STEPSS

Static and Transient Electric Power Systems Simulation

Documentation of models and user guide

May 2, 2025

ACADEMIC PUBLIC LICENSE FOR THE USE OF STEPSS

1. Definitions related to the software

"Software" means a copy of STEPSS, which is distributed under this Academic Public License. "Work based on the Software" means either the Software or any derivative work under copyright law: that is to say, a work containing the Software or a portion of it, either verbatim or with modifications. "Using the Software" means any act of creating executables that contain or directly use libraries that are part of the Software, running any of the tools that are part of the Software, or creating works based on the Software. "You" refers to each licensee.

2. Authors

STEPSS has been developed by Dr Petros Aristidou (email: petros.aristidou@cut.ac.cy) and Dr Thierry Van Cutsem (email: thierry.h.van.cutsem@gmail.com), hereafter referred to as the "Authors".

3. Intellectual property rights

STEPSS is made up of three modules: PFC (for power flow computations), RAMSES (the solver of differential-algebraic equations), and CODEGEN (a tool to develop models).

PFC and CODEGEN are the property of the Authors. RAMSES is the property of the University of Liège, Belgium, which has granted to both Authors a personal, royalty-free, limited, non-exclusive, non-transferable and non-assignable license to distribute free of charge an executable version of RAMSES in accordance with the terms detailed under Articles 4, 5 and 6 below.

4. License terms

Permission is hereby granted to use the Software free of charge for any non-commercial purpose, including teaching and research at universities, colleges and other educational institutions, research at non-profit research institutions, and personal non-profit purposes. For using the Software for commercial purposes, including but not restricted to consulting activities, design of commercial hardware or software products, or if you are a commercial entity participating in research projects, you have to contact the Authors.

You are not required to accept this License. Nothing else grants you permission to use the Software; the law prohibits this action if you do not accept this License. Therefore, by using the Software (or any work based on the Software), you indicate your acceptance of this License and all its terms and conditions.

5. Warranty

The Software is provided “as is”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In no event shall the Authors nor the Intellectual property owners be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the Software.

6. Limited version of the Software

This free-of-charge version of the software is limited to power system models up to 1000 buses (or nodes) and can be executed with parallelization using no more than two cores. For extensions to larger models or execution using more than two cores, you have to contact the Authors.

Contents

Part I

Information pertaining to all three modules

Chapter 1

A quick overview of STEPSS

1.1 The three modules of STEPSS

STEPSS includes three modules:

- **PFC** (for Power Flow Computation) performs a power flow computation in order to determine the initial operating point of a dynamic simulation;
- **RAMSES** (for RApid Multiprocessor Simulation of Electric power Systems) simulates the dynamic evolution of the power system in response to disturbances/actions specified by the user;
- **CODEGEN** (for CODE GENerator) translates a model described by the user in a text file into its equivalent in FORTRAN 2003 language. The latter has to be compiled and linked to a user-defined executable version RAMSES.

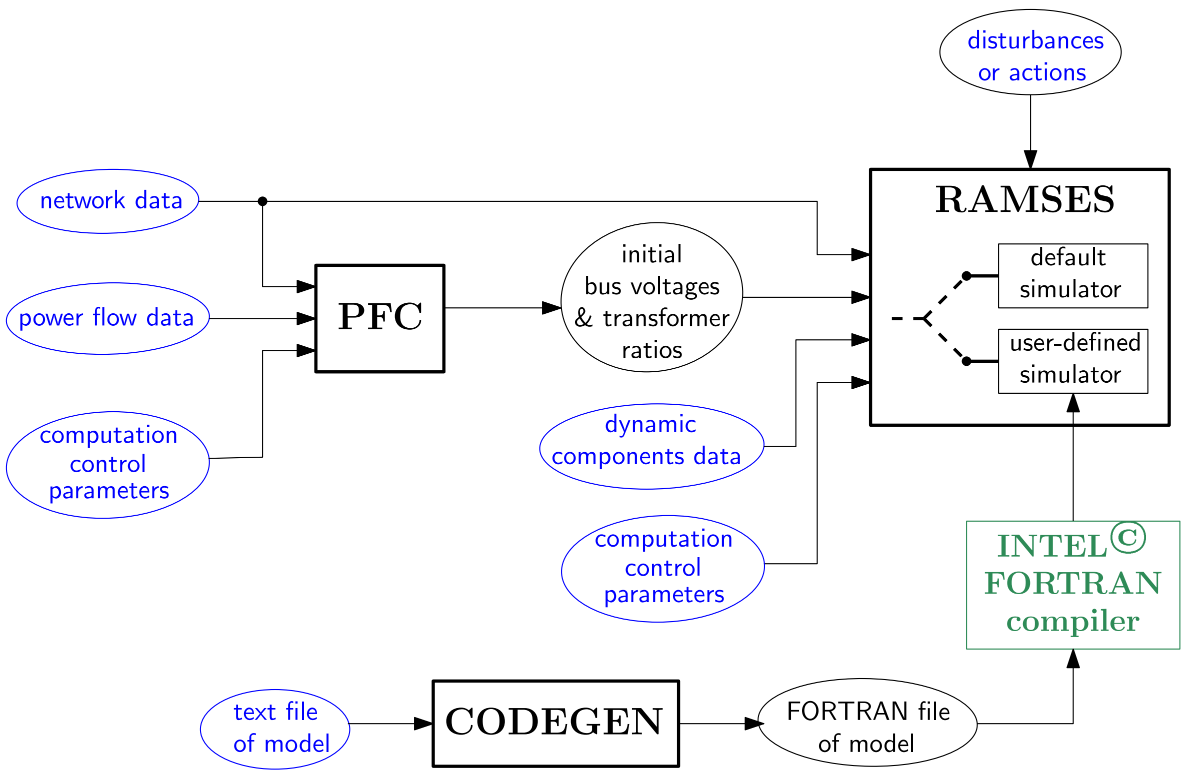


Figure 1.1: STEPSS: Overview of modules and data files. The files shown in blue are provided by the user, those in black are produced internally. The Intel© FORTRAN compiler is not part of STEPSS

The three modules, their relationships and their input/ouput data files are shown graphically in Fig. 1.1.

Note that each of the three modules can be used separately. Here are examples of such uses:

- PFC alone: a power flow computation is run to inspect the power system state and/or save the corresponding power flow solution for use by RAMSES;

- PFC alone: a sequence of power flow computations is performed until obtaining the desired power system state. The corresponding power flow solution is saved for use by RAMSES;
- RAMSES alone: with the initial power flow solution produced by PFC, several dynamic simulations are run starting from that initial state, i.e. without invoking PFC each time;
- CODEGEN alone: a model is built and saved for future incorporation in a user-defined version of RAMSES.

1.2 PFC module

The power flow computation resorts to the well-known Newton(-Raphson) method. Polar coordinates are used.

As shown in Fig. 1.1, the input information consists of:

- network data relative to buses, lines, transformers, etc.
- power flow data specified at PV, PQ and slack buses, respectively
- PFC control parameters. These are settings such as: tolerances on final power mismatches, thresholds to enforce reactive power limits of generators, etc. These are optional data; if they are not provided, default values (listed in this documentation) are used.

Optionally PFC also adjusts the ratios of some transformers, with the objective:

- for an in-phase transformer: to bring the voltage magnitude at a bus inside a specified deadband
- for a phase-shifting transformer: to bring the active power flow in a network branch inside a specified deadband.

PFC produces an output file including:

- the voltage magnitudes and phase angles at all buses of the network
- the adjustable transformer data with updated values of their ratios.

1.3 RAMSES module

RAMSES is aimed at simulating the dynamic response of power system models derived under the phasor approximation (also known as RMS approximation).

It takes as input (see Fig. 1.1):

- the network data, which are shared with PFC[1]
- the data pertaining to the dynamics of the components connected to the network
- RAMSES control parameters. These are settings such as: tolerances for solving the equations, variation of state variables to identify the Jacobians, time steps of plots, reference angular speed, etc.
- the sequence of actions and/or disturbances imposed during the simulation.

The mathematical model involves both differential and algebraic equations. Three algebraization methods are available to integrate the differential equations:

- Backward Euler:
$$x_{k+1} = x_k + h\,\dot{x}_{k+1}$$
- Trapezoidal method:
$$x_{k+1} = x_k + \frac{h}{2}\,(\dot{x}_{k+1} + \dot{x}_k)$$
- Second-order Backward Differentiation Formula (BDF2):
$$x_{k+1} = \frac{4}{3}x_k - \frac{1}{3}x_{k-1} + \frac{2\,h}{3}\dot{x}_{k+1}$$

where h is the time step size. The three methods are implicit, which contributes to numerical robustness. BDF2 is a stiff-decay (or L_1-stable) integration scheme allowing to somewhat increase the time step size when fast transients are not of interest.

The resulting algebraized and the original algebraic equations are solved all together using a Newton method. Some more information follows.

1.3.1 About the solver

The solver was developed in response to the growing demand for simulations that last longer (e.g. long-term stability studies) or involve larger models (e.g. to account for the impact of active distribution networks). A high computational efficiency is made possible by two acceleration techniques: parallel processing and localization.

Parallel processing is based, first, on the decomposition of the power system model into respectively the network, the "injectors" and the "two-ports". Injectors and two-ports are solved independently of each other. However, by resorting to the Schur-complement for the network equations, RAMSES yields the exact same solution as a non-decomposed scheme[2].

Next, the tasks pertaining to injectors and two-ports are assigned to a number of *threads*. Examples of tasks are:

[1] with a few exceptions (detailed in this document)

[2] The solution scheme is thus of the simultaneous type while offering some advantages of a partitioned scheme

- the update and factorization of injector and two-port Jacobians
- the computation of the mismatch vector of Newton method
- the computation of injector contributions to the Schur-complement matrix
- the solution of local linear systems.

The threads can be executed each on a separate processor, the computational load being balanced among the available processors. The freeware version of STEPSS allows exploiting two processors.

The solver in RAMSES has been developed using a shared-memory parallel programming model with the help of the OpenMP Application Programming Interface. The implementation is general: there is no "hand-crafted" optimization particular to the computer system, the power system or the disturbance.

Localization is based on the fact that, after a disturbance, the various components of a (large enough) system exhibit different levels of dynamic activity.

This property is exploited at each time step:

- to accelerate the Newton scheme: thanks to the decomposed solution scheme, Newton iterations are skipped on injectors and two-ports that have already converged
- to exploit component "latency": injectors with high dynamic activity are classified as active, the others as latent. Active injectors have their original model simulated, while latent injectors are replaced by automatically calculated, sensitivity-based models to accelerate the simulation. A fast to compute metrics is used to classify the injectors, which seamlessly switch between categories according to their activity.

More information on the solver can be found in the following references:

- D. Fabozzi, A. Chieh, B. Haut, and T. Van Cutsem, "Accelerated and localized newton schemes for faster dynamic simulation of large power systems," IEEE Trans. on Power Systems, Vol. 28, No 4, pp. 4936-4947, Dec. 2013
  doi: 10.1109/TPWRS.2013.2251915
- P. Aristidou, D. Fabozzi, and T. Van Cutsem, "Dynamic simulation of large-scale power systems using a parallel Schur-complement-based decomposition method," IEEE Trans. on Parallel and Distributed Systems, Vol. 25, No 10, pp. 2561-2570, Sept. 2014,
  doi:10.1109/TPDS.2013.252

1.4 CODEGEN module

CODEGEN allows incorporating user-defined models in RAMSES. Different versions of the RAMSES executable can be loaded and executed. This involves a translation of the user model specified in a text file into FORTRAN 2003 code to be compiled and linked to the rest of the executable. The FORTRAN 2003 language can produce very effcient number crunching code.

Don't panic: in the vast majority of cases, the user does not need to even open the FORTRAN file. The user concentrates on the text file describing his/her model, which is rather easy to read.

Thus, the user model is not interpreted but executed.

There are four types of user-defined models:

- excitation controller of synchronous machine: typically the excitation system and the automatic voltage regulator
- torque controller of synchronous machine: typically the turbine and the speed governor
- injector: a component connected to a single AC bus
- two-port: a component connecting two buses.

While the code of the solver is not made public, the models are expected to be freely shared by users. This feature makes STEPSS an **open-source simulation software**, at least for the modelling part. That is what matters to the user, isn't it ?

Chapter 2

Installing STEPSS

The material of this chapter is specific to the Windows 64-bit version of STEPSS, to its Java-based Graphic User Interface (GUI) and the Intel oneAPI Fortran Compiler.

Before going any further, you have to read and accept the legal terms detailed at the beginning of this document.

All needed files can be downloaded from the following Google drive:
`https://drive.google.com/drive/folders/1HcUSD-FOx6192HURJnxltYJuKuNA8Vwz?usp=drive_link`

Note that the Java, Visual Studio and OneAPI files are not part of STEPSS. They are provided only to ease the installation procedure, avoiding navigation through the Oracle, Windows and Intel Web pages.

2.1 Installing JAVA

STEPSS comes with a Java-based Graphical User Interface (GUI). Everything is packed in a Java archive containing a (legal) copy of all executables and libraries required to run the simulations and display the results.

Using that GUI requires to have Java Version 20 installed on your computer. To check if you have Java installed, and if so, which version it is:

1. open a Windows Command prompt
2. in the latter, enter the command:
   `java -version`
   It should display a message similar to the one shown in Fig. 2.1.
3. close the command prompt.

```
C:\Users\tvanc\onedrive\Travail>java -version
java version "20.0.1" 2023-04-18
Java(TM) SE Runtime Environment (build 20.0.1+9-29)
Java HotSpot(TM) 64-Bit Server VM (build 20.0.1+9-29, mixed mode, sharing)
```

Figure 2.1: Checking the presence of the Java environment

If you have a previous version installed on your computer, it must be uninstalled following the standard Windows procedure:

1. click on Parameters
2. select Applications
3. locate Java
4. click on uninstall.

Then, install Java Version 20 as follows:

1. download the Java SE Development Kit Version 20 (for 64-bit) from the Google drive. The 164-MB file is named `jdk-20_windows-x64_bin.exe`
2. install by double-clicking on that file and following the instructions
3. for an easy execution, associate stepss.jar with Java following the standard Windows procedure:
   (a) right-click on the stepss.jar icon
   (b) select open with
   (c) choose always use this application.

Apart from the Java machine, STEPSS does not require any installation in the usual meaning of the term. Just download the `STEPSS.jar` file from the Google drive and drop it in any convenient place of the computer. The Windows desktop is a very convenient location.

Leave the archive intact; in particular do not uncompress it !

STEPSS is launched by merely double-clicking on `STEPSS.jar` (or a shortcut to the latter).

When STEPSS is launched it creates a temporary working folder and copies all needed executables and libraries into that folder.

To remove STEPSS from your computer, just delete the archive file. STEPSS does not leave anything in your computer !

2.2 Installing Visual Studio

This step can be skipped if you do not plan to develop new models for RAMSES.

The Intel OneAPI compiler relies on the Microsoft Visual Studio environnement as well as some associated C++ libraries.

The Community 2022 version is required. If you have a previous version installed on your computer, it must be uninstalled following the standard Windows procedure.

The installation steps are as follows:

1. download from the Google drive the launcher named
   `vs_community__e942d3d4864d4a80b72975352289d007.exe`
2. execute by double-clicking
3. during the installation, under the “Workloads” view, select the checkbox to install the “Desktop development with C++” component of Visual Studio as shown in Fig. 2.2. This component

is not installed by default.

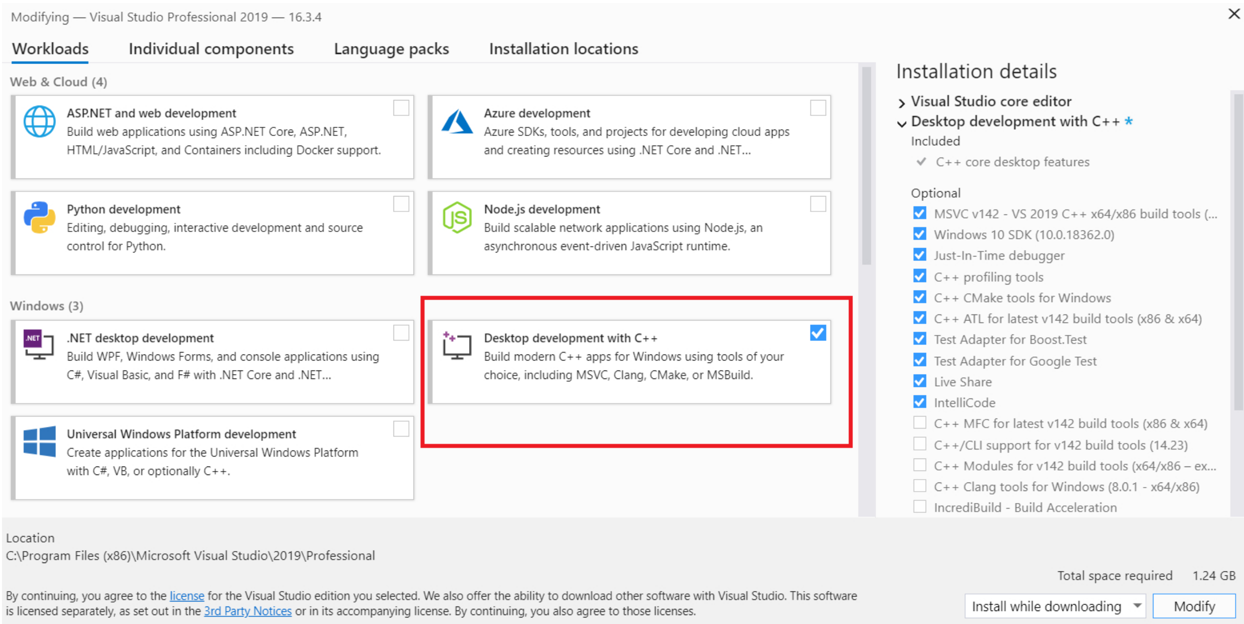


Figure 2.2: Selecting the "Desktop development with C++" component when installing Visual Studio

2.3 Installing the Fortran Compiler

This step can be skipped if you do not plan to develop new models for RAMSES.

There are three packages to download and install.

1. download from the Google drive the OneApi Basekit. The file is named: `w_BaseKit_p_2023.1.0.47256_offline.exe`
2. download from the Google drive the Fortran Compiler Classic package. The file is named: `w_ifort_runtime_p_2023.1.0.46319.exe`
3. download from the Google drive the HPC toolkit. The file is named: `w_HPCKit_p_2023.1.0.46357_offline.exe`
4. install each of the three packages, in the above order, by double-clicking on the .exe files
5. restart your computer.

Chapter 3

Organisation of the data files

Data files are organised into *records* and *comments*, whose syntax is detailed next.

3.1 Records

Each record includes:

- a leading *keyword*, which identifies the information provided in the record
- a number of *fields*. A field is either a real number (numeric field) or a string of characters (character field). There is at least one field
- a *terminating semicolumn* (;), which indicates the end of the record.

☛ The following record, with keyword LINE, specifies a transmission line:
```
LINE A-B BUS_A BUS_B 3.0 30.0 150.0 1400.0 1 ;
```

Inside the record, the keyword, the fields and the terminating semicolon are separated by (at least one) space(s). Anything after the semicolumn is ignored. The next record (or comment) starts with the next line.

☛ Two incorrect versions of the above sample record, owing to missing spaces:
```
LINE A-B BUS_ABUS_B 3.0 30.0 150.0 1400.0 1 ;
LINE A-B BUS_A BUS_B 3.030.0 150.0 1400.0 1 ;
```

Inside a data file, a record may span over multiple lines; the semicolumn indicates the end of the record. Spanning over several lines is highly recommended for records that include many fields. Note that, depending upon the text editor and its settings, a long record could appear truncated when displayed.

☛ An example of long record spanning over two lines:
```
INJEC GFOL VSC1 A 1.0 1.0 0.0 0.0 0.005 0.15 1.00 1044.0 0.005 0.15 33.3
10.0 0.002 -999.0 10.0 0.1667 50.0 0.10 0.4 0.5 1.0 0.95 0.5 99.0 1 ;
```

The nature of each field, and the total number of fields are specified in this documentation. Some records have optional fields, which are always located at the end of the record.

3.1.1 Constraints on numeric fields

There is almost no constraint on numeric fields. They are written in free format. The notation is that of floating-point numbers in MATLAB: with or without a dot, with or without an exponent, exponent denoted by `E` or `D`.

☛ Different valid formats of the same number: `30 30. 30.0 3E01 3.E01 3.0E01 3.E1 3.e1`

3.1.2 Constraints on character fields

Character fields are limited to 20 characters. Only the first 20 characters are read; the remaining of the string is just ignored, without warning. It is thus discouraged to have more than 20 characters in a field.

Furthermore, some character fields are limited to eight characters; the remaining of the string is just ignored.

Uppercase letters are significant within a character field. Thus, two fields that differ by the uppercase/lowercase spelling are different.

If a character field includes a space or a slash (/), it must be enclosed with quotes (' or "). In between the two quotes, the leading spaces are significant while the trailing ones are ignored. Keywords do not include spaces; hence, the use of quotes is useless.

Because the semi-column (;) is a special character (used to indicate the end of a record), **it must not be included in any character field**, even within quotes.

3.2 Comments

There are three ways to insert comments in the data files:

1. a line in which the first non blank character is an exclamation mark (!). At most the first 130 characters after the ! are memorised and reproduced on output files
2. a line in which the first non blank character is the sharp character (#): this line is simply ignored by the program
3. anything written after the semicolon that terminates a record is also ignored. This is convenient to store (short) comments next to a record, without interfering with the latter.

☛ Comments starting with # can be used to identify the fields of a record:

```
#              name bus FP   FQ   P   Q SNOM RS    LLS LSR   RR    LLR
INJEC INDMACH1 SM    2  0.2 0.2 0. 0. 0. 0.031 0.1 3.2 0.018 0.180
#                                    H    A    B    LF
                                    0.7 0.5 0.0 0.6 ;
```

Comments do not span over several lines. If several lines are needed, each of them must start with a ! or a #.

Empty lines are just ignored.

3.3 Sharing data between files

Records may be distributed over an arbitrary number of data files, which will be read sequentially. The order in which the records are placed inside the data files does not matter. The order in which the files are read does not matter either.

For instance, a first data file may be devoted to network data, a second one to data for the initial power flow computation, a third one to the dynamic data of the components connected to the network and a last one to simulation control parameters. The second and third files, for instance, may be swapped in the list without any effect.

Chapter 4

Modelling of network components

The network model includes buses, lines, cables, transformers and shunts. Their models and data are described in the present chapter.

