle	*rad*	Bus.a	
plf	p_{lf}	Line flow	*p.u.*		
pru	$p_{r,u}$	RegUp reserve	*p.u.*		nonneg
prd	$p_{r,d}$	RegDn reserve	*p.u.*		nonneg
M	M	Emulated startup time constant (M=2H)	*s*		nonneg
D	D	Emulated damping coefficient	*p.u.*		nonneg

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
dud	$d_{u,d}$	zonal RegUp reserve requirement	NumOpDual
ddd	$d_{d,d}$	zonal RegDn reserve requirement	NumOpDual
gvsg	S_g	Sum VSG vars vector in shape of zone	ZonalSum

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10
cru	$c_{r,u}$	RegUp reserve coefficient	*$/(p.u.)*	SFRCost.cru
crd	$c_{r,d}$	RegDown reserve coefficient	*$/(p.u.)*	SFRCost.crd
du	d_u	RegUp reserve requirement in percentage	*%*	SFR.du
dd	d_d	RegDown reserve requirement in percentage	*%*	SFR.dd
cm	c_m	Virtual inertia cost	*$/s*	VSGCost.cm
cd	c_d	Virtual damping cost	*$/(p.u.)*	VSGCost.cd
zvsg	$z_{one,vsg}$	VSG zone		VSG.zone
Mmax	M_{max}	Maximum inertia emulation	*s*	VSG.Mmax
Dmax	D_{max}	Maximum damping emulation	*p.u.*	VSG.Dmax
dvm	$d_{v,m}$	Emulated inertia requirement	*s*	VSGR.dvm
dvd	$d_{v,d}$	Emulated damping requirement	*p.u.*	VSGR.dvd

5.3 DCUC

Type for DC-based unit commitment.

Available routines: *UC*, *UCDG*, *UCES*

5.3.1 UC

DC-based unit commitment (UC): The bilinear term in the formulation is linearized with big-M method.

Non-negative var *pdu* is introduced as unserved load with its penalty *cdp*.

Constraints include power balance, ramping, spinning reserve, non-spinning reserve, minimum ON/OFF duration. The cost inludes generation cost, startup cost, shutdown cost, spinning reserve cost, non-spinning reserve cost, and unserved load penalty.

Method `_initial_guess` is used to make initial guess for commitment decision if all generators are online at initial. It is a simple heuristic method, which may not be optimal.

Notes

1. Formulations has been adjusted with interval `config.t`
3. The tie-line flow has not been implemented in formulations.

References

1. Huang, Y., Pardalos, P. M., & Zheng, Q. P. (2017). Electrical power unit commitment: deterministic and two-stage stochastic programming models and algorithms. Springer.

2. D. A. Tejada-Arango, S. Lumbreras, P. Sánchez-Martín and A. Ramos, "Which Unit-Commitment Formulation is Best? A Comparison Framework," in IEEE Transactions on Power Systems, vol. 35, no. 4, pp. 2926-2936, July 2020, doi: 10.1109/TPWRS.2019.2962024.

Objective

Name	Description	Unit	Expression
obj	total cost	$	$min. \sum(c_2 (T_{cfg} z_{u_g})^2 + c_1 (T_{cfg} z_{u_g})) + \sum(u_g c_0 1_{tl}) + \sum(c_{su} v_{g,d} + c_{sd} w_{g,d}) + \sum(c_{sr} p_{r,s}) + \sum(c_{nsr} p_{r,ns}) + \sum(c_{d,p} p_{d,u})$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n} p_{g,0} u_{g,d} + c_{trl} p_{g,min} u_{g,d} <= 0$
pgub	pg max	$p_g - c_{trl,n} p_{g,0} u_{g,d} - c_{trl} p_{g,max} u_{g,d} <= 0$
pb	power balance	$B_{bus}\theta_{bus} + P^{inj}_{bus} 1_{tl} + C_l(p_{d,s} - p_{d,u}) + C_{sh} g_{sh} 1_{tl} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} 1_{tl} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} 1_{tl} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft}\theta_{bus} - \theta_{max} 1_{tl} <= 0$
alfub	line angle difference upper bound	$C^T_{ft}\theta_{bus} - \theta_{max} 1_{tl} <= 0$
prsb	spinning reserve balance	$u_{g,d} p_{g,max} 1_{tl} - z_{u_g} - p_{r,s} = 0$
rsr	spinning reserve requirement	$-S_g p_{r,s} + d_{s,r,z} <= 0$
prnsb	non-spinning reserve balance	$1 - u_{g,d} p_{g,max} 1_{tl} - p_{r,ns} = 0$
rnsr	non-spinning reserve requirement	$-S_g p_{r,ns} + d_{nsr} <= 0$
actv	startup action	$u_{g,d} M_r - v_{g,d}[:, 1:] = 0$
actv0	initial startup action	$u_{g,d}[:, 0] - u_g[:, 0] - v_{g,d}[:, 0] = 0$
actw	shutdown action	$-u_{g,d} M_r - w_{g,d}[:, 1:] = 0$
actw0	initial shutdown action	$-u_{g,d}[:, 0] + u_g[:, 0] - w_{g,d}[:, 0] = 0$
zuglb	zug lower bound	$-z_{u_g} + p_g <= 0$
zugub	zug upper bound	$z_{u_g} - p_g - M_{zug}(1 - u_{g,d}) <= 0$
zugub2	zug upper bound	$z_{u_g} - M_{zug} u_{g,d} <= 0$
don	minimum online duration	$T_{on} v_{g,d} - u_{g,d} <= 0$
doff	minimum offline duration	$T_{off} w_{g,d} - (1 - u_{g,d}) <= 0$
pdumax	unserved demand upper bound	$p_{d,u} - p^+_{d,s} c_{trl,d} 1_{tl} <= 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	2D Gen power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	2D Bus angle	*rad*	Bus.a	
plf	p_{lf}	2D Line flow	*p.u.*		
prs	$p_{r,s}$	2D Spinning reserve	*p.u.*		nonneg
prns	$p_{r,ns}$	2D Non-spinning reserve			nonneg
ugd	$u_{g,d}$	commitment decision		StaticGen.u	boolean
vgd	$v_{g,d}$	startup action		StaticGen.u	boolean
wgd	$w_{g,d}$	shutdown action		StaticGen.u	boolean
zug	z_{ug}	Aux var, $z_{ug} = u_{g,d} * p_g$			pos
pdu	$p_{d,u}$	unserved demand	*p.u.*		nonneg

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Reshaped controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Reshaped non-controllability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
tlv	1_{tl}	time length vector	NumOp
pds	$p_{d,s}$	Scaled load	LoadScale
Mr	M_r	Subtraction matrix for ramping	RampSub
RR30	$R_{30,R}$	Repeated ramp rate	NumHstack
dsrpz	$d_{s,r,p,z}$	zonal spinning reserve requirement in percentage	NumOpDual
dsr	$d_{s,r,z}$	zonal spinning reserve requirement	NumOpDual
dnsrpz	$d_{nsr,p,z}$	zonal non-spinning reserve requirement in percentage	NumOpDual
dnsr	d_{nsr}	zonal non-spinning reserve requirement	NumOpDual
Mzug	M_{zug}	10 times of max of pmax as big M for zug	NumOp
Con	T_{on}	minimum ON coefficient	MinDur
Coff	T_{off}	minimum OFF coefficient	MinDur
pdsp	$p_{d,s}^{+}$	positive demand	NumOp

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10
sd	s_d	zonal load factor for UC		UCTSlot.sd

continues on next page

Table 7 – continued from previous page

Name	Symbol	Description	Unit	Source
timeslot	$t_{s,idx}$	Time slot for multi-period UC		UCTSlot.idx
R30	R_{30}	30-min ramp rate	*p.u./h*	StaticGen.R30
dsr	d_{sr}	spinning reserve requirement in percentage	*%*	SR.demand
csr	c_{sr}	cost for spinning reserve	*$/(p.u.*h)*	SRCost.csr
cnsr	c_{nsr}	cost for non-spinning reserve	*$/(p.u.*h)*	NSRCost.cnsr
dnsr	d_{nsr}	non-spinning reserve requirement in percentage	*%*	NSR.demand
csu	c_{su}	startup cost	*$*	GCost.csu
csd	c_{sd}	shutdown cost	*$*	GCost.csd
cdp	$c_{d,p}$	penalty for unserved load	*$/(p.u.*h)*	DCost.cdp
dctrl	$c_{trl,d}$	load controllability		StaticLoad.ctrl
td1	t_{d1}	minimum ON duration	*h*	StaticGen.td1
td2	t_{d2}	minimum OFF duration	*h*	StaticGen.td2

5.3.2 UCDG

UC with distributed generation *DG*.

Note that UCDG only inlcudes DG output power. If ESD1 is included, UCES should be used instead, otherwise there is no SOC.

Objective

Name	Description	Unit	Expression
obj	total cost	*$*	$min. \sum(c_2(T_{cfg} z_{u_g})^2 + c_1(T_{cfg} z_{u_g})) + \sum(u_g c_0 1_{tl}) + \sum(c_{su} v_{g,d} + c_{sd} w_{g,d}) + \sum(c_{sr} p_{r,s}) + \sum(c_{nsr} p_{r,ns}) + \sum(c_{d,p} p_{d,u})$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n} p_{g,0} u_{g,d} + c_{trl} p_{g,min} u_{g,d} <= 0$
pgub	pg max	$p_g - c_{trl,n} p_{g,0} u_{g,d} - c_{trl} p_{g,max} u_{g,d} <= 0$
pb	power balance	$B_{bus} \theta_{bus} + P^{inj}_{bus} 1_{tl} + C_l (p_{d,s} - p_{d,u}) + C_{sh} g_{sh} 1_{tl} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} 1_{tl} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} 1_{tl} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft} \theta_{bus} - \theta_{max} 1_{tl} <= 0$
alfub	line angle difference upper bound	$C^T_{ft} \theta_{bus} - \theta_{max} 1_{tl} <= 0$
prsb	spinning reserve balance	$u_{g,d} p_{g,max} 1_{tl} - z_{u_g} - p_{r,s} = 0$
rsr	spinning reserve requirement	$-S_g p_{r,s} + d_{s,r,z} <= 0$
prnsb	non-spinning reserve balance	$1 - u_{g,d} p_{g,max} 1_{tl} - p_{r,ns} = 0$
rnsr	non-spinning reserve requirement	$-S_g p_{r,ns} + d_{nsr} <= 0$
actv	startup action	$u_{g,d} M_r - v_{g,d}[:, 1:] = 0$
actv0	initial startup action	$u_{g,d}[:, 0] - u_g[:, 0] - v_{g,d}[:, 0] = 0$
actw	shutdown action	$-u_{g,d} M_r - w_{g,d}[:, 1:] = 0$
actw0	initial shutdown action	$-u_{g,d}[:, 0] + u_g[:, 0] - w_{g,d}[:, 0] = 0$
zuglb	zug lower bound	$-z_{u_g} + p_g <= 0$
zugub	zug upper bound	$z_{u_g} - p_g - M_{zug}(1 - u_{g,d}) <= 0$
zugub2	zug upper bound	$z_{u_g} - M_{zug} u_{g,d} <= 0$
don	minimum online duration	$T_{on} v_{g,d} - u_{g,d} <= 0$
doff	minimum offline duration	$T_{off} w_{g,d} - (1 - u_{g,d}) <= 0$
pdumax	unserved demand upper bound	$p_{d,u} - p^+_{d,s} c_{trl,d} 1_{tl} <= 0$
cdgb	Select DG power from pg	$C_{DG} p_g - p_{g,DG} = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	2D Gen power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	2D Bus angle	*rad*	Bus.a	
plf	p_{lf}	2D Line flow	*p.u.*		
prs	$p_{r,s}$	2D Spinning reserve	*p.u.*		nonneg
prns	$p_{r,ns}$	2D Non-spinning reserve			nonneg
ugd	$u_{g,d}$	commitment decision		StaticGen.u	boolean
vgd	$v_{g,d}$	startup action		StaticGen.u	boolean
wgd	$w_{g,d}$	shutdown action		StaticGen.u	boolean
zug	z_{ug}	Aux var, $z_{ug} = u_{g,d} * p_g$			pos
pdu	$p_{d,u}$	unserved demand	*p.u.*		nonneg
pgdg	$p_{g,DG}$	DG output power	*p.u.*		

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Reshaped controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Reshaped non-controllability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
tlv	1_{tl}	time length vector	NumOp
pds	$p_{d,s}$	Scaled load	LoadScale
Mr	M_r	Subtraction matrix for ramping	RampSub
RR30	$R_{30,R}$	Repeated ramp rate	NumHstack
dsrpz	$d_{s,r,p,z}$	zonal spinning reserve requirement in percentage	NumOpDual
dsr	$d_{s,r,z}$	zonal spinning reserve requirement	NumOpDual
dnsrpz	$d_{nsr,p,z}$	zonal non-spinning reserve requirement in percentage	NumOpDual
dnsr	d_{nsr}	zonal non-spinning reserve requirement	NumOpDual
Mzug	M_{zug}	10 times of max of pmax as big M for zug	NumOp
Con	T_{on}	minimum ON coefficient	MinDur
Coff	T_{off}	minimum OFF coefficient	MinDur
pdsp	$p_{d,s}^{+}$	positive demand	NumOp
cd	C_{DG}	Select DG power from pg	VarSelect

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10

continues on next page

Table 8 – continued from previous page

Name	Symbol	Description	Unit	Source
sd	s_d	zonal load factor for UC		UCTSlot.sd
timeslot	$t_{s,idx}$	Time slot for multi-period UC		UCTSlot.idx
R30	R_{30}	30-min ramp rate	*p.u./h*	StaticGen.R30
dsr	d_{sr}	spinning reserve requirement in percentage	*%*	SR.demand
csr	c_{sr}	cost for spinning reserve	*$/(p.u.*h)*	SRCost.csr
cnsr	c_{nsr}	cost for non-spinning reserve	*$/(p.u.*h)*	NSRCost.cnsr
dnsr	d_{nsr}	non-spinning reserve requirement in percentage	*%*	NSR.demand
csu	c_{su}	startup cost	*$*	GCost.csu
csd	c_{sd}	shutdown cost	*$*	GCost.csd
cdp	$c_{d,p}$	penalty for unserved load	*$/(p.u.*h)*	DCost.cdp
dctrl	$c_{trl,d}$	load controllability		StaticLoad.ctrl
td1	t_{d1}	minimum ON duration	*h*	StaticGen.td1
td2	t_{d2}	minimum OFF duration	*h*	StaticGen.td2
gendg	g_{DG}	gen of DG		DG.gen
gammapd	$\gamma_{p,DG}$	('Ratio of DG.pge w.r.t to that of static generator',)		DG.gammap

5.3.3 UCES

UC with energy storage *ESD1*.

Objective

Name	Description	Unit	Expression
obj	total cost	*$*	$min. \sum(c_2(T_{cfg} z_{u_g})^2 + c_1(T_{cfg} z_{u_g})) + \sum(u_g c_0 1_{tl}) + \sum(c_{su} v_{g,d} + c_{sd} w_{g,d}) + \sum(c_{sr} p_{r,s}) + \sum(c_{nsr} p_{r,ns}) + \sum(c_{d,p} p_{d,u})$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n} p_{g,0} u_{g,d} + c_{trl} p_{g,min} u_{g,d} <= 0$
pgub	pg max	$p_g - c_{trl,n} p_{g,0} u_{g,d} - c_{trl} p_{g,max} u_{g,d} <= 0$
pb	power balance	$B_{bus} \theta_{bus} + P^{inj}_{bus} 1_{tl} + C_l(p_{d,s} - p_{d,u}) + C_{sh} g_{sh} 1_{tl} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} 1_{tl} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} 1_{tl} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft} \theta_{bus} - \theta_{max} 1_{tl} <= 0$
alfub	line angle difference upper bound	$C^T_{ft} \theta_{bus} - \theta_{max} 1_{tl} <= 0$
prsb	spinning reserve balance	$u_{g,d} p_{g,max} 1_{tl} - z_{u_g} - p_{r,s} = 0$
rsr	spinning reserve requirement	$-S_g p_{r,s} + d_{s,r,z} <= 0$
prnsb	non-spinning reserve balance	$1 - u_{g,d} p_{g,max} 1_{tl} - p_{r,ns} = 0$
rnsr	non-spinning reserve requirement	$-S_g p_{r,ns} + d_{nsr} <= 0$
actv	startup action	$u_{g,d} M_r - v_{g,d}[:, 1:] = 0$
actv0	initial startup action	$u_{g,d}[:, 0] - u_g[:, 0] - v_{g,d}[:, 0] = 0$

continues on next page

Table 9 – continued from previous page

Name	Description	Expression
actw	shutdown action	$-u_{g,d}M_r - w_{g,d}[:, 1:] = 0$
actw0	initial shutdown action	$-u_{g,d}[:, 0] + u_g[:, 0] - w_{g,d}[:, 0] = 0$
zuglb	zug lower bound	$-z_{u_g} + p_g <= 0$
zugub	zug upper bound	$z_{u_g} - p_g - M_{zug}(1 - u_{g,d}) <= 0$
zugub2	zug upper bound	$z_{u_g} - M_{zug}u_{g,d} <= 0$
don	minimum online duration	$T_{on}v_{g,d} - u_{g,d} <= 0$
doff	minimum offline duration	$T_{off}w_{g,d} - (1 - u_{g,d}) <= 0$
pdumax	unserved demand upper bound	$p_{d,u} - p_{d,s}^{+}c_{trl,d}1_{tl} <= 0$
cdgb	Select DG power from pg	$C_{DG}p_g - p_{g,DG} = 0$
SOClb	SOC lower bound	$-SOC + SOC_{min} <= 0$
SOCub	SOC upper bound	$SOC - SOC_{max} <= 0$
cdb	Charging decision bound	$u_{c,ESD} + u_{d,ESD} - 1 = 0$
cesb	Select ESD1 power from pg	$C_{ESD}p_g + z_{c,ESD} - z_{d,ESD} = 0$
zce1	zce bound 1	$-z_{c,ESD} + p_{c,ESD} <= 0$
zce2	zce bound 2	$z_{c,ESD} - p_{c,ESD} - M_{big}(1 - u_{c,ESD}) <= 0$
zce3	zce bound 3	$z_{c,ESD} - M_{big}u_{c,ESD} <= 0$
zde1	zde bound 1	$-z_{d,ESD} + p_{d,ESD} <= 0$
zde2	zde bound 2	$z_{d,ESD} - p_{d,ESD} - M_{big}(1 - u_{d,ESD}) <= 0$
zde3	zde bound 3	$z_{d,ESD} - M_{big}u_{d,ESD} <= 0$
SOCb	ESD1 SOC balance	$E_{n,R}SOCM_{r,ES} - T_{cfg}\eta_{c,R}z_{c,ESD}[:, 1:] + T_{cfg}R_{\eta_d,R}z_{d,ESD}[:, 1:] = 0$
SOCb0	ESD1 SOC initial balance	$E_nSOC[:, 0] - SOC_{init} - T_{cfg}\eta_c z_{c,ESD}[:, 0] + T_{cfg}\frac{1}{\eta_d}z_{d,ESD}[:, 0] = 0$
SOCr	SOC requirement	$SOC[:, -1] - SOC_{init} = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	2D Gen power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	2D Bus angle	*rad*	Bus.a	
plf	p_{lf}	2D Line flow	*p.u.*		
prs	$p_{r,s}$	2D Spinning reserve	*p.u.*		nonneg
prns	$p_{r,ns}$	2D Non-spinning reserve			nonneg
ugd	$u_{g,d}$	commitment decision		StaticGen.u	boolean
vgd	$v_{g,d}$	startup action		StaticGen.u	boolean
wgd	$w_{g,d}$	shutdown action		StaticGen.u	boolean
zug	z_{ug}	Aux var, $z_{ug} = u_{g,d} * p_g$			pos
pdu	$p_{d,u}$	unserved demand	*p.u.*		nonneg
pgdg	$p_{g,DG}$	DG output power	*p.u.*		
SOC	SOC	ESD1 State of Charge	*%*		pos
pce	$p_{c,ESD}$	ESD1 charging power	*p.u.*		nonneg
pde	$p_{d,ESD}$	ESD1 discharging power	*p.u.*		nonneg
uce	$u_{c,ESD}$	ESD1 charging decision			boolean
ude	$u_{d,ESD}$	ESD1 discharging decision			boolean
zce	$z_{c,ESD}$	Aux var for charging, $z_{c,ESD} = u_{c,ESD} * p_{c,ESD}$			nonneg
zde	$z_{d,ESD}$	Aux var for discharging, $z_{d,ESD} = u_{d,ESD} * p_{d,ESD}$			nonneg

