

# MODPROD 2016

## iPSL Tutorial

Tin Rabuzin and Luigi Vanfretti

E-mail: [luigiv@kth.se](mailto:luigiv@kth.se)

Web: <http://www.vanfretti.com>



[luigiv@kth.se](mailto:luigiv@kth.se)  
*Associate Professor, Docent*  
Electric Power Systems Dept.  
KTH  
Stockholm, Sweden



[Luigi.Vanfretti@statnett.no](mailto:Luigi.Vanfretti@statnett.no)  
*Special Advisor in Strategy and Public Affairs*  
Research and Development Division  
Statnett SF  
Oslo, Norway

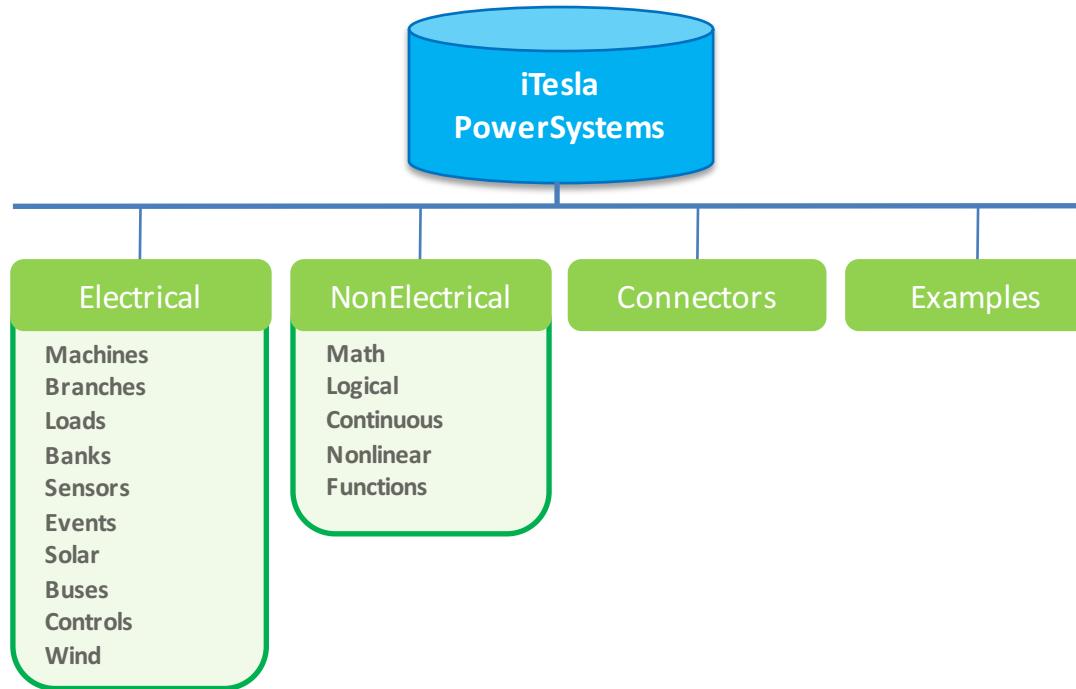
# Outline

1. Aim of the workshop
2. Library introduction
3. Example 1
4. Example 2
5. Example 3

# Aim of the workshop

- Introduction to the iTesla PowerSystems Library
- Modelling and simulation possibilities by using iPSL and Modelica
- Comparison of the performance with a reference simulation software
- 3 use cases with a dynamic simulation and linearization

# Library introduction



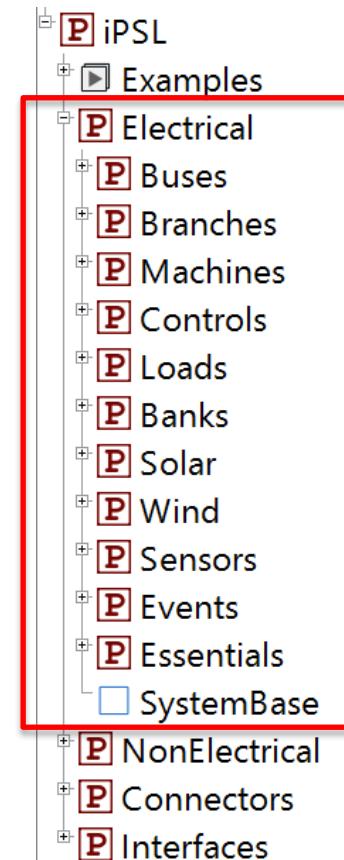
iTesla PowerSystems library is divided in four main categories:

- Examples
- Electrical
- NonElectrical
- Connectors

# Library introduction

## Electrical

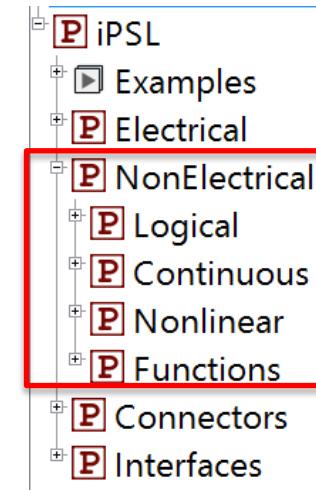
- The *Electrical* package contains most of the components that comprise an actual power network
- E.g., electrical machines, transmission lines, loads, excitation systems, turbine governors, etc.
- These are used to build the power system network models



# Library introduction

## NonElectrical

- The *NonElectrical* package is comprised by functions, blocks or models, which are used to build the aforementioned power system component models
- Transfer functions, logical operators, etc.
- They perform specific operations which were not available in the Modelica Standard Library (MSL)



# Library introduction

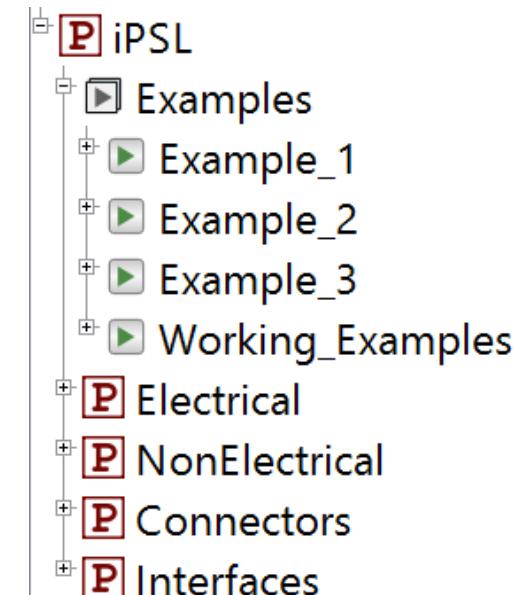
## Connectors

- The *Connectors* package contains a set of specifically developed Modelica connectors to harmonize the models in this library
- As an example, *PwPin* is the connector, which contains voltage and current quantities in phasor representation

# Library introduction

## Examples

- In this workshop, *Examples* package will be used to showcase the possibilities of the library
- In the packages *Example\_1*, *Example\_2* and *Example\_3* prepared use cases can be found where steps to build the models are described
- Package *Working\_Examples* and corresponding subpackages will be used by attendees of the workshop to create use cases on their own
- **Note!** The library provided during the workshop is a stripped version with a reduced number of models.





# OMEEdit and iPSL

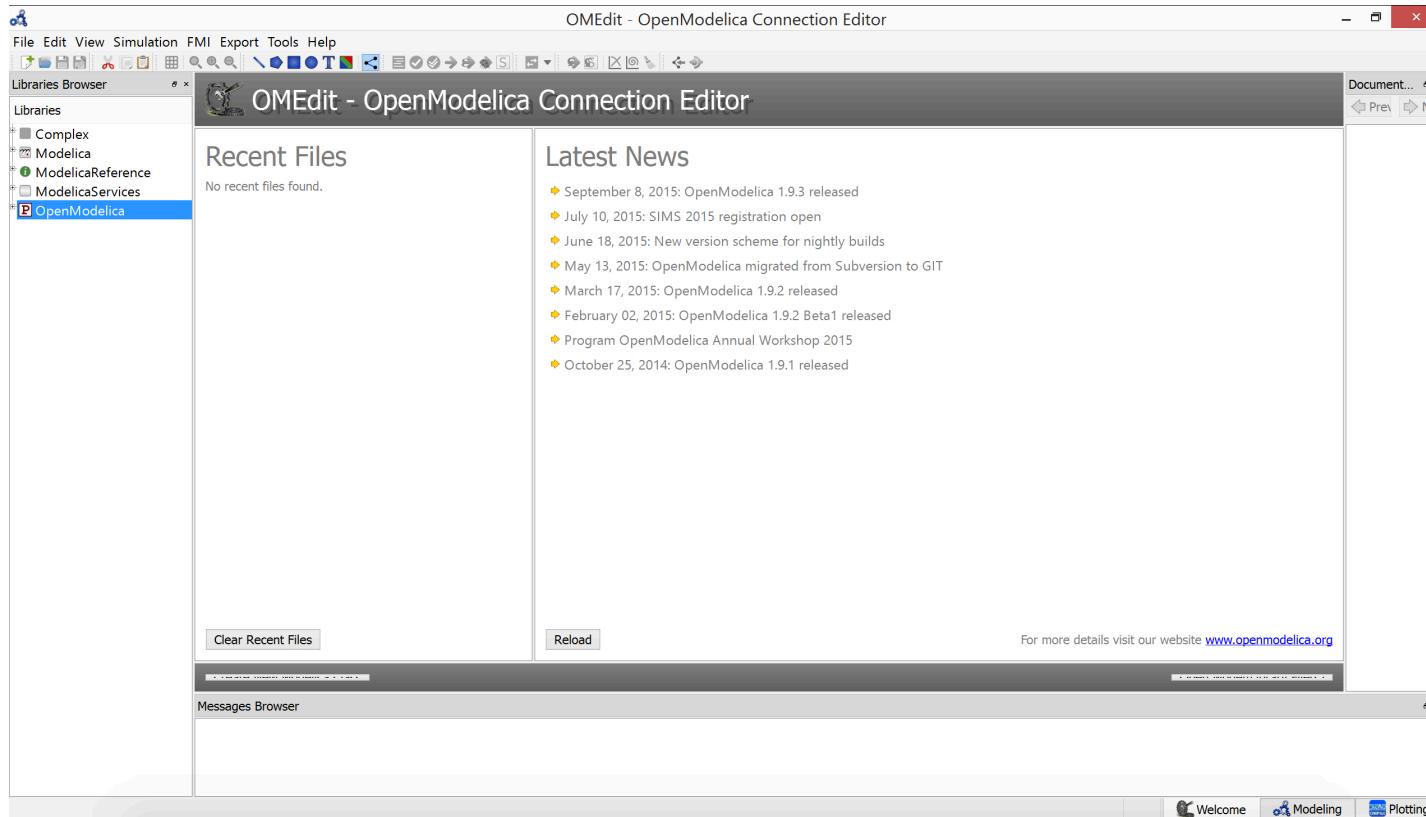
**iPSL along with the additional files should be downloaded from:**