4.1 Buses

In dynamic simulations with RAMSES the only parameter associated with a bus is its nominal voltage (more precisely, the RMS value of the nominal line-to-line voltage). This is used as base voltage to convert parameters from physical to per unit values.

If two buses have different nominal voltages they cannot be connected through a path made up of lines or switches (see next sections). This causes the software to issue an error message and stop.

4.1.1 Data format

The record, with keyword BUS, includes the following fields:

```
BUS NAME VNOM ;
```

where:

- `NAME` is the name of the bus. This is a string of at most 8 characters
- `VNOM` is the nominal voltage, in kV.

Only one BUS record per bus is allowed.

All buses must be declared through BUS records. If the software finds (in other than BUS records) a bus name not defined through a BUS record, it issues an error message and stops.

Caveat. The above record is used in the RAMSES module. For the initial power flow computation with PFC, an extended version of the BUS record is used with six fields instead of two. Please refer to Section 5.1. Thus, when RAMSES is initialized, if a BUS record has six fields, only the first two are read, the others are ignored.

4.2 Lines and cables

4.2.1 Modelling

Lines and cables both have the same pi-equivalent model, which is shown in Fig. 4.1. Note that shunt conductances are neglected.

Under the phasor approximation, series capacitors can also be modelled with this pi-equivalent, by setting R and C to zero and X to a negative value.

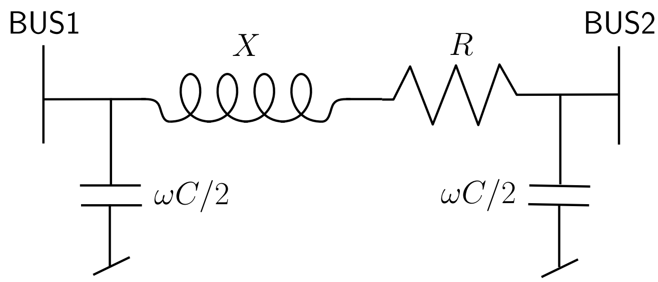


Figure 4.1: Pi-equivalent of lines and cables

4.2.2 Data format

The record, with keyword LINE, includes the following fields:

```
LINE NAME BUS1 BUS2 R X WC2 SNOM BR ;
```

where:

- `NAME` is the name of the line or cable. This is a string of at most 20 characters
- `BUS1` is the name of the first bus. This is a string of at most 8 characters defined in a BUS record
- `BUS2` is the name of the second bus. This is a string of at most 8 characters defined in a BUS record
- `R` is the series resistance R, in Ω
- `X` is the series reactance X, in Ω
- `WC2` is the half shunt susceptance $\omega C/2$, in μS (microSiemens)
- `SNOM` is the nominal apparent power, in MVA. This value is used to display the line loading, or possibly in user-defined models. If not used, it may be set to zero; this will be interpreted as an infinite power
- `BR` is the on/off status of the line breakers. A zero value indicates that the breakers are open at both ends (line out of service); any other value (e.g. 1) means that both breakers are closed (line in service).

The orientation of the line is arbitrary: BUS1 and BUS2 may be swapped.

Only one LINE record per line is allowed.

All lines are memorized, even those that are out of service. A line out of service is not involved (it appears in the output results with a zero power flow) but it can be put into service in the dynamic simulation.

As mentioned in Section 4.1, a line must not connect two buses with different nominal voltages.

To have a line connected through a single end, add a bus at the open end and set BR to a nonzero value.

4.3 Switches

4.3.1 Modelling

A switch is a connection without impedance between two buses. It is treated as a very short line, more precisely a line with $R = 0$, $\omega C/2 = 0$ and X set to a very low value. Thus it has no active power losses and negligible reactive power losses.

4.3.2 Data format

The record, with keyword SWITCH, includes the following fields:

```
SWITCH NAME BUS1 BUS2 BR ;
```

where:

- `NAME` is the name of the switch. This is a string of at most 20 characters
- `BUS1` is the name of the first bus. This is a string of at most 8 characters defined in a BUS record
- `BUS2` is the name of the second bus. This is a string of at most 8 characters defined in a BUS record
- `BR` is the on/off status of the switch. A zero value indicates that the switch is open; any other value means that it is closed.

The orientation of the switch is arbitrary: BUS1 and BUS2 may be swapped.

Only one SWITCH record per switch is allowed.

All switches are memorized, even those which are open. An open switch is not involved (it appears in the output results with a zero power flow) but it can put into service in the dynamic simulation.

As mentioned in Section 4.1, it is not allowed to have a switch connecting two buses with different nominal voltages.

4.4 Transformers

4.4.1 Modelling

Transformers are represented by the two-port shown in Fig. 4.2. Note that R, X, B_1 and B_2 are specified on the “from” side of the transformer.

R corresponds to the copper losses. The iron losses are neglected (no shunt resistance). X is the leakage reactance. B_1 or B_2 are the magnetizing susceptances, which have negative values. Usually one of them is zero. n (resp. ϕ) is the magnitude (resp. the phase angle) of the transformer ratio. A phase-shifting transformer is characterized by a nonzero value of ϕ.

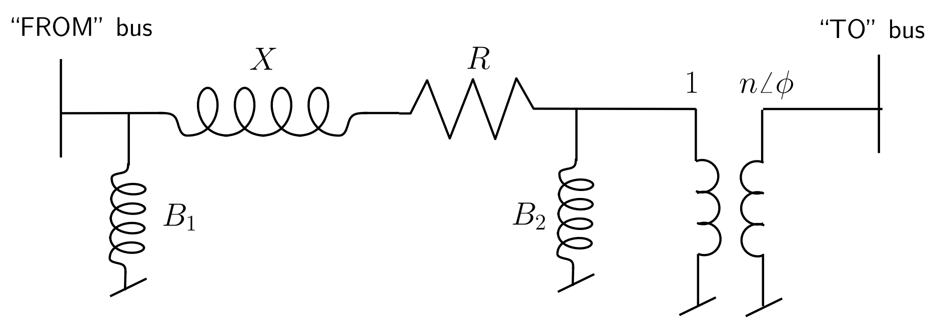


Figure 4.2: Two-port model of transformers

It may be of interest to recall how the values of R, X, B_1 and B_2 relate to the following characteristics, easily obtained from manufacturer data:

- S_{nom} the nominal apparent power
- V_{N1} (resp. V_{N2}) the nominal voltage on the “from” (resp. “to”) side (see Fig. 4.2)
- $R_{baseV_{N1}}$ (resp. $X_{baseV_{N1}}$) the series resistance (resp. leakage reactance) in per unit on the (S_{nom}, V_{N1}) base
- $B_{1\,baseV_{N1}}$ and $B_{2\,baseV_{N1}}$ the shunt susceptances in per unit on the (S_{nom}, V_{N1}) base
- V_{o1} and V_{o2} the open-circuit voltages corresponding to the transformer ratio[1].

[1]Very often the open-circuit voltages coincide with the nominal voltages, i.e. $V_{o1} = V_{N1}$ and $V_{o2} = V_{N2}$

Let V_{B1} and V_{B2} be the nominal voltages of respectively the "from" and the "to" buses, as specified in their BUS records (see Section 4.1).

In STEPSS R, X, B_1 and B_2 are in percent on the (S_{nom}, V_{B1}) base. The following changes of base have to be used:

$$R = 100\ R_{baseV_{N1}} (\frac{V_{N1}}{V_{B1}})^2 \qquad X = 100\ X_{baseV_{N1}} (\frac{V_{N1}}{V_{B1}})^2$$

$$B_1 = 100\ B_{1\,baseV_{N1}} (\frac{V_{B1}}{V_{N1}})^2 \qquad B_2 = 100\ B_{2\,baseV_{N1}} (\frac{V_{B1}}{V_{N1}})^2$$

n is in percent on the (V_{B1}, V_{B2}) base. Its value is given by:

$$n = 100\ \frac{V_{o2}\ V_{B1}}{V_{o1}\ V_{B2}}$$

A second transformer model is available. It is a simplified version with $B_2 = 0$ and $\phi = 0$ in Fig. 4.2 and and includes data for PFC to adjust the transformer ratio (see Section 5.5). This model cannot be used for phase-shifting transformers.

4.4.2 Data format

The record, with keyword TRANSFO, includes the following fields:

```
TRANSFO NAME FROMBUS TOBUS R X B1 B2 N PHI SNOM BR ;
```

where:

- `NAME` is the name of the transformer. This is a string of at most 20 characters
- `FROMBUS` is the name of the "from" bus (see Fig. 4.2). This is a string of at most 8 characters defined in a BUS record
- `TOBUS` is the name of the "to" bus (see Fig. 4.2). This is a string of at most 8 characters defined in a BUS record
- `R` is the series resistance, in % on the base detailed above
- `X` is the leakage reactance, in % on the base detailed above
- `B1` is the shunt suceptance, in % on the base detailed above
- `B2` is the shunt suceptance, in % on the base detailed above
- `N` is the magnitude of the transformer ratio in % (dimensionless)
- `PHI` is the phase angle of the transformer ratio, in degree
- `SNOM` is the nominal apparent power of the transformer, in MVA. This value must not be zero
- `BR` is the on/off status of the transformer breakers. A zero value indicates that the breakers are open at both ends (transformer out of service); any other value means that both breakers are closed (transformer in service).

The second transformer model is described by the record with keyword TRFO:

```
TRFO NAME FROMBUS TOBUS CONBUS R X B N SNOM NFIRST NLAST NBPOS TOLV VDES BR ;
```

where:

- `NAME` is the name of the transformer. This is a string of at most 20 characters
- `FROMBUS` is the name of the "from" bus (see Fig. 4.2). This is a string of at most 8 characters defined in a BUS record
- `TOBUS` is the name of the "to" bus (see Fig. 4.2). This is a string of at most 8 characters defined in a BUS record
- `CONBUS` is used by PFC for adjusting n: see Section 5.5. It is not used by RAMSES, but a (dummy) name must be provided
- `R` is the series resistance, in % on the base detailed above
- `X` is the leakage reactance, in % on the base detailed above
- `B` is the shunt suceptance, in % on the base detailed above
- `N` is the magnitude of the transformer ratio in % (dimensionless)
- `SNOM` is the nominal apparent power of the transformer, in MVA. This value must not be zero
- `NFIRST` is used by PFC for adjusting n: see Section 5.5.
- `NLAST`: same as for `NFIRST`
- `NBPOS`: same as for `NFIRST`
- `TOLV`: same as for `NFIRST`
- `VDES`: same as for `NFIRST`
- `BR` is the on/off status of the transformer breakers. A zero value indicates that the breakers are open at both ends (transformer out of service); any other value means that both breakers are closed (transformer in service).

The following remarks apply to both TRANSFO and TRFO records.

The orientation of the transformer is not arbitrary: FROMBUS and TOBUS cannot be swapped.

Only one TRANSFO or TRFO record per transformer is allowed.

All transformers are memorized, even those that are out of service. A transformer out of service is not involved (it appears in the output results with a zero power flow) but it can be put into service in the dynamic simulation.

To have a transformer connected through a single end, add a bus at the open end and set BR to a nonzero value.

4.5 Non-reciprocal two-ports

4.5.1 Modelling

A non-reciprocal two-port is a two-port with a non-symmetric nodal admittance matrix of the form:

$$\boldsymbol{Y} = \begin{bmatrix} (G_{si} + jB_{si}) + (G_{ij} + jB_{ij}) & -(G_{ij} + jB_{ij}) \\ -(G_{ji} + jB_{ji}) & (G_{ji} + jB_{ji}) + (G_{sj} + jB_{sj}) \end{bmatrix}$$

with:

$$G_{ij} \neq G_{ji} \quad \text{and} \quad B_{ij} \neq B_{ji}$$

where i and j relate to the terminal nodes of the two-port, as shown in Fig. 4.3.

Typically, two-ports are produced when reducing a network that includes phase-shifting transformers, the purpose being to obtain an equivalent.

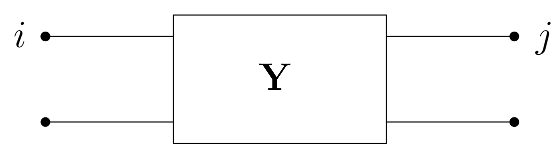


Figure 4.3: A two-port connecting buses i and j

4.5.2 Data format

The record, with keyword NRTP, includes the following fields:

```
NRTP NAME FROMBUS TOBUS GIJ BIJ GJI BJI GSI BSI GSJ BSJ BR ;
```

where:

- `NAME` is the name of the two-port. This is a string of at most 20 characters
- `FROMBUS` is the name of bus i in Fig. 4.3. This is a string of at most 8 characters defined in a BUS record
- `TOBUS` is the name of bus j in Fig. 4.3. This is a string of at most 8 characters defined in a BUS record
- `GIJ` is the G_{ij} conductance, in pu
- `BIJ` is the B_{ij} susceptance, in pu
- `GJI` is the G_{ji} conductance, in pu
- `BJI` is the B_{ji} susceptance, in pu
- `GSI` is the G_{si} conductance, in pu
- `BSI` is the B_{si} susceptance, in pu

- `GSJ` is the G_{sj} conductance, in pu
- `BSJ` is the B_{sj} susceptance, in pu
- `BR` is the on/off status of the two-port breakers. A zero value indicates that the breakers are open at both ends (two-port out of service); any other value means that both breakers are closed (two-port in service).

The parameters of the two-port are given in per unit on the following base: nominal bus voltages, system base power. By default a value of 100 MVA is used[2] but it can be changed: see Section 5.9.

It is allowed for a non-reciprocal two-port to connect two buses with different nominal voltages.

The orientation of the two-port is not arbitrary: FROMBUS and TOBUS cannot be swapped.

Only one NRTP record per two-port is allowed.

A non-reciprocal two-port is treated as a piece of equipment; hence, the presence of the `BR` field. All non-reciprocal two-ports are memorized, even those which are out of service. A non-reciprocal two-port out of service is not involved (it appears in the output results with a zero power flow) but it can be put into service in the dynamic simulation.

4.6 Shunts

4.6.1 Modelling

The shunt element is treated as a purely reactive, constant shunt admittance. Hence, the reactive power Q it produces varies with the square of the voltage V according to:

$$Q = B\,V^2$$

where B is the susceptance. The element may correspond to either a capacitor ($B > 0$) or a reactor ($B < 0$).

4.6.2 Data format

The record, with keyword SHUNT, includes the following fields:

```
SHUNT NAME BUS_NAME QNOM BR ;
```

where:

[2]a typical value for transmisstion systems

- NAME is the name of the shunt. This is a string of at most 20 characters
- BUS_NAME is the name of the bus to which the shunt is connected. This is a string of at most 8 characters defined in a BUS record
- QNOM is the nominal reactive power of the shunt, in Mvar. This is the reactive power produced by the shunt under the nominal voltage of the bus, specified in its BUS record. QNOM is positive (resp. negative) for a shunt capacitor (resp. inductor)
- BR is the on/off status of the shunt. A zero value indicates that the shunt is not connected; any other value means that it is in service.

Only one SHUNT record per named shunt is allowed. Multiple shunts at the same bus are allowed, but each with its own name. In this case, the susceptances are added (taking signs into account).

All shunts are memorized, even those which are disconnected. A disconnected shunt is not involved (it appears in the output results with a zero power flow) but it can be put into service in the dynamic simulation.

Caveat. The above record is used in the RAMSES module. However, for the initial power flow computation with PFC, the shunt data are attached to a bus and are specified in an extended version of the BUS record. Please refer to Section 5.1.

Part II

Power Flow Computation with PFC

Chapter 5

PFC data

The following records, documented in Chapter 4 are used by PFC:

BUS
LINE
SWITCH
TRANSFO
TRFO
NRTP.

The additional (mandatory or optional) records only used in power flow computations are documented in this chapter.

5.1 Load and shunt data

Load and shunt data are attached to buses. They are specified in an extended version of the BUS record, as follows:

```
BUS NAME VNOM PLOAD QLOAD BSHUNT QSHUNT ;
```

where:

- `NAME` is the name of the bus. This is a string of at most 8 characters
- `VNOM` is the nominal voltage, in kV
- `PLOAD` is the total active power load at the bus, in MW. A positive value corresponds to power drawn from the network
- `QLOAD` is the total reactive power load at the bus, in Mvar. A positive value corresponds to power drawn from the network
- `BSHUNT` is the nominal reactive power of the shunt modelled as constant susceptance, in Mvar. This is the reactive power produced by the shunt under the nominal voltage of the bus, specified in its BUS record. BSHUNT is positive (resp. negative) for a shunt capacitor (resp. inductor)
- `QSHUNT` is the reactive power produced by the shunt modelled as constant power, in Mvar. QSHUNT is positive (resp. negative) for a shunt capacitor (resp. inductor).

The total reactive power Q produced by the two shunt components is given, in Mvar, by:

$$Q = BSHUNT\ (\frac{V}{V_{nom}})^2 + QSHUNT$$

where V is the bus voltage and V_{nom} the corresponding nominal voltage.

If no shunt is connected to the bus, BSHUNT and QSHUNT are set to zero.

If no load is connected to the bus, PLOAD and QLOAD are set to zero.

Let us recall that the PLOAD, QLOAD, BSHUNT and QSHUNT fields are ignored by RAMSES.

5.2 Generator data

Generators are specified by GENER records, as follows:

```
GENER NAME BUS P Q VIMP SNOM QMIN QMAX BR ;
```

where:

- `NAME` is the name of the generator. This is a string of a most 20 characters
- `BUS` is the name of the bus which the generator is connected to. This is a string of at most 8 characters
- `P` is the active power produced by the generator, in MW
- `Q` is the reactive power produced by the generator, in Mvar. This value is ignored if the `VIMP` field is nonzero
- `VIMP` is the voltage imposed by the generator, in pu. If `VIMP` is zero, the bus is treated as a PQ bus with the reactive power production set to `Q`; if `VIMP` is nonzero, the bus is treated as a PV bus and the `Q` field is ignored
- `SNOM` is the nominal apparent power of the generator, in MVA
- `QMIN` is the lower reactive power limit, in Mvar. It is used only if `V` is nonzero
- `QMAX` is the upper reactive power limit, in Mvar. It is used only if `V` is nonzero
- `BR` is the on/off status of the generator breaker. A zero value indicates that the breaker is open; any other value means that the breaker is closed.

For all generators with a nonzero value of `VIMP`, the connection bus is initially assumed of the PV type.

If this entails exceeding the generator upper reactive power limit `QMAX`, the bus switches to PQ type, the `QMAX` limit is enforced, and Newton iterations continue. If subsequently the bus voltage gets larger than `VIMP`, the bus switches back to PV type with the voltage imposed at the `VIMP` value.

Similarly, if the generator lower reactive power limit `QMIN` is exceeded, the bus switches to PQ type, the `QMIN` limit is enforced, and Newton iterations continue. If subsequently the bus voltage gets smaller than `VIMP`, the bus switches back to PV type with the voltage imposed at the `VIMP` value.

Only one generator is allowed per bus.

All generators are memorized, even those which are disconnected. A disconnected generator is not involved (it appears in the output results with a zero power output) but it can be put into service in the dynamic simulation.

Remark 1. For simplicity, a generator producing constant active and reactive powers can be modelled as a negative load using the BUS and no GENER record.

Remark 2. There exists a variant of the GENER record with the following syntax:

```
GENER NAME BUS P Q V SNOM QMIN QMAX PMIN PMAX BR ;
```

where `PMIN` (resp. `PMAX`) is the minimum (resp. maximum) active power that the generator can produce, in MW. If this record is present in the data, the `PMIN` and `PMAX` fields are ignored by STEPSS.

5.3 Slack-bus specification

The presence of a slack-bus is mandatory in power flow computations: indeed, not all buses can be of the PV or PQ type, since this would entail specifying the active power losses in the network, which are not known before performing the power flow calculation.

A generator of the PV type must be connected to the slack-bus. Its voltage magnitude, specified in its GENER record, is imposed at the bus, while the voltage phase angle is set to zero.

PFC can handle only one connected network (or island). If the network graph is not connected, *only the connected sub-network including the slack-bus is treated*; the rest of the network is ignored with a warning message.

The SLACK record allows specifying which bus is the slack-bus. The syntax is as follows:

```
SLACK NAME ;
```

where `NAME` is the name of the bus. This is a string of at most 8 characters.

There must be exactly one SLACK record in the whole set of data.

5.4 Static Var Compensators

5.4.1 Modelling

The Static Var Compensator (SVC) is modelled as shown in Fig. 5.1. j is the *monitored* bus, whose voltage is regulated, while i is the *controlled* bus, where the shunt susceptance B is varied. i and j can be any two buses but usually j is the transmission bus on the high-voltage side of the SVC step-up transformer, while the SVC is connected to bus i.

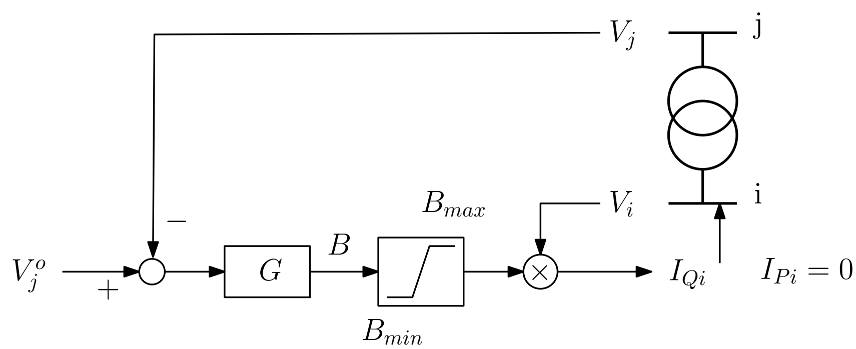


Figure 5.1: Block-diagram of the static var compensator

The SVC being assumed lossless, the active current injected at bus i is zero ($I_{Pi} = 0$), while the reactive current takes on one of the following forms:

$$I_{Qi} = BV_i = G(V_j^o - V_j)V_i \quad \text{under voltage control} \tag{5.1}$$

$$I_{Qi} = B_{max}V_i \quad \text{under upper susceptance limit} \tag{5.2}$$

$$I_{Qi} = B_{min}V_i \quad \text{under lower susceptance limit.} \tag{5.3}$$

Although reference is made to an SVC, the model can be used in general for a component controlling voltage with a droop.