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Reshaped controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Reshaped non-controllability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp
gs	S_g	Sum Gen vars vector in shape of zone	ZonalSum
ds	S_d	Sum pd vector in shape of zone	ZonalSum
pdz	$p_{d,z}$	zonal total load	NumOpDual
tlv	1_{tl}	time length vector	NumOp
pds	$p_{d,s}$	Scaled load	LoadScale
Mr	M_r	Subtraction matrix for ramping	RampSub
RR30	$R_{30,R}$	Repeated ramp rate	NumHstack
dsrpz	$d_{s,r,p,z}$	zonal spinning reserve requirement in percentage	NumOpDual
dsr	$d_{s,r,z}$	zonal spinning reserve requirement	NumOpDual
dnsrpz	$d_{nsr,p,z}$	zonal non-spinning reserve requirement in percentage	NumOpDual
dnsr	d_{nsr}	zonal non-spinning reserve requirement	NumOpDual
Mzug	M_{zug}	10 times of max of pmax as big M for zug	NumOp
Con	T_{on}	minimum ON coefficient	MinDur
Coff	T_{off}	minimum OFF coefficient	MinDur
pdsp	$p^+_{d,s}$	positive demand	NumOp
cd	C_{DG}	Select DG power from pg	VarSelect
REtaD	$\frac{1}{\eta_d}$		NumOp
Mb	M_{big}	10 times of max of pmax as big M	NumOp
ce	C_{ESD}	Select zue from pg	VarSelect
Mre	$M_{r,ES}$	Subtraction matrix for SOC	RampSub
EnR	$E_{n,R}$	Repeated En as 2D matrix, (ng, ng-1)	NumHstack
EtaCR	$\eta_{c,R}$	Repeated Etac as 2D matrix, (ng, ng-1)	NumHstack
REtaDR	$R_{\eta_d,R}$	Repeated REtaD as 2D matrix, (ng, ng-1)	NumHstack

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C^T_{ft}	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh

continues on next page

Table 10 – continued from previous page

Name	Symbol	Description	Unit	Source
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P^{inj}_{bus}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P^{inj}_f	Line power injection vector		MatProcessor.Pfinj
zg	$z_{one,g}$	Gen zone		StaticGen.zone
zd	$z_{one,d}$	Load zone		StaticLoad.zone
R10	R_{10}	10-min ramp rate	*p.u./h*	StaticGen.R10
sd	s_d	zonal load factor for UC		UCTSlot.sd
timeslot	$t_{s,idx}$	Time slot for multi-period UC		UCTSlot.idx
R30	R_{30}	30-min ramp rate	*p.u./h*	StaticGen.R30
dsr	d_{sr}	spinning reserve requirement in percentage	*%*	SR.demand
csr	c_{sr}	cost for spinning reserve	*$/(p.u.*h)*	SRCost.csr
cnsr	c_{nsr}	cost for non-spinning reserve	*$/(p.u.*h)*	NSRCost.cnsr
dnsr	d_{nsr}	non-spinning reserve requirement in percentage	*%*	NSR.demand
csu	c_{su}	startup cost	*$*	GCost.csu
csd	c_{sd}	shutdown cost	*$*	GCost.csd
cdp	$c_{d,p}$	penalty for unserved load	*$/(p.u.*h)*	DCost.cdp
dctrl	$c_{trl,d}$	load controllability		StaticLoad.ctrl
td1	t_{d1}	minimum ON duration	*h*	StaticGen.td1
td2	t_{d2}	minimum OFF duration	*h*	StaticGen.td2
gendg	g_{DG}	gen of DG		DG.gen
gammapd	$\gamma_{p,DG}$	('Ratio of DG.pge w.r.t to that of static generator',)		DG.gammap
En	E_n	Rated energy capacity	*MWh*	ESD1.En
SOCmax	SOC_{max}	Maximum allowed value for SOC in limiter	*%*	ESD1.SOCmax
SOCmin	SOC_{min}	Minimum required value for SOC in limiter	*%*	ESD1.SOCmin
SOCinit	SOC_{init}	Initial SOC	*%*	ESD1.SOCinit
EtaC	η_c	Efficiency during charging	*%*	ESD1.EtaC
EtaD	η_d	Efficiency during discharging	*%*	ESD1.EtaD
genesd	g_{ESD}	gen of ESD1		ESD1.gen
gammapesd	$\gamma_{p,ESD}$	Ratio of ESD1.pge w.r.t to that of static generator		ESD1.gammap

5.4 DED

Type for Distributional economic dispatch.

Available routines: *DOPF*, *DOPFVIS*

5.4.1 DOPF

Linearzied distribution OPF, where power loss are ignored.

UNDER DEVELOPMENT!

Reference:

[1] L. Bai, J. Wang, C. Wang, C. Chen, and F. Li, “Distribution Locational Marginal Pricing (DLMP) for Congestion Management and Voltage Support,” IEEE Trans. Power Syst., vol. 33, no. 4, pp. 4061–4073, Jul. 2018, doi: 10.1109/TPWRS.2017.2767632.

Objective

Name	Description	Unit	Expression
obj	total cost	$\$$	$min. \sum(c_2 power(p_g, 2)) + \sum(c_1 p_g) + \sum(u_g c_0)$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e} p_{g,0} + c_{trl,e} p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e} p_{g,0} - c_{trl,e} p_{g,max} <= 0$
pb	power balance	$B_{bus}\theta_{bus} + P^{inj}_{bus} + C_l p_d + C_{sh} g_{sh} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft}\theta_{bus} - \theta_{max} <= 0$
alfub	line angle difference upper bound	$C^T_{ft}\theta_{bus} - \theta_{max} <= 0$
qglb	qg min	$-q_g + u_g q_{min} <= 0$
qgub	qg max	$q_g - u_g q_{max} <= 0$
vu	Voltage upper limit	$v^2 - v^2_{max} <= 0$
vl	Voltage lower limit	$-v^2 + v^2_{min} <= 0$
lvd	line voltage drop	$C^T_{ft} v^2 - (r p_{lf} + x q_{lf}) = 0$
qb	reactive power balance	$\sum(q_d) - \sum(q_g) = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	Gen active power	*p.u.*	StaticGen.p	
aBus	θ_{bus}	Bus voltage angle	*rad*	Bus.a	
plf	p_{lf}	Line flow	*p.u.*		
qg	q_g	Gen reactive power	*p.u.*	StaticGen.q	
v	v	Bus voltage	*p.u.*	Bus.v	
vsq	v^2	square of Bus voltage	*p.u.*		
qlf	q_{lf}	line reactive power	*p.u.*		

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
qmax	q_{max}	generator maximum reactive power	*p.u.*	StaticGen.qmax
qmin	q_{min}	generator minimum reactive power	*p.u.*	StaticGen.qmin
qd	q_d	reactive demand	*p.u.*	StaticLoad.q0
vmax	v_{max}	Bus voltage upper limit	*p.u.*	Bus.vmax
vmin	v_{min}	Bus voltage lower limit	*p.u.*	Bus.vmin
r	r	line resistance	*p.u.*	Line.r
x	x	line reactance	*p.u.*	Line.x

5.4.2 DOPFVIS

Linearzied distribution OPF with variables for virtual inertia and damping from from REGCV1, where power loss are ignored.

UNDER DEVELOPMENT!

Reference:

[1] L. Bai, J. Wang, C. Wang, C. Chen, and F. Li, "Distribution Locational Marginal Pricing (DLMP) for Congestion Management and Voltage Support," IEEE Trans. Power Syst., vol. 33, no. 4, pp. 4061–4073, Jul. 2018, doi: 10.1109/TPWRS.2017.2767632.

Objective

Name	Description	Unit	Expression
tc	total cost	$\$$	$min. \sum(c_2 p_g^2 + c_1 p_g + u_g c_0 + c_m M + c_d D)$

Constraints

Name	Description	Expression
pglb	pg min	$-p_g + c_{trl,n,e} p_{g,0} + c_{trl,e} p_{g,min} <= 0$
pgub	pg max	$p_g - c_{trl,n,e} p_{g,0} - c_{trl,e} p_{g,max} <= 0$
pb	power balance	$B_{bus} \theta_{bus} + P^{inj}_{bus} + C_l p_d + C_{sh} g_{sh} - C_g p_g = 0$
plflb	line flow lower bound	$-B_f \theta_{bus} - P^{inj}_f - R_{ATEA} <= 0$
plfub	line flow upper bound	$B_f \theta_{bus} + P^{inj}_f - R_{ATEA} <= 0$
alflb	line angle difference lower bound	$-C^T_{ft} \theta_{bus} - \theta_{max} <= 0$
alfub	line angle difference upper bound	$C^T_{ft} \theta_{bus} - \theta_{max} <= 0$
qglb	qg min	$-q_g + u_g q_{min} <= 0$
qgub	qg max	$q_g - u_g q_{max} <= 0$
vu	Voltage upper limit	$v^2 - v^2_{max} <= 0$
vl	Voltage lower limit	$-v^2 + v^2_{min} <= 0$
lvd	line voltage drop	$C^T_{ft} v^2 - (r p_{lf} + x q_{lf}) = 0$
qb	reactive power balance	$\sum(q_d) - \sum(q_g) = 0$

Vars

Name	Symbol	Description	Unit	Source	Properties
pg	p_g	Gen active power	$p.u.$	StaticGen.p	
aBus	θ_{bus}	Bus voltage angle	rad	Bus.a	
plf	p_{lf}	Line flow	$p.u.$		
qg	q_g	Gen reactive power	$p.u.$	StaticGen.q	
v	v	Bus voltage	$p.u.$	Bus.v	
vsq	v^2	square of Bus voltage	$p.u.$		
qlf	q_{lf}	line reactive power	$p.u.$		
M	M	Emulated startup time constant (M=2H) from REGCV1	*s*		
D	D	Emulated damping coefficient from REGCV1	$p.u.$		

Services

Name	Symbol	Description	Type
ctrle	$c_{trl,e}$	Effective Gen controllability	NumOpDual
nctrl	$c_{trl,n}$	Effective Gen uncontrollability	NumOp
nctrle	$c_{trl,n,e}$	Effective Gen uncontrollability	NumOpDual
amax	θ_{max}	max line angle difference	NumOp

Parameters

Name	Symbol	Description	Unit	Source
c2	c_2	Gen cost coefficient 2	*$/(p.u.^2)*	GCost.c2
c1	c_1	Gen cost coefficient 1	*$/(p.u.)*	GCost.c1
c0	c_0	Gen cost coefficient 0	*$*	GCost.c0
ug	u_g	Gen connection status		StaticGen.u
ctrl	c_{trl}	Gen controllability		StaticGen.ctrl
pmax	$p_{g,max}$	Gen maximum active power	*p.u.*	StaticGen.pmax
pmin	$p_{g,min}$	Gen minimum active power	*p.u.*	StaticGen.pmin
p0	$p_{g,0}$	Gen initial active power	*p.u.*	StaticGen.pg0
pd	p_d	active demand	*p.u.*	StaticLoad.p0
rate_a	R_{ATEA}	long-term flow limit	*MVA*	Line.rate_a
gsh	g_{sh}	shunt conductance		Shunt.g
Cg	C_g	Gen connection matrix		MatProcessor.Cg
Cl	C_l	Load connection matrix		MatProcessor.Cl
CftT	C_{ft}^T	Transpose of line connection matrix		MatProcessor.CftT
Csh	C_{sh}	Shunt connection matrix		MatProcessor.Csh
Bbus	B_{bus}	Bus admittance matrix		MatProcessor.Bbus
Bf	B_f	Bf matrix		MatProcessor.Bf
Pbusinj	P_{bus}^{inj}	Bus power injection vector		MatProcessor.Pbusinj
Pfinj	P_f^{inj}	Line power injection vector		MatProcessor.Pfinj
qmax	q_{max}	generator maximum reactive power	*p.u.*	StaticGen.qmax
qmin	q_{min}	generator minimum reactive power	*p.u.*	StaticGen.qmin
qd	q_d	reactive demand	*p.u.*	StaticLoad.q0
vmax	v_{max}	Bus voltage upper limit	*p.u.*	Bus.vmax
vmin	v_{min}	Bus voltage lower limit	*p.u.*	Bus.vmin
r	r	line resistance	*p.u.*	Line.r
x	x	line reactance	*p.u.*	Line.x
cm	c_m	Virtual inertia cost	*$/s*	VSGCost.cm
cd	c_d	Virtual damping cost	*$/(p.u.)*	VSGCost.cd

5.5 PF

Type for power flow routines.

Common Parameters: pd

Common Vars: pg

Available routines: *DCPF*, *PFlow*, *CPF*

5.5.1 DCPF

DC power flow.

Notes

1. DCPF is solved with PYPOWER `runpf` function.
2. DCPF formulation is not complete yet, but this does not affect the results because the data are passed to PYPOWER for solving.

Vars

Name	Symbol	Description	Unit	Source	Properties
aBus	a_{Bus}	bus voltage angle	*rad*	Bus.a	
pg	p_g	actual active power generation	*p.u.*	StaticGen.p	

Parameters

Name	Symbol	Description	Unit	Source
x	x	line reactance	*p.u.*	Line.x
tap	t_{ap}	transformer branch tap ratio	*float*	Line.tap
phi	ϕ	transformer branch phase shift in rad	*radian*	Line.phi
pd	p_d	active deman	*p.u.*	StaticLoad.p0

5.5.2 PFlow

AC Power Flow routine.

Notes

1. AC pwoer flow is solved with PYPOWER `runpf` function.
2. AC power flow formulation in AMS style is NOT DONE YET, but this does not affect the results because the data are passed to PYPOWER for solving.

Vars

Name	Symbol	Description	Unit	Source	Properties
aBus	a_{Bus}	bus voltage angle	*rad*	Bus.a	
vBus	v_{Bus}	bus voltage magnitude	*p.u.*	Bus.v	
pg	p_g	active power generation	*p.u.*	StaticGen.p	
qg	q_g	reactive power generation	*p.u.*	StaticGen.q	

Parameters

Name	Symbol	Description	Unit	Source
x	x	line reactance	*p.u.*	Line.x
tap	t_{ap}	transformer branch tap ratio	*float*	Line.tap
phi	ϕ	transformer branch phase shift in rad	*radian*	Line.phi
pd	p_d	active deman	*p.u.*	StaticLoad.p0
qd	q_d	reactive power load in system base	*p.u.*	StaticLoad.q0

5.5.3 CPF

Continuous power flow.

Still under development, not ready for use.

Vars

Name	Symbol	Description	Unit	Source	Properties
aBus	a_{Bus}	bus voltage angle	*rad*	Bus.a	
vBus	v_{Bus}	bus voltage magnitude	*p.u.*	Bus.v	
pg	p_g	active power generation	*p.u.*	StaticGen.p	
qg	q_g	reactive power generation	*p.u.*	StaticGen.q	

Parameters

Name	Symbol	Description	Unit	Source
x	x	line reactance	*p.u.*	Line.x
tap	t_{ap}	transformer branch tap ratio	*float*	Line.tap
phi	ϕ	transformer branch phase shift in rad	*radian*	Line.phi
pd	p_d	active deman	*p.u.*	StaticLoad.p0
qd	q_d	reactive power load in system base	*p.u.*	StaticLoad.q0

5.6 UndefinedType

The undefined type.

CHAPTER

SIX

MODEL REFERENCE

Use the left navigation pane to locate the group and model and view details.

Supported Groups and Models

Group	Models
ACLine	*Line*
ACTopology	*Bus*
Collection	*Area*, *Region*
Cost	*GCost*, *SFRCost*, *VSGCost*, *DCost*
DG	*PVD1*, *ESD1*
Horizon	*TimeSlot*, *EDTSlot*, *UCTSlot*
Information	*Summary*
RenGen	*REGCA1*
Reserve	*SFR*, *SR*, *NSR*, *VSGR*
StaticGen	*PV*, *Slack*
StaticLoad	*PQ*
StaticShunt	*Shunt*
Undefined	*SRCost*, *NSRCost*
VSG	*REGCV1*, *REGCV2*

6.1 ACLine

Common Parameters: u, name, idx, bus1, bus2, r, x

Available models: *Line*

6.1.1 Line

AC transmission line model.

The model is also used for two-winding transformer. Transformers can set the tap ratio in `tap` and/or phase shift angle `phi`.

Notes

There is a known issue that adding Algeb ud will cause Line.algebs run into AttributeError: 'NoneType' object has no attribute 'n'. Not figured out why yet.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
bus1		idx of from bus			
bus2		idx of to bus			
Sn	S_n	Power rating	100	*MW*	non_zero
fn	f	rated frequency	60	*Hz*	
Vn1	V_{n1}	AC voltage rating	110	*kV*	non_zero
Vn2	V_{n2}	rated voltage of bus2	110	*kV*	non_zero
r	r	line resistance	0.000	*p.u.*	z
x	x	line reactance	0.000	*p.u.*	non_zero,z
b		shared shunt susceptance	0	*p.u.*	y
g		shared shunt conductance	0	*p.u.*	y
b1	b_1	from-side susceptance	0	*p.u.*	y
g1	g_1	from-side conductance	0	*p.u.*	y
b2	b_2	to-side susceptance	0	*p.u.*	y
g2	g_2	to-side conductance	0	*p.u.*	y
trans		transformer branch flag	0	*bool*	
tap	t_{ap}	transformer branch tap ratio	1	*float*	non_negative
phi	ϕ	transformer branch phase shift in rad	0	*radian*	
rate_a	R_{ATEA}	long-term flow limit (placeholder)	999	*MVA*	
rate_b	R_{ATEB}	short-term flow limit (placeholder)	999	*MVA*	
rate_c	R_{ATEC}	emergency flow limit (placeholder)	999	*MVA*	
owner		owner code			
xcoord		x coordinates			
ycoord		y coordinates			
amin	a_{min}	minimum angle difference, from bus - to bus	-6.283	*rad*	
amax	a_{max}	maximum angle difference, from bus - to bus	6.283	*rad*	

6.2 ACTopology

Common Parameters: u, name, idx

Common Variables: a, v

Available models: *Bus*

6.2.1 Bus

AC Bus model.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
Vn	V_n	AC voltage rating	110	*kV*	non_zero
vmax	V_{max}	Voltage upper limit	1.100	*p.u.*	
vmin	V_{min}	Voltage lower limit	0.900	*p.u.*	
v0	V_0	initial voltage magnitude	1	*p.u.*	non_zero
a0	θ_0	initial voltage phase angle	0	*rad*	
xcoord		x coordinate (longitude)	0		
ycoord		y coordinate (latitude)	0		
area		Area code			
zone		Zone code			
owner		Owner code			

Variables

Name	Symbol	Type	Description	Unit
a	θ	Algeb	voltage angle	*rad*
v	V	Algeb	voltage magnitude	*p.u.*

6.3 Collection

Collection of topology models

Common Parameters: u, name, idx

Available models: *Area*, *Region*

6.3.1 Area

Area model.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			

Services

Name	Description	Symbol	Type
Bus		Bus	BackRef
ACTopology		$ACTopology$	BackRef

6.3.2 Region

Region model for zonal vars.