<https://www.dropbox.com/sh/m0tukq5089kjmz/AAAEyls7oGPqjDSZgZHfAjcNa?dl=0>

**Downloaded content should be placed on C:\iPSL\_Workshop**

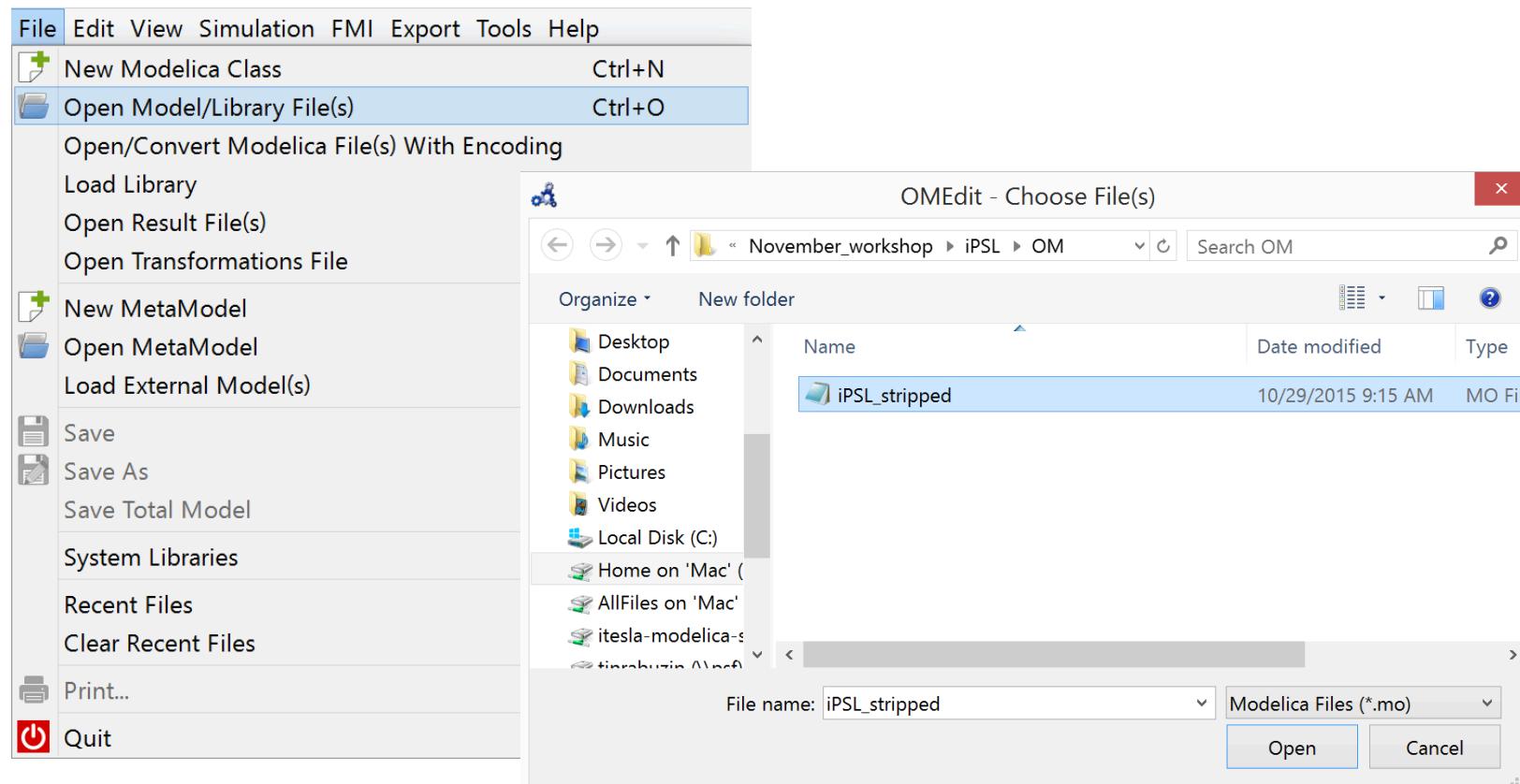
# OMEdit and iPSL

- The workshop will be carried out in OpenModelica Connection Editor (OMEdit)
- The start screen of OMEdit should look like the one shown on the figure



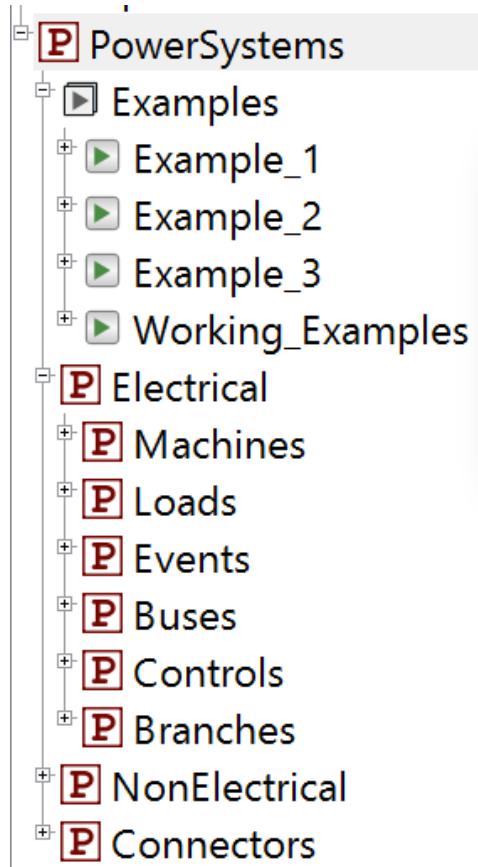
# OMEdit and iPSL

- To open the iPSL one should go to File->Open Model/Library File(s) and navigate to the iPSL file that should be located in `c:/iPSL_Workshop/iPSL_stripped.mo`

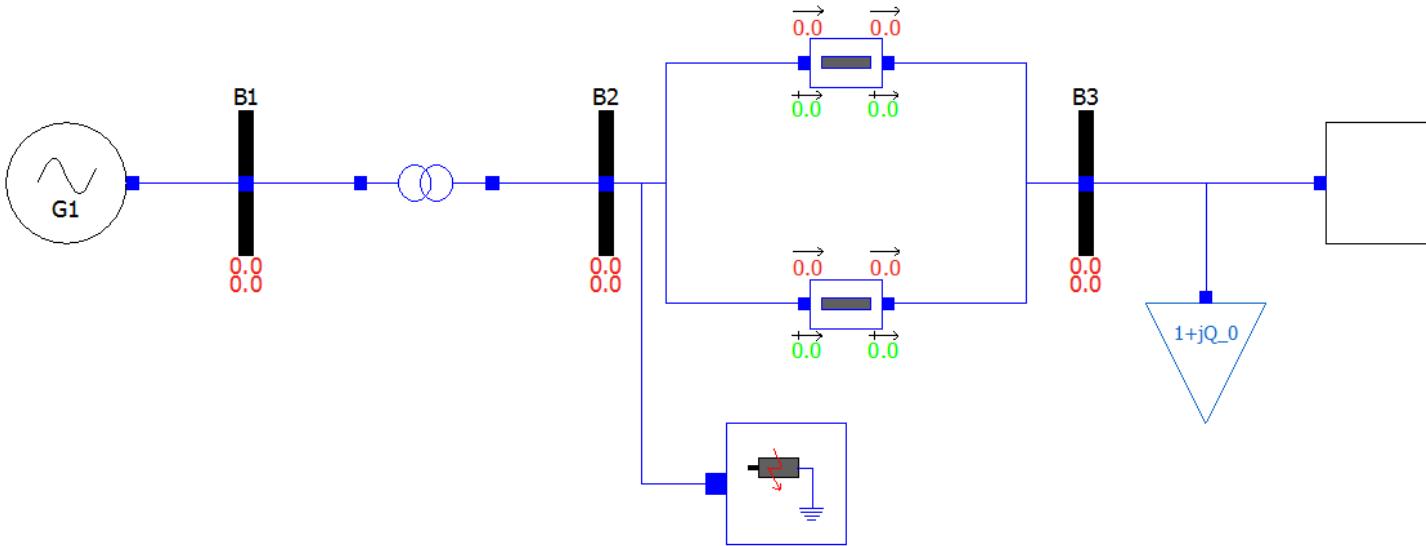


# OMEdit and iPSL

- Now the library is loaded and its structure can be browsed in the Libraries Browser