5.4.2 Data format

The record, with keyword SVC, includes the following fields:

```
SVC NAME CON_BUS MON_BUS V0 Q0 SNOM BMAX BMIN G BR ;
```

where:

- `NAME` is the name of the compensator. This is a string of a most 20 characters
- `CON_BUS` is the name of the controlled bus. This is a string of a most 8 characters

- MON_BUS is the name of the monitored bus. This is a string of a most 8 characters
- V0 is the voltage setpoint V_j^o, in pu (see Fig. 5.1)
- Q0 is the reactive power setpoint, in Mvar. If V0 is set to zero, the SVC is treated under constant power, with $P = 0$ and $Q = Q0$, and no limit is tested. If V0 is nonzero, the Q0 field is ignored
- SNOM is the nominal reactive power of the SVC, in Mvar
- BMAX is the maximal nominal reactive power, in Mvar. This is the reactive power produced by the compensator under $V_i = 1$ pu, when B is at the maximal value B_{max}.
- BMIN is the minimal nominal reactive power, in Mvar. This is the reactive produced by the compensator under $V_i = 1$ pu, when B is at the minimal value B_{min}.
- G is the gain G, in pu on the (V_B, SNOM) base, where V_B is the nominal voltage at the controlled bus, as given by its BUS record
- BR is the on/off status of the SVC breaker. A zero value indicates that the breaker is open, any other value means that it is closed.

It is common for BMAX to be positive and BMIN negative but other combinations are allowed.

For all SVCs with a nonzero value of V0, Eq. (5.1) is solved initially.

If this entails exceeding the susceptance upper reactive power limit BMAX, Eq. (5.2) is substituted, the BMAX limit is enforced, and Newton iterations continue. If subsequently $G(V_j^o - V_j) < B_{max}$, the program reverts to Eq. 5.1 and keeps on iterating.

Similarly, if the SVC lower suscepance limit BMIN is exceeded, Eq. (5.3) is substituted, the BMIN limit is enforced, and Newton iterations continue. If subsequently $G(V_j^o - V_j) > B_{min}$, the program reverts to Eq. (5.1) and keeps on iterating.

Only one SVC is allowed per bus.

It is not allowed to connect both a generator and an SVC to the same bus.

All SVCs are memorized, even those which are disconnected. A disconnected SVC is not involved (it appears in the output results with a zero power output) but it can be put into service in the dynamic simulation.

5.5 Transformer ratio adjustment for voltage control

5.5.1 Modelling

PFC can adjust the ratio of a designated transformer with the objective of bringing a controlled voltage inside a deadband $[V_{des} - \epsilon \;\; V_{des} + \epsilon]$, where V_{des} is the desired voltage and ϵ is a tolerance.

The ratio is changed in discrete steps (as in the real life), between a minimum and a maximum value. During the computation, the ratio is changed by one step at a time, after which Newton iterations are performed until convergence is achieved. The process is repeated until the controlled voltage falls in the deadband. When multiple transformers are adjusted, some may reach their deadbands before others.

5.5.2 First data format

The first way to specify ratio adjustement is through the TRFO record. The following refers to the material presented in Section 4.4.2.

The controlled bus is `CONBUS`. This must be one of the two ending buses of the transformer. An empty or blank string *enclosed within quotes* indicates that the transformer ratio is not to be adjusted. In this case, however, dummy values must be provided for each of the fields listed below.

The fields of concern are:

- `NFIRST`: the ratio in % corresponding to the first tap position. This is the lower bound on the transformer ratio
- `NLAST`: the ratio in % corresponding to the last tap position. This is the upper bound on the transformer ratio
- `NBPOS`: the total number of tap positions including the first and the last
- `TOLV`: the voltage tolerance ϵ, in pu
- `VDES`: the desired voltage V_{des}, in pu.

The transformer ratio n corresponding to position p ($1 \leq p \leq \mathtt{NBPOS}$) of the tap changer is given by:

$$n = \frac{\mathtt{NFIRST}}{100} + \frac{p-1}{\mathtt{NBPOS}-1}\,\frac{\mathtt{NLAST}-\mathtt{NFIRST}}{100} \tag{5.4}$$

The value of p is increased or decreased by one at each adjustment iteration.

An initial value of the transformer ratio is given by the `N` field of the TRFO record. If the transformer is to be adjusted, before starting the power flow computation that value is adjusted to coincide with the nearest tap position, in accordance with Eq. (5.4).

5.5.3 Second data format

The second way to specify ratio adjustement is through a separate record with keyword LTC-V. The syntax is as follows:

```
LTC-V NAME CON_BUS NFIRST NLAST NBPOS TOLV VDES ;
```

where NAME is the name of the controlled transformer (a string of at most 20 characters) and all the other fields have the same meaning as for the TRFO record.

A transformer can be controlled by a single tap changer only. It is more natural to use the LTC-V record in association with a TRANSFO record. Hovever, the LTC-V record can be associated with a TRFO record, provided that no adjustment is specified in the latter.

5.6 Phase-shifting transformer ratio adjustment for power control

5.6.1 Modelling

PFC can also adjust the phase angle of a transformer with the objective of bringing the active power flow in a branch inside a deadband $[P_{des} - \epsilon \;\; P_{des} + \epsilon]$, where P_{des} is the desired power flow and ϵ is a tolerance.

The adjustment is very similar to that of in-phase transformers for voltage control detailed in Section 5.5.

5.6.2 Data format

The record, with keyword PSHIFT-P, includes the following fields:

```
PSHIFT-P CONTRFO MONBRANCH PHAFIRST PHALAST NBPOS SIGN PDES TOLP ;
```

where:

- CONTRFO is the name for the transformer whose phase angle is to be adjusted. This is a string of at most 20 characters, defined in either a TRFO or a TRANSFO record. If the transformer does not exist, the whole record is ignored with a warning message
- MONBRANCH is the name of the branch in which the active power flow P is monitored. This is a string of at most 20 characters, defined in either a LINE, a TRFO or a TRANSFO record. The sign convention is the following: P is the active power flow leaving the first bus specified in the LINE, TRFO or TRANSFO record and entering the branch of concern
- PHAFIRST is the phase angle ϕ, in degrees, corresponding to the first tap position. This is the lower bound on ϕ
- PHALAST is the phase angle ϕ, in degrees, corresponding to the last tap position. This is the upper bound on ϕ
- NBPOS is the number of tap positions
- SIGN is an indication of the direction in which the phase angle ϕ must be adjusted to reach

the objective. A value of 1 indicates that ϕ must be increased to increase the active power flow in the monitored branch. A value of -1 indicates that it must be decreased. Any other value is invalid and causes the program to stop

- `PDES` is the desired active power flow, in MW
- `TOLP` is the tolerance ϵ, in MW.

The phase angle ϕ corresponding to position p ($1 \leq p \leq \texttt{NBPOS}$) of the tap changer is given by:

$$\phi = \texttt{PHAFIRST} + \frac{p-1}{\texttt{NBPOS}-1}\,(\texttt{NLAST} - \texttt{NFIRST}) \tag{5.5}$$

The value of p is increased or decreased by one at each adjustment iteration.

PFC performs a sensitivity analysis to determine whether the phase angle should be increased or decreased. If this analysis indicates a direction opposite to what is specified in `SIGN`, a warning is issued and the value of `SIGN` is ignored. On output, when it produces a file with the records updated, PFC sets `SIGN` to the value corresponding to its sensitivity analysis.

An initial value of the phase angle is given by the `PHI` field of the TRANSFO record. If the transformer is to be adjusted, before starting the power flow computation, that value is adjusted to coincide with the nearest tap position, in accordance with Eq. (5.5).

A transformer cannot be controlled by both an LTC-V and a PSHIFT-P record.

Only one PSHIFT-P record per transformer is allowed. The PSHIFT-P record is intended to be used in association with a TRANSFO record. However, it is allowed to associate it with a TRFO record in spite of the fact that the latter assumes a zero phase angle. In this case, the angle will be initialized to zero and will be controlled as specified in the PSHIFT-P record.

5.7 Bus voltages: initial values and results

On output, PFC produces a file with the computed bus voltage magnitudes and phase angles. The latter are specified in records with keyword LFRESV. The syntax is as follows:

```
LFRESV BUS MODV PHASV ;
```

where:

- `BUS` is the name of the bus. This is a string of a most 8 characters defined in a BUS record
- `MODV` is the bus voltage magnitude, in pu
- `PHAV` is the bus voltage phase angle, in radian, the slack bus being the reference.

If LFRESV records are provided on input, they are used as initial voltages of the Newton iterations, at the specified buses.

If no LFRESV record is specified at a bus:

- the voltage magnitude is initialized to 1 pu if the bus is of the PQ type or to the value imposed by the generator in case of a PV bus
- the phase angle is initialized to 0 degree.

Thus, if the LFRESV records obtained from a first run of PFC are added to the data files, the other data being unchanged, no Newton iteration is going to be performed since we are already at the solution. This is an easy way to check that system data come with their corresponding voltages.

5.8 Share of records beween PFC and RAMSES

A summary of the records used by respectively PFC and RAMSES is given in Table 5.1.

Table 5.1: records used by PFC and RAMSES, respectively

record	in PFC	in RAMSES
BUS	all 6 fields used	only first two fields used
LINE	used	used
SWITCH	used	used
NRTP	used	used
TRANSFO	used	used
TRFO	used	only fields 1 to 9 and 15 used
SHUNT	ignored	used
GENER	used	ignored
SVC	used	ignored
SLACK	used	used
LFRESV	used	used
LTC-V	used	ignored
PHSHIFT-P	used	ignored

5.9 Computation control parameters

5.9.1 Parameters

PFC performs Newton(-Raphson) iterations to solve the power flow equations. At the k-th iteration, the following indices are computed:

$\epsilon_P = \max_i |f_i(\mathbf{v}^{(k)}, \boldsymbol{\theta}^{(k)}) - P_i^o|$ the largest absolute mismatch of the active power equations

$\epsilon_Q = \max_i |g_i(\mathbf{v}^{(k)}, \boldsymbol{\theta}^{(k)}) - Q_i^o|$ the largest absolute mismatch of the reactive power equations

$\epsilon_S = \max_i \sqrt{(f_i(\mathbf{v}^{(k)}, \boldsymbol{\theta}^{(k)}) - P_i^o)^2 + (g_i(\mathbf{v}^{(k)}, \boldsymbol{\theta}^{(k)}) - Q_i^o)^2}$ the largest apparent power mismatch.

Convergence is achieved once :

- both ϵ_P and ϵ_Q are below specified thresholds
- all controls on transformer ratios and phase shifts are satisfied, and
- all generators and SVCs are within their reactive limits.

ϵ_S is used to check that the solution is accurate enough to :

- "freeze" the Jacobian matrix (in order to save some computing time)
- check the generator and SVC reactive power limits and enforce the latter if needed
- adjust the transformer ratios and phase shifs to control voltage magnitudes and active power flows, if specified in the data.

Finally, the iterations stop as soon as divergence is detected. To this purpose the quadratic index:

$$\varphi(k) = \sum_i \sqrt{(f_i(\mathbf{v}^{(k)}, \boldsymbol{\theta}^{(k)}) - P_i^o)^2 + (g_i(\mathbf{v}^{(k)}, \boldsymbol{\theta}^{(k)}) - Q_i^o)^2}$$

is monitored. Under normal convergence conditions, φ decreases from one iteration to the next. Therefore, the increase in φ is used to detect divergence. The algorithm stops at the iteration k such that :

$$\varphi(k) > 1.1\, \varphi(k-1)$$

However, this test is skipped at any iteration k that follows the switching of generators or SVCs under limit, or the adjustment of transformer ratios and phase shifts[1].

5.9.2 Records

The records detailed herafter are used to control the computation. They all start with a $ to distinguish them from the other records. They all have a single field, as detailed in Table 5.2. The default value assigned in the absence of the record is given in the fourth column.

[1] indeed, these adjustments cause increases in φ that have nothing to do with divergence

Table 5.2: Computation control parameters

record	meaning of field	unit	default value
`$SBASE`	base power (on which pu values are expressed)	MVA	100
`$TOLAC`	ϵ_P	MW	0.1
`$TOLREAC`	ϵ_Q	Mvar	0.1
`$NBITMA`	max number of iterations	-	20
`$MISQLIM`	value of ϵ_S below which the reactive power limits are checked and enforced. Set to 0 to skip the limit check	MVA	20
`$MISBLOC`	value of ϵ_S below which the Jabobian is kept constant.	MVA	10
`$MISADJ`	value of ϵ_S below which transformers are adjusted. Set to 0 to skip adjustment	MVA	10
`$DIVDET`	set to 1 to activate divergence test during iterations. Set to 0 to skip test	-	0

Part III

Dynamic Simulation with RAMSES

Chapter 6

Reference frame and model initialization

6.1 Phasor approximation and reference frames

Under the phasor approximation, the network equations can be written in compact form as:

$$\bar{\mathbf{I}} = \mathbf{Y}\bar{\mathbf{V}} \tag{6.1}$$

where:

- $\bar{\mathbf{I}}$ is the vector of complex currents injected into the network at the various buses
- $\bar{\mathbf{V}}$ is the vector of complex voltages at the various buses
- $\mathbf{Y}$ is the bus (or nodal) admittance matrix.

Which frequency consider for the phasors and the admittance matrix ? In dynamic regime, each synchronous machine defines a local frequency. In most cases, those various frequencies remain close to the nominal frequency f_N. For large deviations with respect to the nominal value f_N, it can be envisaged to update the entries of the $\mathbf{Y}$ matrix using the average system frequency, for instance. Otherwise, the admittances are simply computed at frequency f_N.

Under the phasor approximation, the voltage at the i-th bus takes on the form:

$$\begin{aligned} v_i(t) &= \sqrt{2}V_i(t)\cos(\omega_N t + \phi_i(t)) = \sqrt{2}\, re\left[V_i(t)e^{j\,\phi_i(t)}e^{j\,\omega_N t}\right] \\ &= \sqrt{2}\, re\left[(v_{xi}(t) + j v_{yi}(t))\, e^{j\,\omega_N t}\right] \end{aligned} \tag{6.2}$$

where $v_{xi}(t) + j\, v_{yi}(t)$ is the voltage phasor, in rectangular coordinates, expressed with respect to (x, y) axes rotating at the angular speed $\omega_N = 2\pi f_N$, as illustrated in Fig. 6.1.

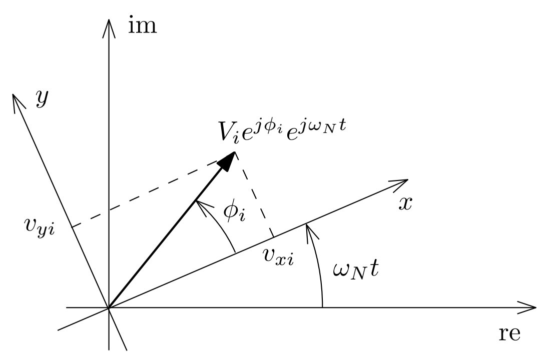


Figure 6.1: Reference axes and voltage phasor

Equations similar to (6.2) can be written for the currents injected into the network, the currents flowing into the network branches, etc.

Instead of determining the "full wave" evolution of voltages or currents, dynamic simulation in phasor mode aims at rendering the time evolution of v_{xi} and v_{yi}[1].

[1]In RAMSES the rectangular components have been preferred to the polar components V_i and ϕ_i

The (x, y) axes rotating at angular speed ω_N make up a *synchronous reference*.

Although simple, this reference frame suffers from a significant limitation. Indeed, assuming that the power system initially operates at the nominal franquency f_N[2], after a disturbance, it will settle at a different frequency f, unless its model includes an infinite bus imposing the frequency f_N. From there on, the various voltage and current phasors rotate at the angular speed $2\pi f \neq \omega_N$. Hence, phasor components such as v_{xi} and v_{yi} will oscillate at a frequency $|f - f_N|$, although the system is at equilibrium from a practical viewpoint. For that reason, the synchronous reference is not suitable for long-term simulations, since tracking the oscillations at frequency $|f - f_N|$ requires using a small enough time step size. The synchronous reference frame is suited to short-term simulations (where frequency has not yet returned to steady state) or when the model includes an infinite bus driving the frequency back to f_N.

In fact, any (more convenient) speed can be considered for the reference axes (x, y). The only constraint is that all voltage and current phasors refer to the same axes.

In the *Center Of Inertia (COI)* reference frame, the (x, y) axes rotate at the angular frequency:

$$\omega_{coi} = \frac{\sum_{i=1}^{m} M_i \omega_i}{\sum_{i=1}^{m} M_i} \tag{6.3}$$

where:

m is the total number of synchronous machines
ω_i is the rotor speed of i-th synchronous machine ($i = 1, \dots, m$)
M_i is the inertia coefficient of the i-th machine ($i = 1, \dots, m$).

The M_i values relate to the inertia constants H_i (in s) of the individual machines, expressed on a common base power S_B (in MVA):

$$M_i = 2H_i \frac{S_{Ni}}{S_B}$$

where S_{Ni} is the nominal apparent power of the i-th machine (in MVA).

The COI reference frame does not suffer from the above mentioned drawback. Indeed, when the system settles at a frequency f, all synchronous machines rotate at the angular speed $2\pi f$, and so do the reference axes ($\omega_{coi} = 2\pi f$). Hence, phasor components such as v_{xi} and v_{yi} tend to constant values, a larger time step size can be used and the simulation is computationally less demanding. The COI reference frame is well suited long-term simulations.

The COI frequency can be used as an average system frequency in injector and two-port models: see Section 11.4.4.

[2] This is a common assumption in power system simulation software

6.1.1 Specifying the reference frame

The reference to be used is specified in the simulation settings: see Chapter 9.

The presence of a Thévenin equivalent (involving a constant frequency voltage source) among the components causes the simulation to use the synchronous reference frame.

6.1.2 Implementation of COI reference frame

To preserve the "sparsity" of the model in spite of Eq. (6.3), which embeds the rotor speeds of all synchronous machines, the value of ω_{coi} at the previous time step is used. See:

> D. Fabozzi and T. Van Cutsem "On angle references in long-term time-domain simulations", IEEE Transactions on Power Systems, Vol. 26, No 1, pp. 483-484, Feb. 2011

for a more detailed presentation of this technique.

6.2 Network equations

With all voltage and current phasors referred to the (x, y) axes, the network equations can be decomposed into:

$$\mathbf{i}_x + j\,\mathbf{i}_y = \mathbf{Y}\,(\mathbf{v}_x + j\,\mathbf{v}_y) = (\mathbf{G} + j\,\mathbf{B})\,(\mathbf{v}_x + j\,\mathbf{v}_y) \tag{6.4}$$

with:

$$\mathbf{v}_x = \begin{bmatrix} v_{x1} \\ \vdots \\ v_{xN} \end{bmatrix} \qquad \mathbf{v}_y = \begin{bmatrix} v_{y1} \\ \vdots \\ v_{yN} \end{bmatrix} \qquad \mathbf{i}_x = \begin{bmatrix} i_{x1} \\ \vdots \\ i_{xN} \end{bmatrix} \qquad \mathbf{i}_y = \begin{bmatrix} i_{y1} \\ \vdots \\ i_{yN} \end{bmatrix} \tag{6.5}$$

where $\mathbf{G}$ is the conductance matrix and $\mathbf{B}$ the susceptance matrix.

Decomposing into real and imaginary parts and assembling into a single equation yields:

$$\begin{bmatrix} \mathbf{i}_x \\ \mathbf{i}_y \end{bmatrix} = \begin{bmatrix} \mathbf{G} & -\mathbf{B} \\ \mathbf{B} & \mathbf{G} \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \end{bmatrix} \tag{6.6}$$

Thus, for a network with N buses, there are $2N$ equations (6.6) involving $4N$ variables.

6.3 Initialization procedure

The dynamic simulation is initialized according to the procedure detailed next.

1. The starting point is the vector of initial bus voltages, as detailed in Section 5.7
2. setting the bus voltages to those values, the active and reactive power flows in the network branches and shunts are computed
3. by summing the power flows in incident branches, the bus power injections are determined. This is illustrated in Fig. 6.2, where P_i^{inj} (resp. Q_i^{inj}) is the active (resp. reactive) power injected into the i-th bus. A *positive value* corresponds to power *entering the network*
4. at each bus, the bus power injection is shared among the various components (generators, loads, injectors, etc.) connected to that bus.
   There are two ways of assigning power to the j-th component:
   (i) by specifying the active and reactive powers injected into the network by that component:
   $$P_j^c = P_j^{c0} \qquad Q_j^c = Q_j^{c0} \tag{6.7}$$
   (ii) by specifying the fraction of the bus power injection taken by that component :
   $$P_j^c = f_{Pj}\, P_i^{inj} \qquad Q_j^c = f_{Qj}\, Q_i^{inj} \tag{6.8}$$
   where f_{Pj} (resp. f_{Qj}) denotes the fraction relative to active (resp. reactive) power.
5. the remaining power, not taken by the components at bus i, i.e.
   $$P_i^r = P_i^{inj} - \sum_{j=1}^{n} P_j^c \qquad Q_i^r = Q_i^{inj} - \sum_{j=1}^{n} Q_j^c$$
   is checked. If larger than an internal tolerance, the power is assigned to an impedance load[3], as shown in Fig. 6.2. The load admittance is such that the power balance is satisfied at $t = 0$:
   $$(G_i^r - j\, B_i^r)\, V_i(0)^2 = -(P_i^r + jQ_i^r) \tag{6.9}$$
   where $V_i(0)$ is the initial bus voltage magnitude[4]. Note that the sign of G_i^r is opposite to that of P_i^r, while the sign of B_i^r is the same as that of Q_i^r.

The two ways mentioned under items (i) and (ii) above are *mutually exclusive*, i.e. either a nonzero power or a nonzero fraction is specified, but not both. In mathematical terms:

$$f_{Pj} \times P_j^{c0} = 0 \qquad f_{Qj} \times Q_j^{c0} = 0 \tag{6.10}$$

Although they are typically in the interval $[0\ 1]$, the values of the f_{Pj}'s and f_{Qj}'s may be negative or larger than one.