Notes

1. Region is a collection of buses.
2. Model `Region` is not actually defined in ANDES.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			

Services

Name	Description	Symbol	Type
Bus		Bus	BackRef
ACTopology		$ACTopology$	BackRef

6.4 Cost

Common Parameters: u, name, idx, gen

Available models: *GCost*, *SFRCost*, *VSGCost*, *DCost*

6.4.1 GCost

Generator cost model, similar to MATPOWER `gencost` format.

`type` is the cost model type. 1 for piecewise linear, 2 for polynomial.

In piecewise linear cost model, cost function f(p) is defined by a set of points: (p0, c0), (p1, c1), (p2, c2), where p0 < p1 < p2.

In quadratic cost model, cost function f(p) is defined by a set of coefficients: f(p) = c2 * p^2 + c1 * p + c0.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
gen		static generator index			mandatory
type	t_{ype}	Cost model type. 1 for piecewise linear, 2 for polynomial	2		
csu	c_{su}	startup cost in US dollars	0	*$*	
csd	c_{sd}	shutdown cost in US dollars	0	*$*	
c2	c_2	coefficient 2	0	*$/(p.u.*h)^2*	
c1	c_1	coefficient 1	0	*$/p.u.*h*	
c0	c_0	coefficient 0	0	*$*	

6.4.2 SFRCost

Linear SFR cost model.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
gen		static generator index			mandatory
cru	c_r	cost for RegUp reserve	0	*$/(p.u.*h)*	
crd	c_r	cost for RegDn reserve	0	*$/(p.u.*h)*	

6.4.3 VSGCost

Linear cost model for VSG emulated inertia (M) and damping (D).

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
reg		Renewable generator idx			mandatory
cm	c_r	cost for emulated inertia (M)	0	*$/s*	
cd	c_r	cost for emulated damping (D)	0	*$/p.u.*	

6.4.4 DCost

Linear cost model for dispatchable loads.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
pq		static load index			mandatory
cdp	$c_{d,p}$	cost for unserve load penalty	999	*$/(p.u.*h)*	

6.5 DG

Distributed generation (small-scale).

See ANDES Documentation SynGen here for the notes on replacing StaticGen and setting the power ratio parameters.

Reference:

[1] ANDES Documentation, SynGen, [Online]

Available:

https://docs.andes.app/en/latest/groupdoc/SynGen.html#syngen

Common Parameters: u, name, idx, bus, fn

Available models: *PVD1*, *ESD1*

6.5.1 PVD1

Distributed PV model, revised from ANDES `PVD1` model for dispatch.

Following parameters are omitted from the original dynamic model: `fn`, `busf`, `xc`, `pqflag`, `igreg`, `v0`, `v1`, `dqdv`, `fdbd`, `ddn`, `ialim`, `vt0`, `vt1`, `vt2`, `vt3`, `vrflag`, `ft0`, `ft1`, `ft2`, `ft3`, `frflag`, `tip`, `tiq`, `recflag`.

Reference:

[1] ANDES Documentation, PVD1

Available:

https://docs.andes.app/en/latest/groupdoc/DG.html#pvd1

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
bus		interface bus id (place holder)			mandatory
gen		static generator index			mandatory
Sn	S_n	device MVA rating	100	*MVA*	
gammap	γ_p	Ratio of ESD1.pref0 w.r.t to that of static PV	1		
gammaq	γ_q	Ratio of ESD1.qref0 w.r.t to that of static PV	1		

6.5.2 ESD1

Distributed energy storage model, revised from ANDES `ESD1` model for dispatch.

Following parameters are omitted from the original dynamic model: `fn`, `busf`, `xc`, `pqflag`, `igreg`, `v0`, `v1`, `dqdv`, `fdbd`, `ddn`, `ialim`, `vt0`, `vt1`, `vt2`, `vt3`, `vrflag`, `ft0`, `ft1`, `ft2`, `ft3`, `frflag`, `tip`, `tiq`, `recflag`.

Reference:

[1] ANDES Documentation, ESD1

Available:

https://docs.andes.app/en/latest/groupdoc/DG.html#esd1

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
bus		interface bus id (place holder)			mandatory
gen		static generator index			mandatory
Sn	S_n	device MVA rating	100	*MVA*	
gammap	γ_p	Ratio of ESD1.pref0 w.r.t to that of static PV	1		
gammaq	γ_q	Ratio of ESD1.qref0 w.r.t to that of static PV	1		
SOCmin	SOC_{min}	Minimum required value for SOC in limiter	0		
SOCmax	SOC_{max}	Maximum allowed value for SOC in limiter	1		
SOCinit	SOC_{init}	Initial state of charge	0.500		
En	E_n	Rated energy capacity	100	*MWh*	
EtaC	Eta_C	Efficiency during charging	1		
EtaD	Eta_D	Efficiency during discharging	1		

6.6 Horizon

Time horizon group.

Common Parameters: u, name, idx

Available models: *TimeSlot*, *EDTSlot*, *UCTSlot*

6.6.1 TimeSlot

Time slot data for rolling horizon.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
sd	s_d	zonal load scaling factor			

6.6.2 EDTSlot

Time slot model for ED.

sd is the zonal load scaling factor. Cells in *sd* should have *nz* values seperated by comma, where *nz* is the number of *Region* in the system.

ug is the unit commitment decisions. Cells in *ug* should have *ng* values seperated by comma, where *ng* is the number of *StaticGen* in the system.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
sd	s_d	zonal load scaling factor			
ug	u_g	unit commitment decisions			

6.6.3 UCTSlot

Time slot model for UC.

sd is the zonal load scaling factor. Cells in *sd* should have *nz* values seperated by comma, where *nz* is the number of *Region* in the system.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
sd	s_d	zonal load scaling factor			

6.7 Information

Group for information container models.

Available models: *Summary*

6.7.1 Summary

Class for storing system summary. Can be used for random information or notes.

Parameters

Name	Symbol	Description	Default	Unit	Properties
field		field name			
comment		information, comment, or anything			
comment2		comment field 2			
comment3		comment field 3			
comment4		comment field 4			

6.8 RenGen

Renewable generator (converter) group.

See ANDES Documentation SynGen here for the notes on replacing StaticGen and setting the power ratio parameters.

Reference:

[1] ANDES Documentation, RenGen, [Online]

Available:

https://docs.andes.app/en/latest/groupdoc/RenGen.html#rengen

Common Parameters: u, name, idx, bus, gen, Sn

Common Variables: Pe, Qe

Available models: *REGCA1*

6.8.1 REGCA1

Renewable generator dispatch model.

Reference:

[1] ANDES Documentation, REGCA1

Available: https://docs.andes.app/en/latest/groupdoc/RenGen.html#regca1

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
bus		interface bus idx			mandatory
gen		static generator index			mandatory
Sn	S_n	device MVA rating	100	*MVA*	
gammap	γ_P	P ratio of linked static gen	1		
gammaq	γ_Q	Q ratio of linked static gen	1		

6.9 Reserve

Common Parameters: u, name, idx, zone

Available models: *SFR*, *SR*, *NSR*, *VSGR*

6.9.1 SFR

Zonal secondary frequency reserve (SFR) model.

Notes

- `Zone` model is required for this model, and zone is defined by Param `Bus.zone`.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
zone		Zone code			
du	d_u	Zonal RegUp reserve demand	0	%	
dd	d_d	Zonal RegDown reserve demand	0	%	

6.9.2 SR

Zonal spinning reserve (SR) model.

Notes

- `Zone` model is required for this model, and zone is defined by Param `Bus.zone`.
- `demand` is multiplied to online unused generation capacity.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
zone		Zone code			
demand	d_{SR}	Zonal spinning reserve demand	0.100	%	

6.9.3 NSR

Zonal non-spinning reserve (NSR) model.

Notes

- `Zone` model is required for this model, and zone is defined by Param `Bus.zone`.
- `demand` is multiplied to offline generation capacity.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
zone		Zone code			
demand	d_{NSR}	Zonal non-spinning reserve demand	0.100	%	

6.9.4 VSGR

Zonal VSG provided reserve model.

Notes

- Zone model is required for this model, and zone is defined by Param `Bus.zone`.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
zone		Zone code			
dvm	d_{VM}	Zonal virtual inertia demand	0	*s*	
dvd	d_{VD}	Zonal virtual damping demand	0	*p.u.*	

6.10 StaticGen

Generator group.

6.10.1 Notes

For co-simulation with ANDES, check ANDES StaticGen for replacing static generators with dynamic generators.

Common Parameters: u, name, idx, Sn, Vn, p0, q0, ra, xs, subidx

Common Variables: p, q

Available models: *PV*, *Slack*

PV

PV generator model.

TODO: implement type conversion in config

6.10.2 Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
Sn	S_n	Power rating	100		non_zero
Vn	V_n	AC voltage rating	110		non_zero
subidx		index for generators on the same bus			
bus		idx of the installed bus			mandatory
busr		bus idx for remote voltage control			
p0	p_0	active power set point in system base	0	*p.u.*	
q0	q_0	reactive power set point in system base	0	*p.u.*	

continues on next page

Table 1 – continued from previous page

Name	Symbol	Description	Default	Unit	Properties
pmax	p_{max}	maximum active power in system base	999	*p.u.*	
pmin	p_{min}	minimum active power in system base	-1	*p.u.*	
qmax	q_{max}	maximim reactive power in system base	999	*p.u.*	
qmin	q_{min}	minimum reactive power in system base	-999	*p.u.*	
v0	v_0	voltage set point	1		
vmax	v_{max}	maximum voltage voltage	1.400		
vmin	v_{min}	minimum allowed voltage	0.600		
ra	r_a	armature resistance	0		
xs	x_s	armature reactance	0.300		
ctrl	c_{trl}	generator controllability	1		
Pc1	P_{c1}	lower real power output of PQ capability curve	0	*p.u.*	
Pc2	P_{c2}	upper real power output of PQ capability curve	0	*p.u.*	
Qc1min	Q_{c1min}	minimum reactive power output at Pc1	0	*p.u.*	
Qc1max	Q_{c1max}	maximum reactive power output at Pc1	0	*p.u.*	
Qc2min	Q_{c2min}	minimum reactive power output at Pc2	0	*p.u.*	
Qc2max	Q_{c2max}	maximum reactive power output at Pc2	0	*p.u.*	
Ragc	R_{agc}	ramp rate for load following/AGC	999	*p.u./h*	
R10	R_{10}	ramp rate for 10 minute reserves	999	*p.u./h*	
R30	R_{30}	30 minute ramp rate	999	*p.u./h*	
Rq	R_q	ramp rate for reactive power (2 sec timescale)	999	*p.u./h*	
apf	a_{pf}	area participation factor	0		
pg0	p_{g0}	active power start point (system base)	0	*p.u.*	
td1	t_{d1}	minimum ON duration	0	*h*	
td2	t_{d2}	minimum OFF duration	0	*h*	
zone		Retrieved zone idx			

6.10.3 Variables

Name	Symbol	Type	Description	Unit
ud	u_d	Algeb	connection status decision	*bool*
p	p	Algeb	active power generation	*p.u.*
q	q	Algeb	reactive power generation	*p.u.*

Slack

Slack generator model.

6.10.4 Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
Sn	S_n	Power rating	100		non_zero
Vn	V_n	AC voltage rating	110		non_zero
subidx		index for generators on the same bus			
bus		idx of the installed bus			mandatory
busr		bus idx for remote voltage control			
p0	p_0	active power set point in system base	0	*p.u.*	
q0	q_0	reactive power set point in system base	0	*p.u.*	
pmax	p_{max}	maximum active power in system base	999	*p.u.*	
pmin	p_{min}	minimum active power in system base	-1	*p.u.*	
qmax	q_{max}	maximim reactive power in system base	999	*p.u.*	
qmin	q_{min}	minimum reactive power in system base	-999	*p.u.*	
v0	v_0	voltage set point	1		
vmax	v_{max}	maximum voltage voltage	1.400		
vmin	v_{min}	minimum allowed voltage	0.600		
ra	r_a	armature resistance	0		
xs	x_s	armature reactance	0.300		
a0	θ_0	reference angle set point	0		
ctrl	c_{trl}	generator controllability	1		
Pc1	P_{c1}	lower real power output of PQ capability curve	0	*p.u.*	
Pc2	P_{c2}	upper real power output of PQ capability curve	0	*p.u.*	
Qc1min	Q_{c1min}	minimum reactive power output at Pc1	0	*p.u.*	
Qc1max	Q_{c1max}	maximum reactive power output at Pc1	0	*p.u.*	
Qc2min	Q_{c2min}	minimum reactive power output at Pc2	0	*p.u.*	
Qc2max	Q_{c2max}	maximum reactive power output at Pc2	0	*p.u.*	
Ragc	R_{agc}	ramp rate for load following/AGC	999	*p.u./h*	
R10	R_{10}	ramp rate for 10 minute reserves	999	*p.u./h*	
R30	R_{30}	30 minute ramp rate	999	*p.u./h*	
Rq	R_q	ramp rate for reactive power (2 sec timescale)	999	*p.u./h*	
apf	a_{pf}	area participation factor	0		
pg0	p_{g0}	active power start point (system base)	0	*p.u.*	
td1	t_{d1}	minimum ON duration	0	*h*	
td2	t_{d2}	minimum OFF duration	0	*h*	
zone		Retrieved zone idx			

6.10.5 Variables

Name	Symbol	Type	Description	Unit
ud	u_d	Algeb	connection status decision	*bool*
p	p	Algeb	active power generation	*p.u.*
q	q	Algeb	reactive power generation	*p.u.*

6.11 StaticLoad

Static load group.

Common Parameters: u, name, idx

Available models: *PQ*

6.11.1 PQ

PQ load model.

TODO: implement type conversion in config

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
bus		linked bus idx			mandatory
Vn	V_n	AC voltage rating	110	*kV*	non_zero
p0	p_0	active power load in system base	0	*p.u.*	
q0	q_0	reactive power load in system base	0	*p.u.*	
vmax	v_{max}	max voltage before switching to impedance	1.200		
vmin	v_{min}	min voltage before switching to impedance	0.800		
owner		owner idx			
zone		Retrieved zone idx			
ctrl	c_{trl}	load controllability	1		

6.12 StaticShunt

Static shunt compensator group.

Common Parameters: u, name, idx

Available models: *Shunt*

6.12.1 Shunt

Phasor-domain shunt compensator Model.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
bus		idx of connected bus			mandatory
Sn	S_n	Power rating	100		non_zero
Vn	V_n	AC voltage rating	110		non_zero
g	g	shunt conductance (real part)	0		y
b	b	shunt susceptance (positive as capacitive)	0		y
fn	f_n	rated frequency	60		

6.13 Undefined

The undefined group. Holds models with no `group`.

Common Parameters: u, name, idx

Available models: *SRCost*, *NSRCost*

6.13.1 SRCost

Linear spinning reserve cost model.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
gen		static generator index			mandatory
csr	c_{sr}	cost for spinning reserve	0	*$/(p.u.*h)*	

6.13.2 NSRCost

Linear non-spinning reserve cost model.

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
gen		static generator index			mandatory
cnsr	c_{nsr}	cost for non-spinning reserve	0	*$/(p.u.*h)*	

6.14 VSG

Renewable generator with virtual synchronous generator (VSG) control group.

Note that this is a group separate from `RenGen` for VSG dispatch study.

Common Parameters: u, name, idx, bus, gen, Sn

Common Variables: Pe, Qe

Available models: *REGCV1*, *REGCV2*

6.14.1 REGCV1

Voltage-controlled converter model (virtual synchronous generator) with inertia emulation.

Here Mmax and Dmax are assumed to be constant, but they might subject to the operating condition of the converter.

Notes

- The generation is defined by group *StaticGen*
- Generation cost is defined by model *GCost*
- Inertia emulation cost is defined by model *VSGCost*

Reference:

[1] ANDES Documentation, REGCV1

Available:

https://docs.andes.app/en/latest/groupdoc/RenGen.html#regcv1

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
bus		interface bus idx			mandatory
gen		static generator index			mandatory
Sn	S_n	device MVA rating	100	*MVA*	
gammap	γ_P	P ratio of linked static gen	1		
gammaq	γ_Q	Q ratio of linked static gen	1		
zone		Retrieved zone idx			
Mmax	M_{max}	Maximum inertia emulation	99	*s*	power
Dmax	D_{max}	Maximum damping emulation	99	*p.u.*	power

6.14.2 REGCV2

Voltage-controlled VSC.

Reference:

[1] ANDES Documentation, REGCV2

Available:

https://docs.andes.app/en/latest/groupdoc/RenGen.html#regcv2

Parameters

Name	Symbol	Description	Default	Unit	Properties
idx		unique device idx			
u	u	connection status	1	*bool*	
name		device name			
bus		interface bus idx			mandatory
gen		static generator index			mandatory
Sn	S_n	device MVA rating	100	*MVA*	
gammap	γ_P	P ratio of linked static gen	1		
gammaq	γ_Q	Q ratio of linked static gen	1		
zone		Retrieved zone idx			
Mmax	M_{max}	Maximum inertia emulation	99	*s*	power
Dmax	D_{max}	Maximum damping emulation	99	*p.u.*	power

CHAPTER SEVEN

API REFERENCE

7.1 System

ams.system	Module for system.

7.1.1 ams.system

Module for system.

Functions

disable_method(func)	
disable_methods(methods)	
example([setup, no_output])	Return an *ams.system.System* object for the `ieee14_uced.xlsx` as an example.

disable_method

`ams.system.`**`disable_method`**(*func*)

disable_methods

`ams.system.`**`disable_methods`**(*methods*)

example

ams.system.**example**(*setup=True*, *no_output=True*, ***kwargs*)

Return an *ams.system.System* object for the ieee14_uced.xlsx as an example.

This function is useful when a user wants to quickly get a System object for testing.

Returns

System
An example *ams.system.System* object.

Classes

System([case, name, config, config_path, ...])	A subclass of andes.system.System, this class encapsulates data, models, and routines for dispatch modeling and analysis in power systems.

ams.system.System

class ams.system.**System**(*case: str | None = None*, *name: str | None = None*, *config: Dict | None = None*, *config_path: str | None = None*, *default_config: bool | None = False*, *options: Dict | None = None*, ***kwargs*)

A subclass of andes.system.System, this class encapsulates data, models, and routines for dispatch modeling and analysis in power systems. Some methods inherited from the parent class are intentionally disabled.

Parameters

case
[str, optional] The path to the case file.

name
[str, optional] Name of the system instance.

config
[dict, optional] Configuration options for the system. Overrides the default configuration if provided.

config_path
[str, optional] The path to the configuration file.

default_config
[bool, optional] If True, the default configuration file is loaded.

options
[dict, optional] Additional configuration options for the system.

****kwargs**
Additional configuration options passed as keyword arguments.

Attributes

name
[str] Name of the system instance.

options
[dict] A dictionary containing configuration options for the system.

models
[OrderedDict] An ordered dictionary holding the model names and instances.

model_aliases
[OrderedDict] An ordered dictionary holding model aliases and their corresponding instances.

groups
[OrderedDict] An ordered dictionary holding group names and instances.

routines
[OrderedDict] An ordered dictionary holding routine names and instances.

types
[OrderedDict] An ordered dictionary holding type names and instances.

mats
[MatrixProcessor, None] A matrix processor instance, initially set to None.

mat
[OrderedDict] An ordered dictionary holding common matrices.

exit_code
[int] Command-line exit code. 0 indicates normal execution, while other values indicate errors.

recent
[RecentSolvedRoutines, None] An object storing recently solved routines, initially set to None.

dyn
[ANDES System, None] linked dynamic system, initially set to None. It is an instance of the ANDES system, which will be automatically set when using `System.to_andes()`.

files
[FileMan] File path manager instance.

is_setup
[bool] Internal flag indicating if the system has been set up.

Methods

setup:	Set up the system.
to_andes:	Convert the system to an ANDES system.

