

---

# **ANDES Manual**

***Release 1.2.0***

**Hantao Cui**

**Oct 11, 2020**



<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Environment . . . . .	3
1.1.1	Setting Up Miniconda . . . . .	3
1.1.2	Existing Python Environment (Advanced) . . . . .	4
1.2	Install ANDES . . . . .	4
1.2.1	User Mode . . . . .	4
1.2.2	Development Mode . . . . .	4
1.3	Updating ANDES . . . . .	5
1.4	Performance Packages (Advanced) . . . . .	6
1.4.1	cxvoptklu . . . . .	6
1.4.2	CVXOPT with ipadd . . . . .	6
<b>2</b>	<b>Tutorial</b>	<b>7</b>
2.1	Command Line Usage . . . . .	7
2.1.1	Basic Usage . . . . .	7
2.1.2	andes selftest . . . . .	8
2.1.3	andes prepare . . . . .	9
2.1.4	andes run . . . . .	9
2.1.5	andes plot . . . . .	13
2.1.6	andes doc . . . . .	14
2.1.7	andes misc . . . . .	16
2.2	Interactive Usage . . . . .	16
2.2.1	Jupyter Notebook . . . . .	17
2.2.2	Import . . . . .	17
2.2.3	Verbosity . . . . .	17
2.2.4	Making a System . . . . .	17
2.2.5	Inspecting Parameter . . . . .	18
2.2.6	Running Studies . . . . .	19
2.2.7	Checking Exit Code . . . . .	19
2.2.8	Plotting TDS Results . . . . .	19
2.2.9	Extracting Data . . . . .	20
2.2.10	Pretty Print of Equations . . . . .	22

2.2.11	Finding Help . . . . .	23
2.3	Notebook Examples . . . . .	23
2.4	I/O Formats . . . . .	23
2.4.1	Input Formats . . . . .	23
2.4.2	ANDES xlsx Format . . . . .	24
2.5	Cheatsheet . . . . .	26
2.6	Make Documentation . . . . .	26
<b>3</b>	<b>Modeling Cookbook</b>	<b>27</b>
3.1	System . . . . .	27
3.1.1	Overview . . . . .	27
3.1.2	DAE Storage . . . . .	30
3.1.3	Model and DAE Values . . . . .	31
3.1.4	Calling Model Methods . . . . .	32
3.1.5	Configuration . . . . .	33
3.2	Models . . . . .	34
3.2.1	Model Data . . . . .	34
3.2.2	Define a DAE Model . . . . .	36
3.2.3	Dynamicity Under the Hood . . . . .	38
3.2.4	Equation Generation . . . . .	39
3.2.5	Jacobian Storage . . . . .	41
3.2.6	Initialization . . . . .	42
3.2.7	Additional Numerical Equations . . . . .	42
3.3	Atom Types . . . . .	42
3.3.1	Value Provider . . . . .	42
3.3.2	Equation Provider . . . . .	43
3.4	Parameters . . . . .	44
3.4.1	Background . . . . .	44
3.4.2	Data Parameters . . . . .	44
3.4.3	Numeric Parameters . . . . .	46
3.4.4	External Parameters . . . . .	47
3.4.5	Timer Parameter . . . . .	47
3.5	Variables . . . . .	48
3.5.1	Variable, Equation and Address . . . . .	49
3.5.2	Value and Equation Strings . . . . .	49
3.5.3	Values Between DAE and Models . . . . .	49
3.5.4	Flags for Value Overwriting . . . . .	49
3.5.5	A <i>v_setter</i> Example . . . . .	50
3.6	Services . . . . .	53
3.6.1	Internal Constants . . . . .	54
3.6.2	External Constants . . . . .	56
3.6.3	Shape Manipulators . . . . .	57
3.6.4	Value Manipulation . . . . .	61
3.6.5	Idx and References . . . . .	61
3.6.6	Events . . . . .	63
3.6.7	Data Select . . . . .	64
3.6.8	Miscellaneous . . . . .	65
3.7	Discrete . . . . .	66

3.7.1	Background . . . . .	66
3.7.2	Limiters . . . . .	66
3.7.3	Comparers . . . . .	68
3.7.4	Deadband . . . . .	70
3.8	Blocks . . . . .	71
3.8.1	Background . . . . .	71
3.8.2	Transfer Functions . . . . .	72
3.8.3	Saturation . . . . .	78
3.8.4	Others . . . . .	79
3.9	Examples . . . . .	79
3.9.1	TGOV1 . . . . .	80
3.9.2	IEEEEST . . . . .	82
<b>4</b>	<b>Test Cases</b>	<b>87</b>
4.1	Directory . . . . .	87
4.2	MATPOWER Cases . . . . .	88
<b>5</b>	<b>Model References</b>	<b>93</b>
5.1	ACLine . . . . .	94
5.1.1	Line . . . . .	94
5.2	ACTopology . . . . .	96
5.2.1	Bus . . . . .	96
5.3	Calculation . . . . .	97
5.3.1	ACE . . . . .	97
5.3.2	COI . . . . .	98
5.4	Collection . . . . .	99
5.4.1	Area . . . . .	100
5.5	DCLink . . . . .	100
5.5.1	Ground . . . . .	100
5.5.2	R . . . . .	101
5.5.3	L . . . . .	102
5.5.4	C . . . . .	103
5.5.5	RCp . . . . .	103
5.5.6	RCs . . . . .	104
5.5.7	RLs . . . . .	105
5.5.8	RLCs . . . . .	106
5.5.9	RLCp . . . . .	107
5.6	DCTopology . . . . .	108
5.6.1	Node . . . . .	109
5.7	DG . . . . .	109
5.7.1	PVD1 . . . . .	110
5.8	Exciter . . . . .	112
5.8.1	EXDC2 . . . . .	112
5.8.2	IEEEEX1 . . . . .	116
5.8.3	ESDC2A . . . . .	119
5.8.4	EXST1 . . . . .	123
5.8.5	ESST3A . . . . .	125
5.8.6	SEXS . . . . .	130

5.9	Experimental . . . . .	132
5.9.1	PI2 . . . . .	132
5.9.2	TestDB1 . . . . .	133
5.9.3	TestPI . . . . .	134
5.9.4	TestLagAWFreeze . . . . .	136
5.10	FreqMeasurement . . . . .	137
5.10.1	BusFreq . . . . .	137
5.10.2	BusROCOF . . . . .	138
5.11	Motor . . . . .	140
5.11.1	Motor3 . . . . .	140
5.11.2	Motor5 . . . . .	142
5.12	PSS . . . . .	144
5.12.1	IEEEEST . . . . .	144
5.12.2	ST2CUT . . . . .	148
5.13	PhasorMeasurement . . . . .	152
5.13.1	PMU . . . . .	152
5.14	RenAerodynamics . . . . .	153
5.14.1	WTARA1 . . . . .	154
5.15	RenExciter . . . . .	154
5.15.1	REECA1 . . . . .	154
5.16	RenGen . . . . .	162
5.16.1	REGCA1 . . . . .	162
5.17	RenGovernor . . . . .	166
5.17.1	WTDTA1 . . . . .	166
5.17.2	WTDS . . . . .	168
5.18	RenPitch . . . . .	170
5.18.1	WTPTA1 . . . . .	170
5.19	RenPlant . . . . .	172
5.19.1	REPCA1 . . . . .	173
5.20	RenTorque . . . . .	178
5.20.1	WTTQA1 . . . . .	178
5.21	StaticACDC . . . . .	181
5.21.1	VSCShunt . . . . .	182
5.22	StaticGen . . . . .	184
5.22.1	PV . . . . .	185
5.22.2	Slack . . . . .	186
5.23	StaticLoad . . . . .	188
5.23.1	PQ . . . . .	188
5.24	StaticShunt . . . . .	190
5.24.1	Shunt . . . . .	190
5.25	SynGen . . . . .	191
5.25.1	GENCLS . . . . .	191
5.25.2	GENROU . . . . .	194
5.26	TimedEvent . . . . .	199
5.26.1	Toggler . . . . .	199
5.26.2	Fault . . . . .	199
5.27	TurbineGov . . . . .	201
5.27.1	TG2 . . . . .	201

5.27.2	TGOV1	203
5.27.3	TGOV1DB	205
5.27.4	IEEEG1	208
5.28	Undefined	211
<b>6</b>	<b>Config References</b>	<b>213</b>
6.1	System	213
6.2	PFlow	214
6.3	TDS	214
6.4	EIG	215
<b>7</b>	<b>Frequently Asked Questions</b>	<b>217</b>
7.1	General	217
7.2	Modeling	217
7.2.1	Admittance matrix	217
<b>8</b>	<b>Troubleshooting</b>	<b>219</b>
8.1	Import Errors	219
8.1.1	ImportError: DLL load failed	219
8.2	Runtime Errors	219
8.2.1	EOFError: Ran out of input	219
<b>9</b>	<b>Miscellaneous</b>	<b>221</b>
9.1	Notes	221
9.1.1	Modeling Blocks	221
9.2	Per Unit System	222
9.3	Profiling Import	222
<b>10</b>	<b>Release Notes</b>	<b>223</b>
10.1	v1.2 Notes	223
10.1.1	v1.2.0 (2020-10-10)	223
10.2	v1.1 Notes	224
10.2.1	v1.1.5 (2020-10-08)	224
10.2.2	v1.1.4 (2020-09-22)	224
10.2.3	v1.1.3 (2020-09-05)	224
10.2.4	v1.1.2 (2020-09-03)	224
10.2.5	v1.1.1 (2020-09-02)	224
10.2.6	v1.1.0 (2020-09-01)	224
10.3	v1.0 Notes	225
10.3.1	v1.0.8 (2020-07-29)	225
10.3.2	v1.0.7 (2020-07-18)	225
10.3.3	v1.0.6 (2020-07-08)	226
10.3.4	v1.0.5 (2020-07-02)	226
10.3.5	v1.0.4 (2020-06-26)	226
10.3.6	v1.0.3 (2020-06-02)	226
10.3.7	v1.0.2 (2020-06-01)	226
10.3.8	v1.0.1 (2020-05-27)	226
10.3.9	v1.0.0 (2020-05-25)	227

10.4	Pre-v1.0.0 . . . . .	227
10.4.1	v0.9.4 (2020-05-20) . . . . .	227
10.4.2	v0.9.3 (2020-05-05) . . . . .	227
10.4.3	v0.9.1 (2020-05-02) . . . . .	228
10.4.4	v0.8.8 (2020-04-28) . . . . .	228
10.4.5	v0.8.7 (2020-04-28) . . . . .	228
10.4.6	v0.8.6 (2020-04-21) . . . . .	229
10.4.7	v0.8.5 (2020-04-17) . . . . .	229
10.4.8	v0.8.4 (2020-04-07) . . . . .	229
10.4.9	v0.8.3 (2020-03-25) . . . . .	229
10.4.10	v0.8.0 (2020-02-12) . . . . .	230
10.4.11	v0.6.9 (2020-02-12) . . . . .	230
<b>11</b>	<b>License</b>	<b>231</b>
11.1	GNU Public License v3 . . . . .	231
<b>12</b>	<b>Subpackages</b>	<b>233</b>
12.1	andes.core package . . . . .	233
12.1.1	Submodules . . . . .	233
12.1.2	andes.core.block module . . . . .	233
12.1.3	andes.core.discrete module . . . . .	251
12.1.4	andes.core.model module . . . . .	260
12.1.5	andes.core.param module . . . . .	272
12.1.6	andes.core.service module . . . . .	278
12.1.7	andes.core.solver module . . . . .	291
12.1.8	andes.core.common module . . . . .	295
12.1.9	andes.core.var module . . . . .	297
12.1.10	Module contents . . . . .	301
12.2	andes.io package . . . . .	302
12.2.1	Submodules . . . . .	302
12.2.2	andes.io.matpower module . . . . .	302
12.2.3	andes.io.psse module . . . . .	302
12.2.4	andes.io.txt module . . . . .	302
12.2.5	andes.io.xlsx module . . . . .	303
12.2.6	Module contents . . . . .	303
12.3	andes.models package . . . . .	304
12.3.1	Submodules . . . . .	304
12.3.2	andes.models.area module . . . . .	304
12.3.3	andes.models.bus module . . . . .	305
12.3.4	andes.models.governor module . . . . .	305
12.3.5	andes.models.group module . . . . .	306
12.3.6	andes.models.jit module . . . . .	311
12.3.7	andes.models.line module . . . . .	311
12.3.8	andes.models.pq module . . . . .	311
12.3.9	andes.models.pv module . . . . .	312
12.3.10	andes.models.shunt module . . . . .	312
12.3.11	andes.models.synchronous module . . . . .	312
12.3.12	andes.models.timer module . . . . .	313



12.3.13	Module contents	314
12.4	andes.routines package	314
12.4.1	Submodules	314
12.4.2	andes.routines.base module	314
12.4.3	andes.routines.eig module	314
12.4.4	andes.routines.pflow module	315
12.4.5	andes.routines.tds module	316
12.4.6	Module contents	318
12.5	andes.utils package	318
12.5.1	Submodules	318
12.5.2	andes.utils.cached module	318
12.5.3	andes.utils.paths module	318
12.5.4	andes.utils.func module	320
12.5.5	andes.utils.misc module	320
12.5.6	andes.utils.tab module	320
12.5.7	Module contents	321
12.6	andes.variables package	321
12.6.1	Submodules	321
12.6.2	andes.variables.dae module	321
12.6.3	andes.variables.fileman module	325
12.6.4	andes.variables.report module	325
12.6.5	Module contents	325
<b>13</b>	<b>Submodules</b>	<b>327</b>
13.1	andes.cli module	327
13.2	andes.main module	327
13.3	andes.plot module	330
13.4	andes.shared module	335
13.5	andes.system module	335
<b>14</b>	<b>Indices and tables</b>	<b>343</b>
	<b>Python Module Index</b>	<b>345</b>
	<b>Index</b>	<b>347</b>



ANDES is a Python-based free software package for power system simulation, control and analysis. It establishes a unique **hybrid symbolic-numeric framework** for modeling differential algebraic equations (DAEs) for numerical analysis. Main features of ANDES include

- a unique hybrid symbolic-numeric approach to modeling and simulation that enables descriptive DAE modeling and automatic numerical code generation
- a rich library of transfer functions and discontinuous components (including limiters, dead-bands, and saturation) available for prototyping models, which can be readily instantiated as multiple devices for system analysis
- comes with the Newton method for power flow calculation, the implicit trapezoidal method for time-domain simulation, and full eigenvalue calculation
- strictly verified models with commercial software. ANDES obtains identical time-domain simulation results for IEEE 14-bus and NPCC system with GENROU and multiple controller models. See the verification link for details.
- developed with performance in mind. While written in Python, ANDES comes with a performance package and can finish a 20-second transient simulation of a 2000-bus system in a few seconds on a typical desktop computer
- out-of-the-box PSS/E raw and dyr file support for available models. Once a model is developed, inputs from a dyr file can be readily supported
- an always up-to-date equation documentation of implemented models

ANDES is currently under active development. To get involved,

- Follow the tutorial at <https://andes.readthedocs.io>
- Checkout the Notebook examples in the [examples folder](#)
- Try ANDES in Jupyter Notebook [with Binder](#)
- Download the PDF manual at [download](#)
- Report issues in the [GitHub issues page](#)
- Learn version control with [the command-line git](#) or [GitHub Desktop](#)
- If you are looking to develop models, read the [Modeling Cookbook](#)

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. **ANDES is made open source as part of the CURENT Large Scale Testbed project.**

ANDES is developed and actively maintained by [Hantao Cui](#). See the GitHub repository for a full list of contributors.



ANDES can be installed in Python 3.6+. Please follow the installation guide carefully.

## 1.1 Environment

### 1.1.1 Setting Up Miniconda

We recommend the Miniconda distribution that includes the conda package manager and Python. Downloaded and install the latest Miniconda (x64, with Python 3) from <https://conda.io/miniconda.html>.

Step 1: Open terminal (on Linux or macOS) or *Anaconda Prompt* (on Windows, **not the cmd program!!**). Make sure you are in a conda environment - you should see (base) prepended to the command-line prompt, such as (base) C:\Users\user>.

Create a conda environment for ANDES (recommended)

```
conda create --name andes python=3.7
```

Activate the new environment with

```
conda activate andes
```

You will need to activate the `andes` environment every time in a new Anaconda Prompt or shell.

Step 2: Add the `conda-forge` channel and set it as default

```
conda config --add channels conda-forge
conda config --set channel_priority flexible
```

If these steps complete without an error, continue to *Install Andes*.

### 1.1.2 Existing Python Environment (Advanced)

This is for advanced user only and is **not recommended on Microsoft Windows**. Please skip it if you have set up a Conda environment.

Instead of using Conda, if you prefer an existing Python environment, you can install ANDES with *pip*:

```
python3 -m pip install andes
```

If you see a *Permission denied* error, you will need to install the packages locally with *-user*

## 1.2 Install ANDES

ANDES can be installed in the user mode and the development mode.

- If you want to use ANDES without modifying the source code, install it in the *User Mode*.
- If you want to develop models or routine, install it in the *Development Mode*.

### 1.2.1 User Mode

In the Anaconda environment, run

```
conda install andes
```

You will be prompted to confirm the installation,

This command installs ANDES into the active environment, which should be called `andes` if you followed all the above steps.

---

**Note:** To use `andes`, you will need to activate the `andes` environment every time in a new Anaconda Prompt or shell.

---

### 1.2.2 Development Mode

This is for users who want to hack into the code and, for example, develop new models or routines. The usage of ANDES is the same in development mode as in user mode. In addition, changes to source code will be reflected immediately without re-installation.

Step 1: Get ANDES source code

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

```
git clone https://github.com/cuihantao/andes
```

In this way, you can easily update to the latest source code using `git`.

Alternatively, you can download the ANDES source code from <https://github.com/cuihantao/andes> and extract all files to the path of your choice. Although this will work, this is not recommended since tracking changes and pushing back code would be painful.

### Step 2: Install dependencies

In the Anaconda environment, use `cd` to change directory to the ANDES root folder.

Install dependencies with

```
conda install --file requirements.txt
conda install --file requirements-dev.txt
```

### Step 3: Install ANDES in the development mode using

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

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

## 1.3 Updating ANDES

Regular ANDES 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 ANDES, you will use one of the following ways to upgrade.

If you installed it from conda (most common for users), run

```
conda install -c conda-forge --yes andes
```

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

```
python3 -m pip install --yes andes
```

If you installed ANDES from source code, and the source was cloned using `git`, you can use `git pull` to pull in changes from remote. However, if your source code was downloaded, you will have to download the new source code again and manually overwrite the existing one.

In rare cases, after updating the source code, command-line `andes` will complain about missing dependency. If this ever happens, it means the new ANDES has introduced new dependencies. In such cases, reinstall `andes` in the development mode to fix. Change directory to the ANDES source code folder that contains `setup.py` and run

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

## 1.4 Performance Packages (Advanced)

The following two forks of `cvxopt`, `cvxoptklu`, `cvxopt` with `spmatrix.ipadd` are optional but can significantly boost the performance of ANDES. **Installation requires a C compiler**, `openblas` and `SuiteSparse` libraries.

---

**Note:** Performance packages can be safely skipped and will not affect the functionality of ANDES.

---

**Warning:** We have not tried to compile either package on Windows. Refer to the CVXOPT installation instructions for Windows at <http://cvxopt.org/install/index.html#windows>

### 1.4.1 cvxoptklu

`cvxoptklu` is a fork of the CVXOPT with KLU by Uriel Sandoval (@sanurielf). In addition to UMF-PACK, `cvxoptklu` interfaces `cvxopt` to KLU, which is roughly 20% faster than UMF-PACK for circuit simulation based on our testing.

To install `cvxoptklu`, on Debian GNU/Linux, one can do

```
sudo apt install libopenblas-dev libsuitesparse-dev
pip install cvxoptklu
```

On macOS, one can install with homebrew using

```
brew install openblas suitesparse
pip install cvxoptklu
```

To install from source code, use the repository at <https://github.com/cuihantao/cvxoptklu>.

### 1.4.2 CVXOPT with ipadd

To install our fork of `cvxopt` with `spmatrix.ipadd`, one need to clone the repository and compile from source.

```
git clone https://github.com/curent/cvxopt
cd cvxopt
python setup.py build
```

The compilation may display some warnings, but make sure there is no error. Then, install it with

```
python setup.py install
```



ANDES can be used as a command-line tool or a library. The command-line interface (CLI) comes handy to run studies. As a library, it can be used interactively in the IPython shell or the Jupyter Notebook. This chapter describes the most common usages.

Please see the cheat sheet if you are looking for quick help.

## 2.1 Command Line Usage

### 2.1.1 Basic Usage

ANDES is invoked from the command line using the command `andes`. Running `andes` without any input is equal to `andes -h` or `andes --help`. It prints out a preamble with version and environment information and help commands:

```

      _ _ _ _ _ | Version 1.1.2
    / _ \ _ _ _ _ | Python 3.7.6 on Linux, 09/05/2020 12:23:05 PM
   / _ \ | ' \ / _ \ / _ \ _ _ < |
  / _ \ \ _ \ | | _ \ _ _ \ _ _ / _ \ | This program comes with ABSOLUTELY NO WARRANTY.

usage: andes [-h] [-v {1,10,20,30,40,50}]
           {run,plot,doc,misc,prepare,selftest} ...

positional arguments:
  {run,plot,doc,misc,prepare,selftest}
    [run] run simulation routine; [plot] plot simulation
    results; [doc] quick documentation; [prepare] run the
    symbolic-to-numeric preparation; [misc] miscellaneous
    functions.

```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -v {1,10,20,30,40,50}, --verbose {1,10,20,30,40,50}
                        Program logging level in 10-DEBUG, 20-INFO,
                        30-WARNING, 40-ERROR or 50-CRITICAL.
```

---

**Note:** If the `andes` command is not found, check if (1) the installation was successful, and (2) you have activated the environment where ANDES is installed.

---

The first level of commands are chosen from `{run,plot,misc,prepare,selftest}`. Each command contains a group of sub-commands, which can be looked up with `-h`. For example, use `andes run -h` to look up the sub-commands in `run`. The most commonly used commands will be explained in the following.

`andes` has an option for the program verbosity level, controlled by `-v` or `--verbose`. Accepted levels are the same as in the logging module: 10 - DEBUG, 20 - INFO, 30 - WARNING, 40 - ERROR, 50 - CRITICAL. To show debugging outputs, use `-v 10`.

### 2.1.2 andes selftest

After installation, it is encouraged to use `andes selftest` from the command line to test functionality. It might take a minute to run the full self-test suite. An example output looks like

```
test_docs (test_1st_system.TestCodegen) ... ok
test_alter_param (test_case.Test5Bus) ... ok
...
... (outputs are omitted)
...
test_pflow_mpc (test_pflow_matpower.TestMATPOWER) ... ok

-----
Ran 23 tests in 13.834s

OK
```

There may be more cases than what is shown above. Make sure that all tests have passed.

**Warning:** ANDES is getting updates frequently. After updating your copy, please run `andes selftest` to confirm the functionality. The command also makes sure the generated code is up to date. See [andes prepare](#) for more details on automatic code generation.

### 2.1.3 andes prepare

The symbolically defined models in ANDES need to be generated into numerical code for simulation. The code generation can be manually called with `andes prepare`. Generated code are stored in the folder `.andes/calls.pkl` in your home directory. In addition, `andes selftest` implicitly calls the code generation. If you are using ANDES as a package in the user mode, you won't need to call it again.

For developers, `andes prepare` needs to be called immediately following any model equation modification. Otherwise, simulation results will not reflect the new equations and will likely lead to an error.

Option `-q` or `--quick` (enabled by default) can be used to speed up the code generation. It skips the generation of *L<sup>A</sup>T<sub>E</sub>X*-formatted equations, which are only used in documentation and the interactive mode.

Option `-i` or `--incremental` can be used to further speed up the code generation during model development. `andes prepare -i` only generates code for models with modified equations.

### 2.1.4 andes run

`andes run` is the entry point for power system analysis routines. `andes run` takes one positional argument, `filename`, along with other optional keyword arguments. `filename` is the test case path, either relative or absolute. Without other options, ANDES will run power flow calculation for the provided file.

#### Routine

Option `-r` or `-routine` is used for specifying the analysis routine, followed by the routine name. Available routine names include `pflow`, `tds`, `eig`. `pflow` for power flow, `tds` for time domain simulation, and `eig` for eigenvalue analysis. `pflow` is default even if `-r` is not given.

For example, to run time-domain simulation for `kundur_full.xlsx` in the *current directory*, run

```
andes run kundur_full.xlsx -r tds
```

The file is located at `andes/cases/kundur/kundur_full.xlsx` relative to the source code root folder. Use `cd` to change directory to that folder on your machine.

Two output files, `kundur_full_out.lst` and `kundur_full_out.npy` will be created for variable names and values, respectively.

Likewise, to run eigenvalue analysis for `kundur_full.xlsx`, use

```
andes run kundur_full.xlsx -r eig
```

The eigenvalue report will be written in a text file named `kundur_full_eig.txt`.

#### Power flow

To perform a power flow study for test case named `kundur_full.xlsx` in the current directory, run

```
andes run kundur_full.xlsx
```

The full path to the case file is also accepted, for example,

```
andes run /home/user/andes/cases/kundur/kundur_full.xlsx
```

Power flow reports will be saved to the current directory in which andes is called. The power flow report contains four sections: a) system statistics, b) ac bus and dc node data, c) ac line data, and d) the initialized values of other algebraic variables and state variables.

## Time-domain simulation

To run the time domain simulation (TDS) for `kundur_full.xlsx`, run

```
andes run kundur_full.xlsx -r tds
```

The output looks like:

```
Parsing input file </Users/user/repos/andes/tests/kundur_full.xlsx>
Input file kundur_full.xlsx parsed in 0.5425 second.
-> Power flow calculation with Newton Raphson method:
0: |F(x)| = 14.9283
1: |F(x)| = 3.60859
2: |F(x)| = 0.170093
3: |F(x)| = 0.00203827
4: |F(x)| = 3.76414e-07
Converged in 5 iterations in 0.0080 second.
Report saved to </Users/user/repos/andes/tests/kundur_full_out.txt> in 0.0036
↪second.
-> Time Domain Simulation:
Initialization tests passed.
Initialization successful in 0.0152 second.
  0%|                                     | 0/100 [00:00<?, ?%/
↪s]
  <Toggle 0>: Applying status toggle on Line idx=Line_8
100%|-----| 100/100 [00:03<00:00, 28.99%/s]
Simulation completed in 3.4500 seconds.
TDS outputs saved in 0.0377 second.
-> Single process finished in 4.4310 seconds.
```

This execution first solves the power flow as a starting point. Next, the numerical integration simulates 20 seconds, during which a predefined breaker opens at 2 seconds.

TDS produces two output files by default: a NumPy data file `ieee14_syn_out.npy` and a variable name list file `ieee14_syn_out.lst`. The list file contains three columns: variable indices, variable name in plain text, and variable name in the  $\LaTeX$  format. The variable indices are needed to plot the needed variable.

## Disable output

The output files can be disabled with option `--no-output` or `-n`. It is useful when only computation is needed without saving the results.

## Profiling

Profiling is useful for analyzing the computation time and code efficiency. Option `--profile` enables the profiling of ANDES execution. The profiling output will be written in two files in the current folder, one ending with `_prof.txt` and the other one with `_prof.prof`.

The text file can be opened with a text editor, and the `.prof` file can be visualized with `snakeviz`, which can be installed with `pip install snakeviz`.

If the output is disabled, profiling results will be printed to `stdio`.

## Multiprocessing

ANDES takes multiple files inputs or wildcard. Multiprocessing will be triggered if more than one valid input files are found. For example, to run power flow for files with a prefix of `case5` and a suffix (file extension) of `.m`, run

```
andes run case5*.m
```

Test cases that match the pattern, including `case5.m` and `case57.m`, will be processed.

Option `--ncpu NCPU` can be used to specify the maximum number of parallel processes. By default, all cores will be used. A small number can be specified to increase operation system responsiveness.

## Format converter

ANDES recognizes a few input formats and can convert input systems into the `xlsx` format. This function is useful when one wants to use models that are unique in ANDES.

The command for converting is `--convert` (or `-c`), following the output format (only `xlsx` is currently supported). For example, to convert `case5.m` into the `xlsx` format, run

```
andes run case5.m --convert xlsx
```

The output messages will look like

```
Parsing input file </Users/user/repos/andes/cases/matpower/case5.m> CASE5 Power flow data
for modified 5 bus, 5 gen case based on PJM 5-bus system Input file case5.m parsed in 0.0033
second.  xlsx file written to </Users/user/repos/andes/cases/matpower/case5.xlsx> Converted
file /Users/user/repos/andes/cases/matpower/case5.xlsx written in 0.5079 second.  -> Single
process finished in 0.8765 second.
```

Note that `--convert` will only create sheets for existing models.

In case one wants to create template sheets to add models later, `--convert-all` can be used instead.

If one wants to add workbooks to an existing `xlsx` file, one can combine option `--add-book ADD_BOOK` (or `-b ADD_BOOK`), where `ADD_BOOK` can be a single model name or comma-separated model names (without any space). For example,

```
andes run kundur.raw -c -b Toggler
```

will convert file `kundur.raw` into an ANDES `xlsx` file (`kundur.xlsx`) and add a template workbook for *Toggler*.

**Warning:** With `--add-book`, the `xlsx` file will be overwritten. Any **empty or non-existent models** will be REMOVED.

### PSS/E inputs

To work with PSS/E input files (`.raw` and `.dyr`), one need to provide the raw file as `casefile` and pass the `dyr` file to `--addfile`. For example, in `andes/andes/cases/wecc`, one can run the power flow using

```
andes run wecc.raw
```

and run a no-disturbance time-domain simulation using

```
andes run wecc.raw --addfile wecc_full.dyr -r tds
```

To create add a disturbance, there are two options. The recommended option is to convert the PSS/E data into an ANDES `xlsx` file, edit and run (see the previous subsection).

The alternative is to edit the `dyr` file and append lines customized for ANDES models. This is for advanced users after referring to `andes/io/psse-dyr.yaml`, at the end of which one can find the format of *Toggler*:

```
# === Custom Models ===
Toggler:
  inputs:
    - model
    - dev
    - t
```

To define two *Togglers* in the `dyr` file, one can append lines to the end of the file using, for example,

```
Line   'Toggler'   Line_2   1 /
Line   'Toggler'   Line_2   1.1 /
```

which is separated by spaces and ended with a slash. The second parameter is fixed to the model name quoted by a pair of single quotation marks, and the others correspond to the fields defined in the above “inputs”.

---

**Note:** When working with PSS/E data, the recommended practice is to edit model dynamic parameters

directly in the `dyr` file so that the data can be easily used by other tools.

### 2.1.5 andes plot

`andes plot` is the command-line tool for plotting. It currently supports time-domain simulation data. Three positional arguments are required, and a dozen of optional arguments are supported.

positional arguments:

Argument	Description
filename	simulation output file name, which should end with <i>out</i> . File extension can be omitted.
x	the X-axis variable index, typically 0 for Time
y	Y-axis variable indices. Space-separated indices or a colon-separated range is accepted

For example, to plot the generator speed variable of synchronous generator 1 `omega GENROU 0` versus time, read the indices of the variable (2) and time (0), run

```
andes plot kundur_full_out.lst 0 2
```

In this command, `andes plot` is the plotting command for TDS output files. `kundur_full_out.lst` is list file name. 0 is the index of Time for the x-axis. 2 is the index of `omega GENROU 0`. Note that for the file name, either `kundur_full_out.lst` or `kundur_full_out.npy` works, as the program will automatically extract the file name.

The y-axis variable indices can also be specified in the Python range fashion. For example, `andes plot kundur_full_out.npy 0 2:21:6` will plot the variables at indices 2, 8, 14 and 20.

`andes plot` will attempt to render with  $\text{\LaTeX}$  if `dvipng` program is in the search path. Figures rendered by  $\text{\LaTeX}$  is considerably better in symbols quality but takes much longer time. In case  $\text{\LaTeX}$  is available but fails (frequently happens on Windows), the option `-d` can be used to disable  $\text{\LaTeX}$  rendering.

Other optional arguments are listed in the following.

**optional arguments:**

Argument	Description
optional arguments:	
-h, --help	show this help message and exit
-xmin LEFT	minimum value for X axis
-xmax RIGHT	maximum value for X axis
-ymax YMAX	maximum value for Y axis
-ymin YMIN	minimum value for Y axis
-find FIND	find variable indices that matches the given pattern
-xargs XARGS	find variable indices and return as a list of arguments usable with "l xargs andes plot"
-exclude EXCLUDE	pattern to exclude in find or xargs results
-x XLABEL, --xlabel XLABEL	x-axis label text
-y YLABEL, --ylabel YLABEL	y-axis label text
-s, --savefig	save figure. The default fault is <i>png</i> .
-format SAVE_FORMAT	format for savefig. Common formats such as png, pdf, jpg are supported
-dpi DPI	image resolution in dot per inch (DPI)
-g, --grid	grid on
--greyscale	greyscale on
-d, --no-latex	disable LaTeX formatting
-n, --no-show	do not show the plot window
-ytimes YTIMES	scale the y-axis values by YTIMES
-c, --tocsv	convert npy output to csv

## 2.1.6 andes doc

`andes doc` is a tool for quick lookup of model and routine documentation. It is intended as a quick way for documentation.

The basic usage of `andes doc` is to provide a model name or a routine name as the positional argument. For a model, it will print out model parameters, variables, and equations to the stdio. For a routine, it will print out fields in the Config file. If you are looking for full documentation, visit [andes.readthedocs.io](https://andes.readthedocs.io).

For example, to check the parameters for model `Toggler`, run

```
$ andes doc Toggler
Model <Toggler> in Group <TimedEvent>

    Time-based connectivity status toggler.

Parameters

Name | Description | Default | Unit | Type |
--Properties
-----+-----+-----+-----+-----+

```

(continues on next page)



(continued from previous page)

u	connection status	1	bool	NumParam	
name	device name			DataParam	
model	Model or Group of the device			DataParam	└
↳mandatory					
	to control				
dev	idx of the device to control			IdxParam	└
↳mandatory					
t	switch time for connection	-1		TimerParam	└
↳mandatory					
	status				

To list all supported models, run

```
$ andes doc -l
```

Supported Groups and Models

Group	Models
ACLine	Line
ACTopology	Bus
Collection	Area
DCLink	Ground, R, L, C, RCp, RCs, RLs, RLCs, RLCp
DCTopology	Node
Exciter	EXDC2
Experimental	PI2
FreqMeasurement	BusFreq, BusROCOF
StaticACDC	VSCShunt
StaticGen	PV, Slack
StaticLoad	PQ
StaticShunt	Shunt
SynGen	GENCLS, GENROU
TimedEvent	Toggler, Fault
TurbineGov	TG2, TGOV1

To view the Config fields for a routine, run

```
$ andes doc TDS
```

Config Fields in [TDS]

Option	Value	Info	Acceptable
↳values			
↳--			
sparselib	klu	linear sparse solver name	('klu', 'umfpack
↳')			
tol	0.000	convergence tolerance	float
t0	0	simulation starting time	>=0
tf	20	simulation ending time	>t0
fixt	0	use fixed step size (1) or variable	(0, 1)
		(0)	
shrinkt	1	shrink step size for fixed method if	(0, 1)

(continues on next page)

(continued from previous page)

			not converged	
tstep		0.010	the initial step size	float
max_iter		15	maximum number of iterations	>=10

### 2.1.7 andes misc

`andes misc` contains miscellaneous functions, such as configuration and output cleaning.

#### Configuration

ANDES uses a configuration file to set runtime configs for the system routines, and models. `--save-config` saves all configs to a file. By default, it saves to `~/ .andes/andes.conf` file, where `~` is the path to your home directory.

With `--edit-config`, you can edit ANDES configuration handy. The command will automatically save the configuration to the default location if not exist. The shorter version `--edit` can be used instead as Python automatically matches it with `--edit-config`.

You can pass an editor name to `--edit`, such as `--edit vim`. If the editor name is not provided, it will use the following defaults: - Microsoft Windows: notepad. - GNU/Linux: the `$EDITOR` environment variable, or `vim` if not exist.

For macOS users, the default is `vim`. If not familiar with `vim`, you can use `nano` with `--edit nano` or `TextEdit` with `--edit "open -a TextEdit"`.

#### Cleanup

`-C, --clean`

Option to remove any generated files. Removes files with any of the following suffix: `_out.txt` (power flow report), `_out.npy` (time domain data), `_out.lst` (time domain variable list), and `_eig.txt` (eigenvalue report).

## 2.2 Interactive Usage

This section is a tutorial for using ANDES in an interactive environment. All interactive shells are supported, including Python shell, IPython, Jupyter Notebook and Jupyter Lab. The examples below uses Jupyter Notebook.

---

**Note:** All following blocks starting with `>>>` are Python code. They should be typed into a Python shell, IPython or Jupyter Notebook, not a Anaconda Prompt or shell.

---

### 2.2.1 Jupyter Notebook

Jupyter notebook is a convenient tool to run Python code and present results. Jupyter notebook can be installed with

```
conda install jupyter notebook
```

After the installation, change directory to the folder that you wish to store notebooks, then start the notebook with

```
jupyter notebook
```

A browser window should open automatically with the notebook browser loaded. To create a new notebook, use the "New" button at the top-right corner.

### 2.2.2 Import

Like other Python libraries, ANDES needs to be imported into an interactive Python environment.

```
>>> import andes
>>> andes.main.config_logger()
```

### 2.2.3 Verbosity

If you are debugging ANDES, you can enable debug messages with

```
>>> andes.main.config_logger(stream_level=10)
```

The `stream_level` uses the same verbosity levels (see [Basic Usage](#)) as for the command-line. If not explicitly enabled, the default level 20 (INFO) will apply.

**Warning:** The verbosity level can only be set once. To set a different level, restart the Python kernel.

### 2.2.4 Making a System

Before running studies, a "System" object needs to be created to hold the system data. The System object can be created by passing the path to the case file the `entrypoint` function. For example, to run the file `kundur_full.xlsx` in the same directory as the notebook, use

```
>>> ss = andes.run('kundur_full.xlsx')
```

This function will parse the input file, run the power flow, and return the system as an object. Outputs will look like

```
Parsing input file </Users/user/notebooks/kundur/kundur_full.xlsx>
Input file kundur_full.xlsx parsed in 0.4172 second.
-> Power flow calculation with Newton Raphson method:
0: |F(x)| = 14.9283
1: |F(x)| = 3.60859
2: |F(x)| = 0.170093
3: |F(x)| = 0.00203827
4: |F(x)| = 3.76414e-07
Converged in 5 iterations in 0.0222 second.
Report saved to </Users/user/notebooks/kundur_full_out.txt> in 0.0015 second.
-> Single process finished in 0.4677 second.
```

In this example, `ss` is an instance of `andes.System`. It contains member attributes for models, routines, and numerical DAE.

Naming convention for the `System` attributes are as follows

- Model attributes share the same name as class names. For example, `ss.Bus` is the `Bus` instance.
- Routine attributes share the same name as class names. For example, `ss.PFlow` and `ss.TDS` are the routine instances.
- The numerical DAE instance is in lower case `ss.dae`.

To work with PSS/E inputs, refer to notebook [Example 2](#).

## Output path

By default, outputs will be saved to the folder where Python is run (or where the notebook is run). In case you need to organize outputs, a path prefix can be passed to `andes.run()` through `output_path`. For example,

```
>>> ss = andes.run('kundur_full.xlsx', output_path='outputs/')
```

will put outputs into folder `outputs` relative to the current path. You can also supply an absolute path to `output_path`.

## No output

Outputs can be disabled by passing `output_path=True` to `andes.run()`. This is useful when one wants to test code without looking at results. For example, do

```
>>> ss = andes.run('kundur_full.xlsx', no_output=True)
```

## 2.2.5 Inspecting Parameter

### DataFrame

Parameters for the loaded system can be easily inspected in Jupyter Notebook using `Pandas`.

Input parameters for each model instance is returned by the `as_df()` function. For example, to view the input parameters for `Bus`, use

```
>>> ss.Bus.as_df()
```

A table will be printed with the columns being each parameter and the rows being `Bus` instances. Parameter in the table is the same as the input file without per-unit conversion.

Parameters have been converted to per unit values under system base. To view the per unit values, use the `as_df_in()` attribute. For example, to view the system-base per unit value of `GENROU`, use

```
>>> ss.GENROU.as_df_in()
```

## Dict

In case you need the parameters in dict, use `as_dict()`. Values returned by `as_dict()` are system-base per unit values. To retrieve the input data, use `as_dict(vin=True)`.

For example, to retrieve the original input data of `GENROU`'s, use

```
>>> ss.GENROU.as_dict(vin=True)
```

## 2.2.6 Running Studies

Three routines are currently supported: `PFlow`, `TDS` and `EIG`. Each routine provides a `run()` method to execute. The `System` instance contains member attributes having the same names. For example, to run the time-domain simulation for `ss`, use

```
>>> ss.TDS.run()
```

## 2.2.7 Checking Exit Code

`andes.System` contains field `exit_code` for checking if error occurred in run time. A normal completion without error should always have `exit_code == 0`. One should read output messages carefully and check the exit code, which is particularly useful for batch simulations.

Error may occur in any phase - data parsing, power flow, or simulation. To diagnose, split the simulation steps and check the outputs from each one.

## 2.2.8 Plotting TDS Results

`TDS` comes with a plotting utility for interactive usage. After running the simulation, a `plotter` attributed will be created for `TDS`. To use the plotter, provide the attribute instance of the variable to plot. For example, to plot all the generator speed, use

```
>>> ss.TDS.plotter.plot(ss.GENROU.omega)
```

---

**Note:** If you see the error

AttributeError: 'NoneType' object has no attribute 'plot'

You will need to manually load plotter with

```
>>> ss.TDS.load_plotter()
```

---

Optional indices is accepted to choose the specific elements to plot. It can be passed as a tuple to the `a` argument

```
>>> ss.TDS.plotter.plot(ss.GENROU.omega, a=(0, ))
```

In the above example, the speed of the "zero-th" generator will be plotted.

## Scaling

A lambda function can be passed to argument `ycalc` to scale the values. This is useful to convert a per-unit variable to nominal. For example, to plot generator speed in Hertz, use

```
>>> ss.TDS.plotter.plot(ss.GENROU.omega, a=(0, ),
                        ycalc=lambda x: 60*x,
                        )
```

## Formatting

A few formatting arguments are supported:

- `grid = True` to turn on grid display
- `greyscale = True` to switch to greyscale
- `ylabel` takes a string for the y-axis label

### 2.2.9 Extracting Data

One can extract data from ANDES for custom plotting. Variable names can be extracted from the following fields of `ss.dae`:

Un-formatted names (non-LaTeX):

- `x_name`: state variable names
- `y_name`: algebraic variable names
- `xy_name`: state variable names followed by algebraic ones

LaTeX-formatted names:

- `x_tex_name`: state variable names
- `y_tex_name`: algebraic variable names
- `xy_tex_name`: state variable names followed by algebraic ones

These lists only contain the variable names used in the current analysis routine. If you only ran power flow, `ss.dae.y_name` will only contain the power flow algebraic variables, and `ss.dae.x_name` will likely be empty. After initializing time-domain simulation, these lists will be extended to include all variables used by TDS.

In case you want to extract the discontinuous flags from TDS, you can set `store_z` to 1 in the config file under section `[TDS]`. When enabled, discontinuous flag names will be populated at

- `ss.dae.z_name`: discontinuous flag names
- `ss.dae.z_tex_name`: LaTeX-formatted discontinuous flag names

If not enabled, both lists will be empty.

## Power flow solutions

The full power flow solutions are stored at `ss.dae.xy` after running power flow (and before initializing dynamic models). You can extract values from `ss.dae.xy`, which corresponds to the names in `ss.dae.xy_name` or `ss.dae.xy_tex_name`.

If you want to extract variables from a particular model, for example, bus voltages, you can directly access the `v` field of that variable

```
>>> import numpy as np
>>> voltages = np.array(ss.Bus.v.v)
```

which stores a **copy** of the bus voltage values. Note that the first `v` is the voltage variable of `Bus`, and the second `v` stands for *value*. It is important to make a copy by using `np.array()` to avoid accidental changes to the solutions.

If you want to extract bus voltage phase angles, do

```
>>> angle = np.array(ss.Bus.a.v)
```

where `a` is the field name for voltage angle.

To find out names of variables in a model, refer to [andes\\_doc](#).

## Time-domain data

Time-domain simulation data will be ready when simulation completes. It is stored in `ss.dae.ts`, which has the following fields:

- `txyz`: a two-dimensional array. The first column is time stamps, and the following are variables. Each row contains all variables for that time step.

- `t`: all time stamps.
- `x`: all state variables (one column per variable).
- `y`: all algebraic variables (one column per variable).
- `z`: all discontinuous flags (if enabled, one column per flag).

If you want the output in pandas DataFrame, call

```
ss.dae.ts.unpack(df=True)
```

Dataframes are stored in the following fields of `ss.dae.ts`:

- `df`: dataframe for states and algebraic variables
- `df_z`: dataframe for discontinuous flags (if enabled)

For both dataframes, time is the index column, and each column correspond to one variable.

## 2.2.10 Pretty Print of Equations

Each ANDES models offers pretty print of  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ -formatted equations in the jupyter notebook environment.

To use this feature, symbolic equations need to be generated in the current session using

```
import andes
ss = andes.System()
ss.prepare()
```

Or, more concisely, one can do

```
import andes
ss = andes.prepare()
```

This process may take a few minutes to complete. To save time, you can selectively generate it only for interested models. For example, to generate for the classical generator model `GENCLS`, do

```
import andes
ss = andes.System()
ss.GENROU.prepare()
```

Once done, equations can be viewed by accessing `ss.<ModelName>.syms.<PrintName>`, where `<ModelName>` is the model name, and `<PrintName>` is the equation or Jacobian name.

---

**Note:** Pretty print only works for the particular `System` instance whose `prepare()` method is called. In the above example, pretty print only works for `ss` after calling `prepare()`.

---

Supported equation names include the following:

- `xy`: variables in the order of *State*, *ExtState*, *Algeb* and *ExtAlgeb*
- `f`: the **right-hand side** of differential equations  $T\dot{\mathbf{x}} = \mathbf{f}$



- $g$ : implicit algebraic equations  $0 = g$
- $df$ : derivatives of  $f$  over all variables  $x, y$
- $dg$ : derivatives of  $g$  over all variables  $x, y$
- $s$ : the value equations for *ConstService*

For example, to print the algebraic equations of model GENCLS, one can use `ss.GENCLS.syms.g`.

## 2.2.11 Finding Help

### General help

To find help on a Python class, method, or function, use the built-in `help()` function. For example, to check how the `get` method of `GENROU` should be called, do

```
help(ss.GENROU.get)
```

In Jupyter notebook, this can be simplified into `?ss.GENROU.get` or `ss.GENROU.get?`.

### Model docs

Model docs can be shown by printing the return of `doc()`. For example, to check the docs of `GENCLS`, do

```
print(ss.GENCLS.doc())
```

It is the same as calling `andes doc GENCLS` from the command line.

## 2.3 Notebook Examples

Check out more examples in Jupyter Notebook in the *examples* folder of the repository at [here](#). You can run the examples in a live Jupyter Notebook online using [Binder](#).

## 2.4 I/O Formats

### 2.4.1 Input Formats

ANDES currently supports the following input formats:

- ANDES Excel (.xlsx)
- PSS/E RAW (.raw) and DYR (.dyr)
- MATPOWER (.m)

## 2.4.2 ANDES xlsx Format

The ANDES xlsx format is a newly introduced format since v0.8.0. This format uses Microsoft Excel for conveniently viewing and editing model parameters. You can use [LibreOffice](#) or [WPS Office](#) alternatively to Microsoft Excel.

### xlsx Format Definition

The ANDES xlsx format contains multiple workbooks (tabs at the bottom). Each workbook contains the parameters of all instances of the model, whose name is the workbook name. The first row in a worksheet is used for the names of parameters available to the model. Starting from the second row, each row corresponds to an instance with the parameters in the corresponding columns. An example of the `Bus` workbook is shown in the following.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	uid	idx	u	name	Vn	vmax	vmin	v0	a0	xcoord	ycoord	area	zone	owner			
2	0	1	1	1	20	1.1	0.9	1	0.570255	0	0	1	1	1			
3	1	2	1	2	20	1.1	0.9	0.99761	0.368746	0	0	1	1	1			
4	2	3	1	12	20	1.1	0.9	0.96263	0.185317	0	0	2	1	1			
5	3	4	1	11	20	1.1	0.9	0.81691	0.462359	0	0	2	1	1			
6	4	5	1	101	230	1.1	0.9	0.97928	0.480203	0	0	1	1	1			
7	5	6	1	102	230	1.1	0.9	0.95796	0.283887	0	0	1	1	1			
8	6	7	1	3	230	1.1	0.9	0.9362	0.126901	0	0	1	1	1			
9	7	8	1	13	230	1.1	0.9	0.87904	-0.08059	0	0	2	1	1			
10	8	9	1	112	230	1.1	0.9	0.89054	0.093618	0	0	2	1	1			
11	9	10	1	111	230	1.1	0.9	0.82958	0.336601	0	0	2	1	1			

A few columns are used across all models, including `uid`, `idx`, `name` and `u`.

- `uid` is an internally generated unique instance index. This column can be left empty if the xlsx file is being manually created. Exporting the xlsx file with `--convert` will automatically assign the `uid`.
- `idx` is the unique instance index for referencing. An unique `idx` should be provided explicitly for each instance. Accepted types for `idx` include numbers and strings without spaces.
- `name` is the instance name.
- `u` is the connectivity status of the instance. Accepted values are 0 and 1. Unexpected behaviors may occur if other numerical values are assigned.

As mentioned above, `idx` is the unique index for an instance to be referenced. For example, a `PQ` instance can reference a `Bus` instance so that the `PQ` is connected to the `Bus`. This is done through providing the `idx` of the desired bus as the `bus` parameter of the `PQ`.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	uid	idx	u	name	bus	Vn	p0	q0	vmax	vmin	owner						
2	0	PQ_0	1		7	230	11.59	-0.735	1.1	0.9	1						
3	1	PQ_1	1		8	230	15.75	-0.899	1.1	0.9	1						
4																	
5																	
6																	
7																	
8																	
9																	
10																	
11																	
12																	
13																	
14																	
15																	
16																	
17																	
18																	
19																	
20																	
21																	

In the example PQ workbook shown above, there are two PQ instances on buses with `idx` being 7 and 8, respectively.

## Convert to xlsx

Please refer to the `--convert` command for converting a recognized file to xlsx. See [format converter](#) for more detail.

## Data Consistency

Input data needs to have consistent types for `idx`. Both string and numerical types are allowed for `idx`, but the original type and the referencing type must be the same. Suppose we have a bus and a connected PQ. The Bus device may use 1 or '1' as its `idx`, as long as the PQ device uses the same value for its `bus` parameter.

The ANDES xlsx reader will try to convert data into numerical types when possible. This is especially relevant when the input `idx` is string literal of numbers, the exported file will have them converted to numbers. The conversion does not affect the consistency of data.

## Parameter Check

The following parameter checks are applied after converting input values to array:

- Any NaN values will raise a `ValueError`
- Any `inf` will be replaced with  $10^8$ , and `-inf` will be replaced with  $-10^8$ .

## 2.5 Cheatsheet

A cheatsheet is available for quick lookup of supported commands.

View the PDF version at

<https://www.cheatography.com//cuihantao/cheat-sheets/andes-for-power-system-simulation/pdf/>

## 2.6 Make Documentation

The documentation can be made locally into a variety of formats. To make HTML documentation, change directory to docs, and do

```
make html
```

After a minute, HTML documentation will be saved to docs/build/html with the index page being index.html.

A list of supported formats is as follows. Note that some format require additional compiler or library

html	to make standalone HTML files
dirhtml	to make HTML files named index.html <b>in</b> directories
singlehtml	to make a single large HTML file
pickle	to make pickle files
json	to make JSON files
htmlhelp	to make HTML files <b>and</b> an HTML help project
qthelp	to make HTML files <b>and</b> a qthelp project
devhelp	to make HTML files <b>and</b> a Devhelp project
epub	to make an epub
latex	to make LaTeX files, you can <b>set</b> PAPER=a4 <b>or</b> PAPER=letter
latexpdf	to make LaTeX <b>and</b> PDF files (default pdflatex)
latexpdfja	to make LaTeX files <b>and</b> run them through platex/dvipdfmx
text	to make text files
man	to make manual pages
texinfo	to make Texinfo files
info	to make Texinfo files <b>and</b> run them through makeinfo
gettext	to make PO message catalogs
changes	to make an overview of <b>all</b> changed/added/deprecated items
xml	to make Docutils-native XML files
pseudoxml	to make pseudoxml-XML files <b>for</b> display purposes
linkcheck	to check <b>all</b> external links <b>for</b> integrity
doctest	to run <b>all</b> doctests embedded <b>in</b> the documentation ( <b>if</b> enabled)
coverage	to run coverage check of the documentation ( <b>if</b> enabled)

This chapter contains advanced topics on modeling and simulation and how they are implemented in ANDES. It aims to provide an in-depth explanation of how the ANDES framework is set up for symbolic modeling and numerical simulation. It also provides an example for interested users to implement customized DAE models.

### 3.1 System

#### 3.1.1 Overview

System is the top-level class for organizing power system models and orchestrating calculations.

```
class andes.system.System(case: Optional[str] = None, name: Optional[str] = None,
                        config_path: Optional[str] = None, default_config: Op-
                        tional[bool] = False, options: Optional[Dict[KT, VT]] =
                        None, **kwargs)
```

System contains models and routines for modeling and simulation.

System contains a several special *OrderedDict* member attributes for housekeeping. These attributes include *models*, *groups*, *routines* and *calls* for loaded models, groups, analysis routines, and generated numerical function calls, respectively.

#### Notes

System stores model and routine instances as attributes. Model and routine attribute names are the same as their class names. For example, *Bus* is stored at `system.Bus`, the power flow calculation routine is at `system.PFlow`, and the numerical DAE instance is at `system.dae`. See attributes for the list of attributes.

### Attributes

**dae** [andes.variables.dae.DAE] Numerical DAE storage  
**files** [andes.variables.fileman.FileMan] File path storage  
**config** [andes.core.Config] System config storage  
**models** [OrderedDict] model name and instance pairs  
**groups** [OrderedDict] group name and instance pairs  
**routines** [OrderedDict] routine name and instance pairs

---

**Note:** *andes.System* is an alias of *andes.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.

```
andes.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.GENCLS` stores the classical generator object,  
`system.models['Bus']` points the same instance as `system.Bus`.

```
andes.system.System.import_groups(self)
```

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

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

```
andes.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 groups will be stored to dictionary `System.groups`.

### Examples

`System.PFlow` is the power flow routine instance, and `System.TDS` and `System.EIG` are time-domain analysis and eigenvalue analysis routines, respectively.

## Code Generation

Under the hood, all symbolically defined equations need to be generated into anonymous function calls for accelerating numerical simulations. This process is automatically invoked for the first time ANDES is run command line. It takes several seconds up to a minute to finish the generation.

---

**Note:** Code generation has been done if one has executed `andes`, `andes selftest`, or `andes prepare`.

---

**Warning:** When models are modified (such as adding new models or changing equation strings), code generation needs to be executed again for consistency. It can be more conveniently triggered from command line with `andes prepare -qi`.

`andes.system.System.prepare(self, quick=False, incremental=False)`

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.

**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()
```

Since the process is slow, generated numerical functions (Python Callable) will be serialized into a file for future speed up. The package used for serializing/de-serializing numerical calls is `dill`. System has a function called `dill` for serializing using the `dill` package.

`andes.system.System.dill(self)`

Serialize generated numerical functions in *System.calls* with package *dill*.

The serialized file will be stored to `~/andes/calls.pkl`, where `~` is the home directory path.

### Notes

This function sets `dill.settings['recurse'] = True` to serialize the function calls recursively.

`andes.system.System.undill(self)`

Deserialize the function calls from `~/andes/calls.pkl` with `dill`.

If no change is made to models, future calls to `prepare()` can be replaced with `undill()` for acceleration.

## 3.1.2 DAE Storage

`System.dae` is an instance of the numerical DAE class.

`andes.variables.dae.DAE(system)`

Class for storing numerical values of the DAE system, including variables, equations and first order derivatives (Jacobian matrices).

Variable values and equation values are stored as `numpy.ndarray`, while Jacobians are stored as `cvxopt.spmatrix`. The defined arrays and descriptions are as follows:

DAE Array	Description
x	Array for state variable values
y	Array for algebraic variable values
z	Array for 0/1 limiter states (if enabled)
f	Array for differential equation derivatives
Tf	Left-hand side time constant array for f
g	Array for algebraic equation mismatches

The defined scalar member attributes to store array sizes are

Scalar	Description
m	The number of algebraic variables/equations
n	The number of algebraic variables/equations
o	The number of limiter state flags



The derivatives of  $f$  and  $g$  with respect to  $x$  and  $y$  are stored in four `cvxopt.spmatrix` sparse matrices: **fx**, **fy**, **gx**, and **gy**, where the first letter is the equation name, and the second letter is the variable name.

## Notes

DAE in ANDES is defined in the form of

$$\begin{aligned} T\dot{x} &= f(x, y) \\ 0 &= g(x, y) \end{aligned}$$

DAE does not keep track of the association of variable and address. Only a variable instance keeps track of its addresses.

### 3.1.3 Model and DAE Values

ANDES uses a decentralized architecture between models and DAE value arrays. In this architecture, variables are initialized and equations are evaluated inside each model. Then, `System` provides methods for collecting initial values and equation values into DAE, as well as copying solved values to each model.

The collection of values from models needs to follow protocols to avoid conflicts. Details are given in the subsection `Variables`.

`andes.system.System.vars_to_dae(self, model)`

Copy variables values from models to `System.dae`.

This function clears `DAE.x` and `DAE.y` and collects values from models.

`andes.system.System.vars_to_models(self)`

Copy variable values from `System.dae` to models.

`andes.system.System._e_to_dae(self, eq_name: Union[str, Tuple] = ('f', 'g'))`

Helper function for collecting equation values into `System.dae.f` and `System.dae.g`.

#### Parameters

**eq\_name** [`'x'` or `'y'` or tuple] Equation type name

### Matrix Sparsity Patterns

The largest overhead in building and solving nonlinear equations is the building of Jacobian matrices. This is especially relevant when we use the implicit integration approach which algebraized the differential equations. Given the unique data structure of power system models, the sparse matrices for Jacobians are built **incrementally**, model after model.

There are two common approaches to incrementally build a sparse matrix. The first one is to use simple in-place add on sparse matrices, such as doing

```
self.fx += spmatrix(v, i, j, (n, n), 'd')
```

Although the implementation is simple, it involves creating and discarding temporary objects on the right hand side and, even worse, changing the sparse pattern of `self.fx`.

The second approach is to store the rows, columns and values in an array-like object and construct the Jacobians at the end. This approach is very efficient but has one caveat: it does not allow accessing the sparse matrix while building.

ANDES uses a pre-allocation approach to avoid the change of sparse patterns by filling values into a known the sparse matrix pattern matrix. System collects the indices of rows and columns for each Jacobian matrix. Before in-place additions, ANDES builds a temporary zero-filled *spmatrix*, to which the actual Jacobian values are written later. Since these in-place add operations are only modifying existing values, it does not change the pattern and thus avoids memory copying. In addition, updating sparse matrices can be done with the exact same code as the first approach.

Still, this approach creates and discards temporary objects. It is however feasible to write a C function which takes three array-likes and modify the sparse matrices in place. This is feature to be developed, and our prototype shows a promising acceleration up to 50%.

```
andes.system.System.store_sparse_pattern(self, models: collections.OrderedDict)
```

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.

### 3.1.4 Calling Model Methods

System is an orchestrator for calling shared methods of models. These API methods are defined for initialization, equation update, Jacobian update, and discrete flags update.

The following methods take an argument *models*, which should be an *OrderedDict* of models with names as keys and instances as values.

```
andes.system.System.init(self, models: collections.OrderedDict, routine: str)
```

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.

```
andes.system.System.e_clear(self, models: collections.OrderedDict)
```

Clear equation arrays in DAE and model variables.

This step must be called before calling *f\_update* or *g\_update* to flush existing values.

`andes.system.System.l_update_var(self, models: collections.OrderedDict)`  
 Update variable-based limiter discrete states by calling `l_update_var` of models.

This function is must be called before any equation evaluation.

`andes.system.System.f_update(self, models: collections.OrderedDict)`  
 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.

`andes.system.System.l_update_eq(self, models: collections.OrderedDict)`  
 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.

`andes.system.System.g_update(self, models: collections.OrderedDict)`  
 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.

`andes.system.System.j_update(self, models: collections.OrderedDict, info=None)`  
 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).

## 3.1.5 Configuration

System, models and routines have a member attribute `config` for model-specific or routine-specific configurations. System manages all configs, including saving to a config file and loading back.

`andes.system.System.get_config(self)`  
 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.

```
andes.system.System.save_config(self, file_path=None, overwrite=False)
```

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.

```
andes.system.System.load_config(conf_path=None)
```

Load config from an rc-formatted file.

#### Parameters

**conf\_path** [None or str] Path to the config file. If is *None*, the function body will not run.

#### Returns

`configparse.ConfigParser`

**Warning:** It is important to note that configs from files is passed to *model constructors* during instantiation. If one needs to modify config for a run, it needs to be done before instantiating `System`, or before running `andes` from command line. Directly modifying `Model.config` may not take effect or have side effect as for the current implementation.

## 3.2 Models

This section introduces the modeling of power system devices. The terminology "model" is used to describe the mathematical representation of a *type* of device, such as synchronous generators or turbine governors. The terminology "device" is used to describe a particular instance of a model, for example, a specific generator.

To define a model in ANDES, two classes, `ModelData` and `Model` need to be utilized. Class `ModelData` is used for defining parameters that will be provided from input files. It provides API for adding data from devices and managing the data. Class `Model` is used for defining other non-input parameters, service variables, and DAE variables. It provides API for converting symbolic equations, storing Jacobian patterns, and updating equations.

### 3.2.1 Model Data

```
class andes.core.model.ModelData(*args, **kwargs)
```

Class for holding parameter data for a model.

This class is designed to hold the parameter data separately from model equations. Models should inherit this class to define the parameters from input files.

Inherit this class to create the specific class for holding input parameters for a new model. The recommended name for the derived class is the model name with `Data`. For example, data for *GENCLS* should be named *GENCLSData*.

Parameters should be defined in the `__init__` function of the derived class.

Refer to `andes.core.param` for available parameter types.

## Notes

Two default parameters, *u* (connection status of type `andes.core.param.NumParam`), and *name* (device name of type `andes.core.param.DataParam`) are pre-defined in `ModelData`, and will be inherited by all models.

## Examples

If we want to build a class `PQData` (for static PQ load) with three parameters, *Vn*, *p0* and *q0*, we can use the following

```
from andes.core.model import ModelData, Model
from andes.core.param import IdxParam, NumParam

class PQData(ModelData):
    super().__init__()
    self.Vn = NumParam(default=110,
                        info="AC voltage rating",
                        unit='kV', non_zero=True,
                        tex_name=r'V_n')
    self.p0 = NumParam(default=0,
                        info='active power load in system base',
                        tex_name=r'p_0', unit='p.u.')
    self.q0 = NumParam(default=0,
                        info='reactive power load in system base',
                        tex_name=r'q_0', unit='p.u.')
```

In this example, all the three parameters are defined as `andes.core.param.NumParam`. In the full `PQData` class, other types of parameters also exist. For example, to store the idx of *owner*, `PQData` uses

```
self.owner = IdxParam(model='Owner', info="owner idx")
```

## Attributes

**cache** A cache instance for different views of the internal data.

**flags** [dict] Flags to control the routine and functions that get called. If the model is using user-defined numerical calls, set *f\_num*, *g\_num* and *j\_num* properly.

## Cache

*ModelData* uses a lightweight class `andes.core.model.Cache` for caching its data as a dictionary or a pandas DataFrame. Four attributes are defined in *ModelData.cache*:

- *dict*: all data in a dictionary with the parameter names as keys and *v* values as arrays.
- *dict\_in*: the same as *dict* except that the values are from *v\_in*, the original input.
- *df*: all data in a pandas DataFrame.
- *df\_in*: the same as *df* except that the values are from *v\_in*.

Other attributes can be added by registering with `cache.add_callback`.

`andes.core.model.Cache.add_callback(self, name: str, callback)`

Add a cache attribute and a callback function for updating the attribute.

### Parameters

**name** [str] name of the cached function return value

**callback** [callable] callback function for updating the cached attribute

## Define Voltage Ratings

If a model is connected to an AC Bus or a DC Node, namely, if *bus*, *bus1*, *node* or *node1* exists as parameter, it must provide the corresponding parameter, *Vn*, *Vn1*, *Vdcn* or *Vdcn1*, for rated voltages.

Controllers not connected to Bus or Node will have its rated voltages omitted and thus  $V_b = V_n = 1$ , unless one uses `andes.core.param.ExtParam` to retrieve the bus/node values.

As a rule of thumb, controllers not directly connected to the network shall use system-base per unit for voltage and current parameters. Controllers (such as a turbine governor) may inherit rated power from controlled models and thus power parameters will be converted consistently.

### 3.2.2 Define a DAE Model

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

Base class for power system DAE models.

After subclassing *ModelData*, subclass *Model* to complete a DAE model. Subclasses of *Model* defines DAE variables, services, and other types of parameters, in the constructor `__init__`.

### Examples

Take the static PQ as an example, the subclass of *Model*, *PQ*, should look like

```
class PQ(PQData, Model):
    def __init__(self, system, config):
        PQData.__init__(self)
        Model.__init__(self, system, config)
```

Since *PQ* is calling the base class constructors, it is meant to be the final class and not further derived. It inherits from *PQData* and *Model* and must call constructors in the order of *PQData* and *Model*. If the derived class of *Model* needs to be further derived, it should only derive from *Model* and use a name ending with *Base*. See `andes.models.synchronous.GENBASE`.

Next, in *PQ.\_\_init\_\_*, set proper flags to indicate the routines in which the model will be used

```
self.flags.update({'pflow': True})
```

Currently, flags *pflow* and *tds* are supported. Both are *False* by default, meaning the model is neither used in power flow nor time-domain simulation. **A very common pitfall is forgetting to set the flag.**

Next, the group name can be provided. A group is a collection of models with common parameters and variables. Devices *idx* of all models in the same group must be unique. To provide a group name, use

```
self.group = 'StaticLoad'
```

The group name must be an existing class name in `andes.models.group`. The model will be added to the specified group and subject to the variable and parameter policy of the group. If not provided with a group class name, the model will be placed in the *Undefined* group.

Next, additional configuration flags can be added. Configuration flags for models are load-time variables specifying the behavior of a model. It can be exported to an *andes.rc* file and automatically loaded when creating the *System*. Configuration flags can be used in equation strings, as long as they are numerical values. To add config flags, use

```
self.config.add(OrderedDict([('pq2z', 1), ]))
```

It is recommended to use *OrderedDict* instead of *dict*, although the syntax is verbose. Note that booleans should be provided as integers (1, or 0), since *True* or *False* is interpreted as a string when loaded from the *rc* file and will cause an error.

Next, it's time for variables and equations! The *PQ* class does not have internal variables itself. It uses its *bus* parameter to fetch the corresponding *a* and *v* variables of buses. Equation wise, it imposes an active power and a reactive power load equation.

To define external variables from *Bus*, use

```
self.a = ExtAlgeb(model='Bus', src='a',
                  indexer=self.bus, tex_name=r'\theta')
self.v = ExtAlgeb(model='Bus', src='v',
                  indexer=self.bus, tex_name=r'V')
```

Refer to the subsection Variables for more details.

The simplest *PQ* model will impose constant P and Q, coded as

```
self.a.e_str = "u * p"
self.v.e_str = "u * q"
```

where the *e\_str* attribute is the equation string attribute. *u* is the connectivity status. Any parameter, config, service or variables can be used in equation strings. An addition variable *dae\_t* for the current

simulation time can be used if the model has flag *tds*.

The above example is overly simplified. Our *PQ* model wants a feature to switch itself to a constant impedance if the voltage is out of the range (*vmin*, *vmax*). To implement this, we need to introduce a discrete component called *Limiter*, which yields three arrays of binary flags, *zi*, *zl*, and *zu* indicating in range, below lower limit, and above upper limit, respectively.

First, create an attribute *vcmp* as a *Limiter* instance

```
self.vcmp = Limiter(u=self.v, lower=self.vmin, upper=self.vmax,
                    enable=self.config.pq2z)
```

where *self.config.pq2z* is a flag to turn this feature on or off. After this line, we can use *vcmp\_zi*, *vcmp\_zl*, and *vcmp\_zu* in other equation strings.

```
self.a.e_str = "u * (p0 * vcmp_zi + " \
               "p0 * vcmp_zl * (v ** 2 / vmin ** 2) + " \
               "p0 * vcmp_zu * (v ** 2 / vmax ** 2)) "

self.v.e_str = "u * (q0 * vcmp_zi + " \
               "q0 * vcmp_zl * (v ** 2 / vmin ** 2) + "\
               "q0 * vcmp_zu * (v ** 2 / vmax ** 2)) "
```

Note that *PQ.a.e\_str* can use the three variables from *vcmp* even before defining *PQ.vcmp*, as long as *PQ.vcmp* is defined, because *vcmp\_zi* is just a string literal in *e\_str*.

The two equations above implements a piecewise power injection equation. It selects the original power demand if within range, and uses the calculated power when out of range.

Finally, to let ANDES pick up the model, the model name needs to be added to *models/\_\_init\_\_.py*. Follow the examples in the *OrderedDict*, where the key is the file name, and the value is the class name.

### Attributes

**num\_params** [OrderedDict] {name: instance} of numerical parameters, including internal and external ones

## 3.2.3 Dynamicity Under the Hood

The magic for automatic creation of variables are all hidden in *andes.core.model.Model.\_\_setattr\_\_()*, and the code is incredible simple. It sets the name, *tex\_name*, and owner model of the attribute instance and, more importantly, does the book keeping. In particular, when the attribute is a *andes.core.block.Block* subclass, *\_\_setattr\_\_* captures the exported instances, recursively, and prepends the block name to exported ones. All these convenience owe to the dynamic feature of Python.

During the code generation phase, the symbols are created by checking the book-keeping attributes, such as *states*, *algebs*, and attributes in *Model.cache*.

In the numerical evaluation phase, *Model* provides a method, *andes.core.model.get\_inputs()*, to collect the variable value arrays in a dictionary, which can be effortlessly passed as arguments to numerical functions.



## Commonly Used Attributes in Models

The following `Model` attributes are commonly used for debugging. If the attribute is an *OrderedDict*, the keys are attribute names in `str`, and corresponding values are the instances.

- `params` and `params_ext`, two *OrderedDict* for internal (both numerical and non-numerical) and external parameters, respectively.
- `num_params` for numerical parameters, both internal and external.
- `states` and `algebs`, two *OrderedDict* for state variables and algebraic variables, respectively.
- `states_ext` and `algebs_ext`, two *OrderedDict* for external states and algebraics.
- `discrete`, an *OrderedDict* for discrete components.
- `blocks`, an *OrderedDict* for blocks.
- `services`, an *OrderedDict* for services with `v_str`.
- `services_ext`, an *OrderedDict* for externally retrieved services.

## Attributes in *Model.cache*

Attributes in *Model.cache* are additional book-keeping structures for variables, parameters and services. The following attributes are defined.

- `all_vars`: all the variables.
- `all_vars_names`, a list of all variable names.
- `all_params`, all parameters.
- `all_params_names`, a list of all parameter names.
- `algebs_and_ext`, an *OrderedDict* of internal and external algebraic variables.
- `states_and_ext`, an *OrderedDict* of internal and external differential variables.
- `services_and_ext`, an *OrderedDict* of internal and external service variables.
- `vars_int`, an *OrderedDict* of all internal variables, states and then algebs.
- `vars_ext`, an *OrderedDict* of all external variables, states and then algebs.

### 3.2.4 Equation Generation

`Model.syms`, an instance of `SymProcessor`, handles the symbolic to numeric generation when called. The equation generation is a multi-step process with symbol preparation, equation generation, Jacobian generation, initializer generation, and pretty print generation.

**class** `andes.core.model.SymProcessor` (*parent*)

A helper class for symbolic processing and code generation.

#### Parameters

**parent** [Model] The *Model* instance to document

#### Attributes

**xy** [sympy.Matrix] variables pretty print in the order of State, ExtState, Algeb, ExtAlgeb

**f** [sympy.Matrix] differential equations pretty print

**g** [sympy.Matrix] algebraic equations pretty print

**df** [sympy.SparseMatrix]  $df/d(xy)$  pretty print

**dg** [sympy.SparseMatrix]  $dg/d(xy)$  pretty print

**inputs\_dict** [OrderedDict] All possible symbols in equations, including variables, parameters, discrete flags, and config flags. It has the same variables as what `get_inputs()` returns.

**vars\_dict** [OrderedDict] variable-only symbols, which are useful when getting the Jacobian matrices.

**non\_vars\_dict** [OrderedDict] symbols in `input_syms` but not in `var_syms`.

#### **generate\_init()**

Generate lambda functions for initial values.

#### **generate\_jacobians()**

Generate Jacobians and store to corresponding triplets.

The internal indices of equations and variables are stored, alongside the lambda functions.

For example,  $dg/dy$  is a sparse matrix whose elements are `(row, col, val)`, where `row` and `col` are the internal indices, and `val` is the numerical lambda function. They will be stored to

`row -> self.calls._igy col -> self.calls._jgy val -> self.calls._vgy`

#### **generate\_symbols()**

Generate symbols for symbolic equation generations.

This function should run before other generate equations.

#### Attributes

**inputs\_dict** [OrderedDict] name-symbol pair of all parameters, variables and configs

**vars\_dict** [OrderedDict] name-symbol pair of all variables, in the order of `(states_and_ext + algebs_and_ext)`

**non\_vars\_dict** [OrderedDict] name-symbol pair of all non-variables, namely, `(inputs_dict - vars_dict)`

Next, function `generate_equation` converts each DAE equation set to one numerical function calls and store it in `Model.calls`. The attributes for differential equation set and algebraic equation set are `f` and `g`. Differently, service variables will be generated one by one and store in an `OrderedDict` in `Model.calls.s`.

### 3.2.5 Jacobian Storage

#### Abstract Jacobian Storage

Using the `.jacobian` method on `sympy.Matrix`, the symbolic Jacobians can be easily obtained. The complexity lies in the storage of the Jacobian elements. Observed that the Jacobian equation generation happens before any system is loaded, thus only the variable indices in the variable array is available. For each non-zero item in each Jacobian matrix, ANDES stores the equation index, variable index, and the Jacobian value (either a constant number or a callable function returning an array).

Note that, again, a non-zero entry in a Jacobian matrix can be either a constant or an expression. For efficiency, constant numbers and lambdified callables are stored separately. Constant numbers, therefore, can be loaded into the sparse matrix pattern when a particular system is given.

**Warning:** Data structure for the Jacobian storage has changed. Pending documentation update. Please check `andes.core.common.JacTriplet` class for more details.

The triplets, the equation (row) index, variable (column) index, and values (constant numbers or callable) are stored in `Model` attributes with the name of `_{i, j, v}{Jacobian Name}{c or None}`, where `{i, j, v}` is a single character for row, column or value, `{Jacobian Name}` is a two-character Jacobian name chosen from `fx`, `fy`, `gx`, and `gy`, and `{c or None}` is either character `c` or no character, indicating whether it corresponds to the constants or non-constants in the Jacobian.