# Example 1\*



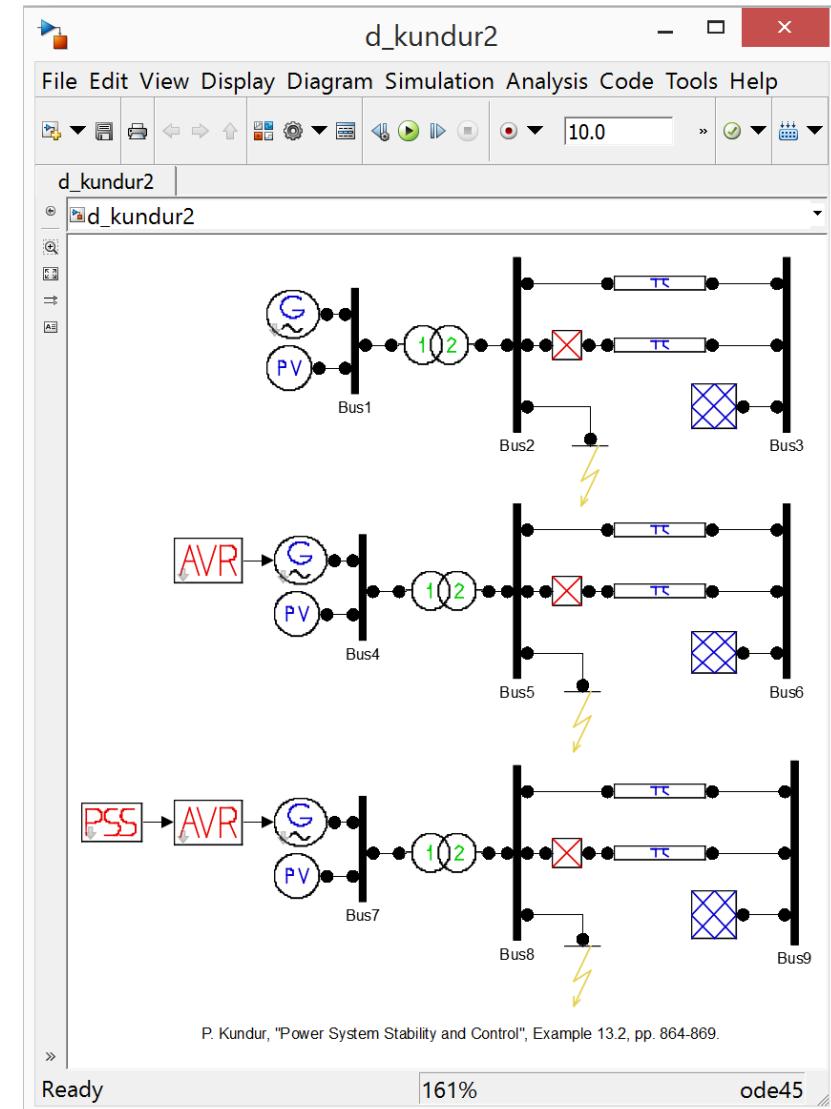
- Single Machine Infinite Bus (SMIB) system
- Analysis of the transient stability of the system including the effects of rotor circuit dynamics and excitation control
- Four machines represented by one connected via transformer and parallel lines to the infinite bus

\* P. Kundur, "Power System Stability and Control", Example 13.2

# Example 1

## Power flow

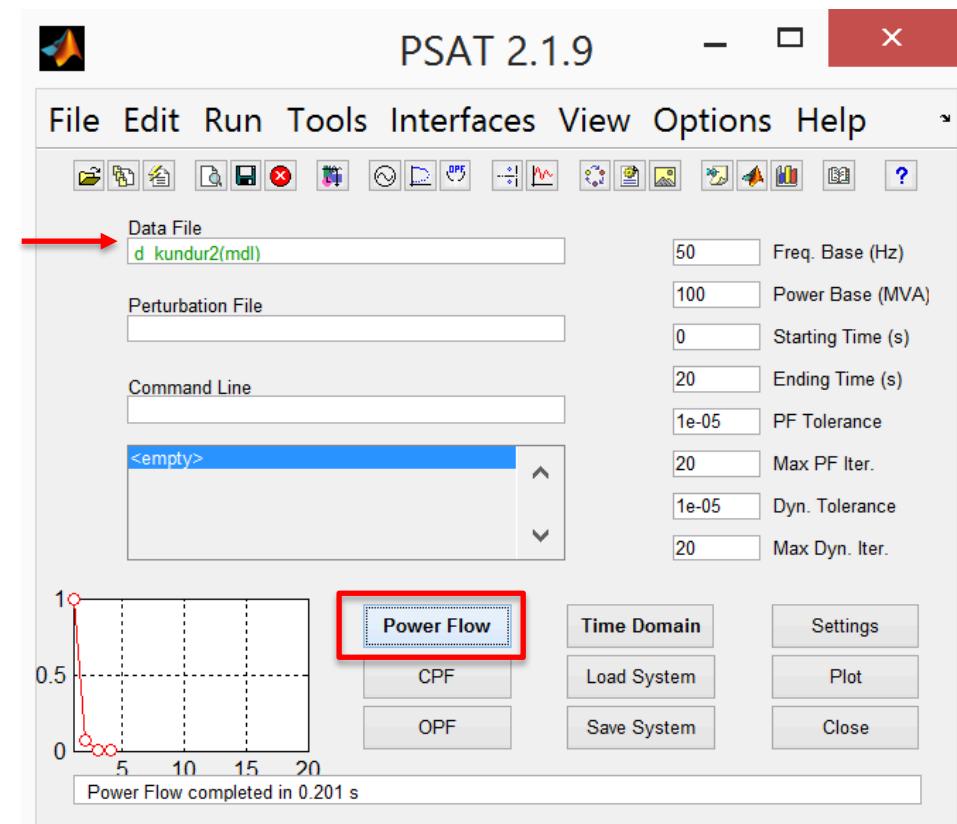
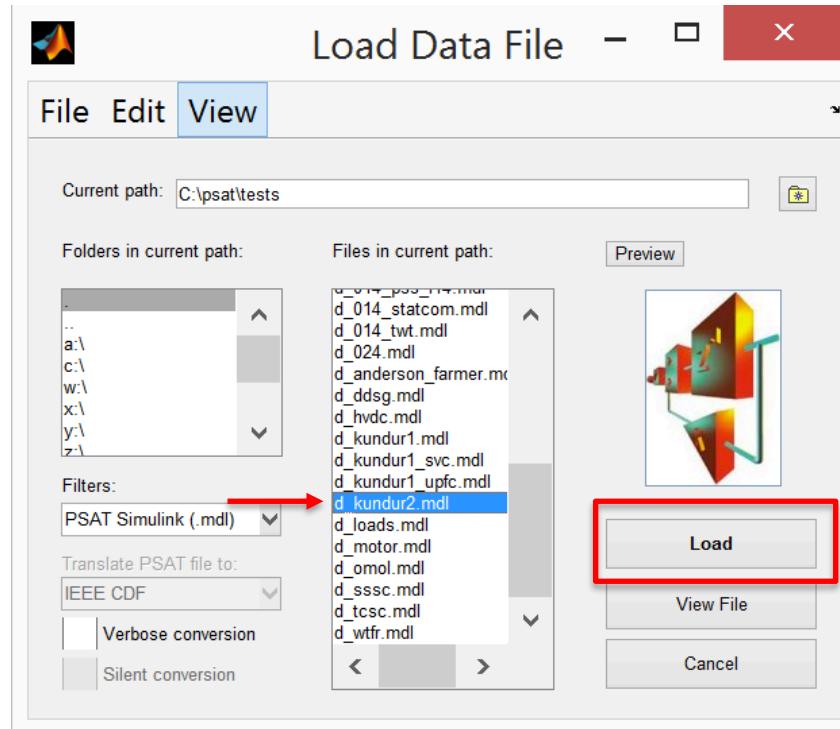
- Power flow results were obtained by PSAT
- Prepared Example 1 already exists in PSAT and can be used for power flow calculations and dynamic simulations



# Example 1

## Power flow

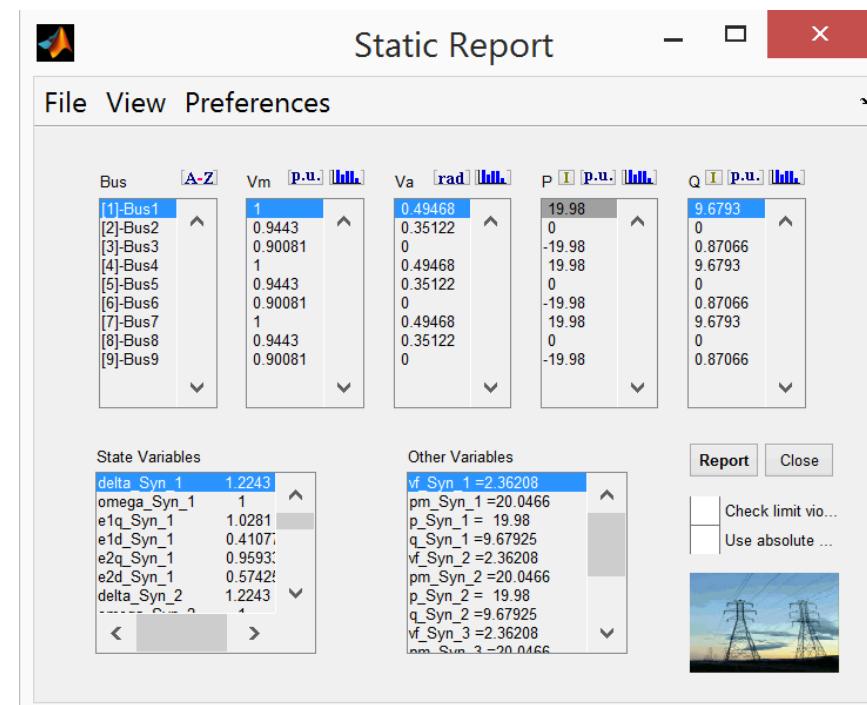
- Example 1 is loaded and the power flow calculations are executed