`__init__`(*case: str | None = None*, *name: str | None = None*, *config: Dict | None = None*, *config_path: str | None = None*, *default_config: bool | None = False*, *options: Dict | None = None*, ***kwargs*)

Methods

add(model[, param_dict])	Add a device instance for an existing model.
as_dict([vin, skip_empty])	Return system data as a dict where the keys are model names and values are dicts.
calc_pu_coeff()	Perform per unit value conversion.
call_models(method, models, *args, **kwargs)	Call methods on the given models.
collect_config(**kwargs)	Collect config data from models.
collect_ref()	Collect indices into *BackRef* for all models.
connectivity([info])	Perform connectivity check for system.
e_clear(**kwargs)	Clear equation arrays in DAE and model variables.
f_update(**kwargs)	Call the differential equation update method for models in sequence.
fg_to_dae(**kwargs)	Collect equation values into the DAE arrays.
find_devices()	Add dependent devices for all model based on *DeviceFinder*.
find_models(flag[, skip_zero])	Find models with at least one of the flags as True.
from_ipysheet(**kwargs)	Set an ipysheet object back to model.
g_islands(**kwargs)	Reset algebraic mismatches for islanded buses.
g_update(**kwargs)	Call the algebraic equation update method for models in sequence.
get_z(**kwargs)	Get all discrete status flags in a numpy array.
import_groups()	Import all groups classes defined in `models/group.py`.
import_models()	Import and instantiate models as System member attributes.
import_routines()	Import routines as defined in `routines/__init__.py`.
import_types()	Import all types classes defined in `routines/type.py`.
init(**kwargs)	Initialize the variables for each of the specified models.
j_islands(**kwargs)	Set gy diagonals to eps for *a* and *v* variables of islanded buses.
j_update(**kwargs)	Call the Jacobian update method for models in sequence.
l_update_eq(**kwargs)	Update equation-dependent limiter discrete components by calling `l_check_eq` of models.
l_update_var(**kwargs)	Update variable-based limiter discrete states by calling `l_update_var` of models.
link_ext_param([model])	Retrieve values for `ExtParam` for the given models.
precompile(**kwargs)	Trigger precompilation for the given models.
prepare(**kwargs)	Generate numerical functions from symbolically defined models.
reload(**kwargs)	Reload a new case in the same System object.
remove_pycapsule(**kwargs)	Remove PyCapsule objects in solvers.
report()	Write system routine reports to a plain-text file.
reset([force])	Reset to the state after reading data and setup.
s_update_post(**kwargs)	Update variable services by calling `s_update_post` of models.
s_update_var(**kwargs)	Update variable services by calling `s_update_var` of models.

continues on next page

Table 1 – continued from previous page

save_config(**kwargs)	Save all system, model, and routine configurations to an rc-formatted file.
set_address(models)	Set addresses for differential and algebraic variables.
set_config(**kwargs)	Set configuration for the System object.
set_dae_names(**kwargs)	Set variable names for differential and algebraic variables, right-hand side of external equations, and discrete flags.
set_output_subidx(**kwargs)	Process `andes.models.misc.Output` data and store the sub-indices into `dae.xy`.
set_var_arrays(**kwargs)	Set arrays (*v* and *e*) for internal variables to access dae arrays in place.
setup()	Set up system for studies.
store_adder_setter(**kwargs)	Store non-inplace adders and setters for variables and equations.
store_existing()	Store existing models in *System.existing*.
store_no_check_init(**kwargs)	Store differential variables with `check_init == False`.
store_sparse_pattern(**kwargs)	Collect and store the sparsity pattern of Jacobian matrices.
store_switch_times(**kwargs)	Store event switching time in a sorted Numpy array in `System.switch_times` and an OrderedDict `System.switch_dict`.
summary()	Print out system summary.
supported_models([export])	Return the support group names and model names in a table.
supported_routines([export])	Return the support type names and routine names in a table.
switch_action(**kwargs)	Invoke the actions associated with switch times.
to_andes([setup, addfile])	Convert the AMS system to an ANDES system.
to_ipysheet(**kwargs)	Return an ipysheet object for editing in Jupyter Notebook.
undill(**kwargs)	Reload generated function functions, from either the `$HOME/.andes/pycode` folder.
vars_to_dae(model)	Copy variables values from models to *System.dae*.
vars_to_models()	Copy variable values from *System.dae* to models.

System.add

`System.add`(*model*, *param_dict=None*, ***kwargs*)

Add a device instance for an existing model.

This methods calls the `add` method of *model* and registers the device *idx* to group.

System.as_dict

System.**as_dict**(*vin=False*, *skip_empty=True*)

Return system data as a dict where the keys are model names and values are dicts. Each dict has parameter names as keys and corresponding data in an array as values.

Returns

OrderedDict

System.calc_pu_coeff

System.**calc_pu_coeff**()

Perform per unit value conversion.

This function calculates the per unit conversion factors, stores input parameters to *vin*, and perform the conversion.

System.call_models

System.**call_models**(*method: str*, *models: OrderedDict*, **args*, ***kwargs*)

Call methods on the given models.

Parameters

method
[str] Name of the model method to be called

models
[OrderedDict, list, str] Models on which the method will be called

args
Positional arguments to be passed to the model method

kwargs
Keyword arguments to be passed to the model method

Returns

The return value of the models in an OrderedDict

System.collect_config

System.**collect_config**(***kwargs*)

Collect config data from models.

Returns

dict
a dict containing the config from devices; class names are keys and configs in a dict are values.

System.collect_ref

`System.collect_ref()`

Collect indices into *BackRef* for all models.

System.connectivity

`System.connectivity(info=True)`

Perform connectivity check for system.

Parameters

info
[bool] True to log connectivity summary.

System.e_clear

`System.e_clear(**kwargs)`

Clear equation arrays in DAE and model variables.

This step must be called before calling *f_update* or *g_update* to flush existing values.

System.f_update

`System.f_update(**kwargs)`

Call the differential equation update method for models in sequence.

Notes

Updated equation values remain in models and have not been collected into DAE at the end of this step.

System.fg_to_dae

`System.fg_to_dae(**kwargs)`

Collect equation values into the DAE arrays.

Additionally, the function resets the differential equations associated with variables pegged by anti-windup limiters.

System.find_devices

`System.find_devices()`

Add dependent devices for all model based on *DeviceFinder*.

System.find_models

System.**find_models**(*flag: str | Tuple | None*, *skip_zero: bool = True*)

Find models with at least one of the flags as True.

Parameters

flag
[list, str] Flags to find

skip_zero
[bool] Skip models with zero devices

Returns

OrderedDict
model name : model instance

> **Warning:** Checking the number of devices has been centralized into this function. `models` passed to most System calls must be retrieved from here.

System.from_ipysheet

System.**from_ipysheet**(***kwargs*)

Set an ipysheet object back to model.

System.g_islands

System.**g_islands**(***kwargs*)

Reset algebraic mismatches for islanded buses.

System.g_update

System.**g_update**(***kwargs*)

Call the algebraic equation update method for models in sequence.

Notes

Like *f_update*, updated values have not collected into DAE at the end of the step.

System.get_z

System.**get_z**(****kwargs*)

Get all discrete status flags in a numpy array. Values are written to `dae.z` in place.

Returns

numpy.array

System.import_groups

System.**import_groups**()

Import all groups classes defined in `models/group.py`.

Groups will be stored as instances with the name as class names. All groups will be stored to dictionary `System.groups`.

System.import_models

System.**import_models**()

Import and instantiate models as System member attributes.

Models defined in `models/__init__.py` will be instantiated *sequentially* as attributes with the same name as the class name. In addition, all models will be stored in dictionary `System.models` with model names as keys and the corresponding instances as values.

Examples

`system.Bus` stores the *Bus* object, and `system.PV` stores the PV generator object.

`system.models['Bus']` points the same instance as `system.Bus`.

System.import_routines

System.**import_routines**()

Import routines as defined in `routines/__init__.py`.

Routines will be stored as instances with the name as class names. All routines will be stored to dictionary `System.routines`.

Examples

`System.PFlow` is the power flow routine instance.

System.import_types

`System.import_types()`

Import all types classes defined in `routines/type.py`.

Types will be stored as instances with the name as class names. All types will be stored to dictionary `System.types`.

System.init

`System.init(**kwargs)`

Initialize the variables for each of the specified models.

For each model, the initialization procedure is:

- Get values for all *ExtService*.
- Call the model *init()* method, which initializes internal variables.
- Copy variables to DAE and then back to the model.

System.j_islands

`System.j_islands(**kwargs)`

Set gy diagonals to eps for *a* and *v* variables of islanded buses.

System.j_update

`System.j_update(**kwargs)`

Call the Jacobian update method for models in sequence.

The procedure is - Restore the sparsity pattern with `andes.variables.dae.DAE.restore_sparse()` - For each sparse matrix in (fx, fy, gx, gy), evaluate the Jacobian function calls and add values.

Notes

Updated Jacobians are immediately reflected in the DAE sparse matrices (fx, fy, gx, gy).

System.l_update_eq

`System.l_update_eq(**kwargs)`

Update equation-dependent limiter discrete components by calling `l_check_eq` of models. Force set equations after evaluating equations.

This function is must be called after differential equation updates.

System.l_update_var

`System.l_update_var(**kwargs)`

Update variable-based limiter discrete states by calling `l_update_var` of models.

This function is must be called before any equation evaluation.

System.link_ext_param

`System.link_ext_param(model=None)`

Retrieve values for `ExtParam` for the given models.

System.precompile

`System.precompile(**kwargs)`

Trigger precompilation for the given models.

Arguments are the same as `prepare`.

System.prepare

`System.prepare(**kwargs)`

Generate numerical functions from symbolically defined models.

All procedures in this function must be independent of test case.

Parameters

quick
[bool, optional] True to skip pretty-print generation to reduce code generation time.

incremental
[bool, optional] True to generate only for modified models, incrementally.

models
[list, OrderedDict, None] List or OrderedList of models to prepare

nomp
[bool] True to disable multiprocessing

> **Warning:** Generated lambda functions will be serialized to file, but pretty prints (SymPy objects) can only exist in the System instance on which prepare is called.

Notes

Option `incremental` compares the md5 checksum of all var and service strings, and only regenerate for updated models.

Examples

If one needs to print out LaTeX-formatted equations in a Jupyter Notebook, one need to generate such equations with

```
import andes
sys = andes.prepare()
```

Alternatively, one can explicitly create a System and generate the code

```
import andes
sys = andes.System()
sys.prepare()
```

System.reload

System.**reload**(***kwargs*)

Reload a new case in the same System object.

System.remove_pycapsule

System.**remove_pycapsule**(***kwargs*)

Remove PyCapsule objects in solvers.

System.report

System.**report**()

Write system routine reports to a plain-text file.

Returns

bool
True if the report is written successfully.

System.reset

System.**reset**(*force=False*)

Reset to the state after reading data and setup.

System.s_update_post

`System.s_update_post(**kwargs)`

Update variable services by calling `s_update_post` of models.

This function is called at the end of *System.init()*.

System.s_update_var

`System.s_update_var(**kwargs)`

Update variable services by calling `s_update_var` of models.

This function is must be called before any equation evaluation after limiter update function *l_update_var*.

System.save_config

`System.save_config(**kwargs)`

Save all system, model, and routine configurations to an rc-formatted file.

Parameters

file_path
[str, optional] path to the configuration file default to *~/andes/andes.rc*.

overwrite
[bool, optional] If file exists, True to overwrite without confirmation. Otherwise prompt for confirmation.

Warning: Saved config is loaded back and populated *at system instance creation time*. Configs from the config file takes precedence over default config values.

System.set_address

`System.set_address(models)`

Set addresses for differential and algebraic variables.

System.set_config

`System.set_config(**kwargs)`

Set configuration for the System object.

Config for models are routines are passed directly to their constructors.

System.set_dae_names

System.**set_dae_names**(***kwargs*)

Set variable names for differential and algebraic variables, right-hand side of external equations, and discrete flags.

System.set_output_subidx

System.**set_output_subidx**(***kwargs*)

Process `andes.models.misc.Output` data and store the sub-indices into `dae.xy`.

Parameters

models
[OrderedDict] Models currently in use for the routine

System.set_var_arrays

System.**set_var_arrays**(***kwargs*)

Set arrays (*v* and *e*) for internal variables to access dae arrays in place.

This function needs to be called after de-serializing a System object, where the internal variables are incorrectly assigned new memory.

Parameters

models
[OrderedDict, list, Model, optional] Models to execute.

inplace
[bool] True to retrieve arrays that share memory with dae

alloc
[bool] True to allocate for arrays internally

System.setup

System.**setup**()

Set up system for studies.

This function is to be called after adding all device data.

System.store_adder_setter

System.**store_adder_setter**(***kwargs*)

Store non-inplace adders and setters for variables and equations.

System.store_existing

System.**store_existing**()

Store existing models in *System.existing*.

TODO: Models with *TimerParam* will need to be stored anyway. This will allow adding switches on the fly.

System.store_no_check_init

System.**store_no_check_init**(***kwargs*)

Store differential variables with `check_init == False`.

System.store_sparse_pattern

System.**store_sparse_pattern**(***kwargs*)

Collect and store the sparsity pattern of Jacobian matrices.

This is a runtime function specific to cases.

Notes

For *gy* matrix, always make sure the diagonal is reserved. It is a safeguard if the modeling user omitted the diagonal term in the equations.

System.store_switch_times

System.**store_switch_times**(***kwargs*)

Store event switching time in a sorted Numpy array in `System.switch_times` and an OrderedDict `System.switch_dict`.

`System.switch_dict` has keys as event times and values as the OrderedDict of model names and instances associated with the event.

Parameters

models
[OrderedDict] model name : model instance

eps
[float] The small time step size to use immediately before and after the event

Returns

array-like
self.switch_times

System.summary

System.**summary**()

Print out system summary.

System.supported_models

System.**supported_models**(*export='plain'*)

Return the support group names and model names in a table.

Returns

str
A table-formatted string for the groups and models

System.supported_routines

System.**supported_routines**(*export='plain'*)

Return the support type names and routine names in a table.

Returns

str
A table-formatted string for the types and routines

System.switch_action

System.**switch_action**(***kwargs*)

Invoke the actions associated with switch times.

This function will not be called if `flat=True` is passed to system.

System.to_andes

System.**to_andes**(*setup=True*, *addfile=None*, ***kwargs*)

Convert the AMS system to an ANDES system.

A preferred dynamic system file to be added has following features: 1. The file contains both power flow and dynamic models. 2. The file can run in ANDES natively. 3. Power flow models are in the same shape as the AMS system. 4. Dynamic models, if any, are in the same shape as the AMS system.

Parameters

setup
[bool, optional] Whether to call *setup()* after the conversion. Default is True.

addfile
[str, optional] The additional file to be converted to ANDES dynamic mdoels.

****kwargs**
[dict] Keyword arguments to be passed to *andes.system.System*.

Returns

andes
[andes.system.System] The converted ANDES system.

Examples

```
>>> import ams
>>> import andes
>>> sp = ams.load(ams.get_case('ieee14/ieee14_rted.xlsx'), setup=True)
>>> sa = sp.to_andes(setup=False,
...                   addfile=andes.get_case('ieee14/ieee14_wt3.xlsx'),
...                   overwrite=True, no_keep=True, no_output=True)
```

System.to_ipysheet

`System.`**`to_ipysheet`**`(`*`**kwargs`*`)`

Return an ipysheet object for editing in Jupyter Notebook.

System.undill

`System.`**`undill`**`(`*`**kwargs`*`)`

Reload generated function functions, from either the `$HOME/.andes/pycode` folder.

If no change is made to models, future calls to `prepare()` can be replaced with `undill()` for acceleration.

Parameters

autogen_stale: bool
True to automatically call code generation if stale code is detected. Regardless of this option, codegen is trigger if importing existing code fails.

System.vars_to_dae

`System.`**`vars_to_dae`**`(`*`model`*`)`

Copy variables values from models to *System.dae*.

This function clears *DAE.x* and *DAE.y* and collects values from models.

System.vars_to_models

`System.`**`vars_to_models`**`()`

Copy variable values from *System.dae* to models.

7.2 Model

ams.core.model	Module for Model class.
ams.core.param	Base class for parameters.
ams.core.service	Service.

7.2.1 ams.core.model

Module for Model class.

Classes

Model([system, config])	Base class for power system dispatch models.

ams.core.model.Model

class ams.core.model.**Model**(*system=None*, *config=None*)

Base class for power system dispatch models.

This class is revised from `andes.core.model.Model`.

__init__(*system=None*, *config=None*)

Methods

alter(src, idx, value)	Alter values of input parameters or constant service.
doc([max_width, export])	Retrieve model documentation as a string.
get(src, idx[, attr, allow_none, default])	Get the value of an attribute of a model property.
get_idx()	Return the index of the model instance.
idx2uid(idx)	Convert idx to the 0-indexed unique index.
list2array()	Convert all the value attributes `v` to NumPy arrays.
set(src, idx, attr, value)	Set the value of an attribute of a model property.
set_backref(name, from_idx, to_idx)	Helper function for setting idx-es to `BackRef`.

Model.alter

Model.**alter**(*src*, *idx*, *value*)

Alter values of input parameters or constant service.

If the method operates on an input parameter, the new data should be in the same base as that in the input file. This function will convert the new value to per unit in the system base.

The values for storing the input data, i.e., the `vin` field of the parameter, will be overwritten, thus the update will be reflected in the dumped case file.

Parameters

src
[str] The parameter name to alter

idx
[str, float, int] The device to alter

value
[float] The desired value

Model.doc

`Model.doc(max_width=78, export='plain')`

Retrieve model documentation as a string.

Model.get

`Model.get(src: str, idx, attr: str = 'v', allow_none=False, default=0.0)`

Get the value of an attribute of a model property.

The return value is `self.<src>.<attr>[idx]`

Parameters

src
[str] Name of the model property

idx
[str, int, float, array-like] Indices of the devices

attr
[str, optional, default='v'] The attribute of the property to get. `v` for values, `a` for address, and `e` for equation value.

allow_none
[bool] True to allow None values in the indexer

default
[float] If *allow_none* is true, the default value to use for None indexer.

Returns

array-like
`self.<src>.<attr>[idx]`

Model.get_idx

`Model.get_idx()`

Return the index of the model instance. Equivalent to `self.idx.v`, develoepd for consistency with group method `get_idx`.

Model.idx2uid

`Model.idx2uid(idx)`

Convert idx to the 0-indexed unique index.

Parameters

idx
[array-like, numbers, or str] idx of devices

Returns

list
A list containing the unique indices of the devices

Model.list2array

`Model.list2array()`

Convert all the value attributes `v` to NumPy arrays.

Value attribute arrays should remain in the same address afterwards. Namely, all assignments to value array should be operated in place (e.g., with [:]).

Model.set

`Model.set(src, idx, attr, value)`

Set the value of an attribute of a model property.

Performs `self.<src>.<attr>[idx] = value`. This method will not modify the input values from the case file that have not been converted to the system base. As a result, changes applied by this method will not affect the dumped case file.

To alter parameters and reflect it in the case file, use `alter()` instead.

Parameters

src
[str] Name of the model property

idx
[str, int, float, array-like] Indices of the devices

attr
[str, optional, default='v'] The internal attribute of the property to get. `v` for values, `a` for address, and `e` for equation value.

value
[array-like] New values to be set

Returns

bool
True when successful.

Model.set_backref

Model.**set_backref**(*name*, *from_idx*, *to_idx*)

Helper function for setting idx-es to `BackRef`.

Attributes

class_name	Return the class name

Model.class_name

property Model.**class_name**

Return the class name

7.2.2 ams.core.param

Base class for parameters.