For example, the triplets for the constants in Jacobian `gy` are stored in `_igyc`, `_jgyc`, and `_vgyc`.

In terms of the non-constant entries in Jacobians, the callable functions are stored in the corresponding `_v{Jacobian Name}` array. Note the differences between, for example, `_vgy` and `_vgyc`: `_vgy` is a list of callables, while `_vgyc` is a list of constant numbers.

#### Concrete Jacobian Storage

When a specific system is loaded and the addresses are assigned to variables, the abstract Jacobian triplets, more specifically, the rows and columns, are replaced with the array of addresses. The new addresses and values will be stored in `Model` attributes with the names `{i, j, v}{Jacobian Name}{c or None}`. Note that there is no underscore for the concrete Jacobian triplets.

For example, if model `PV` has a list of variables `[p, q, a, v]`. The equation associated with `p` is  $-u * p_0$ , and the equation associated with `q` is  $u * (v_0 - v)$ . Therefore, the derivative of equation  $v_0 - v$  over `v` is  $-u$ . Note that `u` is unknown at generation time, thus the value is NOT a constant and should to go `vgy`.

The values in `_igy`, `_jgy` and `_vgy` contains, respectively, 1, 3, and a lambda function which returns  $-u$ .

When a specific system is loaded, for example, a 5-bus system, the addresses for the `q` and `v` are `[11, 13, 15]`, and `[5, 7, 9]`. `PV.igy` and `PV.jgy` will thus query the corresponding address list based on `PV._igy` and `PV._jgy` and store `[11, 13, 15]`, and `[5, 7, 9]`.

### 3.2.6 Initialization

Value providers such as services and DAE variables need to be initialized. Services are initialized before any DAE variable. Both Services and DAE Variables are initialized *sequentially* in the order of declaration.

Each Service, in addition to the standard `v_str` for symbolic initialization, provides a `v_numeric` hook for specifying a custom function for initialization. Custom initialization functions for DAE variables, are lumped in a single function in `Model.v_numeric`.

ANDES has an *experimental* Newton-Krylov method based iterative initialization. All DAE variables with `v_iter` will be initialized using the iterative approach

### 3.2.7 Additional Numerical Equations

Addition numerical equations are allowed to complete the "hybrid symbolic-numeric" framework. Numerical function calls are useful when the model DAE is non-standard or hard to be generalized. Since the symbolic-to-numeric generation is an additional layer on top of the numerical simulation, it is fundamentally the same as user-provided numerical function calls.

ANDES provides the following hook functions in each `Model` subclass for custom numerical functions:

- `v_numeric`: custom initialization function
- `s_numeric`: custom service value function
- `g_numeric`: custom algebraic equations; update the `e` of the corresponding variable.
- `f_numeric`: custom differential equations; update the `e` of the corresponding variable.
- `j_numeric`: custom Jacobian equations; the function should append to `_i`, `_j` and `_v` structures.

For most models, numerical function calls are unnecessary and not recommended as it increases code complexity. However, when the data structure or the DAE are difficult to generalize in the symbolic framework, the numerical equations can be used.

For interested readers, see the `COI` symbolic implementation which calculated the center-of-inertia speed of generators. The `COI` could have been implemented numerically with for loops instead of `NumReduce`, `NumRepeat` and external variables.

## 3.3 Atom Types

ANDES contains three types of atom classes for building DAE models. These types are parameter, variable and service.

### 3.3.1 Value Provider

Before addressing specific atom classes, the terminology *v-provider*, and *e-provider* are discussed. A value provider class (or *v-provider* for short) references any class with a member attribute named `v`, which should be a list or a 1-dimensional array of values. For example, all parameter classes are v-providers, since a parameter class should provide values for that parameter.

---

**Note:** In fact, all types of atom classes are *v*-providers, meaning that an instance of an atom class must contain values.

---

The values in the *v* attribute of a particular instance are values that will substitute the instance for computation. If in a model, one has a parameter

```
self.v0 = NumParam()
self.b = NumParam()

# where self.v0.v = np.array([1., 1.05, 1.1]
#   and self.b.v = np.array([10., 10., 10.]
```

Later, this parameter is used in an equation, such as

```
self.v = ExtAlgeb(model='Bus', src='v',
                  indexer=self.bus,
                  e_str='v0 **2 * b')
```

While computing  $v0 ** 2 * b$ , *v0* and *b* will be substituted with the values in *self.v0.v* and *self.b.v*.

Sharing this interface *v* allows interoperability among parameters and variables and services. In the above example, if one defines *v0* as a *ConstService* instance, such as

```
self.v0 = ConstService(v_str='1.0')
```

Calculations will still work without modification.

### 3.3.2 Equation Provider

Similarly, an equation provider class (or *e-provider*) references any class with a member attribute named *e*, which should be a 1-dimensional array of values. The values in the *e* array are the results from the equation and will be summed to the numerical DAE at the addresses specified by the attribute *a*.

---

**Note:** Currently, only variables are *e-provider* types.

---

If a model has an external variable that links to *Bus.v* (voltage), such as

```
self.v = ExtAlgeb(model='Bus', src='v',
                  indexer=self.bus,
                  e_str='v0 **2 * b')
```

The addresses of the corresponding voltage variables will be retrieved into *self.a*, and the equation evaluation results will be stored in *self.v.e*

## 3.4 Parameters

### 3.4.1 Background

Parameter is a type of building atom for DAE models. Most parameters are read directly from an input file and passed to equation, and other parameters can be calculated from existing parameters.

The base class for parameters in ANDES is *BaseParam*, which defines interfaces for adding values and checking the number of values. *BaseParam* has its values stored in a plain list, the member attribute *v*. Subclasses such as *NumParam* stores values using a NumPy ndarray.

An overview of supported parameters is given below.

Subclasses	Description
DataParam	An alias of <i>BaseParam</i> . Can be used for any non-numerical parameters.
NumParam	The numerical parameter type. Used for all parameters in equations
IdxParam	The parameter type for storing <i>idx</i> into other models
ExtParam	Externally defined parameter
TimerParam	Parameter for storing the action time of events

### 3.4.2 Data Parameters

```
class andes.core.param.BaseParam(default: Union[float, str, int, None] = None, name:
                                Optional[str] = None, tex_name: Optional[str]
                                = None, info: Optional[str] = None, unit: Op-
                                tional[str] = None, mandatory: bool = False, ex-
                                port: bool = True)
```

The base parameter class.

This class provides the basic data structure and interfaces for all types of parameters. Parameters are from input files and in general constant once initialized.

Subclasses should overload the *n()* method for the total count of elements in the value array.

#### Parameters

**default** [str or float, optional] The default value of this parameter if None is provided

**name** [str, optional] Parameter name. If not provided, it will be automatically set to the attribute name defined in the owner model.

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

**info** [str, optional] Descriptive information of parameter

**mandatory** [bool] True if this parameter is mandatory

**export** [bool] True if the parameter will be exported when dumping data into files. True for most parameters. False for *BackRef*.

**Warning:** The most distinct feature of `BaseParam`, `DataParam` and `IdxParam` is that values are stored in a list without conversion to array. `BaseParam`, `DataParam` or `IdxParam` are **not allowed** in equations.

### Attributes

**v** [list] A list holding all the values. The `BaseParam` class does not convert the `v` attribute into NumPy arrays.

**property** [dict] A dict containing the truth values of the model properties.

```
class andes.core.param.DataParam(default: Union[float, str, int, None] = None, name:
                                Optional[str] = None, tex_name: Optional[str] =
                                None, info: Optional[str] = None, unit: Op-
                                tional[str] = None, mandatory: bool = False, ex-
                                port: bool = True)
```

An alias of the `BaseParam` class.

This class is used for string parameters or non-computational numerical parameters. This class does not provide a `to_array` method. All input values will be stored in `v` as a list.

See also:

[`andes.core.param.BaseParam`](#) Base parameter class

```
class andes.core.param.IdxParam(default: Union[float, str, int, None] = None, name:
                                Optional[str] = None, tex_name: Optional[str] =
                                None, info: Optional[str] = None, unit: Op-
                                tional[str] = None, mandatory: bool = False,
                                unique: bool = False, export: bool = True, model:
                                Optional[str] = None)
```

An alias of `BaseParam` with an additional storage of the owner model name

This class is intended for storing `idx` into other models. It can be used in the future for data consistency check.

### Notes

This will be useful when, for example, one connects two TGs to one SynGen.

### Examples

A PQ model connected to Bus model will have the following code

```
class PQModel(...):
    def __init__(...):
        ...
        self.bus = IdxParam(model='Bus')
```

### 3.4.3 Numeric Parameters

```
class andes.core.param.NumParam (default: Union[float, str, Callable, None] = None,
                                name: Optional[str] = None, tex_name: Op-
                                tional[str] = None, info: Optional[str] = None, unit:
                                Optional[str] = None, vrange: Union[List[T], Tu-
                                ple, None] = None, non_zero: bool = False, posi-
                                tive: bool = False, mandatory: bool = False, power:
                                bool = False, ipower: bool = False, voltage: bool =
                                False, current: bool = False, z: bool = False, y: bool
                                = False, r: bool = False, g: bool = False, dc_voltage:
                                bool = False, dc_current: bool = False, export: bool
                                = True)
```

A computational numerical parameter.

Parameters defined using this class will have their *v* field converted to a NumPy array after adding. The original input values will be copied to *vin*, and the system-base per-unit conversion coefficients (through multiplication) will be stored in *pu\_coeff*.

#### Parameters

**default** [str or float, optional] The default value of this parameter if no value is provided

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

**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

**mandatory** [bool] True if this parameter is mandatory

**unit** [str, optional] Unit of the parameter

**vrange** [list, tuple, optional] Typical value range

#### Other Parameters

**Sn** [str] Name of the parameter for the device base power.

**Vn** [str] Name of the parameter for the device base voltage.

**non\_zero** [bool] True if this parameter must be non-zero

**positive: bool** True if this parameter must be positive

**mandatory** [bool] True if this parameter must not be None

**power** [bool] True if this parameter is a power per-unit quantity under the device base

**ipower** [bool] True if this parameter is an inverse-power per-unit quantity under the device base

**voltage** [bool] True if the parameter is a voltage pu quantity under the device base



**current** [bool] True if the parameter is a current pu quantity under the device base

**z** [bool] True if the parameter is an AC impedance pu quantity under the device base

**y** [bool] True if the parameter is an AC admittance pu quantity under the device base

**r** [bool] True if the parameter is a DC resistance pu quantity under the device base

**g** [bool] True if the parameter is a DC conductance pu quantity under the device base

**dc\_current** [bool] True if the parameter is a DC current pu quantity under device base

**dc\_voltage** [bool] True if the parameter is a DC voltage pu quantity under device base

### 3.4.4 External Parameters

**class** `andes.core.param.ExtParam`(*model: str, src: str, indexer=None, dtype=<class 'float'>, allow\_none=False, default=0.0, \*\*kwargs*)

A parameter whose values are retrieved from an external model or group.

#### Parameters

**model** [str] Name of the model or group providing the original parameter

**src** [str] The source parameter name

**indexer** [BaseParam] A parameter defined in the model defining this ExtParam instance. *indexer.v* should contain indices into *model.src.v*. If is None, the source parameter values will be fully copied. If *model* is a group name, the indexer cannot be None.

#### Attributes

**parent\_model** [Model] The parent model providing the original parameter.

### 3.4.5 Timer Parameter

**class** `andes.core.param.TimerParam`(*callback: Optional[Callable] = None, default: Union[float, str, Callable, None] = None, name: Optional[str] = None, tex\_name: Optional[str] = None, info: Optional[str] = None, unit: Optional[str] = None, non\_zero: bool = False, mandatory: bool = False, export: bool = True*)

A parameter whose values are event occurrence times during the simulation.

The constructor takes an additional Callable *self.callback* for the action of the event. *TimerParam* has a default value of -1, meaning deactivated.

## Examples

A connectivity status toggler class *Toggler* takes a parameter  $t$  for the toggle time. Inside *Toggler*.`__init__`, one would have

```
self.t = TimerParam()
```

The *Toggler* class also needs to define a method for toggling the connectivity status

```
def _u_switch(self, is_time: np.ndarray):
    action = False
    for i in range(self.n):
        if is_time[i] and (self.u.v[i] == 1):
            instance = self.system.__dict__[self.model.v[i]]
            # get the original status and flip the value
            u0 = instance.get(src='u', attr='v', idx=self.dev.v[i])
            instance.set(src='u',
                        attr='v',
                        idx=self.dev.v[i],
                        value=1-u0)
        action = True
    return action
```

Finally, in *Toggler*.`__init__`, assign the function as the callback for *self.t*

```
self.t.callback = self._u_switch
```

## 3.5 Variables

DAE Variables, or variables for short, are unknowns to be solved using numerical or analytical methods. A variable stores values, equation values, and addresses in the DAE array. The base class for variables is *BaseVar*. In this subsection, *BaseVar* is used to represent any subclass of *VarBase* list in the table below.

Class	Description
State	A state variable and associated diff. equation $T\dot{x} = f$
Algeb	An algebraic variable and an associated algebraic equation $0 = g$
ExtState	An external state variable and part of the differential equation (uncommon)
ExtAlgeb	An external algebraic variable and part of the algebraic equation

*BaseVar* has two types: the differential variable type *State* and the algebraic variable type *Algeb*. State variables are described by differential equations, whereas algebraic variables are described by algebraic equations. State variables can only change continuously, while algebraic variables can be discontinuous.

Based on the model the variable is defined, variables can be internal or external. Most variables are internal and only appear in equations in the same model. Some models have "public" variables that can be accessed by other models. For example, a *Bus* defines  $v$  for the voltage magnitude. Each device attached to a particular bus needs to access the value and impose the reactive power injection. It can be done with *ExtAlgeb* or *ExtState*, which links with an existing variable from a model or a group.

### 3.5.1 Variable, Equation and Address

Subclasses of *BaseVar* are value providers and equation providers. Each *BaseVar* has member attributes  $v$  and  $e$  for variable values and equation values, respectively. The initial value of  $v$  is set by the initialization routine, and the initial value of  $e$  is set to zero. In the process of power flow calculation or time domain simulation,  $v$  is not directly modifiable by models but rather updated after solving non-linear equations.  $e$  is updated by the models and summed up before solving equations.

Each *BaseVar* also stores addresses of this variable, for all devices, in its member attribute  $a$ . The addresses are 0-based indices into the numerical DAE array,  $f$  or  $g$ , based on the variable type.

For example, *Bus* has `self.a = Algeb()` as the voltage phase angle variable. For a 5-bus system, `Bus.a.a` stores the addresses of the  $a$  variable for all the five *Bus* devices. Conventionally, `Bus.a.a` will be assigned `np.array([0, 1, 2, 3, 4])`.

### 3.5.2 Value and Equation Strings

The most important feature of the symbolic framework is allowing to define equations using strings. There are three types of strings for a variable, stored in the following member attributes, respectively:

- $v\_str$ : equation string for **explicit** initialization in the form of  $v = v\_str(x, y)$ .
- $v\_iter$ : equation string for **implicit** initialization in the form of  $v\_iter(x, y) = 0$
- $e\_str$ : equation string for (full or part of) the differential or algebraic equation.

The difference between  $v\_str$  and  $v\_iter$  should be clearly noted.  $v\_str$  evaluates directly into the initial value, while all  $v\_iter$  equations are solved numerically using the Newton-Krylov iterative method.

### 3.5.3 Values Between DAE and Models

ANDES adopts a decentralized architecture which provides each model a copy of variable values before equation evaluation. This architecture allows to parallelize the equation evaluation (in theory, or in practice if one works round the Python GIL). However, this architecture requires a coherent protocol for updating the DAE arrays and the *BaseVar* arrays. More specifically, how the variable and equations values from model *VarBase* should be summed up or forcefully set at the DAE arrays needs to be defined.

The protocol is relevant when a model defines subclasses of *BaseVar* that are supposed to be "public". Other models share this variable with *ExtAlgeb* or *ExtState*.

By default, all  $v$  and  $e$  at the same address are summed up. This is the most common case, such as a *Bus* connected by multiple devices: power injections from devices should be summed up.

In addition, *BaseVar* provides two flags,  $v\_setter$  and  $e\_setter$ , for cases when one *VarBase* needs to overwrite the variable or equation values.

### 3.5.4 Flags for Value Overwriting

*BaseVar* have special flags for handling value initialization and equation values. This is only relevant for public or external variables. The  $v\_setter$  is used to indicate whether a particular *BaseVar* instance sets the

initial value. The *e\_setter* flag indicates whether the equation associated with a *BaseVar* sets the equation value.

The *v\_setter* flag is checked when collecting data from models to the numerical DAE array. If *v\_setter* is *False*, variable values of the same address will be added. If one of the variable or external variable has *v\_setter* is *True*, it will, at the end, set the values in the DAE array to its value. Only one *BaseVar* of the same address is allowed to have *v\_setter* == *True*.

### 3.5.5 A *v\_setter* Example

A Bus is allowed to default the initial voltage magnitude to 1 and the voltage phase angle to 0. If a PV device is connected to a Bus device, the PV should be allowed to override the voltage initial value with the voltage set point.

In *Bus.\_\_init\_\_()*, one has

```
self.v = Algeb(v_str='1')
```

In *PV.\_\_init\_\_*, one can use

```
self.v0 = Param()
self.bus = IdxParam(model='Bus')

self.v = ExtAlgeb(src='v',
                  model='Bus',
                  indexer=self.bus,
                  v_str='v0',
                  v_setter=True)
```

where an *ExtAlgeb* is defined to access *Bus.v* using indexer *self.bus*. The *v\_str* line sets the initial value to *v0*. In the variable initialization phase for *PV*, *PV.v.v* is set to *v0*.

During the value collection into *DAE.y* by the *System* class, *PV.v*, as a final *v\_setter*, will overwrite the voltage magnitude for Bus devices with the indices provided in *PV.bus*.

```
class andes.core.var.BaseVar (name: Optional[str] = None, tex_name: Optional[str] =
                             None, info: Optional[str] = None, unit: Optional[str] =
                             None, v_str: Union[str, float, None] = None, v_iter: Op-
                             tional[str] = None, e_str: Optional[str] = None, discrete:
                             Optional[andes.core.discrete.Discrete] = None, v_setter:
                             Optional[bool] = False, e_setter: Optional[bool] =
                             False, addressable: Optional[bool] = True, export: Op-
                             tional[bool] = True, diag_eps: Optional[float] = 0.0)
```

Base variable class.

Derived classes *State* and *Algeb* should be used to build model variables.

#### Parameters

**name** [str, optional] Variable name

**info** [str, optional] Descriptive information

**unit** [str, optional] Unit

**tex\_name** [str] LaTeX-formatted variable name. If is None, use *name* instead.

**discrete** [Discrete] Associated discrete component. Will call *check\_var* on the discrete component.

#### Attributes

**a** [array-like] variable address

**v** [array-like] local-storage of the variable value

**e** [array-like] local-storage of the corresponding equation value

**e\_str** [str] the string/symbolic representation of the equation

```
class andes.core.var.ExtVar(model: str, src: str, indexer: Union[List[T],
    numpy.ndarray, andes.core.param.BaseParam, andes.core.service.BaseService, None] = None, allow_none:
    Optional[bool] = False, name: Optional[str] = None,
    tex_name: Optional[str] = None, info: Optional[str]
    = None, unit: Optional[str] = None, v_str: Union[str,
    float, None] = None, v_iter: Optional[str] = None,
    e_str: Optional[str] = None, v_setter: Optional[bool]
    = False, e_setter: Optional[bool] = False, addressable:
    Optional[bool] = True, export: Optional[bool] = True,
    diag_eps: Optional[float] = 0.0)
```

Externally defined algebraic variable

This class is used to retrieve the addresses of externally- defined variable. The *e* value of the *ExtVar* will be added to the corresponding address in the DAE equation.

#### Parameters

**model** [str] Name of the source model

**src** [str] Source variable name

**indexer** [BaseParam, BaseService] A parameter of the hosting model, used as indices into the source model and variable. If is None, the source variable address will be fully copied.

**allow\_none** [bool] True to allow None in indexer

#### Attributes

**parent\_model** [Model] The parent model providing the original parameter.

**uid** [array-like] An array containing the absolute indices into the parent\_instance values.

**e\_code** [str] Equation code string; copied from the parent instance.

**v\_code** [str] Variable code string; copied from the parent instance.

```
class andes.core.var.State (name: Optional[str] = None, tex_name: Optional[str]
    = None, info: Optional[str] = None, unit: Optional[str]
    = None, v_str: Union[str, float, None] = None, v_iter:
    Optional[str] = None, e_str: Optional[str] = None,
    discrete: Optional[andes.core.discrete.Discrete] =
    None, t_const: Union[andes.core.param.BaseParam,
    andes.core.common.DummyValue,
    andes.core.service.BaseService, None] = None, v_setter:
    Optional[bool] = False, e_setter: Optional[bool] = False,
    addressable: Optional[bool] = True, export: Optional[bool]
    = True, diag_eps: Optional[float] = 0.0)
```

Differential variable class, an alias of the *BaseVar*.

#### Parameters

**t\_const** [BaseParam, DummyValue] Left-hand time constant for the differential equation. Time constants will not be evaluated as part of the differential equation. They will be collected to array *dae.Tf* to multiply to the right-hand side *dae.f*.

#### Attributes

**e\_code** [str] Equation code string, equals string literal *f*

**v\_code** [str] Variable code string, equals string literal *x*

```
class andes.core.var.Algeb (name: Optional[str] = None, tex_name: Optional[str] =
    None, info: Optional[str] = None, unit: Optional[str] =
    None, v_str: Union[str, float, None] = None, v_iter: Op-
    tional[str] = None, e_str: Optional[str] = None, discrete:
    Optional[andes.core.discrete.Discrete] = None, v_setter:
    Optional[bool] = False, e_setter: Optional[bool] = False,
    addressable: Optional[bool] = True, export: Optional[bool]
    = True, diag_eps: Optional[float] = 0.0)
```

Algebraic variable class, an alias of the *BaseVar*.

#### Attributes

**e\_code** [str] Equation code string, equals string literal *g*

**v\_code** [str] Variable code string, equals string literal *y*

```
class andes.core.var.ExtState (model: str, src: str, indexer: Union[List[T],
    numpy.ndarray, andes.core.param.BaseParam, an-
    des.core.service.BaseService, None] = None, al-
    low_none: Optional[bool] = False, name: Op-
    tional[str] = None, tex_name: Optional[str] = None,
    info: Optional[str] = None, unit: Optional[str] =
    None, v_str: Union[str, float, None] = None, v_iter:
    Optional[str] = None, e_str: Optional[str] = None,
    v_setter: Optional[bool] = False, e_setter: Op-
    tional[bool] = False, addressable: Optional[bool]
    = True, export: Optional[bool] = True, diag_eps:
    Optional[float] = 0.0)
```

External state variable type.

**Warning:** `ExtState` is not allowed to set `t_const`, as it will conflict with the source `State` variable. In fact, one should not set `e_str` for `ExtState`.

```
class andes.core.var.ExtAlgeb(model: str, src: str, indexer: Union[List[T],
    numpy.ndarray, andes.core.param.BaseParam, andes.core.service.BaseService, None] = None, allow_none: Optional[bool] = False, name: Optional[str] = None, tex_name: Optional[str] = None, info: Optional[str] = None, unit: Optional[str] = None, v_str: Union[str, float, None] = None, v_iter: Optional[str] = None, e_str: Optional[str] = None, v_setter: Optional[bool] = False, e_setter: Optional[bool] = False, addressable: Optional[bool] = True, export: Optional[bool] = True, diag_eps: Optional[float] = 0.0)
```

External algebraic variable type.

```
class andes.core.var.AliasState(var, **kwargs)
    Alias state variable.
```

Refer to the docs of `AliasAlgeb`.

```
class andes.core.var.AliasAlgeb(var, **kwargs)
    Alias algebraic variable. Essentially ExtAlgeb that links to a model's own variable.
```

`AliasAlgeb` is useful when the final output of a model is from a block, but the model must provide the final output in a pre-defined name. Using `AliasAlgeb`, A model can avoid adding an additional variable with a dummy equations.

Like `ExtVar`, labels of `AliasAlgeb` will not be saved in the final output. When plotting from file, one need to look up the original variable name.

## 3.6 Services

Services are helper variables outside the DAE variable list. Services are most often used for storing intermediate constants but can be used for special operations to work around restrictions in the symbolic framework. Services are value providers, meaning each service has an attribute `v` for storing service values. The base class of services is `BaseService`, and the supported services are listed as follows.

Class	Description
ConstService	Internal service for constant values.
VarService	Variable service updated at each iteration before equations.
ExtService	External service for retrieving values from value providers.
PostInitService	Constant service evaluated after TDS initialization
NumReduce	The service type for reducing linear 2-D arrays into 1-D arrays
NumRepeat	The service type for repeating a 1-D array to linear 2-D arrays
IdxRepeat	The service type for repeating a 1-D list to linear 2-D list
EventFlag	Service type for flagging changes in inputs as an event
VarHold	Hold input value when a hold signal is active
ExtendedEvent	Extend an event signal for a given period of time
DataSelect	Select optional str data if provided or use the fallback
NumSelect	Select optional numerical data if provided
DeviceFinder	Finds or creates devices linked to the given devices
BackRef	Collects idx-es for the backward references
RefFlatten	Converts BackRef list of lists into a 1-D list
InitChecker	Checks initial values against typical values
FlagValue	Flags values that equals the given value
Replace	Replace values that returns True for the given lambda func

### 3.6.1 Internal Constants

The most commonly used service is *ConstService*. It is used to store an array of constants, whose value is evaluated from a provided symbolic string. They are only evaluated once in the model initialization phase, ahead of variable initialization. *ConstService* comes handy when one wants to calculate intermediate constants from parameters.

For example, a turbine governor has a *NumParam* *R* for the droop. *ConstService* allows to calculate the inverse of the droop, the gain, and use it in equations. The snippet from a turbine governor's `__init__()` may look like

```
self.R = NumParam()
self.G = ConstService(v_str='u/R')
```

where *u* is the online status parameter. The model can thus use *G* in subsequent variable or equation strings.

```
class andes.core.service.ConstService (v_str: Optional[str] = None, v_numeric:
                                     Optional[Callable] = None, vtype: Op-
                                     tional[type] = None, name: Optional[str] =
                                     None, tex_name=None, info=None)
```

A type of Service that stays constant once initialized.

ConstService are usually constants calculated from parameters. They are only evaluated once in the initialization phase before variables are initialized. Therefore, uninitialized variables must not be used in *v\_str*.

#### Parameters

**name** [str] Name of the ConstService



**v\_str** [str] An equation string to calculate the variable value.

**v\_numeric** [Callable, optional] A callable which returns the value of the ConstService

### Attributes

**v** [array-like or a scalar] ConstService value

```
class andes.core.service.VarService(v_str: Optional[str] = None, v_numeric:
Optional[Callable] = None, vtype: Optional[type] = None, name: Optional[str] =
None, tex_name=None, info=None)
```

Variable service that gets updated in each step/loop as variables change.

This class is useful when one has non-differentiable algebraic equations, which make use of *abs()*, *re* and *im*. Instead of creating *Algeb*, one can put the equation in *VarService*, which will be updated before solving algebraic equations.

**Warning:** *VarService* is not solved with other algebraic equations, meaning that there is one step "delay" between the algebraic variables and *VarService*. Use an algebraic variable whenever possible.

### Examples

In ESST3A model, the voltage and current sensors ( $v_d + jv_q$ ), ( $I_d + jI_q$ ) estimate the sensed VE using equation

$$VE = |K_{PC} * (v_d + 1jv_q) + 1j(K_I + K_{PC} * X_L) * (I_d + 1jI_q)|$$

One can use *VarService* to implement this equation

```
self.VE = VarService(tex_name='V_E',
                    info='VE',
                    v_str='Abs(KPC*(vd + 1j*vq) + 1j*(KI + KPC*XL)*(Id_
↵+ 1j*Iq))',
                    )
```

```
class andes.core.service.PostInitService(v_str: Optional[str] = None,
v_numeric: Optional[Callable]
= None, vtype: Optional[type] =
None, name: Optional[str] = None,
tex_name=None, info=None)
```

Constant service that gets stored once after init.

This service is useful when one need to store initialization values stored in variables.

## Examples

In ESST3A model, the  $vf$  variable is initialized followed by other variables. One can store the initial  $vf$  into  $vf0$  so that equation  $vf - vf0 = 0$  will hold.

```
self.vref0 = PostInitService(info='Initial reference voltage input',
                             tex_name='V_{ref0}',
                             v_str='vref',
                             )
```

Since all *ConstService* are evaluated before equation evaluation, without using *PostInitService*, one will need to create lots of *ConstService* to store values in the initialization path towards  $vf0$ , in order to correctly initialize  $vf$ .

### 3.6.2 External Constants

Service constants whose value is retrieved from an external model or group. Using *ExtService* is similar to using external variables. The values of *ExtService* will be retrieved once during the initialization phase before *ConstService* evaluation.

For example, a synchronous generator needs to retrieve the  $p$  and  $q$  values from static generators for initialization. *ExtService* is used for this purpose. In the `__init__()` of a synchronous generator model, one can define the following to retrieve *StaticGen.p* as  $p0$ :

```
self.p0 = ExtService(src='p',
                     model='StaticGen',
                     indexer=self.gen,
                     tex_name='P_0')
```

```
class andes.core.service.ExtService(model: str, src: str, indexer:
                                     Union[andes.core.param.BaseParam,
                                     andes.core.service.BaseService], attr: str =
                                     'v', allow_none: bool = False, default=0,
                                     name: str = None, tex_name: str = None,
                                     vtype=None, info: str = None)
```

Service constants whose value is from an external model or group.

#### Parameters

**src** [str] Variable or parameter name in the source model or group

**model** [str] A model name or a group name

**indexer** [IdxParam or BaseParam] An "Indexer" instance whose `v` field contains the `idx` of devices in the model or group.

## Examples

A synchronous generator needs to retrieve the  $p$  and  $q$  values from static generators for initialization. *ExtService* is used for this purpose.

In a synchronous generator, one can define the following to retrieve `StaticGen.p` as `p0`:

```
class GENCLSMModel(Model):
    def __init__(...):
        ...
        self.p0 = ExtService(src='p',
                             model='StaticGen',
                             indexer=self.gen,
                             tex_name='P_0')
```

### 3.6.3 Shape Manipulators

This section is for advanced model developer.

All generated equations operate on 1-dimensional arrays and can use algebraic calculations only. In some cases, one model would use *BackRef* to retrieve 2-dimensional indices and will use such indices to retrieve variable addresses. The retrieved addresses usually has a different length of the referencing model and cannot be used directly for calculation. Shape manipulator services can be used in such case.

*NumReduce* is a helper Service type which reduces a linearly stored 2-D ExtParam into 1-D Service. *NumRepeat* is a helper Service type which repeats a 1-D value into linearly stored 2-D value based on the shape from a *BackRef*.

**class** `andes.core.service.BackRef` (*\*\*kwargs*)

A special type of reference collector.

*BackRef* is used for collecting device indices of other models referencing the parent model of the *BackRef*. The *v* field will be a list of lists, each containing the *idx* of other models referencing each device of the parent model.

*BackRef* can be passed as indexer for params and vars, or shape for *NumReduce* and *NumRepeat*. See examples for illustration.

**See also:**

***andes.core.service.NumReduce*** A more complete example using *BackRef* to build the COI model

### Examples

A Bus device has an *IdxParam* of *area*, storing the *idx* of area to which the bus device belongs. In `Bus.__init__()`, one has

```
self.area = IdxParam(model='Area')
```

Suppose *Bus* has the following data

idx	area	Vn
1	1	110
2	2	220
3	1	345
4	1	500

The Area model wants to collect the indices of Bus devices which points to the corresponding Area device. In `Area.__init__`, one defines

```
self.Bus = BackRef()
```

where the member attribute name *Bus* needs to match exactly model name that *Area* wants to collect *idx* for. Similarly, one can define `self.ACTopology = BackRef()` to collect devices in the *ACTopology* group that references *Area*.

The collection of *idx* happens in `andes.system.System._collect_ref_param()`. It has to be noted that the specific *Area* entry must exist to collect model *idx*-dx referencing it. For example, if *Area* has the following data

```
idx
1
```

Then, only Bus 1, 3, and 4 will be collected into `self.Bus.v`, namely, `self.Bus.v == [ [1, 3, 4] ]`.

If *Area* has data

```
idx
1
2
```

Then, `self.Bus.v` will end up with `[ [1, 3, 4], [2] ]`.

```
class andes.core.service.NumReduce(u, ref: andes.core.service.BackRef, fun:
                                Callable, name=None, tex_name=None,
                                info=None, cache=True)
```

A helper Service type which reduces a linearly stored 2-D ExtParam into 1-D Service.

NumReduce works with ExtParam whose *v* field is a list of lists. A reduce function which takes an array-like and returns a scalar need to be supplied. NumReduce calls the reduce function on each of the lists and return all the scalars in an array.

### Parameters

**u** [ExtParam] Input ExtParam whose *v* contains linearly stored 2-dimensional values

**ref** [BackRef] The BackRef whose 2-dimensional shapes are used for indexing

**fun** [Callable] The callable for converting a 1-D array-like to a scalar

## Examples

Suppose one wants to calculate the mean value of the  $V_n$  in one Area. In the `Area` class, one defines

```
class AreaModel(...):
    def __init__(...):
        ...
        # backward reference from `Bus`
        self.Bus = BackRef()

        # collect the  $V_n$  in an 1-D array
        self.Vn = ExtParam(model='Bus',
                             src='Vn',
                             indexer=self.Bus)

        self.Vn_mean = NumReduce(u=self.Vn,
                                  fun=np.mean,
                                  ref=self.Bus)
```

Suppose we define two areas, 1 and 2, the `Bus` data looks like

idx	area	$V_n$
1	1	110
2	2	220
3	1	345
4	1	500

Then, `self.Bus.v` is a list of two lists `[ [1, 3, 4], [2] ]`. `self.Vn.v` will be retrieved and linearly stored as `[110, 345, 500, 220]`. Based on the shape from `self.Bus`, `numpy.mean()` will be called on `[110, 345, 500]` and `[220]` respectively. Thus, `self.Vn_mean.v` will become `[318.33, 220]`.

**class** `andes.core.service.NumRepeat` (*u*, *ref*, *\*\*kwargs*)

A helper Service type which repeats a v-provider's value based on the shape from a `BackRef`

## Examples

`NumRepeat` was originally designed for computing the inertia-weighted average rotor speed (center of inertia speed). COI speed is computed with

$$\omega_{COI} = \frac{\sum M_i * \omega_i}{\sum M_i}$$

The numerator can be calculated with a mix of `BackRef`, `ExtParam` and `ExtState`. The denominator needs to be calculated with `NumReduce` and `Service Repeat`. That is, use `NumReduce` to calculate the sum, and use `NumRepeat` to repeat the summed value for each device.

In the `COI` class, one would have

```

class COIModel(...):
    def __init__(...):
        ...
        self.SynGen = BackRef()
        self.SynGenIdx = RefFlatten(ref=self.SynGen)
        self.M = ExtParam(model='SynGen',
                           src='M',
                           indexer=self.SynGenIdx)

        self.wgen = ExtState(model='SynGen',
                              src='omega',
                              indexer=self.SynGenIdx)

        self.Mt = NumReduce(u=self.M,
                             fun=np.sum,
                             ref=self.SynGen)

        self.Mtr = NumRepeat(u=self.Mt,
                              ref=self.SynGen)

        self.pidx = IdxRepeat(u=self.idx, ref=self.SynGen)

```

Finally, one would define the center of inertia speed as

```

self.wcoi = Algeb(v_str='1', e_str='-wcoi')

self.wcoi_sub = ExtAlgeb(model='COI',
                          src='wcoi',
                          e_str='M * wgen / Mtr',
                          v_str='M / Mtr',
                          indexer=self.pidx,
                          )

```

It is very worth noting that the implementation uses a trick to separate the average weighted sum into  $n$  sub-equations, each calculating the  $(M_i * \omega_i) / (\sum M_i)$ . Since all the variables are preserved in the sub-equation, the derivatives can be calculated correctly.

**class** andes.core.service.IdxRepeat (u, ref, \*\*kwargs)

Helper class to repeat IdxParam.

This class has the same functionality as `andes.core.service.NumRepeat` but only operates on IdxParam, DataParam or NumParam.

**class** andes.core.service.RefFlatten (ref, \*\*kwargs)

A service type for flattening `andes.core.service.BackRef` into a 1-D list.

## Examples

This class is used when one wants to pass *BackRef* values as indexer.

`andes.models.coi.COI` collects referencing `andes.models.group.SynGen` with

```
self.SynGen = BackRef(info='SynGen idx lists', export=False)
```

After collecting BackRefs, *self.SynGen.v* will become a two-level list of indices, where the first level correspond to each COI and the second level correspond to generators of the COI.

Convert *self.SynGen* into 1-d as *self.SynGenIdx*, which can be passed as indexer for retrieving other parameters and variables

```
self.SynGenIdx = RefFlatten(ref=self.SynGen)

self.M = ExtParam(model='SynGen', src='M',
                  indexer=self.SynGenIdx, export=False,
                  )
```

### 3.6.4 Value Manipulation

**class** andes.core.service.**Replace** (*old\_val*, *flt*, *new\_val*, *name=None*,  
*tex\_name=None*, *info=None*, *cache=True*)

Replace parameters with new values if the function returns True

**class** andes.core.service.**FlagValue** (*u*, *value*, *flag=0*, *name=None*, *tex\_name=None*,  
*info=None*, *cache=True*)

Class for flagging values that equal to the given value.

By default, values that equal to *value* will be flagged as 0. Non-matching values will be flagged as 1.

#### Parameters

**u** Input parameter

**value** Value to flag. Can be None, string, or a number.

**flag** [0 by default, only 0 or 1 is accepted.] The flag for the matched ones

**Warning:** *FlagNotNone* can only be applied to *BaseParam* with *cache=True*. Applying to *Service* will fail unless *cache* is False (at a performance cost).

### 3.6.5 Idx and References

**class** andes.core.service.**DeviceFinder** (*u*, *link*, *idx\_name*, *name=None*,  
*tex\_name=None*, *info=None*)

Service for finding indices of optionally linked devices.

If not provided, *DeviceFinder* will add devices at the beginning of *System.setup*.

#### Examples

IEEEEST stabilizer takes an optional *busf* (IdxParam) for specifying the connected BusFreq, which is needed for mode 6. To avoid reimplementing *BusFreq* within IEEEEST, one can do

```
self.busfreq = DeviceFinder(self.busf, link=self.buss, idx_name='bus')
```

where *self.busf* is the optional input, *self.buss* is the bus indices that *busf* should measure, and *idx\_name* is the name of a *BusFreq* parameter through which the measured bus indices are specified. For each *None* values in *self.busf*, a *BusFreq* is created to measure the corresponding bus in *self.buss*.

That is, `BusFreq[idx_name].v = [link]`. *DeviceFinder* will find / create *BusFreq* devices so that the returned list of *BusFreq* indices are connected to *self.buss*, respectively.

**class** andes.core.service.**BackRef** (\*\*kwargs)

A special type of reference collector.

*BackRef* is used for collecting device indices of other models referencing the parent model of the *BackRef*. The *v* field will be a list of lists, each containing the *idx* of other models referencing each device of the parent model.

*BackRef* can be passed as indexer for params and vars, or shape for *NumReduce* and *NumRepeat*. See examples for illustration.

**See also:**

*andes.core.service.NumReduce* A more complete example using *BackRef* to build the COI model

## Examples

A *Bus* device has an *IdxParam* of *area*, storing the *idx* of area to which the bus device belongs. In *Bus.\_\_init\_\_()*, one has

```
self.area = IdxParam(model='Area')
```

Suppose *Bus* has the following data

idx	area	Vn
1	1	110
2	2	220
3	1	345
4	1	500

The *Area* model wants to collect the indices of *Bus* devices which points to the corresponding *Area* device. In *Area.\_\_init\_\_*, one defines

```
self.Bus = BackRef()
```

where the member attribute name *Bus* needs to match exactly model name that *Area* wants to collect *idx* for. Similarly, one can define `self.ACTopology = BackRef()` to collect devices in the *ACTopology* group that references *Area*.



The collection of *idx* happens in `andes.system.System._collect_ref_param()`. It has to be noted that the specific *Area* entry must exist to collect model *idx-dx* referencing it. For example, if *Area* has the following data

```
idx
1
```

Then, only Bus 1, 3, and 4 will be collected into *self.Bus.v*, namely, `self.Bus.v == [ [1, 3, 4] ]`.

If *Area* has data

```
idx
1
2
```

Then, *self.Bus.v* will end up with `[ [1, 3, 4], [2] ]`.

**class** `andes.core.service.RefFlatten` (*ref*, **\*\*kwargs**)  
 A service type for flattening `andes.core.service.BackRef` into a 1-D list.

## Examples

This class is used when one wants to pass *BackRef* values as indexer.

`andes.models.coi.COI` collects referencing `andes.models.group.SynGen` with

```
self.SynGen = BackRef(info='SynGen idx lists', export=False)
```

After collecting *BackRefs*, *self.SynGen.v* will become a two-level list of indices, where the first level correspond to each COI and the second level correspond to generators of the COI.

Convert *self.SynGen* into 1-d as *self.SynGenIdx*, which can be passed as indexer for retrieving other parameters and variables

```
self.SynGenIdx = RefFlatten(ref=self.SynGen)

self.M = ExtParam(model='SynGen', src='M',
                  indexer=self.SynGenIdx, export=False,
                  )
```

## 3.6.6 Events

**class** `andes.core.service.EventFlag` (*u*, *vtype*: `Optional[type]` = `None`, *name*:  
`Optional[str]` = `None`, *tex\_name*=`None`,  
*info*=`None`)

Service to flag events.

*EventFlag.v* stores the values of the input variable from the previous iteration/step.

```
class andes.core.service.ExtendedEvent (u, t_ext: Union[int, float, andes.core.param.BaseParam, andes.core.service.BaseService] = 0.0, trig: str = 'rise', enable=True, v_disabled=0, extend_only=False, vtype: Optional[type] = None, name: Optional[str] = None, tex_name=None, info=None)
```

Service to flag events that extends for period of time after event disappears.

*EventFlag.v* stores the flags whether the extended time has completed. Outputs will become 1 once then event starts until the extended time ends.

#### Parameters

**trig** [str, rise, fall] Triggering edge for the inception of an event. *rise* by default.

**enable** [bool or v-provider] If disabled, the output will be *v\_disabled*

**extend\_only** [bool] Only output during the extended period, not the event period.

**Warning:** The performance of this class needs to be optimized.

### 3.6.7 Data Select

```
class andes.core.service.DataSelect (optional, fallback, name: Optional[str] = None, tex_name: Optional[str] = None, info: Optional[str] = None)
```

Class for selecting values for optional DataParam or NumParam.

This service is a v-provider that uses optional DataParam if available with a fallback.

DataParam will be tested for *None*, and NumParam will be tested with *np.isnan()*.

#### Notes

An use case of DataSelect is remote bus. One can do

```
self.buss = DataSelect(option=self.busr, fallback=self.bus)
```

Then, pass *self.buss* instead of *self.bus* as indexer to retrieve voltages.

Another use case is to allow an optional turbine rating. One can do

```
self.Tn = NumParam(default=None)
self.Sg = ExtParam(...)
self.Sn = DataSelect(Tn, Sg)
```

```
class andes.core.service.NumSelect (optional, fallback, name: Optional[str] = None, tex_name: Optional[str] = None, info: Optional[str] = None)
```

Class for selecting values for optional NumParam.

## Notes

One use case is to allow an optional turbine rating. One can do

```
self.Tn = NumParam(default=None)
self.Sg = ExtParam(...)
self.Sn = DataSelect(Tn, Sg)
```

### 3.6.8 Miscellaneous

```
class andes.core.service.InitChecker (u, lower=None, upper=None, equal=None,
                                       not_equal=None, enable=True, error_out=False, **kwargs)
```

Class for checking init values against known typical values.

Instances will be stored in *Model.services\_post* and *Model.services\_ichk*, which will be checked in *Model.post\_init\_check()* after initialization.

#### Parameters

**u** v-provider to be checked

**lower** [float, BaseParam, BaseVar, BaseService] lower bound

**upper** [float, BaseParam, BaseVar, BaseService] upper bound

**equal** [float, BaseParam, BaseVar, BaseService] values that the value from *v\_str* should equal

**not\_equal** [float, BaseParam, BaseVar, BaseService] values that should not equal

**enable** [bool] True to enable checking

## Examples

Let's say generator excitation voltages are known to be in the range of 1.6 - 3.0 per unit. One can add the following instance to *GENBase*

```
self._vfc = InitChecker(u=self.vf,
                        info='vf range',
                        lower=1.8,
                        upper=3.0,
                        )
```

*lower* and *upper* can also take v-providers instead of float values.

One can also pass float values from Config to make it adjustable as in our implementation of *GENBase.\_vfc*.

## 3.7 Discrete

### 3.7.1 Background

The discrete component library contains a special type of block for modeling the discontinuity in power system devices. Such continuities can be device-level physical constraints or algorithmic limits imposed on controllers.

The base class for discrete components is `andes.core.discrete.Discrete`.

```
class andes.core.discrete.Discrete(name=None, tex_name=None, info=None,
                                     no_warn=False)
```

Base discrete class.

Discrete classes export flag arrays (usually boolean) .

The uniqueness of discrete components is the way it works. Discrete components take inputs, criteria, and exports a set of flags with the component-defined meanings. These exported flags can be used in algebraic or differential equations to build piece-wise equations.

For example, *Limiter* takes a v-provider as input, two v-providers as the upper and the lower bound. It exports three flags: *zi* (within bound), *zl* (below lower bound), and *zu* (above upper bound). See the code example in `models/pv.py` for an example voltage-based PQ-to-Z conversion.

It is important to note when the flags are updated. Discrete subclasses can use three methods to check and update the value and equations. Among these methods, *check\_var* is called *before* equation evaluation, but *check\_eq* and *set\_eq* are called *after* equation update. In the current implementation, *check\_var* updates flags for variable-based discrete components (such as *Limiter*). *check\_eq* updates flags for equation-involved discrete components (such as *AntiWindup*). *set\_var* is currently only used by *AntiWindup* to store the pegged states.

ANDES includes the following types of discrete components.

### 3.7.2 Limiters

```
class andes.core.discrete.Limiter(u, lower, upper, enable=True, name=None,
                                   tex_name=None, info=None, no_upper=False,
                                   no_lower=False, equal=True, zu=0.0, zl=0.0,
                                   zi=1.0)
```

Base limiter class.

This class compares values and sets limit values. Exported flags are *zi*, *zl* and *zu*.

#### Parameters

- u** [BaseVar] Input Variable instance
- lower** [BaseParam] Parameter instance for the lower limit
- upper** [BaseParam] Parameter instance for the upper limit
- no\_lower** [bool] True to only use the upper limit
- no\_upper** [bool] True to only use the lower limit

**equal:** **bool** True to include equal signs in comparison ( $\geq$  or  $\leq$ ).

**zu** [0 or 1] Default value for *zu* if not enabled

**zl** [0 or 1] Default value for *zl* if not enabled

**zi** [0 or 1] Default value for *zi* if not enabled

## Notes

If not enabled, the default flags are  $zu = zl = 0, zi = 1$ .

### Attributes

**zl** [array-like] Flags of elements violating the lower limit; A array of zeros and/or ones.

**zi** [array-like] Flags for within the limits

**zu** [array-like] Flags for violating the upper limit

```
class andes.core.discrete.SortedLimiter (u, lower, upper, enable=True, n_select:  
                                         Optional[int] = None, name=None,  
                                         tex_name=None)
```

A comparer with the top value selection.

```
class andes.core.discrete.HardLimiter (u, lower, upper, enable=True,  
                                         name=None, tex_name=None, info=None,  
                                         no_upper=False, no_lower=False,  
                                         equal=True, zu=0.0, zl=0.0, zi=1.0)
```

Hard limiter for algebraic or differential variable. This class is an alias of *Limiter*.

```
class andes.core.discrete.AntiWindup (u, lower, upper, enable=True,  
                                         no_lower=False, no_upper=False,  
                                         name=None, tex_name=None, info=None,  
                                         state=None)
```

Anti-windup limiter.

Anti-windup limiter prevents the wind-up effect of a differential variable. The derivative of the differential variable is reset if it continues to increase in the same direction after exceeding the limits. During the derivative return, the limiter will be inactive

```
if x > xmax and x dot > 0: x = xmax and x dot = 0  
if x < xmin and x dot < 0: x = xmin and x dot = 0
```

This class takes one more optional parameter for specifying the equation.

### Parameters

**state** [State, ExtState] A State (or ExtState) whose equation value will be checked and, when condition satisfies, will be reset by the anti-windup-limiter.

### 3.7.3 Comparers

```
class andes.core.discrete.LessThan(u, bound, equal=False, enable=True,  
                                   name=None, tex_name=None, info=None,  
                                   cache=False, z0=0, z1=1)
```

Less than (<) comparison function.

Exports two flags: *z1* and *z0*. For elements satisfying the less-than condition, the corresponding *z1* = 1. *z0* is the element-wise negation of *z1*.

#### Notes

The default *z0* and *z1*, if not enabled, can be set through the constructor.

```
class andes.core.discrete.Selector(*args, fun, tex_name=None, info=None)  
Selection between two variables using the provided reduce function.
```

The reduce function should take the given number of arguments. An example function is *np.maximum.reduce* which can be used to select the maximum.

Names are in *s0*, *s1*.

**Warning:** A potential bug when more than two inputs are provided, and values in different inputs are equal. Only two inputs are allowed.

See also:

`numpy.ufunc.reduce` NumPy reduce function

`andes.core.block.HVGate`

`andes.core.block.LVGate`

#### Notes

A common pitfall is the 0-based indexing in the Selector flags. Note that exported flags start from 0. Namely, *s0* corresponds to the first variable provided for the Selector constructor.

#### Examples

Example 1: select the largest value between *v0* and *v1* and put it into *vmax*.

After the definitions of *v0* and *v1*, define the algebraic variable *vmax* for the largest value, and a selector *vs*

```

self.vmax = Algeb(v_str='maximum(v0, v1)',
                  tex_name='v_{max}',
                  e_str='vs_s0 * v0 + vs_s1 * v1 - vmax')

self.vs = Selector(self.v0, self.v1, fun=np.maximum.reduce)

```

The initial value of  $v_{max}$  is calculated by `maximum(v0, v1)`, which is the element-wise maximum in SymPy and will be generated into `np.maximum(v0, v1)`. The equation of  $v_{max}$  is to select the values based on  $vs\_s0$  and  $vs\_s1$ .

**class** `andes.core.discrete.Switcher` (*u, options: Union[list, Tuple], name: str = None, tex\_name: str = None, cache=True*)

Switcher based on an input parameter.

The switch class takes one v-provider, compares the input with each value in the option list, and exports one flag array for each option. The flags are 0-indexed.

Exported flags are named with  $\_s0$ ,  $\_s1$ , ..., with a total number of `len(options)`. See the examples section.

## Notes

Switches needs to be distinguished from Selector.

Switcher is for generating flags indicating option selection based on an input parameter. Selector is for generating flags at run time based on variable values and a selection function.

## Examples

The IEEEEST model takes an input for selecting the signal. Options are 1 through 6. One can construct

```

self.IC = NumParam(info='input code 1-6') # input code
self.SW = Switcher(u=self.IC, options=[1, 2, 3, 4, 5, 6])

```

If the IC values from the data file ends up being

```

self.IC.v = np.array([1, 2, 2, 4, 6])

```

Then, the exported flag arrays will be

```

{'IC_s0': np.array([1, 0, 0, 0, 0]),
 'IC_s1': np.array([0, 1, 1, 0, 0]),
 'IC_s2': np.array([0, 0, 0, 0, 0]),
 'IC_s3': np.array([0, 0, 0, 1, 0]),
 'IC_s4': np.array([0, 0, 0, 0, 0]),
 'IC_s5': np.array([0, 0, 0, 0, 1])
}

```

### 3.7.4 Deadband

```
class andes.core.discrete.DeadBand(u, center, lower, upper, enable=True,  
                                   equal=True, zu=0.0, zl=0.0, zi=0.0,  
                                   name=None, tex_name=None, info=None)
```

The basic deadband type.

#### Parameters

- u** [NumParam] The pre-deadband input variable
- center** [NumParam] Neutral value of the output
- lower** [NumParam] Lower bound
- upper** [NumParam] Upper bound
- enable** [bool] Enabled if True; Disabled and works as a pass-through if False.

#### Notes

Input changes within a deadband will incur no output changes. This component computes and exports three flags.

#### Three flags computed from the current input:

- **zl**: True if the input is below the lower threshold
- **zi**: True if the input is within the deadband
- **zu**: True if is above the lower threshold

Initial condition:

All three flags are initialized to zero. All flags are updated during *check\_var* when enabled. If the deadband component is not enabled, all of them will remain zero.

#### Examples

Exported deadband flags need to be used in the algebraic equation corresponding to the post-deadband variable. Assume the pre-deadband input variable is *var\_in* and the post-deadband variable is *var\_out*. First, define a deadband instance *db* in the model using

```
self.db = DeadBand(u=self.var_in, center=self.dbc,  
                  lower=self.dbl, upper=self.dbu)
```

To implement a no-memory deadband whose output returns to center when the input is within the band, the equation for *var* can be written as

```
var_out.e_str = 'var_in * (1 - db_zi) + \  
                (dbc * db_zi) - var_out'
```



## 3.8 Blocks

### 3.8.1 Background

The block library contains commonly used blocks (such as transfer functions and nonlinear functions). Variables and equations are pre-defined for blocks to be used as "lego pieces" for scripting DAE models. The base class for blocks is `andes.core.block.Block`.

The supported blocks include `Lag`, `LeadLag`, `Washout`, `LeadLagLimit`, `PIController`. In addition, the base class for piece-wise nonlinear functions, `PieceWise` is provided. `PieceWise` is used for implementing the quadratic saturation function `MagneticQuadSat` and exponential saturation function `MagneticExpSat`.

All variables in a block must be defined as attributes in the constructor, just like variable definition in models. The difference is that the variables are "exported" from a block to the capturing model. All exported variables need to be placed in a dictionary, `self.vars` at the end of the block constructor.

Blocks can be nested as advanced usage. See the following API documentation for more details.

```
class andes.core.block.Block (name: Optional[str] = None, tex_name: Optional[str] =  
None, info: Optional[str] = None, namespace: str = 'lo-  
cal')
```

Base class for control blocks.

Blocks are meant to be instantiated as Model attributes to provide pre-defined equation sets. Subclasses must overload the `__init__` method to take custom inputs. Subclasses of `Block` must overload the `define` method to provide initialization and equation strings. Exported variables, services and blocks must be constructed into a dictionary `self.vars` at the end of the constructor.

Blocks can be nested. A block can have blocks but itself as attributes and therefore reuse equations. When a block has sub-blocks, the outer block must be constructed with a "name".

Nested block works in the following way: the parent block modifies the sub-block's `name` attribute by prepending the parent block's name at the construction phase. The parent block then exports the sub-block as a whole. When the parent Model class picks up the block, it will recursively import the variables in the block and the sub-blocks correctly. See the example section for details.

#### Parameters

**name** [str, optional] Block name

**tex\_name** [str, optional] Block LaTeX name

**info** [str, optional] Block description.

**namespace** [str, local or parent] Namespace of the exported elements. If 'local', the block name will be prepended by the parent. If 'parent', the original element name will be used when exporting.

**Warning:** It is a good practice to avoid more than one level of nesting, to avoid multi-underscore variable names.

## Examples

Example for two-level nested blocks. Suppose we have the following hierarchy

```
SomeModel  instance M
|
LeadLag A  exports (x, y)
|
Lag B      exports (x, y)
```

SomeModel instance M contains an instance of LeadLag block named A, which contains an instance of a Lag block named B. Both A and B exports two variables x and y.

In the code of Model, the following code is used to instantiate LeadLag

```
class SomeModel:
    def __init__(...):
        ...
        self.A = LeadLag(name='A',
                          u=self.foo1,
                          T1=self.foo2,
                          T2=self.foo3)
```

To use Lag in the LeadLag code, the following lines are found in the constructor of LeadLag

```
class LeadLag:
    def __init__(name, ...):
        ...
        self.B = Lag(u=self.y, K=self.K, T=self.T)
        self.vars = {..., 'A': self.A}
```

The `__setattr__` magic of LeadLag takes over the construction and assigns `A_B` to `B.name`, given A's name provided at run time. `self.A` is exported with the internal name A at the end.

Again, the LeadLag instance name (A in this example) MUST be provided in *SomeModel*'s constructor for the name prepending to work correctly. If there is more than one level of nesting, other than the leaf-level block, all parent blocks' names must be provided at instantiation.

When A is picked up by *SomeModel.\_\_setattr\_\_*, B is captured from A's exports. Recursively, B's variables are exported. Recall that `B.name` is now `A_B`, following the naming rule (parent block's name + variable name), B's internal variables become `A_B_x` and `A_B_y`.

In this way, B's `define()` needs no modification since the naming rule is the same. For example, B's internal y is always `{self.name}_y`, although B has gotten a new name `A_B`.

### 3.8.2 Transfer Functions

The following transfer function blocks have been implemented. They can be imported to build new models.

## Algebraic

**class** andes.core.block.**Gain**(*u, K, name=None, tex\_name=None, info=None*)  
Gain block.



Exports an algebraic output  $y$ .

**define**()  
Implemented equation and the initial condition are

$$y = Ku$$

$$y^{(0)} = Ku^{(0)}$$

## First Order

**class** andes.core.block.**Integrator**(*u, T, K, y0, name=None, tex\_name=None, info=None*)  
Integrator block.



Exports a differential variable  $y$ . The initial output is specified by  $y0$  and default to zero.

**define**()  
Implemented equation and the initial condition are

$$\dot{y} = Ku$$

$$y^{(0)} = 0$$

**class** andes.core.block.**IntegratorAntiWindup**(*u, T, K, y0, lower, upper, name=None, tex\_name=None, info=None*)  
Integrator block with anti-windup limiter.



Exports a differential variable  $y$  and an AntiWindup *lim*. The initial output must be specified through  $y0$ .

**define()**

Implemented equation and the initial condition are

$$\dot{y} = Ku$$

$$y^{(0)} = 0$$

**class** andes.core.block.**Lag**(*u, T, K, name=None, tex\_name=None, info=None*)

Lag (low pass filter) transfer function.



Exports one state variable  $y$  as the output.

### Parameters

**K** Gain

**T** Time constant

**u** Input variable

**define()**

### Notes

Equations and initial values are

$$T\dot{y} = (Ku - y)$$

$$y^{(0)} = Ku$$

**class** andes.core.block.**LagAntiWindup**(*u, T, K, lower, upper, name=None, tex\_name=None, info=None*)

Lag (low pass filter) transfer function block with an anti-windup limiter.



Exports one state variable  $y$  as the output and one AntiWindup instance *lim*.

### Parameters

**K** Gain

**T** Time constant

**u** Input variable

**define()**

### Notes

Equations and initial values are

$$T\dot{y} = (Ku - y)$$

$$y^{(0)} = Ku$$

**class** andes.core.block.**Washout** (*u, T, K, name=None, tex\_name=None, info=None*)

Washout filter (high pass) block.



Exports state  $x$  (symbol  $x'$ ) and output algebraic variable  $y$ .

**define()**

### Notes

Equations and initial values:

$$T\dot{x}' = (u - x')$$

$$Ty = K(u - x')$$

$$x'^{(0)} = u$$

$$y^{(0)} = 0$$

**class** andes.core.block.**WashoutOrLag** (*u, T, K, name=None, zero\_out=True, tex\_name=None, info=None*)

Washout with the capability to convert to Lag when  $K = 0$ .

Can be enabled with *zero\_out*. Need to provide *name* to construct.

Exports state  $x$  (symbol  $x'$ ), output algebraic variable  $y$ , and a LessThan block *LT*.

### Parameters

**zero\_out** [bool, optional] If True,  $sT$  will become 1, and the washout will become a low-pass filter. If False, functions as a regular Washout.

**define()**

## Notes

Equations and initial values:

$$\begin{aligned} T\dot{x}' &= (u - x') \\ Ty &= z_0 K(u - x') + z_1 Tx \\ x^{(0)} &= u \\ y^{(0)} &= 0 \end{aligned}$$

where  $z\_0$  is a flag array for the greater-than-zero elements, and  $z\_1$  is that for the less-than or equal-to zero elements.

```
class andes.core.block.LeadLag(u, T1, T2, K=1, zero_out=True, name=None,
                               tex_name=None, info=None)
```

Lead-Lag transfer function block in series implementation

$$u \rightarrow \left[ \begin{array}{c} \frac{1 + sT1}{1 + sT2} \end{array} \right] K \rightarrow y$$

Exports two variables: internal state  $x$  and output algebraic variable  $y$ .

## Parameters

**T1** [BaseParam] Time constant 1

**T2** [BaseParam] Time constant 2

**zero\_out** [bool] True to allow zeroing out lead-lag as a pass through (when  $T1=T2=0$ )

## Notes

To allow zeroing out lead-lag as a pure gain, set `zero_out` to *True*.

```
define ()
```

## Notes

Implemented equations and initial values

$$\begin{aligned} T_2\dot{x}' &= (u - x') \\ T_2y &= KT_1(u - x') + KT_2x' + E_2, \text{ where} \\ E_2 &= \begin{cases} (y - Kx') & \text{if } T_1 = T_2 = 0 \& \text{zero\_out} = \text{True} \\ 0 & \text{otherwise} \end{cases} \\ x^{(0)} &= u \\ y^{(0)} &= Ku \end{aligned}$$

**class** andes.core.block.**LeadLagLimit**(*u*, *T1*, *T2*, *lower*, *upper*, *name=None*,  
*tex\_name=None*, *info=None*)  
 Lead-Lag transfer function block with hard limiter (series implementation)



Exports four variables: state *x*, output before hard limiter *ynl*, output *y*, and AntiWindup *lim*.

**define**()

### Notes

Implemented control block equations (without limiter) and initial values

$$\begin{aligned} T_2 \dot{x}' &= (u - x') \\ T_2 y &= T_1(u - x') + T_2 x' \\ x'^{(0)} &= y^{(0)} = u \end{aligned}$$

### Second Order

**class** andes.core.block.**Lag2ndOrd**(*u*, *K*, *T1*, *T2*, *name=None*, *tex\_name=None*,  
*info=None*)  
 Second order lag transfer function (low-pass filter)



Exports one two state variables (*x*, *y*), where *y* is the output.

### Parameters

**u** Input

**K** Gain

**T1** First order time constant

**T2** Second order time constant

**define**()

## Notes

Implemented equations and initial values are

$$T_2 \dot{x} = Ku - y - T_1 x$$

$$\dot{y} = x$$

$$x^{(0)} = 0$$

$$y^{(0)} = Ku$$

```
class andes.core.block.LeadLag2ndOrd(u, T1, T2, T3, T4, zero_out=False,
                                     name=None, tex_name=None, info=None)
```

Second-order lead-lag transfer function block

$$u \rightarrow \left[ \frac{1 + sT_3 + s^2 T_4}{1 + sT_1 + s^2 T_2} \right] \rightarrow y$$

Exports two internal states ( $x_1$  and  $x_2$ ) and output algebraic variable  $y$ .

# TODO: instead of implementing `zero_out` using `LessThan` and an additional term, consider correcting all parameters to 1 if all are 0.

```
define ()
```

## Notes

Implemented equations and initial values are

$$T_2 \dot{x}_1 = u - x_2 - T_1 x_1$$

$$\dot{x}_2 = x_1$$

$$T_2 y = T_2 x_2 + T_2 T_3 x_1 + T_4 (u - x_2 - T_1 x_1) + E_2, \text{ where}$$

$$E_2 = \begin{cases} (y - x_2) & \text{if } T_1 = T_2 = T_3 = T_4 = 0 \& \text{zero\_out} = \text{True} \\ 0 & \text{otherwise} \end{cases}$$

$$x_1^{(0)} = 0$$

$$x_2^{(0)} = y^{(0)} = u$$

### 3.8.3 Saturation

```
class andes.models.exciter.ExcExpSat(E1, SE1, E2, SE2, name=None,
                                     tex_name=None, info=None)
```

Exponential exciter saturation block to calculate A and B from E1, SE1, E2 and SE2. Input parameters will be corrected and the user will be warned. To disable saturation, set either E1 or E2 to 0.

#### Parameters

**E1** [BaseParam] First point of excitation field voltage



**SE1: BaseParam** Coefficient corresponding to E1

**E2** [BaseParam] Second point of excitation field voltage

**SE2: BaseParam** Coefficient corresponding to E2

**define** ()

### Notes

The implementation solves for coefficients  $A$  and  $B$  which satisfy

$$E_1 S_{E1} = A e^{E_1 \times B} \quad E_2 S_{E2} = A e^{E_2 \times B}$$

The solutions are given by

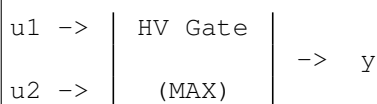
$$E_1 S_{E1} e^{\frac{E_1 \log \left( \frac{E_2 S_{E2}}{E_1 S_{E1}} \right)}{E_1 - E_2}} - \frac{\log \left( \frac{E_2 S_{E2}}{E_1 S_{E1}} \right)}{E_1 - E_2}$$

## 3.8.4 Others

### Value Selector

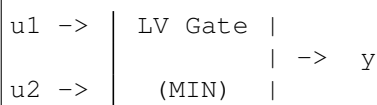
**class** andes.core.block.**HVGate** ( $u1, u2, name=None, tex\_name=None, info=None$ )

High Value Gate. Outputs the maximum of two inputs.



**class** andes.core.block.**LVGate** ( $u1, u2, name=None, tex\_name=None, info=None$ )

Low Value Gate. Outputs the minimum of the two inputs.



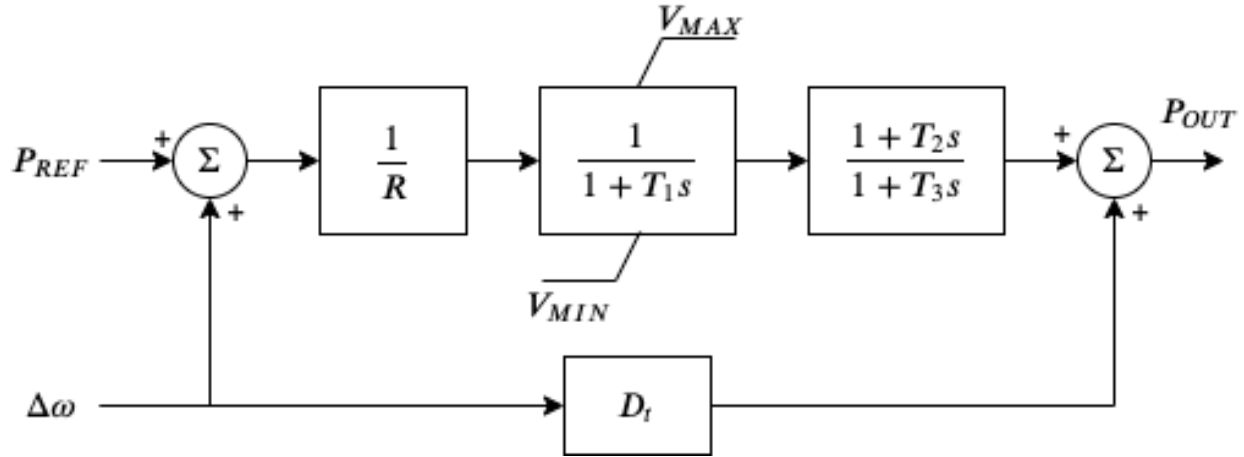
## 3.9 Examples

We show two examples to demonstrate modeling from equations and modeling from control block diagrams.

- The TGOV1 example shows code snippet for equation-based modeling and, as well as code for block-based modeling.
- The IEEEEST example walks through the source code and explains the complete setup, including optional parameters, input selection, and manual per-unit conversion.

### 3.9.1 TGOV1

The *TGOV1* turbine governor model is shown as a practical example using the library.



This model is composed of a lead-lag transfer function and a first-order lag transfer function with an anti-windup limiter, which are sufficiently complex for demonstration. The corresponding differential equations and algebraic equations are given below.

$$\begin{bmatrix} \dot{x}_{LG} \\ \dot{x}_{LL} \end{bmatrix} = \begin{bmatrix} z_{i,lim}^{LG} (P_d - x_{LG}) / T_1 \\ (x_{LG} - x_{LL}) / T_3 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} (1 - \omega) - \omega_d \\ R \times \tau_{m0} - P_{ref} \\ (P_{ref} + \omega_d) / R - P_d \\ D_t \omega_d + y_{LL} - P_{OUT} \\ \frac{T_2}{T_3} (x_{LG} - x_{LL}) + x_{LL} - y_{LL} \\ u (P_{OUT} - \tau_{m0}) \end{bmatrix}$$

where *LG* and *LL* denote the lag block and the lead-lag block,  $\dot{x}_{LG}$  and  $\dot{x}_{LL}$  are the internal states,  $y_{LL}$  is the lead-lag output,  $\omega$  the generator speed,  $\omega_d$  the generator under-speed,  $P_d$  the droop output,  $\tau_{m0}$  the steady-state torque input, and  $P_{OUT}$  the turbine output that will be summed at the generator.

The code to describe the above model using equations is given below. The complete code can be found in class `TGOV1ModelAlt` in `andes/models/governor.py`.

```
def __init__(self, system, config):
    # 1. Declare parameters from case file inputs.
    self.R = NumParam(info='Turbine governor droop',
                      non_zero=True, ipower=True)
    # Other parameters are omitted.

    # 2. Declare external variables from generators.
    self.omega = ExtState(src='omega',
                          model='SynGen',
                          indexer=self.syn,
                          info='Generator speed')
    self.tm = ExtAlgeb(src='tm',
```

(continues on next page)

(continued from previous page)

```

        model='SynGen',
        indexer=self.syn,
        e_str='u*(pout-tm0)',
        info='Generator torque input')

# 3. Declare initial values from generators.
self.tm0 = ExtService(src='tm',
                      model='SynGen',
                      indexer=self.syn,
                      info='Initial torque input')

# 4. Declare variables and equations.
self.pref = Algeb(info='Reference power input',
                  v_str='tm0*R',
                  e_str='tm0*R-pref')
self.wd = Algeb(info='Generator under speed',
                e_str='(1-omega)-wd')
self.pd = Algeb(info='Droop output',
                v_str='tm0',
                e_str='(wd+pref)/R-pd')
self.LG_x = State(info='State in the lag TF',
                  v_str='pd',
                  e_str='LG_lim_zi*(pd-LG_x)/T1')
self.LG_lim = AntiWindup(u=self.LG_x,
                        lower=self.VMIN,
                        upper=self.VMAX)
self.LL_x = State(info='State in the lead-lag TF',
                  v_str='LG_x',
                  e_str='(LG_x-LL_x)/T3')
self.LL_y = Algeb(info='Lead-lag Output',
                  v_str='LG_x',
                  e_str='T2/T3*(LG_x-LL_x)+LL_x-LL_y')
self.pout = Algeb(info='Turbine output power',
                  v_str='tm0',
                  e_str='(LL_y+Dt*wd)-pout')

```

Another implementation of *TGOVI* makes extensive use of the modeling blocks. The resulting code is more readable as follows.

```

def __init__(self, system, config):
    TGBase.__init__(self, system, config)

    self.gain = ConstService(v_str='u/R')

    self.pref = Algeb(info='Reference power input',
                      tex_name='P_{ref}',
                      v_str='tm0 * R',
                      e_str='tm0 * R - pref',
                      )

    self.wd = Algeb(info='Generator under speed',

```

(continues on next page)

(continued from previous page)

```

        unit='p.u.',
        tex_name=r'\omega_{dev}',
        v_str='0',
        e_str='(wref - omega) - wd',
    )
    self.pd = Algeb(info='Pref plus under speed times gain',
        unit='p.u.',
        tex_name="P_d",
        v_str='u * tm0',
        e_str='u*(wd + pref + paux) * gain - pd')

    self.LAG = LagAntiWindup(u=self.pd,
        K=1,
        T=self.T1,
        lower=self.VMIN,
        upper=self.VMAX,
    )

    self.LL = LeadLag(u=self.LAG_y,
        T1=self.T2,
        T2=self.T3,
    )

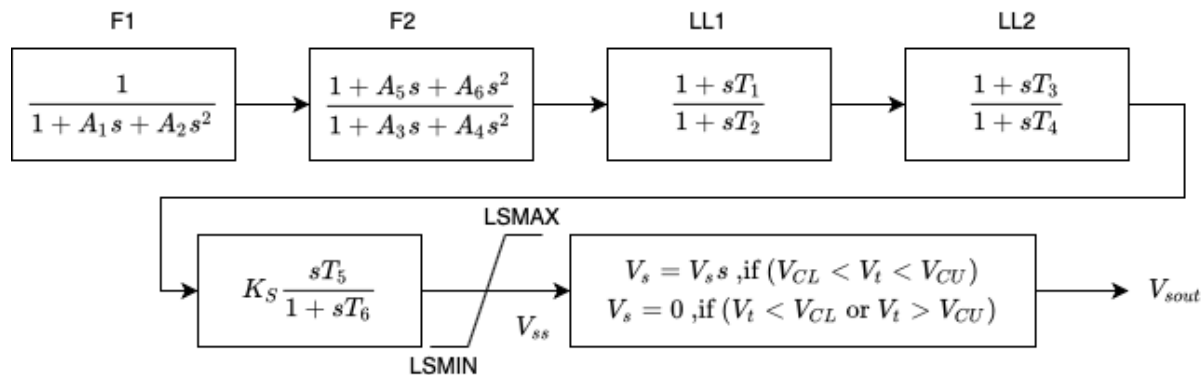
    self.pout.e_str = '(LL_y + Dt * wd) - pout'

```

The complete code can be found in class `TGOV1Model` in `andes/models/governor.py`.

### 3.9.2 IEEEEST

In this example, we will explain step-by-step how *IEEEEST* is programmed. The block diagram of IEEEEST is given as follows. We recommend you to open up the source code in `andes/models/pss.py` and then continue reading.



First of all, modeling components are imported at the beginning.

Next, `PSSBaseData` is defined to hold parameters shared by all PSSs. `PSSBaseData` inherits from `ModelData` and calls the base constructor. There is only one field `avr` defined for the linked exciter idx.

Then, IEEEESTData defines the input parameters for IEEEEST. Use `IdxParam` for fields that store idx-es of devices that IEEEEST devices link to. Use `NumParam` for numerical parameters.

## PSSBase

PSSBase is defined for the common (external) parameters, services and variables shared by all PSSs. The class and constructor signatures are

```
class PSSBase(Model):
    def __init__(self, system, config):
        super().__init__(system, config)
```

PSSBase inherits from `Model` and calls the base constructor. Note that the call to `Model`'s constructor takes two positional arguments, `system` and `config` of types `System` and `ModelConfig`. Next, the group is specified, and the model flags are set.

```
self.group = 'PSS'
self.flags.update({'tds': True})
```

Next, `Replace` is used to replace input parameters that satisfy a lambda function with new values.

```
self.VCUr = Replace(self.VCU, lambda x: np.equal(x, 0.0), 999)
self.VCLr = Replace(self.VCL, lambda x: np.equal(x, 0.0), -999)
```

The value replacement happens when `VCUr` and `VCLr` is first accessed. `Replace` is executed in the model initialization phase (at the end of services update).