[3] to be more precise a constant shunt impedance, or equivalently a constant shunt admittance load

[4] the left-hand side of Eq. (6.9) is the complex power consumed by the admittance, while $P_i^r + jQ_i^r$ is counted positive when the power is injected into the network. Hence, the minus sign in the right hand side of the equation

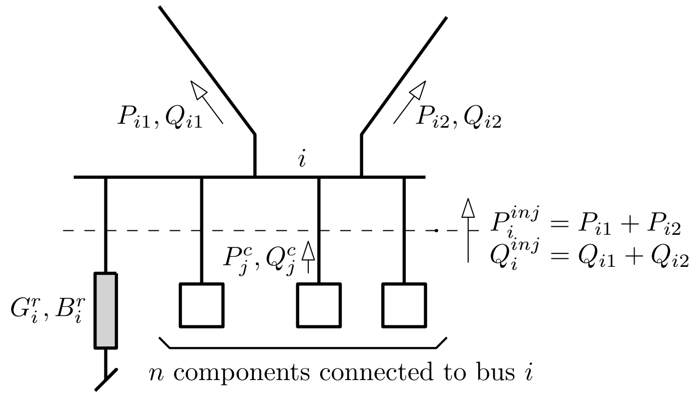


Figure 6.2: initial powers at bus i

In the quite common case when a single component is connected to a bus, and it takes the whole power consumed or produced at that bus, it is convenient to specify:

$$f_{Pj} = 1 \qquad P_j^{c0} = 0 \qquad f_{Qj} = 1 \qquad Q_j^{c0} = 0$$

With fractions instead of powers, the data can be re-used at another initial operating point[5].

The constant admittance (or impedance) loads corresponding to Eq. (6.9) are created automatically at system initialization. They are given names of the type `M_bus`, where `M` stands for "mismatch" and `bus` is the name of the bus to which the admittance is connected. Names starting with `M_` are reserved and must not be used when defining constant admittance loads.

A large value of G_i^r (resp. B_i^r) may be intentional, when a load is to be modelled as a constant shunt admittance and the task is left to RAMSES. However, it may also result from a mistake in the initial power balance. It is recommended to check the values of the `M_` loads through the menu "Load Initialization" in the STEPSS interface.

☛ Here is an example of output produced by RAMSES at initialization.

```
NUMBER OF IMPEDANCE LOADS :     3  (M_ type:     3 )

load name              bus name                  P          Q

M_2                    2                    90.002     17.997
M_3                    3                     0.013     -0.011
M_4                    4                    -0.017     -0.022
```

In this example, three `M_` loads have been produced at initialization, respectively at buses 2, 3 and 4. The one at bus 3 consumes 0.013 MW and produces 0.011 Mvar. This load was created because it is larger than the internal tolerance but, for a transmission system, this is a negligible value. The same holds true for the `M_` load at bus

[5] i.e. when another set of initial voltages is used

4. On the other hand, the `M_` load at bus 2 is non negligible. This could be due to forgetting to specify a load at bus 2.

6.4 Data format

The way to specify the values of f_{Pj}, P_j^{c0}, f_{Qj} and Q_j^{c0} is detailed in the following subsequent sections of this documentation:

- for a synchronous machine: Section **??**
- for a Thévenin equivalent: Section **??**
- for a constant impedance load: Section **??**
- for an injector with user-defined model: Section 11.6.1
- for a two-port with user-defined model: Section 11.6.2

Chapter 7

Modelling of synchronous machines, Thévenin equivalents and impedance loads

7.1 Synchronous Machines and Controls

7.1.1 Synchronous Machine

SYNC_MACH Sync_Mach_Name BUS_NAME FP FQ P Q Snom Pnom H D ibratio XT/RL Xl Xd X'd X"d Xq X'q X"q m n Ra T'do T"do T'qo T"qo

EXC EXC_TYPE parameters_passed_to_EXC

TOR TOR_TYPE parameters_passed_to_TOR ;

7.2 Injectors

INJEC INJ_TYPE NAME BUS_NAME FP FQ P Q parameters_passed_to_INJ ;

7.3 Infinite Bus

INJEC THEVEQ INJEC_NAME BUS_NAME FP FQ P Q MVA ;

7.4 Impedance Load

IMPLOAD loadname BUS_NAME FP FQ P Q ;

Chapter 8

Disturbances

Disturbances need to have a continuity.

8.1 Continue Solver

time(s) CONTINUE SOLVER disc_meth max_h(s) min_h(s) latency(pu) upd_over

Discretization method (disc_meth):

- TR: Trapezoidal
- BE: Backward Euler
- BD: BDF2

Jacobian update override (upd_over):

- ALL: Update all injectors and network
- NET: Update only network
- ABL: Update only injectors
- IBL: Update all injectors and network
- NOT: Do not override

It is mainly used to modify the settings of the solver and has to exist at the first line. For example:

0.000 CONTINUE SOLVER BD 0.0200 0.001 0. ALL

8.2 Set stopping criteria for voltage

Define in the data files the following record:

DCTL SIM_MINMAXVOLT CTRL_Name VMAX(pu) VMIN(pu) DEADTIME(s) Stop_Simulation(T/F) ;

8.3 Set stopping criteria for machine speed

Define in the data files the following record:

DCTL SIM_MINMAXVOLT CTRL_Name MAX_SYNC_SPEED(pu) MIN_SYNC_SPEED(pu) DEADTIME(s) Stop_Simulation(T/F) ;

8.4 Stop

time(s) STOP

It is used to signal the end of the simulation and has to exist at the last line.

For example:

100.000 STOP

8.5 Trip line

time(s) BREAKER BRANCH name_of_line orig_break(0/1) extrem_break(0/1)

Used to open/close the breakers of a line.

For example, opening both ends of a line:

10.000 BREAKER BRANCH 1044-4032 0 0

8.6 Trip synchronous machine / injector

time(s) BREAKER SYNC_MACH/INJ name_of_synch_mach/name_of_injector breaker(0/1)

Used to open/close the breaker (trip) of a synchronous machine or injector.

For example:

10.000 BREAKER INJ L_11 0

8.7 Three phase short-circuit (with use of resistance to ground)

This demands two commands:

time(s) FAULT BUS name_of_bus rfault [xfault] time(s) CLEAR BUS name_of_bus

The first line is used to declare the starting of the short-circuit. The fault has a resistance of *rfault+j*xfault* to the ground where the values of rfault and xfault are in Ohm. If xfault is not defined, a fully resistive fault is assumed.

Example of a 100ms short-circuit directly to ground:

10.000 FAULT BUS 1044 0. 0.
10.100 CLEAR BUS 1044

8.8 Three phase short-circuit (with use of voltage reached after fault)

This demands two commands:

time(s) VFAULT BUS name_of_bus Voltage_reached_after_fault

time(s) CLEAR BUS name_of_bus

The first line is used to declare the starting of the short-circuit. The fault has an unknown resistance *(j*xfault)* to the ground. Based on the value of the voltage reached after the fault (declared in pu), xfault is computed and used for simulating the fault.

Example of a 100ms short-circuit where the voltage at the faulted bus reached 0.5 pu after the fault:

10.000 VFAULT BUS 1044 0.5
10.100 CLEAR BUS 1044

Supported: Version 3.13 and above.

8.9 Change parameters

Used to change the parameters of a model during the simulation.

8.9.1 BRANCH

time(s) CHGPRM BRANCH name_of_line MAGN/PHAN ±increment

8.9.2 SHUNT

time(s) CHGPRM SHUNT name_of_shunt QNOM ±increment

The increment should be expressed in MVAr and it is per unitized internally by RAMSES.

8.9.3 EXC

time(s) CHGPRM EXC name_of_equipment name_of_parameter ±increment (MVAr/%) duration(s)

The units is not obligatory. If nothing is given then the parameter is modified in absolute value. If MVAr is given, then the increment is per unitized using *Snom* of the machine before applying the change. If % is given, then the parameter is changed as a percentage of the original value. If $duration = 0$ then a step change is applied, otherwise the change is applied as a ramp over the given duration (in seconds).

For example:

10.000 CHGPRM EXC g1 V0 +10 % 10

This means the parameter V0 of the exciter of synchronous machine g1 is ramped by +10% between 10 and 20 seconds.

8.9.4 TOR

time(s) CHGPRM TOR name_of_equipment name_of_parameter ±increment (MW/%) duration(s)

The units is not obligatory. If nothing is given then the parameter is modified in absolute value. If MW is given, then the increment is per unitized using *Pnom* of the machine before applying the change. If % is given, then the parameter is changed as a percentage of the original value. If $duration = 0$ then a step change is applied, otherwise the change is applied as a ramp over the given duration (in seconds).

For example:

10.000 CHGPRM TOR g1 P0 +1 MW 10

This means the parameter P0 of the torque controller of synchronous machine g1 is ramped by +1 MW between 10 and 20 seconds.

8.9.5 INJ/TWOP/DCTL

time(s) CHGPRM INJ/TWOP/DCTL name_of_equipment name_of_parameter ±increment (MW/MVAr/%/SETP) duration(s)

The units is not obligatory. If nothing is given then the parameter is modified in absolute value. If MW or MVAr is given, then the increment is per unitized using system's *Sbase* before applying the change. If % is given, then the parameter is changed as a percentage of the original value. IF SETP is given then the value increment is actually the new setpoint. If $duration = 0$ then a step change is applied, otherwise the change is applied as a ramp over the given duration (in seconds).

For example:

10.000 CHGPRM INJ L_11 P0 +50 % 60
10.000 CHGPRM INJ L_11 Q0 +30 % 60

This means the parameter P0 (resp. Q0) of the injector L_11 is ramped by +50% (resp. 30%) between 10 and 70 seconds. In this case, if L_11 is a load model, it can be used to simulate a load increase.

8.10 Export Jacobian matrix

time(s) JAC 'name_of_filename'

Also, make sure to add these to your settings:

$OMEGA_REF SYN ;

$SCHEME IN;

8.11 Export load flow

Takes a snapshot of the system and exports the load flow at a specific time.

time(s) LFRESV 'name_of_filename'

Chapter 9

Solver Settings

9.1 Sampling time for observed variables

$PLOT_STEP time(s) ;

9.2 Display Profiling Results

$DISP_PROF T/F ;

9.3 Run-time observables refresh interval

$GP_REFRESH_RATE time_interval(s) ;

9.4 Time constant of load restoration

$T_LOAD_REST time(s) ;

9.5 Omega Reference

$OMEGA_REF SYN/COI ;

Synchronous reference frame or center of inertia reference frame.

9.6 Maximum Fault Value

$MAX_FAULT value ;

9.7 Base Power

Sets the global base power of the system.

$S_BASE BASE(MVA) ;

9.8 Nominal Frequency

FNOM Frequency(Hz) ;

9.9 Newton Tolerance

$NEWTON_TOLER NETWORK_TOLERANCE INJ_RELATIVE_TOLERANCE INJ_ABSOLUTE_TOLERANCE ;

Set's the solver Newton iteration tolerance for stopping. Default values are: 1e-03, 5e-04, 5e-04.

9.10 Finite Difference Values

$FIN_DIFFER proportional_value absolute_value ;

Values used to calculate numerically Jacobian matrices of injectors.

9.11 Full Jacobian Update

$FULL_UPDATE T/F ;

Disable partial Jacobian updates.

9.12 Skip Converged Blocks

$SKIP_CONV T/F ;

Activate/Deactivate stopping to solve converged injectors.

9.13 Latency Tolerance

$LATENCY OBS_TIME_WINDOW(s) EARLY_STOP(T/F) ;

9.14 Solution Scheme

$SCHEME DE/IN ;

Integrated or Decomposed solution scheme.

9.15 Number of Threads for parallel computing

$NB_THREADS Number ;

9.16 Way of injector distribution over parallel threads

$OMP STA/DYN/GUI chunk ;

STA is for static assignment (better for NUMA architecture computers), DYN is for dynamic assignment (better for UMA architecture computers) and GUI is for guided. Chunk is the number of consecutive injectors assigned to each thread.

9.17 Update network elements with frequency

$NET_FREQ_UPD T/F ;

Check Network update with frequency.

Chapter 10

Discrete Controllers

10.1 Discrete Controllers

10.1.1 LTC

DCTL LTC CTLNAME TRFONAME BUS_NAME DIR NMIN NMAX NBPOS TOL DELAY1 DELAY2 ;

10.1.2 Real-time synchronizer

DCTL RT CTLNAME ratio_to_rt ;

If ratio_to_rt is set to 1.0, the simulation will be slowed down when it's faster than RT to synchronize. The moments it is slower, nothing will be done. Setting ratio_to_rt to 2.0, means twice faster than RT (if possible), etc.

10.2 Two-port Injectors

Part IV

Adding user-defined models with CODEGEN

Chapter 11

User-defined models: mathematical formulation and syntax of description

11.1 States and equations

11.1.1 States

The four categories of user-defined models and their acronyms are as follows:

- EXC excitation controller of synchronous machine: typically the excitation system and the Automatic Voltage Regulator (AVR), including the Power System Stabilizer (PSS)
- TOR torque controller of synchronous machine: typically the turbine and the speed governor
- INJ injector: a component connected to a single AC bus
- TWOP two-port: a component connecting two buses.

Whichever its type, any model has input states (grouped in $\mathbf{x}_{IN}$), internal states (in $\mathbf{x}_{ITL}$) and output states (in $\mathbf{x}_{OUT}$) as sketched in Fig. 11.1. Note that states can be indifferently differential or algebraic states.

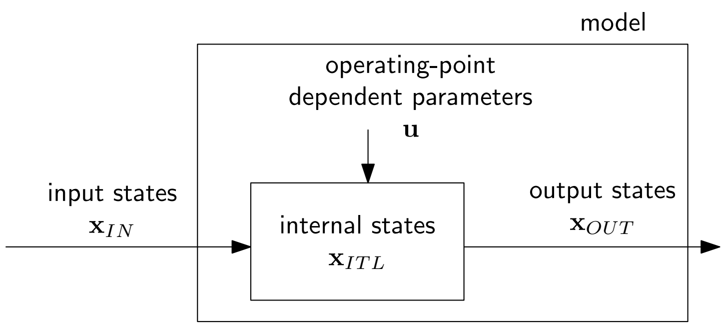


Figure 11.1: Outline of a model

The input and output states pertaining to each category of model are detailed in Table 11.1. The notation is as follows:

For an excitation controller:

- V terminal voltage of the machine, in pu
- P active power produced, in pu on the machine rated apparent power (MVA)
- Q reactive power produced, in pu on the machine rated apparent power (MVA)
- ω rotor speed, in pu
- i_f field current, in pu on the exciter base current
- v_f field voltage, in pu on the exciter base voltage.

For a torque controller:

- ω rotor speed, in pu
- P active power produced, in pu on the turbine rated power (MW)
- T_m mechanical torque applied to rotor, in pu on the turbine rated torque.

Table 11.1: Input and output states

type of model	input states	output states
excitation controller of synchronous machine	V, P, Q, ω, i_f of machine	v_f of machine
torque controller of synchronous machine	P, ω of machine	T_m of machine
injector	v_x, v_y at bus ω_{coi}	i_x, i_y into bus
two-port	$v_{x1}, v_{y1}, v_{x2}, v_{y2}$ at buses ω_{coi}	$i_{x1}, i_{y1}, i_{x2}, i_{y2}$ into buses

For an injector:

- v_x, v_y rectangular components of phasor of voltage at the connection bus (the (x, y) reference axes have been defined in Section 6.1)
- i_x, i_y rectangular components of phasor of current injected into the connection bus
- ω_{coi} angular frequency of center of inertia, in pu[1].

For a two-port:

- v_{x1}, v_{y1} rectangular components of phasor of voltage at bus 1
- v_{x2}, v_{y2} rectangular components of phasor of voltage at bus 2
- i_{x1}, i_{y1} rectangular components of phasor of current injected into bus 1
- i_{x2}, i_{y2} rectangular components of phasor of current injected into bus 2
- ω_{coi} angular frequency of center of inertia, in pu.

The models also include operating-point dependent parameters (grouped in $\mathbf{u}$).

The three categories of states are treated as follows:

- **at the initialization** of the model:
  - the input states are given. They are obtained from the synchronous machine states (see Section ???) or the initial power injected into buses (see Section 6.1)
  - the output states are also given, using the same information[2]
  - the internal states are initialized either explicitly by the user or automatically by RAMSES
  - the operating-point dependent parameters are initialized by the user, in agreement with the initial values of the states

[1] This can be used as an average system frequency. One of the modelling blocks (see next Chapter) offers the possibility to estimate the "local" frequency at the connection bus

[2] at first glance, specifying both the input and the output states would lead to an overdetermined system with more equations than states; this is not the case since the excess equations are used to initialize the operating-point dependent parameters $\mathbf{u}$. The number of the latter is equal to the number of output states

- **during the simulation**:
  - the input, the internal and the output states are computed together with the other system states
  - the operating-point dependent parameters remain constant, unless they are modified by a user action (see Section **??**).

Note that the model must have a number of equations at least equal to the number of output states, otherwise it is not correctly formulated.

Note also that not all input states need be used.

☛ Consider the very simple excitation system shown in Fig. 11.2. The following equations are easily derived:

$$0 = V^o - V - dV \tag{11.1}$$
$$T\,\frac{dv_f}{dt} = -v_f + G\,dV \tag{11.2}$$

The model has a single input (V), a single internal state (dV) and the requested output state (v_f). dV is an algebraic state, while v_f is a differential one. V^o is the single component of the $\mathbf{u}$ vector.

At initialization, the system is assumed in steady state: $\frac{dv_f}{dt} = 0$. Hence, $dV(0) = \frac{v_f(0)}{G}$, where (0) denotes values at $t = 0$. V^o is obtained from Eq. (11.1) as: $V^o = V(0) + dV(0)$.

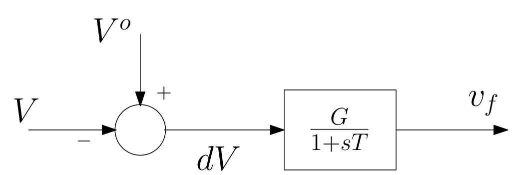


Figure 11.2: A very simple model of excitation system

11.1.2 Equations

As they are determined from the machine and the network equations, the input states are not part of the user model state vector, which thus takes on the form:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_{ITL} \\ \mathbf{x}_{OUT} \end{bmatrix} \tag{11.3}$$

The differential-algebraic equations can be written in compact form as:

$$\mathbf{T}\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{x}_{IN}, \boldsymbol{u}, \boldsymbol{z}) \tag{11.4}$$

where $\mathbf{x}$ and $\mathbf{f}$ have the same dimension. The i-th row of matrix $\mathbf{T}$ includes:

- only zeros if the i-th equation is algebraic
- a single nonzero term T_{ij} if the i-th equation is differential with:

$$T_{ij}\dot{x}_j = f_i(\mathbf{x}, \mathbf{x}_{IN}, \mathbf{u}, \mathbf{z})$$

  where i and j may be different.

The nonzero components of $\mathbf{T}$ are time constants, typically.

$\mathbf{z}$ is a vector of discrete variables whose role is detailed in the next section.

11.2 Discrete transitions

The material of this section is provided for information in so far as the corresponding treatment is performed automatically by STEPSS. Nevertheless, it may help interpreting the output curves and/or some execution messages.

11.2.1 Formulation

☛ Consider now a little more detailed version of the model in Fig. 11.2, including non-windup limits on the integrator embedded in the first-order transfer function, as shown in Fig. 11.3.

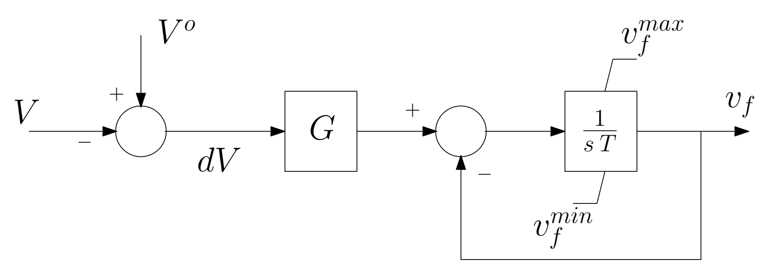


Figure 11.3: A little more detailed version of the system in Fig 11.2

The system is modelled by one of the following three sets of equations:

- if the integrator is not at any of its limits ($z = 0$):

$$0 = V^o - V - dV \tag{11.5}$$
$$T\,\frac{dv_f}{dt} = -v_f + G\,dV \tag{11.6}$$

- if the integrator is at its upper limit ($z = 1$):

$$0 = V^o - V - dV \tag{11.7}$$
$$0 = v_f^{max} - v_f \tag{11.8}$$

- if the integrator is at its lower limit ($z = -1$):

$$0 = V^o - V - dV \tag{11.9}$$
$$0 = v_f^{min} - v_f \tag{11.10}$$

As shown in the above example, a discrete variable z is used to identify which set of equations must be solved at any given time of the simulation. The values assigned to z are completely arbitrary. Changing z from one value to another corresponds to substituting one set of equations to another. Such "discrete transitions" take place when some inequality constraints are violated. A typical example is when a state exceeds its prescribed limit.

Note incidentally that differential equations can be changed into algebraic ones, and conversely. This is a feature offered by RAMSES.

☛ In the example of Eqs. (11.5-11.10), here is the pseudo-code performing the discrete transitions relative to the non-windup integrator:

if $z = 0$ **then**
  if $v_f > v_f^{max}$ **then**
    $z \leftarrow 1$
  else if $v_f < v_f^{min}$ **then**
    $z \leftarrow -1$
  end if
else if $z = 1$ **then**
  if $(-v_f + G\,dV)/T < 0$ **then**
    $z \leftarrow 0$
  end if
else if $z = -1$ **then**
  if $(-v_f + G\,dV)/T > 0$ **then**
    $z \leftarrow 0$
  end if
end if

The z variables are initialized at the beginning of the simulation, as for the states.

☛ In the example of Eqs. (11.5-11.10), here is the pseudo-code initializing the z variable relative to the non-windup integrator:

if $v_f > v_f^{max}$ **then**
  $z \leftarrow 1$
else if $v_f < v_f^{min}$ **then**
  $z \leftarrow -1$

else
$z \leftarrow 0$
end if

In the above example, at $t = 0$, if v_f violates one of its limits, it is brought back to that limit. This means that a discrete transition will take place right away at $t = 0$.

11.2.2 Discrete transition identification and treatment scheme

This section deals with the handling of discrete transitions when passing from time t to time $t + h$, where h is the time step size. It is assumed that the system equations have been irrevocably solved until time t.

When the solver detects that the condition for a discrete transition has been satisfied in the time interval $[t \;\; t+h]$, it does not attempt to identify the exact time t' $(t < t' \leq t+h)$ when the condition became satisfied. Instead, the following *ex post* solution scheme is used:

1. for the value of the discrete variables $\mathbf{z}$ prevailing at time t, Eqs. (11.4) are integrated from t to $t + h$ with full accuracy[3]
2. at the resulting point, the inequality constraints associated with discrete transitions are checked. If needed, $\mathbf{z}$ is changed, i.e. Eqs. (11.4) are changed
3. the integration step from t to $t + h$ is canceled, all states are reset to their values at time t, and the new equations (11.4) are integrated from t to $t + h$ with full accuracy
4. if needed, steps 2 and 3 are repeated until no change in $\mathbf{z}$ takes place. As the solver could be trapped in a limit cycle of discrete transitions, a maximum number of $\mathbf{z}$ changes at the same time is allowed. If that number is reached, the integration time step size is temporarily decreased from h to its minimum value h_{min} (see Section ???). If the limit cycle problem persists with the minimum step size, the simulation stops; the model should be adjusted with respect to the sequence of discrete transitions.