# Example 1

## Power flow

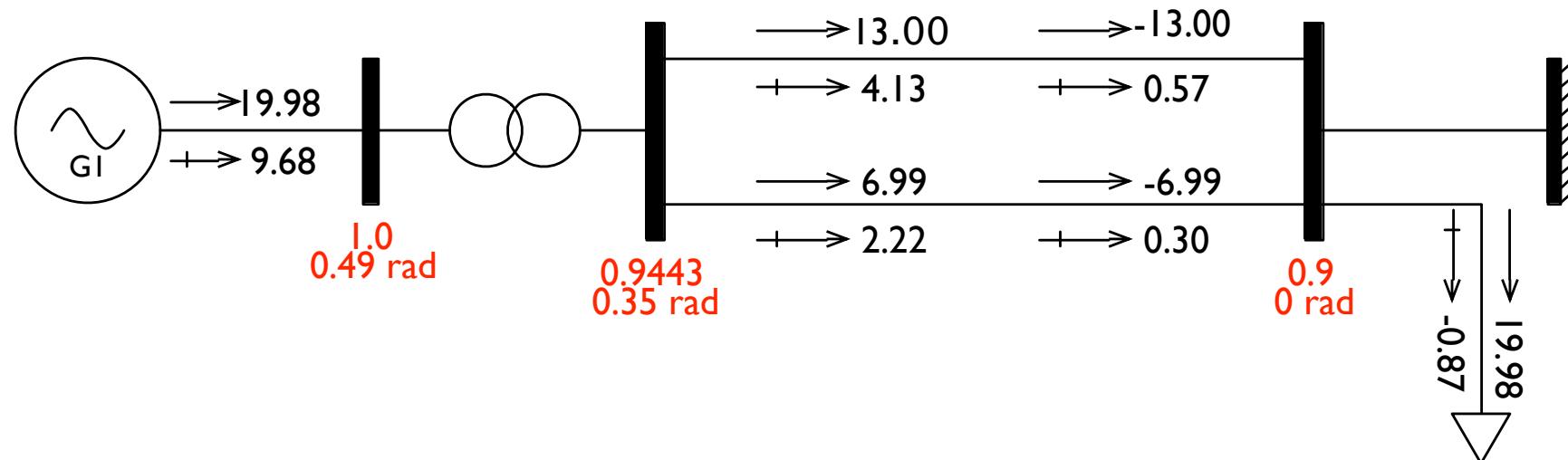
- Static Report can be accessed where all of the power flow results are listed along with the initial values of various state variables of the models
- In this tutorial, there is no need to run the power flow in PSAT since the data will be provided, but feel free to explore PSAT later



# Example 1

## Power flow

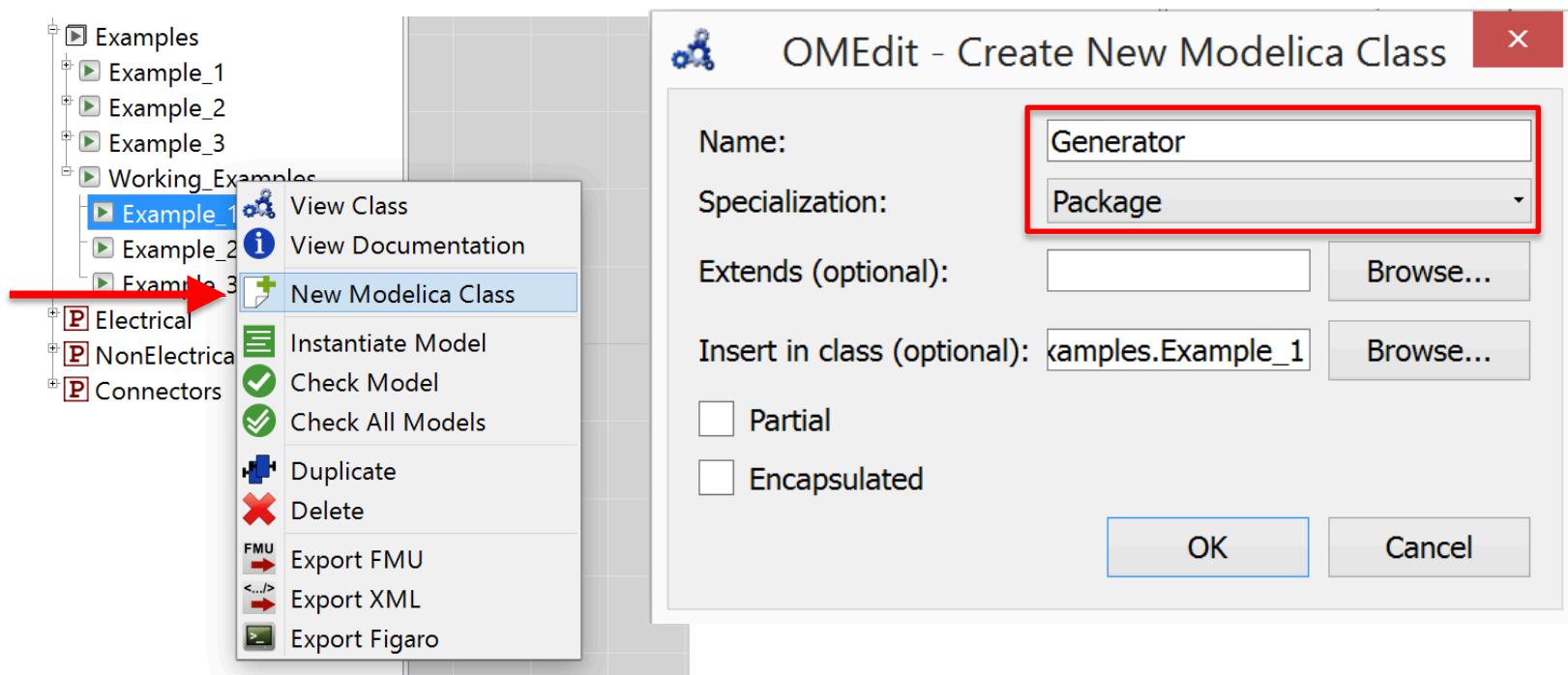
- The summary of all of the relevant data from the power flow is given on the figure below



# Example 1

## Generator model – Step 1

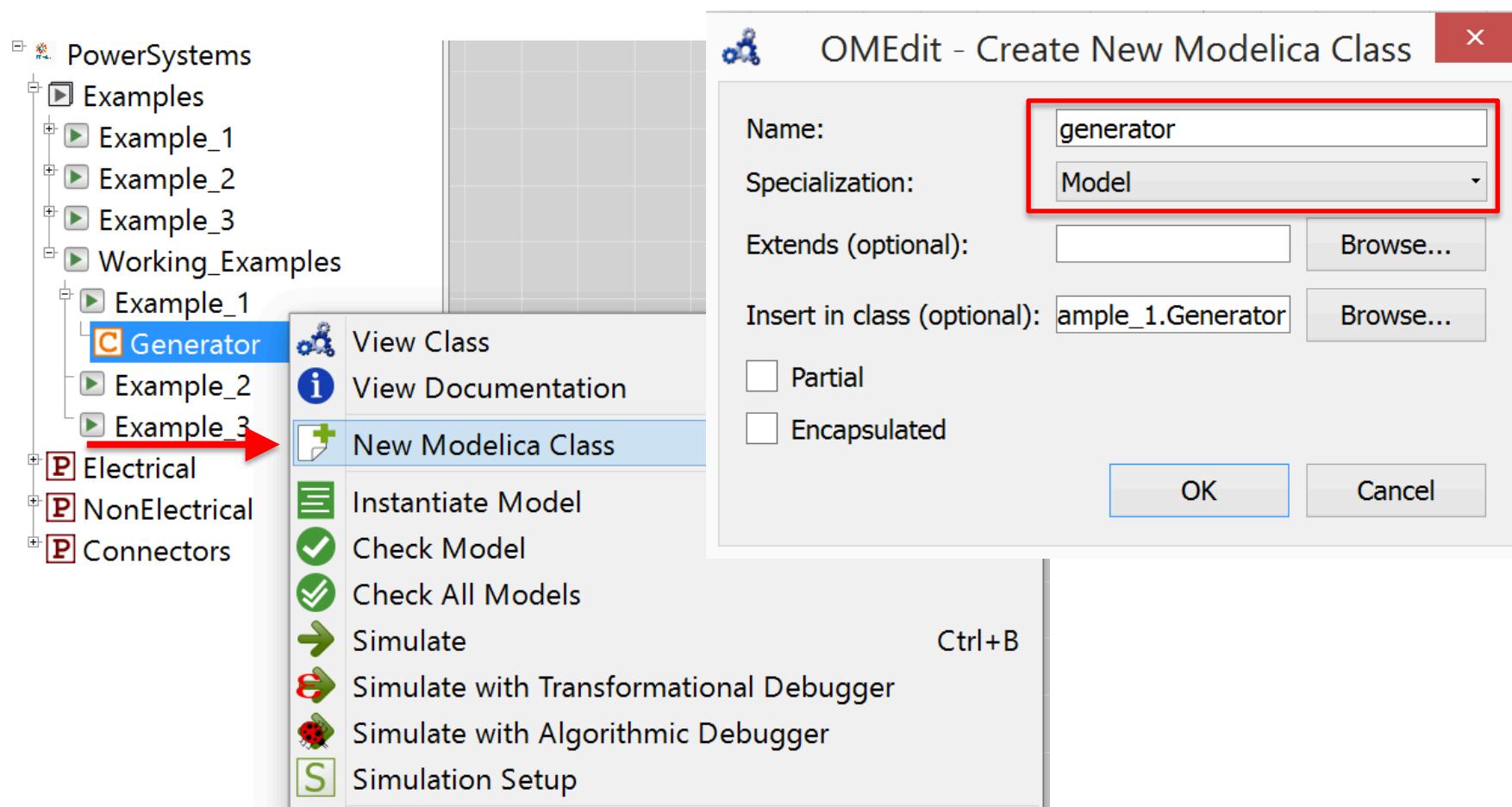
- First, the package where the generator model will be located has to be created
- This is done by right clicking on the *Example\_1* in the *Working\_Examples* package
- The package should be named *Generator*



# Example 1

## Generator model – Step 1

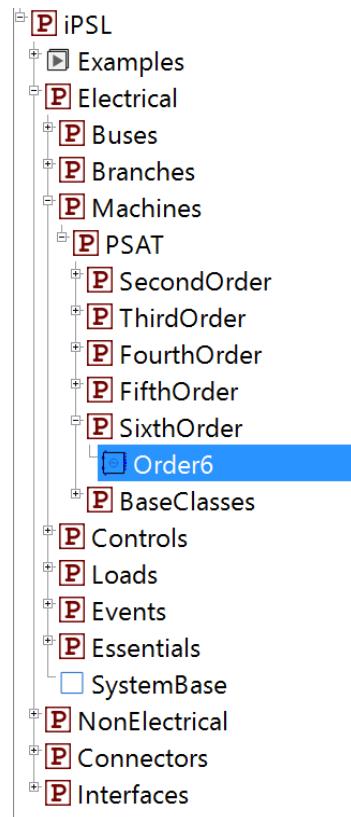
- Within the *Generator* package, model of the generator shall be created