Next, the indices of connected generators, buses, and bus frequency measurements are retrieved. Synchronous generator idx is retrieved with

```
self.syn = ExtParam(model='Exciter', src='syn', indexer=self.avr,
    ↳export=False,
                        info='Retrieved generator idx', dtype=str)
```

Using the retrieved `self.syn`, it retrieves the buses to which the generators are connected.

```
self.bus = ExtParam(model='SynGen', src='bus', indexer=self.syn, export=False,
                    info='Retrieved bus idx', dtype=str, default=None,
                    )
```

PSS models support an optional remote bus specified through parameter `busr`. When `busr` is `None`, the generator-connected bus should be used. The following code uses `DataSelect` to select `busr` if available but falls back to `bus` otherwise.

```
self.buss = DataSelect(self.busr, self.bus, info='selected bus (bus or busr)')
```

Each PSS links to a bus frequency measurement device. If the input data does not specify one or the specified one does not exist, `DeviceFinder` can find the correct measurement device for the bus where frequency measurements should be taken.

```
self.busfreq = DeviceFinder(self.busf, link=self.buss, idx_name='bus')
```

where `busf` is the optional frequency measurement device `idx`, `buss` is the bus `idx` for which measurement device needs to be found or created.

Next, external parameters, variables and services are retrieved. Note that the PSS output `vsout` is pre-allocated but the equation string is left to specific models.

## IEEESTModel

IEEESTModel inherits from PSSBase and adds specific model components. After calling PSSBase's constructor, IEEESTModel adds config entries to allow specifying the model for frequency measurement, because there may be multiple frequency measurement models in the future.

```
self.config.add(OrderedDict([('freq_model', 'BusFreq')]))
self.config.add_extra('_help', {'freq_model': 'default freq. measurement model
→'})
self.config.add_extra('_alt', {'freq_model': ('BusFreq',)})
```

We set the chosen measurement model to `busf` so that DeviceFinder knows which model to use if it needs to create new devices.

```
self.busf.model = self.config.freq_model
```

Next, because bus voltage is an algebraic variable, we use `Derivative` to calculate the finite difference to approximate its derivative.

```
self.dv = Derivative(self.v, tex_name='dV/dt', info='Finite difference of bus_
→voltage')
```

Then, we retrieve the coefficient to convert power from machine base to system base using `ConstService`, given by  $S_b / S_n$ . This is needed for input mode 3, electric power in machine base.

```
self.SnSb = ExtService(model='SynGen', src='M', indexer=self.syn, attr='pu_
→coeff',
                        info='Machine base to sys base factor for power',
                        tex_name='(Sb/Sn)')
```

Note that the `ExtService` access the `pu_coeff` field of the `M` variables of synchronous generators. Since `M` is a machine-base power quantity, `M.pu_coeff` stores the multiplication coefficient to convert each of them from machine bases to the system base, which is  $S_b / S_n$ .

The input mode is parsed into boolean flags using `Switcher`:

```
self.SW = Switcher(u=self.MODE,
                   options=[0, 1, 2, 3, 4, 5, 6],
                   )
```

where the input `u` is the `MODE` parameter, and `options` is a list of accepted values. `Switcher` boolean arrays `s0, s1, ..., sN`, where  $N = \text{len}(\text{options}) - 1$ . We added 0 to `options` for padding so that `SW_s1` corresponds to `MODE 1`. It improves the readability of the code as we will see next.

The input signal `sig` is an algebraic variable given by

```
self.sig = Algeb(tex_name='S_{ig}',
                 info='Input signal',
                 )

self.sig.v_str = 'SW_s1*(omega-1) + SW_s2*0 + SW_s3*(tm0/SnSb) + ' \
                 'SW_s4*(tm-tm0) + SW_s5*v + SW_s6*0'

self.sig.e_str = 'SW_s1*(omega-1) + SW_s2*(f-1) + SW_s3*(te/SnSb) + ' \
                 'SW_s4*(tm-tm0) + SW_s5*v + SW_s6*dv_v - sig'
```

The `v_str` and `e_str` are separated from the constructor to improve readability. They construct piecewise functions to select the correct initial values and equations based on mode. For any variables in `v_str`, they must be defined before `sig` so that they will be initialized ahead of `sig`. Clearly, `omega`, `tm`, and `v` are defined in `PSSBase` and thus come before `sig`.

The following comes the most effective part: modeling using transfer function blocks. We utilized several blocks to describe the model from the diagram. Note that the output of a block is always the block name followed by `_y`. For example, the input of `F2` is the output of `F1`, given by `F1_y`.

```
self.F1 = Lag2ndOrd(u=self.sig, K=1, T1=self.A1, T2=self.A2)

self.F2 = LeadLag2ndOrd(u=self.F1_y, T1=self.A3, T2=self.A4,
                       T3=self.A5, T4=self.A6, zero_out=True)

self.LL1 = LeadLag(u=self.F2_y, T1=self.T1, T2=self.T2, zero_out=True)
self.LL2 = LeadLag(u=self.LL1_y, T1=self.T3, T2=self.T4, zero_out=True)

self.Vks = Gain(u=self.LL2_y, K=self.KS)

self.WO = WashoutOrLag(u=self.Vks_y, T=self.T6, K=self.T5, name='WO',
                      zero_out=True) # WO_y == Vss

self.VLIM = Limiter(u=self.WO_y, lower=self.LSMIN, upper=self.LSMAX,
                    info='Vss limiter')

self.Vss = Algeb(tex_name='V_{ss}', info='Voltage output before output limiter
→',
                 e_str='VLIM_zi * WO_y + VLIM_zu * LSMAX + VLIM_zl * LSMIN -
→Vss')

self.OLIM = Limiter(u=self.v, lower=self.VCLr, upper=self.VCUr,
                    info='output limiter')

self.vsout.e_str = 'OLIM_zi * Vss - vsout'
```

In the end, the output equation is assigned to `vsout.e_str`. It completes the equations of the IEEEEST model.

## Finalize

Assemble IEEEESTData and IEEEESTModel into IEEEEST:

```
class IEEEEST(IEEEESTData, IEEEESTModel):
    def __init__(self, system, config):
        IEEEESTData.__init__(self)
        IEEEESTModel.__init__(self, system, config)
```

Locate `andes/models/__init__.py`, in `non_jit`, find the key `pss` and add `IEEEEST` to its value list. In `non_jit`, keys are the `.py` file names under the folder `models`, and values are class names to be imported from that file. If the file name does not exist as a key in `non_jit`, add it after all prerequisite models. For example, `PSS` should be added after `exciters` (and `generators`, of course).

Finally, locate `andes/models/group.py`, check if the class with `PSS` exist. It is the name of `IEEEEST`'s group name. If not, create one by inheriting from `GroupBase`:

```
class PSS(GroupBase):
    """Power system stabilizer group."""

    def __init__(self):
        super().__init__()
        self.common_vars.extend(('vsout',))
```

where we added `vsout` to the `common_vars` list. All models in the `PSS` group must have a variable named `vsout`, which is defined in `PSSBase`.

This completes the `IEEEEST` model. When developing new models, use `andes prepare` to generate numerical code and start debugging.



### 4.1 Directory

ANDES comes with several test cases in the `andes/cases/` folder. Currently, the Kundur's 2-area system, IEEE 14-bus system, NPCC 140-bus system, and the WECC 179-bus system has been verified against DSATools TSAT.

The test case library will continue to build as more models get implemented.

A tree view of the test directory is as follows.

```
cases/
├── 5bus/
│   └── pjm5bus.xlsx
├── GBnetwork/
│   ├── GBnetwork.m
│   ├── GBnetwork.xlsx
│   └── README.md
├── ieee14/
│   ├── ieee14.dyr
│   └── ieee14.raw
└── kundur/
    ├── kundur.raw
    ├── kundur_aw.xlsx
    ├── kundur_coi.xlsx
    ├── kundur_coi_empty.xlsx
    ├── kundur_esdc2a.xlsx
    ├── kundur_esst3a.xlsx
    ├── kundur_exdc2_zero_tb.xlsx
    ├── kundur_exst1.xlsx
    └── kundur_freq.xlsx
```

(continues on next page)

(continued from previous page)

—	kundur_full.dyr
—	kundur_full.xlsx
—	kundur_gentrip.xlsx
—	kundur_ieeeeg1.xlsx
—	kundur_ieeeest.xlsx
—	kundur_sexs.xlsx
—	kundur_st2cut.xlsx
—	matpower/
—	case118.m
—	case14.m
—	case300.m
—	case5.m
—	nordic44/
—	N44_BC.dyr
—	N44_BC.raw
—	README.md
—	npcc/
—	npcc.raw
—	npcc_full.dyr
—	wecc/
—	wecc.raw
—	wecc.xlsx
—	wecc_full.dyr
—	wecc_gencls.dyr
—	wsc9/
—	wsc9.raw
—	wsc9.xlsx

## 4.2 MATPOWER Cases

MATPOWER cases have been tested in ANDES for power flow calculation. All following cases are calculated with the provided initial values using the full Newton-Raphson iterative approach.

The numerical library used for sparse matrix factorization is KLU. In addition, Jacobians are updated in place `spmatrix.ipadd`. Computations are performed on macOS 10.15.4 with i9-9980H, 16 GB 2400 MHz DDR4, running ANDES 0.9.1, CVXOPT 1.2.4 and NumPy 1.18.1.

The statistics of convergence, number of iterations, and solution time (including equation evaluation, Jacobian, and factorization time) are reported in the following table. The computation time may vary depending on operating system and hardware.

File Name	Converged?	# of Iterations	Time [s]
case30.m	1	3	0.012
case_ACTIVSg500.m	1	3	0.019
case13659pegase.m	1	5	0.531
case9Q.m	1	3	0.011
case_ACTIVSg200.m	1	2	0.013

Continued on next page

Table 1 – continued from previous page

File Name	Converged?	# of Iterations	Time [s]
case24_ieee_rts.m	1	4	0.014
case300.m	1	5	0.026
case6495rte.m	1	5	0.204
case39.m	1	1	0.009
case18.m	1	4	0.013
case_RTS_GMLC.m	1	3	0.014
case1951rte.m	1	3	0.047
case6ww.m	1	3	0.010
case5.m	1	3	0.010
case69.m	1	3	0.014
case6515rte.m	1	4	0.168
case2383wp.m	1	6	0.084
case30Q.m	1	3	0.011
case2868rte.m	1	4	0.074
case1354pegase.m	1	4	0.047
case2848rte.m	1	3	0.063
case4_dist.m	1	3	0.010
case6470rte.m	1	4	0.175
case2746wp.m	1	4	0.074
case_SyntheticUSA.m	1	21	11.120
case118.m	1	3	0.014
case30pwl.m	1	3	0.021
case57.m	1	3	0.017
case89pegase.m	1	5	0.024
case6468rte.m	1	6	0.232
case2746wop.m	1	4	0.075
case85.m	1	3	0.011
case22.m	1	2	0.008
case4gs.m	1	3	0.012
case14.m	1	2	0.010
case_ACTIVSg10k.m	1	4	0.251
case2869pegase.m	1	6	0.136
case_ieee30.m	1	2	0.010
case2737sop.m	1	5	0.087
case9target.m	1	5	0.013
case1888rte.m	1	2	0.037
case145.m	1	3	0.018
case_ACTIVSg2000.m	1	3	0.059
case_ACTIVSg70k.m	1	15	7.043
case9241pegase.m	1	6	0.497
case9.m	1	3	0.010
case141.m	1	3	0.012
case_ACTIVSg25k.m	1	7	1.040

Continued on next page

Table 1 – continued from previous page

File Name	Converged?	# of Iterations	Time [s]
case118.m	1	3	0.015
case1354pegase.m	1	4	0.048
case13659pegase.m	1	5	0.523
case14.m	1	2	0.011
case141.m	1	3	0.013
case145.m	1	3	0.017
case18.m	1	4	0.012
case1888rte.m	1	2	0.037
case1951rte.m	1	3	0.052
case22.m	1	2	0.011
case2383wp.m	1	6	0.086
case24_ieee_rts.m	1	4	0.015
case2736sp.m	1	4	0.074
case2737sop.m	1	5	0.108
case2746wop.m	1	4	0.093
case2746wp.m	1	4	0.089
case2848rte.m	1	3	0.065
case2868rte.m	1	4	0.079
case2869pegase.m	1	6	0.137
case30.m	1	3	0.033
case300.m	1	5	0.102
case30Q.m	1	3	0.013
case30pwl.m	1	3	0.013
case39.m	1	1	0.008
case4_dist.m	1	3	0.010
case4gs.m	1	3	0.010
case5.m	1	3	0.011
case57.m	1	3	0.015
case6468rte.m	1	6	0.229
case6470rte.m	1	4	0.170
case6495rte.m	1	5	0.198
case6515rte.m	1	4	0.169
case69.m	1	3	0.012
case6ww.m	1	3	0.011
case85.m	1	3	0.013
case89pegase.m	1	5	0.020
case9.m	1	3	0.010
case9241pegase.m	1	6	0.487
case9Q.m	1	3	0.013
case9target.m	1	5	0.015
case_ACTIVSg10k.m	1	4	0.257
case_ACTIVSg200.m	1	2	0.014
case_ACTIVSg2000.m	1	3	0.058

Continued on next page

Table 1 – continued from previous page

File Name	Converged?	# of Iterations	Time [s]
case_ACTIVSg25k.m	1	7	1.118
case_ACTIVSg500.m	1	3	0.027
case_ACTIVSg70k.m	1	15	6.931
case_RTS_GMLC.m	1	3	0.014
case_SyntheticUSA.m	1	21	11.103
case_ieee30.m	1	2	0.010
case3375wp.m	0	•	0.061
case33bw.m	0	•	0.007
case3120sp.m	0	•	0.037
case3012wp.m	0	•	0.082
case3120sp.m	0	•	0.039
case3375wp.m	0	•	0.059
case33bw.m	0	•	0.007



## CHAPTER 5

---

### Model References

---

Supported Groups and Models

Group	Models
ACLine	Line
ACTopology	Bus
Calculation	ACE, COI
Collection	Area
DCLink	Ground, R, L, C, RCp, RCs, RLs, RLCs, RLCp
DCTopology	Node
DG	PVD1
Exciter	EXDC2, IEEEEX1, ESDC2A, EXST1, ESST3A, SEXS
Experimental	PI2, TestDB1, TestPI, TestLagAWFreeze
FreqMeasurement	BusFreq, BusROCOF
Motor	Motor3, Motor5
PSS	IEEEEST, ST2CUT
PhasorMeasurement	PMU
RenAerodynamics	WTARA1
RenExciter	REECA1
RenGen	REGCA1
RenGovernor	WTDTA1, WTDS
RenPitch	WTPTA1
RenPlant	REPCA1
RenTorque	WTTQA1
StaticACDC	VSCShunt
StaticGen	PV, Slack
StaticLoad	PQ
StaticShunt	Shunt
SynGen	GENCLS, GENROU
TimedEvent	Toggler, Fault
TurbineGov	TG2, TGOV1, TGOV1DB, IEEEG1

## 5.1 ACLine

Common Parameters: u, name, bus1, bus2, r, x

Common Variables: v1, v2, a1, a2

Available models: *Line*

### 5.1.1 Line

Group *ACLine*

AC transmission line model.

To reduce the number of variables, line injections are summed at bus equations and are not stored. Current injections are not computed.



## Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
bus1		idx of from bus			IdxParam	
bus2		idx of to bus			IdxParam	
Sn	$S_n$	Power rating	100		NumParam	non_zero
fn	$f$	rated frequency	60		NumParam	
Vn1	$V_{n1}$	AC voltage rating	110		NumParam	non_zero
Vn2	$V_{n2}$	rated voltage of bus2	110		NumParam	non_zero
r	$r$	line resistance	0.000		NumParam	z
x	$x$	line reactance	0.000		NumParam	z
b		shared shunt susceptance	0		NumParam	y
g		shared shunt conductance	0		NumParam	y
b1	$b_1$	from-side susceptance	0		NumParam	
g1	$g_1$	from-side conductance	0		NumParam	
b2	$b_2$	to-side susceptance	0		NumParam	
g2	$g_2$	to-side conductance	0		NumParam	
trans		transformer branch flag	0		NumParam	
tap	$t_{ap}$	transformer branch tap ratio	1		NumParam	
phi	$\phi$	transformer branch phase shift in rad	0		NumParam	
owner		owner code			IdxParam	
xcoord		x coordinates			DataParam	
ycoord		y coordinates			DataParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
a1	$a_1$		phase angle of the from bus		
a2	$a_2$		phase angle of the to bus		
v1	$v_1$		voltage magnitude of the from bus		
v2	$v_2$		voltage magnitude of the to bus		

## Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
a1	$a_1$	ExtAl- geb	$u \left( -1/t_{ap} v_1 v_2 \left( -b_{hk} \sin(\phi - a_1 + a_2) + g_{hk} \cos(\phi - a_1 + a_2) \right) + 1/t_{ap}^2 v_1^2 (g_h + g_{hk}) \right)$
a2	$a_2$	ExtAl- geb	$u \left( -1/t_{ap} v_1 v_2 \left( b_{hk} \sin(\phi - a_1 + a_2) + g_{hk} \cos(\phi - a_1 + a_2) \right) + v_2^2 (g_h + g_{hk}) \right)$
v1	$v_1$	ExtAl- geb	$u \left( -1/t_{ap} v_1 v_2 \left( -b_{hk} \cos(\phi - a_1 + a_2) - g_{hk} \sin(\phi - a_1 + a_2) \right) - 1/t_{ap}^2 v_1^2 (b_h + b_{hk}) \right)$
v2	$v_2$	ExtAl- geb	$u \left( 1/t_{ap} v_1 v_2 \left( b_{hk} \cos(\phi - a_1 + a_2) - g_{hk} \sin(\phi - a_1 + a_2) \right) - v_2^2 (b_h + b_{hk}) \right)$

### Services

Name	Symbol	Equation	Type
gh	$g_h$	$0.5g + g_1$	ConstService
bh	$b_h$	$0.5b + b_1$	ConstService
gk	$g_k$	$0.5g + g_2$	ConstService
bk	$b_k$	$0.5b + b_2$	ConstService
yh	$y_h$	$u (ib_h + g_h)$	ConstService
yk	$y_k$	$u (ib_k + g_k)$	ConstService
yhk	$y_{hk}$	$\frac{u}{r+i(x+1.0 \cdot 10^{-8})+1.0 \cdot 10^{-8}}$	ConstService
ghk	$g_{hk}$	$\text{re}(y_{hk})$	ConstService
bhk	$b_{hk}$	$\text{im}(y_{hk})$	ConstService
itap	$1/t_{ap}$	$\frac{1}{t_{ap}}$	ConstService
itap2	$1/t_{ap}^2$	$\frac{1}{t_{ap}^2}$	ConstService

## 5.2 ACTopology

Common Parameters: u, name

Common Variables: a, v

Available models: *Bus*

### 5.2.1 Bus

Group *ACTopology*

AC Bus model.

Power balance equation have the form of  $\text{load} - \text{injection} = 0$ . Namely, load is positively summed, while injections are negative.

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
Vn	$V_n$	AC voltage rating	110	<i>kV</i>	NumParam	non_zero
vmax	$V_{max}$	Voltage upper limit	1.100	<i>p.u.</i>	NumParam	
vmin	$V_{min}$	Voltage lower limit	0.900	<i>p.u.</i>	NumParam	
v0	$V_0$	initial voltage magnitude	1	<i>p.u.</i>	NumParam	non_zero
a0	$\theta_0$	initial voltage phase angle	0	<i>rad</i>	NumParam	
xcoord		x coordinate (longitude)	0		DataParam	
ycoord		y coordinate (latitude)	0		DataParam	
area		Area code			IdxParam	
zone		Zone code			IdxParam	
owner		Owner code			IdxParam	

### Variables

Name	Symbol	Initial Value	Description	Unit	Properties
a	$\theta$	$\theta_0 (1 - z_{flat}) + 1.0 \cdot 10^{-8} z_{flat}$	voltage angle	<i>rad</i>	v_str
v	$V$	$V_0 (1 - z_{flat}) + z_{flat}$	voltage magnitude	<i>p.u.</i>	v_str

### Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
a	$\theta$	Algeb	0
v	$V$	Algeb	0

### Config Fields in [Bus]

Option	Symbol	Value	Info	Accepted values
flat_start	$z_{flat}$	0	flat start for voltages	(0, 1)

## 5.3 Calculation

Group of classes that calculates based on other models.

Common Parameters: u, name

Available models: *ACE*, *COI*

### 5.3.1 ACE

Group *Calculation*

Area Control Error model.

Note: area idx is automatically retrieved from *bus*.

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
bus		bus idx for freq. measurement			IdxParam	mandatory
bias	$\beta$	bias parameter	1	<i>MW/0.1Hz</i>	NumParam	power
busf		Optional BusFreq idx			IdxParam	
area			0		ExtParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
ace	$ace$		area control error	<i>MW (p.u.)</i>	
f	$f$		Bus frequency		

Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
ace	$ace$	Algeb	$10\beta (v^{f_s} - 1) - ace$
f	$f$	ExtAlgeb	0

Discrete

Name	Symbol	Type	Info
fs	$f_s$	Sampling	Sampled freq.

Config Fields in [ACE]

Option	Symbol	Value	Info	Accepted values
freq_model		BusFreq	default freq. measurement model	('BusFreq',)
interval		4	sampling time interval	
offset		0	sampling time offset	

### 5.3.2 COI

Group *Calculation*

Center of inertia calculation class.

## Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
M		Linearly stored SynGen.M	0		ExtParam	

## Variables

Name	Sym- bol	Initial Value	Description	Unit	Properties
wgen	$\omega_{gen}$		Linearly stored SynGen.omega		
agen	$\delta_{gen}$		Linearly stored SynGen.delta		
omega	$\omega_{coi}$	$\omega_{gen,0,avg}$	COI speed		v_str,v_setter
delta	$\delta_{coi}$	$\delta_{gen,0,avg}$	COI rotor angle		v_str,v_setter
omega_sub	$\omega_{sub}$		COI frequency contribution of each generator		
delta_sub	$\delta_{sub}$		COI angle contribution of each generator		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
wgen	$\omega_{gen}$	ExtState	0	
agen	$\delta_{gen}$	ExtState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
omega	$\omega_{coi}$	Algeb	$-\omega_{coi}$
delta	$\delta_{coi}$	Algeb	$-\delta_{coi}$
omega_sub	$\omega_{sub}$	ExtAlgeb	$M_w \omega_{gen}$
delta_sub	$\delta_{sub}$	ExtAlgeb	$M_w \delta_{gen}$

## Services

Name	Symbol	Equation	Type
Mw	$M_w$	$\frac{M}{M_{tr}}$	ConstService
d0w	$\delta_{gen,0,w}$	$M_w \delta_{gen,0}$	ConstService
a0w	$\omega_{gen,0,w}$	$M_w \omega_{gen,0}$	ConstService

## 5.4 Collection

Collection of topology models

Common Parameters: u, name

Available models: *Area*

### 5.4.1 Area

Group *Collection*

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	<i>u</i>	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
Vn			0		ExtParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
a	<i>a</i>		Bus voltage angle		
v	<i>v</i>		Bus voltage magnitude		

Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
a	<i>a</i>	ExtAlgeb	0
v	<i>v</i>	ExtAlgeb	0

## 5.5 DCLink

Basic DC links

Common Parameters: u, name

Available models: *Ground, R, L, C, RCp, RCs, RLs, RLCs, RLCp*

### 5.5.1 Ground

Group *DCLink*

Ground model that sets the voltage of the connected DC node.

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
node		Node index			IdxParam	mandatory
voltage	$V_0$	Ground voltage (typically 0)	0	<i>p.u.</i>	NumParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
Idc	$I_{dc}$	0	Fictitious current injection from ground		v_str
v	$v$				

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Idc	$I_{dc}$	Algeb	$u(-V_0 + v)$
v	$v$	ExtAlgeb	$-I_{dc}$

## 5.5.2 R

Group *DCLink*

Resistive dc line

## Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
node1		Node 1 index			IdxParam	mandatory
node2		Node 2 index			IdxParam	mandatory
Vdcn1	$V_{dcn1}$	DC voltage rating on node 1	100	<i>kV</i>	NumParam	non_zero
Vdcn2	$V_{dcn2}$	DC voltage rating on node 2	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
R		DC line resistance	0.010	<i>p.u.</i>	NumParam	non_zero,r

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
Idc	$I_{dc}$	$\frac{u(-v_1+v_2)}{R}$	Current from node 2 to 1	<i>p.u.</i>	v_str
v1	$v_1$		DC voltage on node 1		
v2	$v_2$		DC voltage on node 2		

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Idc	$I_{dc}$	Algeb	$-I_{dc} + \frac{u(-v_1+v_2)}{R}$
v1	$v_1$	ExtAlgeb	$-I_{dc}$
v2	$v_2$	ExtAlgeb	$I_{dc}$

## 5.5.3 L

Group *DCLink*

Inductive dc line

## Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
node1		Node 1 index			IdxParam	mandatory
node2		Node 2 index			IdxParam	mandatory
Vdcn1	$V_{dcn1}$	DC voltage rating on node 1	100	<i>kV</i>	NumParam	non_zero
Vdcn2	$V_{dcn2}$	DC voltage rating on node 2	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
L		DC line inductance	0.001	<i>p.u.</i>	NumParam	non_zero,r

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
IL	$I_L$	0	Inductance current	<i>p.u.</i>	v_str
v1	$v_1$		DC voltage on node 1		
v2	$v_2$		DC voltage on node 2		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
IL	$I_L$	State	$-\frac{u(v_1-v_2)}{L}$	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
v1	$v_1$	ExtAlgeb	$-I_L$
v2	$v_2$	ExtAlgeb	$I_L$



### 5.5.4 C

Group *DCLink*

Capacitive dc branch

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
node1		Node 1 index			IdxParam	mandatory
node2		Node 2 index			IdxParam	mandatory
Vdcn1	$V_{dcn1}$	DC voltage rating on node 1	100	<i>kV</i>	NumParam	non_zero
Vdcn2	$V_{dcn2}$	DC voltage rating on node 2	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
C		DC capacitance	0.001	<i>p.u.</i>	NumParam	non_zero,g

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
vC	$v_C$	0	Capacitor current	<i>p.u.</i>	v_str
Idc	$I_{dc}$	0	Current from node 2 to 1	<i>p.u.</i>	v_str
v1	$v_1$		DC voltage on node 1		
v2	$v_2$		DC voltage on node 2		

Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
vC	$v_C$	State	$-\frac{I_{dc}u}{C}$	

Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Idc	$I_{dc}$	Algeb	$I_{dc}(1 - u) + u(-v_1 + v_2 + v_C)$
v1	$v_1$	ExtAlgeb	$-I_{dc}$
v2	$v_2$	ExtAlgeb	$I_{dc}$

### 5.5.5 RCp

Group *DCLink*

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
node1		Node 1 index			IdxParam	mandatory
node2		Node 2 index			IdxParam	mandatory
Vdcn1	$V_{dcn1}$	DC voltage rating on node 1	100	<i>kV</i>	NumParam	non_zero
Vdcn2	$V_{dcn2}$	DC voltage rating on node 2	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
R	$R$	DC line resistance	0.010	<i>p.u.</i>	NumParam	non_zero,r
C	$C$	DC capacitance	0.001	<i>p.u.</i>	NumParam	non_zero,g

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
vC	$v_C$	$v_1 - v_2$	Capacitor current	<i>p.u.</i>	v_str
Idc	$I_{dc}$	$\frac{-v_1 + v_2}{R}$	Current from node 2 to 1	<i>p.u.</i>	v_str
v1	$v_1$		DC voltage on node 1		
v2	$v_2$		DC voltage on node 2		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
vC	$v_C$	State	$-\frac{u(I_{dc} - \frac{v_C}{R})}{C}$	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Idc	$I_{dc}$	Algeb	$I_{dc}(1 - u) + u(-v_1 + v_2 + v_C)$
v1	$v_1$	ExtAlgeb	$-I_{dc}$
v2	$v_2$	ExtAlgeb	$I_{dc}$

## 5.5.6 RCs

Group *DCLink*

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
node1		Node 1 index			IdxParam	mandatory
node2		Node 2 index			IdxParam	mandatory
Vdcn1	$V_{dcn1}$	DC voltage rating on node 1	100	<i>kV</i>	NumParam	non_zero
Vdcn2	$V_{dcn2}$	DC voltage rating on node 2	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
R	$R$	DC line resistance	0.010	<i>p.u.</i>	NumParam	non_zero,r
C	$C$	DC capacitance	0.001	<i>p.u.</i>	NumParam	non_zero,g

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
vC	$v_C$	$v_1 - v_2$	Capacitor current	<i>p.u.</i>	v_str
Idc	$I_{dc}$	$\frac{-v_1 + v_2}{R}$	Current from node 2 to 1	<i>p.u.</i>	v_str
v1	$v_1$		DC voltage on node 1		
v2	$v_2$		DC voltage on node 2		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
vC	$v_C$	State	$-\frac{I_{dc}u}{C}$	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Idc	$I_{dc}$	Algeb	$I_{dc}(1 - u) + u(-I_{dc}R - v_1 + v_2 + v_C)$
v1	$v_1$	ExtAlgeb	$-I_{dc}$
v2	$v_2$	ExtAlgeb	$I_{dc}$

## 5.5.7 RLs

Group *DCLink*

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
node1		Node 1 index			IdxParam	mandatory
node2		Node 2 index			IdxParam	mandatory
Vdcn1	$V_{dcn1}$	DC voltage rating on node 1	100	<i>kV</i>	NumParam	non_zero
Vdcn2	$V_{dcn2}$	DC voltage rating on node 2	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
R	$R$	DC line resistance	0.010	<i>p.u.</i>	NumParam	non_zero,r
L	$L$	DC line inductance	0.001	<i>p.u.</i>	NumParam	non_zero,r

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
IL	$I_L$	$\frac{v_1 - v_2}{R}$	Inductance current	<i>p.u.</i>	v_str
Idc	$I_{dc}$	$-\frac{u(v_1 - v_2)}{R}$	Current from node 2 to 1	<i>p.u.</i>	v_str
v1	$v_1$		DC voltage on node 1		
v2	$v_2$		DC voltage on node 2		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
IL	$I_L$	State	$\frac{u(-I_L R + v_1 - v_2)}{L}$	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Idc	$I_{dc}$	Algeb	$-I_L u - I_{dc}$
v1	$v_1$	ExtAlgeb	$-I_{dc}$
v2	$v_2$	ExtAlgeb	$I_{dc}$

## 5.5.8 RLCs

Group *DCLink*

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
node1		Node 1 index			IdxParam	mandatory
node2		Node 2 index			IdxParam	mandatory
Vdcn1	$V_{dcn1}$	DC voltage rating on node 1	100	<i>kV</i>	NumParam	non_zero
Vdcn2	$V_{dcn2}$	DC voltage rating on node 2	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
R	$R$	DC line resistance	0.010	<i>p.u.</i>	NumParam	non_zero,r
L	$L$	DC line inductance	0.001	<i>p.u.</i>	NumParam	non_zero,r
C	$C$	DC capacitance	0.001	<i>p.u.</i>	NumParam	non_zero,g

### Variables

Name	Symbol	Initial Value	Description	Unit	Properties
IL	$I_L$	0	Inductance current	<i>p.u.</i>	v_str
vC	$v_C$	$v_1 - v_2$	Capacitor current	<i>p.u.</i>	v_str
Idc	$I_{dc}$	0	Current from node 2 to 1	<i>p.u.</i>	v_str
v1	$v_1$		DC voltage on node 1		
v2	$v_2$		DC voltage on node 2		

### Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
IL	$I_L$	State	$\frac{u(-I_L R + v_1 - v_2 - v_C)}{L}$	
vC	$v_C$	State	$\frac{I_L u}{C}$	

### Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Idc	$I_{dc}$	Algeb	$-I_L - I_{dc}$
v1	$v_1$	ExtAlgeb	$-I_{dc}$
v2	$v_2$	ExtAlgeb	$I_{dc}$

## 5.5.9 RLCp

Group *DCLink*

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
node1		Node 1 index			IdxParam	mandatory
node2		Node 2 index			IdxParam	mandatory
Vdcn1	$V_{dcn1}$	DC voltage rating on node 1	100	<i>kV</i>	NumParam	non_zero
Vdcn2	$V_{dcn2}$	DC voltage rating on node 2	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
R	$R$	DC line resistance	0.010	<i>p.u.</i>	NumParam	non_zero,r
L	$L$	DC line inductance	0.001	<i>p.u.</i>	NumParam	non_zero,r
C	$C$	DC capacitance	0.001	<i>p.u.</i>	NumParam	non_zero,g

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
IL	$I_L$	0	Inductance current	<i>p.u.</i>	v_str
vC	$v_C$	$v_1 - v_2$	Capacitor current	<i>p.u.</i>	v_str
Idc	$I_{dc}$	$\frac{-v_1 + v_2}{R}$	Current from node 2 to 1	<i>p.u.</i>	v_str
v1	$v_1$		DC voltage on node 1		
v2	$v_2$		DC voltage on node 2		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
IL	$I_L$	State	$\frac{uv_C}{L}$	
vC	$v_C$	State	$-\frac{u(-I_L + I_{dc} - \frac{v_C}{R})}{C}$	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Idc	$I_{dc}$	Algeb	$I_{dc}(1 - u) + u(-v_1 + v_2 + v_C)$
v1	$v_1$	ExtAlgeb	$-I_{dc}$
v2	$v_2$	ExtAlgeb	$I_{dc}$

## 5.6 DCTopology

Common Parameters: u, name

Common Variables: v

Available models: *Node*

### 5.6.1 Node

Group *DCTopology*

DC Node model.

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
Vdcn	$V_{dcn}$	DC voltage rating	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
v0	$V_{dc0}$	initial voltage magnitude	1	<i>p.u.</i>	NumParam	
xcoord		x coordinate (longitude)	0		DataParam	
ycoord		y coordinate (latitude)	0		DataParam	
area		Area code			IdxParam	
zone		Zone code			IdxParam	
owner		Owner code			IdxParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
v	$V_{dc}$	$V_{dc0} (1 - z_{flat}) + z_{flat}$	voltage magnitude	<i>p.u.</i>	v_str

Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
v	$V_{dc}$	Algeb	0

Config Fields in [Node]

Option	Symbol	Value	Info	Accepted values
flat_start	$z_{flat}$	0	flat start for voltages	(0, 1)

## 5.7 DG

Distributed generation (small-scale).

Common Parameters: u, name

Available models: *PVD1*

### 5.7.1 PVD1

Group *DG*

Distributed PV model. (TODO: work in progress)

Power rating specified in  $S_n$ .

Reference: ESIG, WECC Distributed and Small PV Plants Generic Model (PVD1), [Online], Available: <https://www.esig.energy/wiki-main-page/wecc-distributed-and-small-pv-plants-generic-model-pvd1/>

Parameters



Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
bus		interface bus id			Idx- Param	manda- tory
gen		static generator index			Idx- Param	manda- tory
Sn	$S_n$	Model MVA base	100	<i>MVA</i>	NumParam	
xc	$x_c$	coupling reactance	0	<i>p.u.</i>	NumParam	z
igreg		Remote bus idx for droop response, None for lo- cal			Idx- Param	
qmx	$q_{mx}$	Max. reactive power command	0.330		NumParam	power
qmn	$q_{mn}$	Min. reactive power command	- 0.330		NumParam	power
v0	$v_0$	Lower limit of deadband for Vdroop response	0	<i>pu</i>	NumParam	
v1	$v_1$	Upper limit of deadband for Vdroop response	0	<i>pu</i>	NumParam	
dqdv	$dq/dv$	Q-V droop characteristics	0		NumParam	power
fdbd	$f_{dbd}$	frequency deviation deadband	- 0.010		NumParam	
ddn	$D_{dn}$	Gain after f deadband	0		NumParam	
ialim	$I_{alim}$	Apparent power limit	1.300		NumParam	current
vt0	$V_{t0}$	Voltage tripping response curve point 0	0.880		NumParam	
vt1	$V_{t1}$	Voltage tripping response curve point 1	0.900		NumParam	
vt2	$V_{t2}$	Voltage tripping response curve point 2	1.100		NumParam	
vt3	$V_{t3}$	Voltage tripping response curve point 3	1.200		NumParam	
vr- flag	$z_{VR}$	Voltage tripping is latching (0) or partially self- resetting (0-1)	0		NumParam	
ft0	$f_{t0}$	Frequency tripping response curve point 0	59.500		NumParam	
ft1	$f_{t1}$	Frequency tripping response curve point 1	59.700		NumParam	
ft2	$f_{t2}$	Frequency tripping response curve point 2	60.300		NumParam	
ft3	$f_{t3}$	Frequency tripping response curve point 3	60.500		NumParam	
fr- flag	$z_{FR}$	Frequency tripping is latching (0) or partially self-resetting (0-1)	0		NumParam	
tip	$T_{ip}$	Inverter active current lag time constant	0.020	<i>s</i>	NumParam	
tiq	$T_{iq}$	Inverter reactive current lag time constant	0.020	<i>s</i>	NumParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
v	$V$		bus (or igreg) terminal voltage	<i>p.u.</i>	
a	$\theta$		bus (or igreg) phase angle	<i>rad.</i>	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
v	$V$	ExtAlgeb	0
a	$\theta$	ExtAlgeb	0

## 5.8 Exciter

Exciter group for synchronous generators.

Common Parameters: u, name, syn

Common Variables: vout, vi

Available models: *EXDC2*, *IEEEX1*, *ESDC2A*, *EXST1*, *ESST3A*, *SEXS*

### 5.8.1 EXDC2

Group *Exciter*

EXDC2 model.

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
syn		Synchronous generator idx			IdxParam	mandatory
TR	$T_R$	Sensing time constant	0.010	<i>p.u.</i>	NumParam	
TA	$T_A$	Lag time constant in anti-windup lag	0.040	<i>p.u.</i>	NumParam	
TC	$T_C$	Lead time constant in lead-lag	1	<i>p.u.</i>	NumParam	
TB	$T_B$	Lag time constant in lead-lag	1	<i>p.u.</i>	NumParam	
TE	$T_E$	Exciter integrator time constant	0.800	<i>p.u.</i>	NumParam	
TF1	$T_{F1}$	Feedback washout time constant	1	<i>p.u.</i>	NumParam	non_zero
KF1	$K_{F1}$	Feedback washout gain	0.030	<i>p.u.</i>	NumParam	
KA	$K_A$	Gain in anti-windup lag TF	40	<i>p.u.</i>	NumParam	
KE	$K_E$	Gain added to saturation	1	<i>p.u.</i>	NumParam	
VR- MAX	$V_{RMAX}$	Maximum excitation limit	7.300	<i>p.u.</i>	NumParam	
VRMIN	$V_{RMIN}$	Minimum excitation limit	-7.300	<i>p.u.</i>	NumParam	
E1	$E_1$	First saturation point	0	<i>p.u.</i>	NumParam	
SE1	$S_{E1}$	Value at first saturation point	0	<i>p.u.</i>	NumParam	
E2	$E_2$	Second saturation point	1	<i>p.u.</i>	NumParam	
SE2	$S_{E2}$	Value at second saturation point	1	<i>p.u.</i>	NumParam	
Sn	$S_m$	Rated power from generator	0	<i>MVA</i>	ExtParam	
Vn	$V_m$	Rated voltage from generator	0	<i>kV</i>	ExtParam	
bus	$bus$	Bus idx of the generators	0		ExtParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
vp	$V_p$	$v_{f0}$	Voltage after saturation feedback, before speed term	$p.u.$	v_str
LS_y	$y_{LS}$	$1.0V$	State in lag transfer function		v_str
LL_x	$x'_{LL}$	$V_i$	State in lead-lag		v_str
LA_y	$y_{LA}$	$K_A y_{LL}$	State in lag TF		v_str
W_x	$x'_W$	$V_p$	State in washout filter		v_str
omega	$\omega$		Generator speed		
vout	$v_{out}$	$v_{f0}$	Exciter final output voltage		v_str
vref	$V_{ref}$	$V_{ref0}$	Reference voltage input	$p.u.$	v_str
Se	$S_e( V_{out} )$	$S_{e0}$	saturation output		v_str
vi	$V_i$	$V_{b0}$	Total input voltages	$p.u.$	v_str
LL_y	$y_{LL}$	$V_i$	Output of lead-lag		v_str
W_y	$y_W$	0	Output of washout filter		v_str
vf	$v_f$		Excitation field voltage to generator		
Xad-Ifd	$X_{ad}I_{fd}$		Armature excitation current		
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

## Differential Equations

Name	Symbol	Type	RHS of Equation " $\dot{x} = f(x, y)$ "	T (LHS)
vp	$V_p$	State	$-K_E V_p - S_e( V_{out} )V_p + y_{LA}$	$T_E$
LS_y	$y_{LS}$	State	$1.0V - y_{LS}$	$T_R$
LL_x	$x'_{LL}$	State	$V_i - x'_{LL}$	$T_B$
LA_y	$y_{LA}$	State	$K_A y_{LL} - y_{LA}$	$T_A$
W_x	$x'_W$	State	$V_p - x'_W$	$T_{F1}$
omega	$\omega$	ExtState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
vout	$v_{out}$	Algeb	$V_p \omega - v_{out}$
vref	$V_{ref}$	Algeb	$V_{ref0} - V_{ref}$
Se	$S_e( V_{out} )$	Algeb	$\frac{B_{SAT}^q z_0^{SL} (-A_{SAT}^q + V_p)^2}{V_p} - S_e( V_{out} )$
vi	$V_i$	Algeb	$-V_i + V_{ref} - y_{LS} - y_W$
LL_y	$y_{LL}$	Algeb	$LL_{LT1z1} LL_{LT2z1} (-x'_{LL} + y_{LL}) + T_B x'_{LL} - T_B y_{LL} + T_C (V_i - x'_{LL})$
W_y	$y_W$	Algeb	$K_{F1} (V_p - x'_W) - T_{F1} y_W$
vf	$v_f$	ExtAl- geb	$u(-v_{f0} + v_{out})$
Xad- Ifd	$X_{ad} I_{fd}$	ExtAl- geb	0
a	$\theta$	ExtAl- geb	0
v	$V$	ExtAl- geb	0

## Services

Name	Symbol	Equation	Type
SAT_E1	$E_{SAT}^{1c}$	$E_1$	ConstService
SAT_E2	$E_{SAT}^{2c}$	$E_2$	ConstService
SAT_SE1	$SE_{SAT}^{1c}$	$S_{E1}$	ConstService
SAT_SE2	$SE_{SAT}^{2c}$	$S_{E2} - 2z_{SAT}^{SE2} + 2$	ConstService
SAT_a	$a_{SAT}$	$\sqrt{\frac{E_{SAT}^{1c} SE_{SAT}^{1c}}{E_{SAT}^{2c} SE_{SAT}^{2c}}} ((SE_{SAT}^{2c} > 0) + (SE_{SAT}^{2c} < 0))$	ConstService
SAT_A	$A_{SAT}^q$	$E_{SAT}^{2c} - \frac{E_{SAT}^{1c} - E_{SAT}^{2c}}{a_{SAT} - 1}$	ConstService
SAT_B	$B_{SAT}^q$	$\frac{E_{SAT}^{2c} SE_{SAT}^{2c} (a_{SAT} - 1)^2 ((a_{SAT} > 0) + (a_{SAT} < 0))}{(E_{SAT}^{1c} - E_{SAT}^{2c})^2}$	ConstService
Se0	$S_{e0}$	$\frac{B_{SAT}^q (A_{SAT}^q - v_{f0})^2 (v_{f0} > A_{SAT}^q)}{v_{f0}}$	ConstService
vr0	$V_{r0}$	$v_{f0} (K_E + S_{e0})$	ConstService
vb0	$V_{b0}$	$\frac{V_{r0}}{K_A}$	ConstService
vref0	$V_{ref0}$	$V + V_{b0}$	ConstService

## Discrete

Name	Symbol	Type	Info
SL	$SL$	LessThan	
LL_LT1	$LT_{LL}$	LessThan	
LL_LT2	$LT_{LL}$	LessThan	
LA_lim	$lim_{LA}$	AntiWindup	Limiter in Lag

## Blocks

Name	Symbol	Type	Info
SAT	$SAT$	ExcQuadSat	Field voltage saturation
LS	$LS$	Lag	Sensing lag TF
LL	$LL$	LeadLag	Lead-lag for internal delays
LA	$LA$	LagAntiWindup	Anti-windup lag
W	$W$	Washout	Signal conditioner

## 5.8.2 IEEEEX1

Group *Exciter*

IEEEEX1 Type 1 exciter (DC)

Derived from EXDC2 by varying the limiter bounds.

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
syn		Synchronous generator idx			IdxParam	mandatory
TR	$T_R$	Sensing time constant	0.010	<i>p.u.</i>	NumParam	
TA	$T_A$	Lag time constant in anti-windup lag	0.040	<i>p.u.</i>	NumParam	
TC	$T_C$	Lead time constant in lead-lag	1	<i>p.u.</i>	NumParam	
TB	$T_B$	Lag time constant in lead-lag	1	<i>p.u.</i>	NumParam	
TE	$T_E$	Exciter integrator time constant	0.800	<i>p.u.</i>	NumParam	
TF1	$T_{F1}$	Feedback washout time constant	1	<i>p.u.</i>	NumParam	non_zero
KF1	$K_{F1}$	Feedback washout gain	0.030	<i>p.u.</i>	NumParam	
KA	$K_A$	Gain in anti-windup lag TF	40	<i>p.u.</i>	NumParam	
KE	$K_E$	Gain added to saturation	1	<i>p.u.</i>	NumParam	
VR-MAX	$V_{RMAX}$	Maximum excitation limit	7.300	<i>p.u.</i>	NumParam	
VRMIN	$V_{RMIN}$	Minimum excitation limit	-7.300	<i>p.u.</i>	NumParam	
E1	$E_1$	First saturation point	0	<i>p.u.</i>	NumParam	
SE1	$S_{E1}$	Value at first saturation point	0	<i>p.u.</i>	NumParam	
E2	$E_2$	Second saturation point	1	<i>p.u.</i>	NumParam	
SE2	$S_{E2}$	Value at second saturation point	1	<i>p.u.</i>	NumParam	
Sn	$S_m$	Rated power from generator	0	<i>MVA</i>	ExtParam	
Vn	$V_m$	Rated voltage from generator	0	<i>kV</i>	ExtParam	
bus	$bus$	Bus idx of the generators	0		ExtParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
vp	$V_p$	$v_{f0}$	Voltage after saturation feedback, before speed term	<i>p.u.</i>	v_str
LS_y	$y_{LS}$	$1.0V$	State in lag transfer function		v_str
LL_x	$x'_{LL}$	$V_i$	State in lead-lag		v_str
LA_y	$y_{LA}$	$K_A y_{LL}$	State in lag TF		v_str
W_x	$x'_W$	$V_p$	State in washout filter		v_str
omega	$\omega$		Generator speed		
vout	$v_{out}$	$v_{f0}$	Exciter final output voltage		v_str
vref	$V_{ref}$	$V_{ref0}$	Reference voltage input	<i>p.u.</i>	v_str
Se	$S_e( V_{out} )$	$S_{e0}$	saturation output		v_str
vi	$V_i$	$V_{b0}$	Total input voltages	<i>p.u.</i>	v_str
LL_y	$y_{LL}$	$V_i$	Output of lead-lag		v_str
W_y	$y_W$	0	Output of washout filter		v_str
vf	$v_f$		Excitation field voltage to generator		
Xad-Ifd	$X_{ad}I_{fd}$		Armature excitation current		
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
vp	$V_p$	State	$-K_E V_p - S_e( V_{out} )V_p + y_{LA}$	$T_E$
LS_y	$y_{LS}$	State	$1.0V - y_{LS}$	$T_R$
LL_x	$x'_{LL}$	State	$V_i - x'_{LL}$	$T_B$
LA_y	$y_{LA}$	State	$K_A y_{LL} - y_{LA}$	$T_A$
W_x	$x'_W$	State	$V_p - x'_W$	$T_{F1}$
omega	$\omega$	ExtState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
vout	$v_{out}$	Algeb	$V_p - v_{out}$
vref	$V_{ref}$	Algeb	$V_{ref0} - V_{ref}$
Se	$S_e( V_{out} )$	Algeb	$\frac{B_{SAT}^q z_0^{SL} (-A_{SAT}^q + V_p)^2}{V_p} - S_e( V_{out} )$
vi	$V_i$	Algeb	$-V_i + V_{ref} - y_{LS} - y_W$
LL_y	$y_{LL}$	Algeb	$LL_{LT1z1} LL_{LT2z1} (-x'_{LL} + y_{LL}) + T_B x'_{LL} - T_B y_{LL} + T_C (V_i - x'_{LL})$
W_y	$y_W$	Algeb	$K_{F1} (V_p - x'_W) - T_{F1} y_W$
vf	$v_f$	ExtAl- geb	$u(-v_{f0} + v_{out})$
Xad- Ifd	$X_{ad} I_{fd}$	ExtAl- geb	0
a	$\theta$	ExtAl- geb	0
v	$V$	ExtAl- geb	0

## Services

Name	Symbol	Equation	Type
SAT_E1	$E_{SAT}^{1c}$	$E_1$	ConstService
SAT_E2	$E_{SAT}^{2c}$	$E_2$	ConstService
SAT_SE1	$SE_{SAT}^{1c}$	$S_{E1}$	ConstService
SAT_SE2	$SE_{SAT}^{2c}$	$S_{E2} - 2z_{SAT}^{SE_2^2} + 2$	ConstService
SAT_a	$a_{SAT}$	$\sqrt{\frac{E_{SAT}^{1c} SE_{SAT}^{1c}}{E_{SAT}^{2c} SE_{SAT}^{2c}}} ((SE_{SAT}^{2c} > 0) + (SE_{SAT}^{2c} < 0))$	ConstService
SAT_A	$A_{SAT}^q$	$E_{SAT}^{2c} - \frac{E_{SAT}^{1c} - E_{SAT}^{2c}}{a_{SAT} - 1}$	ConstService
SAT_B	$B_{SAT}^q$	$\frac{E_{SAT}^{2c} SE_{SAT}^{2c} (a_{SAT} - 1)^2 ((a_{SAT} > 0) + (a_{SAT} < 0))}{(E_{SAT}^{1c} - E_{SAT}^{2c})^2}$	ConstService
Se0	$S_{e0}$	$\frac{B_{SAT}^q (A_{SAT}^q - v_{f0})^2 (v_{f0} > A_{SAT}^q)}{v_{f0}}$	ConstService
vr0	$V_{r0}$	$v_{f0} (K_E + S_{e0})$	ConstService
vb0	$V_{b0}$	$\frac{V_{r0}}{K_A}$	ConstService
vref0	$V_{ref0}$	$V + V_{b0}$	ConstService
VRTMAX	$V_{RMAX} V_T$	$V V_{RMAX}$	VarService
VRTMIN	$V_{RMIN} V_T$	$V V_{RMIN}$	VarService

## Discrete

Name	Symbol	Type	Info
SL	$SL$	LessThan	
LL_LT1	$LT_{LL}$	LessThan	
LL_LT2	$LT_{LL}$	LessThan	
LA_lim	$lim_{LA}$	AntiWindup	Limiter in Lag

## Blocks



Name	Symbol	Type	Info
SAT	<i>SAT</i>	ExcQuadSat	Field voltage saturation
LS	<i>LS</i>	Lag	Sensing lag TF
LL	<i>LL</i>	LeadLag	Lead-lag for internal delays
LA	<i>LA</i>	LagAntiWindup	Anti-windup lag
W	<i>W</i>	Washout	Signal conditioner

### 5.8.3 ESDC2A

Group *Exciter*

ESDC2A model.

This model is implemented as described in the PSS/E manual, except that the HVGate is not in use. Due to the HVGate and saturation function, the results are close to but different from TSAT.

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
syn		Synchronous generator idx			IdxParam	mandatory
TR	$T_R$	Sensing time constant	0.010	<i>p.u.</i>	NumParam	
KA	$K_A$	Regulator gain	80		NumParam	
TA	$T_A$	Lag time constant in regulator	0.040	<i>p.u.</i>	NumParam	
TB	$T_B$	Lag time constant in lead-lag	1	<i>p.u.</i>	NumParam	
TC	$T_C$	Lead time constant in lead-lag	1	<i>p.u.</i>	NumParam	
VR- MAX	$V_{RMAX}$	Max. exc. limit (0-unlimited)	7.300	<i>p.u.</i>	NumParam	
VRMIN	$V_{RMIN}$	Min. excitation limit	-7.300	<i>p.u.</i>	NumParam	
KE	$K_E$	Saturation feedback gain	1	<i>p.u.</i>	NumParam	
TE	$T_E$	Integrator time constant	0.800	<i>p.u.</i>	NumParam	
KF	$K_F$	Feedback gain	0.100		NumParam	
TF1	$T_{F1}$	Feedback washout time constant	1	<i>p.u.</i>	NumParam	positive
Switch	$S_w$	Switch that PSS/E did not imple- ment	0	<i>bool</i>	NumParam	
E1	$E_1$	First saturation point	0	<i>p.u.</i>	NumParam	
SE1	$S_{E1}$	Value at first saturation point	0	<i>p.u.</i>	NumParam	
E2	$E_2$	Second saturation point	0	<i>p.u.</i>	NumParam	
SE2	$S_{E2}$	Value at second saturation point	0	<i>p.u.</i>	NumParam	
Sn	$S_m$	Rated power from generator	0	<i>MVA</i>	ExtParam	
Vn	$V_m$	Rated voltage from generator	0	<i>kV</i>	ExtParam	
bus	$bus$	Bus idx of the generators	0		ExtParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
LG_y	$y_{LG}$	$V$	State in lag transfer function		v_str
LL_x	$x'_{LL}$	$V_i$	State in lead-lag		v_str
LA_y	$y_{LA}$	$K_{AYLL}$	State in lag TF		v_str
INT_y	$y_{INT}$	$v_{f0}$	Integrator output		v_str
WF_x	$x'_{WF}$	$y_{INT}$	State in washout filter		v_str
omega	$\omega$		Generator speed		
vout	$v_{out}$	$v_{f0}$	Exciter final output voltage		v_str
vref	$V_{ref}$	$V_{ref0}$	Reference voltage input	<i>p.u.</i>	v_str
vi	$V_i$	$-V + V_{ref0}$	Total input voltages	<i>p.u.</i>	v_str
LL_y	$y_{LL}$	$V_i$	Output of lead-lag		v_str
UEL	$U_{EL}$	0	Interface var for under exc. limiter		v_str
HG_y	$y_{HG}$	$HG_{sls0}U_{EL} + HG_{sls1}y_{LL}$	HVGate output		v_str
Se	$S_e( V_{out} )$	$S_{e0}$	saturation output		v_str
VFE	$V_{FE}$	$V_{FE0}$	Combined saturation feedback	<i>p.u.</i>	v_str
WF_y	$y_{WF}$	0	Output of washout filter		v_str
vf	$v_f$		Excitation field voltage to generator		
Xad-Ifd	$X_{ad}I_{fd}$		Armature excitation current		
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

### Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
LG_y	$y_{LG}$	State	$V - y_{LG}$	$T_R$
LL_x	$x'_{LL}$	State	$V_i - x'_{LL}$	$T_B$
LA_y	$y_{LA}$	State	$K_{AYLL} - y_{LA}$	$T_A$
INT_y	$y_{INT}$	State	$-V_{FE} + y_{LA}$	$T_E$
WF_x	$x'_{WF}$	State	$-x'_{WF} + y_{INT}$	$T_{F1}$
omega	$\omega$	ExtState	0	

### Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
vout	$v_{out}$	Algeb	$-v_{out} + y_{INT}$
vref	$V_{ref}$	Algeb	$V_{ref0} - V_{ref}$
vi	$V_i$	Algeb	$-V - V_i + V_{ref} - y_{WF}$
LL_y	$y_{LL}$	Algeb	$LL_{LT1z1}LL_{LT2z1}(-x'_{LL} + y_{LL}) + T_Bx'_{LL} - T_By_{LL} + T_C(V_i - x'_{LL})$
UEL	$U_{EL}$	Algeb	$-U_{EL}$
HG_y	$y_{HG}$	Algeb	$HG_{sls0}U_{EL} + HG_{sls1}y_{LL} - y_{HG}$
Se	$S_e( V_{out} )$	Algeb	$\frac{B_{SAT}^q z_0^{SL} (-A_{SAT}^q + y_{INT})^2}{y_{INT}} - S_e( V_{out} )$
VFE	$V_{FE}$	Algeb	$-V_{FE} + y_{INT}(K_E + S_e( V_{out} ))$
WF_y	$y_{WF}$	Algeb	$K_F(-x'_{WF} + y_{INT}) - T_{F1}y_{WF}$
vf	$v_f$	ExtAl- geb	$u(-v_{f0} + v_{out})$
Xad- Ifd	$X_{ad}I_{fd}$	ExtAl- geb	0
a	$\theta$	ExtAl- geb	0
v	$V$	ExtAl- geb	0

## Services

Name	Symbol	Equation	Type
VRMAXc	$VRMAXc$	$VRMAX - 999z_{VRMAX} + 999$	ConstService
SAT_E1	$E_{SAT}^{1c}$	$E_1$	ConstService
SAT_E2	$E_{SAT}^{2c}$	$E_2$	ConstService
SAT_SE1	$SE_{SAT}^{1c}$	$SE_1$	ConstService
SAT_SE2	$SE_{SAT}^{2c}$	$SE_2 - 2z_{SAT}^{SE2} + 2$	ConstService
SAT_a	$a_{SAT}$	$\sqrt{\frac{E_{SAT}^{1c} SE_{SAT}^{1c}}{E_{SAT}^{2c} SE_{SAT}^{2c}}} ((SE_{SAT}^{2c} > 0) + (SE_{SAT}^{2c} < 0))$	ConstService
SAT_A	$A_{SAT}^q$	$E_{SAT}^{2c} - \frac{E_{SAT}^{1c} - E_{SAT}^{2c}}{a_{SAT} - 1}$	ConstService
SAT_B	$B_{SAT}^q$	$\frac{E_{SAT}^{2c} SE_{SAT}^{2c} (a_{SAT} - 1)^2 ((a_{SAT} > 0) + (a_{SAT} < 0))}{(E_{SAT}^{1c} - E_{SAT}^{2c})^2}$	ConstService
Se0	$S_{e0}$	$\frac{B_{SAT}^q (A_{SAT}^q - v_{f0})^2 (v_{f0} > A_{SAT}^q)}{v_{f0}}$	ConstService
vfe0	$V_{FE0}$	$v_{f0}(K_E + S_{e0})$	ConstService
vref0	$V_{ref0}$	$V + \frac{V_{FE0}}{K_A}$	ConstService
VRU	$V_T V_{RMAX}$	$V V_{RMAX} c$	VarService
VRL	$V_T V_{RMIN}$	$V V_{RMIN}$	VarService

## Discrete

Name	Symbol	Type	Info
LL_LT1	$LT_{LL}$	LessThan	
LL_LT2	$LT_{LL}$	LessThan	
HG_sl	$None_{HG}$	Selector	HVGate Selector
LA_lim	$lim_{LA}$	AntiWindup	Limiter in Lag
SL	$SL$	LessThan	

## Blocks

Name	Symbol	Type	Info
LG	$LG$	Lag	Transducer delay
SAT	$SAT$	ExcQuadSat	Field voltage saturation
LL	$LL$	LeadLag	Lead-lag compensator
HG	$HG$	HVGate	HVGate for under excitation
LA	$LA$	LagAntiWindup	Anti-windup lag
INT	$INT$	Integrator	Integrator
WF	$WF$	Washout	Feedback to input

## 5.8.4 EXST1

### Group *Exciter*

EXST1-type static excitation system.

### Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
syn		Synchronous generator idx			IdxParam	mandatory
TR	$T_R$	Measurement delay	0.010		NumParam	
VIMAX	$V_{IMAX}$	Max. input voltage	0.200		NumParam	
VIMIN	$V_{IMIN}$	Min. input voltage	0		NumParam	
TC	$T_C$	LL numerator	1		NumParam	
TB	$T_B$	LL denominator	1		NumParam	
KA	$K_A$	Regulator gain	80		NumParam	
TA	$T_A$	Regulator delay	0.050		NumParam	
VRMAX	$V_{RMAX}$	Max. regulator output	8		NumParam	
VRMIN	$V_{RMIN}$	Min. regulator output	-3		NumParam	
KC	$K_C$	Coef. for Ifd	0.200		NumParam	
KF	$K_F$	Feedback gain	0.100		NumParam	
TF	$T_F$	Feedback delay	1		NumParam	positive
Sn	$S_m$	Rated power from generator	0	<i>MVA</i>	ExtParam	
Vn	$V_m$	Rated voltage from generator	0	<i>kV</i>	ExtParam	
bus	$bus$	Bus idx of the generators	0		ExtParam	

## Variables

Name	Sym- bol	Initial Value	Description	Unit	Prop- ties
LG_y	$y_{LG}$	$V$	State in lag transfer function		v_str
LL_x	$x'_{LL}$	$V_l$	State in lead-lag		v_str
LR_y	$y_{LR}$	$K_A y_{LL}$	State in lag transfer function		v_str
WF_x	$x'_{WF}$	$y_{LR}$	State in washout filter		v_str
omega	$\omega$		Generator speed		
vout	$v_{out}$	$v_{f0}$	Exciter final output voltage		v_str
vref	$V_{ref}$	$V_{ref0}$	Reference voltage input	<i>p.u.</i>	v_str
vi	$V_i$	$\frac{v_{f0}}{K_A}$	Total input voltages	<i>p.u.</i>	v_str
vl	$V_l$	$V_i z_i^{HLI} + V_{IMAX} z_u^{HLI} + V_{IMIN} z_l^{HLI}$	Input after limiter		v_str
LL_y	$y_{LL}$	$V_l$	Output of lead-lag		v_str
WF_y	$y_{WF}$	0	Output of washout filter		v_str
vf-max	$V_{fmax}$	$-K_C X_{ad} I_{fd} + V_{RMAX}$	Upper bound of output limiter		v_str
vfmin	$V_{fmin}$	$-K_C X_{ad} I_{fd} + V_{RMIN}$	Lower bound of output limiter		v_str
vf	$v_f$		Excitation field voltage to generator		
Xad-Ifd	$X_{ad} I_{fd}$		Armature excitation current		
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
LG_y	$y_{LG}$	State	$V - y_{LG}$	$T_R$
LL_x	$x'_{LL}$	State	$V_l - x'_{LL}$	$T_B$
LR_y	$y_{LR}$	State	$K_A y_{LL} - y_{LR}$	$T_A$
WF_x	$x'_{WF}$	State	$-x'_{WF} + y_{LR}$	$T_F$
omega	$\omega$	ExtState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
vout	$v_{out}$	Algeb	$V_{fmax}z_u^{HLR} + V_{fmin}z_l^{HLR} - v_{out} + y_{LR}z_i^{HLR}$
vref	$V_{ref}$	Algeb	$V_{ref0} - V_{ref}$
vi	$V_i$	Algeb	$-V_i + V_{ref} - y_{LG} - y_{WF}$
vl	$V_l$	Algeb	$V_i z_i^{HLI} - V_l + V_{IMAX} z_u^{HLI} + V_{IMIN} z_l^{HLI}$
LL_y	$y_{LL}$	Algeb	$LL_{LT1z1} LL_{LT2z1} (-x'_{LL} + y_{LL}) + T_B x'_{LL} - T_B y_{LL} + T_C (V_l - x'_{LL})$
WF_y	$y_{WF}$	Algeb	$K_F (-x'_{WF} + y_{LR}) - T_F y_{WF}$
vfmax	$V_{fmax}$	Algeb	$-K_C X_{ad} I_{fd} + V_{RMAX} - V_{fmax}$
vfmin	$V_{fmin}$	Algeb	$-K_C X_{ad} I_{fd} + V_{RMIN} - V_{fmin}$
vf	$v_f$	ExtAlgeb	$u(-v_{f0} + v_{out})$
Xad-Ifd	$X_{ad} I_{fd}$	ExtAlgeb	0
a	$\theta$	ExtAlgeb	0
v	$V$	ExtAlgeb	0

## Services

Name	Symbol	Equation	Type
vref0	$V_{ref0}$	$V + \frac{v_{f0}}{K_A}$	ConstService

## Discrete

Name	Symbol	Type	Info
HLI	$HLI$	HardLimiter	Hard limiter on input
LL_LT1	$LT_{LL}$	LessThan	
LL_LT2	$LT_{LL}$	LessThan	
HLR	$HLR$	HardLimiter	Hard limiter on regulator output

## Blocks

Name	Symbol	Type	Info
LG	$LG$	Lag	Sensing delay
LL	$LL$	LeadLag	Lead-lag compensator
LR	$LR$	Lag	Regulator
WF	$WF$	Washout	Stablizing circuit feedback

### 5.8.5 ESST3A

#### Group *Exciter*

Static exciter type 3A model

## Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
syn		Synchronous generator idx			Idx- Param	manda- tory
TR	$T_R$	Sensing time constant	0.010	<i>p.u.</i>	NumParam	
VI- MAX	$V_{IMAX}$	Max. input voltage	0.800		NumParam	
VIMIN	$V_{IMIN}$	Min. input voltage	- 0.100		NumParam	
KM	$K_M$	Forward gain constant	500		NumParam	
TC	$T_C$	Lead time constant in lead-lag	3		NumParam	
TB	$T_B$	Lag time constant in lead-lag	15		NumParam	
KA	$K_A$	Gain in anti-windup lag TF	50		NumParam	
TA	$T_A$	Lag time constant in anti-windup lag	0.100		NumParam	
VR- MAX	$V_{RMAX}$	Maximum excitation limit	8	<i>p.u.</i>	NumParam	
VR- MIN	$V_{RMIN}$	Minimum excitation limit	0	<i>p.u.</i>	NumParam	
KG	$K_G$	Feedback gain of inner field regulator	1		NumParam	
KP	$K_P$	Potential circuit gain coeff.	4		NumParam	
KI	$K_I$	Potential circuit gain coeff.	0.100		NumParam	
VB- MAX	$V_{BMAX}$	VB upper limit	18	<i>p.u.</i>	NumParam	
KC	$K_C$	Rectifier loading factor proportional to com- mutating reactance	0.100		NumParam	
XL	$X_L$	Potential source reactance	0.010		NumParam	
VG- MAX	$V_{GMAX}$	VG upper limit	4	<i>p.u.</i>	NumParam	
THETAP	$\theta_P$	Rectifier firing angle	0	<i>de- gree</i>	NumParam	
TM	$K_C$	Inner field regulator forward time constant	0.100		NumParam	
VM- MAX	$V_{MMAX}$	Maximum VM limit	1	<i>p.u.</i>	NumParam	
VM- MIN	$V_{RMIN}$	Minimum VM limit	0.100	<i>p.u.</i>	NumParam	
Sn	$S_m$	Rated power from generator	0	<i>MVA</i>	ExtParam	
Vn	$V_m$	Rated voltage from generator	0	<i>kV</i>	ExtParam	
bus	<i>bus</i>	Bus idx of the generators	0		ExtParam	



## Variables

Name	Sym- bol	Initial Value	Description	Unit	Prop- erties
LG_y	$y_{LG}$	$V$	State in lag transfer function		v_str
LL_x	$x'_{LL}$	$y_{HG}$	State in lead-lag		v_str
LAW1_y	$y_{LAW1}$	$K_{AYLL}$	State in lag TF		v_str
LAW2_y	$y_{LAW2}$	$K_M V_{RS}$	State in lag TF		v_str
omega	$\omega$		Generator speed		
vout	$v_{out}$	$v_{f0}$	Exciter final output voltage		v_str
UEL	$U_{EL}$	0	Interface var for under exc. limiter		v_str
IN	$I_N$	$\frac{K_C X_{ad} I_{fd}}{V_E}$	Input to FEX		v_str
FEX_y	$y_{FEX}$	$\begin{cases} 1 & \text{for } I_N \leq 0 \\ 1 - 0.577 I_N & \text{for } I_N \leq 0.433 \\ \sqrt{0.75 - I_N^2} & \text{for } I_N \leq 0.707 \\ 1.732 - 1.732 I_N & \text{for } I_N \leq 1 \\ 0 & \text{otherwise} \end{cases}$	Output of piecewise		v_str
VB_x	$x_{VB}$	$V_E y_{FEX}$	Gain output before limiter		v_str
VB_y	$y_{VB}$	$\frac{V_{Blimzi} x_{VB}}{V_{Blimzu} V_{BMAX}}$	Gain output after limiter		v_str
VG_x	$x_{VG}$	$K_G v_{out}$	Gain output before limiter		v_str
VG_y	$y_{VG}$	$\frac{V_{Glimzi} x_{VG}}{V_{Glimzu} V_{GMAX}}$	Gain output after limiter		v_str
vrs	$V_{RS}$	$\frac{v_{f0}}{K_M y_{VB}}$	VR subtract feedback VG		v_str
vref	$V_{ref}$	$V + \frac{V_{RS} + y_{VG}}{K_A}$	Reference voltage input	p.u.	v_str
vi	$V_i$	$-V + V_{ref}$	Total input voltages	p.u.	v_str
vil	$V_{il}$	$V_i z_i^{HLI} + V_{IMAX} z_u^{HLI} + V_{IMIN} z_l^{HLI}$	Input voltage after limit		v_str
HG_y	$y_{HG}$	$HG_{sls0} U_{EL} + HG_{sls1} V_{il}$	HVGate output		v_str
LL_y	$y_{LL}$	$y_{HG}$	Output of lead-lag		v_str
vf	$v_f$		Excitation field voltage to generator		
XadIfd	$X_{ad} I_{fd}$		Armature excitation current		
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		
vd	$V_d$		d-axis machine voltage		
vq	$V_q$		q-axis machine voltage		
Id	$I_d$		d-axis machine current		
Iq	$I_q$		q-axis machine current		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
LG_y	$y_{LG}$	State	$V - y_{LG}$	$T_R$
LL_x	$x'_{LL}$	State	$-x'_{LL} + y_{HG}$	$T_B$
LAW1_y	$y_{LAW1}$	State	$K_A y_{LL} - y_{LAW1}$	$T_A$
LAW2_y	$y_{LAW2}$	State	$K_M V_{RS} - y_{LAW2}$	$K_C$
omega	$\omega$	ExtState	0	

Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
vout	$v_{out}$	Algeb	$-v_{out} + y_{LAW2}y_{VB}$
UEL	$U_{EL}$	Algeb	$-U_{EL}$
IN	$I_N$	Algeb	$-I_N + \frac{K_C X_{ad} I_{fd}}{V_E}$
FEX_y	$y_{FEX}$	Algeb	$-y_{FEX} + \begin{cases} 1 & \text{for } I_N \leq 0 \\ 1 - 0.577 I_N & \text{for } I_N \leq 0.433 \\ \sqrt{0.75 - I_N^2} & \text{for } I_N \leq 0.75 \\ 1.732 - 1.732 I_N & \text{for } I_N \leq 1 \\ 0 & \text{otherwise} \end{cases}$
VB_x	$x_{VB}$	Algeb	$V_E y_{FEX} - x_{VB}$
VB_y	$y_{VB}$	Algeb	$V B_{limzi} x_{VB} + V B_{limzu} V_{BMAX} - y_{VB}$
VG_x	$x_{VG}$	Algeb	$K_G v_{out} - x_{VG}$
VG_y	$y_{VG}$	Algeb	$V G_{limzi} x_{VG} + V G_{limzu} V_{GMAX} - y_{VG}$
vrs	$V_{RS}$	Algeb	$-V_{RS} + y_{LAW1} - y_{VG}$
vref	$V_{ref}$	Algeb	$V_{ref0} - V_{ref}$
vi	$V_i$	Algeb	$-V_i + V_{ref} - y_{LG}$
vil	$V_{il}$	Algeb	$V_i z_i^{HLI} + V_{IMAX} z_u^{HLI} + V_{IMIN} z_l^{HLI} - V_{il}$
HG_y	$y_{HG}$	Algeb	$HG_{sls0} U_{EL} + HG_{sls1} V_{il} - y_{HG}$
LL_y	$y_{LL}$	Algeb	$LL_{LT1z1} LL_{LT2z1} (-x'_{LL} + y_{LL}) + T_B x'_{LL} - T_B y_{LL} + T_C (-x'_{LL} + y_{HG})$
vf	$v_f$	ExtAlgeb	$u(-v_{f0} + v_{out})$
Xad-Ifd	$X_{ad} I_{fd}$	ExtAlgeb	0
a	$\theta$	ExtAlgeb	0
v	$V$	ExtAlgeb	0
vd	$V_d$	ExtAlgeb	0
vq	$V_q$	ExtAlgeb	0
Id	$I_d$	ExtAlgeb	0
Iq	$I_q$	ExtAlgeb	0

## Services

Name	Symbol	Equation	Type
KPC	$K_{PC}$	$K_P e^{i \text{radians}(\theta_P)}$	ConstService
VE	$V_E$	$ K_{PC}(V_d + iV_q) + i(I_d + iI_q)(K_I + K_{PC}X_L) $	VarService
vref0	$V_{ref0}$	$V_{ref}$	PostInitService

Discrete

Name	Symbol	Type	Info
VB_lim	$lim_{VB}$	HardLimiter	
VG_lim	$lim_{VG}$	HardLimiter	
HG_sl	$None_{HG}$	Selector	HVGate Selector
LL_LT1	$LT_{LL}$	LessThan	
LL_LT2	$LT_{LL}$	LessThan	
LAW1_lim	$lim_{LAW1}$	AntiWindup	Limiter in Lag
HLI	$HLI$	HardLimiter	Input limiter
LAW2_lim	$lim_{LAW2}$	AntiWindup	Limiter in Lag

Blocks

Name	Symbol	Type	Info
LG	$LG$	Lag	Voltage transducer
FEX	$FEX$	Piecewise	Piecewise function FEX
VB	$VB$	GainLimiter	VB with limiter
VG	$VG$	GainLimiter	Feedback gain with HL
HG	$HG$	HVGate	HVGate for under excitation
LL	$LL$	LeadLag	Regulator
LAW1	$LAW1$	LagAntiWindup	Lag AW on VR
LAW2	$LAW2$	LagAntiWindup	Lag AW on VM

## 5.8.6 SEXS

Group *Exciter*

Simplified Excitation System Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
syn		Synchronous generator idx			IdxParam	mandatory
TATB	$T_A/T_B$	Time constant TA/TB	0.400		NumParam	
TB	$T_B$	Time constant TB in LL	5		NumParam	
K	$K$	Gain	20		NumParam	non_zero
TE	$T_E$	AW Lag time constant	1		NumParam	
EMIN	$E_{MIN}$	lower limit	-99		NumParam	
EMAX	$E_{MAX}$	upper limit	99		NumParam	
Sn	$S_m$	Rated power from generator	0	<i>MVA</i>	ExtParam	
Vn	$V_m$	Rated voltage from generator	0	<i>kV</i>	ExtParam	
bus	$bus$	Bus idx of the generators	0		ExtParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
LL_x	$x'_{LL}$	$V_i$	State in lead-lag		v_str
LAW_y	$y_{LAW}$	$Ky_{LL}$	State in lag TF		v_str
omega	$\omega$		Generator speed		
vout	$v_{out}$	$v_{f0}$	Exciter final output voltage		v_str
vref	$V_{ref}$	$V_{ref0}$	Reference voltage input	<i>p.u.</i>	v_str
vi	$V_i$	$-V + V_{ref0}$	Total input voltages	<i>p.u.</i>	v_str
LL_y	$y_{LL}$	$V_i$	Output of lead-lag		v_str
vf	$v_f$		Excitation field voltage to generator		
XadIfd	$X_{ad}I_{fd}$		Armature excitation current		
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
LL_x	$x'_{LL}$	State	$V_i - x'_{LL}$	$T_B$
LAW_y	$y_{LAW}$	State	$Ky_{LL} - y_{LAW}$	$T_E$
omega	$\omega$	ExtState	0	

## Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
vout	$v_{out}$	Algeb	$-v_{out} + y_{LAW}$
vref	$V_{ref}$	Algeb	$V_{ref0} - V_{ref}$
vi	$V_i$	Algeb	$-V - V_i + V_{ref}$
LL_y	$y_{LL}$	Algeb	$LL_{LT1z1}LL_{LT2z1}(-x'_{LL} + y_{LL}) + TA(V_i - x'_{LL}) + T_Bx'_{LL} - T_By_{LL}$
vf	$v_f$	ExtAl- geb	$u(-v_{f0} + v_{out})$
Xad- Ifd	$X_{ad}I_{fd}$	ExtAl- geb	0
a	$\theta$	ExtAl- geb	0
v	$V$	ExtAl- geb	0

## Services

Name	Symbol	Equation	Type
TA	$TA$	$T_A/T_B T_B$	ConstService
vref0	$V_{ref0}$	$V + \frac{v_{f0}}{K}$	ConstService

## Discrete

Name	Symbol	Type	Info
LL_LT1	$LT_{LL}$	LessThan	
LL_LT2	$LT_{LL}$	LessThan	
LAW_lim	$lim_{LAW}$	AntiWindup	Limiter in Lag

Blocks

Name	Symbol	Type	Info
LL	$LL$	LeadLag	
LAW	$LAW$	LagAntiWindup	

## 5.9 Experimental

Experimental group

Common Parameters: u, name

Available models: *PI2*, *TestDB1*, *TestPI*, *TestLagAWFreeze*

### 5.9.1 PI2

Group *Experimental*

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
Kp					NumParam	
Ki					NumParam	
Wmax					NumParam	
Wmin					NumParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
uin	$u_{in}$	0			v_str
x	$x$	0.05			v_str
y	$y$	0.05			v_str
w	$w$	0.05			v_str

Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
uin	$u_{in}$	State	$\begin{cases} 0 & \text{for } t_{dae} \leq 0 \\ 1 & \text{for } t_{dae} \leq 2 \\ -1 & \text{for } t_{dae} < 6 \\ 1 & \text{otherwise} \end{cases}$	
x	$x$	State	$K i u_{in} z_i^{HL}$	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
y	$y$	Algeb	$K p u_{in} + x - y$
w	$w$	Algeb	$W max z_u^{HL} + W min z_l^{HL} - w + y z_i^{HL}$

## Discrete

Name	Symbol	Type	Info
HL	$HL$	HardLimiter	

## 5.9.2 TestDB1

Group *Experimental*

Test model for *DeadBand1*.

## Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	

## Variables

Name	Sym- bol	Initial Value	Description	Unit	Proper- ties
uin	$u_{in}$	-10			v_str
DB_y	$y_{DB}$	$DB_{dbzl}(u_{in} + 5)$ $DB_{dbzu}(u_{in} - 5)$	+ Deadband type 1 out- put		v_str

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
uin	$u_{in}$	Algeb	$t_{dae} - u_{in} - 10$
DB_y	$y_{DB}$	Algeb	$DB_{dbzl}(u_{in} + 5) + DB_{dbzu}(u_{in} - 5) - y_{DB}$

Discrete

Name	Symbol	Type	Info
DB_db	$db_{DB}$	DeadBand	

Blocks

Name	Symbol	Type	Info
DB	$DB$	DeadBand1	

### 5.9.3 TestPI

Group *Experimental*

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
Text		Extended event time	1	<i>s</i>	NumParam	

Variables

Name	Sym- bol	Initial Value	Description	Unit	Prop- erties
PI_xi	$xi_{PI}$	0.0	Integrator output		v_str
PIF_xi	$xi_{PIF}$	0	Integrator output		v_str
PIAW_xi	$xi_{PIAW}$	0.0	Integrator output		v_str
PI-AWF_xi	$xi_{PIAWF}$	0	Integrator output		v_str
uin	$u_{in}$	0			v_str
zf	$z_f$	0			v_str
PI_y	$y_{PI}$	$u_{in}$	PI output		v_str
PIF_y	$y_{PIF}$	$0.5u_{in}$	PI output		v_str
PIAW_ys	$ys_{PIAW}$	$0.5u_{in}$	PI summation before limit		v_str
PIAW_y	$y_{PIAW}$	$PIAW_{limzi}ys_{PIAW} - 0.5PIAW_{limzl} + 0.5PIAW_{limzu}$	PI output		v_str
PI-AWF_ys	$ys_{PIAWF}$	$0.5u_{in}$	PI summation before limit		v_str
PI-AWF_y	$y_{PIAWF}$	$PIAWF_{limzi}ys_{PIAWF} - 0.5PIAWF_{limzl} + 0.5PIAWF_{limzu}$	PI output		v_str
ze	$ze$	$ExtEvent$			v_str



## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
PI_xi	$xi_{PI}$	State	$0.1u_{in}$	
PIF_xi	$xi_{PIF}$	State	$u_{in}(0.5 - 0.5z_f)$	
PIAW_xi	$xi_{PIAW}$	State	$0.5u_{in} + 1.0y_{PIAW} - 1.0y_{SPIAW}$	
PIAWF_xi	$xi_{PIAWF}$	State	$(0.5 - 0.5z_f)(u_{in} + 2y_{PIAWF} - 2y_{SPIAWF})$	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
uin	$u_{in}$	Algeb	$-u_{in} + \sin(t_{dae})$
zf	$z_f$	Algeb	$-z_f + \begin{cases} 0 & \text{for } t_{dae} \leq 2 \\ 1 & \text{for } t_{dae} \leq 6 \\ 0 & \text{for } t_{dae} \leq 12 \\ 1 & \text{for } t_{dae} \leq 15 \\ 0 & \text{otherwise} \end{cases}$
PI_y	$y_{PI}$	Algeb	$u_{in} + xi_{PI} - y_{PI}$
PIF_y	$y_{PIF}$	Algeb	$(1 - z_f)(0.5u_{in} + xi_{PIF} - y_{PIF})$
PIAW_y	$y_{PIAW}$	Algeb	$0.5u_{in} + xi_{PIAW} - y_{SPIAW}$
PIAWF_y	$y_{PIAWF}$	Algeb	$PIAW_{limzi}y_{SPIAW} - 0.5PIAW_{limzl} + 0.5PIAW_{limzu} - y_{PIAW}$
PI-AWF_y	$y_{SPIAWF}$	Algeb	$(1 - z_f)(0.5u_{in} + xi_{PIAWF} - y_{SPIAWF})$
PI-AWF_y	$y_{PIAWF}$	Algeb	$(1 - z_f)(PIAWF_{limzi}y_{SPIAWF} - 0.5PIAWF_{limzl} + 0.5PIAWF_{limzu} - y_{PIAWF})$
ze	$ze$	Algeb	$ExtEvent - ze$

## Services

Name	Symbol	Equation	Type
PIF_flag	$z_{PIF}^{flag}$	0	EventFlag
PIAWF_flag	$z_{PIAWF}^{flag}$	0	EventFlag
ExtEvent	$ExtEvent$	0	ExtendedEvent

## Discrete

Name	Symbol	Type	Info
PIAW_lim	$\lim_{PIAW}$	HardLimiter	
PIAWF_lim	$\lim_{PIAWF}$	HardLimiter	

Blocks

Name	Symbol	Type	Info
PI	$PI$	PIController	
PIF	$PIF$	PIFreeze	
PIAW	$PIAW$	PITrackAW	
PIAWF	$PIAWF$	PITrackAWFreeze	

### 5.9.4 TestLagAWFreeze

Group *Experimental*

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
LGF_y	$y_{LGF}$	$1.0u_{in}$	State in lag transfer function		v_str
LGAWF_y	$y_{LGAWF}$	$1.0u_{in}$	State in lag TF		v_str
uin	$u_{in}$	0			v_str
zf	$z_f$	0			v_str

Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
LGF_y	$y_{LGF}$	State	$(1 - z_f)(1.0u_{in} - y_{LGF})$	1.0
LGAWF_y	$y_{LGAWF}$	State	$(1 - z_f)(1.0u_{in} - y_{LGAWF})$	1.0

Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
uin	$u_{in}$	Algeb	$-u_{in} + \sin(t_{dae})$
zf	$z_f$	Algeb	$-z_f + \begin{cases} 0 & \text{for } t_{dae} \leq 2 \\ 1 & \text{for } t_{dae} \leq 6 \\ 0 & \text{otherwise} \end{cases}$

## Services

Name	Symbol	Equation	Type
LGF_flag	$z_{LGF}^{flag}$	0	EventFlag
LGAWF_flag	$z_{LGAWF}^{flag}$	0	EventFlag

## Discrete

Name	Symbol	Type	Info
LGAWF_lim	$lim_{LGAWF}$	AntiWindup	Limiter in Lag

## Blocks

Name	Symbol	Type	Info
LGF	$LGF$	LagFreeze	
LGAWF	$LGAWF$	LagAWFreeze	

## 5.10 FreqMeasurement

Frequency measurements.

Common Parameters: u, name

Common Variables: f

Available models: *BusFreq*, *BusROCOF*

### 5.10.1 BusFreq

Group *FreqMeasurement*

Bus frequency measurement.

Bus frequency output variable is  $f$ .

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
bus		bus idx			IdxParam	mandatory
Tf	$T_f$	input digital filter time const	0.020	<i>sec</i>	NumParam	
Tw	$T_w$	washout time const	0.020	<i>sec</i>	NumParam	
fn	$f_n$	nominal frequency	60	<i>Hz</i>	NumParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
L_y	$y_L$	$\theta - \theta_0$	State in lag transfer function		v_str
WO_x	$x'_{WO}$	$y_L$	State in washout filter		v_str
WO_y	$y_{WO}$	0	Output of washout filter		v_str
f	$f$	1	frequency output	<i>p.u. (Hz)</i>	v_str
a	$\theta$				
v	$V$				

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
L_y	$y_L$	State	$\theta - \theta_0 - y_L$	$T_f$
WO_x	$x'_{WO}$	State	$-x'_{WO} + y_L$	$T_w$

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
WO_y	$y_{WO}$	Algeb	$1/\omega_n (-x'_{WO} + y_L) - T_w y_{WO}$
f	$f$	Algeb	$-f + y_{WO} + 1$
a	$\theta$	ExtAlgeb	0
v	$V$	ExtAlgeb	0

## Services

Name	Symbol	Equation	Type
iwn	$1/\omega_n$	$\frac{u}{2\pi f_n}$	ConstService

## Blocks

Name	Symbol	Type	Info
L	$L$	Lag	digital filter
WO	$WO$	Washout	angle washout

## 5.10.2 BusROCOF

Group *FreqMeasurement*

Bus frequency and ROCOF measurement.

The ROCOF output variable is  $wf\_y$ .

## Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
bus		bus idx			IdxParam	mandatory
Tf	$T_f$	input digital filter time const	0.020	<i>sec</i>	NumParam	
Tw	$T_w$	washout time const	0.020	<i>sec</i>	NumParam	
fn	$f_n$	nominal frequency	60	<i>Hz</i>	NumParam	
Tr	$T_r$	frequency washout time constant	0.100		NumParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
L_y	$y_L$	$\theta - \theta_0$	State in lag transfer function		v_str
WO_x	$x'_{WO}$	$y_L$	State in washout filter		v_str
Wf_x	$x'_{Wf}$	$f$	State in washout filter		v_str
WO_y	$y_{WO}$	0	Output of washout filter		v_str
f	$f$	1	frequency output	<i>p.u. (Hz)</i>	v_str
Wf_y	$y_{Wf}$	0	Output of washout filter		v_str
a	$\theta$				
v	$V$				

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
L_y	$y_L$	State	$\theta - \theta_0 - y_L$	$T_f$
WO_x	$x'_{WO}$	State	$-x'_{WO} + y_L$	$T_w$
Wf_x	$x'_{Wf}$	State	$f - x'_{Wf}$	$T_r$

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
WO_y	$y_{WO}$	Algeb	$1/\omega_n (-x'_{WO} + y_L) - T_w y_{WO}$
f	$f$	Algeb	$-f + y_{WO} + 1$
Wf_y	$y_{Wf}$	Algeb	$-T_r y_{Wf} + f - x'_{Wf}$
a	$\theta$	ExtAlgeb	0
v	$V$	ExtAlgeb	0

## Services

Name	Symbol	Equation	Type
iwn	$1/\omega_n$	$\frac{u}{2\pi f_n}$	ConstService

## Blocks

Name	Symbol	Type	Info
L	$L$	Lag	digital filter
WO	$WO$	Washout	angle washout
Wf	$Wf$	Washout	frequency washout yielding ROCOF

## 5.11 Motor

Induction Motor group

Common Parameters: u, name

Available models: *Motor3*, *Motor5*

### 5.11.1 Motor3

Group *Motor*

Third-order induction motor model.

See "Power System Modelling and Scripting" by F. Milano.

To simulate motor startup, set the motor status u to 0 and use a `Toggler` to control the model.

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
bus		interface bus id			IdxParam	mandatory
Sn	$S_n$	Power rating	100		NumParam	
Vn	$V_n$	AC voltage rating	110		NumParam	
fn	$f$	rated frequency	60		NumParam	
rs	$r_s$	rotor resistance	0.010		NumParam	non_zero,z
xs	$x_s$	rotor reactance	0.150		NumParam	non_zero,z
rr1	$r_{R1}$	1st cage rotor resistance	0.050		NumParam	non_zero,z
xr1	$x_{R1}$	1st cage rotor reactance	0.150		NumParam	non_zero,z
rr2	$r_{R2}$	2st cage rotor resistance	0.001		NumParam	non_zero,z
xr2	$x_{R2}$	2st cage rotor reactance	0.040		NumParam	non_zero,z
xm	$x_m$	magnetization reactance	5		NumParam	non_zero,z
Hm	$H_m$	Inertia constant	3	<i>kWs/KVA</i>	NumParam	power
c1	$c_1$	1st coeff. of Tm(w)	0.100		NumParam	
c2	$c_2$	2nd coeff. of Tm(w)	0.020		NumParam	
c3	$c_3$	3rd coeff. of Tm(w)	0.020		NumParam	
zb	$z_b$	Allow working as brake	1		NumParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
slip	$\sigma$	$1.0u$			v_str
e1d	$e'_d$	$0.05u$	real part of 1st cage voltage		v_str
e1q	$e'_q$	$0.9u$	imaginary part of 1st cage voltage		v_str
vd	$V_d$		d-axis voltage		
vq	$V_q$		q-axis voltage		
p	$P$	$u(I_d V_d + I_q V_q)$			v_str
q	$Q$	$u(I_d V_q - I_q V_d)$			v_str
Id	$I_d$	1			v_str
Iq	$I_q$				
te	$\tau_e$	$u(I_d e'_d + I_q e'_q)$			v_str
tm	$\tau_m$	$u(\alpha + \beta \sigma + \sigma^2 c_2)$			v_str
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

### Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
slip	$\sigma$	State	$u(-\tau_e + \tau_m)$	$M$
e1d	$e'_d$	State	$u\left(\omega_b \sigma e'_q - \frac{I_q(-x' + x_0) + e'_d}{T'_0}\right)$	
e1q	$e'_q$	State	$u\left(-\omega_b \sigma e'_d - \frac{-I_d(-x' + x_0) + e'_q}{T'_0}\right)$	

### Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
vd	$V_d$	Algeb	$-Vu \sin(\theta) - V_d$
vq	$V_q$	Algeb	$Vu \cos(\theta) - V_q$
p	$P$	Algeb	$-P + u(I_d V_d + I_q V_q)$
q	$Q$	Algeb	$-Q + u(I_d V_q - I_q V_d)$
Id	$I_d$	Algeb	$u(-I_d r_s + I_q x' + V_d - e'_d)$
Iq	$I_q$	Algeb	$u(-I_d x' - I_q r_s + V_q - e'_q)$
te	$\tau_e$	Algeb	$-\tau_e + u(I_d e'_d + I_q e'_q)$
tm	$\tau_m$	Algeb	$-\tau_m + u(\alpha + \beta \sigma + \sigma^2 c_2)$
a	$\theta$	ExtAlgeb	$P$
v	$V$	ExtAlgeb	$Q$

### Services

Name	Symbol	Equation	Type
wb	$\omega_b$	$2\pi f$	ConstService
x0	$x_0$	$x_m + x_s$	ConstService
x1	$x'$	$\frac{x_m x_{R1}}{x_m + x_{R1}} + x_s$	ConstService
T10	$T'_0$	$\frac{x_m + x_{R1}}{\omega_b r_{R1}}$	ConstService
M	$M$	$2H_m$	ConstService
aa	$\alpha$	$c_1 + c_2 + c_3$	ConstService
bb	$\beta$	$-c_2 - 2c_3$	ConstService

### 5.11.2 Motor5

#### Group *Motor*

Fifth-order induction motor model.

See "Power System Modelling and Scripting" by F. Milano.

To simulate motor startup, set the motor status `u` to 0 and use a `Toggler` to control the model.

#### Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
bus		interface bus id			IdxParam	mandatory
Sn	$S_n$	Power rating	100		NumParam	
Vn	$V_n$	AC voltage rating	110		NumParam	
fn	$f$	rated frequency	60		NumParam	
rs	$r_s$	rotor resistance	0.010		NumParam	non_zero,z
xs	$x_s$	rotor reactance	0.150		NumParam	non_zero,z
rr1	$r_{R1}$	1st cage rotor resistance	0.050		NumParam	non_zero,z
xr1	$x_{R1}$	1st cage rotor reactance	0.150		NumParam	non_zero,z
rr2	$r_{R2}$	2st cage rotor resistance	0.001		NumParam	non_zero,z
xr2	$x_{R2}$	2st cage rotor reactance	0.040		NumParam	non_zero,z
xm	$x_m$	magnetization reactance	5		NumParam	non_zero,z
Hm	$H_m$	Inertia constant	3	<i>kWs/KVA</i>	NumParam	power
c1	$c_1$	1st coeff. of Tm(w)	0.100		NumParam	
c2	$c_2$	2nd coeff. of Tm(w)	0.020		NumParam	
c3	$c_3$	3rd coeff. of Tm(w)	0.020		NumParam	
zb	$z_b$	Allow working as brake	1		NumParam	

#### Variables



Name	Symbol	Initial Value	Description	Unit	Properties
slip	$\sigma$	$1.0u$			v_str
e1d	$e'_d$	$0.05u$	real part of 1st cage voltage		v_str
e1q	$e'_q$	$0.9u$	imaginary part of 1st cage voltage		v_str
e2d	$e''_d$	$0.05u$	real part of 2nd cage voltage		v_str
e2q	$e''_q$	$0.9u$	imag part of 2nd cage voltage		v_str
vd	$V_d$		d-axis voltage		
vq	$V_q$		q-axis voltage		
p	$P$	$u(I_d V_d + I_q V_q)$			v_str
q	$Q$	$u(I_d V_q - I_q V_d)$			v_str
Id	$I_d$	$0.9u$			v_str
Iq	$I_q$	$0.1u$			v_str
te	$\tau_e$	$u(I_d e''_d + I_q e''_q)$			v_str
tm	$\tau_m$	$u(\alpha + \beta\sigma + \sigma^2 c_2)$			v_str
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

## Differential Equations

Name	Sym- bol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
slip	$\sigma$	State	$u(-\tau_e + \tau_m)$	$M$
e1d	$e'_d$	State	$u\left(\omega_b \sigma e'_q - \frac{I_q(-x' + x_0) + e'_d}{T'_0}\right)$	
e1q	$e'_q$	State	$u\left(-\omega_b \sigma e'_d - \frac{-I_d(-x' + x_0) + e'_q}{T'_0}\right)$	
e2d	$e''_d$	State	$u\left(\omega_b \sigma e'_q - \omega_b \sigma (-e''_q + e'_q) - \frac{I_q(-x' + x_0) + e'_d}{T'_0} + \frac{-I_q(x' - x'') - e''_d + e'_d}{T''_0}\right)$	
e2q	$e''_q$	State	$u\left(-\omega_b \sigma e'_d + \omega_b \sigma (-e''_d + e'_d) - \frac{-I_d(-x' + x_0) + e'_q}{T'_0} + \frac{I_d(x' - x'') - e''_q + e'_q}{T''_0}\right)$	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
vd	$V_d$	Algeb	$-Vu \sin(\theta) - V_d$
vq	$V_q$	Algeb	$Vu \cos(\theta) - V_q$
p	$P$	Algeb	$-P + u(I_d V_d + I_q V_q)$
q	$Q$	Algeb	$-Q + u(I_d V_q - I_q V_d)$
Id	$I_d$	Algeb	$u(-I_d r_s + I_q x'' + V_d - e''_d)$
Iq	$I_q$	Algeb	$u(-I_d x'' - I_q r_s + V_q - e''_q)$
te	$\tau_e$	Algeb	$-\tau_e + u(I_d e''_d + I_q e''_q)$
tm	$\tau_m$	Algeb	$-\tau_m + u(\alpha + \beta\sigma + \sigma^2 c_2)$
a	$\theta$	ExtAlgeb	$P$
v	$V$	ExtAlgeb	$Q$

## Services

Name	Symbol	Equation	Type
wb	$\omega_b$	$2\pi f$	ConstService
x0	$x_0$	$x_m + x_s$	ConstService
x1	$x'$	$\frac{x_m x_{R1}}{x_m + x_{R1}} + x_s$	ConstService
T10	$T'_0$	$\frac{x_m + x_{R1}}{\omega_b r_{R1}}$	ConstService
M	$M$	$2H_m$	ConstService
aa	$\alpha$	$c_1 + c_2 + c_3$	ConstService
bb	$\beta$	$-c_2 - 2c_3$	ConstService
x2	$x''$	$\frac{\frac{x_m x_{R1} x_{R2}}{x_m x_{R1} + x_m x_{R2} + x_{R1} x_{R2}}}{\frac{x_m x_{R1}}{x_m + x_{R1}} + x_{R2}} + x_s$	ConstService
T20	$T''_0$	$\frac{\frac{x_m x_{R1}}{x_m + x_{R1}} + x_{R2}}{\omega_b r_{R2}}$	ConstService

## 5.12 PSS

Power system stabilizer group.

Common Parameters: u, name

Common Variables: vsout

Available models: *IEEEST*, *ST2CUT*

### 5.12.1 IEEEST

Group *PSS*

IEEEST stabilizer model. Automatically adds frequency measurement devices if not provided.

Input signals (MODE):

1 - Rotor speed deviation (p.u.), 2 - Bus frequency deviation (\*) (p.u.), 3 - Generator P electrical in Gen MVABase (p.u.), 4 - Generator accelerating power (p.u.), 5 - Bus voltage (p.u.), 6 - Derivative of p.u. bus voltage.

(\*) Due to the frequency measurement implementation difference, mode 2 is likely to yield different results across software.

Blocks are named *F1*, *F2*, *LL1*, *LL2* and *WO* in sequence. Two limiters are named *VLIM* and *OLIM* in sequence.

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
avr		Exciter idx			IdxParam	mandatory
MODE		Input signal			NumParam	mandatory
busr		Optional remote bus idx			IdxParam	
busf		BusFreq idx for mode 2			IdxParam	
A1	$A_1$	filter time const. (pole)	1		NumParam	
A2	$A_2$	filter time const. (pole)	1		NumParam	
A3	$A_3$	filter time const. (pole)	1		NumParam	
A4	$A_4$	filter time const. (pole)	1		NumParam	
A5	$A_5$	filter time const. (zero)	1		NumParam	
A6	$A_6$	filter time const. (zero)	1		NumParam	
T1	$T_1$	first leadlag time const. (zero)	1		NumParam	
T2	$T_2$	first leadlag time const. (pole)	1		NumParam	
T3	$T_3$	second leadlag time const. (pole)	1		NumParam	
T4	$T_4$	second leadlag time const. (pole)	1		NumParam	
T5	$T_5$	washout time const. (zero)	1		NumParam	
T6	$T_6$	washout time const. (pole)	1		NumParam	
KS	$K_S$	Gain before washout	1		NumParam	
LSMAX	$L_{SMAX}$	Max. output limit	0.300		NumParam	
LSMIN	$L_{SMIN}$	Min. output limit	-0.300		NumParam	
VCU	$V_{CU}$	Upper enabling bus voltage	999	<i>p.u.</i>	NumParam	
VCL	$V_{CL}$	Upper enabling bus voltage	-999	<i>p.u.</i>	NumParam	
syn		Retrieved generator idx	0		ExtParam	
bus		Retrieved bus idx			ExtParam	
Sn	$S_n$	Generator power base	0		ExtParam	

## Variables

Name	Sym- bol	Initial Value	Description	Unit	Proper- ties
F1_x	$x'_{F1}$	0	State in 2nd order LPF		v_str
F1_y	$y_{F1}$	$S_{ig}$	Output of 2nd order LPF		v_str
F2_x1	$x'_{F2}$	0	State #1 in 2nd order lead-lag		v_str
F2_x2	$x''_{F2}$	$y_{F1}$	State #2 in 2nd order lead-lag		v_str
LL1_x	$x'_{LL1}$	$y_{F2}$	State in lead-lag		v_str
LL2_x	$x'_{LL2}$	$y_{LL1}$	State in lead-lag		v_str
WO_x	$x'_{WO}$	$y_{Vks}$	State in washout filter		v_str
omega	$\omega$		Generator speed	<i>p.u.</i>	
vsout	$v_{sout}$		PSS output voltage to exciter		
sig	$S_{ig}$	$V s_5^{SW} + s_1^{SW}(\omega - 1) + s_4^{SW}(\tau_m - \tau_{m0}) + \frac{\tau_{m0} s_3^{SW}}{(Sb/Sn)}$	Input signal		v_str
F2_y	$y_{F2}$	$y_{F1}$	Output of 2nd order lead-lag		v_str
LL1_y	$y_{LL1}$	$y_{F2}$	Output of lead-lag		v_str
LL2_y	$y_{LL2}$	$y_{LL1}$	Output of lead-lag		v_str
Vks_y	$y_{Vks}$	$K_{SYLL2}$	Gain output		v_str
WO_y	$y_{WO}$	$WO_{LTz1} x'_{WO}$	Output of washout filter		v_str
Vss	$V_{ss}$		Voltage output before output limiter		
tm	$\tau_m$		Generator mechanical input		
te	$\tau_e$		Generator electrical output		
v	$V$		Bus (or busr, if given) terminal voltage		
f	$f$		Bus frequency		
vi	$v_i$		Exciter input voltage		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
F1_x	$x'_{F1}$	State	$-A_1 x'_{F1} + S_{ig} - y_{F1}$	$A_2$
F1_y	$y_{F1}$	State	$x'_{F1}$	
F2_x1	$x'_{F2}$	State	$-A_3 x'_{F2} - x''_{F2} + y_{F1}$	$A_4$
F2_x2	$x''_{F2}$	State	$x'_{F2}$	
LL1_x	$x'_{LL1}$	State	$-x'_{LL1} + y_{F2}$	$T_2$
LL2_x	$x'_{LL2}$	State	$-x'_{LL2} + y_{LL1}$	$T_4$
WO_x	$x'_{WO}$	State	$-x'_{WO} + y_{Vks}$	$T_6$
omega	$\omega$	ExtState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
vsout	$v_{sout}$	Algeb	$V_{ss}z_i^{OLIM} - v_{sout}$
sig	$S_{ig}$	Algeb	$-S_{ig} + V s_5^{SW} + \frac{V^{dV} s_6^{SW}}{dt} + s_1^{SW} (\omega - 1) + s_2^{SW} (f - 1) + s_4^{SW} (\tau_m - \tau_{m0}) + \frac{\tau_e s_3^{SW}}{(Sb/Sn)}$
F2_y	$y_{F2}$	Algeb	$A_4 A_5 x'_{F2} + A_4 x''_{F2} - A_4 y_{F2} + A_6 (-A_3 x'_{F2} - x''_{F2} + y_{F1}) + F_{2LT1z1} F_{2LT2z1} F_{2LT3z1} F_{2LT4z1} (-x''_{F2} + y_{F2})$
LL1_y	$y_{LL1}$	Algeb	$LL_{1LT1z1} LL_{1LT2z1} (-x'_{LL1} + y_{LL1}) + T_1 (-x'_{LL1} + y_{F2}) + T_2 x'_{LL1} - T_2 y_{LL1}$
LL2_y	$y_{LL2}$	Algeb	$LL_{2LT1z1} LL_{2LT2z1} (-x'_{LL2} + y_{LL2}) + T_3 (-x'_{LL2} + y_{LL1}) + T_4 x'_{LL2} - T_4 y_{LL2}$
Vks_y	$y_{Vks}$	Algeb	$K_{SYLL2} - y_{Vks}$
WO_y	$y_{WO}$	Algeb	$T_5 WO_{LTz0} (-x'_{WO} + y_{Vks}) + T_6 WO_{LTz1} x'_{WO} - T_6 y_{WO}$
Vss	$V_{ss}$	Algeb	$L_{SMAX} z_u^{VLIM} + L_{SMIN} z_l^{VLIM} - V_{ss} + y_{WO} z_i^{VLIM}$
tm	$\tau_m$	ExtAlgeb	0
te	$\tau_e$	ExtAlgeb	0
v	$V$	ExtAlgeb	0
f	$f$	ExtAlgeb	0
vi	$v_i$	ExtAlgeb	$uv_{sout}$

## Discrete

Name	Symbol	Type	Info
dv	$dV/dt$	Derivative	Finite difference of bus voltage
SW	$SW$	Switcher	
F2_LT1	$LT_{F2}$	LessThan	
F2_LT2	$LT_{F2}$	LessThan	
F2_LT3	$LT_{F2}$	LessThan	
F2_LT4	$LT_{F2}$	LessThan	
LL1_LT1	$LT_{LL1}$	LessThan	
LL1_LT2	$LT_{LL1}$	LessThan	
LL2_LT1	$LT_{LL2}$	LessThan	
LL2_LT2	$LT_{LL2}$	LessThan	
WO_LT	$LT_{WO}$	LessThan	
VLIM	$VLIM$	Limiter	Vss limiter
OLIM	$OLIM$	Limiter	output limiter

## Blocks

Name	Symbol	Type	Info
F1	$F1$	Lag2ndOrd	
F2	$F2$	LeadLag2ndOrd	
LL1	$LL1$	LeadLag	
LL2	$LL2$	LeadLag	
Vks	$V_{ks}$	Gain	
WO	$WO$	WashoutOrLag	

## Config Fields in [IEEEST]

Option	Symbol	Value	Info	Accepted values
freq_model		BusFreq	default freq. measurement model	('BusFreq',)

### 5.12.2 ST2CUT

Group *PSS*

ST2CUT stabilizer model. Automatically adds frequency measurement devices if not provided.

Input signals (MODE and MODE2):

0 - Disable input signal 1 (s1) - Rotor speed deviation (p.u.), 2 (s2) - Bus frequency deviation (\*) (p.u.), 3 (s3) - Generator P electrical in Gen MVABase (p.u.), 4 (s4) - Generator accelerating power (p.u.), 5 (s5) - Bus voltage (p.u.), 6 (s6) - Derivative of p.u. bus voltage.

(\*) Due to the frequency measurement implementation difference, mode 2 is likely to yield different results across software.

Blocks are named *LL1*, *LL2*, *LL3*, *LL4* in sequence. Two limiters are named *VSS\_lim* and *OLIM* in sequence.

## Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
avr		Exciter idx			IdxParam	mandatory
MODE		Input signal 1			NumParam	mandatory
busr		Remote bus 1			NumParam	
busf		BusFreq idx for signal 1 mode 2			IdxParam	
MODE2		Input signal 2			NumParam	
busr2		Remote bus 2			NumParam	
busf2		BusFreq idx for signal 2 mode 2			IdxParam	
K1	$K_1$	Transducer 1 gain	1		NumParam	
K2	$K_2$	Transducer 2 gain	1		NumParam	
T1	$T_1$	Transducer 1 time const.	1		NumParam	
T2	$T_2$	Transducer 2 time const.	1		NumParam	
T3	$T_3$	Washout int. time const.	1		NumParam	
T4	$T_4$	Washout delay time const.	0.200		NumParam	
T5	$T_5$	Leadlag 1 time const. (1)	1		NumParam	
T6	$T_6$	Leadlag 1 time const. (2)	0.500		NumParam	
T7	$T_7$	Leadlag 2 time const. (1)	1		NumParam	
T8	$T_8$	Leadlag 2 time const. (2)	1		NumParam	
T9	$T_9$	Leadlag 3 time const. (1)	1		NumParam	
T10	$T_{10}$	Leadlag 3 time const. (2)	0.200		NumParam	
LSMAX	$L_{SMAX}$	Max. output limit	0.300		NumParam	
LSMIN	$L_{SMIN}$	Min. output limit	-0.300		NumParam	
VCU	$V_{CU}$	Upper enabling bus voltage	999	<i>p.u.</i>	NumParam	
VCL	$V_{CL}$	Upper enabling bus voltage	-999	<i>p.u.</i>	NumParam	
syn		Retrieved generator idx	0		ExtParam	
bus		Retrieved bus idx			ExtParam	
Sn	$S_n$	Generator power base	0		ExtParam	

Variables

Name	Sym- bol	Initial Value	Description	Unit	Prop- erties
L1_y	$y_{L1}$	$K_1 S_{ig}$	State in lag transfer func- tion		v_str
L2_y	$y_{L2}$	$K_2 S_{ig2}$	State in lag transfer func- tion		v_str
WO_x	$x'_{WO}$	$I_N$	State in washout filter		v_str
LL1_x	$x'_{LL1}$	$y_{WO}$	State in lead-lag		v_str
LL2_x	$x'_{LL2}$	$y_{LL1}$	State in lead-lag		v_str
LL3_x	$x'_{LL3}$	$y_{LL2}$	State in lead-lag		v_str
omega	$\omega$		Generator speed	<i>p.u.</i>	
vsout	$v_{sout}$		PSS output voltage to ex- citer		
sig	$S_{ig}$	$V s_5^{SW} + s_1^{SW} (\omega - 1) + s_4^{SW} (\tau_m - \tau_{m0}) + \frac{\tau_{m0} s_3^{SW}}{(Sb/Sn)}$	Input signal		v_str
sig2	$S_{ig2}$	$V s_5^{SW_2} + s_1^{SW_2} (\omega - 1) + s_4^{SW_2} (\tau_m - \tau_{m0}) + \frac{\tau_{m0} s_3^{SW_2}}{(Sb/Sn)}$	Input signal 2		v_str
IN	$I_N$	$y_{L1} + y_{L2}$	Sum of inputs		v_str
WO_y	$y_{WO}$	$WO_{LTz1} x'_{WO}$	Output of washout filter		v_str
LL1_y	$y_{LL1}$	$y_{WO}$	Output of lead-lag		v_str
LL2_y	$y_{LL2}$	$y_{LL1}$	Output of lead-lag		v_str
LL3_y	$y_{LL3}$	$y_{LL2}$	Output of lead-lag		v_str
VSS_x	$x_{VSS}$	$y_{LL3}$	Gain output before limiter		v_str
VSS_y	$y_{VSS}$	$L_{SMAX} VSS_{limzu} + L_{SMIN} VSS_{limzl} + VSS_{limzi} x_{VSS}$	Gain output after limiter		v_str
tm	$\tau_m$		Generator mechanical input		
te	$\tau_e$		Generator electrical output		
v	$V$		Bus (or busr, if given) ter- minal voltage		
f	$f$		Bus frequency		
vi	$v_i$		Exciter input voltage		
v2	$V$		Bus (or busr2, if given) ter- minal voltage		
f2	$f_2$		Bus frequency 2		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
L1_y	$y_{L1}$	State	$K_1 S_{ig} - y_{L1}$	$T_1$
L2_y	$y_{L2}$	State	$K_2 S_{ig2} - y_{L2}$	$T_2$
WO_x	$x'_{WO}$	State	$I_N - x'_{WO}$	$T_4$
LL1_x	$x'_{LL1}$	State	$-x'_{LL1} + y_{WO}$	$T_6$
LL2_x	$x'_{LL2}$	State	$-x'_{LL2} + y_{LL1}$	$T_8$
LL3_x	$x'_{LL3}$	State	$-x'_{LL3} + y_{LL2}$	$T_{10}$
omega	$\omega$	ExtState	0	



## Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
vsout	$v_{sout}$	Algeb	$-v_{sout} + y_{VSS} z_i^{OLIM}$
sig	$S_{ig}$	Algeb	$-S_{ig} + V s_5^{SW} + V^{dv} s_6^{SW} + s_1^{SW} (\omega - 1) + s_2^{SW} (f - 1) + s_4^{SW} (\tau_m - \tau_{m0}) + \frac{\tau_e s_3^{SW}}{(Sb/Sn)}$
sig2	$S_{ig2}$	Algeb	$-S_{ig2} + V s_5^{SW_2} + V^{dv_2} s_6^{SW_2} + s_1^{SW_2} (\omega - 1) + s_2^{SW_2} (f_2 - 1) + s_4^{SW_2} (\tau_m - \tau_{m0}) + \frac{\tau_e s_3^{SW_2}}{(Sb/Sn)}$
IN	$I_N$	Algeb	$-I_N + y_{L1} + y_{L2}$
WO_y	$y_{WO}$	Algeb	$T_3 WO_{LTz0} (I_N - x'_{WO}) + T_4 WO_{LTz1} x'_{WO} - T_4 y_{WO}$
LL1_y	$y_{LL1}$	Algeb	$LL_{1LT1z1} LL_{1LT2z1} (-x'_{LL1} + y_{LL1}) + T_5 (-x'_{LL1} + y_{WO}) + T_6 x'_{LL1} - T_6 y_{LL1}$
LL2_y	$y_{LL2}$	Algeb	$LL_{2LT1z1} LL_{2LT2z1} (-x'_{LL2} + y_{LL2}) + T_7 (-x'_{LL2} + y_{LL1}) + T_8 x'_{LL2} - T_8 y_{LL2}$
LL3_y	$y_{LL3}$	Algeb	$LL_{3LT1z1} LL_{3LT2z1} (-x'_{LL3} + y_{LL3}) + T_9 (-x'_{LL3} + y_{LL2}) + T_{10} x'_{LL3} - T_{10} y_{LL3}$
VSS_x	$x_{VSS}$	Algeb	$-x_{VSS} + y_{LL3}$
VSS_y	$y_{VSS}$	Algeb	$L_{SMAX} VSS_{limzu} + L_{SMIN} VSS_{limzl} + VSS_{limzi} x_{VSS} - y_{VSS}$
tm	$\tau_m$	ExtAl- geb	0
te	$\tau_e$	ExtAl- geb	0
v	$V$	ExtAl- geb	0
f	$f$	ExtAl- geb	0
vi	$v_i$	ExtAl- geb	$uv_{sout}$
v2	$V$	ExtAl- geb	0
f2	$f_2$	ExtAl- geb	0

## Services

Name	Symbol	Equation	Type
VOU	$VOU$	$VCUr + V_0$	ConstService
VOL	$VOL$	$VCLr + V_0$	ConstService

## Discrete

Name	Symbol	Type	Info
dv	$dv$	Derivative	
dv2	$dv2$	Derivative	
SW	$SW$	Switcher	
SW2	$SW2$	Switcher	
WO_LT	$LT_{WO}$	LessThan	
LL1_LT1	$LT_{LL1}$	LessThan	
LL1_LT2	$LT_{LL1}$	LessThan	
LL2_LT1	$LT_{LL2}$	LessThan	
LL2_LT2	$LT_{LL2}$	LessThan	
LL3_LT1	$LT_{LL3}$	LessThan	
LL3_LT2	$LT_{LL3}$	LessThan	
VSS_lim	$lim_{VSS}$	HardLimiter	
OLIM	$OLIM$	Limiter	output limiter

Blocks

Name	Symbol	Type	Info
L1	$L1$	Lag	Transducer 1
L2	$L2$	Lag	Transducer 2
WO	$WO$	WashoutOrLag	
LL1	$LL1$	LeadLag	
LL2	$LL2$	LeadLag	
LL3	$LL3$	LeadLag	
VSS	$VSS$	GainLimiter	

Config Fields in [ST2CUT]

Option	Symbol	Value	Info	Accepted values
freq_model		BusFreq	default freq. measurement model	('BusFreq',)

## 5.13 PhasorMeasurement

Phasor measurements

Common Parameters: u, name

Common Variables: am, vm

Available models: *PMU*

### 5.13.1 PMU

Group *PhasorMeasurement*

Simple phasor measurement unit model.

This model tracks the bus voltage magnitude and phase angle, each using a low-pass filter.

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
bus		bus idx			IdxParam	mandatory
Ta	$T_a$	angle filter time constant	0.100		NumParam	
Tv	$T_v$	voltage filter time constant	0.100		NumParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
am	$\theta_m$	$\theta$	phase angle measurement	<i>rad.</i>	v_str
vm	$V_m$	$V$	voltage magnitude measurement	<i>p.u.(kV)</i>	v_str
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

Differential Equations

Name	Symbol	Type	RHS of Equation " $\dot{x} = f(x, y)$ "	T (LHS)
am	$\theta_m$	State	$\dot{\theta} - \theta_m$	$T_a$
vm	$V_m$	State	$\dot{V} - V_m$	$T_v$

Algebraic Equations

Name	Symbol	Type	RHS of Equation " $0 = g(x, y)$ "
a	$\theta$	ExtAlgeb	0
v	$V$	ExtAlgeb	0

## 5.14 RenAerodynamics

Renewable aerodynamics group.

Common Parameters: u, name, rego

Common Variables: theta

Available models: *WTARAI*

### 5.14.1 WTARA1

Group *RenAerodynamics*

Wind turbine aerodynamics model.

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
rego		Renewable governor idx			IdxParam	mandatory
Ka	$K_a$	Aerodynamics gain	1	<i>p.u./deg.</i>	NumParam	positive
theta0	$\theta_0$	Initial pitch angle	0	<i>deg.</i>	NumParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
theta	$\theta$	$\theta_{0r}$	Pitch angle	<i>rad</i>	v_str
Pmg	$Pmg$				

Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
theta	$\theta$	Algeb	$-\theta + \theta_{0r}$
Pmg	$Pmg$	ExtAlgeb	$-\theta (\theta - \theta_0)$

Services

Name	Symbol	Equation	Type
theta0r	$\theta_{0r}$	$\frac{\pi\theta_0}{180}$	ConstService

## 5.15 RenExciter

Renewable electrical control (exciter) group.

Common Parameters: u, name, reg

Common Variables: Pref, Qref, wg, Pord

Available models: *REECA1*

### 5.15.1 REECA1

Group *RenExciter*

Renewable energy electrical control.

A message *RuntimeWarning: invalid value encountered in sqrt* may show up following a bus-to-ground fault. This warning can be safely ignored.

#### Parameters

Name	Symbol	Description	Default	U
idx		unique device idx		
u	$u$	connection status	1	bo
name		device name		
reg		Renewable generator idx		
busr		Optional remote bus for voltage control		
PFFLAG		Power factor control flag; 1-PF control, 0-Q control		bo
VFLAG		Voltage control flag; 1-Q control, 0-V control		bo
QFLAG		Q control flag; 1-V or Q control, 0-const. PF or Q		bo
PFLAG		P speed-dependency flag; 1-has speed dep., 0-no dep.		bo
PQFLAG		P/Q priority flag for I limit; 0-Q priority, 1-P priority		bo
Vdip	$V_{dip}$	Low V threshold to activate Iqinj logic	0.800	p.
Vup	$V_{up}$	V threshold above which to activate Iqinj logic	1.200	p.
Trv	$T_{rv}$	Voltage filter time constant	0.020	
dbd1	$d_{bd1}$	Lower bound of the voltage deadband ( $\leq 0$ )	-0.020	
dbd2	$d_{bd2}$	Upper bound of the voltage deadband ( $\geq 0$ )	0.020	
Kqv	$K_{qv}$	Gain to compute Iqinj from V error	1	
Iqh1	$I_{qh1}$	Upper limit on Iqinj	999	
Iql1	$I_{ql1}$	Lower limit on Iqinj	-999	
Vref0	$V_{ref0}$	User defined Vref (if 0, use initial bus V)	1	
Iqfrz	$I_{qfrz}$	Value at which Iqinj is held for Thld (if >0) seconds following a Vdip	0	
Thld	$T_{hld}$	Time for which Iqinj is held. If >0, hold at Iqinj; if <0, hold at State 1 for abs(Thld)	0	s
Thld2	$T_{hld2}$	Time for which IPMAX is held after voltage dip ends	0	s
Tp	$T_p$	Filter time constant for Pe	0.020	s
QMax	$Q_{max}$	Upper limit for reactive power regulator	999	
QMin	$Q_{min}$	Lower limit for reactive power regulator	-999	
VMAX	$V_{max}$	Upper limit for voltage control	999	
VMIN	$V_{min}$	Lower limit for voltage control	-999	
Kqp	$K_{qp}$	Proportional gain for reactive power error	1	
Kqi	$K_{qi}$	Integral gain for reactive power error	0.100	
Kvp	$K_{vp}$	Proportional gain for voltage error	1	
Kvi	$K_{vi}$	Integral gain for voltage error	0.100	
Vref1	$V_{ref1}$	Voltage ref. if VFLAG=0	1	
Tiq	$T_{iq}$	Filter time constant for Iq	0.020	
dPmax	$d_{Pmax}$	Power reference max. ramp rate (>0)	999	
dPmin	$d_{Pmin}$	Power reference min. ramp rate (<0)	-999	
PMax	$P_{max}$	Max. active power limit > 0	999	
PMIN	$P_{min}$	Min. active power limit	0	
Imax	$I_{max}$	Max. apparent current limit	999	

Table 1 – continued from previous page

Name	Symbol	Description	Default	U
Tpord	$T_{pord}$	Filter time constant for power setpoint	0.020	
Vq1	$V_{q1}$	Reactive power V-I pair (point 1), voltage	0.200	
Iq1	$I_{q1}$	Reactive power V-I pair (point 1), current	2	
Vq2	$V_{q2}$	Reactive power V-I pair (point 2), voltage	0.400	
Iq2	$I_{q2}$	Reactive power V-I pair (point 2), current	4	
Vq3	$V_{q3}$	Reactive power V-I pair (point 3), voltage	0.800	
Iq3	$I_{q3}$	Reactive power V-I pair (point 3), current	8	
Vq4	$V_{q4}$	Reactive power V-I pair (point 4), voltage	1	
Iq4	$I_{q4}$	Reactive power V-I pair (point 4), current	10	
Vp1	$V_{p1}$	Active power V-I pair (point 1), voltage	0.200	
Ip1	$I_{p1}$	Active power V-I pair (point 1), current	2	
Vp2	$V_{p2}$	Active power V-I pair (point 2), voltage	0.400	
Ip2	$I_{p2}$	Active power V-I pair (point 2), current	4	
Vp3	$V_{p3}$	Active power V-I pair (point 3), voltage	0.800	
Ip3	$I_{p3}$	Active power V-I pair (point 3), current	8	
Vp4	$V_{p4}$	Active power V-I pair (point 4), voltage	1	
Ip4	$I_{p4}$	Active power V-I pair (point 4), current	12	
bus		Retrieved bus idx		
gen		Retrieved StaticGen idx		

## Variables

Name	Symbol	Initial Value
s0_y	$y_{s0}$	$V$
S1_y	$y_{S1}$	$Pe$
PIQ_xi	$xi_{PIQ}$	0.0
s4_y	$y_{s4}$	$\frac{PF_{sel}}{V_p}$
pfilt_y	$y_{P_{filt}}$	$P_{ref}$
s5_y	$y_{s5}$	$P_{sel}$
PIV_xi	$xi_{PIV}$	0.0
Pord	$P_{ord}$	
vp	$V_p$	$V z_i^{V_{Lower}} + 0.01 z_l^{V_{Lower}}$
pfaref	$\Phi_{ref}$	$\Phi_{ref0}$
Qcpf	$Q_{cpf}$	$Q_0$
Qref	$Q_{ref}$	$Q_0$
PFsel	$PF_{sel}$	$Q_{cpf} SWPF_{s1} + Q_{ref} SWPF_{s0}$
Qerr	$Q_{err}$	$PF_{sel} z_i^{PF_{lim}} + Q_{max} z_u^{PF_{lim}} + Q_{min} z_l^{PF_{lim}} - Q_e$
PIQ_ys	$ys_{PIQ}$	$K_{qp} Q_{err}$
PIQ_y	$y_{PIQ}$	$PIQ_{limzi} ys_{PIQ} + PIQ_{limzl} V_{min} + PIQ_{limzu} V_{max}$
Vsel_x	$x_{V_{sel}}$	$SWV_{s0} V_{ref1} + SWV_{s1} y_{PIQ}$
Vsel_y	$y_{V_{sel}}$	$V_{max} V_{sel_{limzu}} + V_{min} V_{sel_{limzl}} + V_{sel_{limzi}} x_{V_{sel}}$
Qsel	$Q_{sel}$	$SWQ_{s0} y_{s4} + SWQ_{s1} y_{PIV}$
Verr	$V_{err}$	$V_{ref0} - y_{s0}$

Table 2 – continued 1

Name	Symbol	Initial Value
dbV_y	$y_{dbV}$	$dbV_{dbzl} (V_{err} - d_{bd1}) + dbV_{dbzu} (V_{err} - d_{bd2})$
Iqinj	$I_{qinj}$	$K_{qv} y_{dbV} z_{Vdip} + fThld (1 - z_{Vdip}) (I_{qfrz} pThld + K_{qv} nThld y_{dbV})$
wg	$\omega_g$	1.0
Pref	$P_{ref}$	$\frac{P_0}{\omega_g}$
Psel	$P_{sel}$	$SW P_{s0} y_{P_{filt}} + SW P_{s1} \omega_g y_{P_{filt}}$
Ipulim	$I_{pulim}$	$\frac{y_{s5}}{V_p}$
VDL1_y	$y_{VDL1}$	$\begin{cases} I_{q1} & \text{for } V_{q1} \geq y_{s0} \\ I_{q1} + k_{Vq12} (-V_{q1} + y_{s0}) & \text{for } V_{q2} \geq y_{s0} \\ I_{q2} + k_{Vq23} (-V_{q2} + y_{s0}) & \text{for } V_{q3} \geq y_{s0} \\ I_{q3} + k_{Vq34} (-V_{q3} + y_{s0}) & \text{for } V_{q4} \geq y_{s0} \\ I_{q4} & \text{otherwise} \end{cases}$
VDL2_y	$y_{VDL2}$	$\begin{cases} I_{p1} & \text{for } V_{p1} \geq y_{s0} \\ I_{p1} + k_{Vp12} (-V_{p1} + y_{s0}) & \text{for } V_{p2} \geq y_{s0} \\ I_{p2} + k_{Vp23} (-V_{p2} + y_{s0}) & \text{for } V_{p3} \geq y_{s0} \\ I_{p3} + k_{Vp34} (-V_{p3} + y_{s0}) & \text{for } V_{p4} \geq y_{s0} \\ I_{p4} & \text{otherwise} \end{cases}$
Ipmax	$I_{pmax}$	$(1 - fThld_2) \left( \sqrt{I_{pmax20,nn}^2} SW P Q_{s0} + SW P Q_{s1} (z_{VDL2} (Imaxr (1 - VDL2c) + VDL2cy_{V_{DL2}}) - 100000000) \right)$
Iqmax	$I_{qmax}$	$\sqrt{I_{qmax,nn}^2} SW P Q_{s1} + SW P Q_{s0} (z_{VDL1} (Imaxr (1 - VDL1c) + VDL1cy_{V_{DL1}}) - 100000000)$
PIV_ys	$y_{SPIV}$	$K_{vp} (-SW V_{s0} y_{s0} + y_{Vsel})$
PIV_y	$y_{PIV}$	$I_{qmax} PIV_{limzu} + I_{qmin} PIV_{limzl} + PIV_{limzi} y_{SPIV}$
IpHL_x	$x_{IpHL}$	$\frac{y_{s5}}{V_p}$
IpHL_y	$y_{IpHL}$	$I_{pmax} IpHL_{limzu} + I_{pmin} IpHL_{limzl} + IpHL_{limzi} x_{IpHL}$
IqHL_x	$x_{IqHL}$	$I_{qinj} + Q_{sel}$
IqHL_y	$y_{IqHL}$	$I_{qmax} IqHL_{limzu} + I_{qmin} IqHL_{limzl} + IqHL_{limzi} x_{IqHL}$
a	$\theta$	
v	$V$	
Pe	$Pe$	
Qe	$Qe$	
Ipcmd	$Ipcmd$	
Iqcmd	$Iqcmd$	

Differential Equations

Name	Sym- bol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
s0_y	$y_{s0}$	State	$V - y_{s0}$	$T_{rv}$
S1_y	$y_{S1}$	State	$Pe - y_{S1}$	$T_p$
PIQ_xi	$xi_{PIQ}$	State	$K_{qi} (1 - z_{Vdip}) (Q_{err} + 2y_{PIQ} - 2ys_{PIQ})$	
s4_y	$y_{s4}$	State	$(1 - z_{Vdip}) \left( \frac{PF_{sel}}{V_p} - y_{s4} \right)$	$T_{iq}$
pfilt_y	$y_{P_{filt}}$	State	$P_{ref} - y_{P_{filt}}$	0.02
s5_y	$y_{s5}$	State	$(1 - z_{Vdip}) (P_{sel} - y_{s5})$	$T_{pord}$
PIV_xi	$xi_{PIV}$	State	$K_{vi} (1 - z_{Vdip}) (-SWV_{s0}y_{s0} + 2y_{PIV} + y_{V_{sel}} - 2ys_{PIV})$	
Pord	$P_{ord}$	AliasState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
vp	$V_p$	Algeb	$Vz_i^{V_{Lower}} - V_p + 0.01z_l^{V_{Lower}}$
pfaref	$\Phi_{ref}$	Algeb	$\Phi_{ref0} - \Phi_{ref}$
Qcpf	$Q_{cpf}$	Algeb	$-Q_{cpf} + y_{S1} \tan(\Phi_{ref})$
Qref	$Q_{ref}$	Algeb	$Q_0 - Q_{ref}$
PFsel	$PF_{sel}$	Algeb	$-PF_{sel} + Q_{cpf}SWPF_{s1} + Q_{ref}SWPF_{s0}$
Qerr	$Q_{err}$	Algeb	$PF_{sel}z_i^{PF_{lim}} - Q_{err} + Q_{max}z_u^{PF_{lim}} + Q_{min}z_l^{PF_{lim}} - Q_e$
PIQ_ys	$ys_{PIQ}$	Algeb	$(1 - z_{Vdip}) (K_{qp}Q_{err} + xi_{PIQ} - ys_{PIQ})$
PIQ_y	$y_{PIQ}$	Algeb	$(1 - z_{Vdip}) (PIQ_{limzi}ys_{PIQ} + PIQ_{limzl}V_{min} + PIQ_{limzu}V_{max} - y_{PIQ})$
Vsel_x	$x_{V_{sel}}$	Algeb	$SWV_{s0}V_{ref1} + SWV_{s1}y_{PIQ} - x_{V_{sel}}$
Vsel_y	$y_{V_{sel}}$	Algeb	$V_{max}V_{sel_{limzu}} + V_{min}V_{sel_{limzl}} + V_{sel_{limzi}}x_{V_{sel}} - y_{V_{sel}}$
Qsel	$Q_{sel}$	Algeb	$-Q_{sel} + SWQ_{s0}y_{s4} + SWQ_{s1}y_{PIV}$
Verr	$V_{err}$	Algeb	$-V_{err} + V_{ref0} - y_{s0}$
dbV_y	$y_{dbV}$	Algeb	$dbV_{dbzl} (V_{err} - d_{bd1}) + dbV_{dbzu} (V_{err} - d_{bd2}) - y_{dbV}$
Iqinj	$I_{qinj}$	Algeb	$-I_{qinj} + K_{qv}y_{dbV}z_{Vdip} + fThld (1 - z_{Vdip}) (I_{qfrz}pThld + K_{qv}nThldy_{dbV})$
wg	$\omega_g$	Algeb	$1.0 - \omega_g$
Pref	$P_{ref}$	Algeb	$\frac{P_0}{\omega_g} - P_{ref}$
Psel	$P_{sel}$	Algeb	$-P_{sel} + SWP_{s0}y_{P_{filt}} + SWP_{s1}\omega_gy_{P_{filt}}$
Ipulim	$I_{pulim}$	Algeb	$-I_{pulim} + \frac{ys_5}{V_p}$
VDL1_y	$y_{VDL1}$	Algeb	$-y_{VDL1} + \begin{cases} I_{q1} & \text{for } V_{q1} \geq y_{s0} \\ I_{q1} + k_{Vq12}(-V_{q1} + y_{s0}) & \text{for } V_{q2} \geq y_{s0} \\ I_{q2} + k_{Vq23}(-V_{q2} + y_{s0}) & \text{for } V_{q3} \geq y_{s0} \\ I_{q3} + k_{Vq34}(-V_{q3} + y_{s0}) & \text{for } V_{q4} \geq y_{s0} \\ I_{q4} & \text{otherwise} \end{cases}$
VDL2_y	$y_{VDL2}$	Algeb	$-y_{VDL2} + \begin{cases} I_{p1} & \text{for } V_{p1} \geq y_{s0} \\ I_{p1} + k_{Vp12}(-V_{p1} + y_{s0}) & \text{for } V_{p2} \geq y_{s0} \\ I_{p2} + k_{Vp23}(-V_{p2} + y_{s0}) & \text{for } V_{p3} \geq y_{s0} \\ I_{p3} + k_{Vp34}(-V_{p3} + y_{s0}) & \text{for } V_{p4} \geq y_{s0} \\ I_{p4} & \text{otherwise} \end{cases}$



Table 3 – continued from previous page

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Ipmax	$I_{pmax}$	Algeb	$-I_{pmax} + I_{pmax} h fThld_2 + (1 - fThld_2) \left( \sqrt{I_{pmax}^2} SWPQ_{s0} + SWPQ_{s1} (z_{VDL1} \right.$
Iqmax	$I_{qmax}$	Algeb	$\left. \sqrt{I_{qmax}^2} SWPQ_{s1} - I_{qmax} + SWPQ_{s0} (z_{VDL1} (Imaxr (1 - VDL1c) + VDL1c y_{PIV} \right.$
PIV_ys	$y_{sPIV}$	Algeb	$(1 - z_{Vdip}) (K_{vp} (-SWV_{s0} y_{s0} + y_{Vsel}) + x_{iPIV} - y_{sPIV})$
PIV_y	$y_{PIV}$	Algeb	$(1 - z_{Vdip}) (I_{qmax} PIV_{limzu} + I_{qmin} PIV_{limzl} + PIV_{limzi} y_{sPIV} - y_{PIV})$
IpHL_x	$x_{IpHL}$	Algeb	$-x_{IpHL} + \frac{y_{s5}}{V_p}$
IpHL_y	$y_{IpHL}$	Algeb	$I_{pmax} IpHL_{limzu} + I_{pmin} IpHL_{limzl} + IpHL_{limzi} x_{IpHL} - y_{IpHL}$
IqHL_x	$x_{IqHL}$	Algeb	$I_{qinj} + Q_{sel} - x_{IqHL}$
IqHL_y	$y_{IqHL}$	Algeb	$I_{qmax} IqHL_{limzu} + I_{qmin} IqHL_{limzl} + IqHL_{limzi} x_{IqHL} - y_{IqHL}$
a	$\theta$	ExtAlgeb	0
v	$V$	ExtAlgeb	0
Pe	$Pe$	ExtAlgeb	0
Qe	$Qe$	ExtAlgeb	0
Ipcmd	$Ipcmd$	ExtAlgeb	$-Ipcmd_0 + y_{IpHL}$
Iqcmd	$Iqcmd$	ExtAlgeb	$-Iqcmd_0 - y_{IqHL}$

Services

Name	Symbol	Equation	Type
Ipcmd0	$Ipcmd0$	$\frac{P_0}{V}$	ConstService
Iqcmd0	$Iqcmd0$	$-\frac{Q_0}{V}$	ConstService
pfaref0	$\Phi_{ref0}$	$\text{atan}\left(\frac{Q_0}{P_0}\right)$	ConstService
Volt_dip	$z_{Vdip}$	$1 - V_{cmpzi}$	VarService
PIQ_flag	$z_{PIQ}^{flag}$	0	EventFlag
s4_flag	$z_{s4}^{flag}$	0	EventFlag
pThld	$pThld$	$T_{hld} > 0$	ConstService
nThld	$nThld$	$T_{hld} < 0$	ConstService
Thld_abs	$ Thld $	$\text{abs}(T_{hld})$	ConstService
fThld	$fThld$	0	ExtendedEvent
s5_flag	$z_{s5}^{flag}$	0	EventFlag
kVq12	$k_{Vq12}$	$\frac{-I_{q1}+I_{q2}}{-V_{q1}+V_{q2}}$	ConstService
kVq23	$k_{Vq23}$	$\frac{-I_{q2}+I_{q3}}{-V_{q2}+V_{q3}}$	ConstService
kVq34	$k_{Vq34}$	$\frac{-I_{q3}+I_{q4}}{-V_{q3}+V_{q4}}$	ConstService
zVDL1	$z_{VDL1}$	$I_{q1} \leq I_{q2} \wedge I_{q2} \leq I_{q3} \wedge I_{q3} \leq I_{q4} \wedge V_{q1} \leq V_{q2} \wedge V_{q2} \leq V_{q3} \wedge V_{q3} \leq V_{q4}$	ConstService
kVp12	$k_{Vp12}$	$\frac{-I_{p1}+I_{p2}}{-V_{p1}+V_{p2}}$	ConstService
kVp23	$k_{Vp23}$	$\frac{-I_{p2}+I_{p3}}{-V_{p2}+V_{p3}}$	ConstService
kVp34	$k_{Vp34}$	$\frac{-I_{p3}+I_{p4}}{-V_{p3}+V_{p4}}$	ConstService
zVDL2	$z_{VDL2}$	$I_{p1} \leq I_{p2} \wedge I_{p2} \leq I_{p3} \wedge I_{p3} \leq I_{p4} \wedge V_{p1} \leq V_{p2} \wedge V_{p2} \leq V_{p3} \wedge V_{p3} \leq V_{p4}$	ConstService
fThld2	$fThld2$	0	ExtendedEvent
VDL1c	$VDL1c$	$y_{VDL1} < I_{maxr}$	VarService
VDL2c	$VDL2c$	$y_{VDL2} < I_{maxr}$	VarService
Ip-max2sq0	$I_{pmax20,nn}^2$	$\begin{cases} 0.0 & \text{for } I_{max}^2 - Iqcmd_0^2 \leq 0.0 \\ I_{max}^2 - Iqcmd_0^2 & \text{otherwise} \end{cases}$	ConstService
Ip-max2sq	$I_{pmax2}^2$	$\begin{cases} 0.0 & \text{for } I_{max}^2 - y_{IqHL}^2 \leq 0.0 \\ I_{max}^2 - y_{IqHL}^2 & \text{otherwise} \end{cases}$	VarService
Ipmaxh	$Ipmaxh$	0	VarHold
Ip-max2sq0	$I_{qmax,nn}^2$	$\begin{cases} 0.0 & \text{for } I_{max}^2 - Ipcmd_0^2 \leq 0.0 \\ I_{max}^2 - Ipcmd_0^2 & \text{otherwise} \end{cases}$	ConstService
Iq-	$I_{qmax2}^2$	$\begin{cases} 0.0 & \text{for } I_{max}^2 - y_{IqHL}^2 \leq 0.0 \\ I_{max}^2 - y_{IqHL}^2 & \text{otherwise} \end{cases}$	VarService

## Discrete

Name	Symbol	Type	Info
SWPF	$SW_{PF}$	Switcher	
SWV	$SW_V$	Switcher	
SWQ	$SW_V$	Switcher	
SWP	$SW_P$	Switcher	
SWPQ	$SW_{PQ}$	Switcher	
Vcmp	$V_{cmp}$	Limiter	Voltage dip comparator
VLower	$V_{Lower}$	Limiter	Limiter for lower voltage cap
PFlim	$P_{Flim}$	Limiter	
PIQ_lim	$lim_{PIQ}$	HardLimiter	
Vsel_lim	$lim_{V_{sel}}$	HardLimiter	
dbV_db	$db_{dbV}$	DeadBand	
pfilt_lim	$lim_{P_{filt}}$	RateLimiter	Rate limiter in Lag
s5_lim	$lim_{s5}$	AntiWindup	Limiter in Lag
PIV_lim	$lim_{PIV}$	HardLimiter	
IpHL_lim	$lim_{IpHL}$	HardLimiter	
IqHL_lim	$lim_{IqHL}$	HardLimiter	

## Blocks

Name	Symbol	Type	Info
s0	$s_0$	Lag	Voltage filter
S1	$S_1$	Lag	Pe filter
PIQ	$PIQ$	PITrackAWFreeze	
Vsel	$V_{sel}$	GainLimiter	Selection output of VFLAG
s4	$s_4$	LagFreeze	Filter for calculated voltage with freeze
dbV	$dbV$	DeadBand1	Deadband for voltage error (ref0)
pfilt	$P_{filt}$	LagRate	Active power filter with rate limits
s5	$s_5$	LagAWFreeze	
VDL1	$V_{DL1}$	Piecewise	Piecewise linear characteristics of Vq-Iq
VDL2	$V_{DL2}$	Piecewise	Piecewise linear characteristics of Vp-Ip
PIV	$PIV$	PITrackAWFreeze	
IpHL	$IpHL$	GainLimiter	
IqHL	$IqHL$	GainLimiter	

## Config Fields in [REECA1]

Option	Symbol	Value	Info	Accepted values
kqs	$K_{qs}$	2	Q PI controller tracking gain	
kvs	$K_{vs}$	2	Voltage PI controller tracking gain	
tpfilt	$T_{pfilt}$	0.020	Time const. for Pref filter	

## 5.16 RenGen

Renewable generator (converter) group.

Common Parameters: u, name, bus, gen

Common Variables: Ipcmd, Iqcmd, Pe, Qe

Available models: *REGCAI*

### 5.16.1 REGCA1

Group *RenGen*

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
bus		interface bus id			Idx- Param	manda- tory
gen		static generator index			Idx- Param	manda- tory
Tg	$T_g$	converter time const.	0.100	<i>s</i>	NumParam	
Rrpwr	$R_{rpwr}$	Low voltage power logic (LVPL) ramp limit	999	<i>p.u.</i>	NumParam	
Brkpt	$B_{rkpt}$	LVPL characteristic voltage 2	1	<i>p.u.</i>	NumParam	
Ze- rox	$Z_{erox}$	LVPL characteristic voltage 1	0.500	<i>p.u.</i>	NumParam	
Lvpl1	$L_{vpl1}$	LVPL gain	1	<i>p.u.</i>	NumParam	
Volim	$V_{olim}$	Voltage lim for high volt. reactive current mgnt.	1.200	<i>p.u.</i>	NumParam	
Lvpnt1	$L_{vpnt1}$	High volt. point for low volt. active current mgnt.	1	<i>p.u.</i>	NumParam	
Lvpnt0	$L_{vpnt0}$	Low volt. point for low volt. active current mgnt.	0.400	<i>p.u.</i>	NumParam	
Iolim	$I_{olim}$	lower current limit for high volt. reactive current mgnt.	0	<i>p.u.</i>	NumParam	
Tfltr	$T_{fltr}$	Voltage filter T const for low volt. active current mgnt.	0.100	<i>s</i>	NumParam	
Khv	$K_{hv}$	Overvolt. compensation gain in high volt. reac- tive current mgnt.	0.700		NumParam	
Iqr- max	$I_{qrmax}$	Upper limit on the ROC for reactive current	999	<i>p.u.</i>	NumParam	
Iqr- min	$I_{qrmin}$	Lower limit on the ROC for reactive current	-999	<i>p.u.</i>	NumParam	
Ac- cel	$A_{ccel}$	Acceleration factor	0		NumParam	
Iq- max	$I_{qmax}$	Upper limit for reactive current	999	<i>p.u.</i>	NumParam	
Iqmin	$I_{qmin}$	Lower limit for reactive current	-999	<i>p.u.</i>	NumParam	
ra	$r_a$		0		ExtParam	
xs	$x_s$		0		ExtParam	

Variables

Name	Sym- bol	Initial Value	Description	Unit	Prop- erties
S1_y	$y_{S_1}$	$-I_{qcmd}$	State in lag TF		v_str
S2_y	$y_{S_2}$	$1.0V$	State in lag transfer function		v_str
S0_y	$y_{S_0}$	$I_{pcmd}$	State in lag TF		v_str
Ipcmd	$I_{pcmd}$	$I_{pcmd0}$	current component for active power		v_str
Iqcmd	$I_{qcmd}$	$I_{qcmd0}$	current component for reactive power		v_str
LVG_y	$y_{LVG}$	$\begin{cases} 0 & \text{for } L_{vpnt0} \geq V \\ k_{LVG}(-L_{vpnt0} + V) & \text{for } L_{vpnt1} \geq V \\ 1 & \text{otherwise} \end{cases}$	Output of piecewise		v_str
LVPL_y	$y_{LVPL}$	$\begin{cases} 9999 - 9999z_{Lvplsw} & \text{for } Z_{erox} \geq y_{S_2} \\ k_{LVPL}(-Z_{erox} + y_{S_2}) - 9999z_{Lvplsw} + 9999 & \text{for } Z_{erox} < y_{S_2} \\ 9999 & \text{otherwise} \end{cases}$	Output of piecewise		v_str
Ipout	$I_{pout}$	$I_{pcmd}y_{LVG}$	Output Ip current		v_str
HVG_x	$x_{HVG}$	$K_{hv}(V - V_{olim})$	Gain output before limiter		v_str
HVG_y	$y_{HVG}$	$HVG_{limzi}x_{HVG}$	Gain output after limiter		v_str
Iqout_x	$x_{Iqout}$	$-y_{HVG} + y_{S_1}$	Gain output before limiter		v_str
Iqout_y	$y_{Iqout}$	$I_{olim}Iqout_{limzl} + Iqout_{limzi}x_{Iqout}$	Gain output after limiter		v_str
Pe	$P_e$	$P_0$	Active power output		v_str
Qe	$Q_e$	$Q_0$	Reactive power output		v_str
a	$\theta$		Bus voltage angle		
v	$V$		Bus voltage magnitude		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
S1_y	$y_{S_1}$	State	$-I_{qcmd} - y_{S_1}$	$T_g$
S2_y	$y_{S_2}$	State	$1.0V - y_{S_2}$	$T_{fltr}$
S0_y	$y_{S_0}$	State	$I_{pcmd} - y_{S_0}$	$T_g$

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Ipcmd	$I_{pcmd}$	Al-geb	$I_{pcmd0} - I_{pcmd}$
Iqcmd	$I_{qcmd}$	Al-geb	$I_{qcmd0} - I_{qcmd}$
LVG_y	$y_{LVG}$	Al-geb	$-y_{LVG} + \begin{cases} 0 & \text{for } L_{vpnt0} \geq V \\ k_{LVG}(-L_{vpnt0} + V) & \text{for } L_{vpnt1} \geq V \\ 1 & \text{otherwise} \end{cases}$
LVPL_y	$y_{LVPL}$	Al-geb	$-y_{LVPL} + \begin{cases} 9999 - 9999z_{Lvplsw} & \text{for } Z_{erox} \geq y_{S_2} \\ k_{LVPL}(-Z_{erox} + y_{S_2}) - 9999z_{Lvplsw} + 9999 & \text{for } B_{rkpt} \geq y_{S_2} \\ 9999 & \text{otherwise} \end{cases}$
Ipout	$I_{pout}$	Al-geb	$-I_{pout} + y_{LVG}y_{S_0}$
HVG_x	$x_{HVG}$	Al-geb	$K_{hv}(V - V_{olim}) - x_{HVG}$
HVG_y	$y_{HVG}$	Al-geb	$HVG_{limzi}x_{HVG} - y_{HVG}$
Iqout_x	$x_{Iqout}$	Al-geb	$-x_{Iqout} - y_{HVG} + y_{S_1}$
Iqout_y	$y_{Iqout}$	Al-geb	$I_{olim}Iqout_{limzl} + Iqout_{limzi}x_{Iqout} - y_{Iqout}$
Pe	$P_e$	Al-geb	$I_{pout}V - P_e$
Qe	$Q_e$	Al-geb	$-Q_e + Vy_{Iqout}$
a	$\theta$	ExtAl-geb	$-P_e$
v	$V$	ExtAl-geb	$-Q_e$

Services

Name	Symbol	Equation	Type
Ipcmd0	$I_{pcmd0}$	$\frac{P_0}{V}$	ConstService
Iqcmd0	$I_{qcmd0}$	$-\frac{Q_0}{V}$	ConstService
kLVG	$k_{LVG}$	$\frac{1}{-L_{vpnt0} + L_{vpnt1}}$	ConstService
kLVPL	$k_{LVPL}$	$\frac{L_{vpl1}z_{Lvplsw}}{B_{rkpt} - Z_{erox}}$	ConstService

Discrete

Name	Symbol	Type	Info
S1_lim	$lim_{S_1}$	AntiWindupRate	Limiter in Lag
S0_lim	$lim_{S_0}$	AntiWindupRate	Limiter in Lag
HVG_lim	$lim_{HVG}$	HardLimiter	
Iqout_lim	$lim_{I^{qout}}$	HardLimiter	

Blocks

Name	Symbol	Type	Info
S1	$S_1$	LagAntiWindupRate	Iqcmd delay
LVG	$L_{VG}$	Piecewise	Low voltage current gain
S2	$S_2$	Lag	Voltage filter with no anti-windup
LVPL	$L_{VPL}$	Piecewise	Low voltage Ipcmd upper limit
S0	$S_0$	LagAntiWindupRate	
HVG	$H_{VG}$	GainLimiter	High voltage gain block
Iqout	$I^{qout}$	GainLimiter	Iq output block

## 5.17 RenGovernor

Renewable turbine governor group.

Common Parameters: u, name, ree, w0, Sn, Pe0

Common Variables: Pm, wr0, wt, wg, s3\_y

Available models: *WTDTA1*, *WTDS*

### 5.17.1 WTDTA1

Group *RenGovernor*

WTDTA wind turbine drive-train model.

User-provided reference speed should be specified in parameter *w0*. Internally, *w0* is set to the algebraic variable *wr0*.