The solution scheme is presented in greater detail in:

D. Fabozzi, A. S. Chieh, P. Panciatici and T. Van Cutsem "On simplified handling of state events in time-domain simulation", Proc. of the 17th Power System Computation Conference (PSCC), 2011

It is illustrated graphically in Fig. 11.4, for a single state x and a single discrete variable z. The numbers refer to the above steps and times are shown in parentheses.

[3] solving them with less accuracy, to the purpose of gaining time, might lead to wrong identification of the discrete transitions

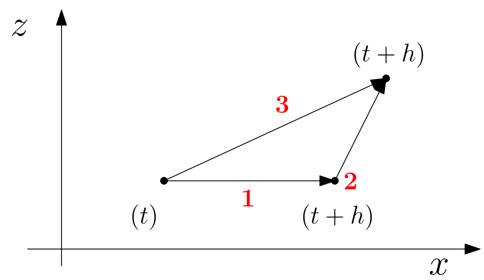


Figure 11.4: Graphical representation of the solution scheme to handle discrete transitions

11.3 Model assembly

In order for CODEGEN to generate the differential-algebraic equations of an arbitrarily complex user model, the latter is decomposed into a set of simple, interconnected *modelling blocks*. The latter correspond to time constants, integrators, PID controllers, non-linearities, etc. The library of available modelling blocks is documented in Chapter 12.

The EXC, TOR, INJ or TWOP model is thus handled as a set of interconnected modelling blocks, as illustrated in Fig. 11.5.

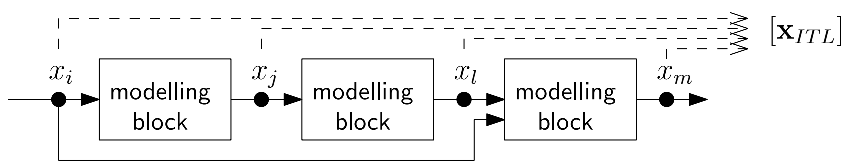


Figure 11.5: A user model made up of interconnected modelling blocks

Each block contributes with its algebraic and/or differential equations. Equations (11.4) are obtained by gathering the equations of all the blocks.

The blocks are interconnected through links. A distinct internal state (contributing to $\mathbf{x}_{ITL}$) is associated with each link. It can be differential or algebraic. Each of these states must be given by the user a unique name as well as an initial value.

Most of the modelling blocks also involve internal states (contributing to $\mathbf{x}_{ITL}$). Unlike the states associated with links between the blocks, the user does not need to name them, nor to initialize them; this is done automatically by the code generated by CODEGEN.

Finally, some modelling blocks involve discrete variables $\mathbf{z}$.

> ☛ Example. The modelling block `tf1p1z` is an example of block with one internal state. It implements the transfer function with one pole and one zero shown in Fig. 11.6.
>
> x_i is either an input or an internal state, while x_j is either an internal or an output state. The model requires three data: G, T_z and T_p.

$$x_i \longrightarrow \boxed{G\frac{1+sT_z}{1+sT_p}} \longrightarrow x_j$$

Figure 11.6: The `tf1p1z` modelling block

CODEGEN generates the following equations:

$$\dot{x_1} = G\,x_i - x_j \tag{11.11}$$
$$0 = T_p\,x_j - G\,T_z\,x_i - \,x_1 \tag{11.12}$$

where x_1 is an internal state. The latter is automatically initialized to :

$$x_1(0) = G\,(T_p - T_z)x_i(0)$$

which is obtained by setting the derivative of x_1 to zero in the above equations.

11.4 Syntax of the model description

A model is specified in a text file with the contents and structure shown in Fig. 11.7. The name of the file can be freely chosen.

The file is made up of six sections, which are detailed hereafter. All sections must be present and in the order shown in Fig. 11.7.

All keywords must be written exactly as specified in this documentation. In particular the upper/lowercase must be the same and no blank must be inserted or skipped.

11.4.1 Header

The header includes two lines in which:
<type of model> specifies the type of model. It can be `exc`, `tor`, `inj` or `twop`, as explained in Section 11.1.1
<name of model> specifies the name of the model. This is a string of at most 16 characters

If a model type `abc` and a model name `xyz` are specified, the complete name of the model will be `abc_xyz.f90`. This name has to be used in the data files (see Section ???). CODEGEN will correspondingly produce a file named `abc_xyz.f90` with the FORTRAN code of the model.

```
<type of model>
<name of model>
%data
<name of data 1>
<name of data 2>
    ⋮
%parameters
<name of parameter 1>=<mathematical expression 1>
<name of parameter 2>=<mathematical expression 2>
    ⋮
%states
<name of internal state 1>=<mathematical expression of initial value 1>
<name of internal state 2>=<mathematical expression of initial value 2>
    ⋮
%observables
<name of state, data or parameter 1>
<name of state, data or parameter 2>
    ⋮
%models
& <name of modelling block 1>
<name of state 1>
<name of state 2>
    ⋮
<name of data 1, name of parameter 1 or mathematical expression 1>
<name of data 2, name of parameter 2 or mathematical expression 2>
    ⋮
& <name of modelling block 2>
<name of state 1>
<name of state 2>
    ⋮
<name of data 1, name of parameter 1 or mathematical expression 1>
<name of data 2, name of parameter 2 or mathematical expression 2>
    ⋮
```

Figure 11.7: Syntax of model files

11.4.2 Data section

The data section starts with the keyword `%data`.

Table 11.2: Base powers that can be used in models (reserved names)

model type	name	meaning
exc	-	-
tor	-	-
inj	{sbase}	base power used for per unit values in the network (MVA)
twop	{sbase1}	base power used for per unit values in the subnetwork including bus 1 (MVA)
	{sbase2}	base power used for per unit values in the subnetwork including bus 2 (MVA)

Each data must be given a unique name. That name, enclosed with braces {}, is used in the rest of the model description.

The braces must not be used when a data is first defined, i.e. before the = symbol[4]. If used, the brace(s) will be treated as part of the data name, which most likely will lead to an error and the failure to compile the model.

Each name can be followed by a comment, on the same line, starting with an exclamation mark !.

At the initialization of a simulation, RAMSES maps the data present in the input file with those declared in the data section of the model. The data are read from the data file in the order specified in the data section.

The base power used for per unit values in the network is a data available by default in the `inj` and `twop` models, as detailed in Table 11.2. The names are enclosed with braces, as for other data. The names are reserved and must not be used for another data.

11.4.3 Parameter section

The parameter section starts with the keyword `%parameters`.

Each parameter must be given a unique name. That name, enclosed with braces { }, is used in the rest of the model description.

The braces must not be used when a parameter is first defined, i.e. before the = symbol[5]. If used, the brace(s) will be treated as part of the parameter name, which most likely will lead to an error and the failure to compile the model.

Each name can be followed by a comment, on the same line, starting with an exclamation mark !.

A data and a parameter cannot be given the same name.

[4] since CODEGEN expects to find a data, there is no ambiguity
[5] since CODEGEN expects to find a parameter, there is no ambiguity

At initialization of the simulation, each parameter is given the value specified by the mathematical expression after the = symbol. This expression usually involves data but it can also involve parameters which have been previously defined[6]. It may involve standard mathematical functions such as cos, $**$, $sqrt$, etc. The syntax is that of the FORTRAN language. It can be found for instance at:

https://www.intel.com/content/www/us/en/docs/fortran-compiler/developer-guide-reference/2023-0/categories-of-intrinsic-functions.html

Attention is drawn on the following syntactic items:

- the exponent is denoted with $**$, not ˆ (as with MATLAB)
- the operators in boolean expressions are denoted as follows: .lt. (smaller than), .le. (smaller or equal to), .gt. (greater than), .ge. (greater or equal to), .eq. (equal to), .ne. (not equal to).

11.4.4 State section

The state section starts with the keyword `%states`.

Each internal state must be given a unique name. That name, enclosed with brackets [ ], is used in most places of the model description.

The brackets must not be used when a state is first defined, i.e. before the = symbol[7]. If used, the bracket(s) will be treated as part of the state name, which most likely will lead to an error and the failure to compile the model.

Similarly, brackets must not be used when specifying the input and/or output states of a block.

Each state declaration can be followed by a comment, on the same line, starting with an exclamation mark !.

A state cannot have the same name as a data or a parameter.

Input and output states have their own, reserved names that cannot be used for internal states. They are listed in Table 11.3. All values are in pu on the bases detailed in Section 11.1.1. As for other states, their names must be enclosed with brackets.

Each internal state is initialized at the value specified by the mathematical expression after the = symbol. This expression may involve data, parameters or states (the initial values of those states, to be precise) that have been previously defined or are listed in Table 11.3. The mathematical expression may involve standard mathematical functions: see previous section.

[6]the names of those parameters must be enclosed in braces

[7]since CODEGEN expects to find a state, there is no ambiguity

Table 11.3: Reserved names that must not be used for internal states

model type	reserved names	meaning of state
exc	[v]	terminal voltage of machine
	[p]	active power produced by machine
	[q]	reactive power produced by machine
	[omega]	rotor speed of machine
	[if]	field current of machine
	[vf]	field voltage of machine
tor	[p]	mechanical power produced by turbine
	[omega]	rotor speed of machine
	[tm]	mechanical torque of turbine
inj	[vx]	x component of bus voltage
	[vy]	y component of bus voltage
	[omega]	angular speed of center of inertia
	[ix]	x component of current injected into network
	[iy]	y component of current injected into network
twop	[vx1]	x component of voltage at bus 1
	[vy1]	y component of voltage at bus 1
	[vx2]	x component of voltage at bus 2
	[vy2]	y component of voltage at bus 2
	[omega1]	angular speed of center of inertia of subsystem including bus 1
	[omega2]	angular speed of center of inertia of subsystem including bus 2
	[ix1]	x component of current injected into network at bus 1
	[iy1]	y component of current injected into network at bus 1
	[ix2]	x component of current injected into network at bus 2
	[iy2]	y component of current injected into network at bus 2

Only internal states are declared, input and output states are not. Let us recall that their initial values are known.

11.4.5 Observable section

The observable section starts with the keyword `%observables`.

Observables are quantities candidate to be plotted as functions of time at the end of the simulation. They are usually (input, output or internal) states but data or parameters are also allowed[8].

The data and the parameter names must not be enclosed with braces. Similarly, the state names must not be enclosed with brackets.

[8]this allows displaying the time evolution of a controller setpoint, for instance

11.4.6 Model section

The model section starts with the keyword `%models`.

The modelling blocks and their data are enumerated sequentially. The order does not matter. Each modelling block is identified by the & symbol, **followed by a space**, followed by the name of the block. The information required by each block is detailed in the next chapter.

Each name can be followed by a comment, on the same line, starting with an exclamation mark !.

The end of the section coincides with the end of the file.

11.4.7 An example

Here is the code of the simple excitation system shown in Fig. 11.3. The name of the model is "simple_avr". There are three observables: one parameter, one internal state and one output state

```
exc
simple_avr
%data
G                    ! gain of exciter
T                    ! time constant of the exciter
vfmin                ! lower fielf voltage
vfmax                ! upper field voltage
%parameters
Vo = [v]+ [vf]/{G}   ! voltage setpoint of AVR
%states
dv = [vf]/{G}        ! voltage error
%observables
Vo
dv
vf
%models
& algeq              ! calculation of voltage error
{Vo}-[v]-[dv]
& tf1plim            ! exciter transfer function
dv
vf
{G}
{T}
{vfmin}
```

```
{vfmax}
```

Here is the trace of execution, showing the counts of equations and states, respectively, as the modelling blocks are assembled.

```
           WELCOME TO CODEGEN    v5
       the model generator of STEPSS

Input file with model description: simple_avr.txt

MODEL NAME : exc_simple_avr

Processing data...
     prm(  1)=  G                   ! gain of exciter
     prm(  2)=  T                   ! time constant of the exciter
     prm(  3)=  vfmin               ! lower fielf voltage
     prm(  4)=  vfmax               ! upper field voltage

Processing parameters...
     prm(  5)=  Vo   voltage setpoint of AVR

Processing states...

   Output states
     x(  1)=  vf           field voltage
   Internal states defined by user
     x(  2)=  dv                     voltage error

Processing observables...
     Vo
     dv
     vf

Number of user-defined and output d/a states :   2

Processing models...
     & algeq             ! calculation of voltage error
                                          1 d/a eqs     2 d/a states     0 disc states
     & tf1plim           ! exciter transfer function
                                          2 d/a eqs     2 d/a states     1 disc states

Merging temporary files...
com.tmp
head.tmp
obs.tmp
init.tmp
evalf.tmp
updz.tmp
```

11.4.8 What about time ?

Time is neither a data, nor a parameter, nor a state. Although this is infrequent, time may appear explicitely in some models. It is denoted `t` (without brackets).

11.5 Error detection

CODEGEN performs some "sanity checks" to detect mistakes such as:

- unbalanced braces or brackets
- missing keyword %data, %parameters, %states, %observables or %models
- multiply defined data, states or parameters
- usage of a reserved name for an internal state
- typo leading to an unknown name of state or modelling block
- ouput variable not appearing in any equation of the model
- number of states different from number of equations (differential or algebraic).

However, not all mistakes are flagged by CODEGEN. Typos or syntax errors may not be detected, leading to error messages by the compiler when the .f90 file is compiled. Some examples are:

- typos in the name of a mathematical function. For instance, "cus([delta])" instead of "cos([delta])", exponent denoted by ˆ instead of **
- forgotten braces { } enclosing the name of a data or a parameter
- forgotten brackets [ ] enclosing the name of a state.

11.6 Data format

11.6.1 Injectors

The data of an injector are defined in the Data section of the user-defined model, see Section 11.4.2. The corresponding values are to be provided in an `INJEC` record with the following syntax:

```
INJEC MODEL_NAME INJ_NAME BUS FP FQ P Q DATA1 DATA2 ...;
```

where:

- MODEL_NAME is the name of injector model, as defined in the header of the model description, see Section 11.4.1. This is a string of at most 20 characters
- INJ_NAME is the name of the injector (a particular instance of the model). This is a string of at most 20 characters
- BUS is the name of the bus to which the injector is connected. This is a string of at most 8 characters
- FP is the fraction f_{Pj} defined by Eq. (6.8)
- FQ is the fraction f_{Qj} defined by Eq. (6.8)
- P is the initial active power defined by Eq. (6.7). Let us recall that FP and P must obey Eq. (6.10)
- Q is the initial reactive power defined by Eq. 6.7. Let us recall that FQ and Q must obey Eq. 6.10
- DATA1 DATA2 ... are the successive values of the data, as defined in the Data section of the user-defined model. In particular they appear in the order defined in that section.

11.6.2 Two-ports

The data of a two-port are defined in the Data section of the user-defined model, see Section 11.4.2. The corresponding values are to be provided in an TWOP record with the following syntax:

```
TWOP MODEL_NAME TWOP_NAME BUS1 BUS2 IND FP1 FQ1 P2 Q2 DATA1 DATA2 ...;
```

where:

- MODEL_NAME is the name of two-port model, as defined in the header of the model description, see Section 11.4.1. This is a string of at most 20 characters
- INJ_NAME is the name of the two-port (a particular instance of the model). This is a string of at most 20 characters
- BUS1 is the name of the first bus to which the two-port is connected. This is a string of at most 8 characters
- BUS2 is the name of the second bus to which the two-port is connected. This is a string of at most 8 characters
- IND is a "synchronization" indicator:
  - if it is set to "S", the two-port causes the subnetworks of BUS1 and BUS2 to be "synchronous", i.e. to operate at the same frequency, as for an AC transmission line, for instance
  - if it is set to "A", the two sub-networks are left "asynchronous", i.e. they operate at different frequencies, as for an HDVC link, for instance.

  Other values of IND are incorrect.

- `FP1` is the fraction f_{Pj} defined by Eq. (6.8), relative to `BUS1`
- `FQ1` is the fraction f_{Qj} defined by Eq. (6.8), relative to `BUS1`
- `P1` is the initial active power defined by Eq. (6.7) relative to `BUS1`. Let us recall that `FP1` and `P1` must obey Eq. (6.10)
- `Q1` is the initial reactive power defined by Eq. 6.7 relative to `BUS1`. Let us recall that `FQ1` and `Q1` must obey Eq. 6.10
- `FP2` is the fraction f_{Pj} defined by Eq. (6.8), relative to `BUS2`
- `FQ2` is the fraction f_{Qj} defined by Eq. (6.8), relative to `BUS2`
- `P2` is the initial active power defined by Eq. (6.7) relative to `BUS2`. Let us recall that `FP2` and `P2` must obey Eq. (6.10)
- `Q2` is the initial reactive power defined by Eq. 6.7 relative to `BUS2`. Let us recall that `FQ2` and `Q2` must obey Eq. 6.10
- `DATA1 DATA2 ...` are the successive values of the data, as defined in the Data section of the user-defined model. In particular they appear in the order defined in that section.

Chapter 12

Library of modelling blocks

12.1 List of modelling blocks

The list of modelling blocks is given in the table below, together with a brief description.

Table 12.1: List of modelling blocks

abs	Absolute value of input
algeq	Algebraic equation
db	Deadband
f_inj	Estimate of frequency at bus of injector
ftwop_bus1	Estimate of frequency at first bus of two-port
ftwop_bus2	Estimate of frequency at second bus of two-port
fsa	Finite State Automaton
hyst	Hysteresis
int	Integrator with time constant
inlim	Integrator with time constant and non-windup limits on output
invlim	Integrator with time constant and non-windup variable limits on output
lim	Limiter with constant bounds
limvb	Limiter with variable bounds
max1v1c	Maximum between a state and a constant
max2v	Maximum between two states
min1v1c	Minimum between a state and a constant
min2v	Minimum between two states
nint	Integer nearest to the input shifted by a constant
pictl	Proportional-Integral (PI) controller
pictllim	Proportional-Integral (PI) controller with non-windup limit on integral term
pictl2lim	PI controller with non-windup limit on integral term and limit on proportional term
pictlieee	PI controller with non-windup limit on integral term, compliant with IEEE standards
pwlin	Piece-wise linear function of input
switch	Set output to one among n inputs, based on value of a controlling state
swsign	Switch between two input states, based on sign of a third input state
tf1p	Transfer function between input and output: one time constant
tf1plim	same as tf1p with non-windup limits on output
tf1pvlim	same as tf1p with variable non-windup limits on output
tf1p2lim	same as tf1p with limits on rate of change of output and non-windup limits on output
tfder1p	Transfer function: derivative with one time constant
tf1p1z	Transfer function between input and output: one zero and one pole
tf2p2z	Transfer function between input and output: two real zeros and two real poles
timer	Timer with delay varying piecewise linearly with monitored variable
timersc	Timer with delay varying as a staircase function of monitored variable
tsa	Two-state automaton with transitions based on signs of two inputs

12.2 Information provided for each block

Most blocks have input and output states. **The names of those states must not be enclosed with backets**[1]. If used, the brackets will be treated as part of the state name, which most likely will to a wrong reference and the failure to compile the model.

Most (but not all) blocks require parameters. Each of them is either a data (enclosed with braces) or a parameter in the sense defined in the previous chapter (also enclosed with braces) or a mathematical expression involving data and/or parameters.

☛ Here are three examples of information that can be specified for a time constant:

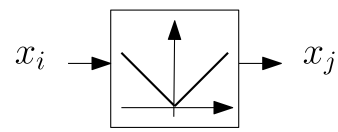
2.
{T}
1/{omegac}

12.3 Library

[1] since CODEGEN expects to find a state name, there is no ambiguity

abs

Absolute value of input.



$$x_j = |x_i|$$

Syntax : & abs
name of state x_i
name of state x_j

Internal states : none

Discrete variable : $z \in \{-1, 1\}$

Equations :

$$0 = \begin{cases} x_j - x_i & \text{if } z = 1 \\ x_j + x_i & \text{if } z = -1 \end{cases}$$

Discrete transitions :

if $z = 1$ **then**
  if $x_i < 0$ **then**
    $z \leftarrow -1$
  end if
else
  if $x_i > 0$ **then**
    $z \leftarrow 1$
  end if
end if

Initialization of discrete variables:

if $x_i > 0$ **then**
  $z \leftarrow 1$
else
  $z \leftarrow 0$
end if

The model having a discontinuous derivative at $x_i = 0$, it is implemented internally with a small hysteresis; see model **hyst** with $x_I = \epsilon, y_{IB} = -\epsilon, y_{IA} = \epsilon, x_D = -\epsilon, y_{DB} = -\epsilon, y_{DA} = \epsilon$, where ϵ is the absolute accuracy used to solve the algebraic equations in RAMSES.

algeq

algebraic equation

Syntax : & algeq
math expression

Internal states : none

Discrete variables : none

This block forces an algebraic constraint (or equation) involving one of several states :

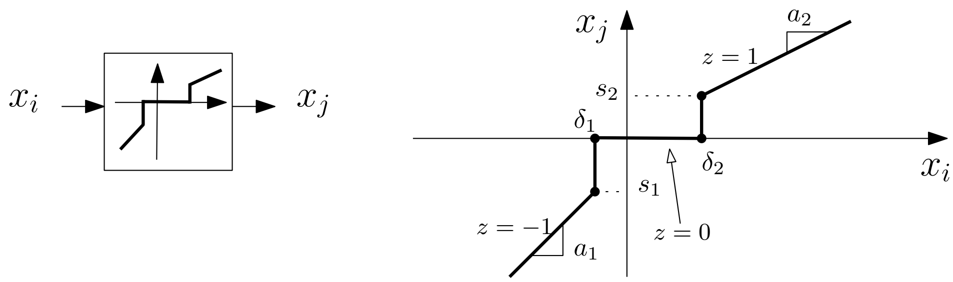
$$f(x_1, x_2, \ldots, x_n) = 0$$

where n is the number of states $(n \geq 1)$

Note that this blocks does not really have “inputs” and “outputs”. The latter stem from the rest of the model involving the algebraic constraint.

db

Deadband.