Classes

RParam([name, tex_name, info, src, unit, ...])	Class for parameters used in a routine.

ams.core.param.RParam

class ams.core.param.**RParam**(*name: str | None = None, tex_name: str | None = None, info: str | None = None, src: str | None = None, unit: str | None = None, model: str | None = None, v: ndarray | None = None, indexer: str | None = None, imodel: str | None = None, expand_dims: int | None = None, no_parse: bool | None = False, nonneg: bool | None = False, nonpos: bool | None = False, complex: bool | None = False, imag: bool | None = False, symmetric: bool | None = False, diag: bool | None = False, hermitian: bool | None = False, boolean: bool | None = False, integer: bool | None = False, pos: bool | None = False, neg: bool | None = False, sparse: list | None = None*)

Class for parameters used in a routine. This class is developed to simplify the routine definition.

RParm is further used to define *Parameter* in the optimization model.

no_parse is used to skip parsing the *RParam* in optimization model. It means that the *RParam* will not be added to the optimization model. This is useful when the RParam contains non-numeric values, or it is not necessary to be added to the optimization model.

Parameters

name
[str, optional] Name of this parameter. If not provided, *name* will be set to the attribute name.

tex_name
[str, optional] LaTeX-formatted parameter name. If not provided, *tex_name* will be assigned the same as *name*.

info
: [str, optional] A description of this parameter

src
: [str, optional] Source name of the parameter.

unit
: [str, optional] Unit of the parameter.

model
: [str, optional] Name of the owner model or group.

v
: [np.ndarray, optional] External value of the parameter.

indexer
: [str, optional] Indexer of the parameter.

imodel
: [str, optional] Name of the owner model or group of the indexer.

no_parse: bool, optional
: True to skip parsing the parameter.

nonneg: bool, optional
: True to set the parameter as non-negative.

nonpos: bool, optional
: True to set the parameter as non-positive.

complex: bool, optional
: True to set the parameter as complex.

imag: bool, optional
: True to set the parameter as imaginary.

symmetric: bool, optional
: True to set the parameter as symmetric.

diag: bool, optional
: True to set the parameter as diagonal.

hermitian: bool, optional
: True to set the parameter as hermitian.

boolean: bool, optional
: True to set the parameter as boolean.

integer: bool, optional
: True to set the parameter as integer.

pos: bool, optional
: True to set the parameter as positive.

neg: bool, optional
: True to set the parameter as negative.

sparse: bool, optional
: True to set the parameter as sparse.

Examples

Example 1: Define a routine parameter from a source model or group.

In this example, we define the parameter *cru* from the source model *SFRCost* with the parameter *cru*.

```
>>> self.cru = RParam(info='RegUp reserve coefficient',
>>>                   tex_name=r'c_{r,u}',
>>>                   unit=r'$/(p.u.)',
>>>                   name='cru',
>>>                   src='cru',
>>>                   model='SFRCost'
>>>                   )
```

Example 2: Define a routine parameter with a user-defined value.

In this example, we define the parameter with a user-defined value. TODO: Add example

__init__(*name: str | None = None, tex_name: str | None = None, info: str | None = None, src: str | None = None, unit: str | None = None, model: str | None = None, v: ndarray | None = None, indexer: str | None = None, imodel: str | None = None, expand_dims: int | None = None, no_parse: bool | None = False, nonneg: bool | None = False, nonpos: bool | None = False, complex: bool | None = False, imag: bool | None = False, symmetric: bool | None = False, diag: bool | None = False, hermitian: bool | None = False, boolean: bool | None = False, integer: bool | None = False, pos: bool | None = False, neg: bool | None = False, sparse: list | None = None*)

Methods

get_idx()	Get the index of the parameter.
parse()	Parse the parameter.
update()	Update the Parameter value.

RParam.get_idx

RParam.**get_idx**()

Get the index of the parameter.

Returns

idx
[list] Index of the parameter.

Notes

- The value will sort by the indexer if indexed.

RParam.parse

RParam.**parse**()

Parse the parameter.

RParam.update

RParam.**update**()

Update the Parameter value.

Attributes

class_name	Return the class name
dtype	Return the data type of the parameter value.
n	Return the szie of the parameter.
shape	Return the shape of the parameter.
size	Return the size.
v	The value of the parameter.

RParam.class_name

property RParam.**class_name**

Return the class name

RParam.dtype

property RParam.**dtype**

Return the data type of the parameter value.

RParam.n

property RParam.**n**

Return the szie of the parameter.

RParam.shape

property RParam.**shape**

Return the shape of the parameter.

RParam.size

property RParam.**size**

Return the size.

RParam.v

property RParam.**v**

The value of the parameter.

Notes

- This property is a wrapper for the `get` method of the owner class.
- The value will sort by the indexer if indexed, used for optmization modeling.

7.2.3 ams.core.service

Service.

Classes

LoadScale(u, sd[, name, tex_name, unit, ...])	Return load.
MinDur(u, u2[, name, tex_name, unit, info, ...])	Defined to form minimum on matrix for minimum online/offline time constraints used in UC.
NumExpandDim(u[, axis, args, name, ...])	Expand the dimensions of the input array along a specified axis using NumPy's `np.expand_dims(u.v, axis=axis)`.
NumHstack(u, ref[, args, name, tex_name, ...])	Repeat an array along the second axis nc times or the length of reference array, using NumPy's hstack function, where nc is the column number of the reference array, `np.hstack([u.v[:, np.newaxis] * ref.shape[1]], **kwargs)`.
NumOp(u, fun[, args, name, tex_name, unit, ...])	Perform an operation on a numerical array using the function `fun(u.v, **args)`.
NumOpDual(u, u2, fun[, args, name, ...])	Performan an operation on two numerical arrays using the function `fun(u.v, u2.v, **args)`.
RBaseService([name, tex_name, unit, info, ...])	Base class for services that are used in a routine.
ROperationService(u[, name, tex_name, unit, ...])	Base calss for operational services used in routine.
RampSub(u[, name, tex_name, unit, info, ...])	Build a substraction matrix for a 2D variable in the shape (nr, nr-1), where nr is the rows of the input.
ValueService(name, value[, tex_name, unit, ...])	Service to store given numeric values.
VarReduction(u, fun[, name, tex_name, unit, ...])	A numerical matrix to reduce a 2D variable to 1D, `np.fun(shape=(1, u.n))`.
VarSelect(u, indexer[, gamma, name, ...])	A numerical matrix to select a subset of a 2D variable, `u.v[:, idx]`.
ZonalSum(u, zone[, name, tex_name, unit, ...])	Build zonal sum matrix for a vector in the shape of collection model, `Area` or `Region`.

ams.core.service.LoadScale

class ams.core.service.**LoadScale**(*u: Callable*, *sd: Callable*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *no_parse: bool = False*, *sparse: bool = False*)

Return load.

Parameters

u
[Callable] nodal load.

sd
[Callable] zonal load factor.

name
[str, optional] Instance name.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable*, *sd: Callable*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *no_parse: bool = False*, *sparse: bool = False*)

Methods

assign_memory(n)	Assign memory for `self.v` and set the array to zero.
get_names()	Return *name* in a list
parse()	Parse the parameter.
update()	Update the Parameter value.

LoadScale.assign_memory

LoadScale.**assign_memory**(*n*)

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

LoadScale.get_names

`LoadScale.get_names()`

Return *name* in a list

Returns

list
A list only containing the name of the service variable

LoadScale.parse

`LoadScale.parse()`

Parse the parameter.

LoadScale.update

`LoadScale.update()`

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.

LoadScale.class_name

`property LoadScale.class_name`

Return the class name

LoadScale.n

`property LoadScale.n`

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

LoadScale.shape

property LoadScale.**shape**

Return the shape of the service.

LoadScale.size

property LoadScale.**size**

Return the size.

LoadScale.v

property LoadScale.**v**

Value of the service.

ams.core.service.MinDur

class ams.core.service.**MinDur**(*u: Callable*, *u2: Callable*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *no_parse: bool = False*, *sparse: bool = False*)

Defined to form minimum on matrix for minimum online/offline time constraints used in UC.

Parameters

u
[Callable] Input, should be a `Var` with horizon.

u2
[Callable] Input2, should be a `RParam`.

name
[str, optional] Instance name.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable*, *u2: Callable*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *no_parse: bool = False*, *sparse: bool = False*)

Methods

assign_memory(n)	Assign memory for `self.v` and set the array to zero.
get_names()	Return *name* in a list
parse()	Parse the parameter.
update()	Update the Parameter value.

MinDur.assign_memory

`MinDur.assign_memory(n)`

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

MinDur.get_names

`MinDur.get_names()`

Return *name* in a list

Returns

list
A list only containing the name of the service variable

MinDur.parse

`MinDur.parse()`

Parse the parameter.

MinDur.update

`MinDur.update()`

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.
v0	
v1	

MinDur.class_name

property MinDur.**class_name**

Return the class name

MinDur.n

property MinDur.**n**

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

MinDur.shape

property MinDur.**shape**

Return the shape of the service.

MinDur.size

property MinDur.**size**

Return the size.

MinDur.v

property MinDur.**v**

Value of the service.

MinDur.v0

property MinDur.**v0**

MinDur.v1

property MinDur.**v1**

ams.core.service.NumExpandDim

class ams.core.service.**NumExpandDim**(*u: Callable, axis: int = 0, args: dict = {}, name: str = None, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, array_out: bool = True, no_parse: bool = False, sparse: bool = False*)

Expand the dimensions of the input array along a specified axis using NumPy's `np.expand_dims(u.v, axis=axis)`.

Parameters

u
[Callable] Input.

axis
[int] Axis along which to expand the dimensions (default is 0).

name
[str, optional] Instance name.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

vtype
[Type, optional] Variable type.

array_out
[bool, optional] Whether to force the output to be an array.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable, axis: int = 0, args: dict = {}, name: str = None, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, array_out: bool = True, no_parse: bool = False, sparse: bool = False*)

Methods

`assign_memory`(n)	Assign memory for `self.v` and set the array to zero.
`get_names`()	Return *name* in a list
`parse`()	Parse the parameter.
`update`()	Update the Parameter value.

NumExpandDim.assign_memory

NumExpandDim.**assign_memory**(*n*)

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

NumExpandDim.get_names

NumExpandDim.**get_names**()

Return *name* in a list

Returns

list
A list only containing the name of the service variable

NumExpandDim.parse

NumExpandDim.**parse**()

Parse the parameter.

NumExpandDim.update

NumExpandDim.**update**()

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.
v0	
v1	

NumExpandDim.class_name

property `NumExpandDim.class_name`

Return the class name

NumExpandDim.n

property `NumExpandDim.n`

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

NumExpandDim.shape

property `NumExpandDim.shape`

Return the shape of the service.

NumExpandDim.size

property `NumExpandDim.size`

Return the size.

NumExpandDim.v

property `NumExpandDim.v`

Value of the service.

NumExpandDim.v0

property `NumExpandDim.v0`

NumExpandDim.v1

property `NumExpandDim.v1`

ams.core.service.NumHstack

class ams.core.service.**NumHstack**(*u: Callable*, *ref: Callable*, *args: dict = {}*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *no_parse: bool = False*, *sparse: bool = False*)

Repeat an array along the second axis nc times or the length of reference array, using NumPy's hstack function, where nc is the column number of the reference array, `np.hstack([u.v[:, np.newaxis] * ref.shape[1]], **kwargs)`.

Parameters

u
[Callable] Input array.

ref
[Callable] Reference array used to determine the number of repetitions.

name
[str, optional] Instance name.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

vtype
[Type, optional] Variable type.

model
[str, optional] Model name.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable*, *ref: Callable*, *args: dict = {}*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *no_parse: bool = False*, *sparse: bool = False*)

Methods

`assign_memory`(n)	Assign memory for `self.v` and set the array to zero.
`get_names`()	Return *name* in a list
`parse`()	Parse the parameter.
`update`()	Update the Parameter value.

NumHstack.assign_memory

NumHstack.**assign_memory**(*n*)

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

NumHstack.get_names

NumHstack.**get_names**()

Return *name* in a list

Returns

list
A list only containing the name of the service variable

NumHstack.parse

NumHstack.**parse**()

Parse the parameter.

NumHstack.update

NumHstack.**update**()

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.
v0	
v1	

NumHstack.class_name

property NumHstack.**class_name**

Return the class name

NumHstack.n

property NumHstack.**n**

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

NumHstack.shape

property NumHstack.**shape**

Return the shape of the service.

NumHstack.size

property NumHstack.**size**

Return the size.

NumHstack.v

property NumHstack.**v**

Value of the service.

NumHstack.v0

property NumHstack.**v0**

NumHstack.v1

property NumHstack.**v1**

ams.core.service.NumOp

class ams.core.service.**NumOp**(*u: Callable*, *fun: Callable*, *args: dict = {}*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *expand_dims: int = None*, *array_out=True*, *no_parse: bool = False*, *sparse: bool = False*)

Perform an operation on a numerical array using the function `fun(u.v, **args)`.

Note that the scalar output is converted to a 1D array.

The optional kwargs are passed to the input function.

Parameters

u
[Callable] Input.

name
[str, optional] Instance name.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

vtype
[Type, optional] Variable type.

model
[str, optional] Model name.

rfun
[Callable, optional] Function to apply to the output of `fun`.

rargs
[dict, optional] Keyword arguments to pass to `rfun`.

expand_dims
[int, optional] Expand the dimensions of the output array along a specified axis.

array_out
[bool, optional] Whether to force the output to be an array.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable*, *fun: Callable*, *args: dict = {}*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *expand_dims: int = None*, *array_out=True*, *no_parse: bool = False*, *sparse: bool = False*)

Methods

assign_memory(n)	Assign memory for `self.v` and set the array to zero.
get_names()	Return *name* in a list
parse()	Parse the parameter.
update()	Update the Parameter value.

NumOp.assign_memory

`NumOp.assign_memory(n)`

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

NumOp.get_names

`NumOp.get_names()`

Return *name* in a list

Returns

list
A list only containing the name of the service variable

NumOp.parse

`NumOp.parse()`

Parse the parameter.

NumOp.update

`NumOp.update()`

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.
v0	
v1	

NumOp.class_name

property NumOp.**class_name**

Return the class name

NumOp.n

property NumOp.**n**

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

NumOp.shape

property NumOp.**shape**

Return the shape of the service.

NumOp.size

property NumOp.**size**

Return the size.

NumOp.v

property NumOp.**v**

Value of the service.

NumOp.v0

property NumOp.**v0**

NumOp.v1

property NumOp.**v1**

ams.core.service.NumOpDual

class ams.core.service.**NumOpDual**(*u: Callable, u2: Callable, fun: Callable, args: dict = {}, name: str = None, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, rfun: Callable = None, rargs: dict = {}, expand_dims: int = None, array_out=True, no_parse: bool = False, sparse: bool = False*)

Performan an operation on two numerical arrays using the function `fun(u.v, u2.v, **args)`.

Note that the scalar output is converted to a 1D array.

The optional kwargs are passed to the input function.

Parameters

u
[Callable] Input.

u2
[Callable] Input2.

name
[str, optional] Instance name.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

vtype
[Type, optional] Variable type.

model
[str, optional] Model name.

rfun
[Callable, optional] Function to apply to the output of `fun`.

rargs
[dict, optional] Keyword arguments to pass to `rfun`.

expand_dims
[int, optional] Expand the dimensions of the output array along a specified axis.

array_out
[bool, optional] Whether to force the output to be an array.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable, u2: Callable, fun: Callable, args: dict = {}, name: str = None, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, rfun: Callable = None, rargs: dict = {}, expand_dims: int = None, array_out=True, no_parse: bool = False, sparse: bool = False*)

Methods

assign_memory(n)	Assign memory for `self.v` and set the array to zero.
get_names()	Return *name* in a list
parse()	Parse the parameter.
update()	Update the Parameter value.

NumOpDual.assign_memory

`NumOpDual.assign_memory(n)`

Assign memory for `self.v` and set the array to zero.

Parameters

n

[int] Number of elements of the value array. Provided by caller (Model.list2array).

NumOpDual.get_names

`NumOpDual.get_names()`

Return *name* in a list

Returns

list

A list only containing the name of the service variable

NumOpDual.parse

`NumOpDual.parse()`

Parse the parameter.

NumOpDual.update

`NumOpDual.update()`

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.
v0	
v1	

NumOpDual.class_name

property NumOpDual.**class_name**

Return the class name

NumOpDual.n

property NumOpDual.**n**

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

NumOpDual.shape

property NumOpDual.**shape**

Return the shape of the service.

NumOpDual.size

property NumOpDual.**size**

Return the size.

NumOpDual.v

property NumOpDual.**v**

Value of the service.

NumOpDual.v0

property NumOpDual.**v0**

NumOpDual.v1

property NumOpDual.**v1**

ams.core.service.RBaseService

class ams.core.service.**RBaseService**(*name: str = None, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, no_parse: bool = False, sparse: bool = False*)

Base class for services that are used in a routine. Revised from module *andes.core.service.BaseService*.

Parameters

name
[str, optional] Instance name.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

vtype
[Type, optional] Variable type.

model
[str, optional] Model name.

no_parse: bool, optional
True to skip parsing the service.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*name: str = None, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, no_parse: bool = False, sparse: bool = False*)

Methods

`assign_memory`(n)	Assign memory for `self.v` and set the array to zero.
`get_names`()	Return *name* in a list
`parse`()	Parse the parameter.
`update`()	Update the Parameter value.

RBaseService.assign_memory

RBaseService.**assign_memory**(*n*)

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

RBaseService.get_names

`RBaseService.get_names()`

Return *name* in a list

Returns

list
A list only containing the name of the service variable

RBaseService.parse

`RBaseService.parse()`

Parse the parameter.

RBaseService.update

`RBaseService.update()`

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.

RBaseService.class_name

`property RBaseService.class_name`

Return the class name

RBaseService.n

`property RBaseService.n`

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

RBaseService.shape

property RBaseService.**shape**

Return the shape of the service.

RBaseService.size

property RBaseService.**size**

Return the size.

RBaseService.v

property RBaseService.**v**

Value of the service.

ams.core.service.ROperationService

class ams.core.service.**ROperationService**(*u: Callable*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *no_parse: bool = False*, *sparse: bool = False*)

Base calss for operational services used in routine.

Parameters

u
[Callable] Input.

name
[str, optional] Instance name.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

vtype
[Type, optional] Variable type.

model
[str, optional] Model name.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *no_parse: bool = False*, *sparse: bool = False*)

Methods

assign_memory(n)	Assign memory for `self.v` and set the array to zero.
get_names()	Return *name* in a list
parse()	Parse the parameter.
update()	Update the Parameter value.

ROperationService.assign_memory

ROperationService.**assign_memory**(*n*)

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

ROperationService.get_names

ROperationService.**get_names**()

Return *name* in a list

Returns

list
A list only containing the name of the service variable

ROperationService.parse

ROperationService.**parse**()

Parse the parameter.

ROperationService.update

ROperationService.**update**()

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.

ROperationService.class_name

property ROperationService.**class_name**

Return the class name

ROperationService.n

property ROperationService.**n**

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

ROperationService.shape

property ROperationService.**shape**

Return the shape of the service.

ROperationService.size

property ROperationService.**size**

Return the size.

ROperationService.v

property ROperationService.**v**

Value of the service.

ams.core.service.RampSub

class ams.core.service.**RampSub**(*u: Callable, name: str = None, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, rfun: Callable = None, rargs: dict = {}, no_parse: bool = False, sparse: bool = False*)

Build a substraction matrix for a 2D variable in the shape (nr, nr-1), where nr is the rows of the input.

This can be used for generator ramping constraints in multi-period optimization problems.

The subtraction matrix is constructed as follows: `np.eye(nr, nc, k=-1) - np.eye(nr, nc, k=0)`.

Parameters

u
[Callable] Input.

horizon
[Callable] Horizon reference.

name
[str] Instance name.

tex_name
[str] TeX name.

unit
[str] Unit.

info
[str] Description.

vtype
[Type] Variable type.

model
[str] Model name.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *no_parse: bool = False*, *sparse: bool = False*)

Methods

`assign_memory(n)`	Assign memory for `self.v` and set the array to zero.
`get_names()`	Return *name* in a list
`parse()`	Parse the parameter.
`update()`	Update the Parameter value.