Parameters



Name	Sym- bol	Description	De- fault	Unit	Type	Properties
idx		unique device idx			Data-Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data-Param	
ree		Renewable exciter idx			Idx-Param	mandatory
Sn	$S_n$	Model MVA base	100	<i>MVA</i>	NumParam	
fn	$f_n$	nominal frequency	60	<i>Hz</i>	NumParam	
Ht	$H_t$	Turbine inertia	3	<i>MWs/MVA</i>	NumParam	non_zero,power
Hg	$H_g$	Generator inertia	3	<i>MWs/MVA</i>	NumParam	non_zero,power
Dshaft	$D_{shaft}$	Damping coefficient	1	<i>p.u. (gen base)</i>	NumParam	power
Kshaft	$K_{shaft}$	Spring constant	1	<i>p.u. (gen base)</i>	NumParam	power
w0	$\omega_0$	Default speed if not using a torque model	1	<i>p.u.</i>	NumParam	
reg			0		ExtParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
s1_y	$y_{s1}$	$\omega_{r0}$	Integrator output		v_str
s2_y	$y_{s2}$	$\omega_{r0}$	Integrator output		v_str
s3_y	$y_{s3}$	$\frac{P_{e0}}{K_{shaft}\omega_{r0}}$	Integrator output		v_str
wt	$\omega_t$		Alias of s1_y		
wg	$\omega_g$		Alias of s2_y		
wr0	$\omega_{r0}$	$\omega_0$	speed set point	<i>p.u.</i>	v_str
Pm	$P_m$	$P_{e0}$	Mechanical power		v_str
pd	$P_d$	0.0	Output after damping		v_str
wge	$wge$				
Pe	$Pe$		Retrieved Pe of RenGen		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
s1_y	$y_{s1}$	State	$-1.0K_{shaft}y_{s3} - 1.0P_d + \frac{1.0P_m}{y_{s1}}$	$2H_t$
s2_y	$y_{s2}$	State	$1.0K_{shaft}y_{s3} + 1.0P_d - \frac{1.0P_e}{y_{s2}}$	$2H_g$
s3_y	$y_{s3}$	State	$1.0y_{s1} - 1.0y_{s2}$	1.0
wt	$\omega_t$	AliasState	0	
wg	$\omega_g$	AliasState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
wr0	$\omega_{r0}$	Algeb	$\omega_0 - \omega_{r0}$
Pm	$P_m$	Algeb	$-P_m + P_{e0}$
pd	$P_d$	Algeb	$D_{shaft}(y_{s1} - y_{s2}) - P_d$
wge	$wge$	ExtAlgeb	$y_{s2} - 1.0$
Pe	$P_e$	ExtAlgeb	0

#### Services

Name	Symbol	Equation	Type
Ht2	$2H_t$	$2H_t$	ConstService
Hg2	$2H_g$	$2H_g$	ConstService

#### Blocks

Name	Symbol	Type	Info
s1	$s1$	Integrator	
s2	$s2$	Integrator	
s3	$s3$	Integrator	

### 5.17.2 WTDS

#### Group *RenGovernor*

Custom wind turbine model with a single swing-equation.

This model is used to simulate the mechanical swing of the combined machine and turbine mass. The speed output is `s1_y` which will be fed to `RenExciter.wg`.

PFLAG needs to be set to 1 in exciter to consider speed for Pref.

#### Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Properties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	$bool$	NumParam	
name		device name			Data- Param	
ree		Renewable exciter idx			Idx- Param	mandatory
Sn	$S_n$	Model MVA base	100	$MVA$	NumParam	
fn	$f_n$	nominal frequency	60	$Hz$	NumParam	
H	$H_t$	Total inertia	3	$MWs/MVA$	NumParam	non_zero,power
D	$D_{shaft}$	Damping coefficient	1	$p.u.$	NumParam	power
w0	$\omega_0$	Default speed if not using a torque model	1	$p.u.$	NumParam	
reg			0		ExtParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
s1_y	$y_{s1}$	$\omega_{r0}$	Integrator output		v_str
s3_y	$y_{s3}$		Dummy state variable		
wt	$\omega_t$		Alias of s1_y		
wg	$\omega_g$		Alias of s1_y		
Pm	$P_m$	$P_{e0}$	Mechanical power		v_str
wr0	$\omega_{r0}$	$\omega_0$	speed set point	$p.u.$	v_str
wge	$wge$				
Pe	$P_e$		Retrieved Pe of RenGen		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
s1_y	$y_{s1}$	State	$-1.0D_{shaft}(-\omega_{r0} + y_{s1}) + \frac{1.0(P_m - P_e)}{wge}$	$2H$
s3_y	$y_{s3}$	State	0	
wt	$\omega_t$	AliasState	0	
wg	$\omega_g$	AliasState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Pm	$P_m$	Algeb	$-P_m + P_{e0}$
wr0	$\omega_{r0}$	Algeb	$\omega_0 - \omega_{r0}$
wge	$wge$	ExtAlgeb	$y_{s1} - 1.0$
Pe	$P_e$	ExtAlgeb	0

## Services

Name	Symbol	Equation	Type
H2	$2H$	$2H_t$	ConstService
Kshaft	$K_{shaft}$	1.0	ConstService

## Blocks

Name	Symbol	Type	Info
s1	$s1$	Integrator	

## 5.18 RenPitch

Renewable generator pitch controller group.

Common Parameters: u, name, rea

Available models: *WTPTA1*

### 5.18.1 WTPTA1

Group *RenPitch*

Wind turbine pitch control model.

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
rea		Renewable aerodynamics model idx			IdxParam	mandatory
Kiw	$K_{iw}$	Pitch-control integral gain	0.100	<i>p.u.</i>	NumParam	
Kpw	$K_{pw}$	Pitch-control proportional gain	0	<i>p.u.</i>	NumParam	
Kic	$K_{ic}$	Pitch-compensation integral gain	0.100	<i>p.u.</i>	NumParam	
Kpc	$K_{pc}$	Pitch-compensation proportional gain	0	<i>p.u.</i>	NumParam	
Kcc	$K_{cc}$	Gain for P diff	0	<i>p.u.</i>	NumParam	
Tp	$T_{\theta}$	Blade response time const.	0.300	<i>s</i>	NumParam	
thmax	$\theta_{max}$	Max. pitch angle	30	<i>deg.</i>	NumParam	
thmin	$\theta_{min}$	Min. pitch angle	0	<i>deg.</i>	NumParam	
dth- max	$\theta_{max}$	Max. pitch angle rate	5	<i>deg.</i>	NumParam	
dthmin	$\theta_{min}$	Min. pitch angle rate	-5	<i>deg.</i>	NumParam	
rego			0		ExtParam	
ree			0		ExtParam	

## Variables

Name	Sym- bol	Initial Value	Description	Unit	Proper- ties
Plc_xi	$xi_{PI_c}$	0.0	Integrator output		v_str
PIw_xi	$xi_{PI_w}$	0.0	Integrator output		v_str
LG_y	$y_{LG}$	$1.0y_{PI_c} + 1.0y_{PI_w}$	State in lag TF		v_str
Plc_yul	$y_{PI_c}^{ul}$	$K_{pc}(Pord - Pref)$			v_str
Plc_y	$y_{PI_c}$	$PI_{chlzi}y_{PI_c}^{ul} + PI_{chlzl}\theta_{min} + PI_{chlzu}\theta_{max}$	PI output		v_str
wref	$\omega_{ref}$	$wt$	optional speed reference		v_str
PIw_yul	$y_{PI_w}^{ul}$	$K_{pw}(K_{cc}(Pord - Pref) - \omega_{ref} + wt)$			v_str
PIw_y	$y_{PI_w}$	$PI_{w_hlzi}y_{PI_w}^{ul} + PI_{w_hlzl}\theta_{min} + PI_{w_hlzu}\theta_{max}$	PI output		v_str
wt	$wt$				
theta	$\theta$				
Pord	$Pord$				
Pref	$Pref$				

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
PIc_xi	$xi_{PI_c}$	State	$K_{ic}(Pord - Pref)$	
PIw_xi	$xi_{PI_w}$	State	$K_{iw}(K_{cc}(Pord - Pref) - \omega_{ref} + wt)$	
LG_y	$y_{LG}$	State	$-y_{LG} + 1.0y_{PI_c} + 1.0y_{PI_w}$	$T_\theta$

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
PIc_yul	$y_{PI_c}^{ul}$	Algeb	$K_{pc}(Pord - Pref) + xi_{PI_c} - y_{PI_c}^{ul}$
PIc_y	$y_{PI_c}$	Algeb	$PI_{chlzi}y_{PI_c}^{ul} + PI_{chlzi}\theta_{min} + PI_{chlzu}\theta_{max} - y_{PI_c}$
wref	$\omega_{ref}$	Algeb	$-\omega_{ref} + wt$
PIw_yul	$y_{PI_w}^{ul}$	Algeb	$K_{pw}(K_{cc}(Pord - Pref) - \omega_{ref} + wt) + xi_{PI_w} - y_{PI_w}^{ul}$
PIw_y	$y_{PI_w}$	Algeb	$PI_{whlzi}y_{PI_w}^{ul} + PI_{whlzi}\theta_{min} + PI_{whlzu}\theta_{max} - y_{PI_w}$
wt	$wt$	ExtAlgeb	0
theta	$\theta$	ExtAlgeb	$-\theta_0 + y_{LG}$
Pord	$Pord$	ExtAlgeb	0
Pref	$Pref$	ExtAlgeb	0

## Discrete

Name	Symbol	Type	Info
PIc_aw	$aw_{PI_c}$	AntiWindup	
PIc_hl	$hl_{PI_c}$	HardLimiter	
PIw_aw	$aw_{PI_w}$	AntiWindup	
PIw_hl	$hl_{PI_w}$	HardLimiter	
LG_lim	$lim_{LG}$	AntiWindupRate	Limiter in Lag

## Blocks

Name	Symbol	Type	Info
PIc	$PI_c$	PIAWHardLimit	PI for active power diff compensation
PIw	$PI_w$	PIAWHardLimit	PI for speed and active power deviation
LG	$LG$	LagAntiWindupRate	Output lag anti-windup rate limiter

## 5.19 RenPlant

Renewable plant control group.

Common Parameters: u, name

Available models: *REPCA1*

### 5.19.1 REPCA1

Group *RenPlant*

REPCA1 plat control model.

Parameters

Name	Symbol	Description	Default	Unit	Type	P
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
ree		RenExciter idx			IdxParam	m
line		Idx of line that connect to measured bus			IdxParam	m
busr		Optional remote bus for voltage and freq. measurement			IdxParam	
busf		BusFreq idx for mode 2			IdxParam	
VCFlag		Droop flag; 0-with droop if power factor ctrl, 1-line drop comp.		<i>bool</i>	NumParam	m
RefFlag		Q/V select; 0-Q control, 1-V control		<i>bool</i>	NumParam	m
Fflag		Frequency control flag; 0-disable, 1-enable		<i>bool</i>	NumParam	m
Tfltr	$T_{fltr}$	V or Q filter time const.	0.020		NumParam	
Kp	$K_p$	Q proportional gain	1		NumParam	
Ki	$K_i$	Q integral gain	0.100		NumParam	
Tft	$T_{ft}$	Lead time constant	1		NumParam	
Tfv	$T_{fv}$	Lag time constant	1		NumParam	
Vfrz	$V_{frz}$	Voltage below which s2 is frozen	0.800		NumParam	
Rc	$R_c$	Line drop compensation R			NumParam	
Xc	$X_c$	Line drop compensation R			NumParam	
Kc	$K_c$	Reactive power compensation gain	0		NumParam	
emax	$e_{max}$	Upper limit on deadband output	999		NumParam	
emin	$e_{min}$	Lower limit on deadband output	-999		NumParam	
dbd1	$dbd1$	Lower threshold for reactive power control deadband ( $\leq 0$ )	-0.100		NumParam	P
dbd2	$dbd2$	Upper threshold for reactive power control deadband ( $\geq 0$ )	0.100		NumParam	
Qmax	$Q_{max}$	Upper limit on output of V-Q control	999		NumParam	
Qmin	$Q_{min}$	Lower limit on output of V-Q control	-999		NumParam	
Kpg	$K_{pg}$	Proportional gain for power control	1		NumParam	
Kig	$K_{ig}$	Integral gain for power control	0.100		NumParam	
Tp	$T_p$	Time constant for P measurement	0.020		NumParam	
fdbd1	$f_{dbd1}$	Lower threshold for freq. error deadband	-0.010		NumParam	
fdbd2	$f_{dbd2}$	Upper threshold for freq. error deadband	0.010		NumParam	
femax	$f_{emax}$	Upper limit for freq. error	0.050		NumParam	
femin	$f_{emin}$	Lower limit for freq. error	-0.050		NumParam	
Pmax	$P_{max}$	Upper limit on power error (used by PI ctrl.)	999		NumParam	
Pmin	$P_{min}$	Lower limit on power error (used by PI ctrl.)	-999		NumParam	
Tg	$T_g$	Power controller lag time constant	0.020		NumParam	
Ddn	$D_{dn}$	Reciprocal of droop for over-freq. conditions	10		NumParam	
Dup	$D_{up}$	Reciprocal of droop for under-freq. conditions	10		NumParam	

Continued on n

Table 4 – continued from previous page

Name	Symbol	Description	Default	Unit	Type	P
reg		Retrieved RenGen idx			ExtParam	
bus		Retrieved bus idx			ExtParam	
bus1		Retrieved Line.bus1 idx			ExtParam	
bus2		Retrieved Line.bus2 idx			ExtParam	
r		Retrieved Line.r			ExtParam	
x		Retrieved Line.x			ExtParam	

## Variables

Name	Symbol	Initial Value	Description
s0_y	$y_{s0}$	$SWVC_{s0}(K_c Q_{line} + V) + SWVC_{s1} V_{comp}$	State in lag transfer function
s1_y	$y_{s1}$	$Q_{line}$	State in lag transfer function
s2_xi	$x'_{s2}$	0.0	Integrator output
s3_x	$x'_{s3}$	$y_{s2}$	State in lead-lag
s4_y	$y_{s4}$	$P_{line}$	State in lag transfer function
s5_xi	$x'_{s5}$	0.0	Integrator output
s6_y	$y_{s6}$	$y_{s5}$	State in lag transfer function
Vref	$Q_{ref}$	$V_{ref0}$	
Qlinef	$Q_{linef}$	$Q_{line0}$	
Refsel	$R_{efsel}$	$SWRef_{s0}(Q_{linef} - y_{s1}) + SWRef_{s1}(Q_{ref} - y_{s0})$	
dbd_y	$y_{dbd}$	$dbd_{dbzl}(R_{efsel} - d_{bd1}) + dbd_{dbzu}(R_{efsel} - d_{bd2})$	Deadband type 1 output
enf	$e_{nf}$	$eHL_{zi}y_{dbd} + eHL_{zl}e_{min} + eHL_{zu}e_{max}$	e Hardlimit output before freeze
s2_ys	$y_{s2}$	$K_p e_{hld}$	PI summation before limit
s2_y	$y_{s2}$	$Q_{max}s_{2limzu} + Q_{min}s_{2limzl} + s_{2limzi}y_{s2}$	PI output
s3_y	$y_{s3}$	$y_{s2}$	Output of lead-lag
ferr	$f_{err}$	$-f + f_{ref}$	Frequency deviation
fdbd_y	$y_{fdbd}$	$fdbd_{dbzl}(-f_{dbd1} + f_{err}) + fdbd_{dbzu}(-f_{dbd2} + f_{err})$	Deadband type 1 output
Plant_pref	$P_{ref}$	$P_{line0}$	Plant P ref
Plerr	$P_{lerr}$	$P_{ref} - y_{s4}$	Pline error
Perr	$P_{err}$	$D_{dn}f_{dlt0z1}y_{fdbd} + D_{up}f_{dlt0z0}y_{fdbd} + P_{lerr}$	Power error before fe limits
s5_ys	$y_{s5}$	$K_{pg}(P_{err}f_{eHL_{zi}} + f_{emax}f_{eHL_{zu}} + f_{emin}f_{eHL_{zl}})$	PI summation before limit
s5_y	$y_{s5}$	$P_{max}s_{5limzu} + P_{min}s_{5limzl} + s_{5limzi}y_{s5}$	PI output
Pext	$P_{ext}$		Pref from RenExciter renamed as Pext
Qext	$Q_{ext}$		Qref from RenExciter renamed as Qext
v	$V$		Bus (or busr, if given) terminal voltage
a	$\theta$		Bus (or busr, if given) phase angle
f	$f$		Bus frequency
v1	$V_1$		Voltage at Line.bus1
v2	$V_2$		Voltage at Line.bus2
a1	$\theta_1$		Angle at Line.bus1
a2	$\theta_2$		Angle at Line.bus2

## Differential Equations



Name	Sym- bol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
s0_y	$y_{s0}$	State	$SWVC_{s0} (K_c Q_{line} + V) + SWVC_{s1} V_{comp} - y_{s0}$	$T_{fltr}$
s1_y	$y_{s1}$	State	$Q_{line} - y_{s1}$	$T_{fltr}$
s2_xi	$xi_{s2}$	State	$K_i (e_{hld} + 2y_{s2} - 2ys_{s2})$	
s3_x	$x'_{s3}$	State	$-x'_{s3} + y_{s2}$	$T_{fv}$
s4_y	$y_{s4}$	State	$P_{line} - y_{s4}$	$T_p$
s5_xi	$xi_{s5}$	State	$K_{ig} (P_{err} feHL_{zi} + f_{emax} feHL_{zu} + f_{emin} feHL_{zl} + 2y_{s5} - 2ys_{s5})$	
s6_y	$y_{s6}$	State	$y_{s5} - y_{s6}$	$T_g$

Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
Vref	$Q_{ref}$	Algeb	$-Q_{ref} + V_{ref0}$
Qlinef	$Q_{linef}$	Algeb	$Q_{line0} - Q_{linef}$
Refsel	$R_{efsel}$	Algeb	$-R_{efsel} + SWRef_{s0}(Q_{linef} - y_{s1}) + SWRef_{s1}(Q_{ref} - y_{s0})$
dbd_y	$y_{dbd}$	Algeb	$dbd_{dbzl}(R_{efsel} - d_{bd1}) + dbd_{dbzu}(R_{efsel} - d_{bd2}) - y_{dbd}$
enf	$e_{nf}$	Algeb	$eHL_{zi}y_{dbd} + eHL_{zl}e_{min} + eHL_{zu}e_{max} - e_{nf}$
s2_ys	$ys_{s2}$	Algeb	$K_{pe}h_{ld} + xi_{s2} - ys_{s2}$
s2_y	$y_{s2}$	Algeb	$Q_{max}s_{2limzu} + Q_{min}s_{2limzl} + s_{2limzi}ys_{s2} - y_{s2}$
s3_y	$y_{s3}$	Algeb	$T_{ft}(-x'_{s3} + y_{s2}) + T_{fv}x'_{s3} - T_{fv}y_{s3} + s_{3LT1z1}s_{3LT2z1}(-x'_{s3} + y_{s3})$
ferr	$f_{err}$	Algeb	$-f - f_{err} + f_{ref}$
fdbd_y	$y_{fdbd}$	Algeb	$fdbd_{dbzl}(-f_{dbd1} + f_{err}) + fdbd_{dbzu}(-f_{dbd2} + f_{err}) - y_{fdbd}$
Plant_pref	$P_{ref}$	Algeb	$P_{line0} - P_{ref}$
Plerr	$P_{lerr}$	Algeb	$-P_{lerr} + P_{ref} - y_{s4}$
Perr	$P_{err}$	Algeb	$D_{dn}fdlt_{0z1}y_{fdbd} + D_{up}fdlt_{0z0}y_{fdbd} - P_{err} + P_{lerr}$
s5_ys	$ys_{s5}$	Algeb	$K_{pg}(P_{err}feHL_{zi} + f_{emax}feHL_{zu} + f_{emin}feHL_{zl}) + xi_{s5} - ys_{s5}$
s5_y	$y_{s5}$	Algeb	$P_{max}s_{5limzu} + P_{min}s_{5limzl} + s_{5limzi}ys_{s5} - y_{s5}$
Pext	$P_{ext}$	ExtAl- geb	$SWF_{s1}y_{s6}$
Qext	$Q_{ext}$	ExtAl- geb	$y_{s3}$
v	$V$	ExtAl- geb	0
a	$\theta$	ExtAl- geb	0
f	$f$	ExtAl- geb	0
v1	$V_1$	ExtAl- geb	0
v2	$V_2$	ExtAl- geb	0
a1	$\theta_1$	ExtAl- geb	0
a2	$\theta_2$	ExtAl- geb	0

Services

Name	Symbol	Equation	Type
Isign	$I_{sign}$	0	CurrentSign
Iline	$I_{line}$	$\frac{I_{sign}(V_1 e^{i\theta_1} - V_2 e^{i\theta_2})}{r + ix}$	VarService
Iline0	$I_{line0}$	$I_{line}$	ConstService
Pline	$P_{line}$	$\text{re} \left( I_{sign} V_1 \text{conj} \left( \frac{V_1 e^{i\theta_1} - V_2 e^{i\theta_2}}{r + ix} \right) e^{i\theta_1} \right)$	VarService
Pline0	$P_{line0}$	$P_{line}$	ConstService
Qline	$Q_{line}$	$\text{im} \left( I_{sign} V_1 \text{conj} \left( \frac{V_1 e^{i\theta_1} - V_2 e^{i\theta_2}}{r + ix} \right) e^{i\theta_1} \right)$	VarService
Qline0	$Q_{line0}$	$Q_{line}$	ConstService
Vcomp	$V_{comp}$	$\text{abs} \left( -I_{line} (R_{cs} + iX_{cs}) + V e^{i\theta} \right)$	VarService
Vref0	$V_{ref0}$	$SWVC_{s0} (K_c Q_{line0} + V) + SWVC_{s1} V_{comp}$	ConstService
zf	$z_f$	$f_{rz} (V < V_{frz})$	VarService
eHld	$e_{hld}$	0	VarHold
Freq_ref	$f_{ref}$	1.0	ConstService

## Discrete

Name	Symbol	Type	Info
SWVC	$SW_{VC}$	Switcher	
SWRef	$SW_{Ref}$	Switcher	
SWF	$SW_F$	Switcher	
dbd_db	$db_{dbd}$	DeadBand	
eHL	$e_{HL}$	Limiter	Hardlimit on deadband output
s2_lim	$lim_{s_2}$	HardLimiter	
s3_LT1	$LT_{s_3}$	LessThan	
s3_LT2	$LT_{s_3}$	LessThan	
fdbd_db	$db_{fdbd}$	DeadBand	
fdlt0	$f_{dlt0}$	LessThan	frequency deadband output less than zero
feHL	$f_{eHL}$	Limiter	Limiter for power (frequency) error
s5_lim	$lim_{s_5}$	HardLimiter	

## Blocks

Name	Symbol	Type	Info
s0	$s_0$	Lag	V filter
s1	$s_1$	Lag	
dbd	$d^{bd}$	DeadBand1	
s2	$s_2$	PITrackAW	PI controller for eHL output
s3	$s_3$	LeadLag	
s4	$s_4$	Lag	Pline filter
fdbd	$f^{dbd}$	DeadBand1	frequency error deadband
s5	$s_5$	PITrackAW	PI for fe limiter output
s6	$s_6$	Lag	Output filter for Pext

Config Fields in [REPCA1]

Option	Symbol	Value	Info	Accepted values
kqs	$K_{qs}$	2	Tracking gain for reactive power PI controller	
ksg	$K_{sg}$	2	Tracking gain for active power PI controller	
freeze	$f_{rz}$	1	Voltage dip freeze flag; 1-enable, 0-disable	

## 5.20 RenTorque

Renewable torque (Pref) controller.

Common Parameters: u, name

Available models: *WTTQA1*

### 5.20.1 WTTQA1

Group *RenTorque*

Wind turbine generator torque (Pref) model.

PI state freeze following voltage dip has not been implemented.

Resets  $wg$  in *REECA1* model to 1.0 when torque model is connected. This effectively ignores *PFLAG* of *REECA1*.

Parameters

Name	Symbol	Description	Default	Unit	Type	Properties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
rep		RenPitch controller idx			IdxParam	mandatory
Kip	$K_{ip}$	Pref-control integral gain	0.100	<i>p.u.</i>	NumParam	
Kpp	$K_{pp}$	Pref-control proportional gain	0	<i>p.u.</i>	NumParam	
Tp	$T_p$	Pe sensing time const.	0.050	<i>s</i>	NumParam	
Twref	$T_{wref}$	Speed reference time const.	30	<i>s</i>	NumParam	
Temax	$T_{emax}$	Max. electric torque	1.200	<i>p.u.</i>	NumParam	power
Temin	$T_{emin}$	Min. electric torque	0	<i>p.u.</i>	NumParam	power
Tflag		Tflag; 1-power error, 0-speed error		<i>bool</i>	NumParam	mandatory
p1	$p_1$	Active power point 1	0.200	<i>p.u.</i>	NumParam	power
sp1	$s_{p1}$	Speed power point 1	0.580	<i>p.u.</i>	NumParam	
p2	$p_2$	Active power point 2	0.400	<i>p.u.</i>	NumParam	power
sp2	$s_{p2}$	Speed power point 2	0.720	<i>p.u.</i>	NumParam	
p3	$p_3$	Active power point 3	0.600	<i>p.u.</i>	NumParam	power
sp3	$s_{p3}$	Speed power point 3	0.860	<i>p.u.</i>	NumParam	
p4	$p_4$	Active power point 4	0.800	<i>p.u.</i>	NumParam	power
sp4	$s_{p4}$	Speed power point 4	1	<i>p.u.</i>	NumParam	
rea			0		ExtParam	
rego			0		ExtParam	
ree			0		ExtParam	
reg			0		ExtParam	
Sn	$S_n$		0		ExtParam	
w0	$\omega_0$		0		ExtParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
s1_y	$y_{s1}$	$1.0P_e$	State in lag transfer function		v_str
s2_y	$y_{s2}$	$1.0y_{fPe}$	State in lag transfer function		v_str
PI_xi	$xi_{PI}$	$\frac{P_{ref0}}{y_{fPe}}$	Integrator output		v_str
wg	$\omega_g$	$y_{fPe}$			v_str,v_setter
wt	$\omega_t$	$y_{fPe}$			v_str,v_setter
s3_y	$y_{s3}$	$\frac{P_{ref0}}{K_{shaft}\omega_g}$			v_str,v_setter
fPe_y	$y_{fPe}$	$\begin{cases} s_{p1} & \text{for } p_1 \geq y_{s1} \\ k_{p1}(-p_1 + y_{s1}) + s_{p1} & \text{for } p_2 \geq y_{s1} \\ k_{p2}(-p_2 + y_{s1}) + s_{p2} & \text{for } p_3 \geq y_{s1} \\ k_{p3}(-p_3 + y_{s1}) + s_{p3} & \text{for } p_4 \geq y_{s1} \\ s_{p4} & \text{otherwise} \end{cases}$	Output of piecewise		v_str
Tsel	$T_{sel}$	$\frac{SWT_{s0}(-\omega_g + y_{s2})}{\omega_g} + \frac{SWT_{s1}(P_e - P_{ref0})}{y_{fPe}}$	Output after Tflag selector		v_str
PI_yul	$y_{PI}^{ul}$	$K_{pp}T_{sel} + \frac{P_{ref0}}{y_{fPe}}$			v_str
PI_y	$y_{PI}$	$\pi_{hlzi}y_{PI}^{ul} + \pi_{hlzl}T_{emin} + \pi_{hlzu}T_{emax}$	PI output		v_str
Pe	$P_e$				
wr0	$\omega_{r0}$	$y_{fPe}$	Retrieved initial w0 from RenGovernor		v_str,v_setter
wge	$\omega_{ge}$	1.0			v_str,v_setter
Pref	$P_{ref}$	$\omega_g y_{PI}$			v_str,v_setter

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
s1_y	$y_{s1}$	State	$1.0P_e - y_{s1}$	$T_p$
s2_y	$y_{s2}$	State	$1.0y_{fPe} - y_{s2}$	$T_{wref}$
PI_xi	$xi_{PI}$	State	$K_{ip}T_{sel}$	
wg	$\omega_g$	ExtState	0	
wt	$\omega_t$	ExtState	0	
s3_y	$y_{s3}$	ExtState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
fPe_y	$y_{fPe}$	Algeb	$-y_{fPe} + \begin{cases} s_{p1} & \text{for } p_1 \geq y_{s1} \\ k_{p1}(-p_1 + y_{s1}) + s_{p1} & \text{for } p_2 \geq y_{s1} \\ k_{p2}(-p_2 + y_{s1}) + s_{p2} & \text{for } p_3 \geq y_{s1} \\ k_{p3}(-p_3 + y_{s1}) + s_{p3} & \text{for } p_4 \geq y_{s1} \\ s_{p4} & \text{otherwise} \end{cases}$
Tsel	$T_{sel}$	Algeb	$SWT_{s0}(-\omega_g + y_{s2}) + \frac{SWT_{s1}(P_e - P_{ref0})}{\omega_g} - T_{sel}$
PI_yul	$y_{PI}^{ul}$	Algeb	$K_{pp}T_{sel} + xi_{PI} - y_{PI}^{ul}$
PI_y	$y_{PI}$	Algeb	$\pi_{hlzi}y_{PI}^{ul} + \pi_{hlzl}T_{emin} + \pi_{hlzu}T_{emax} - y_{PI}$
Pe	$P_e$	ExtAlgeb	0
wr0	$\omega_{r0}$	ExtAlgeb	$-\omega_0 + y_{fPe}$
wge	$\omega_{ge}$	ExtAlgeb	$1 - y_{fPe}$
Pref	$P_{ref}$	ExtAlgeb	$-\frac{P_{ref0}}{\omega_{ge}} + \omega_g y_{PI}$

### Services

Name	Symbol	Equation	Type
kp1	$k_{p1}$	$\frac{-s_{p1} + s_{p2}}{-p_1 + p_2}$	ConstService
kp2	$k_{p2}$	$\frac{-s_{p2} + s_{p3}}{-p_2 + p_3}$	ConstService
kp3	$k_{p3}$	$\frac{-s_{p3} + s_{p4}}{-p_3 + p_4}$	ConstService

### Discrete

Name	Symbol	Type	Info
SWT	$SW_T$	Switcher	
PI_aw	$aw_{PI}$	AntiWindup	
PI_hl	$hl_{PI}$	HardLimiter	

### Blocks

Name	Symbol	Type	Info
s1	$s_1$	Lag	Pe filter
fPe	$f_{Pe}$	Piecewise	Piecewise Pe to wref mapping
s2	$s_2$	Lag	speed filter
PI	$PI$	PIAWHardLimit	PI controller

## 5.21 StaticACDC

AC DC device for power flow

Common Parameters: u, name

Available models: *VSCShunt*

### **5.21.1 VSCShunt**

Group *StaticACDC*

Data for VSC Shunt in power flow Parameters



Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
bus		idx of connected bus			Idx- Param	manda- tory
node1		Node 1 index			Idx- Param	manda- tory
node2		Node 2 index			Idx- Param	manda- tory
Vn	$V_n$	AC voltage rating	110		NumParam	non_zero
Vdcn1	$V_{dcn1}$	DC voltage rating on node 1	100	<i>kV</i>	NumParam	non_zero
Vdcn2	$V_{dcn2}$	DC voltage rating on node 2	100	<i>kV</i>	NumParam	non_zero
Idcn	$I_{dcn}$	DC current rating	1	<i>kA</i>	NumParam	non_zero
rsh	$r_{sh}$	AC interface resistance	0.003	<i>ohm</i>	NumParam	z
xsh	$x_{sh}$	AC interface reactance	0.060	<i>ohm</i>	NumParam	z
con- trol		Control method: 0-PQ, 1-PV, 2-vQ or 3-vV			NumParam	manda- tory
v0		AC voltage setting (PV or vV) or initial guess (PQ or vQ)	1		NumParam	
p0		AC active power setting	0	<i>pu</i>	NumParam	
q0		AC reactive power setting	0	<i>pu</i>	NumParam	
vdc0	$v_{dc0}$	DC voltage setting	1	<i>pu</i>	NumParam	
k0		Loss coefficient - constant	0		NumParam	
k1		Loss coefficient - linear	0		NumParam	
k2		Loss coefficient - quadratic	0		NumParam	
droop		Enable dc voltage droop control	0	<i>boolean</i>	NumParam	
K		Droop coefficient	0		NumParam	
vhigh		Upper voltage threshold in droop control	9999	<i>pu</i>	NumParam	
vlow		Lower voltage threshold in droop control	0	<i>pu</i>	NumParam	
vsh- max		Maximum ac interface voltage	1.100	<i>pu</i>	NumParam	
vsh- min		Minimum ac interface voltage	0.900	<i>pu</i>	NumParam	
Ish- max		Maximum ac current	2	<i>pu</i>	NumParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
ash	$\theta_{sh}$	$a$	voltage phase behind the transformer	$rad$	v_str
vsh	$V_{sh}$	$v_0$	voltage magnitude behind transformer	$p.u.$	v_str
psh	$P_{sh}$	$p_0 (s_0^{mode} + s_1^{mode})$	active power injection into VSC	$p.u.$	v_str
qsh	$Q_{sh}$	$q_0 (s_0^{mode} + s_2^{mode})$	reactive power injection into VSC		v_str
pdc	$P_{dc}$	0	DC power injection		v_str
a	$a$		AC bus voltage phase		
v	$v$		AC bus voltage magnitude		
v1	$v_1$		DC node 1 voltage		
v2	$v_2$		DC node 2 voltage		

## Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
ash	$\theta_{sh}$	Algeb	$-P_{sh} + u (V_{sh} b_{sh} v \sin(\theta_{sh} - a) - V_{sh} g_{sh} v \cos(\theta_{sh} - a) + g_{sh} v^2)$
vsh	$V_{sh}$	Algeb	$-Q_{sh} + u (V_{sh} b_{sh} v \cos(\theta_{sh} - a) + V_{sh} g_{sh} v \sin(\theta_{sh} - a) - b_{sh} v^2)$
psh	$P_{sh}$	Algeb	$u (-P_{sh} + p_0) (s_0^{mode} + s_1^{mode}) + u (s_2^{mode} + s_3^{mode}) (v_1 - v_2 - v_{dc0})$
qsh	$Q_{sh}$	Algeb	$u (-Q_{sh} + q_0) (s_0^{mode} + s_2^{mode}) + u (s_1^{mode} + s_3^{mode}) (-v + v_0)$
pdc	$P_{dc}$	Algeb	$P_{dc} + u (V_{sh}^2 g_{sh} - V_{sh} b_{sh} v \sin(\theta_{sh} - a) - V_{sh} g_{sh} v \cos(\theta_{sh} - a))$
a	$a$	ExtAl- geb	$-P_{sh}$
v	$v$	ExtAl- geb	$-Q_{sh}$
v1	$v_1$	ExtAl- geb	$-\frac{P_{dc}}{v_1 - v_2}$
v2	$v_2$	ExtAl- geb	$\frac{P_{dc}}{v_1 - v_2}$

## Services

Name	Symbol	Equation	Type
gsh	$g_{sh}$	$\frac{\text{re}(r_{sh}) - \text{im}(x_{sh})}{(\text{re}(r_{sh}) - \text{im}(x_{sh}))^2 + (\text{re}(x_{sh}) + \text{im}(r_{sh}))^2}$	ConstService
bsh	$b_{sh}$	$\frac{-\text{re}(x_{sh}) - \text{im}(r_{sh})}{(\text{re}(r_{sh}) - \text{im}(x_{sh}))^2 + (\text{re}(x_{sh}) + \text{im}(r_{sh}))^2}$	ConstService

## Discrete

Name	Symbol	Type	Info
mode	$mode$	Switcher	

## 5.22 StaticGen

Static generator group for power flow calculation

Common Parameters:  $u$ , name,  $S_n$ ,  $V_n$ ,  $p_0$ ,  $q_0$ ,  $r_a$ ,  $x_s$ , subidx

Common Variables:  $p$ ,  $q$ ,  $a$ ,  $v$

Available models: *PV*, *Slack*

### 5.22.1 PV

Group *StaticGen*

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
$u$	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
$S_n$	$S_n$	Power rating	100		NumParam	non_zero
$V_n$	$V_n$	AC voltage rating	110		NumParam	non_zero
subidx		index for generators on the same bus			Data- Param	
bus		idx of the installed bus			IdxParam	
busr		bus idx for remote voltage control			IdxParam	
$p_0$	$p_0$	active power set point in system base	0	<i>p.u.</i>	NumParam	
$q_0$	$q_0$	reactive power set point in system base	0	<i>p.u.</i>	NumParam	
$p_{max}$	$p_{max}$	maximum active power in system base	999	<i>p.u.</i>	NumParam	
$p_{min}$	$p_{min}$	minimum active power in system base	-1	<i>p.u.</i>	NumParam	
$q_{max}$	$q_{max}$	maximum reactive power in system base	999	<i>p.u.</i>	NumParam	
$q_{min}$	$q_{min}$	minimum reactive power in system base	-999	<i>p.u.</i>	NumParam	
$v_0$	$v_0$	voltage set point	1		NumParam	
$v_{max}$	$v_{max}$	maximum voltage	1.400		NumParam	
$v_{min}$	$v_{min}$	minimum allowed voltage	0.600		NumParam	
$r_a$	$r_a$	armature resistance	0.010		NumParam	
$x_s$	$x_s$	armature reactance	0.300		NumParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
$p$	$p$	$p_0$	actual active power generation	<i>p.u.</i>	v_str
$q$	$q$	$q_0$	actual reactive power generation	<i>p.u.</i>	v_str
$a$	$\theta$				
$v$	$V$	$v_0$			v_str, v_setter

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
p	$p$	Algeb	$u(-p + p_0)$
q	$q$	Algeb	$u\left(z_i^{qlim}(-V + v_0) + z_l^{qlim}(-q + q_{min}) + z_u^{qlim}(-q + q_{max})\right)$
a	$\theta$	ExtAlgeb	$-pu$
v	$V$	ExtAlgeb	$-qu$

## Discrete

Name	Symbol	Type	Info
qlim	$qlim$	SortedLimiter	

## Config Fields in [PV]

Option	Symbol	Value	Info	Accepted values
pv2pq	$z_{pv2pq}$	0	convert PV to PQ in PFlow at Q limits	(0, 1)
npv2pq	$n_{pv2pq}$	1	max. # of pv2pq conversion in each iteration	$\geq 0$

## 5.22.2 Slack

Group *StaticGen*

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
Sn	$S_n$	Power rating	100		NumParam	non_zero
Vn	$V_n$	AC voltage rating	110		NumParam	non_zero
subidx		index for generators on the same bus			Data- Param	
bus		idx of the installed bus			IdxParam	
busr		bus idx for remote voltage control			IdxParam	
p0	$p_0$	active power set point in system base	0	<i>p.u.</i>	NumParam	
q0	$q_0$	reactive power set point in system base	0	<i>p.u.</i>	NumParam	
pmax	$p_{max}$	maximum active power in system base	999	<i>p.u.</i>	NumParam	
pmin	$p_{min}$	minimum active power in system base	-1	<i>p.u.</i>	NumParam	
qmax	$q_{max}$	maximum reactive power in system base	999	<i>p.u.</i>	NumParam	
qmin	$q_{min}$	minimum reactive power in system base	-999	<i>p.u.</i>	NumParam	
v0	$v_0$	voltage set point	1		NumParam	
vmax	$v_{max}$	maximum voltage	1.400		NumParam	
vmin	$v_{min}$	minimum allowed voltage	0.600		NumParam	
ra	$r_a$	armature resistance	0.010		NumParam	
xs	$x_s$	armature reactance	0.300		NumParam	
a0	$\theta_0$	reference angle set point	0		NumParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
p	$p$	$p_0$	actual active power generation	<i>p.u.</i>	v_str
q	$q$	$q_0$	actual reactive power generation	<i>p.u.</i>	v_str
a	$\theta$	$\theta_0$			v_str, v_setter
v	$V$	$v_0$			v_str, v_setter

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
p	$p$	Algeb	$u \left( z_i^{plim} (-\theta + \theta_0) + z_l^{plim} (-p + p_{min}) + z_u^{plim} (-p + p_{max}) \right)$
q	$q$	Algeb	$u \left( z_i^{qlim} (-V + v_0) + z_l^{qlim} (-q + q_{min}) + z_u^{qlim} (-q + q_{max}) \right)$
a	$\theta$	ExtAlgeb	$-pu$
v	$V$	ExtAlgeb	$-qu$

Discrete

Name	Symbol	Type	Info
qlim	<i>qlim</i>	SortedLimiter	
plim	<i>plim</i>	SortedLimiter	

Config Fields in [Slack]

Option	Symbol	Value	Info	Accepted values
pv2pq	<i>z<sub>pv2pq</sub></i>	0	convert PV to PQ in PFlow at Q limits	(0, 1)
npv2pq	<i>n<sub>pv2pq</sub></i>	1	max. # of pv2pq conversion in each iteration	>=0
av2pv	<i>z<sub>av2pv</sub></i>	0	convert Slack to PV in PFlow at P limits	(0, 1)

## 5.23 StaticLoad

Static load group.

Common Parameters: u, name

Available models: *PQ*

### 5.23.1 PQ

Group *StaticLoad*

PQ load model.

Implements an automatic pq2z conversion during power flow when the voltage is outside [vmin, vmax]. The conversion can be turned off by setting *pq2z* to 0 in the Config file.

Before time-domain simulation, PQ load will be converted to impedance, current source, and power source based on the weights in the Config file.

Weights (p2p, p2i, p2z) corresponds to the weights for constant power, constant current and constant impedance. p2p, p2i and p2z must be in decimal numbers and sum up exactly to 1. The same rule applies to (q2q, q2i, q2z).

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
bus		linked bus idx			IdxParam	mandatory
Vn	$V_n$	AC voltage rating	110	<i>kV</i>	NumParam	non_zero
p0	$p_0$	active power load in system base	0	<i>p.u.</i>	NumParam	
q0	$q_0$	reactive power load in system base	0	<i>p.u.</i>	NumParam	
vmax	$v_{max}$	max voltage before switching to impedance	1.200		NumParam	
vmin	$v_{min}$	min voltage before switching to impedance	0.800		NumParam	
owner		owner idx			IdxParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
a	$\theta$				
v	$V$				

## Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
a	$\theta$	ExtAl- geb	$u \left( I_{peq} V \gamma_{p2i} + P_{pf} \gamma_{p2p} + R_{eq} V^2 \gamma_{p2z} \right) (t_{dae} > 0) +$ $u \left( R_{lb} V^2 z_l^{vcmp} + R_{ub} V^2 z_u^{vcmp} + p_0 z_i^{vcmp} \right) (t_{dae} \leq 0)$
v	$V$	ExtAl- geb	$u \left( I_{qeq} V \gamma_{q2i} + Q_{pf} \gamma_{q2q} + V^2 X_{eq} \gamma_{q2z} \right) (t_{dae} > 0) +$ $u \left( V^2 X_{lb} z_l^{vcmp} + V^2 X_{ub} z_u^{vcmp} + q_0 z_i^{vcmp} \right) (t_{dae} \leq 0)$

## Services

Name	Symbol	Equation	Type
Rub	$R_{ub}$	$\frac{p_0}{v_{max}^2}$	ConstService
Xub	$X_{ub}$	$\frac{q_0}{v_{max}^2}$	ConstService
Rlb	$R_{lb}$	$\frac{p_0}{v_{min}^2}$	ConstService
Xlb	$X_{lb}$	$\frac{q_0}{v_{min}^2}$	ConstService
Ppf	$P_{pf}$	$R_{lb}V_0^2 z_l^{vcmp} + R_{ub}V_0^2 z_u^{vcmp} + p_0 z_i^{vcmp}$	ConstService
Qpf	$Q_{pf}$	$V_0^2 X_{lb} z_l^{vcmp} + V_0^2 X_{ub} z_u^{vcmp} + q_0 z_i^{vcmp}$	ConstService
Req	$R_{eq}$	$\frac{P_{pf}}{V_0^2}$	ConstService
Xeq	$X_{eq}$	$\frac{Q_{pf}}{V_0^2}$	ConstService
Ipeq	$I_{peq}$	$\frac{P_{pf}}{V_0}$	ConstService
Iqeq	$I_{qeq}$	$\frac{Q_{pf}}{V_0}$	ConstService

Discrete

Name	Symbol	Type	Info
vcmp	$vcmp$	Limiter	

Config Fields in [PQ]

Op- tion	Sym- bol	Value	Info	Accepted val- ues
pq2z	$z_{pq2z}$	1	pq2z conversion if out of voltage limits	(0, 1)
p2p	$\gamma_{p2p}$	0	P constant power percentage for TDS. Must have (p2p+p2i+p2z)=1	float
p2i	$\gamma_{p2i}$	0	P constant current percentage	float
p2z	$\gamma_{p2z}$	1	P constant impedance percentage	float
q2q	$\gamma_{q2q}$	0	Q constant power percentage for TDS. Must have (q2q+q2i+q2z)=1	float
q2i	$\gamma_{q2i}$	0	Q constant current percentage	float
q2z	$\gamma_{q2z}$	1	Q constant impedance percentage	float

## 5.24 StaticShunt

Static shunt compensator group.

Common Parameters: u, name

Available models: *Shunt*

### 5.24.1 Shunt

Group *StaticShunt*



## Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
bus		idx of connected bus			IdxParam	mandatory
Sn	$S_n$	Power rating	100		NumParam	non_zero
Vn	$V_n$	AC voltage rating	110		NumParam	non_zero
g	$g$	shunt conductance (real part)	0		NumParam	y
b	$b$	shunt susceptance (positive as cap- atance)	0		NumParam	y
fn	$f$	rated frequency	60		NumParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
a	$\theta$				
v	$V$				

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
a	$\theta$	ExtAlgeb	$V^2 g u$
v	$V$	ExtAlgeb	$-V^2 b u$

## 5.25 SynGen

Synchronous generator group.

Common Parameters: u, name, Sn, Vn, fn, bus, M, D

Common Variables: omega, delta, tm, te, vf, XadIfd, vd, vq, Id, Iq, a, v

Available models: *GENCLS*, *GENROU*

### 5.25.1 GENCLS

Group *SynGen*

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Properties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
bus		interface bus id			Idx- Param	mandatory
gen		static generator index			Idx- Param	mandatory
coi		center of inertia index			Idx- Param	
Sn	$S_n$	Power rating	100		NumParam	
Vn	$V_n$	AC voltage rating	110		NumParam	
fn	$f$	rated frequency	60		NumParam	
D	$D$	Damping coefficient	0		NumParam	power
M	$M$	machine start up time (2H)	6		NumParam	non_zero,power
ra	$r_a$	armature resistance	0		NumParam	z
xl	$x_l$	leakage reactance	0		NumParam	z
xd1	$x'_d$	d-axis transient reactance	0.302		NumParam	z
kp	$k_p$	active power feedback gain	0		NumParam	
kw	$k_w$	speed feedback gain	0		NumParam	
S10	$S_{1.0}$	first saturation factor	0		NumParam	
S12	$S_{1.2}$	second saturation factor	1		NumParam	
subidx		Generator idx in plant; only used by PSS/E data	0		ExtParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
delta	$\delta$	$\delta_0$	rotor angle	<i>rad</i>	v_str
omega	$\omega$	$u$	rotor speed	<i>pu (Hz)</i>	v_str
Id	$I_d$	$I_{d0}u$	d-axis current		v_str
Iq	$I_q$	$I_{q0}u$	q-axis current		v_str
vd	$V_d$	$V_{d0}u$	d-axis voltage		v_str
vq	$V_q$	$V_{q0}u$	q-axis voltage		v_str
tm	$\tau_m$	$\tau_{m0}$	mechanical torque		v_str
te	$\tau_e$	$P_0u$	electric torque		v_str
vf	$v_f$	$uvf_0$	excitation voltage	<i>pu</i>	v_str
XadIfd	$X_{ad}I_{fd}$	$uvf_0$	d-axis armature excitation current	<i>p.u (kV)</i>	v_str
psid	$\psi_d$	$\psi_{d0}u$	d-axis flux		v_str
psiq	$\psi_q$	$\psi_{q0}u$	q-axis flux		v_str
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
delta	$\delta$	State	$2\pi f u (\omega - 1)$	
omega	$\omega$	State	$\frac{u(-D(\omega-1)-\tau_e+\tau_m)}{M}$	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
Id	$I_d$	Algeb	$I_d x q + \psi_d - v_f$
Iq	$I_q$	Algeb	$I_q x q + \psi_q$
vd	$V_d$	Algeb	$V u \sin(\delta - \theta) - V_d$
vq	$V_q$	Algeb	$V u \cos(\delta - \theta) - V_q$
tm	$\tau_m$	Algeb	$-\tau_m + \tau_{m0}$
te	$\tau_e$	Algeb	$-\tau_e + u(-I_d \psi_q + I_q \psi_d)$
vf	$v_f$	Algeb	$u v_{f0} - v_f$
XadIfd	$X_{ad} I_{fd}$	Algeb	$-X_{ad} I_{fd} + u v_{f0}$
psid	$\psi_d$	Algeb	$-\psi_d + u(I_q r_a + V_q)$
psiq	$\psi_q$	Algeb	$\psi_q + u(I_d r_a + V_d)$
a	$\theta$	ExtAlgeb	$-u(I_d V_d + I_q V_q)$
v	$V$	ExtAlgeb	$-u(I_d V_q - I_q V_d)$

## Services

Name	Symbol	Equation	Type
_V	$V_c$	$V e^{i\theta}$	ConstService
_S	$S$	$P_0 - iQ_0$	ConstService
_I	$I_c$	$\frac{S}{\text{conj}(V_c)}$	ConstService
_E	$E$	$I_c(r_a + i x q) + V_c$	ConstService
_deltac	$\delta_c$	$\log\left(\frac{E}{\text{abs}(E)}\right)$	ConstService
delta0	$\delta_0$	$u \text{im}(\delta_c)$	ConstService
vdq	$V_{dq}$	$V_c u e^{-\delta_c + 0.5i\pi}$	ConstService
Idq	$I_{dq}$	$I_c u e^{-\delta_c + 0.5i\pi}$	ConstService
Id0	$I_{d0}$	$\text{re}(I_{dq})$	ConstService
Iq0	$I_{q0}$	$\text{im}(I_{dq})$	ConstService
vd0	$V_{d0}$	$\text{re}(V_{dq})$	ConstService
vq0	$V_{q0}$	$\text{im}(V_{dq})$	ConstService
tm0	$\tau_{m0}$	$u(I_{d0}(I_{d0} r_a + V_{d0}) + I_{q0}(I_{q0} r_a + V_{q0}))$	ConstService
psid0	$\psi_{d0}$	$I_{q0} r_a u + V_{q0}$	ConstService
psiq0	$\psi_{q0}$	$-I_{d0} r_a u - V_{d0}$	ConstService
vf0	$v_{f0}$	$I_{d0} x q + I_{q0} r_a + V_{q0}$	ConstService

## Config Fields in [GENCLS]

Option	Symbol	Value	Info	Accepted values
vf_lower		1	lower limit for vf warning	
vf_upper		5	upper limit for vf warning	

### 5.25.2 GENROU

Group *SynGen*

Round rotor generator with quadratic saturation

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Properties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
bus		interface bus id			Idx- Param	mandatory
gen		static generator index			Idx- Param	mandatory
coi		center of inertia index			Idx- Param	
Sn	$S_n$	Power rating	100		NumParam	
Vn	$V_n$	AC voltage rating	110		NumParam	
fn	$f$	rated frequency	60		NumParam	
D	$D$	Damping coefficient	0		NumParam	power
M	$M$	machine start up time (2H)	6		NumParam	non_zero,power
ra	$r_a$	armature resistance	0		NumParam	z
xl	$x_l$	leakage reactance	0		NumParam	z
xd1	$x'_d$	d-axis transient reactance	0.302		NumParam	z
kp	$k_p$	active power feedback gain	0		NumParam	
kw	$k_w$	speed feedback gain	0		NumParam	
S10	$S_{1.0}$	first saturation factor	0		NumParam	
S12	$S_{1.2}$	second saturation factor	1		NumParam	
xd	$x_d$	d-axis synchronous reactance	1.900		NumParam	z
xq	$x_q$	q-axis synchronous reactance	1.700		NumParam	z
xd2	$x''_d$	d-axis sub-transient reactance	0.204		NumParam	z
xq1	$x'_q$	q-axis transient reactance	0.500		NumParam	z
xq2	$x''_q$	q-axis sub-transient reactance	0.300		NumParam	z
Td10	$T'_{d0}$	d-axis transient time constant	8		NumParam	
Td20	$T''_{d0}$	d-axis sub-transient time constant	0.040		NumParam	
Tq10	$T'_{q0}$	q-axis transient time constant	0.800		NumParam	
Tq20	$T''_{q0}$	q-axis sub-transient time constant	0.020		NumParam	
subidx		Generator idx in plant; only used by PSS/E data	0		ExtParam	

Variables

Name	Symbol	Initial Value	Description	Unit	Properties
delta	$\delta$	$\delta_0$	rotor angle	<i>rad</i>	v_str
omega	$\omega$	$u$	rotor speed	<i>pu (Hz)</i>	v_str
e1q	$e'_q$	$e'_{q0}u$	q-axis transient voltage		v_str
e1d	$e'_d$	$e'_{d0}u$	d-axis transient voltage		v_str
e2d	$e''_d$	$e''_{d0}u$	d-axis sub-transient voltage		v_str
e2q	$e''_q$	$e''_{q0}u$	q-axis sub-transient voltage		v_str
Id	$I_d$	$I_{d0}u$	d-axis current		v_str
Iq	$I_q$	$I_{q0}u$	q-axis current		v_str
vd	$V_d$	$V_{d0}u$	d-axis voltage		v_str
vq	$V_q$	$V_{q0}u$	q-axis voltage		v_str
tm	$\tau_m$	$\tau_{m0}$	mechanical torque		v_str
te	$\tau_e$	$P_0u$	electric torque		v_str
vf	$v_f$	$uvf_0$	excitation voltage	<i>pu</i>	v_str
XadIfd	$X_{ad}I_{fd}$	$uvf_0$	d-axis armature excitation current	<i>p.u (kV)</i>	v_str
psid	$\psi_d$	$\psi_{d0}u$	d-axis flux		v_str
psiq	$\psi_q$	$\psi_{q0}u$	q-axis flux		v_str
psi2q	$\psi_{aq}$	$\psi_{aq0}$	q-axis air gap flux		v_str
psi2d	$\psi_{ad}$	$\psi_{ad0}u$	d-axis air gap flux		v_str
psi2	$\psi_a$	$u \text{ abs } (\psi''_{0,dq})$	air gap flux magnitude		v_str
Se	$S_e( \psi_a )$	$S_{e0}u$	saturation output		v_str
XaqI1q	$X_{aq}I_{1q}$	0	q-axis reaction	<i>p.u (kV)</i>	v_str
a	$\theta$		Bus voltage phase angle		
v	$V$		Bus voltage magnitude		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
delta	$\delta$	State	$2\pi f u (\omega - 1)$	
omega	$\omega$	State	$\frac{u(-D(\omega-1)-\tau_e+\tau_m)}{M}$	
e1q	$e'_q$	State	$-X_{ad}I_{fd} + v_f$	$T'_{d0}$
e1d	$e'_d$	State	$-X_{aq}I_{1q}$	$T'_{q0}$
e2d	$e''_d$	State	$-I_d(x'_d - x_l) - e''_d + e'_q$	$T''_{d0}$
e2q	$e''_q$	State	$I_q(x'_q - x_l) - e''_q + e'_d$	$T''_{q0}$

## Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
Id	$I_d$	Algeb	$I_d x_d'' + \psi_d - \psi_{ad}$
Iq	$I_q$	Algeb	$I_q x_q'' + \psi_q + \psi_{aq}$
vd	$V_d$	Algeb	$V u \sin(\delta - \theta) - V_d$
vq	$V_q$	Algeb	$V u \cos(\delta - \theta) - V_q$
tm	$\tau_m$	Algeb	$-\tau_m + \tau_{m0}$
te	$\tau_e$	Algeb	$-\tau_e + u(-I_d \psi_q + I_q \psi_d)$
vf	$v_f$	Algeb	$uv f_0 - v_f$
XadIfd	$X_{ad} I_{fd}$	Algeb	$-X_{ad} I_{fd} + u(S_e( \psi_a )\psi_{ad} + e_q' + (-x_d' + x_d)(I_d \gamma_{d1} - \gamma_{d2} e_d'' + \gamma_{d2} e_q'))$
psid	$\psi_d$	Algeb	$-\psi_d + u(I_q r_a + V_q)$
psiq	$\psi_q$	Algeb	$\psi_q + u(I_d r_a + V_d)$
psi2q	$\psi_{aq}$	Algeb	$\gamma_{q1} e_d' - \psi_{aq} + e_q''(1 - \gamma_{q1})$
psi2d	$\psi_{ad}$	Algeb	$\gamma_{d1} e_q' + \gamma_{d2} e_d''(x_d' - x_l) - \psi_{ad}$
psi2	$\psi_a$	Algeb	$-\psi_a^2 + \psi_{ad}^2 + \psi_{aq}^2$
Se	$S_e( \psi_a )$	Algeb	$B_{SAT}^q z_0^{SL} \left(-A_{SAT}^q + \psi_a\right)^2 - S_e( \psi_a )\psi_a$
XaqI1q	$X_{aq} I_{1q}$	Algeb	$S_e( \psi_a )\gamma_{qd}\psi_{aq} - X_{aq} I_{1q} + e_d' + (-x_q' + x_q)(-I_q \gamma_{q1} - \gamma_{q2} e_q'' + \gamma_{q2} e_d')$
a	$\theta$	ExtAl- geb	$-u(I_d V_d + I_q V_q)$
v	$V$	ExtAl- geb	$-u(I_d V_q - I_q V_d)$

## Services

Name	Symbol	Equation	Type
gd1	$\gamma_{d1}$	$\frac{x_d'' - x_l}{x_d' - x_l}$	ConstService
gq1	$\gamma_{q1}$	$\frac{x_q'' - x_l}{x_q' - x_l}$	ConstService
gd2	$\gamma_{d2}$	$\frac{-x_d'' + x_d'}{(x_d' - x_l)^2}$	ConstService
gq2	$\gamma_{q2}$	$\frac{-x_q'' + x_q'}{(x_q' - x_l)^2}$	ConstService
gqd	$\gamma_{qd}$	$\frac{-x_l + x_q}{x_d - x_l}$	ConstService
_S12	$S_{1.2}$	$S_{1.2} - f S_{12} + 1$	ConstService
SAT_E1	$E_{SAT}^{1c}$	1.0	ConstService
SAT_E2	$E_{SAT}^{2c}$	1.2	ConstService
SAT_SE1	$SE_{SAT}^{1c}$	$S_{1.0}$	ConstService
SAT_SE2	$SE_{SAT}^{2c}$	$S_{1.2} - 2z_{SAT}^{SE2} + 2$	ConstService
SAT_a	$a_{SAT}$	$\sqrt{\frac{E_{SAT}^{1c} SE_{SAT}^{1c}}{E_{SAT}^{2c} SE_{SAT}^{2c}}} \left( (SE_{SAT}^{2c} > 0) + (SE_{SAT}^{2c} < 0) \right)$	ConstService
SAT_A	$A_{SAT}^q$	$E_{SAT}^{2c} - \frac{E_{SAT}^{1c} - E_{SAT}^{2c}}{a_{SAT} - 1}$	ConstService
SAT_B	$B_{SAT}^q$	$\frac{E_{SAT}^{2c} SE_{SAT}^{2c} (a_{SAT} - 1)^2 ((a_{SAT} > 0) + (a_{SAT} < 0))}{(E_{SAT}^{1c} - E_{SAT}^{2c})^2}$	ConstService

Continued on next page

Table 6 – continued from previous page

Name	Symbol	Equation	Type
_V	$V_c$	$V e^{i\theta}$	ConstService
_S	$S$	$P_0 - iQ_0$	ConstService
_Zs	$Z_s$	$r_a + i x_d''$	ConstService
_It	$I_t$	$\frac{S}{\text{conj}(V_c)}$	ConstService
_Is	$I_s$	$I_t + \frac{V_c}{Z_s}$	ConstService
psi20	$\psi_0''$	$I_s Z_s$	ConstService
psi20_arg	$\theta_{\psi''0}$	$\arg(\psi_0'')$	ConstService
psi20_abs	$ \psi_0'' $	$\text{abs}(\psi_0'')$	ConstService
_It_arg	$\theta_{It0}$	$\arg(I_t)$	ConstService
_psi20_It_arg	$\theta_{\psi a I t}$	$-\theta_{It0} + \theta_{\psi''0}$	ConstService
Se0	$S_{e0}$	$\frac{B_{SAT}^q (-A_{SAT}^q +  \psi_0'' )^2 ( \psi_0''  \geq A_{SAT}^q)}{ \psi_0'' }$	ConstService
_a	$a'$	$ \psi_0''  (S_{e0} \gamma_{qd} + 1)$	ConstService
_b	$b'$	$(x_q'' - x_q) \text{abs}(I_t)$	ConstService
delta0	$\delta_0$	$\theta_{\psi''0} + \text{atan}\left(\frac{b' \cos(\theta_{\psi a I t})}{-a' + b' \sin(\theta_{\psi a I t})}\right)$	ConstService
_Tdq	$T_{dq}$	$-i \sin(\delta_0) + \cos(\delta_0)$	ConstService
psi20_dq	$\psi_{0,dq}''$	$T_{dq} \psi_0''$	ConstService
It_dq	$I_{t,dq}$	$\text{conj}(I_t T_{dq})$	ConstService
psi2d0	$\psi_{ad0}$	$\text{re}(\psi_{0,dq}'')$	ConstService
psi2q0	$\psi_{aq0}$	$-\text{im}(\psi_{0,dq}'')$	ConstService
Id0	$I_{d0}$	$\text{im}(I_{t,dq})$	ConstService
Iq0	$I_{q0}$	$\text{re}(I_{t,dq})$	ConstService
vd0	$V_{d0}$	$-I_{d0} r_a + I_{q0} x_q'' + \psi_{aq0}$	ConstService
vq0	$V_{q0}$	$-I_{d0} x_d'' - I_{q0} r_a + \psi_{ad0}$	ConstService
tm0	$\tau_{m0}$	$u(I_{d0}(I_{d0} r_a + V_{d0}) + I_{q0}(I_{q0} r_a + V_{q0}))$	ConstService
vf0	$v_{f0}$	$I_{d0}(-x_d'' + x_d) + \psi_{ad0}(S_{e0} + 1)$	ConstService
psid0	$\psi_{d0}$	$I_{q0} r_a u + V_{q0}$	ConstService
psiq0	$\psi_{q0}$	$-I_{d0} r_a u - V_{d0}$	ConstService
e1q0	$e'_{q0}$	$I_{d0}(x_d' - x_d) - S_{e0} \psi_{ad0} + v_{f0}$	ConstService
e1d0	$e'_{d0}$	$I_{q0}(-x_q' + x_q) - S_{e0} \gamma_{qd} \psi_{aq0}$	ConstService
e2d0	$e''_{d0}$	$I_{d0}(-x_d + x_l) - S_{e0} \psi_{ad0} + v_{f0}$	ConstService
e2q0	$e''_{q0}$	$-I_{q0}(x_l - x_q) - S_{e0} \gamma_{qd} \psi_{aq0}$	ConstService

Discrete

Name	Symbol	Type	Info
SL	$SL$	LessThan	

Blocks

Name	Symbol	Type	Info
SAT	$S_{AT}$	ExcQuadSat	



Config Fields in [GENROU]

Option	Symbol	Value	Info	Accepted values
vf_lower		1	lower limit for vf warning	
vf_upper		5	upper limit for vf warning	

## 5.26 TimedEvent

Timed event group

Common Parameters: u, name

Available models: *Toggler*, *Fault*

### 5.26.1 Toggler

Group *TimedEvent*

Time-based connectivity status toggler.

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			DataParam	
u	<i>u</i>	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
model		Model or Group of the device to control			DataParam	mandatory
dev		idx of the device to control			IdxParam	mandatory
t		switch time for connection status	-1		Timer-Param	mandatory

Services

Name	Symbol	Equation	Type
<u>u</u>	<i>u</i>	1	ConstService

### 5.26.2 Fault

Group *TimedEvent*

Three-phase to ground fault.

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Proper- ties
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
bus		linked bus idx			IdxParam	mandatory
tf		Bus fault start time	-1	<i>second</i>	Timer- Param	mandatory
tc		Bus fault end time	-1	<i>second</i>	Timer- Param	
xf	$x_f$	Fault to ground impedance (posi- tive)	0.000	<i>p.u.(sys)</i>	NumParam	
rf	$x_f$	Fault to ground resistance (posi- tive)	0	<i>p.u.(sys)</i>	NumParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
a	$\theta$		Bus voltage angle	<i>p.u.(kV)</i>	
v	$V$		Bus voltage magnitude	<i>p.u.(kV)</i>	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
a	$\theta$	ExtAlgeb	$V^2 g_f u u_f$
v	$V$	ExtAlgeb	$-V^2 b_f u u_f$

## Services

Name	Symbol	Equation	Type
gf	$g_f$	$\frac{\text{re}(x_f) - \text{im}(x_f)}{(\text{re}(x_f) - \text{im}(x_f))^2 + (\text{re}(x_f) + \text{im}(x_f))^2}$	ConstService
bf	$b_f$	$\frac{-\text{re}(x_f) - \text{im}(x_f)}{(\text{re}(x_f) - \text{im}(x_f))^2 + (\text{re}(x_f) + \text{im}(x_f))^2}$	ConstService
uf	$u_f$	0	ConstService

## Config Fields in [Fault]

Option	Symbol	Value	Info	Accepted values
restore		1	restore algebraic variables to pre-fault values	(0, 1)
scale		1	scaling factor of restored algebraic values	

## 5.27 TurbineGov

Turbine governor group for synchronous generator.

Common Parameters: u, name

Common Variables: pout

Available models: *TG2*, *TGOV1*, *TGOVIDB*, *IEEEG1*

### 5.27.1 TG2

Group *TurbineGov*

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Properties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
syn		Synchronous generator idx			Idx- Param	manda- tory,unique
Tn	$T_n$	Turbine power rating. Equal to Sn if not provided.		<i>MVA</i>	NumParam	
wref0	$\omega_{ref0}$	Base speed reference	1	<i>p.u.</i>	NumParam	
R	$R$	Speed regulation gain (mach. base de- fault)	0.050	<i>p.u.</i>	NumParam	ipower
pmax	$p_{max}$	Maximum power output	999	<i>p.u.</i>	NumParam	power
pmin	$p_{min}$	Minimum power output	0	<i>p.u.</i>	NumParam	power
dbl	$L_{db}$	Deadband lower limit	- 0.000	<i>p.u.</i>	NumParam	
dbu	$U_{db}$	Deadband upper limit	0.000	<i>p.u.</i>	NumParam	
dbc	$C_{db}$	Deadband neutral value	0	<i>p.u.</i>	NumParam	
T1	$T_1$	Transient gain time	0.200		NumParam	
T2	$T_2$	Governor time constant	10		NumParam	
Sg	$S_n$	Rated power from generator	0	<i>MVA</i>	ExtParam	
Vn	$V_n$	Rated voltage from generator	0	<i>kV</i>	ExtParam	

Variables

Name	Sym- bol	Initial Value	Description	Unit	Proper- ties
ll_x	$x'_{ll}$	$\omega_{dmG}$	State in lead-lag		v_str
omega	$\omega$		Generator speed	<i>p.u.</i>	
paux	$P_{aux}$	$P_{aux0}$	Auxiliary power input		v_str
pout	$P_{out}$	$\tau_{m0}u$	Turbine final output power		v_str
wref	$\omega_{ref}$	$\omega_{ref0}$	Speed reference variable		v_str
w_d	$\omega_{dev}$	0	Generator speed deviation before dead band (positive for under speed)		v_str
w_dm	$\omega_{dm}$	0	Measured speed deviation after dead band		v_str
w_dmG	$\omega_{dmG}$	0	Speed deviation after dead band after gain		v_str
ll_y	$y_{ll}$	$\omega_{dmG}$	Output of lead-lag		v_str
pnl	$P_{nl}$	$\tau_{m0}$	Power output before hard limiter		v_str
tm	$\tau_m$		Mechanical power interface to SynGen		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
ll_x	$x'_{ll}$	State	$\omega_{dmG} - x'_{ll}$	$T_2$
omega	$\omega$	ExtState	0	

## Algebraic Equations

Name	Symbol	Type	RHS of Equation "0 = g(x, y)"
paux	$P_{aux}$	Algeb	$P_{aux0} - P_{aux}$
pout	$P_{out}$	Algeb	$P_{nl}z_i^{plim} - P_{out} + p_{max}z_u^{plim} + p_{min}z_l^{plim}$
wref	$\omega_{ref}$	Algeb	$\omega_{ref0} - \omega_{ref}$
w_d	$\omega_{dev}$	Algeb	$-\omega_{dev} + u(-\omega + \omega_{ref})$
w_dm	$\omega_{dm}$	Algeb	$L_{db}z_{lr}^{wdb} + U_{db}z_{ur}^{wdb} + \omega_{dev}(1 - z_i^{wdb}) - \omega_{dm}$
w_dmG	$\omega_{dmG}$	Algeb	$G\omega_{dm} - \omega_{dmG}$
ll_y	$y_{ll}$	Algeb	$T_1(\omega_{dmG} - x'_{ll}) + T_2x'_{ll} - T_2y_{ll} + ll_{LT1z1}ll_{LT2z1}(-x'_{ll} + y_{ll})$
pnl	$P_{nl}$	Algeb	$-P_{nl} + \tau_{m0} + y_{ll}$
tm	$\tau_m$	ExtAlgeb	$u(P_{out} - \tau_{m0})$

## Services

Name	Symbol	Equation	Type
paux0	$P_{aux0}$	0	ConstService
gain	$G$	$\frac{u}{R}$	ConstService

## Discrete

Name	Symbol	Type	Info
w_db	$w_{db}$	DeadBandRT	
ll_LT1	$LT_{ll}$	LessThan	
ll_LT2	$LT_{ll}$	LessThan	
plim	$plim$	HardLimiter	

Blocks

Name	Symbol	Type	Info
ll	$ll$	LeadLag	

Config Fields in [TG2]

Option	Symbol	Value	Info	Accepted values
deadband	$z_{deadband}$	0	enable input dead band	(0, 1)
hardlimit	$z_{hardlimit}$	1	enable output hard limit	(0, 1)

## 5.27.2 TGOV1

Group *TurbineGov*

TGOV1 turbine governor model.

Implements the PSS/E TGOV1 model without deadband.

Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Properties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
syn		Synchronous generator idx			Idx- Param	manda- tory,unique
Tn	$T_n$	Turbine power rating. Equal to Sn if not provided.		<i>MVA</i>	NumParam	
wref0	$\omega_{ref0}$	Base speed reference	1	<i>p.u.</i>	NumParam	
R	$R$	Speed regulation gain (mach. base de- fault)	0.050	<i>p.u.</i>	NumParam	ipower
VMAX	$V_{max}$	Maximum valve position	1.200	<i>p.u.</i>	NumParam	power
VMIN	$V_{min}$	Minimum valve position	0	<i>p.u.</i>	NumParam	power
T1	$T_1$	Valve time constant	0.100		NumParam	
T2	$T_2$	Lead-lag lead time constant	0.200		NumParam	
T3	$T_3$	Lead-lag lag time constant	10		NumParam	
Dt	$D_t$	Turbine damping coefficient	0		NumParam	power
Sg	$S_n$	Rated power from generator	0	<i>MVA</i>	ExtParam	
Vn	$V_n$	Rated voltage from generator	0	<i>kV</i>	ExtParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
LAG_y	$y_{LAG}$	$P_d$	State in lag TF		v_str
LL_x	$x'_{LL}$	$y_{LAG}$	State in lead-lag		v_str
omega	$\omega$		Generator speed	<i>p.u.</i>	
paux	$P_{aux}$	$P_{aux0}$	Auxiliary power input		v_str
pout	$P_{out}$	$\tau_{m0}u$	Turbine final output power		v_str
wref	$\omega_{ref}$	$\omega_{ref0}$	Speed reference variable		v_str
pref	$P_{ref}$	$R\tau_{m0}$	Reference power input		v_str
wd	$\omega_{dev}$	0	Generator under speed	<i>p.u.</i>	v_str
pd	$P_d$	$\tau_{m0}u$	Pref plus under speed times gain	<i>p.u.</i>	v_str
LL_y	$y_{LL}$	$y_{LAG}$	Output of lead-lag		v_str
tm	$\tau_m$		Mechanical power interface to SynGen		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
LAG_y	$y_{LAG}$	State	$P_d - y_{LAG}$	$T_1$
LL_x	$x'_{LL}$	State	$-x'_{LL} + y_{LAG}$	$T_3$
omega	$\omega$	ExtState	0	

## Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
paux	$P_{aux}$	Algeb	$P_{aux0} - P_{aux}$
pout	$P_{out}$	Algeb	$D_t \omega_{dev} - P_{out} + y_{LL}$
wref	$\omega_{ref}$	Algeb	$\omega_{ref0} - \omega_{ref}$
pref	$P_{ref}$	Algeb	$-P_{ref} + R\tau_{m0}$
wd	$\omega_{dev}$	Algeb	$-\omega - \omega_{dev} + \omega_{ref}$
pd	$P_d$	Algeb	$G_u (P_{aux} + P_{ref} + \omega_{dev}) - P_d$
LL_y	$y_{LL}$	Algeb	$LL_{LT1z1} LL_{LT2z1} (-x'_{LL} + y_{LL}) + T_2 (-x'_{LL} + y_{LAG}) + T_3 x'_{LL} - T_3 y_{LL}$
tm	$\tau_m$	ExtAl- geb	$u (P_{out} - \tau_{m0})$

## Services

Name	Symbol	Equation	Type
paux0	$P_{aux0}$	0	ConstService
gain	$G$	$\frac{u}{R}$	ConstService

## Discrete

Name	Symbol	Type	Info
LAG_lim	$lim_{LAG}$	AntiWindup	Limiter in Lag
LL_LT1	$LT_{LL}$	LessThan	
LL_LT2	$LT_{LL}$	LessThan	

## Blocks

Name	Symbol	Type	Info
LAG	$LAG$	LagAntiWindup	
LL	$LL$	LeadLag	

## 5.27.3 TGOV1DB

Group *TurbineGov*

TGOV1 turbine governor model with speed input deadband.