Syntax : & db
name of variable x_i
name of variable x_j
data name, parameter name or math expression for δ_1
data name, parameter name or math expression for s_1
data name, parameter name or math expression for a_1
data name, parameter name or math expression for δ_2
data name, parameter name or math expression for s_2
data name, parameter name or math expression for a_2

Internal states : none

Discrete variables : $z \in \{0, 1, -1\}$

Equations :

$$0 = \begin{cases} x_j & \text{if } z = 0 \\ x_j - s_2 - a_2(x_i - \delta_2) & \text{if } z = 1 \\ x_j - s_1 - a_1(x_i - \delta_1) & \text{if } z = -1 \end{cases}$$

Discrete transitions :

if $z \in \{0, 1\}$ **then**
  if $x_i < \delta_1$ **then**
    $z \leftarrow -1$
  end if
else if $z \in \{-1, 0\}$ **then**
  if $x_i > \delta_2$ **then**
    $z \leftarrow 1$
  end if
else if $z \in \{-1, 1\}$ **then**
  if $\delta_1 < x_i < \delta_2$ **then**
    $z \leftarrow 0$
  end if
end if

Initialization of discrete variables :

if $x_i > \delta_2$ **then**
  $z \leftarrow 1$
else if $x_i < \delta_1$ **then**
  $z \leftarrow -1$
else
  $z \leftarrow 0$
end if

The data must obey $\delta_1 < \delta_2$, $a_1 \geq 0$ and $a_2 \geq 0$ (see for instance the diagram above).

A particular case is $s_1 = s_2 = 0$ and $a_1 = a_2 = 1$ (but all values are allowed for these four parameters).

The pwlin4, pwlin5 or pwlin6 block can be used as an alternative to this block.

f_inj

Computes f, an estimate of the frequency (in per unit) at a given bus, from the evolution of the rectangular components v_x and v_y of the bus voltage. A measurement time constant T is involved. Can be used in the model of an injector only.

Syntax : & f_inj
name of variable f
data name, parameter name or math expression for T

Internal states : v_{xm} and v_{ym}.

Discrete variables : none

Equations :

$$\dot{v}_{xm} = \frac{v_x - v_{xm}}{T} \quad (12.1)$$

$$\dot{v}_{ym} = \frac{v_y - v_{ym}}{T} \quad (12.2)$$

$$0 = \omega_{ref,pu} + \frac{(v_y - v_{ym})v_{xm} - (v_x - v_{xm})v_{ym}}{2\pi f_N T(v_{xm}^2 + v_{ym}^2)} - f \quad (12.3)$$

Initialization of internal states : $v_{xm} = v_x$ and $v_{ym} = v_y$.

Explanation of model

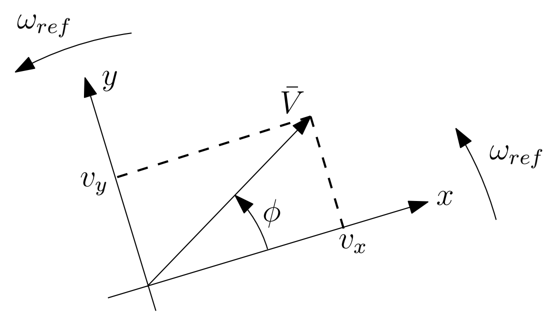
The model uses v_x and v_y as inputs but these variables are automatically inherited and must not be declared. Only the name given to the estimated frequency and the time constant T must be provided.

As shown in the figure below, v_x and v_y are the projections of the bus voltage phasor on the references axes x and y rotating at the angular speed ω_{ref} (rad/s). The latter is known from the settings of the simulation. The angular frequency of the corresponding rotating vector is given by :

$$\omega = \omega_{ref} + \frac{d\phi}{dt}$$

with:

$$\phi = \arctan \frac{v_y}{v_x}$$



The frequency in per unit is given by :

$$f = \frac{\omega}{2\pi f_N} = \frac{\omega_{ref}}{2\pi f_N} + \frac{1}{2\pi f_N}\frac{d\phi}{dt} = \omega_{ref,pu} + \frac{1}{2\pi f_N}\frac{d}{dt}\left(\arctan\frac{v_y}{v_x}\right)$$

where f_N is the nominal frequency (known from the system data).

By developing the last term, it is easily found that:

$$f = \omega_{ref,pu} + \frac{1}{2\pi f_N}\frac{\dot{v_y}v_x - \dot{v_x}v_y}{v_x^2 + v_y^2} \tag{12.4}$$

To filter the transients affecting v_x and v_y, a measurement device with a time constant T is simulated. The "measured" values, denoted v_{xm} and v_{ym}, evolve according to (12.1, 12.2). These measured values and their derivatives are then used in (12.4), which becomes:

$$f = \omega_{ref,pu} + \frac{1}{2\pi f_N}\frac{\dot{v}_{ym}v_{xm} - \dot{v}_{xm}v_{ym}}{v_{xm}^2 + v_{ym}^2}$$

Replacing $\dot{v}_{xm}$ and $\dot{v}_{ym}$ by their expressions (12.1,12.2) yields Eq. (12.3).

A recommended value for T is in the order to $0.05 - 0.10$ s. A zero value for T is not allowed. If too small a value is specified for T, the solver may encounter a singularity and the simulation may not proceed.

f_twop_bus1

Similar to f_inj, computes f, an estimate of the frequency (in per unit) at the first bus of a given two-port. Can be used in the model of a two-port only.

Syntax :	& f_twop_bus1
	name of variable f
	data name, parameter name or math expression for T

Please refer to f_inj.

f_twop_bus2

Similar to f_inj, computes f, an estimate of the frequency (in per unit) at the second bus of a given two-port. Can be used in the model of a two-port only.

Syntax : & f_twop_bus2
name of variable f
data name, parameter name or math expression for T

Please refer to f_inj.

fsa

Finite State Automaton

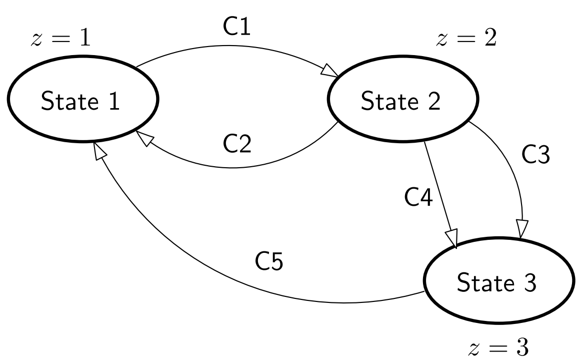
This block forces a set of n algebraic equations. There are s possible sets, each of them corresponding to a value of the discrete state z. The change form one set to another takes place when boolean expressions are true.

Syntax : see example below

Internal states : none

Discrete variables: z, the number of the state currently active

Example. Consider the example below with three states ($s = 3$). $n = 2$ is assumed.



& fsa
initial state of the system
1
algebraic constraint No. 1
algebraic constraint No. 2
−> 2
boolean expression C1
2
algebraic constraint No. 3
algebraic constraint No. 4
−> 1
boolean expression C2
−> 3
boolean expression C3
−> 3
boolean expression C4
3
algebraic constraint No. 5
algebraic constraint No. 6
−> 1
boolean expression C5
##

The # symbol indicates the start of the section relative to a state. It is followed by the number of the state. The states must be numbered consecutively from 1 to s, and they must be listed by increasing order. The ## symbol indicates the end of the list of states.

The initial state is specified as an integer in the second line.

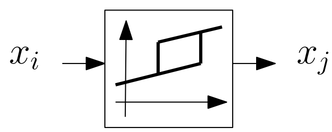
The number n of algebraic constraints must be the same in all states.

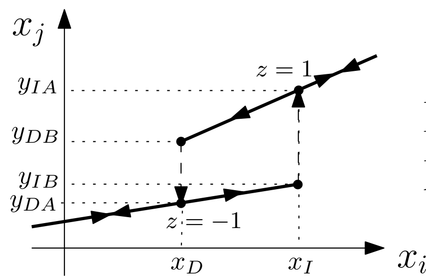
The $->$ symbol indicates a transition. It is followed by the number of the state reached after the transition has taken place. The next line is the corresponding boolean expression, involving states and possibly parameters.

In the example above, there are two possible transitions from State 2 to State 3. Alternatively a single transition can be specified with the combined condition “C3 or C4”.

hyst

Hysteresis





D stands for "Decreasing"
I stands for "Increasing"
B stands for "Before jumping"
A stands for "After jumping"

Syntax :	& hyst
	name of variable x_i
	name of variable x_j
	data name, parameter name or math expression for x_I
	data name, parameter name or math expression for y_{IB}
	data name, parameter name or math expression for y_{IA}
	data name, parameter name or math expression for x_D
	data name, parameter name or math expression for y_{DB}
	data name, parameter name or math expression for y_{DA}
	data name, parameter name or math expression for $z0$

Internal states : none

Discrete variables : $z \in \{-1, 1\}$

Equations :

$$0 = \begin{cases} x_j - y_{IA} - \dfrac{y_{IA} - y_{DB}}{x_I - x_D}(x_i - x_I) & \text{if } z = 1 \\ x_j - y_{DA} - \dfrac{y_{IB} - y_{DA}}{x_I - x_D}(x_i - x_D) & \text{if } z = -1 \end{cases}$$

Discrete transitions :

if $z = -1$ **then**
  if $x_i > x_I$ **then**
    $z \leftarrow 1$
  end if
else
  if $x_i < x_D$ **then**
    $z \leftarrow -1$
  end if
end if

Initialization of discrete variables :

if $x_i > x_I$ **then**
  $z \leftarrow 1$
else if $x_i < x_D$ **then**
  $z \leftarrow -1$
else
  if $z0 \geq 0$ **then**
    $z \leftarrow 1$
  else
    $z \leftarrow 0$
  end if
end if

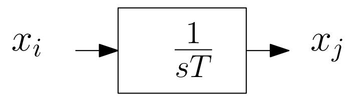
At $t = 0$, if $x_D < x_i(0) < x_I$ the initial state of the system is indeterminate, since it could operate on the (y_{IA}, y_{DB}) line (i.e. with an initial value of z equal to 1) as well as on the (y_{DA}, y_{IB}) line (i.e. with an initial value of z equal to -1). Hence, the user must specify $z0$, the initial value of z.

If $x_i(0) < x_D$ (resp. $x_i(0) > x_I$) the initial value is $z = -1$ (resp. $z = 1$) and $z0$ is not used.

The data must obey $x_D < x_I$, otherwise the model would not correspond to hysteresis. The case $x_D = x_I$ is not allowed either, but it can be handled with the **pwlin4** model.

int

Integrator with (positive) time constant T



Syntax : & int
name of variable x_i
name of variable x_j
data name, parameter name or math expression for T

Internal states : none

Discrete variables : none

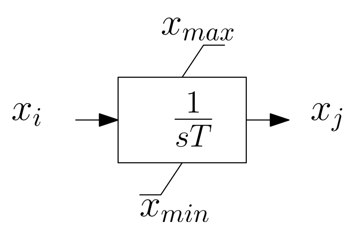
Equations :

$$T\dot{x}_j = x_i \tag{12.5}$$

A zero value for T is not allowed. If too small a value is specified for T, the solver may encounter a singularity and the simulation may not proceed.

inlim

Integrator with (positive) time constant T and non-windup limits on output



Syntax : & inlim
name of variable x_i
name of variable x_j
data name, parameter name or math expression for T
data name, parameter name or math expression for x_{min}
data name, parameter name or math expression for x_{max}

Internal states : none

Discrete variables : $z \in \{0, 1, -1\}$

Equations :

$$\begin{aligned} T\dot{x}_j &= x_i && \text{if } z = 0 \\ 0 &= x_j - x_{min} && \text{if } z = -1 \\ 0 &= x_j - x_{max} && \text{if } z = 1 \end{aligned} \tag{12.6}$$

Discrete transitions :

if $z = 0$ **then**
  if $x_j > x_{max}$ **then**
    $z \leftarrow 1$
  else if $x_j < x_{min}$ **then**
    $z \leftarrow -1$
  end if
else if $z = 1$ **then**
  if $x_i < 0$ **then**
    $z \leftarrow 0$
  end if
else if $z = -1$ **then**
  if $x_i > 0$ **then**
    $z \leftarrow 0$
  end if
end if

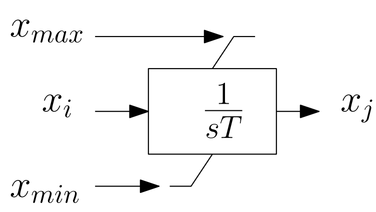
Initialization of discrete variables :

if $x_j > x_{max}$ **then**
  $z \leftarrow 1$
else if $x_j < x_{min}$ **then**
  $z \leftarrow -1$
else
  $z \leftarrow 0$
end if

A zero value for T is not allowed. If too small a value is specified for T, the solver may encounter a singularity and the simulation may not proceed.

invlim

Integrator with (positive) time constant T and non-windup limits on output. The lower and upper limits are variables.



Syntax : & invlim
name of variable x_i
name of variable x_{min}
name of variable x_{max}
name of variable x_j
data name, parameter name or math expression for T

Internal states : none

Discrete variables : $z \in \{0, 1, -1\}$

Equations :

$$\begin{aligned} T\dot{x}_j &= x_i && \text{if } z = 0 \\ 0 &= x_j - x_{min} && \text{if } z = -1 \\ 0 &= x_j - x_{max} && \text{if } z = 1 \end{aligned} \tag{12.7}$$

Discrete transitions :

if $z = 0$ **then**
  if $x_j > x_{max}$ **then**
    $z \leftarrow 1$
  else if $x_j < x_{min}$ **then**
    $z \leftarrow -1$
  end if
else if $z = 1$ **then**
  if $x_i < 0$ **then**
    $z \leftarrow 0$
  end if
else if $z = -1$ **then**
  if $x_i > 0$ **then**
    $z \leftarrow 0$
  end if
end if

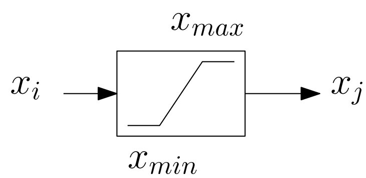
Initialization of discrete variables :

if $x_j > x_{max}$ **then**
  $z \leftarrow 1$
else if $x_j < x_{min}$ **then**
  $z \leftarrow -1$
else
  $z \leftarrow 0$
end if

A zero value for T is not allowed. If too small a value is specified for T, the solver may encounter a singularity and the simulation may not proceed.

lim

Limiter with constant bounds



Syntax : & lim
name of variable x_i
name of variable x_j
data name, parameter name or math expression for x_{min}
data name, parameter name or math expression for x_{max}

Internal states : none

Discrete variables : $z \in \{-1, 0, 1\}$

Equations :

$$0 = \begin{cases} x_j - x_i & \text{if } z = 0 \\ x_j - x_{max} & \text{if } z = 1 \\ x_j - x_{min} & \text{if } z = -1 \end{cases}$$

Discrete transitions :

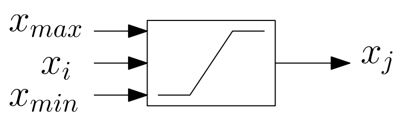
if $z = 0$ **then**
  if $x_i > x_{max}$ **then**
    $z \leftarrow 1$
  else if $x_i < x_{min}$ **then**
    $z \leftarrow -1$
  end if
else if $z = 1$ **then**
  if $x_i < x_{max}$ **then**
    $z \leftarrow 0$
  end if
else if $z = -1$ **then**
  if $x_i > x_{min}$ **then**
    $z \leftarrow 0$
  end if
end if

Initialization of discrete variables :

if $x_j > x_{max}$ **then**
  $z \leftarrow 1$
else if $x_j < x_{min}$ **then**
  $z \leftarrow -1$
else
  $z \leftarrow 0$
end if

limvb

Limiter with variable bounds



Syntax :	& limvb
	name of variable x_i
	name of variable x_{min}
	name of variable x_{max}
	name of variable x_j

Internal states : none

Discrete variables : $z \in \{-1, 0, 1\}$

Equations :

$$0 = \begin{cases} x_j - x_i & \text{if } z = 0 \\ x_j - x_{max} & \text{if } z = 1 \\ x_j - x_{min} & \text{if } z = -1 \end{cases}$$

Discrete transitions :

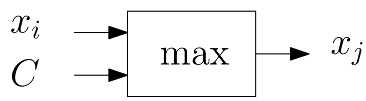
```
if z = 0 then
  if x_i > x_max then
    z ← 1
  else if x_i < x_min then
    z ← −1
  end if
else if z = 1 then
  if x_i < x_max then
    z ← 0
  end if
else if z = −1 then
  if x_i > x_min then
    z ← 0
  end if
end if
```

Initialization of discrete variables :

```
if x_i > x_max then
  z ← 1
else if x_i < x_min then
  z ← −1
else
  z ← 0
end if
```

max1v1c

Maximum between a state and a constant



Syntax : & max1v1c
name of variable x_i
name of variable x_j
data name, parameter name or math expression for C

Internal states : none

Discrete variables : $z \in \{1, 2\}$

Equations :

$$0 = \begin{cases} x_j - C & \text{if } z = 1 \\ x_j - x_i & \text{if } z = 2 \end{cases}$$

Discrete transitions :

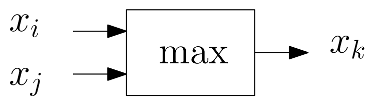
if $z = 1$ **then**
  if $x_i > C$ **then**
    $z \leftarrow 2$
  end if
else if $z = 2$ **then**
  if $x_i <= C$ **then**
    $z \leftarrow 1$
  end if
end if

Initialization of discrete variables :

if $x_i < C$ **then**
  $z \leftarrow 1$
else
  $z \leftarrow 2$
end if

max2v

Maximum between two states



Syntax : & max2v
name of variable x_i
name of variable x_j
name of variable x_k

Internal states : none

Discrete variables : $z \in \{1, 2\}$

Equations :

$$0 = \begin{cases} x_i - x_k & \text{if } z = 1 \\ x_j - x_k & \text{if } z = 2 \end{cases}$$

Discrete transitions :

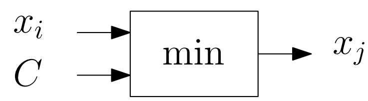
if $z = 1$ **then**
  if $x_i < x_j$ **then**
    $z \leftarrow 2$
  end if
else if $z = 2$ **then**
  if $x_j \leq x_i$ **then**
    $z \leftarrow 1$
  end if
end if

Initialization of discrete variables :

if $x_i > x_j$ **then**
  $z \leftarrow 1$
else
  $z \leftarrow 2$
end if

min1v1c

Minimum between a state and a constant



Syntax : & min1v1c
name of variable x_i
name of variable x_j
data name, parameter name or math expression for C

Internal states : none

Discrete variables : $z \in \{1, 2\}$

Equations :

$$0 = \begin{cases} x_j - x_i & \text{if } z = 1 \\ x_j - C & \text{if } z = 2 \end{cases}$$

Discrete transitions :

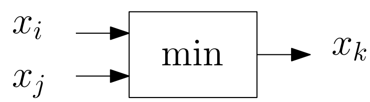
if $z = 1$ **then**
  if $x_i > C$ **then**
    $z \leftarrow 2$
  end if
else if $z = 2$ **then**
  if $x_i <= C$ **then**
    $z \leftarrow 1$
  end if
end if

Initialization of discrete variables :

if $x_i < C$ **then**
  $z \leftarrow 1$
else
  $z \leftarrow 2$
end if

min2v

Minimum between two states



Syntax : & min2v
name of variable x_i
name of variable x_j
name of variable x_k

Internal states : none

Discrete variables : $z \in \{1, 2\}$

Equations :

$$0 = \begin{cases} x_i - x_k & \text{if } \; z = 1 \\ x_j - x_k & \text{if } \; z = 2 \end{cases}$$

Discrete transitions :

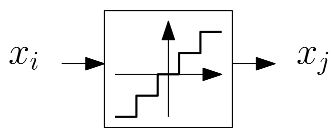
if $z = 1$ **then**
  if $x_i > x_j$ **then**
    $z \leftarrow 2$
  end if
else if $z = 2$ **then**
  if $x_j \geq x_i$ **then**
    $z \leftarrow 1$
  end if
end if

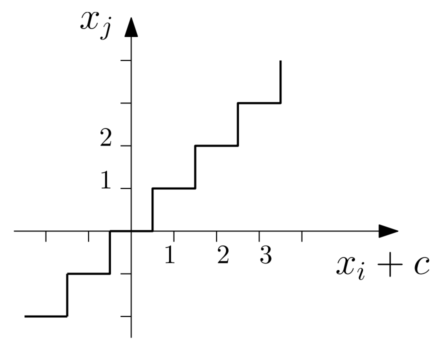
Initialization of discrete variables :

if $x_i < x_j$ **then**
  $z \leftarrow 1$
else
  $z \leftarrow 2$
end if

nint

Integer nearest to the input shifted by a constant c.





Syntax :	& nint
	name of variable x_i
	name of variable x_j
	data name, parameter name or math expression for c

Internal states : none

Discrete variables : $z \in \mathcal{I}$

Equations :

$$0 = x_j - z$$

Discrete transitions :

if nint$(x_i + c) \neq z$ **then**
  $z \leftarrow$ nint$(x_i + c)$
end if

where the nint function returns the nearest integer.

Particular cases:

- with $c = 0$, the output is the integer nearest to x_i;
- with $c = -0.5$, the output is the “floor” integer of x_i, i.e. the largest integer smaller or equal to x_i;

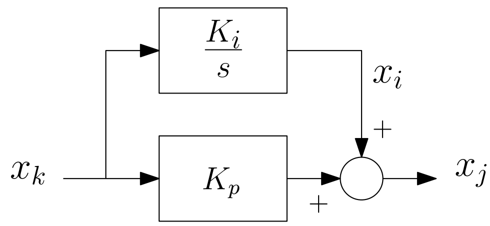
- with $c = 0.5$, the output is the "ceiling" integer of x_i, i.e. the smallest integer larger or equal to x_i.

Initialization of discrete variables :

$$z \leftarrow \mathsf{nint}(x_i + c)$$

pictl

Proportional-Integral (PI) controller



Syntax : & pictl
name of variable x_k
name of variable x_j
data name, parameter name or math expression for K_i
data name, parameter name or math expression for K_p

Internal states : x_i

Discrete variables: none

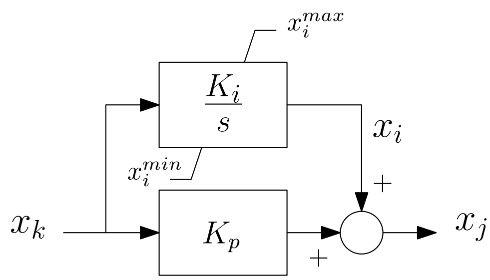
Equations :

$$\begin{aligned} \dot{x}_i &= K_i\, x_k \\ 0 &= K_p\, x_k + x_i - x_j \end{aligned}$$

Initialization of internal states: $x_i = x_j$

pictllim

Proportional-Integral (PI) controller with non-windup limit on the integral term.