RampSub.assign_memory

RampSub.**assign_memory**(*n*)

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

RampSub.get_names

RampSub.**get_names**()

Return *name* in a list

Returns

list
A list only containing the name of the service variable

RampSub.parse

RampSub.**parse**()

Parse the parameter.

RampSub.update

RampSub.**update**()

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.
v0	
v1	

RampSub.class_name

property RampSub.**class_name**

Return the class name

RampSub.n

property RampSub.**n**

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

RampSub.shape

property RampSub.**shape**

Return the shape of the service.

RampSub.size

`property RampSub.size`

Return the size.

RampSub.v

`property RampSub.v`

Value of the service.

RampSub.v0

`property RampSub.v0`

RampSub.v1

`property RampSub.v1`

ams.core.service.ValueService

`class ams.core.service.ValueService(name: str, value: ndarray, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, no_parse: bool = False, sparse: bool = False)`

Service to store given numeric values.

Parameters

name
[str, optional] Instance name.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

vtype
[Type, optional] Variable type.

model
[str, optional] Model name.

sparse: bool, optional
True to return output as scipy csr_matrix.

`__init__(name: str, value: ndarray, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, no_parse: bool = False, sparse: bool = False)`

Methods

assign_memory(n)	Assign memory for `self.v` and set the array to zero.
get_names()	Return *name* in a list
parse()	Parse the parameter.
update()	Update the Parameter value.

ValueService.assign_memory

`ValueService.assign_memory(n)`

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

ValueService.get_names

`ValueService.get_names()`

Return *name* in a list

Returns

list
A list only containing the name of the service variable

ValueService.parse

`ValueService.parse()`

Parse the parameter.

ValueService.update

`ValueService.update()`

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.

ValueService.class_name

property ValueService.**class_name**

Return the class name

ValueService.n

property ValueService.**n**

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

ValueService.shape

property ValueService.**shape**

Return the shape of the service.

ValueService.size

property ValueService.**size**

Return the size.

ValueService.v

property ValueService.**v**

Value of the service.

ams.core.service.VarReduction

class ams.core.service.**VarReduction**(*u: Callable*, *fun: Callable*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *no_parse: bool = False*, *sparse: bool = False*, ***kwargs*)

A numerical matrix to reduce a 2D variable to 1D, `np.fun(shape=(1, u.n))`.

Parameters

u
[Callable] The input matrix variable.

fun
[Callable] The reduction function that takes a shape parameter (1D shape) as input.

name
[str, optional] The name of the instance.

tex_name
[str, optional] The TeX name for the instance.

unit
[str, optional] The unit of the output.

info
[str, optional] A description of the operation.

vtype
[Type, optional] The variable type.

model
[str, optional] The model name associated with the operation.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable*, *fun: Callable*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *no_parse: bool = False*, *sparse: bool = False*, ***kwargs*)

Methods

`assign_memory`(n)	Assign memory for `self.v` and set the array to zero.
`get_names`()	Return *name* in a list
`parse`()	Parse the parameter.
`update`()	Update the Parameter value.

VarReduction.assign_memory

`VarReduction.assign_memory(n)`

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

VarReduction.get_names

`VarReduction.get_names()`

Return *name* in a list

Returns

list
A list only containing the name of the service variable

VarReduction.parse

`VarReduction.parse()`

Parse the parameter.

VarReduction.update

`VarReduction.update()`

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.
v0	
v1	

VarReduction.class_name

property `VarReduction.class_name`

Return the class name

VarReduction.n

property `VarReduction.n`

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

VarReduction.shape

property `VarReduction.shape`

Return the shape of the service.

VarReduction.size

property VarReduction.**size**

Return the size.

VarReduction.v

property VarReduction.**v**

Value of the service.

VarReduction.v0

property VarReduction.**v0**

VarReduction.v1

property VarReduction.**v1**

ams.core.service.VarSelect

class ams.core.service.**VarSelect**(*u: Callable*, *indexer: str*, *gamma: str = None*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *array_out: bool = True*, *no_parse: bool = False*, *sparse: bool = False*, ***kwargs*)

A numerical matrix to select a subset of a 2D variable, `u.v[:, idx]`.

For example, if nned to select Energy Storage output power from StaticGen *pg*, following definition can be used: `` `python class RTED: ... self.ce = VarSelect(u=self.pg, indexer='genE') ... ` ``

Parameters

u
[Callable] The input matrix variable.

indexer: str
The name of the indexer source.

gamma
[str, optional] The name of the indexer gamma.

name
[str, optional] The name of the instance.

tex_name
[str, optional] The TeX name for the instance.

unit
[str, optional] The unit of the output.

info
[str, optional] A description of the operation.

vtype
[Type, optional] The variable type.

rfun
[Callable, optional] Function to apply to the output of `fun`.

rargs
[dict, optional] Keyword arguments to pass to `rfun`.

array_out
[bool, optional] Whether to force the output to be an array.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable*, *indexer: str*, *gamma: str = None*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *array_out: bool = True*, *no_parse: bool = False*, *sparse: bool = False*, ***kwargs*)

Methods

`assign_memory`(n)	Assign memory for `self.v` and set the array to zero.
`get_names`()	Return *name* in a list
`parse`()	Parse the parameter.
`update`()	Update the Parameter value.

VarSelect.assign_memory

`VarSelect.`**`assign_memory`**(*n*)

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

VarSelect.get_names

`VarSelect.`**`get_names`**()

Return *name* in a list

Returns

list
A list only containing the name of the service variable

VarSelect.parse

`VarSelect.parse()`

Parse the parameter.

VarSelect.update

`VarSelect.update()`

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.
v0	
v1	

VarSelect.class_name

property `VarSelect.class_name`

Return the class name

VarSelect.n

property `VarSelect.n`

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

VarSelect.shape

property `VarSelect.shape`

Return the shape of the service.

VarSelect.size

property VarSelect.**size**

Return the size.

VarSelect.v

property VarSelect.**v**

Value of the service.

VarSelect.v0

property VarSelect.**v0**

VarSelect.v1

property VarSelect.**v1**

ams.core.service.ZonalSum

class ams.core.service.**ZonalSum**(*u: Callable*, *zone: str*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *no_parse: bool = False*, *sparse: bool = False*)

Build zonal sum matrix for a vector in the shape of collection model, `Area` or `Region`. The value array is in the shape of (nr, nc), where nr is the length of rid instance idx, and nc is the length of the cid value.

In an IEEE-14 Bus system, we have the zonal definition by the `Region` model. Suppose in it we have two regions, "ZONE1" and "ZONE2".

Follwing it, we have a zonal SFR requirement model `SFR` that defines the zonal reserve requirements for each zone.

All 14 buses are classified to a zone by the *IdxParam* `zone`, and the 5 generators are connected to buses (idx): [2, 3, 1, 6, 8], and the zone of these generators are thereby: ['ZONE1', 'ZONE1', 'ZONE2', 'ZONE2', 'ZONE1'].

In the *RTED* model, we have the Vars `pru` and `prd` in the shape of generators.

Then, the Region model has idx ['ZONE1', 'ZONE2'], and the `gsm` value will be [[1, 1, 0, 0, 1], [0, 0, 1, 1, 0]].

Finally, the zonal reserve requirements can be formulated as constraints in the optimization problem: "gsm @ pru <= du" and "gsm @ prd <= dd".

See `gsm` definition in `ams.routines.rted.RTEDModel` for more details.

Parameters

u
[Callable] Input.

zone
[str] Zonal model name, e.g., "Area" or "Region".

name
[str] Instance name.

tex_name
[str] TeX name.

unit
[str] Unit.

info
[str] Description.

vtype
[Type] Variable type.

model
[str] Model name.

sparse: bool, optional
True to return output as scipy csr_matrix.

__init__(*u: Callable*, *zone: str*, *name: str = None*, *tex_name: str = None*, *unit: str = None*, *info: str = None*, *vtype: Type = None*, *rfun: Callable = None*, *rargs: dict = {}*, *no_parse: bool = False*, *sparse: bool = False*)

Methods

`assign_memory(n)`	Assign memory for `self.v` and set the array to zero.
`get_names()`	Return *name* in a list
`parse()`	Parse the parameter.
`update()`	Update the Parameter value.

ZonalSum.assign_memory

`ZonalSum.`**`assign_memory`**`(n)`

Assign memory for `self.v` and set the array to zero.

Parameters

n
[int] Number of elements of the value array. Provided by caller (Model.list2array).

ZonalSum.get_names

`ZonalSum.`**`get_names`**`()`

Return *name* in a list

Returns

list
A list only containing the name of the service variable

ZonalSum.parse

ZonalSum.**parse**()

Parse the parameter.

ZonalSum.update

ZonalSum.**update**()

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the count of values in `self.v`.
shape	Return the shape of the service.
size	Return the size.
v	Value of the service.
v0	
v1	

ZonalSum.class_name

property ZonalSum.**class_name**

Return the class name

ZonalSum.n

property ZonalSum.**n**

Return the count of values in `self.v`.

Needs to be overloaded if `v` of subclasses is not a 1-dimensional array.

Returns

int
The count of elements in this variable

ZonalSum.shape

property ZonalSum.**shape**

Return the shape of the service.

ZonalSum.size

property ZonalSum.**size**

Return the size.

ZonalSum.v

property ZonalSum.**v**

Value of the service.

ZonalSum.v0

property ZonalSum.**v0**

ZonalSum.v1

property ZonalSum.**v1**

7.3 Routines

ams.routines.routine	Module for routine data.

7.3.1 ams.routines.routine

Module for routine data.

Classes

RoutineBase([system, config])	Class to hold descriptive routine models and data mapping.

ams.routines.routine.RoutineBase

class ams.routines.routine.**RoutineBase**(*system=None*, *config=None*)

Class to hold descriptive routine models and data mapping.

__init__(*system=None*, *config=None*)

Methods

addConstrs(name, e_str[, info, type])	Add *Constraint* to the routine.
addRParam(name[, tex_name, info, src, unit, ...])	Add *RParam* to the routine.
addService(name, value[, tex_name, unit, ...])	Add *ValueService* to the routine.
addVars(name[, model, shape, tex_name, ...])	Add a variable to the routine.
dc2ac(**kwargs)	Convert the DC-based results with ACOPF.
disable(name)	Disable a constraint by name.
doc([max_width, export])	Retrieve routine documentation as a string.
enable(name)	Enable a constraint by name.
export_csv([path])	Export dispatch results to a csv file.
get(src, idx[, attr, horizon])	Get the value of a variable or parameter.
get_load(horizon, src[, attr, idx, model, ...])	Get the load value by applying zonal scaling factor defined in `Horizon`.
igmake([directed])	Build an igraph object from the system.
igraph([input, ytimes, decimal, directed, ...])	Plot a system uging *g.plot()* of *igraph*, with optional input.
init([force, no_code])	Initialize the routine.
prepare()	Prepare the routine.
run([force_init, no_code])	Run the routine.
set(src, idx[, attr, value])	Set the value of an attribute of a routine parameter.
solve(**kwargs)	Solve the routine optimization model.
summary(**kwargs)	Summary interface
unpack(**kwargs)	Unpack the results.
update([params, mat_make])	Update the values of Parameters in the optimization model.

RoutineBase.addConstrs

`RoutineBase.addConstrs`(*name: str*, *e_str: str*, *info: str | None = None*, *type: str | None = 'uq'*)

Add *Constraint* to the routine. to the routine.

Parameters

name
[str] Constraint name. One should typically assigning the name directly because it will be automatically assigned by the model. The value of `name` will be the symbol name to be used in expressions.

e_str
[str] Constraint expression string.

info
[str, optional] Descriptive information

type
[str, optional] Constraint type, `uq` for uncertain, `eq` for equality, `ineq` for inequality.

RoutineBase.addRParam

RoutineBase.**addRParam**(*name: str, tex_name: str | None = None, info: str | None = None, src: str | None = None, unit: str | None = None, model: str | None = None, v: ndarray | None = None, indexer: str | None = None, imodel: str | None = None*)

Add *RParam* to the routine.

Parameters

name
[str] Name of this parameter. If not provided, *name* will be set to the attribute name.

tex_name
[str, optional] LaTeX-formatted parameter name. If not provided, *tex_name* will be assigned the same as *name*.

info
[str, optional] A description of this parameter

src
[str, optional] Source name of the parameter.

unit
[str, optional] Unit of the parameter.

model
[str, optional] Name of the owner model or group.

v
[np.ndarray, optional] External value of the parameter.

indexer
[str, optional] Indexer of the parameter.

imodel
[str, optional] Name of the owner model or group of the indexer.

RoutineBase.addService

RoutineBase.**addService**(*name: str, value: ndarray, tex_name: str = None, unit: str = None, info: str = None, vtype: Type = None, model: str = None*)

Add *ValueService* to the routine.

Parameters

name
[str] Instance name.

value
[np.ndarray] Value.

tex_name
[str, optional] TeX name.

unit
[str, optional] Unit.

info
[str, optional] Description.

vtype
[Type, optional] Variable type.

model
[str, optional] Model name.

RoutineBase.addVars

RoutineBase.**addVars**(*name: str*, *model: str | None = None*, *shape: tuple | int | None = None*, *tex_name: str | None = None*, *info: str | None = None*, *src: str | None = None*, *unit: str | None = None*, *horizon: RParam | None = None*, *nonneg: bool | None = False*, *nonpos: bool | None = False*, *complex: bool | None = False*, *imag: bool | None = False*, *symmetric: bool | None = False*, *diag: bool | None = False*, *psd: bool | None = False*, *nsd: bool | None = False*, *hermitian: bool | None = False*, *bool: bool | None = False*, *integer: bool | None = False*, *pos: bool | None = False*, *neg: bool | None = False*)

Add a variable to the routine.

Parameters

name
[str, optional] Variable name. One should typically assigning the name directly because it will be automatically assigned by the model. The value of `name` will be the symbol name to be used in expressions.

model
[str, optional] Name of the owner model or group.

shape
[int or tuple, optional] Shape of the variable. If is None, the shape of *model* will be used.

info
[str, optional] Descriptive information

unit
[str, optional] Unit

tex_name
[str] LaTeX-formatted variable symbol. If is None, the value of *name* will be used.

src
[str, optional] Source variable name. If is None, the value of *name* will be used.

lb
[str, optional] Lower bound

ub
[str, optional] Upper bound

horizon
[ams.routines.RParam, optional] Horizon idx.

nonneg
[bool, optional] Non-negative variable

nonpos
[bool, optional] Non-positive variable

complex
[bool, optional] Complex variable

imag
[bool, optional] Imaginary variable

symmetric
[bool, optional] Symmetric variable

diag
[bool, optional] Diagonal variable

psd
[bool, optional] Positive semi-definite variable

nsd
[bool, optional] Negative semi-definite variable

hermitian
[bool, optional] Hermitian variable

bool
[bool, optional] Boolean variable

integer
[bool, optional] Integer variable

pos
[bool, optional] Positive variable

neg
[bool, optional] Negative variable

RoutineBase.dc2ac

`RoutineBase.dc2ac`(***kwargs*)

Convert the DC-based results with ACOPF.

RoutineBase.disable

`RoutineBase.disable`(*name*)

Disable a constraint by name.

Parameters

name: str or list
name of the constraint to be disabled

RoutineBase.doc

`RoutineBase.doc`(*max_width=78*, *export='plain'*)

Retrieve routine documentation as a string.

RoutineBase.enable

RoutineBase.**enable**(*name*)

Enable a constraint by name.

Parameters

name: str or list
name of the constraint to be enabled

RoutineBase.export_csv

RoutineBase.**export_csv**(*path=None*)

Export dispatch results to a csv file. For multi-period routines, the column "Time" is the time index of `timeslot.v`, which usually comes from `EDTSlot` or `UCTSlot`. The rest columns are the variables registered in `vars`.

For single-period routines, the column "Time" have a pseduo value of "T1".

Parameters

path
[str] path of the csv file to save

Returns

str
The path of the exported csv file

RoutineBase.get

RoutineBase.**get**(*src: str*, *idx*, *attr: str = 'v'*, *horizon: int | str | Iterable | None = None*)

Get the value of a variable or parameter.

Parameters

src: str
Name of the variable or parameter.

idx: int, str, or list
Index of the variable or parameter.

attr: str
Attribute name.

horizon: list, optional
Horizon index.

RoutineBase.get_load

RoutineBase.**get_load**(*horizon: int | str*, *src: str*, *attr: str = 'v'*, *idx=None*, *model: str = 'EDTSlot'*, *factor: str = 'sd'*)

Get the load value by applying zonal scaling factor defined in Horizon.

Parameters

idx: int, str, or list
Index of the desired load.

attr: str
Attribute name.

model: str
Scaling factor owner, EDTSlot or UCTSlot.

factor: str
Scaling factor name, usually sd.

horizon: int or str
Horizon single index.

RoutineBase.igmake

RoutineBase.**igmake**(*directed=True*)

Build an igraph object from the system.

Parameters

directed: bool
Whether the graph is directed.

Returns

igraph.Graph
An igraph object.

RoutineBase.igraph

RoutineBase.**igraph**(*input: RParam | Var | None = None*, *ytimes: float | None = None*, *decimal: int | None = 6*, *directed: bool | None = True*, *dpi: int | None = 100*, *figsize: tuple | None = None*, *adjust_bus: bool | None = False*, *gen_color: str | None = 'red'*, *rest_color: str | None = 'black'*, *vertex_shape: str | None = 'circle'*, *vertex_font: str | None = None*, *no_vertex_label: bool | None = False*, *vertex_label: str | list | None = None*, *vertex_size: float | None = None*, *vertex_label_size: float | None = None*, *vertex_label_dist: float | None = 1.5*, *vertex_label_angle: float | None = 10.2*, *edge_arrow_size: float | None = None*, *edge_arrow_width: float | None = None*, *edge_width: float | None = None*, *edge_align_label: bool | None = True*, *edge_background: str | None = None*, *edge_color: str | None = None*, *edge_curved: bool | None = False*, *edge_font: str | None = None*, *edge_label: str | list | None = None*, *layout: str | None = 'rt'*, *autocurve: bool | None = True*, *ax: Axes | None = None*, *title: str | None = None*, *title_loc: str | None = None*, ***visual_style*)

Plot a system uging *g.plot()* of *igraph*, with optional input. For now, only support plotting of Bus and Line elements as input.

Parameters

input: RParam or Var, optional
The variable or parameter to be plotted.

ytimes: float, optional
The scaling factor of the values.

directed: bool, optional
Whether the graph is directed.

dpi: int, optional
Dots per inch.

figsize: tuple, optional
Figure size.

adjust_bus: bool, optional
Whether to adjust the bus size.

gen_color: str, optional
Color of the generator bus.

rest_color: str, optional
Color of the rest buses.

no_vertex_label: bool, optional
Whether to show vertex labels.

vertex_shape: str, optional
Shape of the vertices.

vertex_font: str, optional
Font of the vertices.

vertex_size: float, optional
Size of the vertices.

vertex_label_size: float, optional
Size of the vertex labels.

vertex_label_dist: float, optional
Distance of the vertex labels.

vertex_label_angle: float, optional
Angle of the vertex labels.

edge_arrow_size: float, optional
Size of the edge arrows.

edge_arrow_width: float, optional
Width of the edge arrows.

edge_width: float, optional
Width of the edges.

edge_align_label: bool, optional
Whether to align the edge labels.

edge_background: str, optional
RGB colored rectangle background of the edge labels.

layout: str, optional
Layout of the graph, ['rt', 'kk', 'fr', 'drl', 'lgl', 'circle', 'grid_fr'].

autocurve: bool, optional
Whether to use autocurve.

ax: plt.Axes, optional
Matplotlib axes.

visual_style: dict, optional
Visual style, see `igraph.plot` for details.