## Parameters

Name	Sym- bol	Description	De- fault	Unit	Type	Properties
idx		unique device idx			Data- Param	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			Data- Param	
syn		Synchronous generator idx			Idx- Param	manda- tory,unique
Tn	$T_n$	Turbine power rating. Equal to Sn if not provided.		<i>MVA</i>	NumParam	
wref0	$\omega_{ref0}$	Base speed reference	1	<i>p.u.</i>	NumParam	
R	$R$	Speed regulation gain (mach. base de- fault)	0.050	<i>p.u.</i>	NumParam	ipower
VMAX	$V_{max}$	Maximum valve position	1.200	<i>p.u.</i>	NumParam	power
VMIN	$V_{min}$	Minimum valve position	0	<i>p.u.</i>	NumParam	power
T1	$T_1$	Valve time constant	0.100		NumParam	
T2	$T_2$	Lead-lag lead time constant	0.200		NumParam	
T3	$T_3$	Lead-lag lag time constant	10		NumParam	
Dt	$D_t$	Turbine damping coefficient	0		NumParam	power
dbL	$db_L$	Lower bound of deadband	0	<i>p.u.</i>	NumParam	
dbU	$db_U$	Upper bound of deadband	0	<i>p.u.</i>	NumParam	
Sg	$S_n$	Rated power from generator	0	<i>MVA</i>	ExtParam	
Vn	$V_n$	Rated voltage from generator	0	<i>kV</i>	ExtParam	

## Variables

Name	Sym- bol	Initial Value	Description	Unit	Prop- ties
LAG_y	$y_{LAG}$	$P_d$	State in lag TF		v_str
LL_x	$x'_{LL}$	$y_{LAG}$	State in lead-lag		v_str
omega	$\omega$		Generator speed	<i>p.u.</i>	
paux	$P_{aux}$	$P_{aux0}$	Auxiliary power input		v_str
pout	$P_{out}$	$\tau_{m0}u$	Turbine final output power		v_str
wref	$\omega_{ref}$	$\omega_{ref0}$	Speed reference variable		v_str
pref	$P_{ref}$	$R\tau_{m0}$	Reference power input		v_str
wd	$\omega_{dev}$	0	Generator under speed	<i>p.u.</i>	v_str
pd	$P_d$	$\tau_{m0}u$	Pref plus under speed times gain	<i>p.u.</i>	v_str
LL_y	$y_{LL}$	$y_{LAG}$	Output of lead-lag		v_str
DB_y	$y_{DB}$	$DB_{dbzl}(\omega_{dev} - db_L)$ $DB_{dbzu}(\omega_{dev} - db_U)$	Deadband type 1 output		v_str
tm	$\tau_m$		Mechanical power interface to SynGen		

## Differential Equations



Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
LAG_y	$y_{LAG}$	State	$P_d - y_{LAG}$	$T_1$
LL_x	$x'_{LL}$	State	$-x'_{LL} + y_{LAG}$	$T_3$
omega	$\omega$	ExtState	0	

## Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
paux	$P_{aux}$	Algeb	$P_{aux0} - P_{aux}$
pout	$P_{out}$	Algeb	$D_t y_{DB} - P_{out} + y_{LL}$
wref	$\omega_{ref}$	Algeb	$\omega_{ref0} - \omega_{ref}$
pref	$P_{ref}$	Algeb	$-P_{ref} + R\tau_{m0}$
wd	$\omega_{dev}$	Algeb	$-\omega - \omega_{dev} + \omega_{ref}$
pd	$P_d$	Algeb	$G_u (P_{aux} + P_{ref} + y_{DB}) - P_d$
LL_y	$y_{LL}$	Algeb	$LL_{LT1z1} LL_{LT2z1} (-x'_{LL} + y_{LL}) + T_2 (-x'_{LL} + y_{LAG}) + T_3 x'_{LL} - T_3 y_{LL}$
DB_y	$y_{DB}$	Algeb	$DB_{dbzl} (\omega_{dev} - db_L) + DB_{dbzu} (\omega_{dev} - db_U) - y_{DB}$
tm	$\tau_m$	ExtAl- geb	$u (P_{out} - \tau_{m0})$

## Services

Name	Symbol	Equation	Type
paux0	$P_{aux0}$	0	ConstService
gain	$G$	$\frac{u}{R}$	ConstService

## Discrete

Name	Symbol	Type	Info
LAG_lim	$lim_{LAG}$	AntiWindup	Limiter in Lag
LL_LT1	$LT_{LL}$	LessThan	
LL_LT2	$LT_{LL}$	LessThan	
DB_db	$db_{DB}$	DeadBand	

## Blocks

Name	Symbol	Type	Info
LAG	$LAG$	LagAntiWindup	
LL	$LL$	LeadLag	
DB	$DB$	DeadBand1	deadband for under speed

## 5.27.4 IEEEG1

Group *TurbineGov*

IEEE Type 1 Speed-Governing Model.

If only one generator is connected, its *idx* must be given to *syn*, and *syn2* must be left blank. Each generator must provide data in its *Sn* base.

*syn* is connected to the high-pressure output (PHP) and the optional *syn2* is connected to the low- pressure output (PLP).

The speed deviation of generator 1 (*syn*) is measured. If the turbine rating *Tn* is not specified, the sum of *Sn* of all connected generators will be used.

Normally,  $K1 + K2 + \dots + K8 = 1.0$ . If the second generator is not connected,  $K1 + K3 + K5 + K7 = 1$ , and  $K2 + K4 + K6 + K8 = 0$ .

Parameters

Name	Symbol	Description	Default	Unit	Type	Property
idx		unique device idx			DataParam	
u	$u$	connection status	1	<i>bool</i>	NumParam	
name		device name			DataParam	
syn		Synchronous generator idx			IdxParam	mandatory
Tn	$T_n$	Turbine power rating. Equal to <i>Sn</i> if not provided.		<i>MVA</i>	NumParam	
wref0	$\omega_{ref0}$	Base speed reference	1	<i>p.u.</i>	NumParam	
syn2		Optional SynGen idx			IdxParam	
K	$K$	Gain (1/R) in mach. base	20	<i>p.u. (power)</i>	NumParam	power
T1	$T_1$	Gov. lag time const.	1		NumParam	
T2	$T_2$	Gov. lead time const.	1		NumParam	
T3	$T_3$	Valve controller time const.	0.100		NumParam	
UO	$U_o$	Max. valve opening rate	0.100	<i>p.u./sec</i>	NumParam	
UC	$U_c$	Max. valve closing rate	-0.100	<i>p.u./sec</i>	NumParam	
PMAX	$P_{MAX}$	Max. turbine power	5		NumParam	power
PMIN	$P_{MIN}$	Min. turbine power	0		NumParam	power
T4	$T_4$	Inlet piping/steam bowl time constant	0.400		NumParam	
K1	$K_1$	Fraction of power from HP	0.500		NumParam	
K2	$K_2$	Fraction of power from LP	0		NumParam	
T5	$T_5$	Time constant of 2nd boiler pass	8		NumParam	
K3	$K_3$	Fraction of HP shaft power after 2nd boiler pass	0.500		NumParam	
K4	$K_4$	Fraction of LP shaft power after 2nd boiler pass	0		NumParam	
T6	$T_6$	Time constant of 3rd boiler pass	0.500		NumParam	
K5	$K_5$	Fraction of HP shaft power after 3rd boiler pass	0		NumParam	
K6	$K_6$	Fraction of LP shaft power after 3rd boiler pass	0		NumParam	
T7	$T_7$	Time constant of 4th boiler pass	0.050		NumParam	
K7	$K_7$	Fraction of HP shaft power after 4th boiler pass	0		NumParam	
K8	$K_8$	Fraction of LP shaft power after 4th boiler pass	0		NumParam	

Continued on next page

Table 7 – continued from previous page

Name	Symbol	Description	Default	Unit	Type	Properties
Sg	$S_n$	Rated power from generator	0	MVA	ExtParam	
Vn	$V_n$	Rated voltage from generator	0	kV	ExtParam	
Sg2	$S_{n2}$	Rated power of Syn2	0	MVA	ExtParam	

## Variables

Name	Symbol	Initial Value	Description	Unit	Properties
LL_x	$x'_{LL}$	$\omega_{dev}$	State in lead-lag		v_str
IAW_y	$y_{IAW}$	$tm_{012}$	AW Integrator output		v_str
L4_y	$y_{L4}$	$y_{IAW}$	State in lag transfer function		v_str
L5_y	$y_{L5}$	$y_{L4}$	State in lag transfer function		v_str
L6_y	$y_{L6}$	$y_{L5}$	State in lag transfer function		v_str
L7_y	$y_{L7}$	$y_{L6}$	State in lag transfer function		v_str
omega	$\omega$		Generator speed	p.u.	
paux	$P_{aux}$	$P_{aux0}$	Auxiliary power input		v_str
pout	$P_{out}$	$\tau_{m0}u$	Turbine final output power		v_str
wref	$\omega_{ref}$	$\omega_{ref0}$	Speed reference variable		v_str
wd	$\omega_{dev}$	0	Generator under speed	p.u.	v_str
LL_y	$y_{LL}$	$\omega_{dev}$	Output of lead-lag		v_str
vs	$V_s$	0	Valve speed		v_str
vsl	$V_{sl}$	$U_c z_l^{HL} + U_o z_u^{HL} + V_s z_i^{HL}$	Valve move speed after limiter		v_str
PHP	$P_{HP}$	$K_1 y_{L4} + K_3 y_{L5} + K_5 y_{L6} + K_7 y_{L7}$	HP output		v_str
PLP	$P_{LP}$	$K_2 y_{L4} + K_4 y_{L5} + K_6 y_{L6} + K_8 y_{L7}$	LP output		v_str
tm	$\tau_m$		Mechanical power interface to SynGen		
tm2	$\tau_{m2}$		Mechanical power to syn2		

## Differential Equations

Name	Symbol	Type	RHS of Equation "T x' = f(x, y)"	T (LHS)
LL_x	$x'_{LL}$	State	$\omega_{dev} - x'_{LL}$	$T_1$
IAW_y	$y_{IAW}$	State	$V_{sl}$	1
L4_y	$y_{L4}$	State	$y_{IAW} - y_{L4}$	$T_4$
L5_y	$y_{L5}$	State	$y_{L4} - y_{L5}$	$T_5$
L6_y	$y_{L6}$	State	$y_{L5} - y_{L6}$	$T_6$
L7_y	$y_{L7}$	State	$y_{L6} - y_{L7}$	$T_7$
omega	$\omega$	ExtState	0	

## Algebraic Equations

Name	Sym- bol	Type	RHS of Equation "0 = g(x, y)"
paux	$P_{aux}$	Algeb	$P_{aux0} - P_{aux}$
pout	$P_{out}$	Algeb	$P_{HP} - P_{out}$
wref	$\omega_{ref}$	Algeb	$\omega_{ref0} - \omega_{ref}$
wd	$\omega_{dev}$	Algeb	$-\omega - \omega_{dev} + \omega_{ref}$
LL_y	$y_{LL}$	Algeb	$KT_1 x'_{LL} + KT_2 (\omega_{dev} - x'_{LL}) + LL_{LT1z1} LL_{LT2z1} (-K x'_{LL} + y_{LL}) - T_1 y_{LL}$
vs	$V_s$	Algeb	$-V_s + \frac{P_{aux} + tm_{012} - y_{IAW} + y_{LL}}{T_3}$
vsl	$V_{sl}$	Algeb	$U_c z_l^{HL} + U_o z_u^{HL} + V_s z_i^{HL} - V_{sl}$
PHP	$P_{HP}$	Algeb	$K_1 y_{L4} + K_3 y_{L5} + K_5 y_{L6} + K_7 y_{L7} - P_{HP}$
PLP	$P_{LP}$	Algeb	$K_2 y_{L4} + K_4 y_{L5} + K_6 y_{L6} + K_8 y_{L7} - P_{LP}$
tm	$\tau_m$	ExtAl- geb	$u(P_{out} - \tau_{m0})$
tm2	$\tau_{m2}$	ExtAl- geb	$u z_{syn2} (P_{LP} - \tau_{m02})$

## Services

Name	Symbol	Equation	Type
paux0	$P_{aux0}$	0	ConstService
_sumK18	$\sum_{i=1}^8 K_i$	$K_1 + K_2 + K_3 + K_4 + K_5 + K_6 + K_7 + K_8$	ConstService
_tm0K2	$tm_{0K2}$	$\tau_{m0} z_{syn2} (K_2 + K_4 + K_6 + K_8)$	PostInitService
_tm02K1	$tm_{02K1}$	$\tau_{m02} (K_1 + K_3 + K_5 + K_7)$	PostInitService
tm012	$tm_{012}$	$\tau_{m02} + \tau_{m0}$	ConstService

## Discrete

Name	Symbol	Type	Info
LL_LT1	$LT_{LL}$	LessThan	
LL_LT2	$LT_{LL}$	LessThan	
HL	$HL$	HardLimiter	Limiter on valve acceleration
IAW_lim	$lim_{IAW}$	AntiWindup	Limiter in integrator

## Blocks

Name	Symbol	Type	Info
LL	$LL$	LeadLag	Signal conditioning for wd
IAW	$IAW$	IntegratorAntiWindup	Valve position integrator
L4	$L4$	Lag	first process
L5	$L5$	Lag	second (reheat) process
L6	$L6$	Lag	third process
L7	$L7$	Lag	fourth (second reheat) process

## 5.28 Undefined

The undefined group. Holds models with no `group`.

Common Parameters: `u`, `name`



---

Config References

---

## 6.1 System

Option	Value	Info	Accepted values
freq	60	base frequency [Hz]	float
mva	100	system base MVA	float
store_z	0	store limiter status in TDS output	(0, 1)
ipadd	1	use spmatrix.ipadd if available	(0, 1)
diag_eps	0.000	small value for Jacobian diagonals	
warn_limits	1	warn variables initialized at limits	(0, 1)
warn_abnormal	1	warn initialization out of normal values	(0, 1)
dime_enabled	0		
dime_name	andes		
dime_protocol	ipc		
dime_address	/tmp/dime2		
numba	0	use numba for JIT compilation	(0, 1)
numba_parallel	0	enable parallel for numba.jit	(0, 1)
save_pycode	0	save generated code to ~/.andes	(0, 1)
yapf_pycode	1	format generated code with yapf	(0, 1)
use_pycode	0	use generated, saved Python code	(0, 1)

## 6.2 PFlow

Option	Value	Info	Accepted values
sparselib	klu	linear sparse solver name	('klu', 'umfpack', 'spsolve', 'cupy')
linsolve	0	solve symbolic factorization each step (enable when KLU segfaults)	(0, 1)
tol	0.000	convergence tolerance	float
max_iter	25	max. number of iterations	>=10
method	NR	calculation method	('NR', 'dishonest')
n_factorize	4	first N iterations to factorize Jacobian in dishonest method	>0
report	1	write output report	(0, 1)
degree	0	use degree in report	(0, 1)
init_tds	0	initialize TDS after PFlow	(0, 1)

## 6.3 TDS

Option	Value	Info	Accepted values
sparselib	klu	linear sparse solver name	('klu', 'umfpack', 'spsolve', 'cupy')
linsolve	0	solve symbolic factorization each step (enable when KLU segfaults)	(0, 1)
tol	0.000	convergence tolerance	float
t0	0	simulation starting time	>=0
tf	20	simulation ending time	>t0
fixt	1	use fixed step size (1) or variable (0)	(0, 1)
shrinkt	1	shrink step size for fixed method if not converged	(0, 1)
honest	0	honest Newton method that updates Jac at each step	(0, 1)
tstep	0.033	the initial step step size	float
max_iter	15	maximum number of iterations	>=10
re-fresh_event	0	refresh events at each step	(0, 1)
g_scale	1	scale algebraic residuals with time step size	(0, 1)
qrt	0	quasi-real-time stepping	bool
kqrt	1	quasi-real-time scaling factor; kqrt > 1 means slowing down	positive



## 6.4 EIG

Option	Value	Info	Accepted values
sparselib	klu	linear sparse solver name	('klu', 'umfpack', 'spsolve', 'cupy')
lin-solve	0	solve symbolic factorization each step (enable when KLU segfaults)	(0, 1)
plot	0	show plot after computation	(0, 1)



---

## Frequently Asked Questions

---

### 7.1 General

Q: What is the Hybrid Symbolic-Numeric Framework in ANDES?

A: It is a modeling and simulation framework that uses symbolic computation for descriptive modeling and code generation for fast numerical simulation. The goal of the framework is to reduce the programming efforts associated with implementing complex models and automate the research workflow of modeling, simulation, and documentation.

The framework reduces the modeling efforts from two aspects: (1) allowing modeling by typing in equations, and (2) allowing modeling using modularized control blocks and discontinuous components. One only needs to describe the model using equations and blocks without having to write the numerical code to implement the computation. The framework automatically generate symbolic expressions, computes partial derivatives, and generates vectorized numerical code.

### 7.2 Modeling

#### 7.2.1 Admittance matrix

Q: Where to find the line admittance matrix?

A: ANDES does not build line admittance matrix for computing line power injections. Instead, line power injections are computed as vectors on the two line terminal. This approach generalizes line as a power injection model.

Q: Without admittance matrix, how to switch out lines?

A: Lines can be switched out and in by using Toggler. See the example in `cases/kundur/kundur_full.xlsx`. One does not need to manually trigger a Jacobian matrix rebuild because Toggler automatically triggers it using the new connectivity status.

### 8.1 Import Errors

#### 8.1.1 ImportError: DLL load failed

Platform: Windows, error message:

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

This usually happens when andes is not installed in a Conda environment but instead in a system-wide Python whose library path was not correctly set in environment variables.

The easiest fix is to install andes in a Conda environment.

### 8.2 Runtime Errors

#### 8.2.1 EOFError: Ran out of input

The error message looks like

```
Traceback (most recent call last):
  File "/home/user/miniconda3/envs/andes/bin/andes", line 11, in <module>
    load_entry_point('andes', 'console_scripts', 'andes')()
  File "/home/user/repos/andes/andes/cli.py", line 179, in main
    return func(cli=True, **vars(args))
  File "/home/user/repos/andes/andes/main.py", line 514, in run
    system = run_case(cases[0], codegen=codegen, **kwargs)
  File "/home/user/repos/andes/andes/main.py", line 304, in run_case
    system = load(case, codegen=codegen, **kwargs)
```

(continues on next page)

(continued from previous page)

```
File "/home/user/repos/andes/andes/main.py", line 284, in load
    system.undill()
File "/home/user/repos/andes/andes/system.py", line 980, in undill
    loaded_calls = self._load_pkl()
File "/home/user/repos/andes/andes/system.py", line 963, in _load_pkl
    loaded_calls = dill.load(f)
File "/home/user/miniconda3/envs/andes/lib/python3.7/site-packages/dill/_
↪dill.py", line 270, in load
    return Unpickler(file, ignore=ignore, **kwds).load()
File "/home/user/miniconda3/envs/andes/lib/python3.7/site-packages/dill/_
↪dill.py", line 473, in load
    obj = StockUnpickler.load(self)
EOFError: Ran out of input
```

**Resolution:**

The error indicates the file for generated code is corrupt or inaccessible. It can be fixed by running `andes prepare` from the command line.

If the issue persists, try removing `~/ .andes/calls.pkl` and running `andes prepare` again.

### 9.1 Notes

#### 9.1.1 Modeling Blocks

##### State Freeze

State freeze is used by converter controllers during fault transients to fix a variable at the pre-fault values. The concept of state freeze is applicable to both state or algebraic variables. For example, in the renewable energy electric control model (REECA), the proportional-integral controllers for reactive power error and voltage error are subject to state freeze when voltage dip is observed. The internal and output states should be frozen when the freeze signal turns one and freed when the signal turns back to zero.

Freezing a state variable can be easily implemented by multiplying the freeze signal with the right-hand side (RHS) of the differential equation:

$$T\dot{x} = (1 - z_f) \times f(x)$$

where  $f(x)$  is the original RHS of the differential equation, and  $z_f$  is the freeze signal. When  $z_f$  becomes zero the differential equation will evaluate to zero, making the increment zero.

Freezing an algebraic variable is more complicate to implement. One might consider a similar solution to freezing a differential variable by constructing a piecewise equation, for example,

$$0 = (1 - z_f) \times g(y)$$

where  $g(y)$  is the original RHS of the algebraic equation. One might also need to add a small value to the diagonals of `dae.gy` associated with the algebraic variable to avoid singularity. The rationale behind this implementation is to zero out the algebraic equation mismatch and thus stop incremental correction: in

the frozen state, since  $z_f$  switches to zero, the algebraic increment should be forced to zero. This method, however, would not work when a dishonest Newton method is used.

If the Jacobian matrix is not updated after  $z_f$  switches to one, in the row associated with the equation, the derivatives will remain the same. For the algebraic equation of the PI controller given by

$$0 = (K_p u + x_i) - y$$

where  $K_p$  is the proportional gain,  $u$  is the input,  $x_i$  is the integrator output, and  $y$  is the PI controller output, the derivatives w.r.t  $u$ ,  $x_i$  and  $y$  are nonzero in the pre-frozen state. These derivative corrects  $y$  following the changes of  $u$  and  $x$ . Although  $x$  has been frozen, if the Jacobian is not rebuilt, correction will still be made due to the change of  $u$ . Since this equation is linear, only one iteration is needed to let  $y$  track the changes of  $u$ . For nonlinear algebraic variables, this approach will likely give wrong results, since the residual is pegged at zero.

To correctly freeze an algebraic variable, the freezing signal needs to be passed to an `EventFlag`, which will set an `custom_event` flag if any input changes. `EventFlag` is a `VarService` that will be evaluated at each iteration after discrete components and before equations.

## 9.2 Per Unit System

The bases for AC system are

- $S_b^{ac}$ : three-phase power in MVA. By default,  $S_b^{ac} = 100 \text{ MVA}$  (in `System.config.mva`).
- $V_b^{ac}$ : phase-to-phase voltage in kV.
- $I_b^{ac}$ : current base  $I_b^{ac} = \frac{S_b^{ac}}{\sqrt{3}V_b^{ac}}$

The bases for DC system are

- $S_b^{dc}$ : power in MVA. It is assumed to be the same as  $S_b^{ac}$ .
- $V_b^{dc}$ : voltage in kV.

## 9.3 Profiling Import

To speed up the command-line program, import profiling is used to breakdown the program loading time.

With tool `profimp`, `andes` can be profiled with `profimp "import andes" --html > andes_import.htm`. The report can be viewed in any web browser.



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

## 10.1 v1.2 Notes

### 10.1.1 v1.2.0 (2020-10-10)

This version contains major refactor for speed improvement.

- Refactored Jacobian calls generation so that for each model, one call is generated for each Jacobian type.
- Refactored Service equation generation so that the exact arguments are passed.

Also contains an experimental Python code dump function.

- Controlled in `System.config`, one can turn on `save_pycode` to dump equation and Jacobian calls to `~/ .andes/pycode`. Requires one call to `andes prepare`.
- The Python code dump can be reformatted with `yapf` through the config option `yapf_pycode`. Requires separate installation.
- The dumped Python code can be used for subsequent simulations through the config option `use_pycode`.

## 10.2 v1.1 Notes

### 10.2.1 v1.1.5 (2020-10-08)

- Allow plotting to existing axes with the same plot API.
- Added TGOV1DB model (TGOV1 with an input dead-band).
- Added an experimental numba support.
- Patched *LazyImport* for a snappier command-line interface.
- `andes selftest -q` now skips code generation.

### 10.2.2 v1.1.4 (2020-09-22)

- Support *BackRef* for groups.
- Added CLI `--pool` to use `multiprocess.Pool` for multiple cases. When combined with `--shell`, `--pool` returns `System Objects` in the list `system`.
- Fixed bugs and improved manual.

### 10.2.3 v1.1.3 (2020-09-05)

- Improved documentation.
- Minor bug fixes.

### 10.2.4 v1.1.2 (2020-09-03)

- Patched time-domain for continuing simulation.

### 10.2.5 v1.1.1 (2020-09-02)

- Added back quasi-real-time speed control through `-qrt` and `-kqrt KQRT`.
- Patched the time-domain routine for the final step.

### 10.2.6 v1.1.0 (2020-09-01)

- Defaulted *BaseVar.diag\_eps* to *System.Config.diag\_eps*.
- Added option *TDS.config.g\_scale* to allow for scaling the algebraic mismatch with step size.
- Added induction motor models *Motor3* and *Motor5* (PSAT models).
- Allow a PFlow-TDS model to skip TDS initialization by setting *ModelFlags.tds\_init* to `False`.
- Added Motor models *Motor3* and *Motor5*.

- Imported *get\_case* and *list\_cases* to the root package level.
- Added test cases (Kundur's system) with wind.

Added Generic Type 3 wind turbine component models:

- Drive-train models *WTDTAI* (dual-mass model) and *WTDS* (single-mass model).
- Aerodynamic model *WTARAI*.
- Pitch controller model *WTPTAI*.
- Torque (a.k.a. Pref) model *WTTQAI*.

## 10.3 v1.0 Notes

### 10.3.1 v1.0.8 (2020-07-29)

New features and models:

- Added renewable energy models *REECAI* and *REPCAI*.
- Added service *EventFlag* which automatically calls events if its input changes.
- Added service *ExtendedEvent* which flags an extended event for a given time.
- Added service *ApplyFunc* to apply a numeric function. For the most cases where one would need *ApplyFunc*, consider using *ConstService* first.
- Allow *selftest -q* for quick selftest by skipping codegen.
- Improved time stepping logic and convergence tests.
- Updated examples.

Default behavior changes include:

- `andes prepare` now takes three mutually exclusive arguments, *full*, *quick* and *incremental*. The command-line now defaults to the quick mode. `andes.prepare()` still uses the full mode.
- `Model.s_update` now evaluates the generated and the user-provided calls in sequence for each service in order.
- Renamed model *REGCAU1* to *REGCAI*.

### 10.3.2 v1.0.7 (2020-07-18)

- Use in-place assignment when updating Jacobian values in Triplets.
- Patched a major but simple bug where the Jacobian refactorization flag is set to the wrong place.
- New models: PMU, REGCAU1 (tests pending).
- New blocks: DeadBand1, PIFreeze, PITrackAW, PITrackAWFreeze (tests pending), and LagFreeze (tests pending).

- *andes plot* supports dashed horizontal and vertical lines through *hline1*, *hline2*, *vline1* and *vline2*.
- Discrete: renamed *DeadBand* to *DeadBandRT* (deadband with return).
- Service: renamed *FlagNotNone* to *FlagValue* with an option to flip the flags.
- Other tweaks.

### **10.3.3 v1.0.6 (2020-07-08)**

- Patched step size adjustment algorithm.
- Added Area Control Error (ACE) model.

### **10.3.4 v1.0.5 (2020-07-02)**

- Minor bug fixes for service initialization.
- Added a wrapper to call `TDS.fg_update` to allow passing variables from caller.
- Added pre-event time to the `switch_times`.

### **10.3.5 v1.0.4 (2020-06-26)**

- Implemented compressed NumPy format (npz) for time-domain simulation output data file.
- Implemented optional attribute *vtype* for specifying data type for Service.
- Patched COI speed initialization.
- Patched PSS/E parser for two-winding transformer winding and impedance modes.

### **10.3.6 v1.0.3 (2020-06-02)**

- Patches *PQ* model equations where the "or" logic "I" is ignored in equation strings. To adjust PQ load in time domain simulation, refer to the note in *pq.py*.
- Allow *Model.alter* to update service values.

### **10.3.7 v1.0.2 (2020-06-01)**

- Patches the conda-forge script to use SymPy < 1.6. After SymPy version 1.5.1, comparison operations cannot be sympified. Pip installations are not affected.

### **10.3.8 v1.0.1 (2020-05-27)**

- Generate one lambda function for each of f and g, instead of generating one for each single f/g equation. Requires to run *andes prepare* after updating.

### 10.3.9 v1.0.0 (2020-05-25)

This release is going to be tagged as v0.9.5 and later tagged as v1.0.0.

- Added verification results using IEEE 14-bus, NPCC, and WECC systems under folder *examples*.
- Patches GENROU and EXDC2 models.
- Updated test cases for WECC, NPCC IEEE 14-bus.
- Documentation improvements.
- Various tweaks.

## 10.4 Pre-v1.0.0

### 10.4.1 v0.9.4 (2020-05-20)

- Added exciter models EXST1, ESST3A, ESDC2A, SEXS, and IEEEEX1, turbine governor model IEEEG1 (dual-machine support), and stabilizer model ST2CUT.
- Added blocks HVGate and LVGate with a work-around for sympy.maximum/ minimum.
- Added services *PostInitService* (for storing initialized values), and *VarService* (variable services that get updated) after limiters and before equations).
- Added service *InitChecker* for checking initialization values against typical values. Warnings will be issued when out of bound or equality/ inequality conditions are not met.
- Allow internal variables to be associated with a discrete component which will be updated before initialization (through *BaseVar.discrete*).
- Allow turbine governors to specify an optional *Tn* (turbine rating). If not provided, turbine rating will fall back to *Sn* (generator rating).
- Renamed *OptionalSelect* to *DataSelect*; Added *NumSelect*, the array-based version of *DataSelect*.
- Allow to regenerate code for updated models through `andes prepare -qi`.
- Various patches to allow zeroing out time constants in transfer functions.

### 10.4.2 v0.9.3 (2020-05-05)

This version contains bug fixes and performance tweaks.

- Fixed an *AntiWindup* issue that causes variables to stuck at limits.
- Allow `TDS.run()` to resume from a stopped simulation and run to the new end time in `TDS.config.tf`.
- Improved TDS data dump speed by not constructing `DataFrame` by default.
- Added tests for *kundur\_full.xlsx* and *kundur\_aw.xlsx* to ensure results are the same as known values.

- Other bug fixes.

### 10.4.3 v0.9.1 (2020-05-02)

This version accelerates computations by about 35%.

- Models with flag `collate=False`, which is the new default, will slice DAE arrays for all internal vars to reduce copying back and forth.
- The change above greatly reduced computation time. For `kundur_ieeest.xlsx`, simulation time is down from 2.50 sec to 1.64 sec.
- The side-effects include a change in variable ordering in output `lst` file. It also eliminated the feasibility of evaluating model equations in parallel, which has not been implemented and does not seem promising in Python.
- Separated symbolic processor and documentation generator from `Model` into `SymProcessor` and `Documenter` classes.
- `andes prepare` now shows progress in the console.
- Store exit code in `System.exit_code` and returns to system when called from CLI.
- Refactored the solver interface.
- Patched `Config.check` for routines.
- SciPy Newton-Krylov power flow solver is no longer supported.
- Patched a bug in v0.9.0 related to *dae.Tf*.

### 10.4.4 v0.8.8 (2020-04-28)

This update contains a quick but significant fix to boost the simulation speed by avoiding calls to empty user-defined numerical calls.

- In *Model.flags* and *Block.flags*, added *f\_num*, *g\_num* and *j\_num* to indicate if user-defined numerical calls exist.
- In *Model.f\_update*, *Model.g\_update* and *Model.j\_update*, check the above flags to avoid unnecessary calls to empty numeric functions.
- For the *kundur\_ieeest.xlsx* case, simulation time was reduced from 3.5s to 2.7s.

### 10.4.5 v0.8.7 (2020-04-28)

- Changed *RefParam* to a service type called *BackRef*.
- Added *DeviceFinder*, a service type to find device idx when not provided. *DeviceFinder* will also automatically add devices if not found.
- Added *OptionalSelect*, a service type to select optional parameters if provided and select fallback ones otherwise.

- Added discrete types *Derivative*, *Delay*, and *Average*,
- Implemented full IEEEEST stabilizer.
- Implemented COI for generator speed and angle measurement.

#### 10.4.6 v0.8.6 (2020-04-21)

This release contains important documentation fixes and two new blocks.

- Fixed documentations in *andes doc* to address a misplacement of symbols and equations.
- Converted all blocks to the division-free formulation (with *dae.zf* renamed to *dae.Tf*).
- Fixed equation errors in the block documentation.
- Implemented two new blocks: *Lag2ndOrd* and *LeadLag2ndOrd*.
- Added a prototype for IEEEEST stabilizer with some fixes needed.

#### 10.4.7 v0.8.5 (2020-04-17)

- Converted the differential equations to the form of  $T \dot{x} = f(x, y)$ , where  $T$  is supplied to `t_const` of `State/ExtState`.
- Added the support for Config fields in documentation (in *andes doc* and on *readthedocs*).
- Added Config consistency checking.
- Converted *Model.idx* from a list to *DataParam*.
- Renamed the API of routines (summary, init, run, report).
- Automatically generated indices now start at 1 (i.e., "GENCLS\_1" is the first GENCLS device).
- Added test cases for WECC system. The model with classical generators is verified against TSAT.
- Minor features: *andes -v l* for debug output with levels and line numbers.

#### 10.4.8 v0.8.4 (2020-04-07)

- Added support for JSON case files. Convert existing case file to JSON with `--convert json`.
- Added support for PSS/E dyr files, loadable with `-addfile ADDFILE`.
- Added `andes plot --xargs` for searching variable name and plotting. See example 6.
- Various bug fixes: Fault power injection fix;

#### 10.4.9 v0.8.3 (2020-03-25)

- Improved storage for Jacobian triplets (see `andes.core.triplet.JacTriplet`).
- On-the-fly parameter alteration for power flow calculations (`Model.alter` method).

- Exported frequently used functions to the root package (`andes.config_logger`, `andes.run`, `andes.prepare` and `andes.load`).
- Return a list of System objects when multiprocessing in an interactive environment.
- Exported classes to *andes.core*.
- Various bug fixes and documentation improvements.

### 10.4.10 v0.8.0 (2020-02-12)

- First release of the hybrid symbolic-numeric framework in ANDES.
- A new framework is used to describe DAE models, generate equation documentation, and generate code for numerical simulation.
- Models are written in the new framework. Supported models include GENCLS, GENROU, EXDC2, TGOV1, TG2
- PSS/E raw parser, MATPOWER parser, and ANDES xlsx parser.
- Newton-Raphson power flow, trapezoidal rule for numerical integration, and full eigenvalue analysis.

### 10.4.11 v0.6.9 (2020-02-12)

- Version 0.6.9 is the last version for the numeric-only modeling framework.
- This version will not be updated any more. But, models, routines and functions will be ported to the new version.



### 11.1 GNU Public License v3

Copyright 2015-2020 Hantao Cui.

ANDES is free software; you can redistribute it and/or modify it under the terms of the [GNU General Public License](#) as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

ANDES is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the [GNU General Public License](#) for more details.



## 12.1 andes.core package

### 12.1.1 Submodules

### 12.1.2 andes.core.block module

```
class andes.core.block.Block (name: Optional[str] = None, tex_name: Optional[str] =  
None, info: Optional[str] = None, namespace: str = 'lo-  
cal')
```

Bases: `object`

Base class for control blocks.

Blocks are meant to be instantiated as Model attributes to provide pre-defined equation sets. Subclasses must overload the `__init__` method to take custom inputs. Subclasses of Block must overload the `define` method to provide initialization and equation strings. Exported variables, services and blocks must be constructed into a dictionary `self.vars` at the end of the constructor.

Blocks can be nested. A block can have blocks but itself as attributes and therefore reuse equations. When a block has sub-blocks, the outer block must be constructed with a `'name'`.

Nested block works in the following way: the parent block modifies the sub-block's `name` attribute by prepending the parent block's name at the construction phase. The parent block then exports the sub-block as a whole. When the parent Model class picks up the block, it will recursively import the variables in the block and the sub-blocks correctly. See the example section for details.

#### Parameters

**name** [str, optional] Block name

**tex\_name** [str, optional] Block LaTeX name

**info** [str, optional] Block description.

**namespace** [str, local or parent] Namespace of the exported elements. If 'local', the block name will be prepended by the parent. If 'parent', the original element name will be used when exporting.

**Warning:** It is a good practice to avoid more than one level of nesting, to avoid multi-underscore variable names.

## Examples

Example for two-level nested blocks. Suppose we have the following hierarchy

```
SomeModel  instance M
|
LeadLag A  exports (x, y)
|
Lag B      exports (x, y)
```

SomeModel instance M contains an instance of LeadLag block named A, which contains an instance of a Lag block named B. Both A and B exports two variables *x* and *y*.

In the code of Model, the following code is used to instantiate LeadLag

```
class SomeModel:
    def __init__(...):
        ...
        self.A = LeadLag(name='A',
                          u=self.foo1,
                          T1=self.foo2,
                          T2=self.foo3)
```

To use Lag in the LeadLag code, the following lines are found in the constructor of LeadLag

```
class LeadLag:
    def __init__(name, ...)
        ...
        self.B = Lag(u=self.y, K=self.K, T=self.T)
        self.vars = {..., 'A': self.A}
```

The `__setattr__` magic of LeadLag takes over the construction and assigns *A\_B* to *B.name*, given *A*'s name provided at run time. *self.A* is exported with the internal name *A* at the end.

Again, the LeadLag instance name (*A* in this example) MUST be provided in *SomeModel*'s constructor for the name prepending to work correctly. If there is more than one level of nesting, other than the leaf-level block, all parent blocks' names must be provided at instantiation.

When *A* is picked up by *SomeModel.\_\_setattr\_\_*, *B* is captured from *A*'s exports. Recursively, *B*'s variables are exported, Recall that *B.name* is now *A\_B*, following the naming rule (parent block's name + variable name), *B*'s internal variables become *A\_B\_x* and *A\_B\_y*.

In this way, B's `define()` needs no modification since the naming rule is the same. For example, B's internal `y` is always `{self.name}_y`, although B has gotten a new name `A_B`.

#### **class\_name**

Return the class name.

#### **define()**

Function for setting the initialization and equation strings for internal variables. This method must be implemented by subclasses.

The equations should be written with the "final" variable names. Let's say the block instance is named `blk` (kept at `self.name` of the block), and an internal variable `v` is defined. The internal variable will be captured as `blk_v` by the parent model. Therefore, all equations should use `{self.name}_v` to represent variable `v`, where `{self.name}` is the name of the block at run time.

On the other hand, the names of externally provided parameters or variables are obtained by directly accessing the `name` attribute. For example, if `self.T` is a parameter provided through the block constructor, `{self.T.name}` should be used in the equation.

See also:

***PIController.define*** Equations for the PI Controller block

### **Examples**

An internal variable `v` has a trivial equation  $T = v$ , where `T` is a parameter provided to the block constructor.

In the model, one has

```
class SomeModel():
    def __init__(...):
        self.input = Algeb()
        self.T = Param()

        self.blk = ExampleBlock(u=self.input, T=self.T)
```

In the `ExampleBlock` function, the internal variable is defined in the constructor as

```
class ExampleBlock():
    def __init__(...):
        self.v = Algeb()
        self.vars = {'v', self.v}
```

In the `define`, the equation is provided as

```
def define(self):
    self.v.v_str = '{self.T.name}'
    self.v.e_str = '{self.T.name} - {self.name}_v'
```

In the parent model,  $v$  from the block will be captured as `blk_v`, and the equation will evaluate into

```
self.blk_v.v_str = 'T'
self.blk_v.e_str = 'T - blk_v'
```

**static enforce\_tex\_name** (*fields*)

Enforce `tex_name` is not `None`

**export** ()

Method for exporting instances defined in this class in a dictionary. This method calls the `define` method first and returns `self.vars`.

### Returns

**dict** Keys are the (last section of the) variable name, and the values are the attribute instance.

**f\_numeric** (\*\**kwargs*)

Function call to update differential equation values.

This function should modify the `e` value of block `State` and `ExtState` in place.

**g\_numeric** (\*\**kwargs*)

Function call to update algebraic equation values.

This function should modify the `e` value of block `Algeb` and `ExtAlgeb` in place.

**j\_numeric** ()

This function stores the constant and variable jacobian information in corresponding lists.

Constant jacobians are stored by indices and values in, for example, *ifxc*, *jfxc* and *vfxc*. Value scalars or arrays are stored in *vfxc*.

Variable jacobians are stored by indices and functions. The function shall return the value of the corresponding jacobian elements.

**j\_reset** ()

Helper function to clear the lists holding the numerical Jacobians.

This function should be only called once at the beginning of `j_numeric` in blocks.

**class** `andes.core.block.DeadBand1` (*u, center, lower, upper, enable=True, name=None, tex\_name=None, info=None, namespace='local'*)

Bases: `andes.core.block.Block`

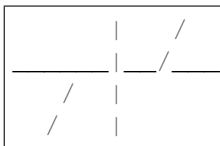
Deadband type 1.

### Parameters

**center** Default value when within the deadband. If the input is an error signal, center should be set to zero.

### Notes

Block diagram



**define** ()

### Notes

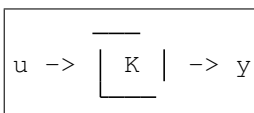
Implemented equation:

$$0 = center + z_u * (u - upper) + z_l * (u - lower) - y$$

**class** andes.core.block.**Gain** (*u*, *K*, *name=None*, *tex\_name=None*, *info=None*)

Bases: *andes.core.block.Block*

Gain block.



Exports an algebraic output *y*.

**define** ()

Implemented equation and the initial condition are

$$y = Ku$$

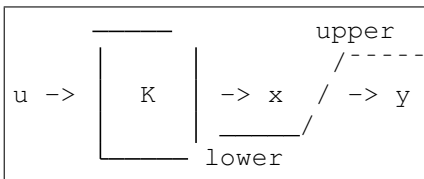
$$y^{(0)} = Ku^{(0)}$$

**class** andes.core.block.**GainLimiter** (*u*, *K*, *upper*, *lower*, *no\_upper=False*, *no\_lower=False*, *name=None*, *tex\_name=None*, *info=None*)

Bases: *andes.core.block.Block*

Gain followed by a limiter.

Exports the limited output *y*, unlimited output *x*, and HardLimiter *lim*.



### Parameters

**u** [str, BaseVar] Input variable, or an equation string for constructing an anonymous variable

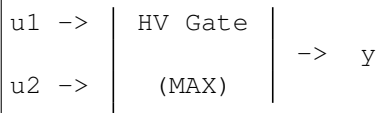
**define()**

TODO: write docstring

**class** andes.core.block.HVGate(u1, u2, name=None, tex\_name=None, info=None)

Bases: *andes.core.block.Block*

High Value Gate. Outputs the maximum of two inputs.



**define()**

Implemented equations and initial conditions

$$0 = s_0^{sl} u_1 + s_1^{sl} u_2 - yy_0 = \text{maximum}(u_1, u_2)$$

## Notes

In the implementation, one should not use

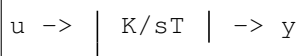
```
self.y.v_str = f'maximum({self.u1.name}, {self.u2.name})',
```

because SymPy processes this equation to *{self.u1.name}*. Not sure if this is a bug or intended.

**class** andes.core.block.Integrator(u, T, K, y0, name=None, tex\_name=None, info=None)

Bases: *andes.core.block.Block*

Integrator block.



Exports a differential variable y. The initial output is specified by y0 and default to zero.

**define()**

Implemented equation and the initial condition are

$$\dot{y} = Ku$$

$$y^{(0)} = 0$$

**class** andes.core.block.IntegratorAntiWindup(u, T, K, y0, lower, upper, name=None, tex\_name=None, info=None)

Bases: *andes.core.block.Block*

Integrator block with anti-windup limiter.





Exports a differential variable  $y$  and an AntiWindup  $lim$ . The initial output must be specified through  $y0$ .

**define()**

Implemented equation and the initial condition are

$$\dot{y} = Ku$$

$$y^{(0)} = 0$$

**class** andes.core.block.LVGate( $u1, u2, name=None, tex\_name=None, info=None$ )

Bases: [andes.core.block.Block](#)

Low Value Gate. Outputs the minimum of the two inputs.



**define()**

Implemented equations and initial conditions

$$0 = s_0^{sl} u_1 + s_1^{sl} u_2 - yy_0 = \text{minimum}(u_1, u_2)$$

## Notes

Same problem as *HVGate* as *minimum* does not sympify correctly.

**class** andes.core.block.Lag( $u, T, K, name=None, tex\_name=None, info=None$ )

Bases: [andes.core.block.Block](#)

Lag (low pass filter) transfer function.



Exports one state variable  $y$  as the output.

## Parameters

**K** Gain

**T** Time constant

**u** Input variable

**define** ()

### Notes

Equations and initial values are

$$T\dot{y} = (Ku - y)$$
$$y^{(0)} = Ku$$

**class** `andes.core.block.Lag2ndOrd`(*u*, *K*, *T1*, *T2*, *name=None*, *tex\_name=None*,  
*info=None*)

Bases: `andes.core.block.Block`

Second order lag transfer function (low-pass filter)



Exports one two state variables (*x*, *y*), where *y* is the output.

### Parameters

**u** Input

**K** Gain

**T1** First order time constant

**T2** Second order time constant

**define** ()

### Notes

Implemented equations and initial values are

$$T_2\dot{x} = Ku - y - T_1x$$
$$\dot{y} = x$$
$$x^{(0)} = 0$$
$$y^{(0)} = Ku$$

```
class andes.core.block.LagAWFreeze(u, T, K, lower, upper, freeze, name=None,
                                tex_name=None, info=None)
```

Bases: `andes.core.block.LagAntiWindup`

Lag with anti-windup limiter and state freeze.

```
define()
```

### Notes

Equations and initial values are

$$T\dot{y} = (1 - freeze)(Ku - y)$$

$$y^{(0)} = Ku$$

```
class andes.core.block.LagAntiWindup(u, T, K, lower, upper, name=None,
                                tex_name=None, info=None)
```

Bases: `andes.core.block.Block`

Lag (low pass filter) transfer function block with an anti-windup limiter.



Exports one state variable  $y$  as the output and one AntiWindup instance  $lim$ .

### Parameters

**K** Gain

**T** Time constant

**u** Input variable

```
define()
```

### Notes

Equations and initial values are

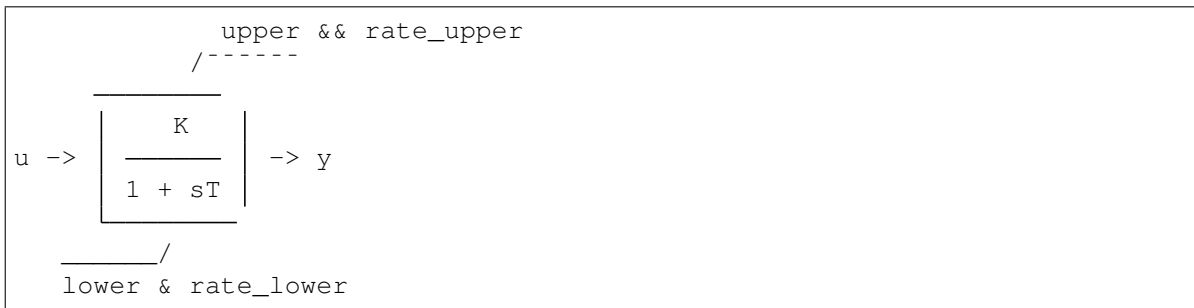
$$T\dot{y} = (Ku - y)$$

$$y^{(0)} = Ku$$

```
class andes.core.block.LagAntiWindupRate(u, T, K, lower, upper,
                                         rate_lower, rate_upper,
                                         no_lower=False, no_upper=False,
                                         rate_no_lower=False,
                                         rate_no_upper=False,
                                         rate_lower_cond=None,
                                         rate_upper_cond=None, name=None,
                                         tex_name=None, info=None)
```

Bases: `andes.core.block.Block`

Lag (low pass filter) transfer function block with a rate limiter and an anti-windup limiter.



Exports one state variable  $y$  as the output and one AntiWindupRate instance  $lim$ .

#### Parameters

**K** Gain

**T** Time constant

**u** Input variable

**define()**

#### Notes

Equations and initial values are

$$T\dot{y} = (Ku - y)$$
$$y^{(0)} = Ku$$

```
class andes.core.block.LagFreeze(u, T, K, freeze, name=None, tex_name=None,
                                info=None)
```

Bases: `andes.core.block.Lag`

Lag with a state freeze input.

**define()**

## Notes

Equations and initial values are

$$T\dot{y} = (1 - freeze) * (Ku - y)$$

$$y^{(0)} = Ku$$

```
class andes.core.block.LagRate(u, T, K, rate_lower, rate_upper, rate_no_lower=False,
                              rate_no_upper=False, rate_lower_cond=None,
                              rate_upper_cond=None, name=None,
                              tex_name=None, info=None)
```

Bases: [andes.core.block.Block](#)

Lag (low pass filter) transfer function block with a rate limiter and an anti-windup limiter.



Exports one state variable  $y$  as the output and one AntiWindupRate instance *lim*.

## Parameters

**K** Gain

**T** Time constant

**u** Input variable

**define()**

## Notes

Equations and initial values are

$$T\dot{y} = (Ku - y)$$

$$y^{(0)} = Ku$$

```
class andes.core.block.LeadLag(u, T1, T2, K=1, zero_out=True, name=None,
                              tex_name=None, info=None)
```

Bases: [andes.core.block.Block](#)

Lead-Lag transfer function block in series implementation

$$u \rightarrow \left[ K \frac{1 + sT_1}{1 + sT_2} \right] \rightarrow y$$

Exports two variables: internal state  $x$  and output algebraic variable  $y$ .

### Parameters

**T1** [BaseParam] Time constant 1

**T2** [BaseParam] Time constant 2

**zero\_out** [bool] True to allow zeroing out lead-lag as a pass through (when  $T_1=T_2=0$ )

### Notes

To allow zeroing out lead-lag as a pure gain, set `zero_out` to *True*.

**define** ()

### Notes

Implemented equations and initial values

$$\begin{aligned} T_2 \dot{x}' &= (u - x') \\ T_2 y &= K T_1 (u - x') + K T_2 x' + E_2, \text{ where} \\ E_2 &= \begin{cases} (y - K x') & \text{if } T_1 = T_2 = 0 \& \text{zero\_out} = \text{True} \\ 0 & \text{otherwise} \end{cases} \\ x'^{(0)} &= u \\ y^{(0)} &= K u \end{aligned}$$

**class** andes.core.block.**LeadLag2ndOrd**( $u$ ,  $T_1$ ,  $T_2$ ,  $T_3$ ,  $T_4$ , `zero_out=False`,  
`name=None`, `tex_name=None`, `info=None`)

Bases: `andes.core.block.Block`

Second-order lead-lag transfer function block

$$u \rightarrow \left[ \frac{1 + sT_3 + s^2 T_4}{1 + sT_1 + s^2 T_2} \right] \rightarrow y$$

Exports two internal states ( $x_1$  and  $x_2$ ) and output algebraic variable  $y$ .

# TODO: instead of implementing `zero_out` using *LessThan* and an additional term, consider correcting all parameters to 1 if all are 0.

**define** ()

## Notes

Implemented equations and initial values are

$$\begin{aligned}
 T_2 \dot{x}_1 &= u - x_2 - T_1 x_1 \\
 \dot{x}_2 &= x_1 \\
 T_2 y &= T_2 x_2 + T_2 T_3 x_1 + T_4 (u - x_2 - T_1 x_1) + E_2, \text{ where} \\
 E_2 &= \begin{cases} (y - x_2) & \text{if } T_1 = T_2 = T_3 = T_4 = 0 \& \text{zero\_out} = \text{True} \\ 0 & \text{otherwise} \end{cases} \\
 x_1^{(0)} &= 0 \\
 x_2^{(0)} &= y^{(0)} = u
 \end{aligned}$$

```
class andes.core.block.LeadLagLimit(u, T1, T2, lower, upper, name=None,
                                   tex_name=None, info=None)
```

Bases: *andes.core.block.Block*

Lead-Lag transfer function block with hard limiter (series implementation)

$$u \rightarrow \left[ \begin{array}{c} \frac{1 + sT_1}{1 + sT_2} \end{array} \right] \begin{array}{c} \xrightarrow{\text{upper}} \\ \xrightarrow{\text{lower}} \end{array} \begin{array}{c} \text{ynl} \\ \text{y} \end{array}$$

Exports four variables: state  $x$ , output before hard limiter  $ynl$ , output  $y$ , and AntiWindup  $lim$ .

```
define ()
```

## Notes

Implemented control block equations (without limiter) and initial values

$$\begin{aligned}
 T_2 \dot{x}' &= (u - x') \\
 T_2 y &= T_1 (u - x') + T_2 x' \\
 x'^{(0)} &= y^{(0)} = u
 \end{aligned}$$

```
class andes.core.block.PIAWHardLimit(u, kp, ki, aw_lower, aw_upper, lower, upper,
                                       no_lower=False, no_upper=False, ref=0.0,
                                       x0=0.0, name=None, tex_name=None,
                                       info=None)
```

Bases: *andes.core.block.PIDController*

PI controller with anti-windup limiter on the integrator and hard limit on the output.

Limits `lower` and `upper` are on the final output, and `aw_lower` `aw_upper` are on the integrator.

```
define ()
```

Define equations for the PI Controller.

### Notes

One state variable `xi` and one algebraic variable `y` are added.

Equations implemented are

$$\begin{aligned}\dot{x}_i &= k_i * (u - ref) \\ y &= x_i + k_p * (u - ref)\end{aligned}$$

```
class andes.core.block.PIController(u, kp, ki, ref=0.0, x0=0.0, name=None,
                                   tex_name=None, info=None)
```

Bases: `andes.core.block.Block`

Proportional Integral Controller.

The controller takes an error signal as the input. It takes an optional `ref` signal, which will be subtracted from the input.

### Parameters

**u** [BaseVar] The input variable instance

**kp** [BaseParam] The proportional gain parameter instance

**ki** [[type]] The integral gain parameter instance

**define()**

Define equations for the PI Controller.

### Notes

One state variable `xi` and one algebraic variable `y` are added.

Equations implemented are

$$\begin{aligned}\dot{x}_i &= k_i * (u - ref) \\ y &= x_i + k_p * (u - ref)\end{aligned}$$

```
class andes.core.block.PIControllerNumeric(u, kp, ki, ref=0.0, name=None,
                                           tex_name=None, info=None)
```

Bases: `andes.core.block.Block`

A PI Controller implemented with numerical function calls.

`ref` must not be a variable.

**define()**

Skip the symbolic definition

**f\_numeric(\*\*kwargs)**

Function call to update differential equation values.

This function should modify the `e` value of block `State` and `ExtState` in place.



**g\_numeric** (\*\*kwargs)

Function call to update algebraic equation values.

This function should modify the `e` value of block `Algeb` and `ExtAlgeb` in place.

**j\_numeric** ()

This function stores the constant and variable jacobian information in corresponding lists.

Constant jacobians are stored by indices and values in, for example, *ifxc*, *jfxc* and *vfxc*. Value scalars or arrays are stored in *vfxc*.

Variable jacobians are stored by indices and functions. The function shall return the value of the corresponding jacobian elements.

**class** `andes.core.block.PIFreeze` (*u, kp, ki, freeze, ref=0.0, x0=0.0, name=None, tex\_name=None, info=None*)

Bases: `andes.core.block.PIController`

PI controller with state freeze.

Freezes state when the corresponding *freeze == 1*.

## Notes

Tested in *experimental.TestPITrackAW.PIFreeze*.

**define** ()

## Notes

One state variable *xi* and one algebraic variable *y* are added.

Equations implemented are

$$\begin{aligned} \dot{x}_i &= k_i * (u - ref) \\ y &= (1 - freeze) * (x_i + k_p * (u - ref)) + freeze * y \end{aligned}$$

**class** `andes.core.block.PITrackAW` (*u, kp, ki, ks, lower, upper, no\_lower=False, no\_upper=False, ref=0.0, x0=0.0, name=None, tex\_name=None, info=None*)

Bases: `andes.core.block.Block`

PI with tracking anti-windup limiter

**define** ()

Function for setting the initialization and equation strings for internal variables. This method must be implemented by subclasses.

The equations should be written with the "final" variable names. Let's say the block instance is named *blk* (kept at `self.name` of the block), and an internal variable *v* is defined. The internal variable will be captured as `blk_v` by the parent model. Therefore, all equations should use `{self.name}_v` to represent variable *v*, where `{self.name}` is the name of the block at run time.

On the other hand, the names of externally provided parameters or variables are obtained by directly accessing the name attribute. For example, if `self.T` is a parameter provided through the block constructor, `{self.T.name}` should be used in the equation.

See also:

*`PIController.define`* Equations for the PI Controller block

## Examples

An internal variable `v` has a trivial equation  $T = v$ , where `T` is a parameter provided to the block constructor.

In the model, one has

```
class SomeModel():
    def __init__(...):
        self.input = Algeb()
        self.T = Param()

        self.blk = ExampleBlock(u=self.input, T=self.T)
```

In the `ExampleBlock` function, the internal variable is defined in the constructor as

```
class ExampleBlock():
    def __init__(...):
        self.v = Algeb()
        self.vars = {'v', self.v}
```

In the `define`, the equation is provided as

```
def define(self):
    self.v.v_str = '{self.T.name}'
    self.v.e_str = '{self.T.name} - {self.name}_v'
```

In the parent model, `v` from the block will be captured as `blk_v`, and the equation will evaluate into

```
self.blk_v.v_str = 'T'
self.blk_v.e_str = 'T - blk_v'
```

```
class andes.core.block.PITrackAWFreeze(u, kp, ki, ks, lower, upper, freeze,
                                       no_lower=False, no_upper=False,
                                       ref=0.0, x0=0.0, name=None,
                                       tex_name=None, info=None)
```

Bases: *`andes.core.block.PITrackAW`*

PI controller with tracking anti-windup limiter and state freeze.

**define()**

Function for setting the initialization and equation strings for internal variables. This method must be implemented by subclasses.

The equations should be written with the "final" variable names. Let's say the block instance is named *blk* (kept at `self.name` of the block), and an internal variable *v* is defined. The internal variable will be captured as `blk_v` by the parent model. Therefore, all equations should use `{self.name}_v` to represent variable *v*, where `{self.name}` is the name of the block at run time.

On the other hand, the names of externally provided parameters or variables are obtained by directly accessing the `name` attribute. For example, if `self.T` is a parameter provided through the block constructor, `{self.T.name}` should be used in the equation.

See also:

*PIController.define* Equations for the PI Controller block

## Examples

An internal variable *v* has a trivial equation  $T = v$ , where *T* is a parameter provided to the block constructor.

In the model, one has

```
class SomeModel():
    def __init__(...):
        self.input = Algeb()
        self.T = Param()

        self.blk = ExampleBlock(u=self.input, T=self.T)
```

In the `ExampleBlock` function, the internal variable is defined in the constructor as

```
class ExampleBlock():
    def __init__(...):
        self.v = Algeb()
        self.vars = {'v', self.v}
```

In the `define`, the equation is provided as

```
def define(self):
    self.v.v_str = '{self.T.name}'
    self.v.e_str = '{self.T.name} - {self.name}_v'
```

In the parent model, *v* from the block will be captured as `blk_v`, and the equation will evaluate into

```
self.blk_v.v_str = 'T'
self.blk_v.e_str = 'T - blk_v'
```

```
class andes.core.block.Piecewise(u, points: Union[List[T], Tuple], fns:
                                Union[List[T], Tuple], name=None,
                                tex_name=None, info=None)
```

Bases: *andes.core.block.Block*

Piecewise block. Outputs an algebraic variable  $y$ .

This block takes a list of  $N$  points,  $[x_0, x_1, \dots, x_{n-1}]$  to define  $N+1$  ranges, namely  $(-\infty, x_0)$ ,  $(x_0, x_1)$ , ...,  $(x_{n-1}, +\infty)$ . and a list of  $N+1$  function strings  $[fun_0, \dots, fun_n]$ .

Inputs that fall within each range applies the corresponding function. The first range  $(-\infty, x_0)$  applies  $fun_0$ , and the last range  $(x_{n-1}, +\infty)$  applies the last function  $fun_n$ .

### Parameters

**points** [list, tuple] A list of piecewise points. Need to be provided in the constructor function.

**funcs** [list, tuple] A list of strings for the piecewise functions. Need to be provided in the overloaded *define* function.

**define()**

Build the equation string for the piecewise equations.

`self.funcs` needs to be provided with the function strings corresponding to each range.

**class** `andes.core.block.Washout` ( $u, T, K, name=None, tex\_name=None, info=None$ )

Bases: `andes.core.block.Block`

Washout filter (high pass) block.

$$u \rightarrow \left[ \frac{sK}{1 + sT} \right] \rightarrow y$$

Exports state  $x$  (symbol  $x'$ ) and output algebraic variable  $y$ .

**define()**

### Notes

Equations and initial values:

$$\begin{aligned} T\dot{x}' &= (u - x') \\ Ty &= K(u - x') \\ x'^{(0)} &= u \\ y^{(0)} &= 0 \end{aligned}$$

**class** `andes.core.block.WashoutOrLag` ( $u, T, K, name=None, zero\_out=True, tex\_name=None, info=None$ )

Bases: `andes.core.block.Washout`

Washout with the capability to convert to Lag when  $K = 0$ .

Can be enabled with `zero_out`. Need to provide `name` to construct.

Exports state  $x$  (symbol  $x'$ ), output algebraic variable  $y$ , and a LessThan block *LT*.

**Parameters**

**zero\_out** [bool, optional] If True, sT will become 1, and the washout will become a low-pass filter. If False, functions as a regular Washout.

**define()**

**Notes**

Equations and initial values:

$$\begin{aligned}Tx' &= (u - x') \\Ty &= z_0 K(u - x') + z_1 Tx \\x^{(0)} &= u \\y^{(0)} &= 0\end{aligned}$$

where  $z\_0$  is a flag array for the greater-than-zero elements, and  $z\_1$  is that for the less-than or equal-to zero elements.

**12.1.3 andes.core.discrete module**

```
class andes.core.discrete.AntiWindup(u,      lower,      upper,      enable=True,
                                     no_lower=False,      no_upper=False,
                                     name=None, tex_name=None, info=None,
                                     state=None)
```

Bases: *andes.core.discrete.Limiter*

Anti-windup limiter.

Anti-windup limiter prevents the wind-up effect of a differential variable. The derivative of the differential variable is reset if it continues to increase in the same direction after exceeding the limits. During the derivative return, the limiter will be inactive

```
if x > xmax and x dot > 0: x = xmax and x dot = 0
if x < xmin and x dot < 0: x = xmin and x dot = 0
```

This class takes one more optional parameter for specifying the equation.

**Parameters**

**state** [State, ExtState] A State (or ExtState) whose equation value will be checked and, when condition satisfies, will be reset by the anti-windup-limiter.

**check\_eq()**

Check the variables and equations and set the limiter flags. Reset differential equation values based on limiter flags.

## Notes

The current implementation reallocates memory for *self.x\_set* in each call. Consider improving for speed. (TODO)

**check\_var** (\*args, \*\*kwargs)

This function is empty. Defers *check\_var* to *check\_eq*.

```
class andes.core.discrete.AntiWindupRate(u,          lower,          upper,
                                         rate_lower,      rate_upper,
                                         no_lower=False,  no_upper=False,
                                         rate_no_lower=False,
                                         rate_no_upper=False,
                                         rate_lower_cond=None,
                                         rate_upper_cond=None, enable=True,
                                         name=None,       tex_name=None,
                                         info=None)
```

Bases: *andes.core.discrete.AntiWindup*, *andes.core.discrete.RateLimiter*

Anti-windup limiter with rate limits

**check\_eq**()

Check the variables and equations and set the limiter flags. Reset differential equation values based on limiter flags.

## Notes

The current implementation reallocates memory for *self.x\_set* in each call. Consider improving for speed. (TODO)

```
class andes.core.discrete.Average(u,    mode='step',    delay=0,    name=None,
                                   tex_name=None, info=None)
```

Bases: *andes.core.discrete.Delay*

Compute the average of a BaseVar over a period of time or a number of samples.

**check\_var**(dae\_t, \*args, \*\*kwargs)

This function is called in *l\_update\_var* before evaluating equations.

It should update internal flags only.

```
class andes.core.discrete.DeadBand(u,    center,    lower,    upper,    enable=True,
                                   equal=True,    zu=0.0,    zl=0.0,    zi=0.0,
                                   name=None, tex_name=None, info=None)
```

Bases: *andes.core.discrete.Limiter*

The basic deadband type.

## Parameters

**u** [NumParam] The pre-deadband input variable

**center** [NumParam] Neutral value of the output

**lower** [NumParam] Lower bound

**upper** [NumParam] Upper bound

**enable** [bool] Enabled if True; Disabled and works as a pass-through if False.

## Notes

Input changes within a deadband will incur no output changes. This component computes and exports three flags.

### Three flags computed from the current input:

- **zl**: True if the input is below the lower threshold
- **zi**: True if the input is within the deadband
- **zu**: True if is above the lower threshold

Initial condition:

All three flags are initialized to zero. All flags are updated during *check\_var* when enabled. If the deadband component is not enabled, all of them will remain zero.

## Examples

Exported deadband flags need to be used in the algebraic equation corresponding to the post-deadband variable. Assume the pre-deadband input variable is *var\_in* and the post-deadband variable is *var\_out*. First, define a deadband instance *db* in the model using

```
self.db = DeadBand(u=self.var_in, center=self.dbc,
                  lower=self.dbl, upper=self.dbu)
```

To implement a no-memory deadband whose output returns to center when the input is within the band, the equation for *var* can be written as

```
var_out.e_str = 'var_in * (1 - db_zi) + \
                (dbc * db_zi) - var_out'
```

**check\_var** (\*args, \*\*kwargs)

## Notes

Updates three flags: **zi**, **zu**, **zl** based on the following rules:

**zu**: 1 if  $u > \text{upper}$ ; 0 otherwise.

**zl**: 1 if  $u < \text{lower}$ ; 0 otherwise.

**zi**:  $\text{not}(\text{zu or zl})$ ;

**class** `andes.core.discrete.DeadBandRT` (*u*, *center*, *lower*, *upper*, *enable=True*)

Bases: `andes.core.discrete.DeadBand`

Deadband with flags for directions of return.

### Parameters

- u** [NumParam] The pre-deadband input variable
- center** [NumParam] Neutral value of the output
- lower** [NumParam] Lower bound
- upper** [NumParam] Upper bound
- enable** [bool] Enabled if True; Disabled and works as a pass-through if False.

### Notes

Input changes within a deadband will incur no output changes. This component computes and exports five flags. The additional two flags on top of *DeadBand* indicate the direction of return:

- **zur**: True if the input is/has been within the deadband and was returned from the upper threshold
- **zlr**: True if the input is/has been within the deadband and was returned from the lower threshold

Initial condition:

All five flags are initialized to zero. All flags are updated during *check\_var* when enabled. If the deadband component is not enabled, all of them will remain zero.

### Examples

To implement a deadband whose output is pegged at the nearest deadband bounds, the equation for *var* can be provided as

```
var_out.e_str = 'var_in * (1 - db_zi) + \
                db_l * db_zlr + \
                db_u * db_zur - var_out'
```

**check\_var** (\*args, \*\*kwargs)

### Notes

Updates five flags: zi, zu, zl; zur, and zlr based on the following rules:

**zu**: 1 if  $u > \text{upper}$ ; 0 otherwise.

**zl**: 1 if  $u < \text{lower}$ ; 0 otherwise.

**zi**:  $\text{not}(\text{zu or zl})$ ;

**zur**:

- set to 1 when  $(\text{previous zu} + \text{present zi} == 2)$
- hold when  $(\text{previous zi} == \text{zi})$
- clear otherwise



**zlr:**

- set to 1 when (previous zl + present zi == 2)
- hold when (previous zi == zi)
- clear otherwise

```
class andes.core.discrete.Delay (u,      mode='step',      delay=0,      name=None,
                                tex_name=None, info=None)
```

Bases: `andes.core.discrete.Discrete`

Delay class to memorize past variable values.

Delay allows to impose a predefined "delay" (in either steps or seconds) for an input variable. The amount of delay is a scalar and has to be fixed at model definition for now.

```
check_var (dae_t, *args, **kwargs)
```

This function is called in `l_update_var` before evaluating equations.

It should update internal flags only.

```
list2array (n)
```

Allocate memory for storage arrays.

```
class andes.core.discrete.Derivative (u,      name=None,      tex_name=None,
                                       info=None)
```

Bases: `andes.core.discrete.Delay`

Compute the derivative of an algebraic variable using numerical differentiation.

```
check_var (dae_t, *args, **kwargs)
```

This function is called in `l_update_var` before evaluating equations.

It should update internal flags only.

```
class andes.core.discrete.Discrete (name=None,      tex_name=None,      info=None,
                                       no_warn=False)
```

Bases: `object`

Base discrete class.

Discrete classes export flag arrays (usually boolean) .

```
check_eq ()
```

This function is called in `l_check_eq` after updating equations.

It updates internal flags, set differential equations, and record pegged variables.

```
check_var (*args, **kwargs)
```

This function is called in `l_update_var` before evaluating equations.

It should update internal flags only.

```
class_name
```

```
get_names ()
```

Available symbols from this class

**get\_tex\_names()**

Return tex\_names of exported flags.

TODO: Fix the bug described in the warning below.

### Returns

**list** A list of tex\_names for all exported flags.

**Warning:** If underscore `_` appears in both flag `tex_name` and `self.tex_name` (for example, when this discrete is within a block), the exported `tex_name` will become invalid for SymPy. Variable name substitution will fail.

**get\_values()**

**list2array(n)**

**warn\_init\_limit()**

Warn if initialized at limits.

**class** andes.core.discrete.**HardLimiter**(*u*, *lower*, *upper*, *enable*=True, *name*=None, *tex\_name*=None, *info*=None, *no\_upper*=False, *no\_lower*=False, *equal*=True, *zu*=0.0, *zl*=0.0, *zi*=1.0)

Bases: [andes.core.discrete.Limiter](#)

Hard limiter for algebraic or differential variable. This class is an alias of *Limiter*.

**class** andes.core.discrete.**LessThan**(*u*, *bound*, *equal*=False, *enable*=True, *name*=None, *tex\_name*=None, *info*=None, *cache*=False, *z0*=0, *z1*=1)

Bases: [andes.core.discrete.Discrete](#)

Less than (<) comparison function.

Exports two flags: *z1* and *z0*. For elements satisfying the less-than condition, the corresponding *z1* = 1. *z0* is the element-wise negation of *z1*.

### Notes

The default *z0* and *z1*, if not enabled, can be set through the constructor.

**check\_var(\*args, \*\*kwargs)**

If enabled, set flags based on inputs. Use cached values if enabled.

**class** andes.core.discrete.**Limiter**(*u*, *lower*, *upper*, *enable*=True, *name*=None, *tex\_name*=None, *info*=None, *no\_upper*=False, *no\_lower*=False, *equal*=True, *zu*=0.0, *zl*=0.0, *zi*=1.0)

Bases: [andes.core.discrete.Discrete](#)

Base limiter class.

This class compares values and sets limit values. Exported flags are *zi*, *zl* and *zu*.

## Parameters

**u** [BaseVar] Input Variable instance

**lower** [BaseParam] Parameter instance for the lower limit

**upper** [BaseParam] Parameter instance for the upper limit

**no\_lower** [bool] True to only use the upper limit

**no\_upper** [bool] True to only use the lower limit

**equal: bool** True to include equal signs in comparison ( $\geq$  or  $\leq$ ).

**zu** [0 or 1] Default value for *zu* if not enabled

**zl** [0 or 1] Default value for *zl* if not enabled

**zi** [0 or 1] Default value for *zi* if not enabled

## Notes

If not enabled, the default flags are  $zu = zl = 0, zi = 1$ .

## Attributes

**zl** [array-like] Flags of elements violating the lower limit; A array of zeros and/or ones.

**zi** [array-like] Flags for within the limits

**zu** [array-like] Flags for violating the upper limit

**check\_var** (\*args, \*\*kwargs)  
Evaluate the flags.

```
class andes.core.discrete.RateLimiter(u,    lower,    upper,    enable=True,
                                     no_lower=False,    no_upper=False,
                                     lower_cond=None,    upper_cond=None,
                                     name=None, tex_name=None, info=None)
```

Bases: `andes.core.discrete.Discrete`

Rate limiter for a differential variable.

RateLimiter does not export any variable. It directly modifies the differential equation value.

**Warning:** RateLimiter cannot be applied to a state variable that already undergoes an Anti-Windup limiter. Use *AntiWindupRate* for a rate-limited anti-windup limiter.

## Notes

RateLimiter inherits from Discrete to avoid internal naming conflicts with *Limiter*.

**check\_eq()**

This function is called in `l_check_eq` after updating equations.

It updates internal flags, set differential equations, and record pegged variables.

```
class andes.core.discrete.Sampling(u, interval=1.0, offset=0.0, name=None,
                                   tex_name=None, info=None)
```

Bases: `andes.core.discrete.Discrete`

Sample an input variable repeatedly at a given time interval.

```
check_var(dae_t, *args, **kwargs)
```

Check and update the output.

### Notes

Present output stored in `v`. Output of the last step is stored in `_last_v`. Time for the last output is stored in `_last_t`.

Initially, store `v` and `_last_v`.

If time progresses and `dae_t` is a multiple of *period*, update `_last_v` and then `v`. Record `_last_t`.

If time does not progress, update `v`.

If time rewinds, restore `_last_v` to `v`.

```
list2array(n)
```

```
class andes.core.discrete.Selector(*args, fun, tex_name=None, info=None)
```

Bases: `andes.core.discrete.Discrete`

Selection between two variables using the provided reduce function.

The reduce function should take the given number of arguments. An example function is `np.maximum.reduce` which can be used to select the maximum.

Names are in `s0`, `s1`.

**Warning:** A potential bug when more than two inputs are provided, and values in different inputs are equal. Only two inputs are allowed.

See also:

`numpy.ufunc.reduce` NumPy reduce function

`andes.core.block.HVGate`

`andes.core.block.LVGate`

## Notes

A common pitfall is the 0-based indexing in the Selector flags. Note that exported flags start from 0. Namely, *s0* corresponds to the first variable provided for the Selector constructor.

## Examples

Example 1: select the largest value between *v0* and *v1* and put it into *vmax*.

After the definitions of *v0* and *v1*, define the algebraic variable *vmax* for the largest value, and a selector *vs*

```
self.vmax = Algeb(v_str='maximum(v0, v1)',
                  tex_name='v_{max}',
                  e_str='vs_s0 * v0 + vs_s1 * v1 - vmax')

self.vs = Selector(self.v0, self.v1, fun=np.maximum.reduce)
```

The initial value of *vmax* is calculated by `maximum(v0, v1)`, which is the element-wise maximum in SymPy and will be generated into `np.maximum(v0, v1)`. The equation of *vmax* is to select the values based on *vs\_s0* and *vs\_s1*.

**check\_var** (\*args, \*\*kwargs)

Set the i-th variable's flags to 1 if the return of the reduce function equals the i-th input.

```
class andes.core.discrete.SortedLimiter(u, lower, upper, enable=True, n_select:
                                     Optional[int] = None, name=None,
                                     tex_name=None)
```

Bases: `andes.core.discrete.Limiter`

A comparer with the top value selection.

**check\_var** (\*args, \*\*kwargs)

Evaluate the flags.

```
class andes.core.discrete.Switcher(u, options: Union[list, Tuple], name: str = None,
                                  tex_name: str = None, cache=True)
```

Bases: `andes.core.discrete.Discrete`

Switcher based on an input parameter.

The switch class takes one v-provider, compares the input with each value in the option list, and exports one flag array for each option. The flags are 0-indexed.

Exported flags are named with *\_s0*, *\_s1*, ..., with a total number of `len(options)`. See the examples section.

## Notes

Switches needs to be distinguished from Selector.

Switcher is for generating flags indicating option selection based on an input parameter. Selector is for generating flags at run time based on variable values and a selection function.

## Examples

The IEEEEST model takes an input for selecting the signal. Options are 1 through 6. One can construct

```
self.IC = NumParam(info='input code 1-6') # input code
self.SW = Switcher(u=self.IC, options=[1, 2, 3, 4, 5, 6])
```

If the IC values from the data file ends up being

```
self.IC.v = np.array([1, 2, 2, 4, 6])
```

Then, the exported flag arrays will be

```
{'IC_s0': np.array([1, 0, 0, 0, 0]),
 'IC_s1': np.array([0, 1, 1, 0, 0]),
 'IC_s2': np.array([0, 0, 0, 0, 0]),
 'IC_s3': np.array([0, 0, 0, 1, 0]),
 'IC_s4': np.array([0, 0, 0, 0, 0]),
 'IC_s5': np.array([0, 0, 0, 0, 1])
}
```

**check\_var** (\*args, \*\*kwargs)

Set the switcher flags based on inputs. Uses cached flags if cache is set to True.

**list2array** (n)

This forces to evaluate Switcher upon System setup

### 12.1.4 andes.core.model module

Base class for building ANDES models.

**class** andes.core.model.Cache

Bases: object

Class for caching the return value of callback functions.

**add\_callback** (name: str, callback)

Add a cache attribute and a callback function for updating the attribute.

#### Parameters

**name** [str] name of the cached function return value

**callback** [callable] callback function for updating the cached attribute

**refresh** (name=None)

Refresh the cached values

#### Parameters

**name** [str, list, optional] name or list of cached to refresh, by default None for refreshing all

```
class andes.core.model.Documenter (parent)
```

Bases: `object`

Helper class for documenting models.