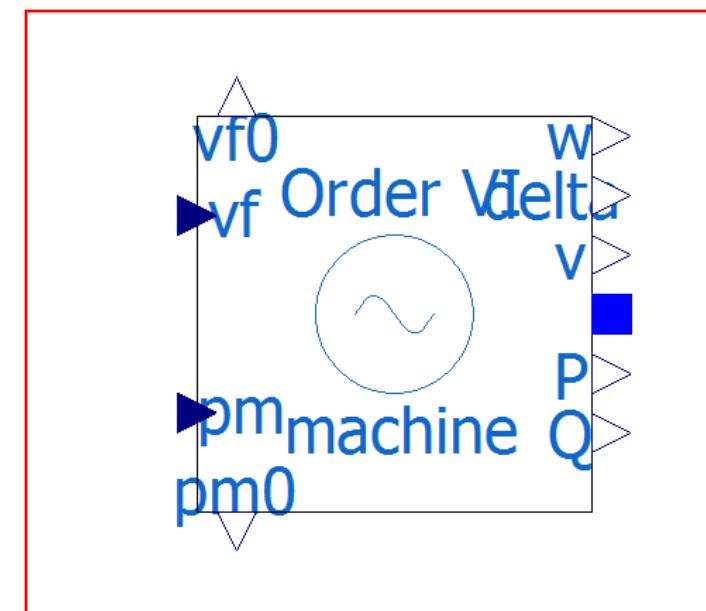
# Example 1

## Generator model – Step 1

- 6<sup>th</sup> order model of the generator from the PSAT is used
- The model is added by dragging the generator from the library and dropping it to the model



Electrical.Machines.PSAT.SixthOrder.Order6



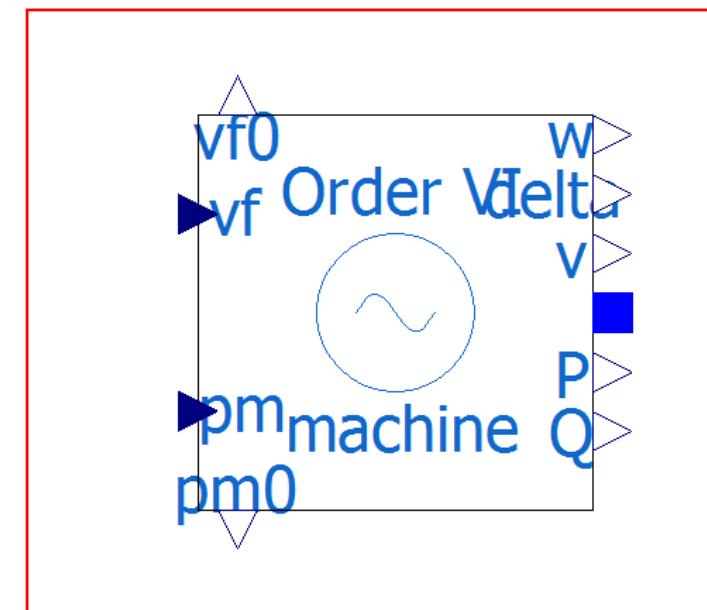
# Example 1

## Generator model – Step 1

- Parameters of the generator are given in the table

$S_n$	2220	$x''_q$	0.25
$V_n$	400	$T'_{d,0}$	8
$r_a$	0.003	$T'_{q,0}$	1
$x_d$	1.81	$T''_{d,0}$	0.03
$x_q$	1.76	$T''_{d,0}$	0.07
$x'_d$	0.3	$T_{aa}$	0.002
$x'_q$	0.65	$M$	7
$x''_d$	0.23	$D$	0

Electrical.Machines.PSAT.SixthOrder.Order6



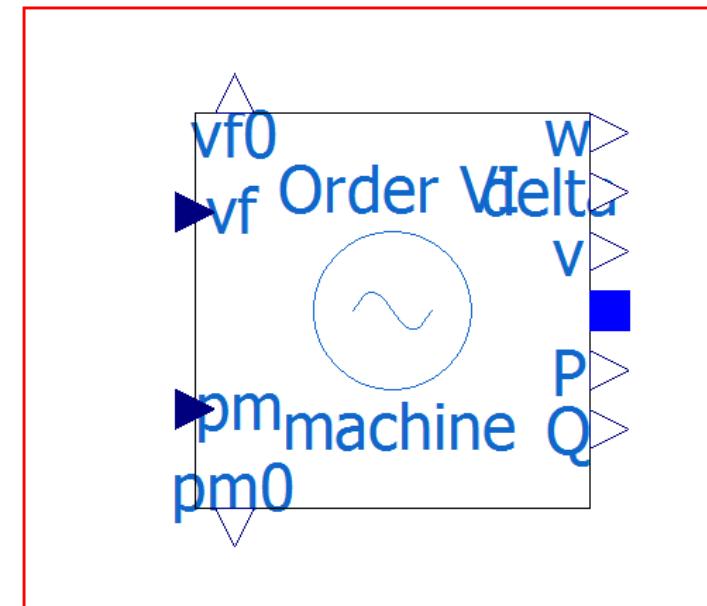
# Example 1

## Generator model – Step 1

- Power flow results:

$V_0$	1
$angle_0$	0.4946
$P_0$	19.9799
$Q_0$	9.6792
$V_b$	400
$S_b$	Do not edit
$f_n$	Do not edit

Electrical.Machines.PSAT.SixthOrder.Order6

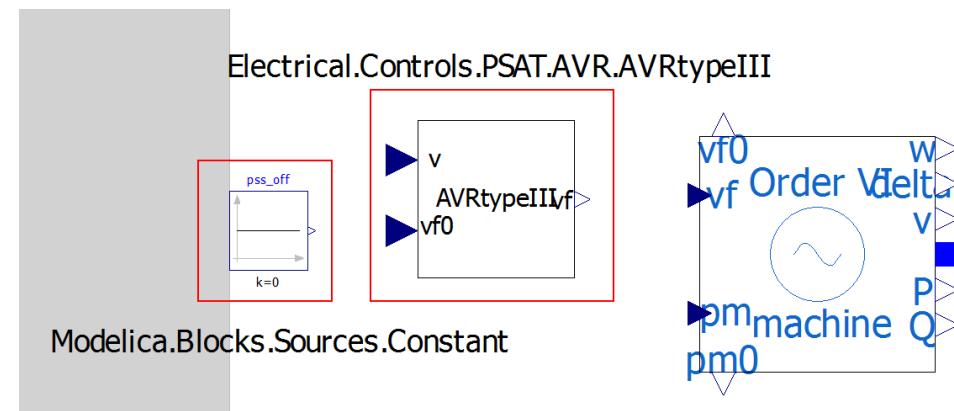


# Example 1

## Generator model – Step 2

- PSAT model of the AVR Type III is used
- Constant block `pss_off` will be used as a zero input to the PSS input signal of the AVR since the PSS is not used
- Parameters:

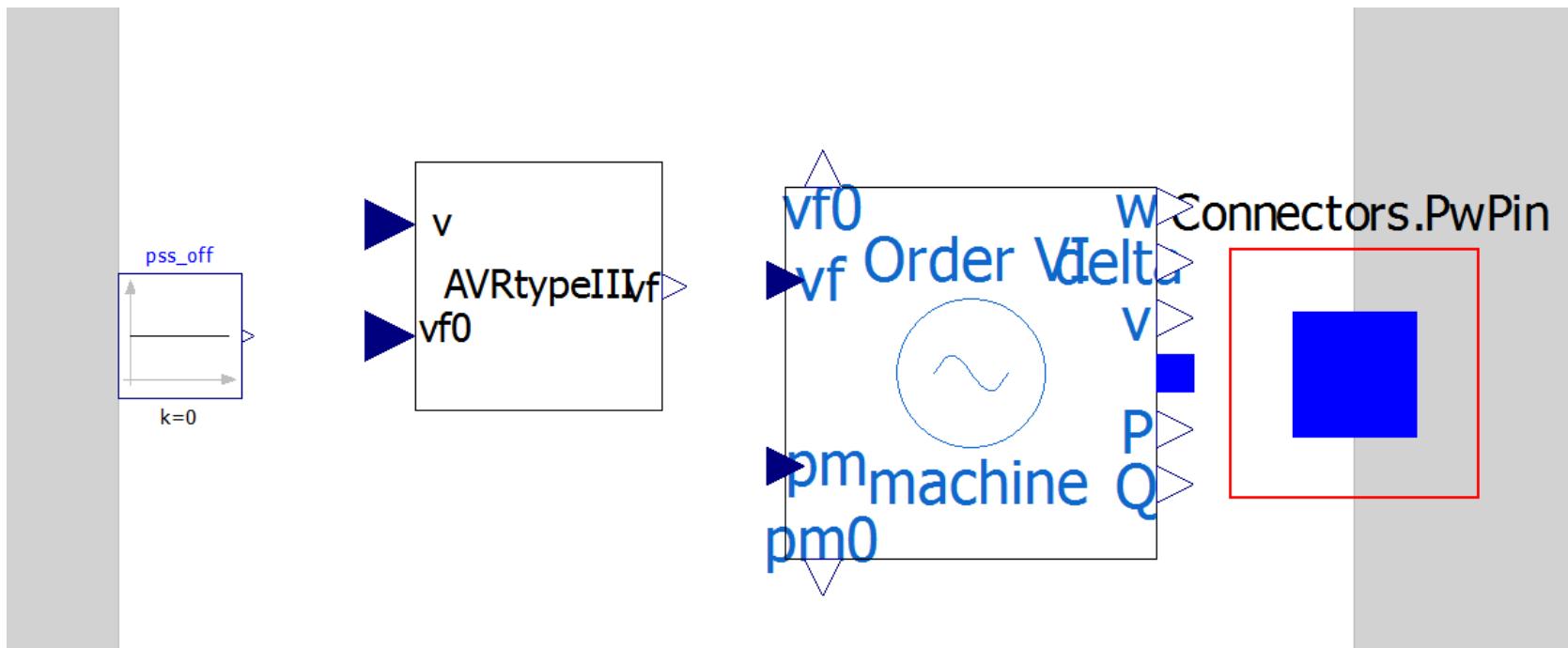
$v_{ref}$	1	$T_1$	1
$v_0$	1	$T_e$	0.0001
$v_{f,max}$	7	$v_{f,0}$	2.42
$v_{f,min}$	-6.4	$T_r$	0.015
$K_0$	200	$s_0$	0
$T_2$	1		