Returns

plt.Axes
Matplotlib axes.

igraph.Graph
An igraph object.

Examples

```
>>> import ams
>>> sp = ams.load(ams.get_case('5bus/pjm5bus_uced.xlsx'))
>>> sp.DCOPF.run()
>>> sp.DCOPF.plot(input=sp.DCOPF.pn,
>>>               ytimes=10,
>>>               adjust_bus=True,
>>>               vertex_size=10,
>>>               vertex_label_size=15,
>>>               vertex_label_dist=2,
>>>               vertex_label_angle=90,
>>>               show=False,
>>>               edge_align_label=True,
>>>               autocurve=True,)
```

RoutineBase.init

`RoutineBase.init`(*force=False*, *no_code=True*, ***kwargs*)

Initialize the routine.

Force initialization (*force=True*) will do the following: - Rebuild the system matrices - Enable all constraints - Reinitialize the optimization model

Parameters

force: bool
Whether to force initialization.

no_code: bool
Whether to show generated code.

RoutineBase.prepare

`RoutineBase.prepare()`

Prepare the routine.

RoutineBase.run

`RoutineBase.run(force_init=False, no_code=True, **kwargs)`

Run the routine.

Force initialization (*force_init=True*) will do the following: - Rebuild the system matrices - Enable all constraints - Reinitialize the optimization model

Parameters

force_init: bool
Whether to force initialization.

no_code: bool
Whether to show generated code.

RoutineBase.set

`RoutineBase.set(src: str, idx, attr: str = 'v', value=0.0)`

Set the value of an attribute of a routine parameter.

RoutineBase.solve

`RoutineBase.solve(**kwargs)`

Solve the routine optimization model.

RoutineBase.summary

`RoutineBase.summary(**kwargs)`

Summary interface

RoutineBase.unpack

`RoutineBase.unpack(**kwargs)`

Unpack the results.

RoutineBase.update

RoutineBase.**update**(*params=None*, *mat_make=True*)

Update the values of Parameters in the optimization model.

This method is particularly important when some *RParams* are linked with system matrices. In such cases, setting *mat_make=True* is necessary to rebuild these matrices for the changes to take effect. This is common in scenarios involving topology changes, connection statuses, or load value modifications. If unsure, it is advisable to use *mat_make=True* as a precautionary measure.

Parameters

params: Parameter, str, or list
Parameter, Parameter name, or a list of parameter names to be updated. If None, all parameters will be updated.

mat_make: bool
True to rebuild the system matrices. Set to False to speed up the process if no system matrices are changed.

Attributes

class_name

RoutineBase.class_name

property RoutineBase.**class_name**

7.4 Optimization

ams.opt.omodel	Module for optimization modeling.

7.4.1 ams.opt.omodel

Module for optimization modeling.

Classes

Constraint([name, e_str, info, type])	Base class for constraints.
OModel(routine)	Base class for optimization models.
Objective([name, e_str, info, unit, sense])	Base class for objective functions.
OptzBase([name, info, unit])	Base class for optimization elements, e.g., Var and Constraint.
Param([name, info, unit, no_parse, nonneg, ...])	Base class for parameters used in a routine.
Var([name, tex_name, info, src, unit, ...])	Base class for variables used in a routine.

ams.opt.omodel.Constraint

class ams.opt.omodel.**Constraint**(*name: str | None = None*, *e_str: str | None = None*, *info: str | None = None*, *type: str | None = 'uq'*)

Base class for constraints.

This class is used as a template for defining constraints. Each instance of this class represents a single constraint.

Parameters

name
[str, optional] A user-defined name for the constraint.

e_str
[str, optional] A mathematical expression representing the constraint.

info
[str, optional] Additional informational text about the constraint.

type
[str, optional] The type of constraint, which determines the mathematical relationship. Possible values include 'uq' (inequality, default) and 'eq' (equality).

Attributes

is_disabled
[bool] Flag indicating if the constraint is disabled, False by default.

rtn
[ams.routines.Routine] The owner routine instance.

__init__(*name: str | None = None*, *e_str: str | None = None*, *info: str | None = None*, *type: str | None = 'uq'*)

Methods

parse([no_code])	Parse the constraint.

Constraint.parse

Constraint.**parse**(*no_code=True*)

Parse the constraint.

Parameters

no_code
[bool, optional] Flag indicating if the code should be shown, True by default.

Attributes

class_name	Return the class name
n	Return the number of elements.
shape	Return the shape.
size	Return the size.
v	Return the CVXPY constraint LHS value.
v2	Return the calculated constraint LHS value.

Constraint.class_name

property `Constraint.`**`class_name`**

Return the class name

Constraint.n

property `Constraint.`**`n`**

Return the number of elements.

Constraint.shape

property `Constraint.`**`shape`**

Return the shape.

Constraint.size

property `Constraint.`**`size`**

Return the size.

Constraint.v

property `Constraint.`**`v`**

Return the CVXPY constraint LHS value.

Constraint.v2

property `Constraint.`**`v2`**

Return the calculated constraint LHS value. Note that `v` should be used primarily as it is obtained from the solver directly. `v2` is for debugging purpose, and should be consistent with `v`.

ams.opt.omodel.OModel

class ams.opt.omodel.**OModel**(*routine*)

Base class for optimization models.

Parameters

routine: Routine
Routine that to be modeled.

Attributes

prob: cvxpy.Problem
Optimization model.

params: OrderedDict
Parameters.

vars: OrderedDict
Decision variables.

constrs: OrderedDict
Constraints.

obj: Objective
Objective function.

__init__(*routine*)

Methods

init([no_code])	Set up the optimization model from the symbolic description.
update(params)	Update the Parameter values.

OModel.init

OModel.**init**(*no_code=True*)

Set up the optimization model from the symbolic description.

This method initializes the optimization model by parsing decision variables, constraints, and the objective function from the associated routine.

Parameters

no_code
[bool, optional] Flag indicating if the parsing code should be displayed, True by default.

Returns

bool
Returns True if the setup is successful, False otherwise.

OModel.update

OModel.**update**(*params*)

Update the Parameter values.

Parameters

params: list
List of parameters to be updated.

Attributes

class_name	Return the class name

OModel.class_name

property OModel.**class_name**

Return the class name

ams.opt.omodel.Objective

class ams.opt.omodel.**Objective**(*name: str | None = None, e_str: str | None = None, info: str | None = None, unit: str | None = None, sense: str | None = 'min'*)

Base class for objective functions.

This class serves as a template for defining objective functions. Each instance of this class represents a single objective function that can be minimized or maximized depending on the sense ('min' or 'max').

Parameters

name
[str, optional] A user-defined name for the objective function.

e_str
[str, optional] A mathematical expression representing the objective function.

info
[str, optional] Additional informational text about the objective function.

sense
[str, optional] The sense of the objective function, default to 'min'. *min* for minimization and *max* for maximization.

Attributes

v
[NoneType] Return the CVXPY objective value.

rtn
[ams.routines.Routine] The owner routine instance.

__init__(*name: str | None = None, e_str: str | None = None, info: str | None = None, unit: str | None = None, sense: str | None = 'min'*)

Methods

parse([no_code])	Parse the objective function.

Objective.parse

`Objective.parse(no_code=True)`

Parse the objective function.

Parameters

no_code
[bool, optional] Flag indicating if the code should be shown, True by default.

Attributes

class_name	Return the class name
n	Return the number of elements.
shape	Return the shape.
size	Return the size.
v	Return the CVXPY objective value.
v2	Return the calculated objective value.

Objective.class_name

`property Objective.class_name`

Return the class name

Objective.n

`property Objective.n`

Return the number of elements.

Objective.shape

`property Objective.shape`

Return the shape.

Objective.size

property Objective.**size**

Return the size.

Objective.v

property Objective.**v**

Return the CVXPY objective value.

Objective.v2

property Objective.**v2**

Return the calculated objective value. Note that v should be used primarily as it is obtained from the solver directly. v2 is for debugging purpose, and should be consistent with v.

ams.opt.omodel.OptzBase

class ams.opt.omodel.**OptzBase**(*name: str | None = None*, *info: str | None = None*, *unit: str | None = None*)

Base class for optimization elements, e.g., Var and Constraint.

Parameters

name
[str, optional] Name.

info
[str, optional] Descriptive information

Attributes

rtn
[ams.routines.Routine] The owner routine instance.

__init__(*name: str | None = None*, *info: str | None = None*, *unit: str | None = None*)

Methods

parse()	Parse the object.

OptzBase.parse

OptzBase.**parse**()

Parse the object.

Attributes

class_name	Return the class name
n	Return the number of elements.
shape	Return the shape.
size	Return the size.

OptzBase.class_name

property OptzBase.**class_name**

Return the class name

OptzBase.n

property OptzBase.**n**

Return the number of elements.

OptzBase.shape

property OptzBase.**shape**

Return the shape.

OptzBase.size

property OptzBase.**size**

Return the size.

ams.opt.omodel.Param

class ams.opt.omodel.**Param**(*name: str | None = None, info: str | None = None, unit: str | None = None, no_parse: bool | None = False, nonneg: bool | None = False, nonpos: bool | None = False, complex: bool | None = False, imag: bool | None = False, symmetric: bool | None = False, diag: bool | None = False, hermitian: bool | None = False, boolean: bool | None = False, integer: bool | None = False, pos: bool | None = False, neg: bool | None = False, sparse: list | None = False*)

Base class for parameters used in a routine.

Parameters

no_parse: bool, optional
True to skip parsing the parameter.

nonneg: bool, optional
True to set the parameter as non-negative.

nonpos: bool, optional
True to set the parameter as non-positive.

complex: bool, optional
True to set the parameter as complex.

imag: bool, optional
True to set the parameter as imaginary.

symmetric: bool, optional
True to set the parameter as symmetric.

diag: bool, optional
True to set the parameter as diagonal.

hermitian: bool, optional
True to set the parameter as hermitian.

boolean: bool, optional
True to set the parameter as boolean.

integer: bool, optional
True to set the parameter as integer.

pos: bool, optional
True to set the parameter as positive.

neg: bool, optional
True to set the parameter as negative.

sparse: bool, optional
True to set the parameter as sparse.

`__init__`(*name: str | None = None, info: str | None = None, unit: str | None = None, no_parse: bool | None = False, nonneg: bool | None = False, nonpos: bool | None = False, complex: bool | None = False, imag: bool | None = False, symmetric: bool | None = False, diag: bool | None = False, hermitian: bool | None = False, boolean: bool | None = False, integer: bool | None = False, pos: bool | None = False, neg: bool | None = False, sparse: list | None = False*)

Methods

`parse`()	Parse the parameter.
`update`()	Update the Parameter value.

Param.parse

`Param.parse()`

Parse the parameter.

Param.update

Param.**update**()

Update the Parameter value.

Attributes

class_name	Return the class name
n	Return the number of elements.
shape	Return the shape.
size	Return the size.

Param.class_name

property Param.**class_name**

Return the class name

Param.n

property Param.**n**

Return the number of elements.

Param.shape

property Param.**shape**

Return the shape.

Param.size

property Param.**size**

Return the size.

ams.opt.omodel.Var

class ams.opt.omodel.**Var**(*name: str | None = None, tex_name: str | None = None, info: str | None = None, src: str | None = None, unit: str | None = None, model: str | None = None, shape: tuple | int | None = None, v0: str | None = None, horizon=None, nonneg: bool | None = False, nonpos: bool | None = False, complex: bool | None = False, imag: bool | None = False, symmetric: bool | None = False, diag: bool | None = False, psd: bool | None = False, nsd: bool | None = False, hermitian: bool | None = False, boolean: bool | None = False, integer: bool | None = False, pos: bool | None = False, neg: bool | None = False*)

Base class for variables used in a routine.

When *horizon* is provided, the variable will be expanded to a matrix, where rows are indexed by the source variable index and columns are indexed by the horizon index.

Parameters

info
[str, optional] Descriptive information

unit
[str, optional] Unit

tex_name
[str] LaTeX-formatted variable symbol. Defaults to the value of `name`.

name
[str, optional] Variable name. One should typically assigning the name directly because it will be automatically assigned by the model. The value of `name` will be the symbol name to be used in expressions.

src
[str, optional] Source variable name. Defaults to the value of `name`.

model
[str, optional] Name of the owner model or group.

horizon
[ams.routines.RParam, optional] Horizon idx.

nonneg
[bool, optional] Non-negative variable

nonpos
[bool, optional] Non-positive variable

complex
[bool, optional] Complex variable

imag
[bool, optional] Imaginary variable

symmetric
[bool, optional] Symmetric variable

diag
[bool, optional] Diagonal variable

psd
[bool, optional] Positive semi-definite variable

nsd
[bool, optional] Negative semi-definite variable

hermitian
[bool, optional] Hermitian variable

boolean
[bool, optional] Boolean variable

integer
[bool, optional] Integer variable

pos
[bool, optional] Positive variable

neg
[bool, optional] Negative variable

Attributes

a
[np.ndarray] Variable address.

_v
[np.ndarray] Local-storage of the variable value.

rtn
[ams.routines.Routine] The owner routine instance.

__init__(*name: str | None = None, tex_name: str | None = None, info: str | None = None, src: str | None = None, unit: str | None = None, model: str | None = None, shape: tuple | int | None = None, v0: str | None = None, horizon=None, nonneg: bool | None = False, nonpos: bool | None = False, complex: bool | None = False, imag: bool | None = False, symmetric: bool | None = False, diag: bool | None = False, psd: bool | None = False, nsd: bool | None = False, hermitian: bool | None = False, boolean: bool | None = False, integer: bool | None = False, pos: bool | None = False, neg: bool | None = False*)

Methods

`get_idx`()	
`parse`()	Parse the variable.

Var.get_idx

`Var.get_idx()`

Var.parse

`Var.parse()`

Parse the variable.

Attributes

`class_name`	Return the class name
`n`	Return the number of elements.
`shape`	Return the shape.
`size`	Return the size.
`v`	Return the CVXPY variable value.

Var.class_name

property `Var.class_name`

Return the class name

Var.n

property `Var.n`

Return the number of elements.

Var.shape

property `Var.shape`

Return the shape.

Var.size

property `Var.size`

Return the size.

Var.v

property `Var.v`

Return the CVXPY variable value.

7.5 I/O

ams.io	AMS input parsers and output formatters.

7.5.1 ams.io

AMS input parsers and output formatters.

Functions

guess(system)	Guess the input format based on extension and content.
parse(system)	Parse input file with the given format in *system.files.input_format*.

guess

ams.io.**guess**(*system*)

Guess the input format based on extension and content.

Also stores the format name to *system.files.input_format*.

Parameters

system
[System] System instance with the file name set to *system.files*

Returns

str
format name

parse

ams.io.**parse**(*system*)

Parse input file with the given format in *system.files.input_format*.

Returns

bool
True if successful; False otherwise.

Modules

ams.io.json	Json reader and writer for AMS.
ams.io.matpower	MATPOWER parser.
ams.io.psse	Excel reader and writer for AMS.
ams.io.pypower	PYPOWER reader for AMS.
ams.io.xlsx	Excel reader and writer for AMS.

ams.io.json

Json reader and writer for AMS.

This module leverages the existing parser and writer in andes.io.json.

Functions

write(system, outfile[, skip_empty, ...])	Write loaded AMS system data into an json file.

write

ams.io.json.**write**(*system*, *outfile*, *skip_empty=True*, *overwrite=None*, *to_andes=False*)

Write loaded AMS system data into an json file. If to_andes is True, only write models that are in ANDES, but the outfile might not be able to be read back into AMS.

Revise function `andes.io.json.write` to skip non-andes models.

Parameters

system
[System] A loaded system with parameters

outfile
[str] Path to the output file

skip_empty
[bool] Skip output of empty models (n = 0)

overwrite
[bool, optional] None to prompt for overwrite selection; True to overwrite; False to not overwrite

to_andes
[bool, optional] Write to an ANDES system, where non-ANDES models are skipped

Returns

bool
True if file written; False otherwise

ams.io.matpower

MATPOWER parser. This module is revised from the existing module `andes.io.matpower`.

Functions

mpc2system(mpc, system)	Load an mpc dict into an empty AMS system.
read(system, file)	Read a MATPOWER data file into mpc, and build andes device elements.
system2mpc(system)	Convert data from an AMS system to an mpc dict.
testlines(infile)	Test if this file is in the MATPOWER format.

mpc2system

ams.io.matpower.**mpc2system**(*mpc: dict*, *system*) → bool

Load an mpc dict into an empty AMS system.

This function is revised from `andes.io.matpower.mpc2system`.

Compared to the original one, this function includes the generator cost data.

Parameters

system
[andes.system.System] Empty system to load the data into.

mpc
[dict] mpc struct names : numpy arrays

Returns

bool
True if successful, False otherwise.

read

`ams.io.matpower.read(system, file)`

Read a MATPOWER data file into mpc, and build andes device elements.

system2mpc

`ams.io.matpower.system2mpc(system)` → dict

Convert data from an AMS system to an mpc dict.

In the `gen` section, slack generators preceeds PV generators.

Compared to the `andes.io.matpower.system2mpc`, this function includes the generator cost data in the `gencost` section. Additionally, `c2` and `c1` are scaled by `base_mva` to match MATPOWER unit `MW`.

Parameters

system
[ams.core.system.System] AMS system

Returns

mpc: dict
MATPOWER mpc dict

testlines

`ams.io.matpower.testlines(infile)`

Test if this file is in the MATPOWER format.

NOT YET IMPLEMENTED.

ams.io.psse

Excel reader and writer for AMS. This module is the existing module in `andes.io.psse`.

ams.io.pypower

PYPOWER reader for AMS.

Functions

ppc2system(ppc, system)	Alias for `mpc2system`.
py2ppc(infile)	Parse PYPOWER file and return a dictionary with the data.
read(system, file)	Read a PYPOWER case file into ppc and return an AMS system by calling `ppc2system`.
system2ppc(system)	Alias for `system2mpc`.
testlines(infile)	Test if this file is in the PYPOWER format.

ppc2system

`ams.io.pypower.`**`ppc2system`**(*ppc: dict*, *system*) → bool

Alias for `mpc2system`. Refer to *ams.io.matpower.mpc2system* for more details.

Load an PYPOWER case dict into an empth AMS system.

Parameters

ppc
[dict] The PYPOWER case dict.

system
[ams.system] Empty AMS system to load data into.

Returns

bool
True if successful; False otherwise.

py2ppc

`ams.io.pypower.`**`py2ppc`**(*infile: str*) → dict

Parse PYPOWER file and return a dictionary with the data.

Parameters

infile
[str] The path to the PYPOWER file.

Returns

ppc
[dict] The PYPOWER case dict.

read

`ams.io.pypower.read(system, file)`

Read a PYPOWER case file into ppc and return an AMS system by calling `ppc2system`.

Parameters

system
[ams.system] Empty AMS system to load data into.

file
[str] The path to the PYPOWER file.

Returns

system
[ams.system.System] The AMS system that loaded the data.

system2ppc

`ams.io.pypower.system2ppc(system) → dict`

Alias for `system2mpc`. Refer to `ams.io.matpower.system2mpc` for more details.

Convert data from an AMS system to an mpc dict.

In the `gen` section, slack generators preceeds PV generators.

testlines

`ams.io.pypower.testlines(infile)`

Test if this file is in the PYPOWER format.

NOT YET IMPLEMENTED.

ams.io.xlsx

Excel reader and writer for AMS.

This module leverages the existing parser and writer in andes.io.xlsx.

Functions

`write`(system, outfile[, skip_empty, ...])	Write loaded AMS system data into an xlsx file

write

ams.io.xlsx.**write**(*system*, *outfile*, *skip_empty=True*, *overwrite=None*, *add_book=None*, *to_andes=False*)

Write loaded AMS system data into an xlsx file

Revised function `andes.io.xlsx.write` to skip non-andes models.