#### Parameters

**parent** [Model] The *Model* instance to document

```
get (max_width=78, export='plain')
```

Return the model documentation in table-formatted string.

#### Parameters

**max\_width** [int] Maximum table width. Automatically set to 0 if format is `rest`.

**export** [str, ('plain', 'rest')] Export format. Use fancy table if is `rest`.

#### Returns

**str** A string with the documentations.

```
class andes.core.model.Model (system=None, config=None)
```

Bases: `object`

Base class for power system DAE models.

After subclassing *ModelData*, subclass *Model* to complete a DAE model. Subclasses of *Model* defines DAE variables, services, and other types of parameters, in the constructor `__init__`.

## Examples

Take the static PQ as an example, the subclass of *Model*, *PQ*, should look like

```
class PQ(PQData, Model):
    def __init__(self, system, config):
        PQData.__init__(self)
        Model.__init__(self, system, config)
```

Since *PQ* is calling the base class constructors, it is meant to be the final class and not further derived. It inherits from *PQData* and *Model* and must call constructors in the order of *PQData* and *Model*. If the derived class of *Model* needs to be further derived, it should only derive from *Model* and use a name ending with *Base*. See `andes.models.synchronous.GENBASE`.

Next, in *PQ.\_\_init\_\_*, set proper flags to indicate the routines in which the model will be used

```
self.flags.update({'pflow': True})
```

Currently, flags *pflow* and *tds* are supported. Both are *False* by default, meaning the model is neither used in power flow nor time-domain simulation. **A very common pitfall is forgetting to set the flag.**

Next, the group name can be provided. A group is a collection of models with common parameters and variables. Devices *idx* of all models in the same group must be unique. To provide a group name, use

```
self.group = 'StaticLoad'
```

The group name must be an existing class name in `andes.models.group`. The model will be added to the specified group and subject to the variable and parameter policy of the group. If not provided with a group class name, the model will be placed in the *Undefined* group.

Next, additional configuration flags can be added. Configuration flags for models are load-time variables specifying the behavior of a model. It can be exported to an *andes.rc* file and automatically loaded when creating the *System*. Configuration flags can be used in equation strings, as long as they are numerical values. To add config flags, use

```
self.config.add(OrderedDict((( 'pq2z', 1), )))
```

It is recommended to use *OrderedDict* instead of *dict*, although the syntax is verbose. Note that booleans should be provided as integers (1, or 0), since *True* or *False* is interpreted as a string when loaded from the *rc* file and will cause an error.

Next, it's time for variables and equations! The *PQ* class does not have internal variables itself. It uses its *bus* parameter to fetch the corresponding *a* and *v* variables of buses. Equation wise, it imposes an active power and a reactive power load equation.

To define external variables from *Bus*, use

```
self.a = ExtAlgeb(model='Bus', src='a',
                  indexer=self.bus, tex_name=r'\theta')
self.v = ExtAlgeb(model='Bus', src='v',
                  indexer=self.bus, tex_name=r'V')
```

Refer to the subsection Variables for more details.

The simplest *PQ* model will impose constant P and Q, coded as

```
self.a.e_str = "u * p"
self.v.e_str = "u * q"
```

where the *e\_str* attribute is the equation string attribute. *u* is the connectivity status. Any parameter, config, service or variables can be used in equation strings. An addition variable *dae\_t* for the current simulation time can be used if the model has flag *tds*.

The above example is overly simplified. Our *PQ* model wants a feature to switch itself to a constant impedance if the voltage is out of the range (*vmin*, *vmax*). To implement this, we need to introduce a discrete component called *Limiter*, which yields three arrays of binary flags, *zi*, *zl*, and *zu* indicating in range, below lower limit, and above upper limit, respectively.

First, create an attribute *vcmp* as a *Limiter* instance

```
self.vcmp = Limiter(u=self.v, lower=self.vmin, upper=self.vmax,
                    enable=self.config.pq2z)
```

where *self.config.pq2z* is a flag to turn this feature on or off. After this line, we can use *vcmp\_zi*, *vcmp\_zl*, and *vcmp\_zu* in other equation strings.



```

self.a.e_str = "u * (p0 * vcmp_zi + " \
               "p0 * vcmp_zl * (v ** 2 / vmin ** 2) + " \
               "p0 * vcmp_zu * (v ** 2 / vmax ** 2))"

self.v.e_str = "u * (q0 * vcmp_zi + " \
               "q0 * vcmp_zl * (v ** 2 / vmin ** 2) + "\
               "q0 * vcmp_zu * (v ** 2 / vmax ** 2))"

```

Note that *PQ.a.e\_str* can use the three variables from *vcmp* even before defining *PQ.vcmp*, as long as *PQ.vcmp* is defined, because *vcmp\_zi* is just a string literal in *e\_str*.

The two equations above implements a piecewise power injection equation. It selects the original power demand if within range, and uses the calculated power when out of range.

Finally, to let ANDES pick up the model, the model name needs to be added to *models/\_\_init\_\_.py*. Follow the examples in the *OrderedDict*, where the key is the file name, and the value is the class name.

### Attributes

**num\_params** [OrderedDict] {name: instance} of numerical parameters, including internal and external ones

### **a\_reset()**

Reset addresses to empty and reset flags.address to `False`.

### **alter** (*src*, *idx*, *value*)

Alter input parameter or service values.

If operates on a parameter, the input should be in the same base as that in the input file. This function will convert the new value to system-base per unit.

### Parameters

**src** [str] The parameter name to alter

**idx** [str, float, int] The device to alter

**value** [float] The desired value

### **class\_name**

Return the class name

### **doc** (*max\_width*=78, *export*='plain')

Retrieve model documentation as a string.

### **e\_clear()**

Clear equation value arrays associated with all internal variables.

### **f\_numeric** (\*\**kwargs*)

Custom fcall functions. Modify equations directly.

### **f\_update()**

Evaluate differential equations.

## Notes

In-place equations: added to the corresponding DAE array. Non-inplace equations: in-place set to internal array to overwrite old values (and avoid clearing).

**g\_numeric** (\*\*kwargs)

Custom gcall functions. Modify equations directly.

**g\_update** ()

Evaluate algebraic equations.

**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]`

**get\_init\_order** ()

Get variable initialization order and send to *logger.info*.

**get\_inputs** (refresh=False)

Get an OrderedDict of the inputs to the numerical function calls.

## Parameters

**refresh** [bool] Refresh the values in the dictionary. This is only used when the memory address of arrays changed. After initialization, all array assignments are inplace. To avoid overhead, refresh should not be used after initialization.

## Returns

**OrderedDict** The input name and value array pairs in an OrderedDict

## Notes

*dae.t* is now a `numpy.ndarray` which has stable memory. There is no need to refresh *dat\_t* in this version.

**get\_md5** ()

Return the md5 hash of concatenated equation strings.

**get\_times()**

Get event switch\_times from *TimerParam*.

**Returns**

**list** A list containing all switching times defined in *TimerParams*

**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

**init(routine)**

Numerical initialization of a model.

Initialization sequence: 1. Sequential initialization based on the order of definition 2. Use Newton-Krylov method for iterative initialization 3. Custom init

**init\_iter()**

Solve the initialization equation using the Newton-Krylov method.

**j\_numeric(\*\*kwargs)**

Custom numeric update functions.

This function should append indices to *\_ifx*, *\_jfx*, and append anonymous functions to *\_vfx*. It is only called once by *store\_sparse\_pattern*.

**j\_update()**

Update Jacobian elements.

Values are stored to `Model.triplets[jname]`, where *jname* is a jacobian name.

**Returns**

**None**

**l\_check\_eq()**

Call the *check\_eq* method of discrete components to update equation-dependent flags.

This function should be called after equation updates. AntiWindup limiters use it to append pegged states to the *x\_set* list.

**Returns**

**None**

**l\_update\_var(dae\_t)**

Call the *check\_var* method of discrete components to update the internal status flags.

The function is variable-dependent and should be called before updating equations.

**Returns**

**None**

**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 `[:]`).

**numba\_jitify** (*parallel=False, cache=False*)

Optionally convert *self.calls.f* and *self.calls.g* to JIT compiled functions.

This function can be turned on by setting `System.config.numba` to 1.

**Warning:** This feature is experimental and does not guarantee a speed up. In fact, the program will likely end up slower due to compilation.

**post\_init\_check()**

Post init checking. Warns if values of *InitChecker* is not True.

**prepare** (*quick=False*)

Symbolic processing and code generation.

**refresh\_inputs()**

This is the helper function to refresh inputs.

The functions collects objects into `OrderedDict` and store to *self.\_input* and *self.\_input\_z*.

**Returns**

None

**refresh\_inputs\_arg()**

Refresh inputs for each function with individual argument list.

**s\_numeric** (*\*\*kwargs*)

Custom service value functions. Modify `Service.v` directly.

**s\_numeric\_var** (*\*\*kwargs*)

Custom variable service value functions. Modify `VarService.v` directly.

This custom numerical function is evaluated at each step/iteration before equation update.

**s\_update()**

Update service equation values.

This function is only evaluated at initialization. Service values are updated sequentially. The `v` attribute of services will be assigned at a new memory address.

**s\_update\_post()**

Update post-initialization services.

**s\_update\_var()**

Update `VarService`.

**set** (*src, idx, attr, value*)

Set the value of an attribute of a model property.

Performs `self.<src>.<attr>[idx] = value`.

### 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.

### `set_in_use()`

Set the *in\_use* attribute. Called at the end of `System.collect_ref`.

This function is overloaded by models with *BackRef* to disable calls when no model is referencing. Models with no back references will have internal variable addresses assigned but external addresses being empty.

For internal equations that has external variables, the row indices will be non-zeros, while the col indices will be empty, which causes an error when updating Jacobians.

Setting *self.in\_use* to False when `len(back_ref_instance.v) == 0` avoids this error. See COI.

### `store_sparse_pattern()`

Store rows and columns of the non-zeros in the Jacobians for building the sparsity pattern.

This function converts the internal 0-indexed equation/variable address to the numerical addresses for the loaded system.

Calling sequence: For each Jacobian name, *fx*, *fy*, *gx* and *gy*, store by a) generated constant and variable Jacobians c) user-provided constant and variable Jacobians, d) user-provided block constant and variable Jacobians

### Notes

If *self.n* == 0, skipping this function will avoid appending empty lists/arrays and non-empty values, which, as a combination, is not accepted by *cvxopt.spmatrix*.

### `switch_action(dae_t)`

Call the switch actions.

### Parameters

**dae\_t** [float] Current simulation time

### Returns

None

**Warning:** Timer exported from blocks are supposed to work but have not been tested.

**v\_numeric** (\*\*kwargs)

Custom variable initialization function.

**class** andes.core.model.ModelCall

Bases: `object`

Class for storing generated function calls and Jacobians.

**append\_ijv** (j\_full\_name, ii, jj, vv)

**clear\_ijv** ()

**zip\_ijv** (j\_full\_name)

Return a zipped iterator for the rows, cols and vals for the specified matrix name.

**class** andes.core.model.ModelData (\*args, \*\*kwargs)

Bases: `object`

Class for holding parameter data for a model.

This class is designed to hold the parameter data separately from model equations. Models should inherit this class to define the parameters from input files.

Inherit this class to create the specific class for holding input parameters for a new model. The recommended name for the derived class is the model name with `Data`. For example, data for *GENCLS* should be named *GENCLSDData*.

Parameters should be defined in the `__init__` function of the derived class.

Refer to *andes.core.param* for available parameter types.

## Notes

Two default parameters, *u* (connection status of type *andes.core.param.NumParam*), and *name* (device name of type *andes.core.param.DataParam*) are pre-defined in *ModelData*, and will be inherited by all models.

## Examples

If we want to build a class *PQData* (for static PQ load) with three parameters, *Vn*, *p0* and *q0*, we can use the following

```
from andes.core.model import ModelData, Model
from andes.core.param import IdxParam, NumParam

class PQData(ModelData):
    super().__init__()
    self.Vn = NumParam(default=110,
                        info="AC voltage rating",
```

(continues on next page)

(continued from previous page)

```

        unit='kV', non_zero=True,
        tex_name=r'V_n')
self.p0 = NumParam(default=0,
                    info='active power load in system base',
                    tex_name=r'p_0', unit='p.u.')
self.q0 = NumParam(default=0,
                    info='reactive power load in system base',
                    tex_name=r'q_0', unit='p.u.')

```

In this example, all the three parameters are defined as `andes.core.param.NumParam`. In the full `PQData` class, other types of parameters also exist. For example, to store the idx of *owner*, `PQData` uses

```
self.owner = IdxParam(model='Owner', info="owner idx")
```

### Attributes

**cache** A cache instance for different views of the internal data.

**flags** [dict] Flags to control the routine and functions that get called. If the model is using user-defined numerical calls, set *f\_num*, *g\_num* and *j\_num* properly.

**add** (\*\*kwargs)

Add a device (an instance) to this model.

### Parameters

**kwargs** model parameters are collected into the kwargs dictionary

**Warning:** This function is not intended to be used directly. Use the `add` method from `System` so that the index can be registered correctly.

**as\_df** ()

Export all parameters as a `pandas.DataFrame` object. This function utilizes `as_dict` for preparing data.

### Returns

**DataFrame** A dataframe containing all model data. An *uid* column is added.

**as\_df\_in** ()

Export all parameters from original input (*vin*) as a `pandas.DataFrame`. This function utilizes `as_dict` for preparing data.

### Returns

**DataFrame** A `pandas.DataFrame` containing all model data. An *uid* column is prepended.

**as\_dict** (*vin=False*)

Export all parameters as a dict.

**Returns**

**dict** a dict with the keys being the *ModelData* parameter names and the values being an array-like of data in the order of adding. An additional *uid* key is added with the value default to range(n).

**find\_idx** (*keys*, *values*, *allow\_none=False*, *default=False*)

Find *idx* of devices whose values match the given pattern.

**Parameters**

**keys** [str, array-like, Sized] A string or an array-like of strings containing the names of parameters for the search criteria

**values** [array, array of arrays, Sized] Values for the corresponding key to search for. If *keys* is a str, *values* should be an array of elements. If *keys* is a list, *values* should be an array of arrays, each corresponds to the key.

**allow\_none** [bool, Sized] Allow key, value to be not found. Used by groups.

**default** [bool] Default idx to return if not found (missing)

**Returns**

**list** indices of devices

**find\_param** (*prop*)

Find params with the given property and return in an OrderedDict.

**Parameters**

**prop** [str] Property name

**Returns**

**OrderedDict**

**class** andes.core.model.SymProcessor (*parent*)

Bases: `object`

A helper class for symbolic processing and code generation.

**Parameters**

**parent** [Model] The *Model* instance to document

**Attributes**

**xy** [sympy.Matrix] variables pretty print in the order of State, ExtState, Algeb, ExtAlgeb

**f** [sympy.Matrix] differential equations pretty print

**g** [sympy.Matrix] algebraic equations pretty print

**df** [sympy.SparseMatrix] df /d (xy) pretty print

**dg** [sympy.SparseMatrix] dg /d (xy) pretty print



**inputs\_dict** [OrderedDict] All possible symbols in equations, including variables, parameters, discrete flags, and config flags. It has the same variables as what `get_inputs()` returns.

**vars\_dict** [OrderedDict] variable-only symbols, which are useful when getting the Jacobian matrices.

**non\_vars\_dict** [OrderedDict] symbols in `input_syms` but not in `var_syms`.

**generate\_equations()**

**generate\_init()**

Generate lambda functions for initial values.

**generate\_jacobians()**

Generate Jacobians and store to corresponding triplets.

The internal indices of equations and variables are stored, alongside the lambda functions.

For example, `dg/dy` is a sparse matrix whose elements are `(row, col, val)`, where `row` and `col` are the internal indices, and `val` is the numerical lambda function. They will be stored to

`row -> self.calls._igy col -> self.calls._jgy val -> self.calls._vgy`

**generate\_pretty\_print()**

Generate pretty print variables and equations.

**generate\_pycode()**

Create output source code file for generated code. NOT WORKING NOW.

**generate\_services()**

Generate calls for services, including `ConstService`, `VarService` among others.

**generate\_symbols()**

Generate symbols for symbolic equation generations.

This function should run before other generate equations.

### Attributes

**inputs\_dict** [OrderedDict] name-symbol pair of all parameters, variables and configs

**vars\_dict** [OrderedDict] name-symbol pair of all variables, in the order of `(states_and_ext + algebs_and_ext)`

**non\_vars\_dict** [OrderedDict] name-symbol pair of all non-variables, namely, `(inputs_dict - vars_dict)`

### 12.1.5 andes.core.param module

```
class andes.core.param.BaseParam (default: Union[float, str, int, None] = None, name:
                                Optional[str] = None, tex_name: Optional[str]
                                = None, info: Optional[str] = None, unit: Op-
                                tional[str] = None, mandatory: bool = False, ex-
                                port: bool = True)
```

Bases: `object`

The base parameter class.

This class provides the basic data structure and interfaces for all types of parameters. Parameters are from input files and in general constant once initialized.

Subclasses should overload the  $n()$  method for the total count of elements in the value array.

#### Parameters

**default** [str or float, optional] The default value of this parameter if None is provided

**name** [str, optional] Parameter name. If not provided, it will be automatically set to the attribute name defined in the owner model.

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

**info** [str, optional] Descriptive information of parameter

**mandatory** [bool] True if this parameter is mandatory

**export** [bool] True if the parameter will be exported when dumping data into files. True for most parameters. False for `BackRef`.

**Warning:** The most distinct feature of `BaseParam`, `DataParam` and `IdxParam` is that values are stored in a list without conversion to array. `BaseParam`, `DataParam` or `IdxParam` are **not allowed** in equations.

#### Attributes

**v** [list] A list holding all the values. The `BaseParam` class does not convert the `v` attribute into NumPy arrays.

**property** [dict] A dict containing the truth values of the model properties.

**add** (*value=None*)

Add a new parameter value (from a new device of the owner model) to the `v` list.

#### Parameters

**value** [str or float, optional] Parameter value of the new element. If None, the default will be used.

## Notes

If the value is `math.nan`, it will set to `None`.

### **class\_name**

Return the class name.

### **get\_names()**

Return `self.name` in a list.

This is a helper function to provide the same API as blocks or discrete components.

### **Returns**

**list** A list only containing the name of the parameter

### **get\_property(property\_name: str)**

Check the boolean value of the given property. If the property does not exist in the dictionary, `False` will be returned.

### **Parameters**

**property\_name** [str] Property name

### **Returns**

**The truth value of the property.**

### **n**

Return the count of elements in the value array.

```
class andes.core.param.DataParam (default: Union[float, str, int, None] = None, name:
                                Optional[str] = None, tex_name: Optional[str]
                                = None, info: Optional[str] = None, unit: Op-
                                tional[str] = None, mandatory: bool = False, ex-
                                port: bool = True)
```

Bases: [andes.core.param.BaseParam](#)

An alias of the *BaseParam* class.

This class is used for string parameters or non-computational numerical parameters. This class does not provide a *to\_array* method. All input values will be stored in *v* as a list.

**See also:**

[andes.core.param.BaseParam](#) Base parameter class

```
class andes.core.param.ExtParam (model: str, src: str, indexer=None, dtype=<class
                                'float'>, allow_none=False, default=0.0, **kwargs)
Bases: andes.core.param.NumParam
```

A parameter whose values are retrieved from an external model or group.

### **Parameters**

**model** [str] Name of the model or group providing the original parameter

**src** [str] The source parameter name

**indexer** [BaseParam] A parameter defined in the model defining this ExtParam instance. *indexer.v* should contain indices into *model.src.v*. If is None, the source parameter values will be fully copied. If *model* is a group name, the indexer cannot be None.

### Attributes

**parent\_model** [Model] The parent model providing the original parameter.

**add** (*value=None*)

ExtParam has an empty *add* method.

**link\_external** (*ext\_model*)

Update parameter values provided by external models. This needs to be called before pu conversion.

### Parameters

**ext\_model** [Model, Group] Instance of the parent model or group, provided by the System calling this method.

**restore** ()

ExtParam has an empty *restore* method

**to\_array** ()

Convert to array when *d\_type* is not str

```
class andes.core.param.IdxParam (default: Union[float, str, int, None] = None, name:
    Optional[str] = None, tex_name: Optional[str] =
    None, info: Optional[str] = None, unit: Op-
    tional[str] = None, mandatory: bool = False,
    unique: bool = False, export: bool = True, model:
    Optional[str] = None)
```

Bases: [andes.core.param.BaseParam](#)

An alias of *BaseParam* with an additional storage of the owner model name

This class is intended for storing *idx* into other models. It can be used in the future for data consistency check.

### Notes

This will be useful when, for example, one connects two TGs to one SynGen.

### Examples

A PQ model connected to Bus model will have the following code

```
class PQModel(...):
    def __init__(...):
        ...
        self.bus = IdxParam(model='Bus')
```

**add** (*value=None*)

Add a new parameter value (from a new device of the owner model) to the *v* list.

### Parameters

**value** [str or float, optional] Parameter value of the new element. If None, the default will be used.

### Notes

If the value is `math.nan`, it will set to None.

```
class andes.core.param.NumParam (default: Union[float, str, Callable, None] = None,
                                name: Optional[str] = None, tex_name: Op-
                                tional[str] = None, info: Optional[str] = None, unit:
                                Optional[str] = None, vrange: Union[List[T], Tu-
                                ple, None] = None, non_zero: bool = False, posi-
                                tive: bool = False, mandatory: bool = False, power:
                                bool = False, ipower: bool = False, voltage: bool =
                                False, current: bool = False, z: bool = False, y: bool
                                = False, r: bool = False, g: bool = False, dc_voltage:
                                bool = False, dc_current: bool = False, export: bool
                                = True)
```

Bases: `andes.core.param.BaseParam`

A computational numerical parameter.

Parameters defined using this class will have their *v* field converted to a NumPy array after adding. The original input values will be copied to *vin*, and the system-base per-unit conversion coefficients (through multiplication) will be stored in *pu\_coeff*.

### Parameters

**default** [str or float, optional] The default value of this parameter if no value is provided

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

**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

**mandatory** [bool] True if this parameter is mandatory

**unit** [str, optional] Unit of the parameter

**vrange** [list, tuple, optional] Typical value range

### Other Parameters

**Sn** [str] Name of the parameter for the device base power.

**Vn** [str] Name of the parameter for the device base voltage.

**non\_zero** [bool] True if this parameter must be non-zero

**positive: bool** True if this parameter must be positive

**mandatory** [bool] True if this parameter must not be None

**power** [bool] True if this parameter is a power per-unit quantity under the device base

**ipower** [bool] True if this parameter is an inverse-power per-unit quantity under the device base

**voltage** [bool] True if the parameter is a voltage pu quantity under the device base

**current** [bool] True if the parameter is a current pu quantity under the device base

**z** [bool] True if the parameter is an AC impedance pu quantity under the device base

**y** [bool] True if the parameter is an AC admittance pu quantity under the device base

**r** [bool] True if the parameter is a DC resistance pu quantity under the device base

**g** [bool] True if the parameter is a DC conductance pu quantity under the device base

**dc\_current** [bool] True if the parameter is a DC current pu quantity under device base

**dc\_voltage** [bool] True if the parameter is a DC voltage pu quantity under device base

**add** (*value=None*)

Add a value to the parameter value list.

In addition to `BaseParam.add`, this method checks for non-zero property and reset to default if is zero.

**See also:**

[`BaseParam.add`](#) the add method of `BaseParam`

**restore** ()

Restore parameter to the original input by copying `self.vin` to `self.v`.

`pu_coeff` will not be overwritten.

**set\_pu\_coeff** (*coeff*)

Store p.u. conversion coefficient into `self.pu_coeff` and calculate the system-base per unit with `self.v = self.vin * self.pu_coeff`.

This function must be called after `self.to_array`.

### Parameters

**coeff** [np.ndarray] An array with the pu conversion coefficients

**to\_array()**

Convert *v* to `np.ndarray` after adding elements. Store a copy if the input in *vin*. Set `pu_coeff` to all ones.

The conversion enables array-based calculation.

**Warning:** After this call, *add* will not be allowed, because data will not be copied over to *vin*.

```
class andes.core.param.TimerParam(callback: Optional[Callable] = None, default:
                                Union[float, str, Callable, None] = None, name:
                                Optional[str] = None, tex_name: Optional[str]
                                = None, info: Optional[str] = None, unit: Op-
                                tional[str] = None, non_zero: bool = False,
                                mandatory: bool = False, export: bool = True)
```

Bases: `andes.core.param.NumParam`

A parameter whose values are event occurrence times during the simulation.

The constructor takes an additional Callable *self.callback* for the action of the event. *TimerParam* has a default value of -1, meaning deactivated.

## Examples

A connectivity status toggler class *Toggler* takes a parameter *t* for the toggle time. Inside *Toggler*.`__init__`, one would have

```
self.t = TimerParam()
```

The *Toggler* class also needs to define a method for toggling the connectivity status

```
def _u_switch(self, is_time: np.ndarray):
    action = False
    for i in range(self.n):
        if is_time[i] and (self.u.v[i] == 1):
            instance = self.system.__dict__[self.model.v[i]]
            # get the original status and flip the value
            u0 = instance.get(src='u', attr='v', idx=self.dev.v[i])
            instance.set(src='u',
                        attr='v',
                        idx=self.dev.v[i],
                        value=1-u0)
        action = True
    return action
```

Finally, in *Toggler*.`__init__`, assign the function as the callback for *self.t*

```
self.t.callback = self._u_switch
```

**is\_time** (*dae\_t*)

Element-wise check if the DAE time is the same as the parameter value. The current implementation uses *np.isclose*

**Parameters**

**dae\_t** [float] Current simulation time

**Returns**

**np.ndarray** The array containing the truth value of if the DAE time is close to the parameter value.

**See also:**

**numpy.isclose** See NumPy.isclose for the warning on absolute tolerance

### 12.1.6 andes.core.service module

**class** `andes.core.service.ApplyFunc` (*u*, *func*, *name=None*, *tex\_name=None*,  
*info=None*, *cache=True*)

Bases: `andes.core.service.BaseService`

Class for applying a numerical function on a parameter..

**Parameters**

**u** Input parameter

**func** A condition function that returns True or False.

**Warning:** This class is not ready.

**v**

**class** `andes.core.service.BackRef` (*\*\*kwargs*)

Bases: `andes.core.service.BaseService`

A special type of reference collector.

*BackRef* is used for collecting device indices of other models referencing the parent model of the *BackRef*. The *v* field will be a list of lists, each containing the *idx* of other models referencing each device of the parent model.

*BackRef* can be passed as indexer for params and vars, or shape for *NumReduce* and *NumRepeat*. See examples for illustration.

**See also:**

**`andes.core.service.NumReduce`** A more complete example using *BackRef* to build the COI model



## Examples

A Bus device has an *IdxParam* of *area*, storing the *idx* of area to which the bus device belongs. In `Bus.__init__()`, one has

```
self.area = IdxParam(model='Area')
```

Suppose *Bus* has the following data

idx	area	Vn
1	1	110
2	2	220
3	1	345
4	1	500

The Area model wants to collect the indices of Bus devices which points to the corresponding Area device. In `Area.__init__`, one defines

```
self.Bus = BackRef()
```

where the member attribute name *Bus* needs to match exactly model name that *Area* wants to collect *idx* for. Similarly, one can define `self.ACTopology = BackRef()` to collect devices in the *ACTopology* group that references *Area*.

The collection of *idx* happens in `andes.system.System._collect_ref_param()`. It has to be noted that the specific *Area* entry must exist to collect model idx-dx referencing it. For example, if *Area* has the following data

```
idx
1
```

Then, only Bus 1, 3, and 4 will be collected into `self.Bus.v`, namely, `self.Bus.v == [ [1, 3, 4] ]`.

If *Area* has data

```
idx
1
2
```

Then, `self.Bus.v` will end up with `[ [1, 3, 4], [2] ]`.

```
class andes.core.service.BaseService(name: str = None, tex_name: str = None,
                                     info: str = None, vtype: Type[CT_co] =
                                     None)
```

Bases: `object`

Base class for Service.

Service is a v-provider type for holding internal and temporary values. Subclasses need to implement *v* as a member attribute or using a property decorator.

**Parameters**

**name** [str] Instance name

**Attributes**

**owner** [Model] The hosting/owner model instance

**class\_name**

Return the class name

**get\_names ()**

Return *name* in a list

**Returns**

**list** A list only containing the name of the service variable

**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

```
class andes.core.service.ConstService (v_str: Optional[str] = None, v_numeric:
                                     Optional[Callable] = None, vtype: Op-
                                     tional[type] = None, name: Optional[str] =
                                     None, tex_name=None, info=None)
```

Bases: `andes.core.service.BaseService`

A type of Service that stays constant once initialized.

ConstService are usually constants calculated from parameters. They are only evaluated once in the initialization phase before variables are initialized. Therefore, uninitialized variables must not be used in `v_str`.

**Parameters**

**name** [str] Name of the ConstService

**v\_str** [str] An equation string to calculate the variable value.

**v\_numeric** [Callable, optional] A callable which returns the value of the ConstService

**Attributes**

**v** [array-like or a scalar] ConstService value

**assign\_memory (n)**

Assign memory for `self.v` and set the array to zero.

```
class andes.core.service.CurrentSign (bus,      bus1,      bus2,      name=None,
                                     tex_name=None, info=None)
```

Bases: `andes.core.service.ConstService`

Service for computing the sign of the current flowing through a series device.

With a given line connecting *bus1* and *bus2*, one can compute the current flow using  $(v1 \cdot \exp(1j \cdot a1) - v2 \cdot \exp(1j \cdot a2)) / (r + jx)$  whose value is the outflow on *bus1*.

*CurrentSign* can be used to compute the sign to be multiplied depending on the observing bus. For each value in *bus*, the sign will be +1 if it appears in *bus1* or -1 otherwise.

```
bus1          bus2
 *----->>-----*
bus (+)       bus (-)
```

**check** (*\*\*kwargs*)

```
class andes.core.service.DataSelect (optional, fallback, name: Optional[str] =
                                     None, tex_name: Optional[str] = None, info:
                                     Optional[str] = None)
```

Bases: *andes.core.service.BaseService*

Class for selecting values for optional DataParam or NumParam.

This service is a v-provider that uses optional DataParam if available with a fallback.

DataParam will be tested for *None*, and NumParam will be tested with *np.isnan()*.

## Notes

An use case of DataSelect is remote bus. One can do

```
self.buss = DataSelect(option=self.busr, fallback=self.bus)
```

Then, pass *self.buss* instead of *self.bus* as indexer to retrieve voltages.

Another use case is to allow an optional turbine rating. One can do

```
self.Tn = NumParam(default=None)
self.Sg = ExtParam(...)
self.Sn = DataSelect(Tn, Sg)
```

**v**

```
class andes.core.service.DeviceFinder (u, link, idx_name, name=None,
                                       tex_name=None, info=None)
```

Bases: *andes.core.service.BaseService*

Service for finding indices of optionally linked devices.

If not provided, *DeviceFinder* will add devices at the beginning of *System.setup*.

## Examples

IEEEEST stabilizer takes an optional *busf* (IdxParam) for specifying the connected BusFreq, which is needed for mode 6. To avoid reimplementing *BusFreq* within IEEEEST, one can do

```
self.busfreq = DeviceFinder(self.busf, link=self.buss, idx_name='bus')
```

where *self.busf* is the optional input, *self.buss* is the bus indices that *busf* should measure, and *idx\_name* is the name of a *BusFreq* parameter through which the measured bus indices are specified. For each *None* values in *self.busf*, a *BusFreq* is created to measure the corresponding bus in *self.buss*.

That is, *BusFreq[idx\_name].v* = [*link*]. *DeviceFinder* will find / create *BusFreq* devices so that the returned list of *BusFreq* indices are connected to *self.buss*, respectively.

```
find_or_add(system)
```

**v**

```
class andes.core.service.EventFlag(u, vtype: Optional[type] = None, name:
                                Optional[str] = None, tex_name=None,
                                info=None)
```

Bases: *andes.core.service.VarService*

Service to flag events.

*EventFlag.v* stores the values of the input variable from the previous iteration/step.

```
check(**kwargs)
```

```
class andes.core.service.ExtService(model: str, src: str, indexer:
                                Union[andes.core.param.BaseParam,
                                andes.core.service.BaseService], attr: str =
                                'v', allow_none: bool = False, default=0,
                                name: str = None, tex_name: str = None,
                                vtype=None, info: str = None)
```

Bases: *andes.core.service.BaseService*

Service constants whose value is from an external model or group.

### Parameters

**src** [str] Variable or parameter name in the source model or group

**model** [str] A model name or a group name

**indexer** [IdxParam or BaseParam] An "Indexer" instance whose *v* field contains the *idx* of devices in the model or group.

### Examples

A synchronous generator needs to retrieve the *p* and *q* values from static generators for initialization. *ExtService* is used for this purpose.

In a synchronous generator, one can define the following to retrieve *StaticGen.p* as *p0*:

```
class GENCLSMModel(Model):
    def __init__(...):
        ...
```

(continues on next page)

(continued from previous page)

```

self.p0 = ExtService(src='p',
                    model='StaticGen',
                    indexer=self.gen,
                    tex_name='P_0')

```

**assign\_memory** (*n*)

Assign memory for `self.v` and set the array to zero.

**link\_external** (*ext\_model*)

Method to be called by `System` for getting values from the external model or group.

**Parameters**

**ext\_model** An instance of a model or group provided by `System`

```

class andes.core.service.ExtendedEvent (u, t_ext: Union[int, float, andes.core.param.BaseParam, andes.core.service.BaseService] = 0.0, trig: str = 'rise', enable=True, v_disabled=0, extend_only=False, vtype: Optional[type] = None, name: Optional[str] = None, tex_name=None, info=None)

```

Bases: `andes.core.service.VarService`

Service to flag events that extends for period of time after event disappears.

*EventFlag.v* stores the flags whether the extended time has completed. Outputs will become 1 once then event starts until the extended time ends.

**Parameters**

**trig** [str, rise, fall] Triggering edge for the inception of an event. *rise* by default.

**enable** [bool or v-provider] If disabled, the output will be *v\_disabled*

**extend\_only** [bool] Only output during the extended period, not the event period.

**Warning:** The performance of this class needs to be optimized.

**assign\_memory** (*n*)

Assign memory for `self.v` and set the array to zero.

**check** (*\*\*kwargs*)

```

class andes.core.service.FlagCondition (u, func, flag=1, name=None, tex_name=None, info=None, cache=True)

```

Bases: `andes.core.service.BaseService`

Class for flagging values based on a condition function.

By default, values whose condition function output equal that equal to `True/1` will be flagged as *1*. *0* otherwise.

**Parameters**

**u** Input parameter

**func** A condition function that returns True or False.

**flag** [1 by default, only 0 or 1 is accepted.] The flag for the inputs whose condition output is True.

**Warning:** This class is not ready.

*FlagCondition* can only be applied to *BaseParam* with *cache=True*. Applying to *Service* will fail unless *cache* is False (at a performance cost).

**v**

```
class andes.core.service.FlagGreaterThan(u, value=0.0, flag=1, equal=False,
                                         name=None, tex_name=None,
                                         info=None, cache=True)
```

Bases: *andes.core.service.FlagCondition*

Service for flagging parameters > or >= the given value element-wise.

Parameters that satisfy the comparison ( $u >$  or  $\geq$  value) will be flagged as *flag* (1 by default).

```
class andes.core.service.FlagLessThan(u, value=0.0, flag=1, equal=False,
                                       name=None, tex_name=None, info=None,
                                       cache=True)
```

Bases: *andes.core.service.FlagCondition*

Service for flagging parameters < or <= the given value element-wise.

Parameters that satisfy the comparison ( $u <$  or  $\leq$  value) will be flagged as *flag* (1 by default).

```
class andes.core.service.FlagValue(u, value, flag=0, name=None, tex_name=None,
                                   info=None, cache=True)
```

Bases: *andes.core.service.BaseService*

Class for flagging values that equal to the given value.

By default, values that equal to *value* will be flagged as 0. Non-matching values will be flagged as 1.

**Parameters**

**u** Input parameter

**value** Value to flag. Can be None, string, or a number.

**flag** [0 by default, only 0 or 1 is accepted.] The flag for the matched ones

**Warning:** *FlagNotNone* can only be applied to *BaseParam* with *cache=True*. Applying to *Service* will fail unless *cache* is False (at a performance cost).

**v**

**class** `andes.core.service.IdxRepeat` (*u, ref, \*\*kwargs*)

Bases: `andes.core.service.OperationService`

Helper class to repeat IdxParam.

This class has the same functionality as `andes.core.service.NumRepeat` but only operates on IdxParam, DataParam or NumParam.

**v**

Return values stored in `self._v`. May be overloaded by subclasses.

**class** `andes.core.service.InitChecker` (*u, lower=None, upper=None, equal=None, not\_equal=None, enable=True, error\_out=False, \*\*kwargs*)

Bases: `andes.core.service.OperationService`

Class for checking init values against known typical values.

Instances will be stored in `Model.services_post` and `Model.services_ichk`, which will be checked in `Model.post_init_check()` after initialization.

#### Parameters

**u** v-provider to be checked

**lower** [float, BaseParam, BaseVar, BaseService] lower bound

**upper** [float, BaseParam, BaseVar, BaseService] upper bound

**equal** [float, BaseParam, BaseVar, BaseService] values that the value from `v_str` should equal

**not\_equal** [float, BaseParam, BaseVar, BaseService] values that should not equal

**enable** [bool] True to enable checking

#### Examples

Let's say generator excitation voltages are known to be in the range of 1.6 - 3.0 per unit. One can add the following instance to *GENBase*

```
self._vfc = InitChecker(u=self.vf,
                        info='vf range',
                        lower=1.8,
                        upper=3.0,
                        )
```

*lower* and *upper* can also take v-providers instead of float values.

One can also pass float values from Config to make it adjustable as in our implementation of `GENBase._vfc`.

**check()**

Check the bounds and equality conditions.

```
class andes.core.service.NumReduce(u, ref: andes.core.service.BackRef, fun:
                                Callable, name=None, tex_name=None,
                                info=None, cache=True)
```

Bases: `andes.core.service.OperationService`

A helper Service type which reduces a linearly stored 2-D ExtParam into 1-D Service.

NumReduce works with ExtParam whose `v` field is a list of lists. A reduce function which takes an array-like and returns a scalar need to be supplied. NumReduce calls the reduce function on each of the lists and return all the scalars in an array.

### Parameters

**u** [ExtParam] Input ExtParam whose `v` contains linearly stored 2-dimensional values

**ref** [BackRef] The BackRef whose 2-dimensional shapes are used for indexing

**fun** [Callable] The callable for converting a 1-D array-like to a scalar

### Examples

Suppose one wants to calculate the mean value of the `Vn` in one Area. In the `Area` class, one defines

```
class AreaModel(...):
    def __init__(...):
        ...
        # backward reference from `Bus`
        self.Bus = BackRef()

        # collect the Vn in an 1-D array
        self.Vn = ExtParam(model='Bus',
                           src='Vn',
                           indexer=self.Bus)

        self.Vn_mean = NumReduce(u=self.Vn,
                                fun=np.mean,
                                ref=self.Bus)
```

Suppose we define two areas, 1 and 2, the Bus data looks like

idx	area	Vn
1	1	110
2	2	220
3	1	345
4	1	500

Then, `self.Bus.v` is a list of two lists `[ [1, 3, 4], [2] ]`. `self.Vn.v` will be retrieved and linearly stored as `[110, 345, 500, 220]`. Based on the shape from `self.Bus`, `numpy.mean()` will be called on `[110, 345, 500]` and `[220]` respectively. Thus, `self.Vn_mean.v` will become `[318.33, 220]`.



**v**

Return the reduced values from the reduction function in an array

**Returns****The array `self._v` storing the reduced values****class** `andes.core.service.NumRepeat` (*u, ref, \*\*kwargs*)Bases: `andes.core.service.OperationService`

A helper Service type which repeats a v-provider's value based on the shape from a BackRef

**Examples**

NumRepeat was originally designed for computing the inertia-weighted average rotor speed (center of inertia speed). COI speed is computed with

$$\omega_{COI} = \frac{\sum M_i * \omega_i}{\sum M_i}$$

The numerator can be calculated with a mix of BackRef, ExtParam and ExtState. The denominator needs to be calculated with NumReduce and Service Repeat. That is, use NumReduce to calculate the sum, and use NumRepeat to repeat the summed value for each device.

In the COI class, one would have

```
class COIModel(...):
    def __init__(...):
        ...
        self.SynGen = BackRef()
        self.SynGenIdx = RefFlatten(ref=self.SynGen)
        self.M = ExtParam(model='SynGen',
                           src='M',
                           indexer=self.SynGenIdx)

        self.wgen = ExtState(model='SynGen',
                              src='omega',
                              indexer=self.SynGenIdx)

        self.Mt = NumReduce(u=self.M,
                             fun=np.sum,
                             ref=self.SynGen)

        self.Mtr = NumRepeat(u=self.Mt,
                              ref=self.SynGen)

        self.pidx = IdxRepeat(u=self.idx, ref=self.SynGen)
```

Finally, one would define the center of inertia speed as

```
self.wcoi = Algeb(v_str='1', e_str='-wcoi')

self.wcoi_sub = ExtAlgeb(model='COI',
```

(continues on next page)

(continued from previous page)

```

src='wcoi',
e_str='M * wgen / Mtr',
v_str='M / Mtr',
indexer=self.pidx,
)

```

It is very worth noting that the implementation uses a trick to separate the average weighted sum into  $n$  sub-equations, each calculating the  $(M_i * \omega_i) / (\sum M_i)$ . Since all the variables are preserved in the sub-equation, the derivatives can be calculated correctly.

**v**

Return the values of the repeated values in a sequential 1-D array

### Returns

The array, **self.\_v** storing the repeated values

```

class andes.core.service.NumSelect (optional, fallback, name: Optional[str] = None,
                                     tex_name: Optional[str] = None, info: Op-
                                     tional[str] = None)

```

Bases: *andes.core.service.OperationService*

Class for selecting values for optional NumParam.

### Notes

One use case is to allow an optional turbine rating. One can do

```

self.Tn = NumParam(default=None)
self.Sg = ExtParam(...)
self.Sn = DataSelect(Tn, Sg)

```

**v**

Return values stored in *self.\_v*. May be overloaded by subclasses.

```

class andes.core.service.OperationService (name=None, tex_name=None,
                                           info=None)

```

Bases: *andes.core.service.BaseService*

Base class for a type of Service which performs specific operations

This class cannot be used by itself.

**See also:**

**NumReduce** Service for Reducing linearly stored 2-D services into 1-D

**NumRepeat** Service for repeating 1-D NumParam/ v-array following a sub-pattern

**IdxRepeat** Service for repeating 1-D IdxParam/ v-list following a sub-pattern

**v**

Return values stored in *self.\_v*. May be overloaded by subclasses.

```
class andes.core.service.ParamCalc(param1, param2, func, name=None,
                                   tex_name=None, info=None, cache=True)
```

Bases: `andes.core.service.BaseService`

Parameter calculation service.

Useful to create parameters calculated instantly from existing ones.

**v**

```
class andes.core.service.PostInitService(v_str: Optional[str] = None,
                                         v_numeric: Optional[Callable]
                                         = None, vtype: Optional[type] =
                                         None, name: Optional[str] = None,
                                         tex_name=None, info=None)
```

Bases: `andes.core.service.ConstService`

Constant service that gets stored once after init.

This service is useful when one need to store initialization values stored in variables.

## Examples

In ESST3A model, the  $v_f$  variable is initialized followed by other variables. One can store the initial  $v_f$  into  $v_f0$  so that equation  $v_f - v_f0 = 0$  will hold.

```
self.vref0 = PostInitService(info='Initial reference voltage input',
                             tex_name='V_{ref0}',
                             v_str='vref',
                             )
```

Since all *ConstService* are evaluated before equation evaluation, without using *PostInitService*, one will need to create lots of *ConstService* to store values in the initialization path towards  $v_f0$ , in order to correctly initialize  $v_f$ .

```
class andes.core.service.RandomService(func=<built-in method rand of
                                       numpy.random.mtrand.RandomState
                                       object>, **kwargs)
```

Bases: `andes.core.service.BaseService`

A service type for generating random numbers.

### Parameters

**name** [str] Name

**func** [Callable] A callable for generating the random variable.

**Warning:** The value will be randomized every time it is accessed. Do not use it if the value needs to be stable for each simulation step.

**v**

This class has  $v$  wrapped by a property decorator.

## Returns

**array-like** Randomly generated service variables

**class** `andes.core.service.RefFlatten(ref, **kwargs)`

Bases: `andes.core.service.OperationService`

A service type for flattening `andes.core.service.BackRef` into a 1-D list.

## Examples

This class is used when one wants to pass *BackRef* values as indexer.

`andes.models.coi.COI` collects referencing `andes.models.group.SynGen` with

```
self.SynGen = BackRef(info='SynGen idx lists', export=False)
```

After collecting BackRefs, `self.SynGen.v` will become a two-level list of indices, where the first level correspond to each COI and the second level correspond to generators of the COI.

Convert `self.SynGen` into 1-d as `self.SynGenIdx`, which can be passed as indexer for retrieving other parameters and variables

```
self.SynGenIdx = RefFlatten(ref=self.SynGen)

self.M = ExtParam(model='SynGen', src='M',
                  indexer=self.SynGenIdx, export=False,
                  )
```

**v**

Return values stored in `self._v`. May be overloaded by subclasses.

**class** `andes.core.service.Replace(old_val, fct, new_val, name=None, tex_name=None, info=None, cache=True)`

Bases: `andes.core.service.BaseService`

Replace parameters with new values if the function returns True

**v**

**class** `andes.core.service.VarHold(u, hold, vtype=None, name=None, tex_name=None, info=None)`

Bases: `andes.core.service.VarService`

Service for holding the input when the hold state is on.

**check** (`**kwargs`)

**class** `andes.core.service.VarService(v_str: Optional[str] = None, v_numeric: Optional[Callable] = None, vtype: Optional[type] = None, name: Optional[str] = None, tex_name=None, info=None)`

Bases: `andes.core.service.ConstService`

Variable service that gets updated in each step/loop as variables change.

This class is useful when one has non-differentiable algebraic equations, which make use of *abs()*, *re* and *im*. Instead of creating *Algeb*, one can put the equation in *VarService*, which will be updated before solving algebraic equations.

**Warning:** *VarService* is not solved with other algebraic equations, meaning that there is one step "delay" between the algebraic variables and *VarService*. Use an algebraic variable whenever possible.

## Examples

In ESST3A model, the voltage and current sensors ( $v_d + jv_q$ ), ( $I_d + jI_q$ ) estimate the sensed VE using equation

$$VE = |K_{PC} * (v_d + 1jv_q) + 1j(K_I + K_{PC} * X_L) * (I_d + 1jI_q)|$$

One can use *VarService* to implement this equation

```
self.VE = VarService(tex_name='V_E',
                      info='VE',
                      v_str='Abs(KPC*(vd + 1j*vq) + 1j*(KI + KPC*XL)*(Id_
↵+ 1j*Iq))',
                      )
```

### 12.1.7 andes.core.solver module

**class** andes.core.solver.CuPySolver

Bases: *andes.core.solver.SciPySolver*

CuPy lsqr solver (GPU-based).

**solve** (*A*, *b*)

Solve linear systems.

#### Parameters

**A** [scipy.csc\_matrix] Sparse N-by-N matrix

**b** [numpy.ndarray] Dense 1-dimensional array of size N

#### Returns

**np.ndarray** Solution *x* to  $Ax = b$

**class** andes.core.solver.KLUSolver

Bases: *andes.core.solver.SuiteSparseSolver*

KLU solver.

Requires package *cvxoptklu*.

**linsolve** (*A*, *b*)

Solve linear equation set  $Ax = b$  and returns the solutions in a 1-D array.

This function performs both symbolic and numeric factorizations every time, and can be slower than `Solver.solve`.

**Parameters**

**A** Sparse matrix

**b** RHS of the equation

**Returns**

The solution in a 1-D np array.

**class** `andes.core.solver.SciPySolver`

Bases: `object`

Base class for scipy family solvers.

**clear** ()**linsolve** (*A*, *b*)

Exactly same functionality as *solve*.

**solve** (*A*, *b*)

Solve linear systems.

**Parameters**

**A** [`scipy.csc_matrix`] Sparse N-by-N matrix

**b** [`numpy.ndarray`] Dense 1-dimensional array of size N

**Returns**

**np.ndarray** Solution *x* to  $Ax = b$

**to\_csc** (*A*)

Convert *A* to `scipy.sparse.csc_matrix`.

**Parameters**

**A** [`cvxopt.spmatrix`] Sparse N-by-N matrix

**Returns**

**scipy.sparse.csc\_matrix** Converted `csc_matrix`

**class** `andes.core.solver.Solver` (*sparselib*='umfpack')

Bases: `object`

Sparse matrix solver class.

This class wraps UMFPACK, KLU, SciPy and CuPy solvers to provide an unified interface for solving sparse linear equations  $Ax = b$ .

Provides methods `solve`, `linsolve` and `clear`.

**clear()**

Remove all cached objects.

**linsolve**(*A*, *b*)

Solve linear equations without caching factorization. Performs full factorization each call.

**Parameters**

**A** [cvxopt.spmatrix] Sparse N-by-N matrix

**b** [cvxopt.matrix or numpy.ndarray] Dense N-by-1 matrix

**Returns**

**numpy.ndarray** Dense N-by-1 array

**solve**(*A*, *b*)

Solve linear equations and cache factorizations if possible.

**Parameters**

**A** [cvxopt.spmatrix] Sparse N-by-N matrix

**b** [cvxopt.matrix or numpy.ndarray] Dense N-by-1 matrix

**Returns**

**numpy.ndarray** Dense N-by-1 array

**class** `andes.core.solver.SpSolve`

Bases: `andes.core.solver.SciPySolver`

`scipy.sparse.linalg.spsolve` Solver.

**solve**(*A*, *b*)

Solve linear systems.

**Parameters**

**A** [`scipy.csc_matrix`] Sparse N-by-N matrix

**b** [`numpy.ndarray`] Dense 1-dimensional array of size N

**Returns**

**np.ndarray** Solution  $x$  to  $Ax = b$

**class** `andes.core.solver.SuiteSparseSolver`

Bases: `object`

Base SuiteSparse solver interface.

Need to be derived by specific solvers such as UMFPACK or KLU.

**clear()**

Remove all cached PyCapsule of C objects

**linsolve**(*A*, *b*)

Solve linear equation set  $Ax = b$  and returns the solutions in a 1-D array.

This function performs both symbolic and numeric factorizations every time, and can be slower than `Solver.solve`.

**Parameters**

- A** Sparse matrix
- b** RHS of the equation

**Returns**

**The solution in a 1-D np array.**

**solve** (*A*, *b*)

Solve linear system  $Ax = b$  using numeric factorization *N* and symbolic factorization *F*. Store the solution in *b*.

This function caches the symbolic factorization in `self.F` and is faster in general. Will attempt `Solver.linsolve` if the cached symbolic factorization is invalid.

**Parameters**

- A** Sparse matrix for the equation set coefficients.
- F** The symbolic factorization of *A* or a matrix with the same non-zero shape as *A*.
- N** Numeric factorization of *A*.
- b** RHS of the equation.

**Returns**

**numpy.ndarray** The solution in a 1-D ndarray

**class** `andes.core.solver.UMFPACKSolver`

Bases: `andes.core.solver.SuiteSparseSolver`

UMFPACK solver.

Utilizes `cvxopt.umfpack` for factorization.

**linsolve** (*A*, *b*)

Solve linear equation set  $Ax = b$  and returns the solutions in a 1-D array.

This function performs both symbolic and numeric factorizations every time, and can be slower than `Solver.solve`.

**Parameters**

- A** Sparse matrix
- b** RHS of the equation

**Returns**

**The solution in a 1-D np array.**



### 12.1.8 andes.core.common module

**class** `andes.core.common.Config` (*name*, *dct=None*, *\*\*kwargs*)

Bases: `object`

A class for storing system, model and routine configurations.

**add** (*dct=None*, *\*\*kwargs*)

Add config fields from a dictionary or keyword args.

Existing configs will NOT be overwritten.

**add\_extra** (*dest*, *dct=None*, *\*\*kwargs*)

Add extra contents for config.

#### Parameters

**dest** [str] Destination string in *\_alt*, *\_help* or *\_tex*.

**dct** [OrderedDict, dict] key: value pairs

**as\_dict** (*refresh=False*)

Return the config fields and values in an `OrderedDict`.

Values are cached in *self.\_dict* unless refreshed.

**check** ()

Check the validity of config values.

**doc** (*max\_width=78*, *export='plain'*, *target=False*, *symbol=True*)

**load** (*config*)

Load from a `ConfigParser` object, *config*.

**tex\_names**

**class** `andes.core.common.DummyValue` (*value*)

Bases: `object`

Class for converting a scalar value to a dummy parameter with *name* and *tex\_name* fields.

A `DummyValue` object can be passed to `Block`, which utilizes the *name* field to dynamically generate equations.

#### Notes

Pass a numerical value to the constructor for most use cases, especially when passing as a v-provider.

**class** `andes.core.common.JacTriplet`

Bases: `object`

Storage class for Jacobian triplet lists.

**append\_ijv** (*j\_full\_name*, *ii*, *jj*, *vv*)

Append triplets to the given sparse matrix triplets.

#### Parameters

**j\_full\_name** [str] Full name of the sparse Jacobian. If is a constant Jacobian, append 'c' to the Jacobian name.

**ii** [array-like] Row indices

**jj** [array-like] Column indices

**vv** [array-like] Value indices

**clear\_ijv()**

Clear stored triplets for all sparse Jacobian matrices

**ijv**(*j\_full\_name*)

Return triplet lists in a tuple in the order of (ii, jj, vv)

**merge**(*triplet*)

Merge another triplet into this one.

**zip\_ijv**(*j\_full\_name*)

Return a zip iterator in the order of (ii, jj, vv)

```
class andes.core.common.ModelFlags (collate=False, pflow=False, tds=False,  
pflow_init=None, tds_init=None, series=False,  
nr_iter=False, f_num=False, g_num=False,  
j_num=False, s_num=False, sv_num=False)
```

Bases: `object`

Model flags.

### Parameters

**collate** [bool] True: collate variables by device; False: by variable. Non-collate (continuous memory) has faster computation speed.

**pflow** [bool] True: called during power flow

**tds** [bool] True if called during tds; if is False, `dae_t` cannot be used

**pflow\_init** [bool or None] True if initialize pflow; False otherwise; None default to *pflow*

**tds\_init** [bool or None] True if initialize tds; False otherwise; None default to *tds*

**series** [bool] True if is series device

**nr\_iter** [bool] True if is series device

**f\_num** [bool] True if the model defines *f\_numeric*

**g\_num** [bool] True if the model defines *g\_numeric*

**j\_num** [bool] True if the model defines *j\_numeric*

**s\_num** [bool] True if the model defines *s\_numeric*

**sv\_num** [bool] True if the model defines *s\_numeric\_var*

**jited** [bool] True if numba JIT code is generated

**update**(*dct*)

`andes.core.common.dummyfy(param)`

Dummify scalar parameter and return a DummyValue object. Do nothing for BaseParam instances.

#### Parameters

**param** [float, int, str, BaseParam] parameter object or scalar value

#### Returns

DummyValue(param) if param is a scalar; param itself, otherwise.

### 12.1.9 andes.core.var module

```
class andes.core.var.Algeb (name: Optional[str] = None, tex_name: Optional[str] =
    None, info: Optional[str] = None, unit: Optional[str] =
    None, v_str: Union[str, float, None] = None, v_iter: Op-
    tional[str] = None, e_str: Optional[str] = None, discrete:
    Optional[andes.core.discrete.Discrete] = None, v_setter:
    Optional[bool] = False, e_setter: Optional[bool] = False,
    addressable: Optional[bool] = True, export: Optional[bool]
    = True, diag_eps: Optional[float] = 0.0)
```

Bases: `andes.core.var.BaseVar`

Algebraic variable class, an alias of the *BaseVar*.

#### Attributes

**e\_code** [str] Equation code string, equals string literal `g`

**v\_code** [str] Variable code string, equals string literal `y`

**e\_code** = 'g'

**v\_code** = 'y'

```
class andes.core.var.AliasAlgeb (var, **kwargs)
```

Bases: `andes.core.var.ExtAlgeb`

Alias algebraic variable. Essentially *ExtAlgeb* that links to a model's own variable.

*AliasAlgeb* is useful when the final output of a model is from a block, but the model must provide the final output in a pre-defined name. Using *AliasAlgeb*, A model can avoid adding an additional variable with a dummy equations.

Like *ExtVar*, labels of *AliasAlgeb* will not be saved in the final output. When plotting from file, one need to look up the original variable name.

```
class andes.core.var.AliasState (var, **kwargs)
```

Bases: `andes.core.var.ExtState`

Alias state variable.

Refer to the docs of *AliasAlgeb*.

```
class andes.core.var.BaseVar (name: Optional[str] = None, tex_name: Optional[str] =
                             None, info: Optional[str] = None, unit: Optional[str] =
                             None, v_str: Union[str, float, None] = None, v_iter: Op-
                             tional[str] = None, e_str: Optional[str] = None, discrete:
                             Optional[andes.core.discrete.Discrete] = None, v_setter:
                             Optional[bool] = False, e_setter: Optional[bool] =
                             False, addressable: Optional[bool] = True, export: Op-
                             tional[bool] = True, diag_eps: Optional[float] = 0.0)
```

Bases: `object`

Base variable class.

Derived classes *State* and *Algeb* should be used to build model variables.

#### Parameters

**name** [str, optional] Variable name

**info** [str, optional] Descriptive information

**unit** [str, optional] Unit

**tex\_name** [str] LaTeX-formatted variable name. If is None, use *name* instead.

**discrete** [Discrete] Associated discrete component. Will call *check\_var* on the discrete component.

#### Attributes

**a** [array-like] variable address

**v** [array-like] local-storage of the variable value

**e** [array-like] local-storage of the corresponding equation value

**e\_str** [str] the string/symbolic representation of the equation

**class\_name**

**get\_names** ()

**reset** ()

Reset the internal numpy arrays and flags.

**set\_address** (addr: *numpy.ndarray*, *contiguous=False*)

Set the address of internal variables.

#### Parameters

**addr** [ndarray] The assigned address for this variable

**contiguous** [bool, optional] If the addresses are contiguous

**set\_arrays** (dae)

Set the equation and values arrays.

It slicing into DAE (when contiguous) or allocating new memory (when not contiguous).

#### Parameters

**dae** [DAE] Reference to System.dae

```
class andes.core.var.ExtAlgeb(model: str, src: str, indexer: Union[List[T],
    numpy.ndarray, andes.core.param.BaseParam, andes.core.service.BaseService, None] = None, allow_none: Optional[bool] = False, name: Optional[str] = None, tex_name: Optional[str] = None, info: Optional[str] = None, unit: Optional[str] = None, v_str: Union[str, float, None] = None, v_iter: Optional[str] = None, e_str: Optional[str] = None, v_setter: Optional[bool] = False, e_setter: Optional[bool] = False, addressable: Optional[bool] = True, export: Optional[bool] = True, diag_eps: Optional[float] = 0.0)
```

Bases: `andes.core.var.ExtVar`

External algebraic variable type.

**e\_code** = 'g'

**v\_code** = 'y'

```
class andes.core.var.ExtState(model: str, src: str, indexer: Union[List[T],
    numpy.ndarray, andes.core.param.BaseParam, andes.core.service.BaseService, None] = None, allow_none: Optional[bool] = False, name: Optional[str] = None, tex_name: Optional[str] = None, info: Optional[str] = None, unit: Optional[str] = None, v_str: Union[str, float, None] = None, v_iter: Optional[str] = None, e_str: Optional[str] = None, v_setter: Optional[bool] = False, e_setter: Optional[bool] = False, addressable: Optional[bool] = True, export: Optional[bool] = True, diag_eps: Optional[float] = 0.0)
```

Bases: `andes.core.var.ExtVar`

External state variable type.

**Warning:** ExtState is not allowed to set `t_const`, as it will conflict with the source State variable. In fact, one should not set `e_str` for ExtState.

**e\_code** = 'f'

**t\_const** = None

**v\_code** = 'x'

```
class andes.core.var.ExtVar(model: str, src: str, indexer: Union[List[T],
                                numpy.ndarray, andes.core.param.BaseParam,
                                andes.core.service.BaseService, None] = None, allow_none:
                                Optional[bool] = False, name: Optional[str] = None,
                                tex_name: Optional[str] = None, info: Optional[str]
                                = None, unit: Optional[str] = None, v_str: Union[str,
                                float, None] = None, v_iter: Optional[str] = None,
                                e_str: Optional[str] = None, v_setter: Optional[bool]
                                = False, e_setter: Optional[bool] = False, addressable:
                                Optional[bool] = True, export: Optional[bool] = True,
                                diag_eps: Optional[float] = 0.0)
```

Bases: `andes.core.var.BaseVar`

Externally defined algebraic variable

This class is used to retrieve the addresses of externally- defined variable. The *e* value of the *ExtVar* will be added to the corresponding address in the DAE equation.

### Parameters

**model** [str] Name of the source model

**src** [str] Source variable name

**indexer** [BaseParam, BaseService] A parameter of the hosting model, used as indices into the source model and variable. If is None, the source variable address will be fully copied.

**allow\_none** [bool] True to allow None in indexer

### Attributes

**parent\_model** [Model] The parent model providing the original parameter.

**uid** [array-like] An array containing the absolute indices into the parent\_instance values.

**e\_code** [str] Equation code string; copied from the parent instance.

**v\_code** [str] Variable code string; copied from the parent instance.

**link\_external** (*ext\_model*)

Update variable addresses provided by external models

This method sets attributes including *parent\_model*, *parent\_instance*, *uid*, *a*, *n*, *e\_code* and *v\_code*. It initializes the *e* and *v* to zero.

### Parameters

**ext\_model** [Model] Instance of the parent model

### Returns

None

**Warning:** *link\_external* does not check if the *ExtVar* type is the same as the original variable to reduce performance overhead. It will be a silent error (a dimension too small error from *dae.build\_pattern*) if a model uses *ExtAlgeb* to access a *State*, or vice versa.

**set\_address** (*addr*, *contiguous=False*)

Empty function.

**set\_arrays** (*dae*)

Empty function.

```
class andes.core.var.State(name: Optional[str] = None, tex_name: Optional[str]
    = None, info: Optional[str] = None, unit: Optional[str]
    = None, v_str: Union[str, float, None] = None, v_iter:
    Optional[str] = None, e_str: Optional[str] = None,
    discrete: Optional[andes.core.discrete.Discrete] =
    None, t_const: Union[andes.core.param.BaseParam,
    andes.core.common.DummyValue,
    andes.core.service.BaseService, None] = None, v_setter:
    Optional[bool] = False, e_setter: Optional[bool] = False,
    addressable: Optional[bool] = True, export: Optional[bool]
    = True, diag_eps: Optional[float] = 0.0)
```

Bases: *andes.core.var.BaseVar*

Differential variable class, an alias of the *BaseVar*.

### Parameters

**t\_const** [BaseParam, DummyValue] Left-hand time constant for the differential equation. Time constants will not be evaluated as part of the differential equation. They will be collected to array *dae.Tf* to multiply to the right-hand side *dae.f*.

### Attributes

**e\_code** [str] Equation code string, equals string literal *f*

**v\_code** [str] Variable code string, equals string literal *x*

**e\_code** = 'f'

**v\_code** = 'x'

## 12.1.10 Module contents

Import subpackage classes

## 12.2 andes.io package

### 12.2.1 Submodules

#### 12.2.2 andes.io.matpower module

Simple MATPOWER format parser

`andes.io.matpower.read(system, file)`

Read a MATPOWER data file into mpc and build andes device elements

`andes.io.matpower.testlines(fid)`

#### 12.2.3 andes.io.psse module

PSS/E file parser.

Include a RAW parser and a DYR parser.

`andes.io.psse.get_block_lines(b, mdata)`

Return the number of lines based on data

`andes.io.psse.read(system, file)`

read PSS/E RAW file v32 format

`andes.io.psse.read_add(system, file)`

Read an addition PSS/E dyr file.

##### Parameters

**system** [System] System instance to which data will be loaded

**file** [str] Path to the additional dyr file

##### Returns

**bool** data parsing status

`andes.io.psse.sort_psse_models(dyr_yaml)`

Sort supported models so that model names are ordered by dependency.

`andes.io.psse.testlines(fid)`

Check the raw file for frequency base

#### 12.2.4 andes.io.txt module

`andes.io.txt.dump_data(text, header, rowname, data, file, width=14, precision=5)`



### 12.2.5 andes.io.xlsx module

Excel reader and writer for ANDES power system parameters

This module utilizes `xlsxwriter` and `pandas.DataFrame`. While I like the simplicity of the dome format, spreadsheet data is easier to read and edit.

`andes.io.xlsx.read(system, infile)`

Read an `xlsx` file with ANDES model data into an empty system

#### Parameters

**system** [System] Empty System instance

**infile** [str] Path to the input file

#### Returns

**System** System instance after succeeded

`andes.io.xlsx.testlines(fid)`

`andes.io.xlsx.write(system, outfile, skip_empty=True, overwrite=None, add_book=None, **kwargs)`

Write loaded ANDES system data into an `xlsx` file

#### 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

#### Returns

**bool** True if file written; False otherwise

### 12.2.6 Module contents

`andes.io.dump(system, output_format, full_path=None, overwrite=False, **kwargs)`

Dump the System data into the requested output format.

#### Parameters

**system** System object

**output\_format** [str] Output format name. 'xlsx' will be used if is not an instance of `str`.

#### Returns

**bool** True if successful; False otherwise.

`andes.io.get_output_ext (out_format)`

`andes.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

`andes.io.parse (system)`

Parse input file with the given format in *system.files.input\_format*.

#### Returns

**bool** True if successful; False otherwise.

## 12.3 andes.models package

### 12.3.1 Submodules

### 12.3.2 andes.models.area module

**class** `andes.models.area.ACE (system, config)`

Bases: *andes.models.area.ACEData*, *andes.core.model.Model*

Area Control Error model.

Note: area idx is automatically retrieved from *bus*.

**class** `andes.models.area.ACEData`

Bases: *andes.core.model.ModelData*

Area Control Error data

**class** `andes.models.area.Area (system, config)`

Bases: *andes.models.area.AreaData*, *andes.core.model.Model*

**bus\_table ()**

Return a formatted table with area idx and bus idx correspondence

#### Returns

**str** Formatted table

**class** `andes.models.area.AreaData`

Bases: *andes.core.model.ModelData*

### 12.3.3 andes.models.bus module

**class** andes.models.bus.**Bus** (*system=None, config=None*)

Bases: *andes.core.model.Model, andes.models.bus.BusData*

AC Bus model.

Power balance equation have the form of  $\text{load} - \text{injection} = 0$ . Namely, load is positively summed, while injections are negative.

**class** andes.models.bus.**BusData**

Bases: *andes.core.model.ModelData*

Class for Bus data

### 12.3.4 andes.models.governor module

**class** andes.models.governor.**IEEEG1** (*system, config*)

Bases: *andes.models.governor.IEEEG1Data, andes.models.governor.IEEEG1Model*

IEEE Type 1 Speed-Governing Model.

If only one generator is connected, its *idx* must be given to *syn*, and *syn2* must be left blank. Each generator must provide data in its *Sn* base.

*syn* is connected to the high-pressure output (PHP) and the optional *syn2* is connected to the low-pressure output (PLP).

The speed deviation of generator 1 (*syn*) is measured. If the turbine rating *Tn* is not specified, the sum of *Sn* of all connected generators will be used.

Normally,  $K1 + K2 + \dots + K8 = 1.0$ . If the second generator is not connected,  $K1 + K3 + K5 + K7 = 1$ , and  $K2 + K4 + K6 + K8 = 0$ .

**class** andes.models.governor.**IEEEG1Data**

Bases: *andes.models.governor.TGBaseData*

**class** andes.models.governor.**IEEEG1Model** (*system, config*)

Bases: *andes.models.governor.TGBase*

**class** andes.models.governor.**TG2** (*system, config*)

Bases: *andes.models.governor.TG2Data, andes.models.governor.TGBase*

**class** andes.models.governor.**TG2Data**

Bases: *andes.models.governor.TGBaseData*

**class** andes.models.governor.**TGBase** (*system, config, add\_sn=True, add\_tm0=True*)

Bases: *andes.core.model.Model*

Base Turbine Governor model.

#### Parameters

**add\_sn** [bool] True to add NumSelect Sn; False to add later in custom models.  
This is useful when the governor connects to two generators.

**add\_tm0** [bool] True to add ExtService tm0.

**class** andes.models.governor.**TGBaseData**

Bases: *andes.core.model.ModelData*

Base data for turbine governors.

**class** andes.models.governor.**TGOV1** (*system, config*)

Bases: *andes.models.governor.TGOV1Data*, *andes.models.governor.TGOV1Model*

TGOV1 turbine governor model.

Implements the PSS/E TGOV1 model without deadband.

**class** andes.models.governor.**TGOV1DB** (*system, config*)

Bases: *andes.models.governor.TGOV1DBData*, *andes.models.governor.TGOV1DBModel*

TGOV1 turbine governor model with speed input deadband.

**class** andes.models.governor.**TGOV1DBData**

Bases: *andes.models.governor.TGOV1Data*

**class** andes.models.governor.**TGOV1DBModel** (*system, config*)

Bases: *andes.models.governor.TGOV1Model*

**class** andes.models.governor.**TGOV1Data**

Bases: *andes.models.governor.TGBaseData*

**class** andes.models.governor.**TGOV1Model** (*system, config*)

Bases: *andes.models.governor.TGBase*

**class** andes.models.governor.**TGOV1ModelAlt** (*system, config*)

Bases: *andes.models.governor.TGBase*

An alternative implementation of TGOV1 from equations (without using Blocks).

### 12.3.5 andes.models.group module

**class** andes.models.group.**ACLine**

Bases: *andes.models.group.GroupBase*

**class** andes.models.group.**ACTopology**

Bases: *andes.models.group.GroupBase*

**class** andes.models.group.**Calculation**

Bases: *andes.models.group.GroupBase*

Group of classes that calculates based on other models.

**class** andes.models.group.**Collection**

Bases: *andes.models.group.GroupBase*

Collection of topology models

```
class andes.models.group.DCLink
    Bases: andes.models.group.GroupBase

    Basic DC links

class andes.models.group.DCTopology
    Bases: andes.models.group.GroupBase

class andes.models.group.DG
    Bases: andes.models.group.GroupBase

    Distributed generation (small-scale).

class andes.models.group.Exciter
    Bases: andes.models.group.GroupBase

    Exciter group for synchronous generators.

class andes.models.group.Experimental
    Bases: andes.models.group.GroupBase

    Experimental group

class andes.models.group.FreqMeasurement
    Bases: andes.models.group.GroupBase

    Frequency measurements.

class andes.models.group.GroupBase
    Bases: object

    Base class for groups

    add (idx, model)
        Register an idx from model_name to the group
```

#### Parameters

**idx:** `Union[str, float, int]` Register an element to a model

**model:** `Model` instance of the model

```
add_model (name: str, instance)
    Add a Model instance to group.
```

#### Parameters

**name** `[str]` Model name

**instance** `[Model]` Model instance

#### Returns

`None`

```
class_name
```

**doc** (*export='plain'*)

Return the documentation of the group in a string.

**doc\_all** (*export='plain'*)

Return documentation of the group and its models.

**Parameters**

**export** ['plain' or 'rest'] Export format, plain-text or RestructuredText

**Returns**

**str**

**find\_idx** (*keys, values, allow\_none=False, default=None*)

Find indices of devices that satisfy the given *key=value* condition.

This method iterates over all models in this group.

**get** (*src: str, idx, attr: str = 'v', allow\_none=False, default=0.0*)

Based on the indexer, get the *attr* field of the *src* parameter or variable.

**Parameters**

**src** [str] param or var name

**idx** [array-like] device idx

**attr** The attribute of the param or var to retrieve

**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**

**The requested param or variable attribute. If *idx* is a list, return a list of values.**

**If *idx* is a single element, return a single value.**

**get\_next\_idx** (*idx=None, model\_name=None*)

Get a no-conflict idx for a new device. Use the provided *idx* if no conflict. Generate a new one otherwise.

**Parameters**

**idx** [str or None] Proposed idx. If None, assign a new one.

**model\_name** [str or None] Model name. If not, prepend the group name.

**Returns**

**str** New device name.

**idx2model** (*idx, allow\_none=False*)

Find model name for the given idx.

**Parameters**

**idx** [float, int, str, array-like] idx or idx-es of devices.

**allow\_none** [bool] If True, return *None* at the positions where *idx* is not found.

### Returns

If *idx* is a list, return a list of model instances.

If *idx* is a single element, return a model instance.

**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

**n**

Total number of devices.

**set** (*src: str, idx, attr, value*)

Set the value of an attribute of a group property. Performs `self.<src>.<attr>[idx] = value`.

The user needs to ensure that the property is shared by all models in this group.

### Parameters

**src** [str] Name of property.

**idx** [str, int, float, array-like] Indices of 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.

**class** `andes.models.group.Motor`

Bases: `andes.models.group.GroupBase`

Induction Motor group

**class** `andes.models.group.PSS`

Bases: `andes.models.group.GroupBase`

Power system stabilizer group.

**class** `andes.models.group.PhaseMeasurement`

Bases: `andes.models.group.GroupBase`

Phasor measurements

**class** `andes.models.group.RenAerodynamics`

Bases: `andes.models.group.GroupBase`

Renewable aerodynamics group.

**class** andes.models.group.**RenExciter**  
Bases: *andes.models.group.GroupBase*

Renewable electrical control (exciter) group.

**class** andes.models.group.**RenGen**  
Bases: *andes.models.group.GroupBase*

Renewable generator (converter) group.

**class** andes.models.group.**RenGovernor**  
Bases: *andes.models.group.GroupBase*

Renewable turbine governor group.

**class** andes.models.group.**RenPitch**  
Bases: *andes.models.group.GroupBase*

Renewable generator pitch controller group.

**class** andes.models.group.**RenPlant**  
Bases: *andes.models.group.GroupBase*

Renewable plant control group.

**class** andes.models.group.**RenTorque**  
Bases: *andes.models.group.GroupBase*

Renewable torque (Pref) controller.

**class** andes.models.group.**StaticACDC**  
Bases: *andes.models.group.GroupBase*

AC DC device for power flow

**class** andes.models.group.**StaticGen**  
Bases: *andes.models.group.GroupBase*

Static generator group for power flow calculation

**class** andes.models.group.**StaticLoad**  
Bases: *andes.models.group.GroupBase*

Static load group.

**class** andes.models.group.**StaticShunt**  
Bases: *andes.models.group.GroupBase*

Static shunt compensator group.

**class** andes.models.group.**SynGen**  
Bases: *andes.models.group.GroupBase*

Synchronous generator group.

**class** andes.models.group.**TimedEvent**  
Bases: *andes.models.group.GroupBase*



Timed event group

**class** andes.models.group.TurbineGov

Bases: *andes.models.group.GroupBase*

Turbine governor group for synchronous generator.

**class** andes.models.group.Undefined

Bases: *andes.models.group.GroupBase*

The undefined group. Holds models with no group.