# Example 1

## Generator model – Step 3

- Additional PwPin will be used to interface generator model's voltages and currents with other components of the network model



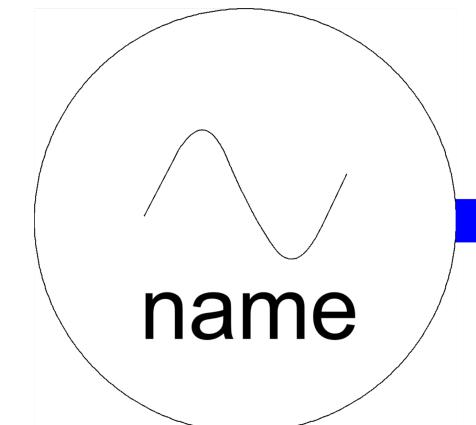
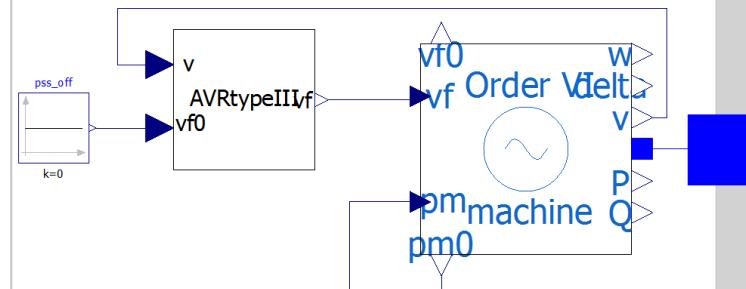
# Example 1

## Generator model – Step 4

- To finish the generator model, different signals need to be connected
- Optionally, icon of the generator model can be altered

### Step 4: Connecting model's signals

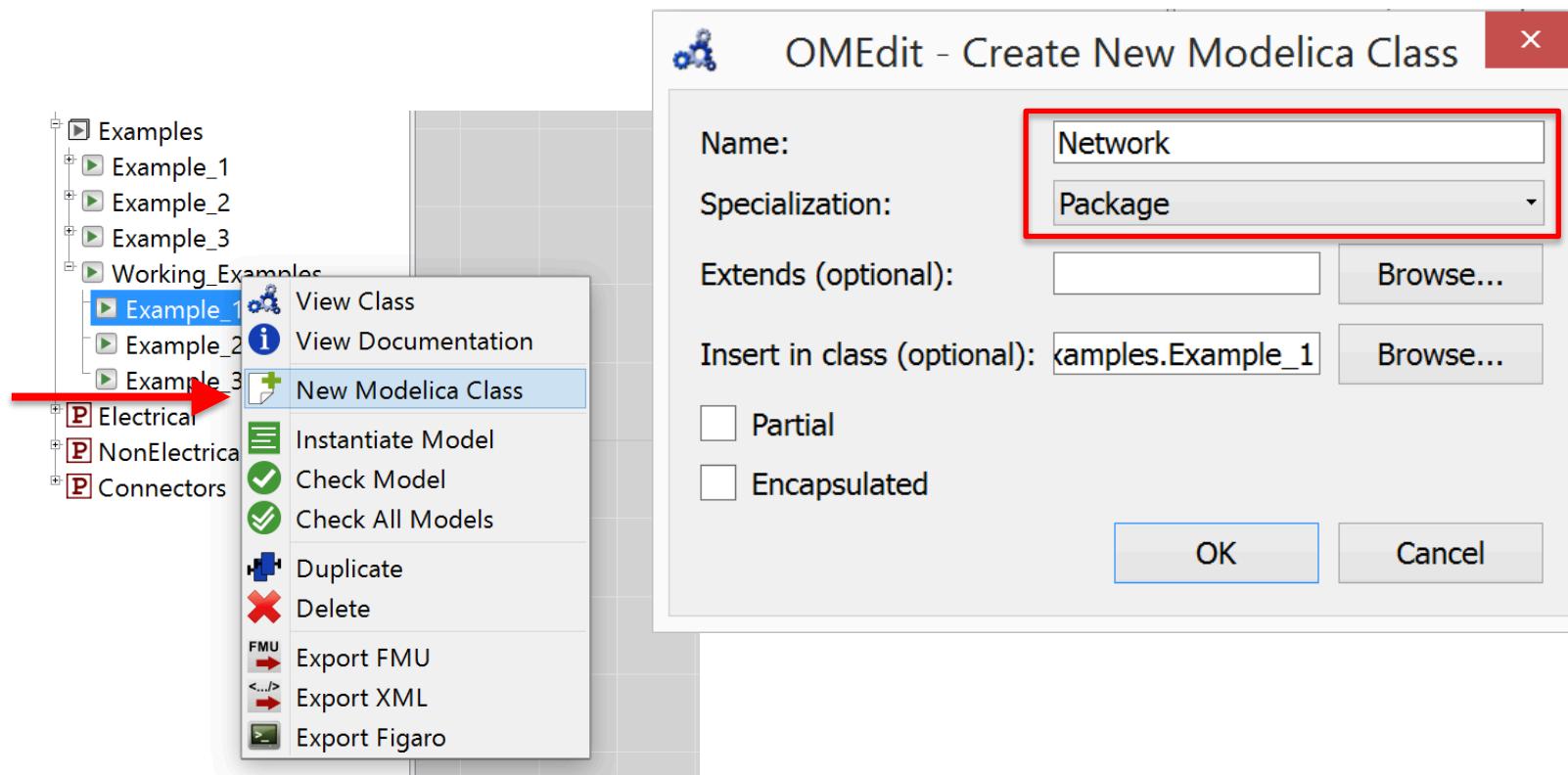
- Machine's terminal voltage to AVR's input signal
- AVR's output field voltage to machine's input field voltage
- Initially calculated mechanical power to input signal of the machine's mechanical power
- Machine's power terminal to the generator model power terminal
- Constant pss\_off to the PSS input at the AVR



# Example 1

## Network model – Step 1

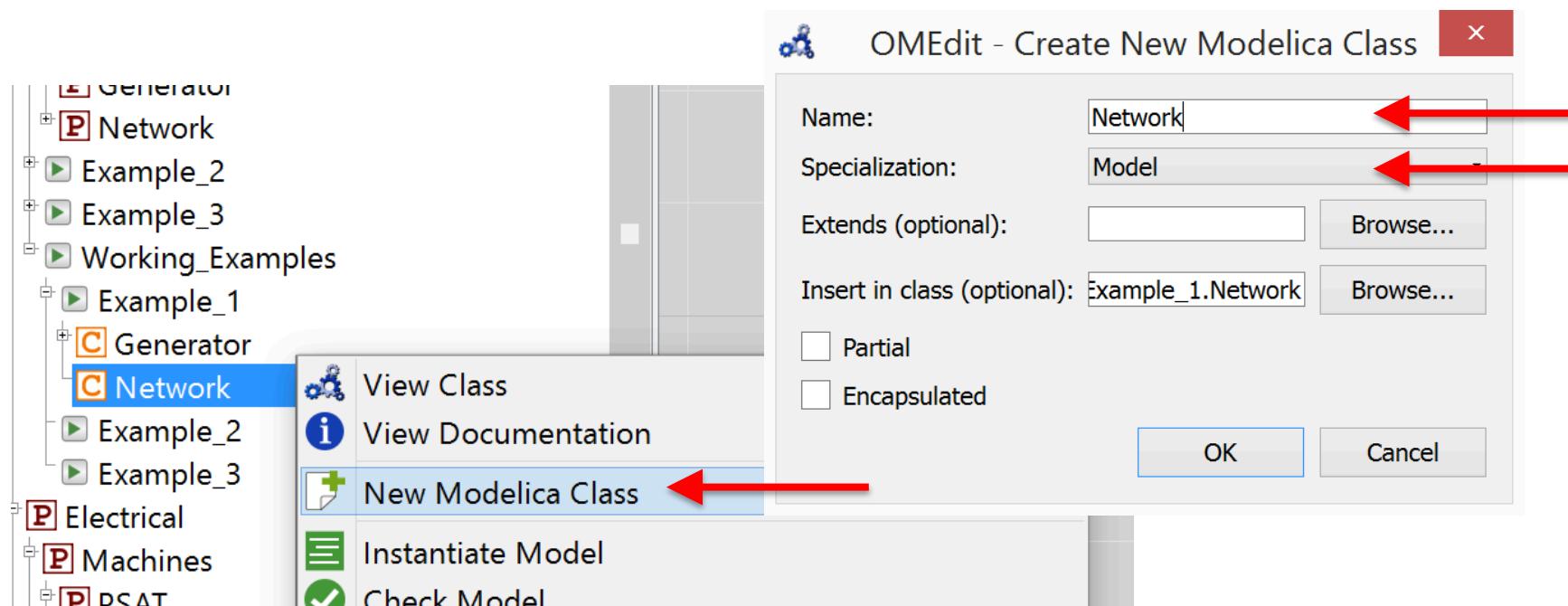
- Network package will be created in the *Example\_1* package
- This package is created by right clicking on the *Example\_1* in the *Working\_Examples* package



# Example 1

## Network model – Step 1

- Network model will be created in the *Network* package
- This package is created by right clicking on the *Network* package
- The name of the network model will be *Example\_1*