Parameters

system
[System] A loaded system with parameters

outfile
[str] Path to the output file

skip_empty
[bool] Skip output of empty models (n = 0)

overwrite
[bool, optional] None to prompt for overwrite selection; True to overwrite; False to not overwrite

add_book
[str, optional] An optional model to be added to the output spreadsheet

to_andes
[bool, optional] Write to an ANDES system, where non-ANDES models are skipped

Returns

bool
True if file written; False otherwise

7.6 Interoperability

ams.interop	Interopability package between AMS and other software.

7.6.1 ams.interop

Interopability package between AMS and other software.

To install dependencies, do:

```
pip install ams[interop]
```

To install dependencies for *development*, in the AMS source code folder, do:

```
pip install -e .[interop]
```

Modules

ams.interop.andes	Interface with ANDES

ams.interop.andes

Interface with ANDES

Functions

build_group_table(adsys, grp_name, param_name)	Build the table for devices in a group in an ANDES System.
make_link_table(adsys)	Build the link table for generators and generator controllers in an ANDES System, including `SynGen` and `DG` for now.
parse_addfile(adsys, amsys, addfile)	Parse the addfile for ANDES dynamic file.
to_andes(system[, setup, addfile])	Convert the AMS system to an ANDES system.

build_group_table

`ams.interop.andes.`**`build_group_table`**(*adsys*, *grp_name*, *param_name*, *mdl_name=None*)

Build the table for devices in a group in an ANDES System.

Parameters

adsys
[andes.system.System] The ANDES system to build the table

grp_name
[string] The ANDES group

param_name
[list of string] The common columns of a group that to be included in the table.

mdl_name
[list of string] The list of models that to be included in the table. Default as all models.

Returns

DataFrame
The output Dataframe contains the columns from the device

make_link_table

ams.interop.andes.**make_link_table**(*adsys*)

Build the link table for generators and generator controllers in an ANDES System, including `SynGen` and `DG` for now.

Parameters

adsys
[andes.system.System] The ANDES system to link

Returns

DataFrame
Each column in the output Dataframe contains the `idx` of linked `StaticGen`, `Bus`, `DG`, `RenGen`, `RenExciter`, `SynGen`, `Exciter`, and `TurbineGov`, `gammap`, `gammaq`.

parse_addfile

ams.interop.andes.**parse_addfile**(*adsys*, *amsys*, *addfile*)

Parse the addfile for ANDES dynamic file.

Parameters

adsys
[andes.system.System] The ANDES system instance.

amsys
[ams.system.System] The AMS system instance.

addfile
[str] The additional file to be converted to ANDES dynamic mdoels.

Returns

adsys
[andes.system.System] The ANDES system instance with dynamic models added.

to_andes

ams.interop.andes.**to_andes**(*system*, *setup=False*, *addfile=None*, ***kwargs*)

Convert the AMS system to an ANDES system.

A preferred dynamic system file to be added has following features: 1. The file contains both power flow and dynamic models. 2. The file can run in ANDES natively. 3. Power flow models are in the same shape as the AMS system. 4. Dynamic models, if any, are in the same shape as the AMS system.

This function is wrapped as the `System` class method `to_andes()`. Using the file conversion `to_andes()` will automatically link the AMS system instance to the converted ANDES system instance in the AMS system attribute `dyn`.

It should be noted that detailed dynamic simualtion requires extra dynamic models to be added to the ANDES system, which can be passed through the `addfile` argument.

Parameters

system
[System] The AMS system to be converted to ANDES format.

setup
[bool, optional] Whether to call *setup()* after the conversion. Default is True.

addfile
[str, optional] The additional file to be converted to ANDES dynamic mdoels.

****kwargs**
[dict] Keyword arguments to be passed to *andes.system.System*.

Returns

adsys
[andes.system.System] The converted ANDES system.

Notes

1. Power flow models in the addfile will be skipped and only dynamic models will be used.
2. The addfile format is guessed based on the file extension. Currently only `xlsx` is supported.
3. Index in the addfile is automatically adjusted when necessary.

Examples

```
>>> import ams
>>> import andes
>>> sp = ams.load(ams.get_case('ieee14/ieee14_uced.xlsx'), setup=True)
>>> sa = sp.to_andes(setup=False,
...                  addfile=andes.get_case('ieee14/ieee14_full.xlsx'),
...                  overwrite=True, no_output=True)
```

Classes

Dynamic([amsys, adsys])	ANDES interface class.

ams.interop.andes.Dynamic

class ams.interop.andes.**Dynamic**(*amsys=None*, *adsys=None*)
ANDES interface class.

Parameters

amsys
[AMS.system.System] The AMS system.

adsys
[ANDES.system.System] The ANDES system.

Notes

1. Using the file conversion `to_andes()` will automatically link the AMS system to the converted ANDES system in the attribute `dyn`.

Examples

```
>>> import ams
>>> import andes
>>> sp = ams.load(ams.get_case('ieee14/ieee14_rted.xlsx'), setup=True)
>>> sa = sp.to_andes(setup=True,
...                  addfile=andes.get_case('ieee14/ieee14_wt3.xlsx'),
...                  overwrite=True, keep=False, no_output=True)
>>> sp.RTED.run()
>>> sp.RTED.dc2ac()
>>> sp.dyn.send()  # send RTED results to ANDES system
>>> sa.PFlow.run()
>>> sp.TDS.run()
>>> sp.dyn.receive()  # receive TDS results from ANDES system
```

Attributes

link
[pandas.DataFrame] The ANDES system link table.

`__init__`(*amsys=None*, *adsys=None*) → None

Methods

link_andes(adsys)	Link the ANDES system to the AMS system.
receive([adsys, routine, no_update])	Receive ANDES system results to AMS devices.
send([adsys, routine])	Send results of the recent sovled AMS dispatch (`sp.recent`) to the target ANDES system.

Dynamic.link_andes

`Dynamic.`**`link_andes`**(*adsys*)

Link the ANDES system to the AMS system.

Parameters

adsys
[ANDES.system.System] The ANDES system instance.

Dynamic.receive

`Dynamic.receive(adsys=None, routine=None, no_update=False)`

Receive ANDES system results to AMS devices.

Parameters

adsys
[adsys.System.system, optional] The target ANDES dynamic system instance. If not provided, use the linked ANDES system isntance (`sp.dyn.adsys`).

routine
[str, optional] The routine to be received from ANDES. If None, `recent` will be used.

no_update
[bool, optional] True to skip update the AMS routine parameters after sync. Default is False.

Dynamic.send

`Dynamic.send(adsys=None, routine=None)`

Send results of the recent sovled AMS dispatch (`sp.recent`) to the target ANDES system.

Note that converged AC conversion DOES NOT guarantee successful dynamic initialization `TDS.init()`. Failed initialization is usually caused by limiter violation.

Parameters

adsys
[adsys.System.system, optional] The target ANDES dynamic system instance. If not provided, use the linked ANDES system isntance (`sp.dyn.adsys`).

routine
[str, optional] The routine to be sent to ANDES. If None, `recent` will be used.

Attributes

is_tds	Indicator of whether the ANDES system is running a TDS.

Dynamic.is_tds

`property Dynamic.is_tds`

Indicator of whether the ANDES system is running a TDS. This property will return `True` as long as TDS is initialized.

Check `adsys.tds.TDS.init()` for more details.

7.7 Others

ams.cli	AMS command-line interface and argument parsers.
ams.main	Main entry point for the AMS CLI and scripting interfaces.
ams.utils.paths	Utility functions for loading ams stock test cases, mainly revised from `andes.utils.paths`.

7.7.1 ams.cli

AMS command-line interface and argument parsers.

Functions

create_parser()	Create a parser for the command-line interface.
main()	Entry point of the ANDES command-line interface.
preamble()	Log the AMS command-line preamble at the *logging.INFO* level

create_parser

`ams.cli.create_parser()`

Create a parser for the command-line interface.

Returns

argparse.ArgumentParser
Parser with all AMS options

main

`ams.cli.main()`

Entry point of the ANDES command-line interface.

preamble

`ams.cli.preamble()`

Log the AMS command-line preamble at the *logging.INFO* level

7.7.2 ams.main

Main entry point for the AMS CLI and scripting interfaces.

Functions

config_logger([stream_level, stream, file, ...])	Configure an AMS logger with a *FileHandler* and a *StreamHandler*.
demo(**kwargs)	TODO: show some demonstrations from CLI.
doc([attribute, list_supported, config])	Quick documentation from command-line.
edit_conf([edit_config])	Edit the Andes config file which occurs first in the search path.
find_log_path(lg)	Find the file paths of the FileHandlers.
load(case[, setup, use_input_path])	Load a case and set up a system without running routine.
misc([edit_config, save_config, ...])	Miscellaneous commands.
print_license()	Print out AMS license to stdout.
remove_output([recursive])	Remove the outputs generated by Andes, including power flow reports `_out.txt`, time-domain list `_out.lst` and data `_out.dat`, eigenvalue analysis report `_eig.txt`.
run(filename[, input_path, verbose, ...])	Entry point to run ANDES routines.
run_case(case, *[, routine, profile, ...])	Run single simulation case for the given full path.
save_conf([config_path, overwrite])	Save the AMS config to a file at the path specified by `save_config`.
selftest([quick, extra])	Run unit tests.
set_logger_level(lg, type_to_set, level)	Set logging level for the given type of handler.
versioninfo()	Print version info for ANDES and dependencies.

config_logger

ams.main.**config_logger**(*stream_level=20*, ***, *stream=True*, *file=True*, *log_file='ams.log'*, *log_path=None*, *file_level=10*)

Configure an AMS logger with a *FileHandler* and a *StreamHandler*.

This function is called at the beginning of `ams.main.main()`. Updating `stream_level` and `file_level` is now supported.

Parameters

stream
[bool, optional] Create a *StreamHandler* for *stdout* if `True`. If `False`, the handler will not be created.

file
[bool, optionsl] True if logging to `log_file`.

log_file
[str, optional] Logg file name for *FileHandler*, `'ams.log'` by default. If `None`, the *FileHandler* will not be created.

log_path
[str, optional] Path to store the log file. By default, the path is generated by get_log_dir() in utils.misc.

stream_level
[{10, 20, 30, 40, 50}, optional] *StreamHandler* verbosity level.

file_level
[{10, 20, 30, 40, 50}, optional] *FileHandler* verbosity level.

Returns

None

demo

`ams.main.demo(**kwargs)`

TODO: show some demonstrations from CLI.

doc

`ams.main.doc(attribute=None, list_supported=False, config=False, **kwargs)`

Quick documentation from command-line.

edit_conf

`ams.main.edit_conf(edit_config: str | bool | None = '')`

Edit the Andes config file which occurs first in the search path.

Parameters

edit_config
[bool] If `True`, try to open up an editor and edit the config file. Otherwise returns.

Returns

bool
`True` is a config file is found and an editor is opened. `False` if `edit_config` is False.

find_log_path

`ams.main.find_log_path(lg)`

Find the file paths of the FileHandlers.

load

`ams.main.load(case, setup=True, use_input_path=True, **kwargs)`

Load a case and set up a system without running routine. Return a system.

Takes other kwargs recognizable by `System`, such as `addfile`, `input_path`, and `no_putput`.

Parameters

case: str
Path to the test case

setup
[bool, optional] Call *System.setup* after loading

use_input_path
[bool, optional] True to use the `input_path` argument to behave the same as `ams.main.run`.

> **Warning:** If one need to add devices in addition to these from the case file, do `setup=False` and call `System.add()` to add devices. When done, manually invoke `setup()` to set up the system.

misc

`ams.main.`**`misc`**(*edit_config=''*, *save_config=''*, *show_license=False*, *clean=True*, *recursive=False*, *overwrite=None*, *version=False*, ***kwargs*)

Miscellaneous commands.

print_license

`ams.main.`**`print_license`**()

Print out AMS license to stdout.

remove_output

`ams.main.`**`remove_output`**(*recursive=False*)

Remove the outputs generated by Andes, including power flow reports `_out.txt`, time-domain list `_out.lst` and data `_out.dat`, eigenvalue analysis report `_eig.txt`.

Parameters

recursive
[bool] Recursively clean all subfolders

Returns

bool
`True` is the function body executes with success. `False` otherwise.

run

`ams.main.`**`run`**(*filename*, *input_path=''*, *verbose=20*, *mp_verbose=30*, *ncpu=1*, *pool=False*, *cli=False*, *shell=False*, ***kwargs*)

Entry point to run ANDES routines.

Parameters

filename
[str] file name (or pattern)

input_path
[str, optional] input search path

verbose
[int, 10 (DEBUG), 20 (INFO), 30 (WARNING), 40 (ERROR), 50 (CRITICAL)] Verbosity level. If `config_logger` is called prior to `run`, this option will be ignored.

mp_verbose
[int] Verbosity level for multiprocessing tasks

ncpu
[int, optional] Number of cpu cores to use in parallel

pool: bool, optional
Use Pool for multiprocessing to return a list of created Systems.

kwargs
Other supported keyword arguments

cli
[bool, optional] If is running from command-line. If True, returns exit code instead of System

shell
[bool, optional] If True, enter IPython shell after routine.

Returns

System or exit_code
An instance of system (if *cli* == *False*) or an exit code otherwise..

run_case

`ams.main.run_case`(*case*, *, *routine='pflow'*, *profile=False*, *convert=''*, *convert_all=''*, *add_book=None*, ***kwargs*)

Run single simulation case for the given full path. Use `run` instead of `run_case` whenever possible.

Argument `input_path` will not be prepended to `case`.

Arguments recognizable by `load` can be passed to `run_case`.

Parameters

case
[str] Full path to the test case

routine
[str, ('pflow', 'tds', 'eig')] Computation routine to run

profile
[bool, optional] True to enable profiler

convert
[str, optional] Format name for case file conversion.

convert_all
[str, optional] Format name for case file conversion, output sheets for all available devices.

add_book
[str, optional] Name of the device to be added to an excel case as a new sheet.

save_conf

ams.main.**save_conf**(*config_path=None*, *overwrite=None*, ***kwargs*)

Save the AMS config to a file at the path specified by `save_config`. The save action will not run if `save_config = ''`.

Parameters

config_path
[None or str, optional, ('' by default)] Path to the file to save the config file. If the path is an emtpy string, the save action will not run. Save to *~/.ams/ams.conf* if `None`.

Returns

bool
`True` is the save action is run. `False` otherwise.

selftest

ams.main.**selftest**(*quick=False*, *extra=False*, ***kwargs*)

Run unit tests.

set_logger_level

ams.main.**set_logger_level**(*lg*, *type_to_set*, *level*)

Set logging level for the given type of handler.

versioninfo

ams.main.**versioninfo**()

Print version info for ANDES and dependencies.

7.7.3 ams.utils.paths

Utility functions for loading ams stock test cases, mainly revised from `andes.utils.paths`.

Functions

ams_root()	Return the root path to the ams source code.
cases_root()	Return the root path to the stock cases
confirm_overwrite(outfile[, overwrite])	Confirm overwriting a file.
get_case(rpath[, check])	Return the path to a stock case for a given path relative to `ams/cases`.
get_config_path([file_name])	Return the path of the config file to be loaded.
get_dot_andes_path()	Return the path to `$HOME/.ams`
get_log_dir()	Get the directory for log file.
get_pkl_path()	Get the path to the picked/dilled function calls.
get_pycode_path([pycode_path, mkdir])	Get the path to the `pycode` folder.
list_cases([rpath, no_print])	List stock cases under a given folder relative to `ams/cases`
tests_root()	Return the root path to the stock cases

ams_root

`ams.utils.paths.ams_root()`

Return the root path to the ams source code.

cases_root

`ams.utils.paths.cases_root()`

Return the root path to the stock cases

confirm_overwrite

`ams.utils.paths.confirm_overwrite(outfile, overwrite=None)`

Confirm overwriting a file.

get_case

`ams.utils.paths.get_case(rpath, check=True)`

Return the path to a stock case for a given path relative to `ams/cases`.

To list all cases, use `ams.list_cases()`.

Parameters

check
[bool] True to check if file exists

Examples

To get the path to the case *kundur_full.xlsx* under folder *kundur*, do

```
ams.get_case('kundur/kundur_full.xlsx')
```

get_config_path

`ams.utils.paths.get_config_path(file_name='ams.rc')`

Return the path of the config file to be loaded.

Search Priority: 1. current directory; 2. home directory.

Parameters

file_name
[str, optional] Config file name with the default as `ams.rc`.

Returns

Config path in string if found; None otherwise.

get_dot_andes_path

`ams.utils.paths.get_dot_andes_path()`

Return the path to `$HOME/.ams`

get_log_dir

`ams.utils.paths.get_log_dir()`

Get the directory for log file.

The default is `<tempdir>/ams`, where `<tempdir>` is provided by `tempfile.gettempdir()`.

Returns

str
The path to the temporary logging directory

get_pkl_path

`ams.utils.paths.get_pkl_path()`

Get the path to the picked/dilled function calls.

Returns

str
Path to the calls.pkl file

get_pycode_path

`ams.utils.paths.get_pycode_path(pycode_path=None, mkdir=False)`

Get the path to the `pycode` folder.

list_cases

`ams.utils.paths.list_cases(rpath='.', no_print=False)`

List stock cases under a given folder relative to `ams/cases`

tests_root

`ams.utils.paths.tests_root()`

Return the root path to the stock cases

Classes

DisplayablePath(path, parent_path, is_last)	

ams.utils.paths.DisplayablePath

class ams.utils.paths.**DisplayablePath**(*path*, *parent_path*, *is_last*)

__init__(*path*, *parent_path*, *is_last*)

Methods

displayable()	
make_tree(root[, parent, is_last, criteria])	

DisplayablePath.displayable

DisplayablePath.**displayable**()

DisplayablePath.make_tree

classmethod DisplayablePath.**make_tree**(*root*, *parent=None*, *is_last=False*, *criteria=None*)

Attributes

display_filename_prefix_last
display_filename_prefix_middle
display_parent_prefix_last
display_parent_prefix_middle
displayname

DisplayablePath.display_filename_prefix_last

DisplayablePath.**display_filename_prefix_last** = '└──'

DisplayablePath.display_filename_prefix_middle

DisplayablePath.**display_filename_prefix_middle** = '├──'

DisplayablePath.display_parent_prefix_last

DisplayablePath.**display_parent_prefix_last** = '│ '

DisplayablePath.display_parent_prefix_middle

DisplayablePath.**display_parent_prefix_middle** = ' '

DisplayablePath.displayname

property DisplayablePath.**displayname**

BIBLIOGRAPHY

[Cui2021] H. Cui, F. Li and K. Tomsovic, "Hybrid Symbolic-Numeric Framework for Power System Modeling and Analysis," in IEEE Transactions on Power Systems, vol. 36, no. 2, pp. 1373-1384, March 2021, doi: 10.1109/TPWRS.2020.3017019.

[PJM5] F. Li and R. Bo, "Small test systems for power system economic studies," IEEE PES General Meeting, 2010, pp. 1-4, doi: 10.1109/PES.2010.5589973.

[IEEE] University of Washington, "Power Systems Test Case Archive", [Online]. Available: https://labs.ece.uw.edu/pstca/

[TSG] X. Wang, F. Li, Q. Zhang, Q. Shi and J. Wang, "Profit-Oriented BESS Siting and Sizing in Deregulated Distribution Systems," in IEEE Transactions on Smart Grid, vol. 14, no. 2, pp. 1528-1540, March 2023, doi: 10.1109/TSG.2022.3150768.

[MATPOWER] R. D. Zimmerman, "MATPOWER", [Online]. Available: https://matpower.org/

[SciData] Q. Zhang and F. Li, “A Dataset for Electricity Market Studies on Western and Northeastern Power Grids in the United States,” Scientific Data, vol. 10, no. 1, p. 646, Sep. 2023, doi: 10.1038/s41597-023-02448-w.

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V

W

Z