### 12.3.6 andes.models.jit module

**class** andes.models.jit.JIT(*system, model, device, name*)

Bases: *object*

Dummy Just-in-Time initialization class

**doc** (*\*\*kwargs*)

**elem\_add** (*idx=None, name=None, \*\*kwargs*)

overloading elem\_add function of a JIT class

**jit\_load** ()

Import and instantiate this JIT object

### 12.3.7 andes.models.line module

**class** andes.models.line.Line(*system=None, config=None*)

Bases: *andes.models.line.LineData, andes.core.model.Model*

AC transmission line model.

To reduce the number of variables, line injections are summed at bus equations and are not stored. Current injections are not computed.

**class** andes.models.line.LineData

Bases: *andes.core.model.ModelData*

### 12.3.8 andes.models.pq module

**class** andes.models.pq.PQ(*system=None, config=None*)

Bases: *andes.models.pq.PQData, andes.core.model.Model*

PQ load model.

Implements an automatic pq2z conversion during power flow when the voltage is outside [vmin, vmax]. The conversion can be turned off by setting *pq2z* to 0 in the Config file.

Before time-domain simulation, PQ load will be converted to impedance, current source, and power source based on the weights in the Config file.

Weights (p2p, p2i, p2z) corresponds to the weights for constant power, constant current and constant impedance. p2p, p2i and p2z must be in decimal numbers and sum up exactly to 1. The same rule applies to (q2q, q2i, q2z).

**class** andes.models.pq.**PQData**  
Bases: *andes.core.model.ModelData*

### 12.3.9 andes.models.pv module

**class** andes.models.pv.**PV** (*system=None, config=None*)  
Bases: *andes.models.pv.PVData, andes.models.pv.PVModel*

**class** andes.models.pv.**PVData**  
Bases: *andes.core.model.ModelData*

**class** andes.models.pv.**PVModel** (*system=None, config=None*)  
Bases: *andes.core.model.Model*

PV generator model (power flow) with q limit and PV-PQ conversion.

**class** andes.models.pv.**Slack** (*system=None, config=None*)  
Bases: *andes.models.pv.SlackData, andes.models.pv.PVModel*

**class** andes.models.pv.**SlackData**  
Bases: *andes.models.pv.PVData*

### 12.3.10 andes.models.shunt module

**class** andes.models.shunt.**Shunt** (*system=None, config=None*)  
Bases: *andes.models.shunt.ShuntData, andes.core.model.Model*

**class** andes.models.shunt.**ShuntData** (*system=None, name=None*)  
Bases: *andes.core.model.ModelData*

### 12.3.11 andes.models.synchronous module

Synchronous generator classes

**class** andes.models.synchronous.**Flux0**  
Bases: *object*

Flux model without electro-magnetic transients and ignore speed deviation

**class** andes.models.synchronous.**Flux1**  
Bases: *object*

Flux model without electro-magnetic transients but considers speed deviation.

**class** andes.models.synchronous.**Flux2**  
Bases: *object*

Flux model with electro-magnetic transients.

```

class andes.models.synchronous.GENBase (system, config)
    Bases: andes.core.model.Model

    v_numeric (**kwargs)
        Custom variable initialization function.

class andes.models.synchronous.GENBaseData
    Bases: andes.core.model.ModelData

class andes.models.synchronous.GENCLS (system, config)
    Bases: andes.models.synchronous.GENBaseData, andes.models.synchronous.GENBase,
           andes.models.synchronous.GENCLSMModel, andes.models.synchronous.Flux0

class andes.models.synchronous.GENCLSMModel
    Bases: object

class andes.models.synchronous.GENROU (system, config)
    Bases: andes.models.synchronous.GENROUData, andes.models.synchronous.GENBase,
           andes.models.synchronous.GENROUModel, andes.models.synchronous.Flux0

    Round rotor generator with quadratic saturation

class andes.models.synchronous.GENROUData
    Bases: andes.models.synchronous.GENBaseData

class andes.models.synchronous.GENROUModel
    Bases: object

```

### 12.3.12 andes.models.timer module

```

class andes.models.timer.Fault (system, config)
    Bases: andes.core.model.ModelData, andes.core.model.Model

    Three-phase to ground fault.

    apply_fault (is_time: numpy.ndarray)
        Apply fault and store pre-fault algebraic variables (voltages and other algebs) to self._vstore.

    clear_fault (is_time: numpy.ndarray)
        Clear fault and restore pre-fault bus algebraic variables (voltages and others).

class andes.models.timer.Toggler (system, config)
    Bases: andes.models.timer.TogglerData, andes.core.model.Model

    Time-based connectivity status toggler.

    v_numeric (**kwargs)
        Custom initialization function that stores and restores the connectivity status.

class andes.models.timer.TogglerData
    Bases: andes.core.model.ModelData

```

### 12.3.13 Module contents

## 12.4 andes.routines package

### 12.4.1 Submodules

### 12.4.2 andes.routines.base module

```
class andes.routines.base.BaseRoutine (system=None, config=None)
    Bases: object
    Base routine class.
    Provides references to system, config, and solver.
    class_name
    doc (max_width=78, export='plain')
    init ()
    report ()
    run (**kwargs)
    summary ()
```

### 12.4.3 andes.routines.eig module

```
class andes.routines.eig.EIG (system, config)
    Bases: andes.routines.base.BaseRoutine
    Eigenvalue analysis routine
    calc_eigvals ()
        Solve eigenvalues of the state matrix self.As
        Returns
        None
    calc_part_factor (As=None)
        Compute participation factor of states in eigenvalues
    calc_state_matrix ()
        Return state matrix and store to self.As.
        Returns
        cvxopt.matrix state matrix
```

## Notes

For systems with the form

$$\begin{aligned} T\dot{x} &= f(x, y) \\ 0 &= g(x, y) \end{aligned}$$

The state matrix is calculated from

$$A_s = T^{-1}(f_x - f_y * g_y^{-1} * g_x)$$

### **export\_state\_matrix()**

Export state matrix to a <CaseName>\_As.mat file with the variable name As, where <CaseName> is the test case name.

State variable names are stored in variables x\_name and x\_tex\_name.

### **Returns**

**bool** True if successful

**plot** (*mu=None, fig=None, ax=None, left=-6, right=0.5, ymin=-8, ymax=8, damping=0.05, line\_width=0.5, dpi=150, show=True, latex=True*)

### **remove\_singular\_rc()**

Remove rows and cols associated with zero time constant.

### **report** (*x\_name=None*)

Save eigenvalue analysis reports

### **Returns**

**None**

### **run** (*\*\*kwargs*)

### **summary** ()

Print out a summary to `logger.info`.

## 12.4.4 andes.routines.pflow module

**class** `andes.routines.pflow.PFlow` (*system=None, config=None*)

Bases: `andes.routines.base.BaseRoutine`

Power flow calculation routine.

### **init** ()

### **newton\_krylov** (*verbose=False*)

Full Newton-Krylov method from SciPy.

### **Parameters**

**verbose** True if verbose.

**Returns**

**np.array** Solutions *dae.xy*.

**Warning:** The result might be wrong if discrete are in use!

**nr\_step()**

Single step using Newton-Raphson method.

**Returns**

**float** maximum absolute mismatch

**report()**

Write power flow report to text file.

**run(\*\*kwargs)**

Full Newton-Raphson method.

**Returns**

**bool** convergence status

**summary()**

Output a summary for the PFlow routine.

## 12.4.5 andes.routines.tds module

**class** andes.routines.tds.**TDS** (*system=None, config=None*)

Bases: *andes.routines.base.BaseRoutine*

Time-domain simulation routine.

**calc\_h(resume=False)**

Calculate the time step size during the TDS.

**Parameters**

**resume** [bool] If True, calculate the initial step size.

**Returns**

**float** computed time step size stored in `self.h`

**Notes**

A heuristic function is used for variable time step size

```
h = min(0.50 * h, hmin), if niter >= 15
    max(1.10 * h, hmax), if niter <= 6
    min(0.95 * h, hmin), otherwise
```

**do\_switch()**

Checks if is an event time and perform switch if true.

Time is approximated with a tolerance of 1e-8.

**init()**

Initialize the status, storage and values for TDS.

**Returns**

**array-like** The initial values of xy.

**load\_plotter()**

Manually load a plotter into `TDS.plotter`.

**reset()**

Reset internal states to pre-init condition.

**rewind(t)**

TODO: rewind to a past time.

**run(no\_pbar=False, no\_summary=False, \*\*kwargs)**

Run time-domain simulation using numerical integration.

The default method is the Implicit Trapezoidal Method (ITM).

**Parameters**

**no\_pbar** [bool] True to disable progress bar

**no\_summary** [bool, optional] True to disable the display of summary

**save\_output(npz=True)**

Save the simulation data into two files: a lst file and a npz file.

**Parameters**

**npz** [bool] True to save in npz format; False to save in npy format.

**Returns**

**bool** True if files are written. False otherwise.

**streaming\_init()**

Send out initialization variables and process init from modules.

**Returns**

**None**

**streaming\_step()**

Sync, handle and streaming for each integration step.

**Returns**

**None**

**summary()**

Print out a summary of TDS options to `logger.info`.

**Returns****None****test\_init()**

Update f and g to see if initialization is successful.

**12.4.6 Module contents****12.5 andes.utils package****12.5.1 Submodules****12.5.2 andes.utils.cached module****class** andes.utils.cached.**cached** (*func, name=None, doc=None*)Bases: `object`

A decorator that converts a function into a lazy property. The function wrapped is called the first time to retrieve the result and then that calculated result is used the next time you access the value:

```
class Foo(object):

    @cached_property
    def foo(self):
        # calculate something important here
        return 42
```

The class has to have a `__dict__` in order for this property to work. See for details: <http://stackoverflow.com/questions/17486104/python-lazy-loading-of-class-attributes>

**12.5.3 andes.utils.paths module**

Utility functions for loading andes stock test cases

**class** andes.utils.paths.**DisplayablePath** (*path, parent\_path, is\_last*)Bases: `object`**display\_filename\_prefix\_last** = '└─'**display\_filename\_prefix\_middle** = '├─'**display\_parent\_prefix\_last** = '│ '**display\_parent\_prefix\_middle** = ' '**displayable()****displayname****classmethod** **make\_tree** (*root, parent=None, is\_last=False, criteria=None*)



```
andes.utils.paths.cases_root()
```

Return the root path to the stock cases

```
andes.utils.paths.confirm_overwrite(outfile, overwrite=None)
```

```
andes.utils.paths.get_case(rpath)
```

Return the path to a stock case for a given path relative to `andes/cases`.

To list all cases, use `andes.list_cases()`.

## Examples

To get the path to the case *kundur\_full.xlsx* under folder *kundur*, do

```
andes.get_case('kundur/kundur_full.xlsx')
```

```
andes.utils.paths.get_config_path(file_name='andes.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 `andes.rc`.

### Returns

**Config path in string if found; None otherwise.**

```
andes.utils.paths.get_dot_andes_path()
```

Return the path to `<HomeDir>/./andes`

```
andes.utils.paths.get_log_dir()
```

Get the directory for log file.

On Linux or macOS, `/tmp/andes` is the default. On Windows, `%APPDATA%/andes` is the default.

### Returns

**str** The path to the temporary logging directory

```
andes.utils.paths.get_pkl_path()
```

Get the path to the picked/dilled function calls.

### Returns

**str** Path to the `calls.pkl` file

```
andes.utils.paths.list_cases(rpath='.', no_print=False)
```

List stock cases under a given folder relative to `andes/cases`

```
andes.utils.paths.tests_root()
```

Return the root path to the stock cases

### 12.5.4 andes.utils.func module

`andes.utils.func.interp_n2(t, x, y)`

Interpolation function for  $N * 2$  value arrays.

#### Parameters

- t** [float] Point for which the interpolation is calculated
- x** [1-d array with two values] x-axis values
- y** [2-d array with size N-by-2] Values corresponding to x

#### Returns

**N-by-1 array** interpolated values at *t*

`andes.utils.func.list_flatten(input_list)`

Flatten a multi-dimensional list into a flat 1-D list.

### 12.5.5 andes.utils.misc module

`andes.utils.misc.elapsed(t0=0.0)`

Get the elapsed time from the give time. If the start time is not given, returns the unix-time.

#### Returns

- t** [float] Elapsed time from the given time; Otherwise the epoch time.
- s** [str] The elapsed time in seconds in a string

`andes.utils.misc.is_interactive()`

Check if is in an interactive shell (python or ipython).

#### Returns

**bool**

`andes.utils.misc.is_notebook()`

`andes.utils.misc.to_number(s)`

Convert a string to a number. If unsuccessful, return the de-blanked string.

### 12.5.6 andes.utils.tab module

**class** `andes.utils.tab.Tab(title=None, header=None, descr=None, data=None, export='plain', max_width=78)`

Bases: `andes.utils.texttable.Texttable`

Use package `texttable` to create well-formatted tables for setting helps and device helps.

#### Parameters

**export** [('plain', 'rest')] Export format in plain text or restructuredText.

**max\_width** [int] Maximum table width. If there are equations in cells, set to 0 to disable wrapping.

**draw()**

Draw the table and return it in a string.

**header** (*header\_list*)

Set the header with a list.

**set\_title** (*val*)

Set table title to *val*.

`andes.utils.tab.make_doc_table(title, max_width, export, plain_dict, rest_dict)`

Helper function to format documentation data into tables.

`andes.utils.tab.math_wrap(tex_str_list, export)`

Wrap each string item in a list with latex math environment  $\dots$ .

#### Parameters

**tex\_str\_list** [list] A list of equations to be wrapped

**export** [str, ('rest', 'plain')] Export format. Only wrap equations if export format is `rest`.

## 12.5.7 Module contents

## 12.6 andes.variables package

### 12.6.1 Submodules

### 12.6.2 andes.variables.dae module

**class** `andes.variables.dae.DAE` (*system*)

Bases: `object`

Class for storing numerical values of the DAE system, including variables, equations and first order derivatives (Jacobian matrices).

Variable values and equation values are stored as `numpy.ndarray`, while Jacobians are stored as `cvxopt.spmatrix`. The defined arrays and descriptions are as follows:

DAE Array	Description
<code>x</code>	Array for state variable values
<code>y</code>	Array for algebraic variable values
<code>z</code>	Array for 0/1 limiter states (if enabled)
<code>f</code>	Array for differential equation derivatives
<code>Tf</code>	Left-hand side time constant array for <code>f</code>
<code>g</code>	Array for algebraic equation mismatches

The defined scalar member attributes to store array sizes are

Scalar	Description
m	The number of algebraic variables/equations
n	The number of algebraic variables/equations
o	The number of limiter state flags

The derivatives of  $f$  and  $g$  with respect to  $x$  and  $y$  are stored in four `cvxopt.spmatrix` sparse matrices: **fx**, **fy**, **gx**, and **gy**, where the first letter is the equation name, and the second letter is the variable name.

## Notes

DAE in ANDES is defined in the form of

$$\begin{aligned}T\dot{x} &= f(x, y) \\ 0 &= g(x, y)\end{aligned}$$

DAE does not keep track of the association of variable and address. Only a variable instance keeps track of its addresses.

**build\_pattern** (*name*)

Build sparse matrices with stored patterns.

Call to *store\_row\_col\_idx* should be made before this function.

### Parameters

**name** [name] jac name

**clear\_arrays** ()

Reset equation and variable arrays to empty.

**clear\_fg** ()

Resets equation arrays to empty

**clear\_ijv** ()

Clear stored triplets.

**clear\_ts** ()

**clear\_xy** ()

Reset variable arrays to empty.

**clear\_z** ()

Reset status arrays to empty

**fg**

Return a concatenated array of [f, g].

**get\_name** (*arr*)

**get\_size** (*name*)

Get the size of an array or sparse matrix based on name.

**Parameters**

**name** [str (f, g, fx, gy, etc.)] array/sparse name

**Returns**

**tuple** sizes of each element in a tuple

**print\_array** (*name*, *values=None*, *tol=None*)

**reset** ()

Reset array sizes to zero and clear all arrays.

**resize\_arrays** ()

Resize arrays to the new sizes *m* and *n*, and *o*.

If `m > len(self.y)` or `n > len(self.x)`, arrays will be extended. Otherwise, new empty arrays will be sliced, starting from 0 to the given size.

**Warning:** This function should not be called directly. Instead, it is called in `System.set_address` which re-points variables used in power flow to the new array for dynamic analyses.

**restore\_sparse** (*names=None*)

Restore all sparse matrices to the sparsity pattern filled with zeros (for variable Jacobian elements) and non-zero constants.

**Parameters**

**names** [None or list] List of Jacobian names to restore sparsity pattern

**set\_t** (*t*)

Helper function for setting time in-place

**store\_sparse\_ijv** (*name*, *row*, *col*, *val*)

Store the sparse pattern triplets.

This function is to be called by System after building the complete sparsity pattern for each Jacobian matrix.

**Parameters**

**name** [str] sparse Jacobian matrix name

**row** [np.ndarray] all row indices

**col** [np.ndarray] all col indices

**val** [np.ndarray] all values

**write\_lst** (*lst\_path*)

Dump the variable name lst file.

**Parameters**

**lst\_path** Path to the lst file.

**Returns**

**bool** succeed flag

**write\_numpy** (*file\_path*)

Write TDS data into NumPy uncompressed format.

**write\_npz** (*file\_path*)

Write TDS data into NumPy compressed format.

**xy**

Return a concatenated array of [x, y].

**xy\_name**

Return a concatenated list of all variable names without format.

**xy\_tex\_name**

Return a concatenated list of all variable names in LaTeX format.

**xyz**

Return a concatenated array of [x, y].

**xyz\_name**

Return a concatenated list of all variable names without format.

**xyz\_tex\_name**

Return a concatenated list of all variable names in LaTeX format.

**class** andes.variables.dae.**DAETimeSeries** (*dae=None*)

Bases: `object`

DAE time series data

**store\_txyz** (*t, xy, z=None*)

Store t, xy, and z in internal storage, respectively.

**Parameters**

**t** [float] simulation time

**xy** [array-like] array data for states and algebraic variables

**z** [array-like or None] discrete flags data

**txyz**

Return the values of [t, x, y, z] in an array.

**unpack** (*df=False*)

Unpack stored data in `_xy` and `_z` into arrays *t*, *xy*, and *z*.

**Parameters**

**df** [bool] True to construct DataFrames *self.df* and *self.df\_z* (time-consuming).

**x**

**y**

### 12.6.3 andes.variables.fileman module

**class** andes.variables.fileman.**FileMan** (*case=None*, *\*\*kwargs*)

Bases: `object`

Define a File Manager class for System

**get\_fullpath** (*fullname=None*)

Return the original full path if full path is specified, otherwise search in the case file path.

#### Parameters

**fullname** [str, optional] Full name of the file. If relative, prepend *input\_path*. Otherwise, leave it as is.

**set** (*case=None*, *\*\*kwargs*)

Perform the input and output set up.

andes.variables.fileman.**add\_suffix** (*fullname*, *suffix*)

Add suffix to a full file name.

### 12.6.4 andes.variables.report module

**class** andes.variables.report.**Report** (*system*)

Bases: `object`

Report class to store system static analysis reports

**info**

**update** ()

Update values based on the requested content

**write** ()

Write report to file.

andes.variables.report.**report\_info** (*system*)

### 12.6.5 Module contents





### 13.1 andes.cli module

`andes.cli.create_parser()`

The main level of command-line interface.

`andes.cli.main()`

Main command-line interface

`andes.cli.preamble()`

Log the ANDES command-line preamble at the *logging.INFO* level

### 13.2 andes.main module

`andes.main.config_logger(stream=True, file=True, stream_level=20,  
log_file='andes.log', log_path=None, file_level=10)`

Configure a logger for the andes package with options for a *FileHandler* and a *StreamHandler*. This function is called at the beginning of `andes.main.main()`.

#### Parameters

**stream** [bool, optional] Create a *StreamHandler* for *stdout* if *True*. If *False*, the handler will not be created.

**file** [bool, optional] *True* if logging to `log_file`.

**log\_file** [str, optional] Log file name for *FileHandler*, 'andes.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**

`andes.main.doc (attribute=None, list_supported=False, init_seq=False, config=False, **kwargs)`  
Quick documentation from command-line.

`andes.main.edit_conf (edit_config: Union[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` if a config file is found and an editor is opened. `False` if `edit_config` is `False`.

`andes.main.find_log_path (lg)`  
Find the file paths of the *FileHandlers*.

`andes.main.load (case, codegen=False, setup=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_output`.

### Parameters

**case:** **str** Path to the test case

**codegen** [bool, optional] Call full *System.prepare* on the returned system. Set to `True` if one needs to inspect pretty-print equations and run simulations.

**setup** [bool, optional] Call *System.setup* after loading

### Warnings

-----

**If one needs 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.**

`andes.main.misc (edit_config=", save_config=", show_license=False, clean=True, recursive=False, overwrite=None, cli=False, **kwargs)`  
Misc functions.

`andes.main.plot (**kwargs)`  
Wrapper for the plot tool.

`andes.main.prepare(quick=False, incremental=False, cli=False, full=False, **kwargs)`  
Run code generation.

#### Returns

System object if *cli* is *False*; *exit\_code* 0 otherwise.

**Warning:** The default behavior has changed since v1.0.8: when *cli* is *True* and *full* is not *True*, quick code generation will be used.

`andes.main.print_license()`

`andes.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.

`andes.main.run(filename, input_path="", verbose=20, mp_verbose=30, ncpu=2, pool=False, cli=False, codegen=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

**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

**codegen** [bool, optional] Run full code generation for System before loading case. Only used for single test case.

**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..

```
andes.main.run_case(case, *, routine='pflow', profile=False, convert="", convert_all="",
                    add_book=None, codegen=False, remove_pycapsule=False,
                    **kwargs)
```

Run a single simulation case.

```
andes.main.save_conf(config_path=None, overwrite=None)
```

Save the Andes 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 empty string, the save action will not run. Save to `~/andes/andes.conf` if None.

#### Returns

**bool** True is the save action is run. False otherwise.

```
andes.main.selftest(quick=False, **kwargs)
```

Run unit tests.

```
andes.main.set_logger_level(lg, type_to_set, level)
```

Set logging level for the given type of handler.

## 13.3 andes.plot module

The Andes plotting tool.

```
class andes.plot.TDSData(full_name=None, mode='file', dae=None, path=None)
```

Bases: `object`

A data container for loading and plotting results from Andes time-domain simulation.

```
bqplot_data(xdata, ydata, *, xheader=None, yheader=None, xlabel=None, ylabel=None,
             left=None, right=None, ymin=None, ymax=None, legend=True, grid=False,
             fig=None, latex=True, dpi=150, line_width=1.0, greyscale=False, save-
             fig=None, save_format=None, show=True, title=None, **kwargs)
```

Plot with `bqplot`. Experimental and incomplete.

```
data_to_df()
```

Convert to `pandas.DataFrame`

```
export_csv(path=None, idx=None, header=None, formatted=False, sort_idx=True,
           fmt='%18e')
```

Export to a csv file.

#### Parameters

**path** [str] path of the csv file to save

**idx** [None or array-like, optional] the indices of the variables to export. Export all by default

**header** [None or array-like, optional] customized header if not *None*. Use the names from the *lst* file by default

**formatted** [bool, optional] Use LaTeX-formatted header. Does not apply when using customized header

**sort\_idx** [bool, optional] Sort by *idx* or not, # TODO: implement sort

**fmt** [str] cell formatter

**find** (*query*, *exclude=None*, *formatted=False*, *idx\_only=False*)

Return variable names and indices matching *query*.

#### Parameters

**query** [str] The string for querying variables. Multiple conditions can be separated by comma without space.

**exclude** [str, optional] A string pattern to be excluded

**formatted** [bool, optional] True to return formatted names, False otherwise

**idx\_only** [bool, optional] True if only return indices

#### Returns

(**list**, **list**) (List of found indices, list of found names)

**get\_call** (*backend=None*)

Get the internal *plot\_data* function for the specified backend.

**get\_header** (*idx*, *formatted=False*)

Return a list of the variable names at the given indices.

#### Parameters

**idx** [list or int] The indices of the variables to retrieve

**formatted** [bool] True to retrieve latex-formatted names, False for unformatted names

#### Returns

**list** A list of variable names (headers)

**get\_values** (*idx*)

Return the variable values at the given indices.

#### Parameters

**idx** [list] The index of the variables to retrieve. *idx=0* is for Time. Variable indices start at 1.

#### Returns

**np.ndarray** Variable data

**guess\_event\_time** ()

Guess the event starting time from the input data by checking when the values start to change

**load\_dae()**

Load from DAE time series

**load\_1st()**

Load the 1st file into internal data structures *\_idx*, *\_fname*, *\_uname*, and counts the number of variables to *nvars*.

**Returns**

None

**load\_npy\_or\_csv(delimiter=',')**

Load the npy, zpy or (the legacy) csv file into the internal data structure *self.\_xy*.

**Parameters**

**delimiter** [str, optional] The delimiter for the case file. Default to comma.

**Returns**

None

**plot**(*yidx*, *xidx*=(0, ), \*, *a*=None, *ytimes*=None, *ycalc*=None, *left*=None, *right*=None, *ymin*=None, *ymax*=None, *xlabel*=None, *ylabel*=None, *xheader*=None, *yheader*=None, *legend*=None, *grid*=False, *greyscale*=False, *latex*=True, *dpi*=150, *line\_width*=1.0, *font\_size*=12, *savefig*=None, *save\_format*=None, *show*=True, *title*=None, *linestyles*=None, *use\_bqplot*=False, *hline1*=None, *hline2*=None, *vline1*=None, *vline2*=None, *fig*=None, *ax*=None, *backend*=None, *\*\*kwargs*)  
Entry function for plotting.

This function retrieves the x and y values based on the *xidx* and *yidx* inputs, applies scaling functions *ytimes* and *ycalc* sequentially, and delegates the plotting to the backend.

**Parameters**

**yidx** [list or int] The indices for the y-axis variables

**xidx** [tuple or int, optional] The index for the x-axis variable

**a** [tuple or list, optional] The 0-indexed sub-indices into *yidx* to plot.

**ytimes** [float, optional] A scaling factor to apply to all y values.

**left** [float] The starting value of the x axis

**right** [float] The ending value of the x axis

**ymin** [float] The minimum value of the y axis

**ymax** [float] The maximum value of the y axis

**ylabel** [str] Text label for the y axis

**yheader** [list] A list containing the variable names for the y-axis variable

**title** [str] Title string to be shown at the top

**fig** Existing figure object to draw the axis on.

**ax** Existing axis object to draw the lines on.

**Returns**

**(fig, ax)** Figure and axis handles for matplotlib backend.

**fig** Figure object for bqplot backend.

**Other Parameters**

**ycalc: callable, optional** A callable to apply to all y values after scaling with *ytimes*.

**xlabel** [str] Text label for the x axis

**xheader** [list] A list containing the variable names for the x-axis variable

**legend** [bool] True to show legend and False otherwise

**grid** [bool] True to show grid and False otherwise

**latex** [bool] True to enable latex and False to disable

**greyscale** [bool] True to use greyscale, False otherwise

**savefig** [bool] True to save to png figure file

**save\_format** [str] File extension string (pdf, png or jpg) for the savefig format

**dpi** [int] Dots per inch for screen print or save. *savefig* uses a minimum of 200 dpi

**line\_width** [float] Plot line width

**font\_size** [float] Text font size (labels and legends)

**show** [bool] True to show the image

**backend** [str or None] *bqplot* to use the bqplot backend in notebook. None for matplotlib.

**hline1: float, optional** Dashed horizontal line 1

**hline2: float, optional** Dashed horizontal line 2

**vline1: float, optional** Dashed horizontal line 1

**vline2: float, optional** Dashed vertical line 2

**plot\_data** (*xdata*, *ydata*, \*, *xheader=None*, *yheader=None*, *xlabel=None*, *ylabel=None*, *linestyles=None*, *left=None*, *right=None*, *ymin=None*, *ymax=None*, *legend=None*, *grid=False*, *fig=None*, *ax=None*, *latex=True*, *dpi=150*, *line\_width=1.0*, *font\_size=12*, *greyscale=False*, *savefig=None*, *save\_format=None*, *show=True*, *title=None*, *hline1=None*, *hline2=None*, *vline1=None*, *vline2=None*, *\*\*kwargs*)

Plot lines for the supplied data and options.

This functions takes *xdata* and *ydata* values. If you provide variable indices instead of values, use *plot()*.

See the argument lists of *plot()* for more.

**Parameters**

**xdata** [array-like] An array-like object containing the values for the x-axis variable

**ydata** [array] An array containing the values of each variables for the y-axis variable. The row of *ydata* must match the row of *xdata*. Each column correspondings to a variable.

### Returns

(**fig**, **ax**) The figure and axis handles

`andes.plot.add_plot(x, y, xl, yl, fig, ax, LATEX=False, linestyle=None, **kwargs)`  
Add plots to an existing plot

`andes.plot.check_init(yval, yl)`  
" Check initialization by comparing t=0 and t=end values for a flat run.

**Warning:** This function is deprecated as the initialization check feature is built into TDS. See `TDS.test_initialization()`.

`andes.plot.eig_plot(name, args)`

`andes.plot.isfloat(value)`

`andes.plot.isint(value)`

`andes.plot.label_latexify(label)`  
Convert a label to latex format by appending surrounding \$ and escaping spaces

### Parameters

**label** [str] The label string to be converted to latex expression

### Returns

**str** A string with \$ surrounding

`andes.plot.parse_y(y, upper, lower=0)`  
Parse command-line input for Y indices and return a list of indices

### Parameters

**y** [Union[List, Set, Tuple]]

**Input for Y indices. Could be single item (with or without colon), or multiple items**

**upper** [int] Upper limit. In the return list y,  $y[i] \leq upper$ .

**lower** [int] Lower limit. In the return list y,  $y[i] \geq lower$ .

`andes.plot.scale_func(k)`  
Return a lambda function that scales its input by k

### Parameters

**k** [float] The scaling factor of the returned lambda function



**Returns**

—

**Lambda function**`andes.plot.set_latex(enable=True)`Enables LaTeX for matplotlib based on the *with\_latex* option and *dvipng* availability.**Parameters****enable** [bool, optional] True for latex on and False for off**Returns****bool** True for LaTeX on, False for off`andes.plot.tdsplot(filename, y, x=(0, ), tocsv=False, find=None, xargs=None, exclude=None, **kwargs)`

TDS plot main function based on the new TDSData class.

**Parameters****filename** [str] Path to the ANDES TDS output data file. Works without extension.**x** [list or int, optional] The index for the x-axis variable. x=0 by default for time**y** [list or int] The indices for the y-axis variable**tocsv** [bool] True if need to export to a csv file**find** [str, optional] if not none, specify the variable name to find**xargs** [str, optional] similar to find, but return the result indices with file name, x idx name for xargs**exclude** [str, optional] variable name pattern to exclude**Returns****TDSData object**

## 13.4 andes.shared module

Shared constants and delayed imports.

## 13.5 andes.system module

System class for power system data and methods

**class** `andes.system.ExistingModels`Bases: `object`

Storage class for existing models

```
class andes.system.System(case: Optional[str] = None, name: Optional[str] = None,  
                           config_path: Optional[str] = None, default_config: Op-  
                           tional[bool] = False, options: Optional[Dict[KT, VT]] =  
                           None, **kwargs)
```

Bases: `object`

System contains models and routines for modeling and simulation.

System contains a several special *OrderedDict* member attributes for housekeeping. These attributes include *models*, *groups*, *routines* and *calls* for loaded models, groups, analysis routines, and generated numerical function calls, respectively.

## Notes

System stores model and routine instances as attributes. Model and routine attribute names are the same as their class names. For example, *Bus* is stored at `system.Bus`, the power flow calculation routine is at `system.PFlow`, and the numerical DAE instance is at `system.dae`. See attributes for the list of attributes.

### Attributes

**dae** [andes.variables.dae.DAE] Numerical DAE storage

**files** [andes.variables.fileman.FileMan] File path storage

**config** [andes.core.Config] System config storage

**models** [OrderedDict] model name and instance pairs

**groups** [OrderedDict] group name and instance pairs

**routines** [OrderedDict] routine name and instance pairs

**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.

**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

**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.

**call\_models** (*method*: str, *models*: collections.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**

**collect\_ref()**

Collect indices into *BackRef* for all models.

**dill()**

Serialize generated numerical functions in *System.calls* with package *dill*.

The serialized file will be stored to `~/andes/calls.pkl`, where `~` is the home directory path.

### Notes

This function sets `dill.settings['recurse'] = True` to serialize the function calls recursively.

**e\_clear** (*models: collections.OrderedDict*)

Clear equation arrays in DAE and model variables.

This step must be called before calling *f\_update* or *g\_update* to flush existing values.

**f\_update** (*models: collections.OrderedDict*)

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.

**fg\_to\_dae()**

Collect equation values into the DAE arrays.

Additionally, the function resets the differential equations associated with variables pegged by anti-windup limiters.

**find\_devices()**

Add dependent devices for all model based on *DeviceFinder*.

**find\_models** (*flag: Union[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.

**g\_update** (*models: collections.OrderedDict*)

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.

**get\_config** ()

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.

**get\_z** (*models: Union[str, List[T], collections.OrderedDict, None] = None*)

Get all discrete status flags in a numpy array.

**Returns**

**numpy.array**

**import\_groups** ()

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

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

**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.GENCLS` stores the classical generator object,

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

**import\_routines** ()

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

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

## Examples

`System.PFlow` is the power flow routine instance, and `System.TDS` and `System.EIG` are time-domain analysis and eigenvalue analysis routines, respectively.

**init** (*models: collections.OrderedDict, routine: str*)

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.

**j\_update** (*models: collections.OrderedDict, info=None*)

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).

**l\_update\_eq** (*models: collections.OrderedDict*)

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.

**l\_update\_var** (*models: collections.OrderedDict*)

Update variable-based limiter discrete states by calling `l_update_var` of models.

This function is must be called before any equation evaluation.

**link\_ext\_param** (*model=None*)

Retrieve values for `ExtParam` for the given models.

**static load\_config** (*conf\_path=None*)

Load config from an rc-formatted file.

### Parameters

**conf\_path** [None or str] Path to the config file. If is *None*, the function body will not run.

### Returns

`configparse.ConfigParser`

**prepare** (*quick=False, incremental=False*)

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.

**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()
```

**reload** (*case, \*\*kwargs*)

Reload a new case in the same System object.

**remove\_pycapsule** ()

Remove PyCapsule objects in solvers.

**reset** (*force=False*)

Reset to the state after reading data and setup (before power flow).

**Warning:** If TDS is initialized, reset will lead to unpredictable state.

**s\_update\_post** (*models: collections.OrderedDict*)

Update variable services by calling `s_update_post` of models.

This function is called at the end of *System.init()*.

**s\_update\_var** (*models: collections.OrderedDict*)

Update variable services by calling *s\_update\_var* of models.

This function must be called before any equation evaluation after limiter update function *l\_update\_var*.

**save\_config** (*file\_path=None, overwrite=False*)

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.

**set\_address** (*models*)

Set addresses for differential and algebraic variables.

**set\_config** (*config=None*)

Set configuration for the System object.

Config for models are routines are passed directly to their constructors.

**set\_dae\_names** (*models*)

Set variable names for differential and algebraic variables, and discrete flags.

**setup** ()

Set up system for studies.

This function is to be called after adding all device data.

**store\_adder\_setter** (*models*)

Store non-inplace adders and setters for variables and equations.

**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.

**store\_sparse\_pattern** (*models: collections.OrderedDict*)

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.

**store\_switch\_times** (*models*, *eps*=0.0001)

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`

**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

**switch\_action** (*models*)

Invoke the actions associated with switch times.

Switch actions will be disabled if *flat=True* is passed to system.

**undill** ()

Deserialize the function calls from `~/andes.calls.pkl` with dill.

If no change is made to models, future calls to `prepare()` can be replaced with `undill()` for acceleration.

**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.

**vars\_to\_models** ()

Copy variable values from `System.dae` to models.



## CHAPTER 14

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### a

`andes.cli`, 327  
`andes.core`, 301  
`andes.core.block`, 233  
`andes.core.common`, 295  
`andes.core.discrete`, 251  
`andes.core.model`, 260  
`andes.core.param`, 272  
`andes.core.service`, 278  
`andes.core.solver`, 291  
`andes.core.var`, 297  
`andes.io`, 303  
`andes.io.matpower`, 302  
`andes.io.psse`, 302  
`andes.io.txt`, 302  
`andes.io.xlsx`, 303  
`andes.main`, 327  
`andes.models`, 314  
`andes.models.area`, 304  
`andes.models.bus`, 305  
`andes.models.governor`, 305  
`andes.models.group`, 306  
`andes.models.jit`, 311  
`andes.models.line`, 311  
`andes.models.pq`, 311  
`andes.models.pv`, 312  
`andes.models.shunt`, 312  
`andes.models.synchronous`, 312  
`andes.models.timer`, 313  
`andes.plot`, 330  
`andes.routines`, 318  
`andes.routines.base`, 314  
`andes.routines.eig`, 314  
`andes.routines.pflow`, 315  
`andes.routines.tds`, 316  
`andes.shared`, 335  
`andes.system`, 335  
`andes.utils`, 321  
`andes.utils.cached`, 318  
`andes.utils.func`, 320  
`andes.utils.misc`, 320  
`andes.utils.paths`, 318  
`andes.utils.tab`, 320  
`andes.variables`, 325  
`andes.variables.dae`, 321  
`andes.variables.fileman`, 325  
`andes.variables.report`, 325



## A

- `a_reset()` (*andes.core.model.Model* method), 263
- ACE* (class in *andes.models.area*), 304
- ACEData* (class in *andes.models.area*), 304
- ACLine* (class in *andes.models.group*), 306
- ACTopology* (class in *andes.models.group*), 306
- `add()` (*andes.core.common.Config* method), 295
- `add()` (*andes.core.model.ModelData* method), 269
- `add()` (*andes.core.param.BaseParam* method), 272
- `add()` (*andes.core.param.ExtParam* method), 274
- `add()` (*andes.core.param.IdxParam* method), 275
- `add()` (*andes.core.param.NumParam* method), 276
- `add()` (*andes.models.group.GroupBase* method), 307
- `add()` (*andes.system.System* method), 336
- `add_callback()` (*andes.core.model.Cache* method), 260
- `add_extra()` (*andes.core.common.Config* method), 295
- `add_model()` (*andes.models.group.GroupBase* method), 307
- `add_plot()` (in module *andes.plot*), 334
- `add_suffix()` (in module *andes.variables.fileman*), 325
- Algeb* (class in *andes.core.var*), 297
- AliasAlgeb* (class in *andes.core.var*), 297
- AliasState* (class in *andes.core.var*), 297
- `alter()` (*andes.core.model.Model* method), 263
- andes.cli* (module), 327
- andes.core* (module), 301
- andes.core.block* (module), 233
- andes.core.common* (module), 295
- andes.core.discrete* (module), 251
- andes.core.model* (module), 260
- andes.core.param* (module), 272
- andes.core.service* (module), 278
- andes.core.solver* (module), 291
- andes.core.var* (module), 297
- andes.io* (module), 303
- andes.io.matpower* (module), 302
- andes.io.psse* (module), 302
- andes.io.txt* (module), 302
- andes.io.xlsx* (module), 303
- andes.main* (module), 327
- andes.models* (module), 314
- andes.models.area* (module), 304
- andes.models.bus* (module), 305
- andes.models.governor* (module), 305
- andes.models.group* (module), 306
- andes.models.jit* (module), 311
- andes.models.line* (module), 311
- andes.models.pq* (module), 311
- andes.models.pv* (module), 312
- andes.models.shunt* (module), 312
- andes.models.synchronous* (module), 312
- andes.models.timer* (module), 313
- andes.plot* (module), 330
- andes.routines* (module), 318
- andes.routines.base* (module), 314
- andes.routines.eig* (module), 314
- andes.routines.pflow* (module), 315
- andes.routines.tds* (module), 316
- andes.shared* (module), 335
- andes.system* (module), 335
- andes.utils* (module), 321
- andes.utils.cached* (module), 318
- andes.utils.func* (module), 320
- andes.utils.misc* (module), 320
- andes.utils.paths* (module), 318

`andes.utils.tab` (module), 320  
`andes.variables` (module), 325  
`andes.variables.dae` (module), 321  
`andes.variables.fileman` (module), 325  
`andes.variables.report` (module), 325  
`AntiWindup` (class in *andes.core.discrete*), 251  
`AntiWindupRate` (class in *andes.core.discrete*), 252  
`append_ijv()` (*andes.core.common.JacTriplet* method), 295  
`append_ijv()` (*andes.core.model.ModelCall* method), 268  
`apply_fault()` (*andes.models.timer.Fault* method), 313  
`ApplyFunc` (class in *andes.core.service*), 278  
`Area` (class in *andes.models.area*), 304  
`AreaData` (class in *andes.models.area*), 304  
`as_df()` (*andes.core.model.ModelData* method), 269  
`as_df_in()` (*andes.core.model.ModelData* method), 269  
`as_dict()` (*andes.core.common.Config* method), 295  
`as_dict()` (*andes.core.model.ModelData* method), 269  
`as_dict()` (*andes.system.System* method), 336  
`assign_memory()` (*andes.core.service.ConstService* method), 280  
`assign_memory()` (*andes.core.service.ExtendedEvent* method), 283  
`assign_memory()` (*andes.core.service.ExtService* method), 283  
`Average` (class in *andes.core.discrete*), 252

## B

`BackRef` (class in *andes.core.service*), 278  
`BaseParam` (class in *andes.core.param*), 272  
`BaseRoutine` (class in *andes.routines.base*), 314  
`BaseService` (class in *andes.core.service*), 279  
`BaseVar` (class in *andes.core.var*), 297  
`Block` (class in *andes.core.block*), 233  
`bqplot_data()` (*andes.plot.TDSData* method), 330  
`build_pattern()` (*andes.variables.dae.DAE* method), 322

`Bus` (class in *andes.models.bus*), 305  
`bus_table()` (*andes.models.area.Area* method), 304  
`BusData` (class in *andes.models.bus*), 305

## C

`Cache` (class in *andes.core.model*), 260  
`cached` (class in *andes.utils.cached*), 318  
`calc_eigvals()` (*andes.routines.eig.EIG* method), 314  
`calc_h()` (*andes.routines.tds.TDS* method), 316  
`calc_part_factor()` (*andes.routines.eig.EIG* method), 314  
`calc_pu_coeff()` (*andes.system.System* method), 336  
`calc_state_matrix()` (*andes.routines.eig.EIG* method), 314  
`Calculation` (class in *andes.models.group*), 306  
`call_models()` (*andes.system.System* method), 336  
`cases_root()` (in module *andes.utils.paths*), 318  
`check()` (*andes.core.common.Config* method), 295  
`check()` (*andes.core.service.CurrentSign* method), 281  
`check()` (*andes.core.service.EventFlag* method), 282  
`check()` (*andes.core.service.ExtendedEvent* method), 283  
`check()` (*andes.core.service.InitChecker* method), 285  
`check()` (*andes.core.service.VarHold* method), 290  
`check_eq()` (*andes.core.discrete.AntiWindup* method), 251  
`check_eq()` (*andes.core.discrete.AntiWindupRate* method), 252  
`check_eq()` (*andes.core.discrete.Discrete* method), 255  
`check_eq()` (*andes.core.discrete.RateLimiter* method), 257  
`check_init()` (in module *andes.plot*), 334  
`check_var()` (*andes.core.discrete.AntiWindup* method), 252  
`check_var()` (*andes.core.discrete.Average* method), 252

- `check_var()` (*andes.core.discrete.DeadBand method*), 253  
`check_var()` (*andes.core.discrete.DeadBandRT method*), 254  
`check_var()` (*andes.core.discrete.Delay method*), 255  
`check_var()` (*andes.core.discrete.Derivative method*), 255  
`check_var()` (*andes.core.discrete.Discrete method*), 255  
`check_var()` (*andes.core.discrete.LessThan method*), 256  
`check_var()` (*andes.core.discrete.Limiter method*), 257  
`check_var()` (*andes.core.discrete.Sampling method*), 258  
`check_var()` (*andes.core.discrete.Selector method*), 259  
`check_var()` (*andes.core.discrete.SortedLimiter method*), 259  
`check_var()` (*andes.core.discrete.Switcher method*), 260  
`class_name` (*andes.core.block.Block attribute*), 235  
`class_name` (*andes.core.discrete.Discrete attribute*), 255  
`class_name` (*andes.core.model.Model attribute*), 263  
`class_name` (*andes.core.param.BaseParam attribute*), 273  
`class_name` (*andes.core.service.BaseService attribute*), 280  
`class_name` (*andes.core.var.BaseVar attribute*), 298  
`class_name` (*andes.models.group.GroupBase attribute*), 307  
`class_name` (*andes.routines.base.BaseRoutine attribute*), 314  
`clear()` (*andes.core.solver.SciPySolver method*), 292  
`clear()` (*andes.core.solver.Solver method*), 292  
`clear()` (*andes.core.solver.SuiteSparseSolver method*), 293  
`clear_arrays()` (*andes.variables.dae.DAE method*), 322  
`clear_fault()` (*andes.models.timer.Fault method*), 313  
`clear_fg()` (*andes.variables.dae.DAE method*), 322  
`clear_ijv()` (*andes.core.common.JacTriplet method*), 296  
`clear_ijv()` (*andes.core.model.ModelCall method*), 268  
`clear_ijv()` (*andes.variables.dae.DAE method*), 322  
`clear_ts()` (*andes.variables.dae.DAE method*), 322  
`clear_xy()` (*andes.variables.dae.DAE method*), 322  
`clear_z()` (*andes.variables.dae.DAE method*), 322  
`collect_ref()` (*andes.system.System method*), 337  
`Collection` (*class in andes.models.group*), 306  
`Config` (*class in andes.core.common*), 295  
`config_logger()` (*in module andes.main*), 327  
`confirm_overwrite()` (*in module andes.utils.paths*), 319  
`ConstService` (*class in andes.core.service*), 280  
`create_parser()` (*in module andes.cli*), 327  
`CuPySolver` (*class in andes.core.solver*), 291  
`CurrentSign` (*class in andes.core.service*), 280
- ## D
- `DAE` (*class in andes.variables.dae*), 321  
`DAETimeSeries` (*class in andes.variables.dae*), 324  
`data_to_df()` (*andes.plot.TDSData method*), 330  
`DataParam` (*class in andes.core.param*), 273  
`DataSelect` (*class in andes.core.service*), 281  
`DCLink` (*class in andes.models.group*), 307  
`DCTopology` (*class in andes.models.group*), 307  
`DeadBand` (*class in andes.core.discrete*), 252  
`DeadBand1` (*class in andes.core.block*), 236  
`DeadBandRT` (*class in andes.core.discrete*), 253  
`define()` (*andes.core.block.Block method*), 235  
`define()` (*andes.core.block.DeadBand1 method*), 237  
`define()` (*andes.core.block.Gain method*), 237  
`define()` (*andes.core.block.GainLimiter method*), 237  
`define()` (*andes.core.block.HVGate method*), 238  
`define()` (*andes.core.block.Integrator method*), 238

`define()` (*andes.core.block.IntegratorAntiWindup method*), 239  
`define()` (*andes.core.block.Lag method*), 240  
`define()` (*andes.core.block.Lag2ndOrd method*), 240  
`define()` (*andes.core.block.LagAntiWindup method*), 241  
`define()` (*andes.core.block.LagAntiWindupRate method*), 242  
`define()` (*andes.core.block.LagAWFreeze method*), 241  
`define()` (*andes.core.block.LagFreeze method*), 242  
`define()` (*andes.core.block.LagRate method*), 243  
`define()` (*andes.core.block.LeadLag method*), 244  
`define()` (*andes.core.block.LeadLag2ndOrd method*), 244  
`define()` (*andes.core.block.LeadLagLimit method*), 245  
`define()` (*andes.core.block.LVGate method*), 239  
`define()` (*andes.core.block.PIAWHardLimit method*), 245  
`define()` (*andes.core.block.PIController method*), 246  
`define()` (*andes.core.block.PIControllerNumeric method*), 246  
`define()` (*andes.core.block.Piecewise method*), 250  
`define()` (*andes.core.block.PIFreeze method*), 247  
`define()` (*andes.core.block.PITrackAW method*), 247  
`define()` (*andes.core.block.PITrackAWFreeze method*), 248  
`define()` (*andes.core.block.Washout method*), 250  
`define()` (*andes.core.block.WashoutOrLag method*), 251  
`Delay` (*class in andes.core.discrete*), 255  
`Derivative` (*class in andes.core.discrete*), 255  
`DeviceFinder` (*class in andes.core.service*), 281  
`DG` (*class in andes.models.group*), 307  
`dill()` (*andes.system.System method*), 337  
`Discrete` (*class in andes.core.discrete*), 255  
`display_filename_prefix_last` (*andes.utils.paths.DisplayablePath attribute*), 318  
`display_filename_prefix_middle` (*andes.utils.paths.DisplayablePath attribute*), 318  
`display_parent_prefix_last` (*andes.utils.paths.DisplayablePath attribute*), 318  
`display_parent_prefix_middle` (*andes.utils.paths.DisplayablePath attribute*), 318  
`displayable()` (*andes.utils.paths.DisplayablePath method*), 318  
`DisplayablePath` (*class in andes.utils.paths*), 318  
`displayname` (*andes.utils.paths.DisplayablePath attribute*), 318  
`do_switch()` (*andes.routines.tds.TDS method*), 316  
`doc()` (*andes.core.common.Config method*), 295  
`doc()` (*andes.core.model.Model method*), 263  
`doc()` (*andes.models.group.GroupBase method*), 307  
`doc()` (*andes.models.jit.JIT method*), 311  
`doc()` (*andes.routines.base.BaseRoutine method*), 314  
`doc()` (*in module andes.main*), 328  
`doc_all()` (*andes.models.group.GroupBase method*), 308  
`Documenter` (*class in andes.core.model*), 260  
`draw()` (*andes.utils.tab.Tab method*), 321  
`dummify()` (*in module andes.core.common*), 296  
`DummyValue` (*class in andes.core.common*), 295  
`dump()` (*in module andes.io*), 303  
`dump_data()` (*in module andes.io.txt*), 302

## E

`e_clear()` (*andes.core.model.Model method*), 263  
`e_clear()` (*andes.system.System method*), 337  
`e_code` (*andes.core.var.Algeb attribute*), 297  
`e_code` (*andes.core.var.ExtAlgeb attribute*), 299  
`e_code` (*andes.core.var.ExtState attribute*), 299  
`e_code` (*andes.core.var.State attribute*), 301  
`edit_conf()` (*in module andes.main*), 328  
`EIG` (*class in andes.routines.eig*), 314  
`eig_plot()` (*in module andes.plot*), 334  
`elapsed()` (*in module andes.utils.misc*), 320



- `elem_add()` (*andes.models.jit.JIT method*), 311  
`enforce_tex_name()` (*andes.core.block.Block static method*), 236  
`EventFlag` (*class in andes.core.service*), 282  
`Exciter` (*class in andes.models.group*), 307  
`ExistingModels` (*class in andes.system*), 335  
`Experimental` (*class in andes.models.group*), 307  
`export()` (*andes.core.block.Block method*), 236  
`export_csv()` (*andes.plot.TDSData method*), 330  
`export_state_matrix()` (*andes.routines.eig.EIG method*), 315  
`ExtAlgeb` (*class in andes.core.var*), 299  
`ExtendedEvent` (*class in andes.core.service*), 283  
`ExtParam` (*class in andes.core.param*), 273  
`ExtService` (*class in andes.core.service*), 282  
`ExtState` (*class in andes.core.var*), 299  
`ExtVar` (*class in andes.core.var*), 299
- ## F
- `f_numeric()` (*andes.core.block.Block method*), 236  
`f_numeric()` (*andes.core.block.PIDControllerNumeric method*), 246  
`f_numeric()` (*andes.core.model.Model method*), 263  
`f_update()` (*andes.core.model.Model method*), 263  
`f_update()` (*andes.system.System method*), 337  
`Fault` (*class in andes.models.timer*), 313  
`fg` (*andes.variables.dae.DAE attribute*), 322  
`fg_to_dae()` (*andes.system.System method*), 337  
`FileMan` (*class in andes.variables.fileman*), 325  
`find()` (*andes.plot.TDSData method*), 331  
`find_devices()` (*andes.system.System method*), 337  
`find_idx()` (*andes.core.model.ModelData method*), 270  
`find_idx()` (*andes.models.group.GroupBase method*), 308  
`find_log_path()` (*in module andes.main*), 328  
`find_models()` (*andes.system.System method*), 337  
`find_or_add()` (*andes.core.service.DeviceFinder method*), 282  
`find_param()` (*andes.core.model.ModelData method*), 270  
`FlagCondition` (*class in andes.core.service*), 283  
`FlagGreaterThan` (*class in andes.core.service*), 284  
`FlagLessThan` (*class in andes.core.service*), 284  
`FlagValue` (*class in andes.core.service*), 284  
`Flux0` (*class in andes.models.synchronous*), 312  
`Flux1` (*class in andes.models.synchronous*), 312  
`Flux2` (*class in andes.models.synchronous*), 312  
`FreqMeasurement` (*class in andes.models.group*), 307
- ## G
- `g_numeric()` (*andes.core.block.Block method*), 236  
`g_numeric()` (*andes.core.block.PIDControllerNumeric method*), 246  
`g_numeric()` (*andes.core.model.Model method*), 264  
`g_update()` (*andes.core.model.Model method*), 264  
`g_update()` (*andes.system.System method*), 338  
`Gain` (*class in andes.core.block*), 237  
`GainLimiter` (*class in andes.core.block*), 237  
`GENBase` (*class in andes.models.synchronous*), 312  
`GENBaseData` (*class in andes.models.synchronous*), 313  
`GENCLS` (*class in andes.models.synchronous*), 313  
`GENCLSModel` (*class in andes.models.synchronous*), 313  
`generate_equations()` (*andes.core.model.SymProcessor method*), 271  
`generate_init()` (*andes.core.model.SymProcessor method*), 271  
`generate_jacobians()` (*andes.core.model.SymProcessor method*), 271  
`generate_pretty_print()` (*andes.core.model.SymProcessor method*), 271  
`generate_pycode()` (*andes.core.model.SymProcessor method*), 271

271  
generate\_services() (andes.core.model.SymProcessor method), 271  
generate\_symbols() (andes.core.model.SymProcessor method), 271  
GENROU (class in andes.models.synchronous), 313  
GENROUData (class in andes.models.synchronous), 313  
GENROUModel (class in andes.models.synchronous), 313  
get() (andes.core.model.Documenter method), 261  
get() (andes.core.model.Model method), 264  
get() (andes.models.group.GroupBase method), 308  
get\_block\_lines() (in module andes.io.psse), 302  
get\_call() (andes.plot.TDSDData method), 331  
get\_case() (in module andes.utils.paths), 319  
get\_config() (andes.system.System method), 338  
get\_config\_path() (in module andes.utils.paths), 319  
get\_dot\_andes\_path() (in module andes.utils.paths), 319  
get\_fullpath() (andes.variables.fileman.FileMan method), 325  
get\_header() (andes.plot.TDSDData method), 331  
get\_init\_order() (andes.core.model.Model method), 264  
get\_inputs() (andes.core.model.Model method), 264  
get\_log\_dir() (in module andes.utils.paths), 319  
get\_md5() (andes.core.model.Model method), 264  
get\_name() (andes.variables.dae.DAE method), 322  
get\_names() (andes.core.discrete.Discrete method), 255  
get\_names() (andes.core.param.BaseParam method), 273  
get\_names() (andes.core.service.BaseService method), 280  
get\_names() (andes.core.var.BaseVar method), 298  
get\_next\_idx() (andes.models.group.GroupBase method), 308  
get\_output\_ext() (in module andes.io), 303  
get\_pkl\_path() (in module andes.utils.paths), 319  
get\_property() (andes.core.param.BaseParam method), 273  
get\_size() (andes.variables.dae.DAE method), 322  
get\_tex\_names() (andes.core.discrete.Discrete method), 255  
get\_times() (andes.core.model.Model method), 264  
get\_values() (andes.core.discrete.Discrete method), 256  
get\_values() (andes.plot.TDSDData method), 331  
get\_z() (andes.system.System method), 338  
GroupBase (class in andes.models.group), 307  
guess() (in module andes.io), 304  
guess\_event\_time() (andes.plot.TDSDData method), 331

## H

HardLimiter (class in andes.core.discrete), 256  
header() (andes.utils.tab.Tab method), 321  
HVGate (class in andes.core.block), 238

## I

idx2model() (andes.models.group.GroupBase method), 308  
idx2uid() (andes.core.model.Model method), 265  
idx2uid() (andes.models.group.GroupBase method), 309  
IdxParam (class in andes.core.param), 274  
IdxRepeat (class in andes.core.service), 284  
IEEEG1 (class in andes.models.governor), 305  
IEEEG1Data (class in andes.models.governor), 305  
IEEEG1Model (class in andes.models.governor), 305  
ijv() (andes.core.common.JacTriplet method), 296

import\_groups() (*andes.system.System* method), 338  
 import\_models() (*andes.system.System* method), 338  
 import\_routines() (*andes.system.System* method), 338  
 info(*andes.variables.report.Report* attribute), 325  
 init() (*andes.core.model.Model* method), 265  
 init() (*andes.routines.base.BaseRoutine* method), 314  
 init() (*andes.routines.pflow.PFlow* method), 315  
 init() (*andes.routines.tds.TDS* method), 317  
 init() (*andes.system.System* method), 339  
 init\_iter() (*andes.core.model.Model* method), 265  
 InitChecker (class in *andes.core.service*), 285  
 Integrator (class in *andes.core.block*), 238  
 IntegratorAntiWindup (class in *andes.core.block*), 238  
 interp\_n2() (in module *andes.utils.func*), 320  
 is\_interactive() (in module *andes.utils.misc*), 320  
 is\_notebook() (in module *andes.utils.misc*), 320  
 is\_time() (*andes.core.param.TimerParam* method), 277  
 isfloat() (in module *andes.plot*), 334  
 isint() (in module *andes.plot*), 334  
**J**  
 j\_numeric() (*andes.core.block.Block* method), 236  
 j\_numeric() (*andes.core.block.PIControllerNumeric* method), 247  
 j\_numeric() (*andes.core.model.Model* method), 265  
 j\_reset() (*andes.core.block.Block* method), 236  
 j\_update() (*andes.core.model.Model* method), 265  
 j\_update() (*andes.system.System* method), 339  
 JacTriplet (class in *andes.core.common*), 295  
 JIT (class in *andes.models.jit*), 311  
 jit\_load() (*andes.models.jit.JIT* method), 311  
**K**  
 KLU Solver (class in *andes.core.solver*), 291

**L**  
 l\_check\_eq() (*andes.core.model.Model* method), 265  
 l\_update\_eq() (*andes.system.System* method), 339  
 l\_update\_var() (*andes.core.model.Model* method), 265  
 l\_update\_var() (*andes.system.System* method), 339  
 label\_latexify() (in module *andes.plot*), 334  
 Lag (class in *andes.core.block*), 239  
 Lag2ndOrd (class in *andes.core.block*), 240  
 LagAntiWindup (class in *andes.core.block*), 241  
 LagAntiWindupRate (class in *andes.core.block*), 241  
 LagAWFreeze (class in *andes.core.block*), 240  
 LagFreeze (class in *andes.core.block*), 242  
 LagRate (class in *andes.core.block*), 243  
 LeadLag (class in *andes.core.block*), 243  
 LeadLag2ndOrd (class in *andes.core.block*), 244  
 LeadLagLimit (class in *andes.core.block*), 245  
 LessThan (class in *andes.core.discrete*), 256  
 Limiter (class in *andes.core.discrete*), 256  
 Line (class in *andes.models.line*), 311  
 LineData (class in *andes.models.line*), 311  
 link\_ext\_param() (*andes.system.System* method), 339  
 link\_external() (*andes.core.param.ExtParam* method), 274  
 link\_external() (*andes.core.service.ExtService* method), 283  
 link\_external() (*andes.core.var.ExtVar* method), 300  
 linsolve() (*andes.core.solver.KLUSolver* method), 291  
 linsolve() (*andes.core.solver.SciPySolver* method), 292  
 linsolve() (*andes.core.solver.Solver* method), 293  
 linsolve() (*andes.core.solver.SuiteSparseSolver* method), 293  
 linsolve() (*andes.core.solver.UMFPACKSolver* method), 294  
 list2array() (*andes.core.discrete.Delay* method), 255  
 list2array() (*andes.core.discrete.Discrete*

*method*), 256  
list2array() (*andes.core.discrete.Sampling*  
*method*), 258  
list2array() (*andes.core.discrete.Switcher*  
*method*), 260  
list2array() (*andes.core.model.Model*  
*method*), 265  
list\_cases() (*in module andes.utils.paths*), 319  
list\_flatten() (*in module andes.utils.func*),  
320  
load() (*andes.core.common.Config method*), 295  
load() (*in module andes.main*), 328  
load\_config() (*andes.system.System static*  
*method*), 339  
load\_dae() (*andes.plot.TDSData method*), 331  
load\_lst() (*andes.plot.TDSData method*), 332  
load\_npy\_or\_csv() (*andes.plot.TDSData*  
*method*), 332  
load\_plotter() (*andes.routines.tds.TDS*  
*method*), 317  
LVGate (*class in andes.core.block*), 239

## M

main() (*in module andes.cli*), 327  
make\_doc\_table() (*in module andes.utils.tab*),  
321  
make\_tree() (*andes.utils.paths.DisplayablePath*  
*class method*), 318  
math\_wrap() (*in module andes.utils.tab*), 321  
merge() (*andes.core.common.JacTriplet method*),  
296  
misc() (*in module andes.main*), 328  
Model (*class in andes.core.model*), 261  
ModelCall (*class in andes.core.model*), 268  
ModelData (*class in andes.core.model*), 268  
ModelFlags (*class in andes.core.common*), 296  
Motor (*class in andes.models.group*), 309

## N

n (*andes.core.param.BaseParam attribute*), 273  
n (*andes.core.service.BaseService attribute*), 280  
n (*andes.models.group.GroupBase attribute*), 309  
newton\_krylov() (*andes.routines.pflow.PFlow*  
*method*), 315  
nr\_step() (*andes.routines.pflow.PFlow method*),  
316  
numba\_jitify() (*andes.core.model.Model*  
*method*), 266

NumParam (*class in andes.core.param*), 275  
NumReduce (*class in andes.core.service*), 285  
NumRepeat (*class in andes.core.service*), 287  
NumSelect (*class in andes.core.service*), 288

## O

OperationService (*class in an-*  
*des.core.service*), 288

## P

ParamCalc (*class in andes.core.service*), 288  
parse() (*in module andes.io*), 304  
parse\_y() (*in module andes.plot*), 334  
PFlow (*class in andes.routines.pflow*), 315  
PhasorMeasurement (*class in an-*  
*des.models.group*), 309  
PIAWHardLimit (*class in andes.core.block*), 245  
PIController (*class in andes.core.block*), 246  
PIControllerNumeric (*class in an-*  
*des.core.block*), 246  
Piecewise (*class in andes.core.block*), 249  
PIFreeze (*class in andes.core.block*), 247  
PITrackAW (*class in andes.core.block*), 247  
PITrackAWFreeze (*class in andes.core.block*),  
248  
plot() (*andes.plot.TDSData method*), 332  
plot() (*andes.routines.eig.EIG method*), 315  
plot() (*in module andes.main*), 328  
plot\_data() (*andes.plot.TDSData method*), 333  
post\_init\_check() (*andes.core.model.Model*  
*method*), 266  
PostInitService (*class in andes.core.service*),  
289  
PQ (*class in andes.models.pq*), 311  
PQData (*class in andes.models.pq*), 312  
preamble() (*in module andes.cli*), 327  
prepare() (*andes.core.model.Model method*),  
266  
prepare() (*andes.system.System method*), 339  
prepare() (*in module andes.main*), 328  
print\_array() (*andes.variables.dae.DAE*  
*method*), 323  
print\_license() (*in module andes.main*), 329  
PSS (*class in andes.models.group*), 309  
PV (*class in andes.models.pv*), 312  
PVData (*class in andes.models.pv*), 312  
PVModel (*class in andes.models.pv*), 312

## R

RandomService (class in *andes.core.service*), 289

RateLimiter (class in *andes.core.discrete*), 257

read() (in module *andes.io.matpower*), 302

read() (in module *andes.io.psse*), 302

read() (in module *andes.io.xlsx*), 303

read\_add() (in module *andes.io.psse*), 302

RefFlatten (class in *andes.core.service*), 290

refresh() (*andes.core.model.Cache* method), 260

refresh\_inputs() (*andes.core.model.Model* method), 266

refresh\_inputs\_arg() (*andes.core.model.Model* method), 266

reload() (*andes.system.System* method), 340

remove\_output() (in module *andes.main*), 329

remove\_pycapsule() (*andes.system.System* method), 340

remove\_singular\_rc() (*andes.routines.eig.EIG* method), 315

RenAerodynamics (class in *andes.models.group*), 309

RenExciter (class in *andes.models.group*), 310

RenGen (class in *andes.models.group*), 310

RenGovernor (class in *andes.models.group*), 310

RenPitch (class in *andes.models.group*), 310

RenPlant (class in *andes.models.group*), 310

RenTorque (class in *andes.models.group*), 310

Replace (class in *andes.core.service*), 290

Report (class in *andes.variables.report*), 325

report() (*andes.routines.base.BaseRoutine* method), 314

report() (*andes.routines.eig.EIG* method), 315

report() (*andes.routines.pflow.PFlow* method), 316

report\_info() (in module *andes.variables.report*), 325

reset() (*andes.core.var.BaseVar* method), 298

reset() (*andes.routines.tds.TDS* method), 317

reset() (*andes.system.System* method), 340

reset() (*andes.variables.dae.DAE* method), 323

resize\_arrays() (*andes.variables.dae.DAE* method), 323

restore() (*andes.core.param.ExtParam* method), 274

restore() (*andes.core.param.NumParam*

method), 276

restore\_sparse() (*andes.variables.dae.DAE* method), 323

rewind() (*andes.routines.tds.TDS* method), 317

run() (*andes.routines.base.BaseRoutine* method), 314

run() (*andes.routines.eig.EIG* method), 315

run() (*andes.routines.pflow.PFlow* method), 316

run() (*andes.routines.tds.TDS* method), 317

run() (in module *andes.main*), 329

run\_case() (in module *andes.main*), 329

## S

s\_numeric() (*andes.core.model.Model* method), 266

s\_numeric\_var() (*andes.core.model.Model* method), 266

s\_update() (*andes.core.model.Model* method), 266

s\_update\_post() (*andes.core.model.Model* method), 266

s\_update\_post() (*andes.system.System* method), 340

s\_update\_var() (*andes.core.model.Model* method), 266

s\_update\_var() (*andes.system.System* method), 341

Sampling (class in *andes.core.discrete*), 258

save\_conf() (in module *andes.main*), 330

save\_config() (*andes.system.System* method), 341

save\_output() (*andes.routines.tds.TDS* method), 317

scale\_func() (in module *andes.plot*), 334

SciPySolver (class in *andes.core.solver*), 292

Selector (class in *andes.core.discrete*), 258

selftest() (in module *andes.main*), 330

set() (*andes.core.model.Model* method), 266

set() (*andes.models.group.GroupBase* method), 309

set() (*andes.variables.fileman.FileMan* method), 325

set\_address() (*andes.core.var.BaseVar* method), 298

set\_address() (*andes.core.var.ExtVar* method), 301

set\_address() (*andes.system.System* method), 341



`set_arrays()` (*andes.core.var.BaseVar method*), 298  
`set_arrays()` (*andes.core.var.ExtVar method*), 301  
`set_config()` (*andes.system.System method*), 341  
`set_dae_names()` (*andes.system.System method*), 341  
`set_in_use()` (*andes.core.model.Model method*), 267  
`set_latex()` (*in module andes.plot*), 335  
`set_logger_level()` (*in module andes.main*), 330  
`set_pu_coeff()` (*andes.core.param.NumParam method*), 276  
`set_t()` (*andes.variables.dae.DAE method*), 323  
`set_title()` (*andes.utils.tab.Tab method*), 321  
`setup()` (*andes.system.System method*), 341  
`Shunt` (*class in andes.models.shunt*), 312  
`ShuntData` (*class in andes.models.shunt*), 312  
`Slack` (*class in andes.models.pv*), 312  
`SlackData` (*class in andes.models.pv*), 312  
`solve()` (*andes.core.solver.CuPySolver method*), 291  
`solve()` (*andes.core.solver.SciPySolver method*), 292  
`solve()` (*andes.core.solver.Solver method*), 293  
`solve()` (*andes.core.solver.SpSolve method*), 293  
`solve()` (*andes.core.solver.SuiteSparseSolver method*), 294  
`Solver` (*class in andes.core.solver*), 292  
`sort_psse_models()` (*in module andes.io.psse*), 302  
`SortedLimiter` (*class in andes.core.discrete*), 259  
`SpSolve` (*class in andes.core.solver*), 293  
`State` (*class in andes.core.var*), 301  
`StaticACDC` (*class in andes.models.group*), 310  
`StaticGen` (*class in andes.models.group*), 310  
`StaticLoad` (*class in andes.models.group*), 310  
`StaticShunt` (*class in andes.models.group*), 310  
`store_adder_setter()` (*andes.system.System method*), 341  
`store_existing()` (*andes.system.System method*), 341  
`store_sparse_ijv()` (*andes.variables.dae.DAE method*), 323  
`store_sparse_pattern()` (*andes.core.model.Model method*), 267  
`store_sparse_pattern()` (*andes.system.System method*), 341  
`store_switch_times()` (*andes.system.System method*), 342  
`store_txyz()` (*andes.variables.dae.DAETimeSeries method*), 324  
`streaming_init()` (*andes.routines.tds.TDS method*), 317  
`streaming_step()` (*andes.routines.tds.TDS method*), 317  
`SuiteSparseSolver` (*class in andes.core.solver*), 293  
`summary()` (*andes.routines.base.BaseRoutine method*), 314  
`summary()` (*andes.routines.eig.EIG method*), 315  
`summary()` (*andes.routines.pflow.PFlow method*), 316  
`summary()` (*andes.routines.tds.TDS method*), 317  
`supported_models()` (*andes.system.System method*), 342  
`switch_action()` (*andes.core.model.Model method*), 267  
`switch_action()` (*andes.system.System method*), 342  
`Switcher` (*class in andes.core.discrete*), 259  
`SymProcessor` (*class in andes.core.model*), 270  
`SynGen` (*class in andes.models.group*), 310  
`System` (*class in andes.system*), 335

## T

`t_const` (*andes.core.var.ExtState attribute*), 299  
`Tab` (*class in andes.utils.tab*), 320  
`TDS` (*class in andes.routines.tds*), 316  
`TDSData` (*class in andes.plot*), 330  
`tdsplot()` (*in module andes.plot*), 335  
`test_init()` (*andes.routines.tds.TDS method*), 318  
`testlines()` (*in module andes.io.matpower*), 302  
`testlines()` (*in module andes.io.psse*), 302  
`testlines()` (*in module andes.io.xlsx*), 303  
`tests_root()` (*in module andes.utils.paths*), 319  
`tex_names` (*andes.core.common.Config attribute*), 295  
`TG2` (*class in andes.models.governor*), 305

- TG2Data (class in *andes.models.governor*), 305  
 TGBase (class in *andes.models.governor*), 305  
 TGBaseData (class in *andes.models.governor*), 306  
 TGOV1 (class in *andes.models.governor*), 306  
 TGOV1Data (class in *andes.models.governor*), 306  
 TGOV1DB (class in *andes.models.governor*), 306  
 TGOV1DBData (class in *andes.models.governor*), 306  
 TGOV1DBModel (class in *andes.models.governor*), 306  
 TGOV1Model (class in *andes.models.governor*), 306  
 TGOV1ModelAlt (class in *andes.models.governor*), 306  
 TimedEvent (class in *andes.models.group*), 310  
 TimerParam (class in *andes.core.param*), 277  
 to\_array() (*andes.core.param.ExtParam* method), 274  
 to\_array() (*andes.core.param.NumParam* method), 276  
 to\_csc() (*andes.core.solver.SciPySolver* method), 292  
 to\_number() (in module *andes.utils.misc*), 320  
 Toggler (class in *andes.models.timer*), 313  
 TogglerData (class in *andes.models.timer*), 313  
 TurbineGov (class in *andes.models.group*), 311  
 txyz (*andes.variables.dae.DAETimeSeries* attribute), 324
- ## U
- UMFPACKSolver (class in *andes.core.solver*), 294  
 Undefined (class in *andes.models.group*), 311  
 undill() (*andes.system.System* method), 342  
 unpack() (*andes.variables.dae.DAETimeSeries* method), 324  
 update() (*andes.core.common.ModelFlags* method), 296  
 update() (*andes.variables.report.Report* method), 325
- ## V
- v (*andes.core.service.ApplyFunc* attribute), 278  
 v (*andes.core.service.DataSelect* attribute), 281  
 v (*andes.core.service.DeviceFinder* attribute), 282  
 v (*andes.core.service.FlagCondition* attribute), 284  
 v (*andes.core.service.FlagValue* attribute), 284  
 v (*andes.core.service.IdxRepeat* attribute), 285  
 v (*andes.core.service.NumReduce* attribute), 286  
 v (*andes.core.service.NumRepeat* attribute), 288  
 v (*andes.core.service.NumSelect* attribute), 288  
 v (*andes.core.service.OperationService* attribute), 288  
 v (*andes.core.service.ParamCalc* attribute), 289  
 v (*andes.core.service.RandomService* attribute), 289  
 v (*andes.core.service.RefFlatten* attribute), 290  
 v (*andes.core.service.Replace* attribute), 290  
 v\_code (*andes.core.var.Algeb* attribute), 297  
 v\_code (*andes.core.var.ExtAlgeb* attribute), 299  
 v\_code (*andes.core.var.ExtState* attribute), 299  
 v\_code (*andes.core.var.State* attribute), 301  
 v\_numeric() (*andes.core.model.Model* method), 268  
 v\_numeric() (*andes.models.synchronous.GENBase* method), 313  
 v\_numeric() (*andes.models.timer.Toggler* method), 313  
 VarHold (class in *andes.core.service*), 290  
 vars\_to\_dae() (*andes.system.System* method), 342  
 vars\_to\_models() (*andes.system.System* method), 342  
 VarService (class in *andes.core.service*), 290
- ## W
- warn\_init\_limit() (*andes.core.discrete.Discrete* method), 256  
 Washout (class in *andes.core.block*), 250  
 WashoutOrLag (class in *andes.core.block*), 250  
 write() (*andes.variables.report.Report* method), 325  
 write() (in module *andes.io.xlsx*), 303  
 write\_lst() (*andes.variables.dae.DAE* method), 323  
 write\_npy() (*andes.variables.dae.DAE* method), 324  
 write\_npz() (*andes.variables.dae.DAE* method), 324
- ## X
- x (*andes.variables.dae.DAETimeSeries* attribute), 324  
 xy (*andes.variables.dae.DAE* attribute), 324  
 xy\_name (*andes.variables.dae.DAE* attribute), 324

`xy_tex_name` (*andes.variables.dae.DAE attribute*), [324](#)

`xyz` (*andes.variables.dae.DAE attribute*), [324](#)

`xyz_name` (*andes.variables.dae.DAE attribute*), [324](#)

`xyz_tex_name` (*andes.variables.dae.DAE attribute*), [324](#)

## Y

`y` (*andes.variables.dae.DAETimeSeries attribute*), [324](#)

## Z

`zip_ijv()` (*andes.core.common.JacTriplet method*), [296](#)

`zip_ijv()` (*andes.core.model.ModelCall method*), [268](#)