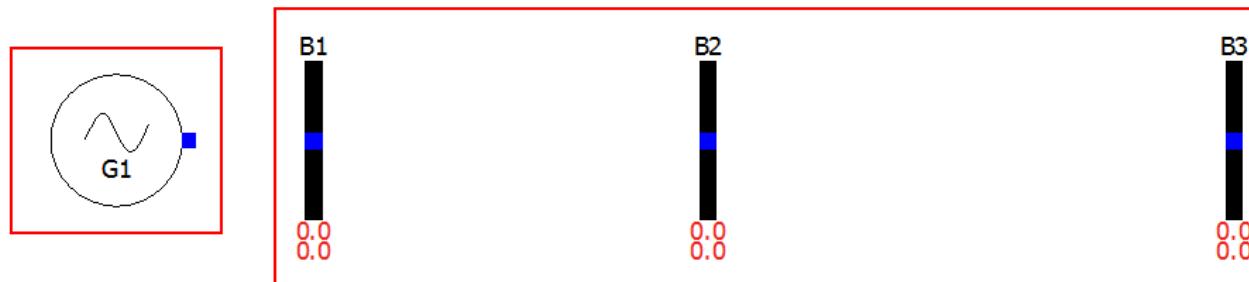


# Example 1

## Network model – Step 1

- Created generator model and three bus models are added to the network model

Electrical.Buses.Bus



- Also, model iPSL.Electrical.SystemBase shall be added to the network model which defines base parameters for all of the components in the network model

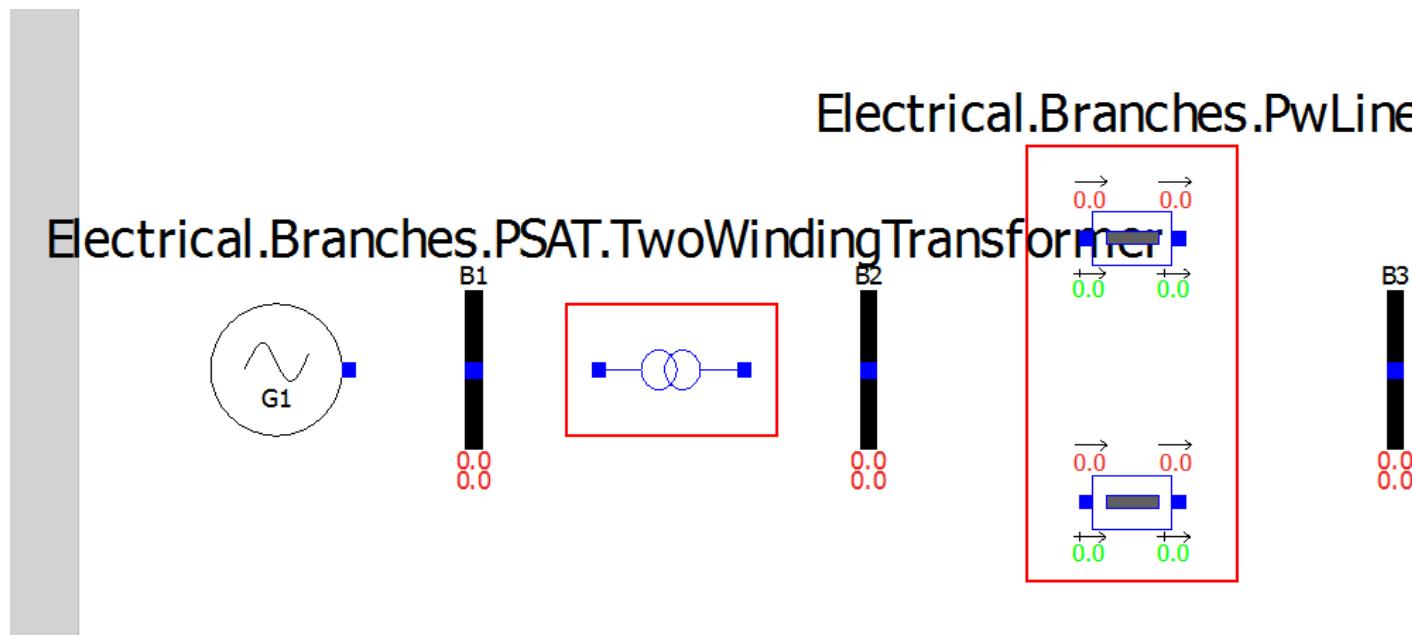
$S_b$	100
$f_n$	60

System Data  
System Base MVA  
Frequency 0.0 Hz

# Example 1

## Network model – Step 2

- Transformer and line models are added



# Example 1

## Network model – Step 2

- Transformer and line parameters

Transformer

$S_b$	100	$f_n$	60
$S_n$	2220	$kT$	1
$V_b$	60	$x$	0.15
$V_n$	0.001	$r$	0

Line 1

$R$	0.0	$G$	0.0
$X$	$0.5*100/2220$	$B$	0.0
$S_b$	100		

Line 2

$R$	0.0	$G$	0.0
$X$	$0.93*100/2220$	$B$	0.0
$S_b$	100		

# Example 1

## Network model – Step 3

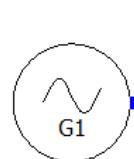
- Infinite bus and load models are added

Load

$S_b$	100	$P_0$	19.9799
$S_n$	100	$Q_0$	-0.8706

Infinite bus

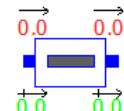
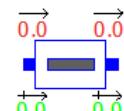
$V$	0.90081	$angle$	0
-----	---------	---------	---



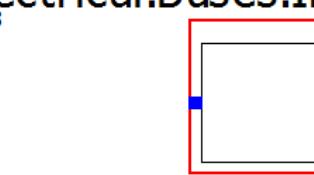
B1  
0.0  
0.0



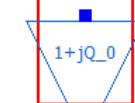
B2  
0.0  
0.0



Electrical.Buses.InfiniteBus



Electrical.Loads.PSAT.LOADPQ



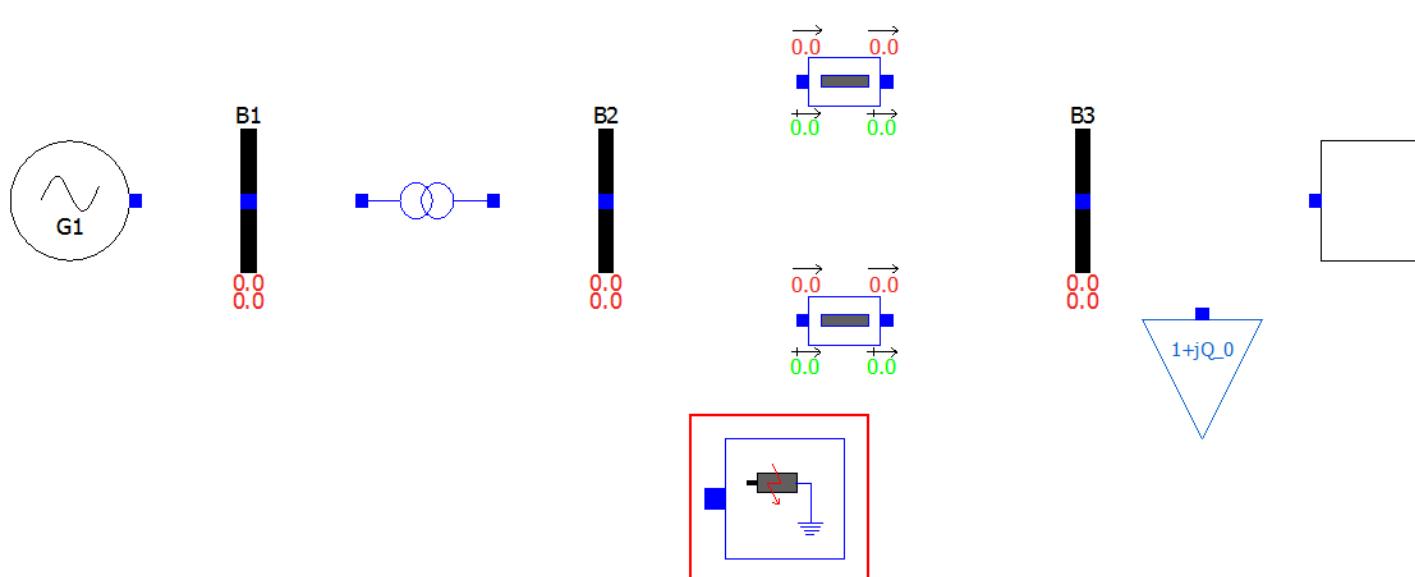
# Example 1

## Network model – Step 4

- 3-phase-to-ground fault is added

Fault

$R$	0	$t_1$	0.5
$X$	$0.01 * 100 / 2220$	$t_2$	0.57

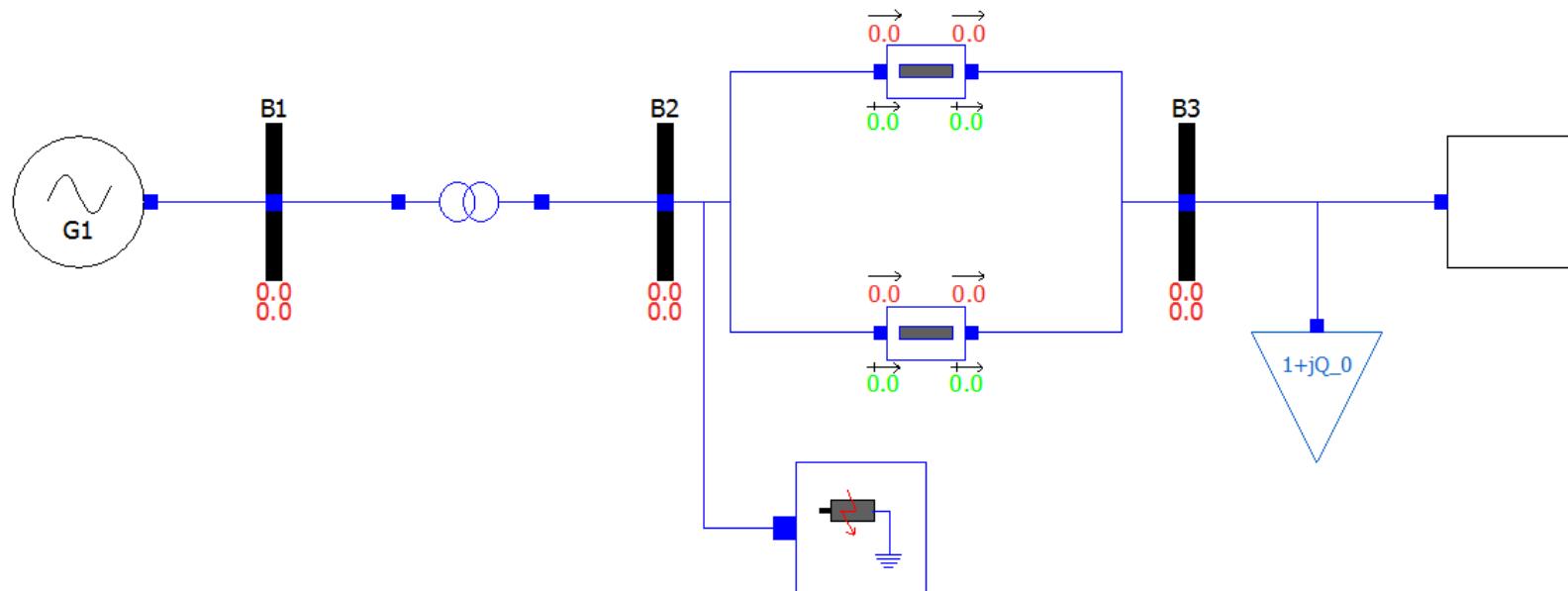


Electrical.Events.PwFault

# Example 1

## Network model – Step 5

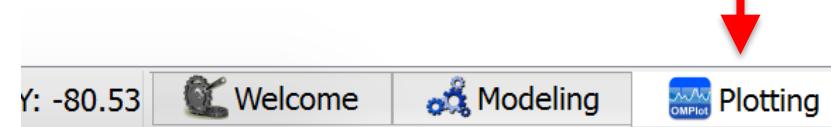
- The network model is completed by connecting all of the components
- Now, the model can be simulated and linearized