Syntax : & pictllim
name of variable x_k
name of variable x_j
data name, parameter name or math expression for K_i
data name, parameter name or math expression for K_p
data name, parameter name or math expression for x_i^{min}
data name, parameter name or math expression for x_i^{max}

Internal states: x_i

Discrete variables : $z \in \{-1, 0, 1\}$

Equations :

$$\begin{cases} \dot{x}_i &= K_i x_k \qquad \text{if } z = 0 \\ 0 &= x_i - x_i^{min} \qquad \text{if } z = -1 \\ 0 &= x_i - x_i^{max} \qquad \text{if } z = 1 \end{cases}$$

$$0 = K_p x_k + x_i - x_j$$

Discrete transitions :

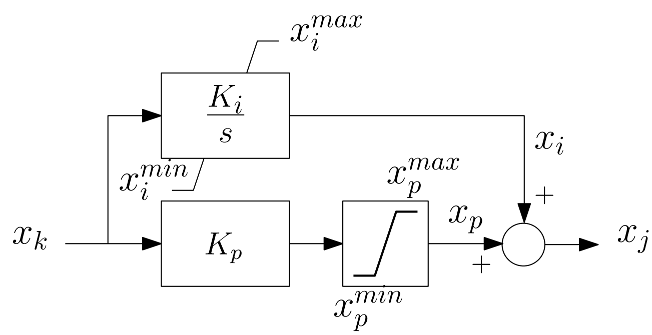
if $z = 0$ **then**
  if $x_i > x_i^{max}$ **then**
    $z \leftarrow 1$
  else if $x_i < x_i^{min}$ **then**
    $z \leftarrow -1$
  end if
else if $z = 1$ **then**
  if $K_i x_k < 0$ **then**
    $z \leftarrow 0$
  end if
else if $z = -1$ **then**
  if $K_i x_k > 0$ **then**
    $z \leftarrow 0$
  end if
end if

Initialization of the internal state and the discrete variable:

if $K_i x_k > 0$ **then**
  $z \leftarrow 1$
  $x_i \leftarrow x_i^{max}$
else if $K_i x_k < 0$ **then**
  $z \leftarrow -1$
  $x_i \leftarrow x_i^{min}$
else
  $z \leftarrow 0$
  $x_i \leftarrow x_j$
end if

pictl2lim

Proportional-Integral (PI) controller with non-windup limit on the integral term and limit on the proportional term.



Syntax : & pictl2lim
name of variable x_k
name of variable x_j
data name, parameter name or math expression for K_i
data name, parameter name or math expression for K_p
data name, parameter name or math expression for x_i^{min}
data name, parameter name or math expression for x_i^{max}
data name, parameter name or math expression for x_p^{min}
data name, parameter name or math expression for x_p^{max}

Internal states : x_i and x_p

Discrete variables : $z_1 \in \{-1, 0, 1\}$ and $z_2 \in \{-1, 0, 1\}$

Equations :

$$\begin{cases} 0 &= K_p x_k - x_p \qquad \text{if } z_1 = 0 \\ 0 &= x_p - x_p^{min} \qquad \text{if } z_1 = -1 \\ 0 &= x_p - x_p^{max} \qquad \text{if } z_1 = 1 \end{cases}$$

$$\begin{cases} \dot{x}_i &= K_i x_k \qquad \text{if } z_2 = 0 \\ 0 &= x_i - x_i^{min} \qquad \text{if } z_2 = -1 \\ 0 &= x_i - x_i^{max} \qquad \text{if } z_2 = 1 \end{cases}$$

$$0 = x_p + x_i - x_j$$

Discrete transitions :

if $z_1 = 0$ **then**
  if $x_p > x_p^{max}$ **then**
    $z_1 \leftarrow 1$
  else if $x_p < x_p^{min}$ **then**
    $z_1 \leftarrow -1$
  end if
else if $z_1 = 1$ **then**
  if $K_p x_k < x_p^{max}$ **then**
    $z_1 \leftarrow 0$
  end if
else if $z_1 = -1$ **then**
  if $K_p x_k > x_p^{min}$ **then**
    $z_1 \leftarrow 0$
  end if
end if

if $z_2 = 0$ **then**
  if $x_i > x_i^{max}$ **then**
    $z_2 \leftarrow 1$
  else if $x_i < x_i^{min}$ **then**
    $z_2 \leftarrow -1$
  end if
else if $z_2 = 1$ **then**
  if $K_i x_k < 0$ **then**
    $z_2 \leftarrow 0$
  end if
else if $z_2 = -1$ **then**
  if $K_i x_k > 0$ **then**
    $z_2 \leftarrow 0$
  end if
end if

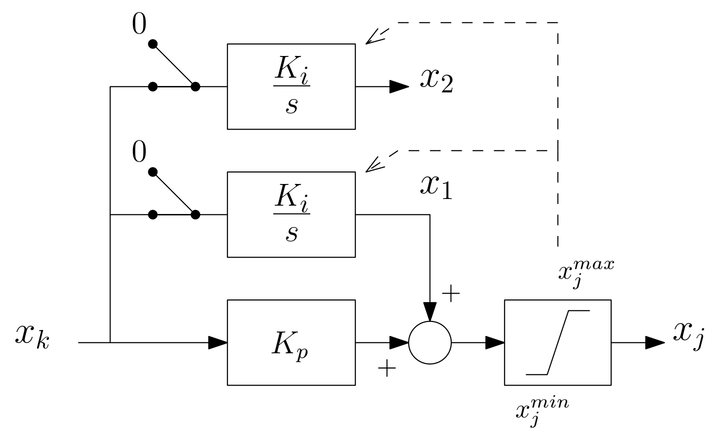
Initialization of the internal state x_p: $x_p = \min\left(x_p^{max}, \max(x_p^{min}, K_p x_k)\right)$

Initialization of internal state x_i and the discrete variables:

if $K_p x_k > x_p^{max}$ **then**
  $z_1 \leftarrow 1$
else if $K_p x_k < x_p^{min}$ **then**
  $z_1 \leftarrow -1$
else
  $z_1 \leftarrow 0$
end if
if $K_i x_k > 0$ **then**
  $z_2 \leftarrow 1$
  $x_i \leftarrow x_i^{max}$
else if $K_i x_k < 0$ **then**
  $z_2 \leftarrow -1$
  $x_i \leftarrow x_i^{min}$
else
  $z_2 \leftarrow 0$
  $x_i \leftarrow x_j - x_p$
end if

pictlieee

Proportional-Integral (PI) controller with non-windup limit on the integral term, compliant with IEEE standards.



This PI controller involves a limiter and a non-windup integrator compliant with the IEEE specifications in:

- IEEE recommended practice for excitation system models for power system stability studies, IEEE Std 421.5-1992
- IEEE recommended practice for excitation system models for power system stability studies, IEEE Std 421.5-2005 (Revision of IEEE Std 421.5-1992)
- IEEE recommended practice for excitation system models for power system stability studies, IEEE Std 421.5-2016 (Revision of IEEE Std 421.5-2005)

The IEEE standard specfies that the x_1 variable is frozen as soon as x_j reaches its (lower or upper) limit. This is done by merely setting the input of the integrator to zero.

However, the standard does not specify the condition under which x_1 is let to vary again, i.e. when the integrator is put back into service. A simple way would be to re-activate the integrator as soon x_j gets back within its limits. However, at that time instant, the x_k variable may have a value such that $x_j = K_p x_k + x_1$ is pushed again to its (lower or upper) limit. The system would then be trapped into a limit cycle (which would not match the behaviour of the system to model !).

With the implementation shown in the above block diagram, the limit cycle is avoided by relying on the upper integrator to decide when the lower integrator can be released. The upper integrator is used to observe what the evolution of the system would be with x_1 free to vary: the lower integrator is re-activated only when $K_p x_k + x_2$ is in $[x_j^{min}\ x_j^{max}]$. As it is in “open loop”, the additional integrator does not interact with the rest of the system.

Syntax : & pictlieee
name of variable x_k
name of variable x_j
data name, parameter name or math expression for K_i
data name, parameter name or math expression for K_p
data name, parameter name or math expression for x_j^{min}
data name, parameter name or math expression for x_j^{max}

Internal states: x_1 and x_2

Discrete variables : $z \in \{-2, -1, 0, 1, 2\}$

Equations :

$$\text{if } z = 0 \; : \; \begin{cases} 0 & = K_p x_k + x_1 - x_j \\ \dot{x}_1 & = K_i \, x_k \\ 0 & = x_2 - x_1 \end{cases}$$

$$\text{if } z = 2 \; : \; \begin{cases} 0 & = x_j^{max} - x_j \\ \dot{x}_1 & = 0 \\ 0 & = x_2 - x_1 \end{cases} \qquad \text{if } z = 1 \; : \; \begin{cases} 0 & = x_j^{max} - x_j \\ \dot{x}_1 & = 0 \\ \dot{x}_2 & = K_i \, x_k \end{cases}$$

$$\text{if } z = -2 \; : \; \begin{cases} 0 & = x_j^{min} - x_j \\ \dot{x}_1 & = 0 \\ 0 & = x_2 - x_1 \end{cases} \qquad \text{if } z = -1 \; : \; \begin{cases} 0 & = x_j^{min} - x_j \\ \dot{x}_1 & = 0 \\ \dot{x}_2 & = K_i \, x_k \end{cases}$$

Discrete transitions (note the use of x_2 in the tests marked with (*)):

if $z = 0$ **then**
  if $x_j > x_j^{max}$ **then**
    $z \leftarrow 2$
  else if $x_j < x_j^{min}$ **then**
    $z \leftarrow -2$
  end if
else if $z = 2$ **then**
  if $K_p x_k + x_1 < x_j^{max}$ **then**
    $z \leftarrow 1$
  end if
else if $z = 1$ **then**
  if $K_p x_k + x_1 > x_j^{max}$ **then**
    $z \leftarrow 2$
  else if $K_p x_k + x_2 < x_j^{max}$ (*) **then**
    $z \leftarrow 0$
  end if
else if $z = -2$ **then**
  if $K_p x_k + x_1 > x_j^{min}$ **then**
    $z \leftarrow -1$
  end if
else if $z = -1$ **then**
  if $K_p x_k + x_1 < x_j^{min}$ **then**
    $z \leftarrow -2$
  else if $K_p x_k + x_2 > x_j^{min}$ (*) **then**
    $z \leftarrow 0$
  end if
end if

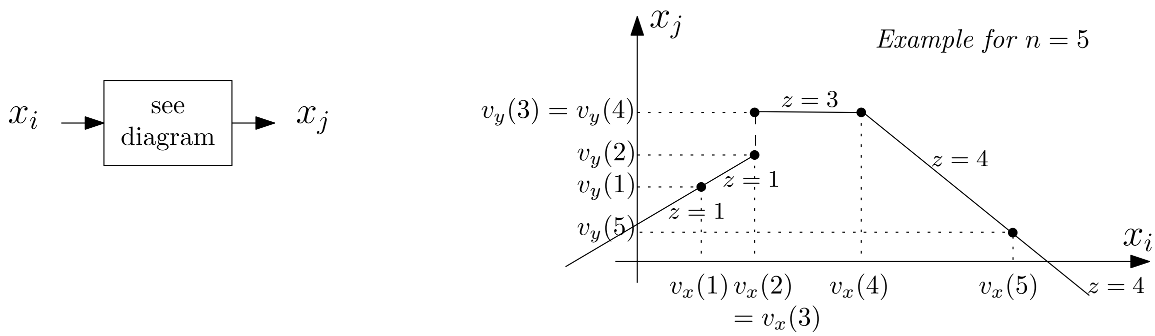
Initialization of the internal state and the discrete variable:

if $x_j \geq x_j^{max}$ **then**
  $z \leftarrow 2$
  $x_1 \leftarrow x_j^{max} - K_p x_k$
  $x_2 \leftarrow x_1$
else if $x_j \leq x_j^{min}$ **then**
  $z \leftarrow -2$
  $x_1 \leftarrow x_j^{min} - K_p x_k$
  $x_2 \leftarrow x_1$
else
  $z \leftarrow 0$
  $x_1 \leftarrow x_j$
  $x_2 \leftarrow x_1$
end if

pwlin*

Piece-wise linear function of input, defined by n points.

Separate blocks exist for $n = 3, 4, 5$ and 6.



Syntax : & pwlin3
name of variable x_i
name of variable x_j
data name, parameter name or math expression for $v_x(1)$
data name, parameter name or math expression for $v_y(1)$
data name, parameter name or math expression for $v_x(2)$
data name, parameter name or math expression for $v_y(2)$
data name, parameter name or math expression for $v_x(3)$
data name, parameter name or math expression for $v_y(3)$

& pwlin4
name of variable x_i
name of variable x_j
data name, parameter name or math expression for $v_x(1)$
data name, parameter name or math expression for $v_y(1)$
data name, parameter name or math expression for $v_x(2)$
data name, parameter name or math expression for $v_y(2)$
data name, parameter name or math expression for $v_x(3)$
data name, parameter name or math expression for $v_y(3)$
data name, parameter name or math expression for $v_x(4)$
data name, parameter name or math expression for $v_y(4)$

& pwlin5
name of variable x_i
name of variable x_j
data name, parameter name or math expression for $v_x(1)$
data name, parameter name or math expression for $v_y(1)$
data name, parameter name or math expression for $v_x(2)$
data name, parameter name or math expression for $v_y(2)$
data name, parameter name or math expression for $v_x(3)$
data name, parameter name or math expression for $v_y(3)$
data name, parameter name or math expression for $v_x(4)$
data name, parameter name or math expression for $v_y(4)$
data name, parameter name or math expression for $v_x(5)$
data name, parameter name or math expression for $v_y(5)$

& pwlin6
name of variable x_i
name of variable x_j
data name, parameter name or math expression for $v_x(1)$
data name, parameter name or math expression for $v_y(1)$
data name, parameter name or math expression for $v_x(2)$
data name, parameter name or math expression for $v_y(2)$
data name, parameter name or math expression for $v_x(3)$
data name, parameter name or math expression for $v_y(3)$
data name, parameter name or math expression for $v_x(4)$
data name, parameter name or math expression for $v_y(4)$
data name, parameter name or math expression for $v_x(5)$
data name, parameter name or math expression for $v_y(5)$
data name, parameter name or math expression for $v_x(6)$
data name, parameter name or math expression for $v_y(6)$

Internal states : none

Discrete variables : $z \in \{1, \ldots, n-1\}$

Equations :

$$0 = v_y(z) + \frac{v_y(z+1) - v_y(z)}{v_x(z+1) - v_x(z)} \left(x_i - v_x(z)\right) \; - \; x_j$$

Discrete transitions :

if $x_i < v_x(1)$ **then**
    $z \leftarrow 1$
else if $x_i \geq v_x(n)$ **then**
    $z \leftarrow n-1$
else
    for $k = 1$ to $n-1$ **do**
        if $v_x(k) \leq x_i$ **and** $x_i < v_x(k+1)$ **then**
            $z \leftarrow k$
        end if
    end for
end if

Initialization of discrete variables : same code as for the discrete transitions

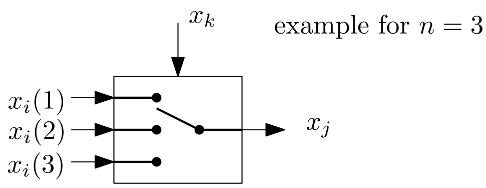
The v_x values must be increasing, i.e. $v_x(1) < v_x(2) \leq v_x(3) \leq \ldots \leq v_x(n-1) < v_x(n)$.

For x_i values smaller than $v_x(1)$ (resp. larger than $v_x(n)$), x_j is obtained by linear extrapolation based on the first two (resp. the last two) points. Hence, the data must be such that $v_x(1) \neq v_x(2)$ and $v_x(n-1) \neq v_x(n)$.

switch*

Set the output state to one among n input states, based on the value of a controlling state.

Separate blocks exist for $n = 2, 3, 4$ and 5.



Syntax :

& switch2
name of variable $x_i(1)$
name of variable $x_i(2)$
name of variable x_k
name of variable x_j

& switch3
name of variable $x_i(1)$
name of variable $x_i(2)$
name of variable $x_i(3)$
name of variable x_k
name of variable x_j

& switch4
name of variable $x_i(1)$
name of variable $x_i(2)$
name of variable $x_i(3)$
name of variable $x_i(4)$
name of variable x_k
name of variable x_j

& switch5
name of variable $x_i(1)$
name of variable $x_i(2)$
name of variable $x_i(3)$
name of variable $x_i(4)$
name of variable $x_i(5)$
name of variable x_k
name of variable x_j

Internal states : none

Discrete variables : $z \in \{1, 2, \ldots, n\}$

Equations :

$$0 = x_j - x_i(z)$$

Discrete transitions :

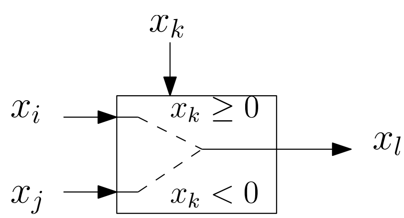
$$z \leftarrow \max\left(1, \min(n, nint(x_k))\right)$$

where the nint function returns the nearest integer.

Initialization of discrete variables : same code as for the discrete transitions

swsign

Switch between two input states, based on the sign of a third input state



Syntax : & swsign
name of variable x_i
name of variable x_j
name of variable x_k
name of variable x_l

Internal states : none

Discrete variables : $z \in \{1, 2\}$

Equations :

$$0 = \begin{cases} x_l - x_i & \text{if } z = 1 \\ x_l - x_j & \text{if } z = 2 \end{cases}$$

Discrete transitions :

if $z = 1$ **then**
  if $x_k < 0$ **then**
    $z \leftarrow 2$
  end if
else if $z = 2$ **then**
  if $x_k \geq 0$ **then**
    $z \leftarrow 1$
  end if
end if

Initialization of discrete variables :

if $x_k < 0$ **then**
    $z \leftarrow 2$
else
    $z \leftarrow 1$
end if

tf1p

Transfer function between input and output: one time constant

$$x_i \longrightarrow \boxed{\frac{G}{1+sT}} \longrightarrow x_j$$

Syntax :	& tf1p
	name of variable x_i
	name of variable x_j
	data name, parameter name or math expression for G
	data name, parameter name or math expression for T

Internal states : none

Discrete variables : none

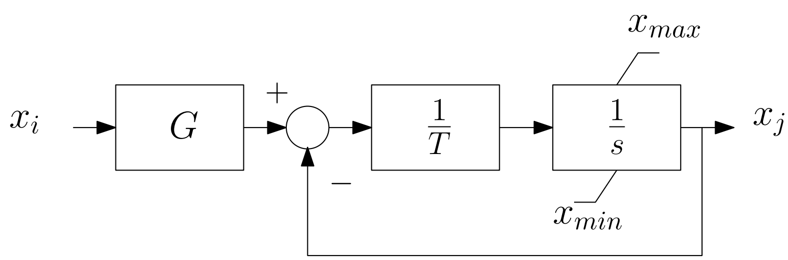
Equations :

$$T\,\dot{x}_j = -x_j + G\,x_i$$

The time constant T can be zero.

tf1plim

Transfer function between input and output: one time constant with non-windup limits on output.



Syntax : & tf1plim
name of variable x_i
name of variable x_j
data name, parameter name or math expression for G
data name, parameter name or math expression for T
data name, parameter name or math expression for x_{min}
data name, parameter name or math expression for x_{max}

Internal states : none

Discrete variables : $z \in \{-1, 0, 1\}$

Equations :

$$\begin{cases} T\,\dot{x}_j & = G\,x_i - x_j & \text{if } z = 0 \\ 0 & = x_j - x_{max} & \text{if } z = 1 \\ 0 & = x_j - x_{min} & \text{if } z = -1 \end{cases}$$

Discrete transitions :

if $z = 0$ **then**
  if $x_j > x_{max}$ **then**
    $z \leftarrow 1$
  else if $x_j < x_{min}$ **then**
    $z \leftarrow -1$
  end if
else if $z = 1$ **then**
  if $G\, x_i - x_j < 0$ **then**
    $z \leftarrow 0$
  end if
else if $z = -1$ **then**
  if $G\, x_i - x_j > 0$ **then**
    $z \leftarrow 0$
  end if
end if

The time constant T can be zero. In this case:

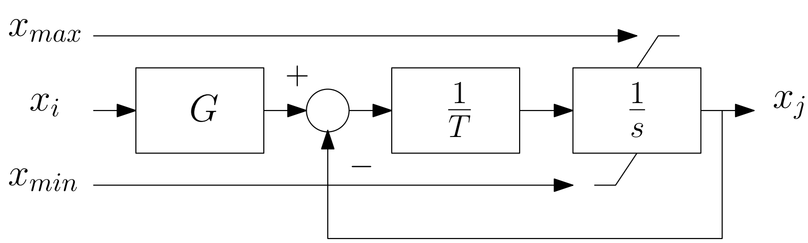
- the block behaves like a gain: $x_j = Gx_i$
- the limits x_{min} and x_{max} remain in effect.

Initialization of discrete variables :

if $x_j \geq x_{max}$ **then**
  $z \leftarrow 1$
else if $x_j \leq x_{min}$ **then**
  $z \leftarrow -1$
else
  $z \leftarrow 0$
end if

tf1pvlim

Transfer function between input and output: one time constant with non-windup limits on output. The limits are variables.



Syntax : & tf1plim
name of variable x_i
name of variable x_j
name of variable x_{min}
name of variable x_{max}
data name, parameter name or math expression for G
data name, parameter name or math expression for T

Internal states : none

Discrete variables : $z \in \{-1, 0, 1\}$

Equations :

$$\begin{cases} T\,\dot{x}_j &= G\,x_i - x_j \qquad \text{if } z = 0 \\ 0 &= x_j - x_{max} \qquad \text{if } z = 1 \\ 0 &= x_j - x_{min} \qquad \text{if } z = -1 \end{cases}$$

Discrete transitions :

if $z = 0$ **then**
  if $x_j > x_{max}$ **then**
    $z \leftarrow 1$
  else if $x_j < x_{min}$ **then**
    $z \leftarrow -1$
  end if
else if $z = 1$ **then**
  if $G\,x_i - x_j < 0$ **then**
    $z \leftarrow 0$
  end if
else if $z = -1$ **then**
  if $G\,x_i - x_j > 0$ **then**
    $z \leftarrow 0$
  end if
end if

The time constant T can be zero. In this case:

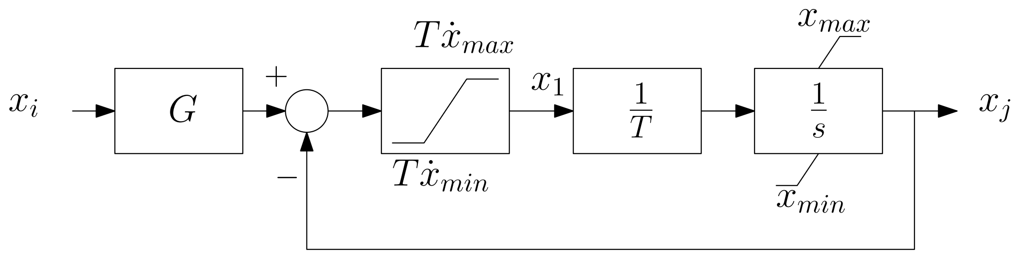
- the block behaves like a gain: $x_j = Gx_i$
- the limits x_{min} and x_{max} remain in effect.

Initialization of discrete variables :

if $x_j \geq x_{max}$ **then**
  $z \leftarrow 1$
else if $x_j \leq x_{min}$ **then**
  $z \leftarrow -1$
else
  $z \leftarrow 0$
end if

tf1p2lim

Transfer function between input and output: one time constant, with limits on rate of change of output and non-windup limits on output



Syntax : & tf1p2lim
name of variable x_i
name of variable x_j
data name, parameter name or math expression for G
data name, parameter name or math expression for T
data name, parameter name or math expression for x_{min}
data name, parameter name or math expression for x_{max}
data name, parameter name or math expression for $\dot{x}_{min}$
data name, parameter name or math expression for $\dot{x}_{max}$

One internal state : x_1 initialized at $\min[\max(0, T\,\dot{x}_{min}), T\,\dot{x}_{max}]$

Two discrete variables : $z_1 \in \{-1, 0, 1\}$ and $z_2 \in \{-1, 0, 1\}$

Two equations :

$$\begin{cases} 0 &= x_1 - G\,x_i + x_j \qquad \text{if } z_1 = 0 \\ 0 &= x_1 - T\,\dot{x}_{max} \qquad \text{if } z_1 = 1 \\ 0 &= x_1 - T\,\dot{x}_{min} \qquad \text{if } z_1 = -1 \end{cases}$$

$$\begin{cases} T\,\dot{x}_j &= x_1 \qquad \text{if } z_2 = 0 \\ 0 &= x_j - x_{max} \qquad \text{if } z_2 = 1 \\ 0 &= x_j - x_{min} \qquad \text{if } z_2 = -1 \end{cases}$$

$\dot{x}_{max}$ (resp. $\dot{x}_{min}$) is the maximum (resp. minimum) rate of change of x with time.

The time constant T can be zero. In this case:

- $x_1 = 0$ throughout the whole simulation
- the block behaves like a gain: $x_j = Gx_i$
- both limiters are ignored: $x_{max} \to \infty,\ x_{min} \to -\infty, T\dot{x}_{max} \to \infty,\ T\dot{x}_{min} \to -\infty$.

Discrete transitions :

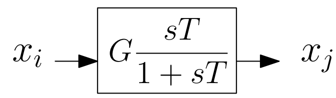
if $z_1 = 0$ **then**
  if $x_1 > T\,\dot{x}_{max}$ **then**
    $z_1 \leftarrow 1$
  else if $x_1 < T\,\dot{x}_{min}$ **then**
    $z_1 \leftarrow -1$
  end if
else if $z_1 = 1$ **then**
  if $Gx_i - x_j < T\,\dot{x}_{max}$ **then**
    $z_1 \leftarrow 0$
  end if
else if $z_1 = -1$ **then**
  if $Gx_i - x_j > T\,\dot{x}_{min}$ **then**
    $z_1 \leftarrow 0$
  end if
end if
if $z_2 = 0$ **then**
  if $x_j > x_{max}$ **then**
    $z_2 \leftarrow 1$
  else if $x_j < x_{min}$ **then**
    $z_2 \leftarrow -1$
  end if
else if $z_2 = 1$ **then**
  if $x_1 < 0$ **then**
    $z_2 \leftarrow 0$
  end if
else if $z_2 = -1$ **then**
  if $x_1 > 0$ **then**
    $z_2 \leftarrow 0$
  end if
end if

Initialization of discrete variables :

if $Gx_i - x_j > T\,\dot{x}_{max}$ **then**
  $z_1 \leftarrow 1$
else if $Gx_i - x_j < T\,\dot{x}_{min}$ **then**
  $z_1 \leftarrow -1$
else
  $z_1 \leftarrow 0$
end if
if $x_j > x_{max}$ **then**
  $z_2 \leftarrow 1$
else if $x_j < x_{min}$ **then**
  $z_2 \leftarrow -1$
else
  $z_2 \leftarrow 0$
end if

tfder1p

Transfer function: derivative with one time constant



Syntax : & tfder1p
name of variable x_i
name of variable x_j
data name, parameter name or math expression for G
data name, parameter name or math expression for T

Internal states : x_1

Discrete variables : none

Equations :

$$T\,\dot{x}_1 = x_j \tag{12.1}$$
$$0 = G\,x_i - x_1 - x_j \tag{12.2}$$

Initialization of internal states: $x_1 = G\,x_i$

The model allows $T = 0$. In this case:

- $x_j = 0$ as expected;
- Eq. (12.2) is useless but is integrated with the rest of the model.

tf1p1z

Transfer function between input and output: one zero and one pole

$$x_i \longrightarrow \boxed{G\frac{1+sT_z}{1+sT_p}} \longrightarrow x_j$$

Syntax : & tf1p1z
name of variable x_i
name of variable x_j
data name, parameter name or math expression for G
data name, parameter name or math expression for T_z
data name, parameter name or math expression for T_p

Internal states : x_1

Discrete variables : none

Equations :

$$\dot{x_1} = G\,x_i - x_j \tag{12.3}$$

$$0 = T_p\,x_j - G\,T_z\,x_i - x_1 \tag{12.4}$$

Initialization of internal states : $x_1 = G\,(T_p - T_z)x_i$

The model allows $T_z = 0$. In this case, simplifying Eq. (12.4) and combining it with Eq. (12.3) yields $T_p\,\dot{x}_j = -x_j + G\,x_i$, which corresponds to $x_j = \dfrac{G}{1+sT_p}x_i$ as expected.

Formally, the model also allows $T_p = 0$. However, in this case, the transfer function between x_i and x_j becomes $G\,(1 + sT_z)$. **This involves a pure derivator. Hence, the solver may produce undesired transients when x_i or $\dot{x}_i$ undergoes a discontinuity.**

The model allows $T_p = T_z = T$. In this case, Equation (12.4) becomes $x_j - G\,x_i = \dfrac{x_1}{T}$, and replacing this result in Eq. (12.3) yields $\dot{x}_1 = -\dfrac{x_1}{T}$, showing that x_1 evolves independently of x_i and x_j. Furthermore, when $T_z = T_p$, x_1 is initialized to zero; hence, it remains equal to zero for the whole simulation. Replacing $x_1 = 0$ yields $x_j = G\,x_i$ as expected.

tf2p2z

Transfer function between input and output: two real zeros and two real poles

$$x_i \longrightarrow \boxed{G\frac{1+n_1s+n_2s^2}{1+d_1s+d_2s^2}} \longrightarrow x_j$$

Syntax : & tf2p2z
name of variable x_i
name of variable x_j
data name, parameter name or math expression for G
data name, parameter name or math expression for n_1
data name, parameter name or math expression for n_2
data name, parameter name or math expression for d_1
data name, parameter name or math expression for d_2

Internal states : x_1 and x_2

Discrete variables : none

Equations : correspond to a second-order state space model with controllable canonical form:

$$\begin{aligned} \dot{x_1} &= x_2 && (12.1) \\ d_2\,\dot{x_2} &= -x_1 - d_1x_2 + d_2x_i && (12.2) \\ 0 &= G(d_2 - n_2)x_1 + G(n_1d_2 - d_1n_2)x_2 + Gn_2d_2x_i - d_2^2x_j && (12.3) \end{aligned}$$

Initialization of internal states : $x_2 = 0$ and $x_1 = d_2x_i$

Exception. If a small value is specified for d_2, namely if $d_2 < 0.005$, both d_2 and n_2 are set to zero and the transfer function becomes:

$$G\frac{1 + n_1s}{1 + d_1s}$$

as considered in the `tf1p1z` block.

If, in addition, a small value is specified for d_1, namely if $d_1 < 0.005$, both d_1 and n_1 are set to zero and the block behaves as a simple gain:

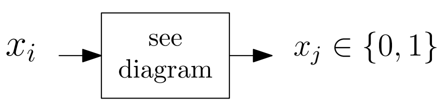
$$x_j = Gx_i$$

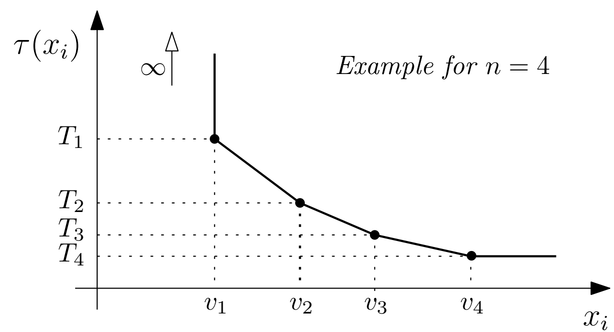
timer*

Timer with varying delay. The latter is a piecewise linear function of the monitored variable

If x_i is smaller than a threshold v_1, the output x_j is equal to zero. Otherwise, x_j changes from zero to one at time $t^\star + \tau(x_i)$ where $t^\star$ is the time at which the input x_i became larger than v_1 and the delay $\tau(x_i)$ varies with x_i according to a piecewise linear characteristic involving n points (see diagram below).

Separate blocks exist for $n = 1, 2, 3, 4$ and 5.





Syntax : & timer1
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1

& timer2
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1
data name, parameter name or math expression for v_2
data name, parameter name or math expression for T_2

& timer3
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1
data name, parameter name or math expression for v_2
data name, parameter name or math expression for T_2
data name, parameter name or math expression for v_3
data name, parameter name or math expression for T_3

& timer4
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1
data name, parameter name or math expression for v_2
data name, parameter name or math expression for T_2
data name, parameter name or math expression for v_3
data name, parameter name or math expression for T_3
data name, parameter name or math expression for v_4
data name, parameter name or math expression for T_4

& timer5
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1
data name, parameter name or math expression for v_2
data name, parameter name or math expression for T_2
data name, parameter name or math expression for v_3
data name, parameter name or math expression for T_3
data name, parameter name or math expression for v_4
data name, parameter name or math expression for T_4
data name, parameter name or math expression for v_5
data name, parameter name or math expression for T_5

Internal states : x_1

Discrete variables : $z \in \{-1, 0, 1\}$

Equations :

$$0 = \begin{cases} x_j & \text{if } z \in \{-1, 0\} \\ x_j - 1 & \text{if } z = 1 \end{cases}$$

$$\begin{cases} 0 & = x_1 \quad \text{if } z = -1 \\ \dot{x_1} & = 1 \quad \text{if } z = 0 \\ \dot{x_1} & = 0 \quad \text{if } z = 1 \end{cases}$$

Discrete transitions :

if $z = -1$ **then**
  if $x_i \geq v_1$ **then**
    $z \leftarrow 0$
  end if
else
  if $x_i < v_1$ **then**
    $z \leftarrow -1$
  end if
end if
if $z = 0$ **then**
  if $x_1 \geq \tau(x_i)$ **then**
    $z \leftarrow 1$
  end if
end if

Initialization of internal states : $x_1 \leftarrow 0$

Initialization of discrete variables : $z \leftarrow -1$

The v_i values must be increasing, but two consecutive values may be equal, i.e. $v_1 \leq v_2 \leq v_3 \leq \ldots \leq v_{n-1} \leq v_n$.

The piecewise linear characteristic is typically used to approximate an inverse-time characteristic, in which case the T values are decreasing, i.e. $T_1 \geq T_2 \geq T_3 \geq \ldots \geq T_{n-1} \geq T_n$. Nevertheless, non decreasing values are also allowed.

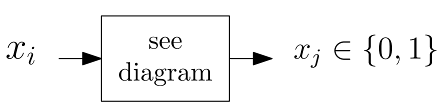
If the initial value of x_i is larger than v_1, x_j will change to one after the time $\tau(x_i)$, unless x_i decreases below v_1 before the delay τ is elapsed.

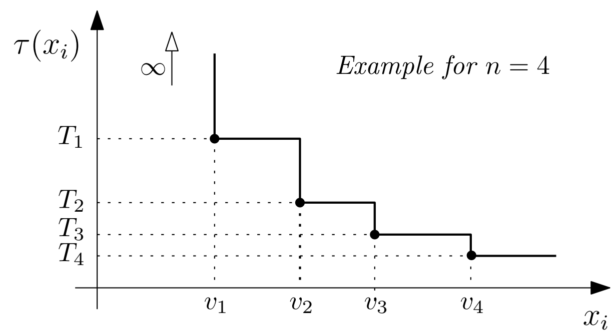
timersc*

Timer with varying delay. The latter is a staircase function of the monitored variable

If x_i is smaller than a threshold v_1, the output x_j is equal to zero. Otherwise, x_j changes from zero to one at time $t^\star + \tau(x_i)$ where $t^\star$ is the time at which the input x_i became larger than v_1 and the delay $\tau(x_i)$ varies with x_i according to a staircase characteristic involving n points (see diagram below).

Separate blocks exist for $n = 1, 2, 3, 4, 5$ and 6.





Syntax : & timersc1
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1

& timersc2
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1
data name, parameter name or math expression for v_2
data name, parameter name or math expression for T_2

& timersc3
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1
data name, parameter name or math expression for v_2
data name, parameter name or math expression for T_2
data name, parameter name or math expression for v_3
data name, parameter name or math expression for T_3

& timersc4
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1
data name, parameter name or math expression for v_2
data name, parameter name or math expression for T_2
data name, parameter name or math expression for v_3
data name, parameter name or math expression for T_3
data name, parameter name or math expression for v_4
data name, parameter name or math expression for T_4

& timersc5
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1
data name, parameter name or math expression for v_2
data name, parameter name or math expression for T_2
data name, parameter name or math expression for v_3
data name, parameter name or math expression for T_3
data name, parameter name or math expression for v_4
data name, parameter name or math expression for T_4
data name, parameter name or math expression for v_5
data name, parameter name or math expression for T_5

& timersc6
name of variable x_i
name of variable x_j
data name, parameter name or math expression for v_1
data name, parameter name or math expression for T_1
data name, parameter name or math expression for v_2
data name, parameter name or math expression for T_2
data name, parameter name or math expression for v_3
data name, parameter name or math expression for T_3
data name, parameter name or math expression for v_4
data name, parameter name or math expression for T_4
data name, parameter name or math expression for v_5
data name, parameter name or math expression for T_5
data name, parameter name or math expression for v_6
data name, parameter name or math expression for T_6

Internal states : x_1

Discrete variables : $z \in \{-1, 0, 1\}$

Equations :

$$0 = \begin{cases} x_j & \text{if } z \in \{-1, 0\} \\ x_j - 1 & \text{if } z = 1 \end{cases}$$

$$\begin{cases} 0 & = x_1 \qquad \text{if } z = -1 \\ \dot{x_1} & = 1 \qquad \text{if } z = 0 \\ \dot{x_1} & = 0 \qquad \text{if } z = 1 \end{cases}$$

Discrete transitions :

if $z = -1$ **then**
  if $x_i \geq v_1$ **then**
    $z \leftarrow 0$
  end if
else
  if $x_i < v_1$ **then**
    $z \leftarrow -1$
  end if
end if
if $z = 0$ **then**
  if $x_1 \geq \tau(x_i)$ **then**
    $z \leftarrow 1$
  end if
end if

Initialization of internal states : $x_1 \leftarrow 0$

Initialization of discrete variables : $z \leftarrow -1$

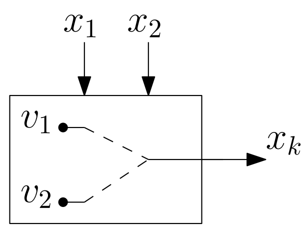
The v_i values must be increasing, but two consecutive values may be equal, i.e. $v_1 \leq v_2 \leq v_3 \leq \ldots \leq v_{n-1} \leq v_n$.

The staircase characteristic is typically used to approximate an inverse-time characteristic, in which case the T values are decreasing, i.e. $T_1 \geq T_2 \geq T_3 \geq \ldots \geq T_{n-1} \geq T_n$. Nevertheless, non decreasing values are also allowed.

If the initial value of x_i is larger than v_1, x_j will change to one after the time $\tau(x_i)$, unless x_i decreases below v_1 before the delay τ is elapsed.

tsa

Two-state automaton with transitions based on signs of two inputs



The output state x_k can take two values: v_1 or v_2. When $x_k = v_1$ the change to v_2 depends on the sign of x_1; when $x_k = v_2$ the change to v_1 depends on the sign of x_2.

Syntax :	& 2sa
	name of variable x_1
	name of variable x_2
	name of variable x_k
	data name, parameter name or math expression for v_1
	data name, parameter name or math expression for v_2

Internal states : none

Discrete variable : $z \in \{1, 2\}$ represents the state of the automaton

Equations :

$$0 = \begin{cases} x_k - v_1 & \text{if } z = 1 \\ x_k - v_2 & \text{if } z = 2 \end{cases}$$

Discrete transitions :

if $z = 1$ and $x_1 > 0$ **then**
  $z \leftarrow 2$
else if $z = 2$ and $x_2 > 0$ **then**
  $z \leftarrow 1$
end if

Initialization of discrete variable : the initial value is $z = 1$, i.e. $x_k = v_1$ is assumed initially.

12.4 Functions available in models

The functions documented in this section can be used in all mathematical expressions mentionned in Fig. 11.7.

The first two are general. The others are power system-oriented. Among them, some functions are restricted to some types of models.

```
double precision function equal(var1,var2)

        double precision::  var1, var2
```

This function returns 1.d0 if $var1$ differs from $var2$ by less than 10^{-6}, and 0.d0 otherwise.

```
double precision function equalstr(var1,var2)

       character::  var1, var2
```

This function returns 1.d0 if the non-blank parts of str1 and str2 are the same, and 0.d0 otherwise.

```
double precision ppower([vx],[vy],[ix],[iy])
```

Returns the active power injected into network

Can be used in models of type: `inj`

Computation :

$$P = v_x i_x + v_y i_y$$

```
double precision qpower([vx],[vy],[ix],[iy])
```

Returns the reactive power injected into network

Can be used in models of type: `inj`

Computation :

$$Q = v_y i_x - v_x i_y$$

`double precision function vrectif([if],[vin],{kc})`

```
double precision:: kc
```

This function is used to model the voltage drop in rectifiers. It gives the output voltage $vrectif$ as a function of the output current if and the input voltage vin. It appears in some IEEE standard excitation models with $vrectif = f_{ex}\,E_{FD}$

Can be used in models of type: `exc`

Computation :

$in = kc\,if/max(vin, 10^{-3})$
if $in \leq 0$ ***then***
  $vrectif = vin$
else if $in \leq 0.433$ ***then***
  $vrectif = vin - 0.577\,kc\,if$
else if $in \leq 0.75$ ***then***
  $vrectif = \sqrt{0.75\,vin^2 - (kc\,if)^2}$
else if $in \leq 1.00$ ***then***
  $vrectif = 1.732(vin - kc\,if)$
else
  $vrectif = 0$
end if

`double precision function vinrectif([if],[vrectif],{kc})`

```
double precision:: kc
```

This function is the inverse of `vrectif`*. It is aimed at being called at initialization. For given values the field current if and rectifier output voltage $vrectif$ it returns the value of the rectifier input voltage $vinrectif$. This determination is iterative because the portion of the nonlinear input-output characteristic on which the rectifier is operating is not known beforehand. The function does not work with a zero rectifier output $vrectif$ since, in this case, the input voltage is indeterminate.*

Can be used in models of type: `exc`

Computation :

$nbtries = 1$
$vinest = vrectif + 0.577\ kc\ if$
loop
$in = kc\ if/max(vinest, 1d - 03)$
if $in \leq 0$ ***then***
$vinrectif = vrectif$
else if $in \leq 0.433$ ***then***
$vinrectif = vrectif + 0.577\ kc\ if$
else if $in \leq 0.75$ ***then***
$vinrectif = \sqrt{vrectif^2 + (kc\ if)^2}/0.75$
else
$vinrectif = (vrectif/1.732) + kc\ if$
end if
if $vinrectif = vinest$ *or* $nbtries > 5$ ***then***
exit loop
end if
$nbtries = nbtries + 1$
$vinest = vinrectif$
end loop

```
double precision function vcomp([v],[p],[q],{Kv},{Rc},{Xc})

        double precision::  Kv, Rc, Xc
```

This function returns the magnitude of a combination of the terminal voltage and the current of a synchronous machine. It appears in some IEEE standard excitation models

Can be used in models of type: `exc`

Computation :

$$
\begin{aligned}
V_{comp} &= |K_v\bar{V} + (R_c + jX_c)\bar{I}| = |K_vV + (R_c + jX_c)(i_P - ji_Q)| \\
&= |K_vV + (R_c + jX_c)(\frac{P}{V} - j\frac{Q}{V})| \\
&= \frac{1}{V}\sqrt{(K_vV^2 + R_cP + X_cQ)^2 + (X_cP - R_cQ)^2}
\end{aligned}
$$

```
double precision function satur([ve],{ve1},{se1},{ve2},{se2})

        double precision::  ve1, se2, ve2, se2
```

This function returns the increment of field current needed to obtain a given output voltage, taking saturation into account. It appears in the model of some excitation systems.

Can be used in models of type: `exc`

Computation :

$$satur = m\ ve^n$$

where:

$$\begin{aligned} n &= \frac{\log_{10}(se1/se2)}{\log_{10}(ve1/ve2)} \\ m &= \frac{se1}{ve1^n} \end{aligned}$$

Exception. *The function returns $satur = 0$ if any of the following conditions holds true:*

$$ve \leq 0 \quad or \quad ve1 \leq 0 \quad or \quad ve2 \leq 0 \quad or \quad ve1 = ve2 \quad or \quad se1 = 0 \quad or \quad se2 = 0$$