# Example 1

## Simulation

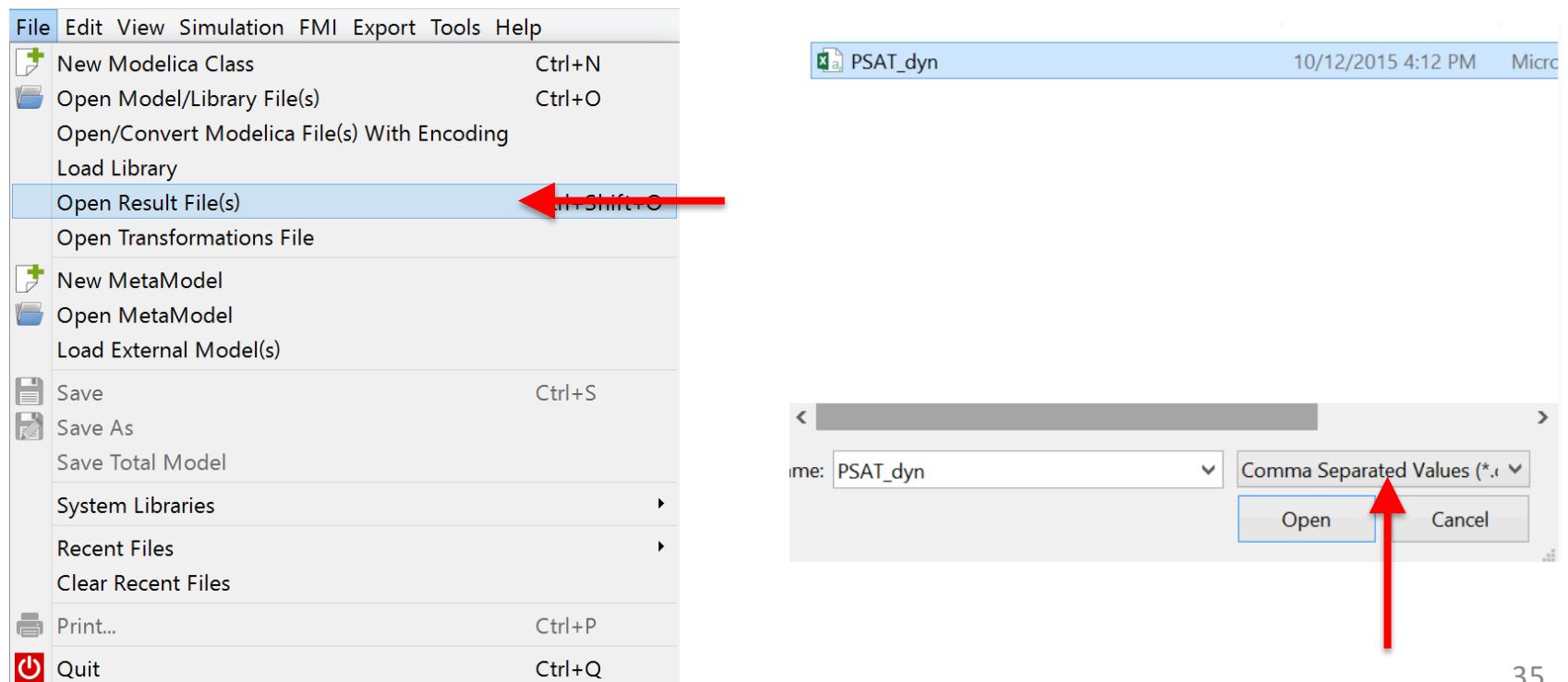
- System will be simulated with 3-phase-to-ground fault at  $t=0.5\text{s}$  with a duration of 70ms
- Simulation results will be compared with the reference results from the PSAT that will be loaded first
- PSAT results are provided in a file “PSAT\_dyn.csv”
- To load the file, the view should be switched to “Plotting” tab



# Example 1

## Simulation

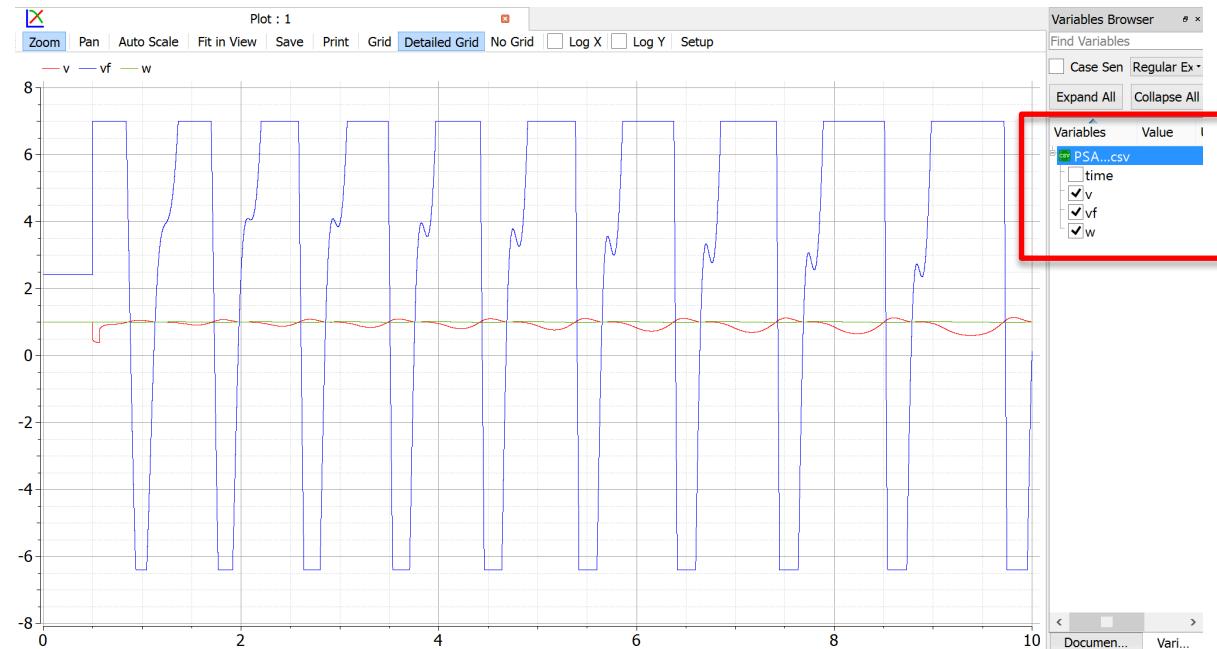
- Result file can be opened by navigating the menu to File->Open Result File(s)
- In the pop-up menu, one has to select “Comma Separated Values” as a file type, navigate to the directory where the file is located and open it



# Example 1

## Simulation

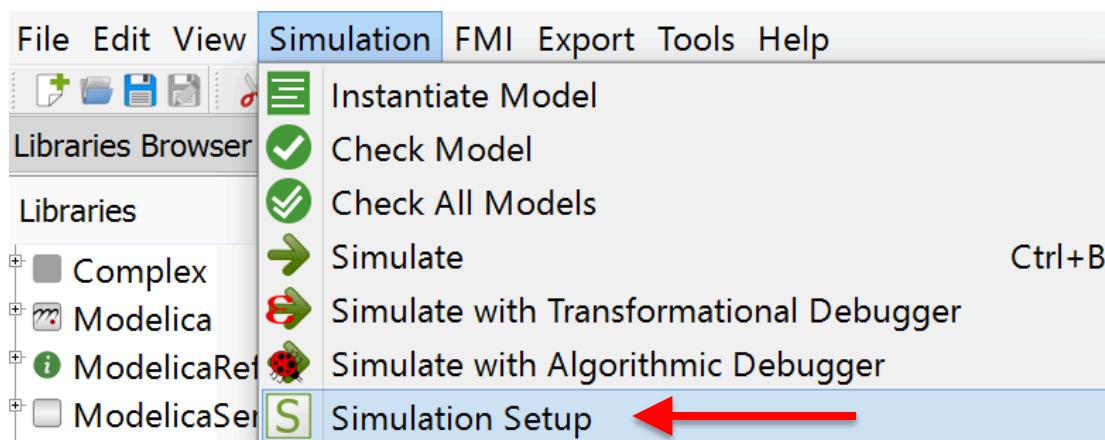
- In the variable browser, three waveforms from the PSAT results are loaded which can be displayed on the plot as it is shown in the figure below
- Loaded waveforms are generator terminal voltage, excitation field voltage and the generator speed



# Example 1

## Simulation

- Before the simulation, solver and its parameters are set to be the same as in the PSAT
- Solver is chosen to be Runge-Kutta 2 with a fixed step
- More solvers can be chosen in Modelica (depending on the tool), however, to match the model's response with the one in PSAT choice of the solver is limited

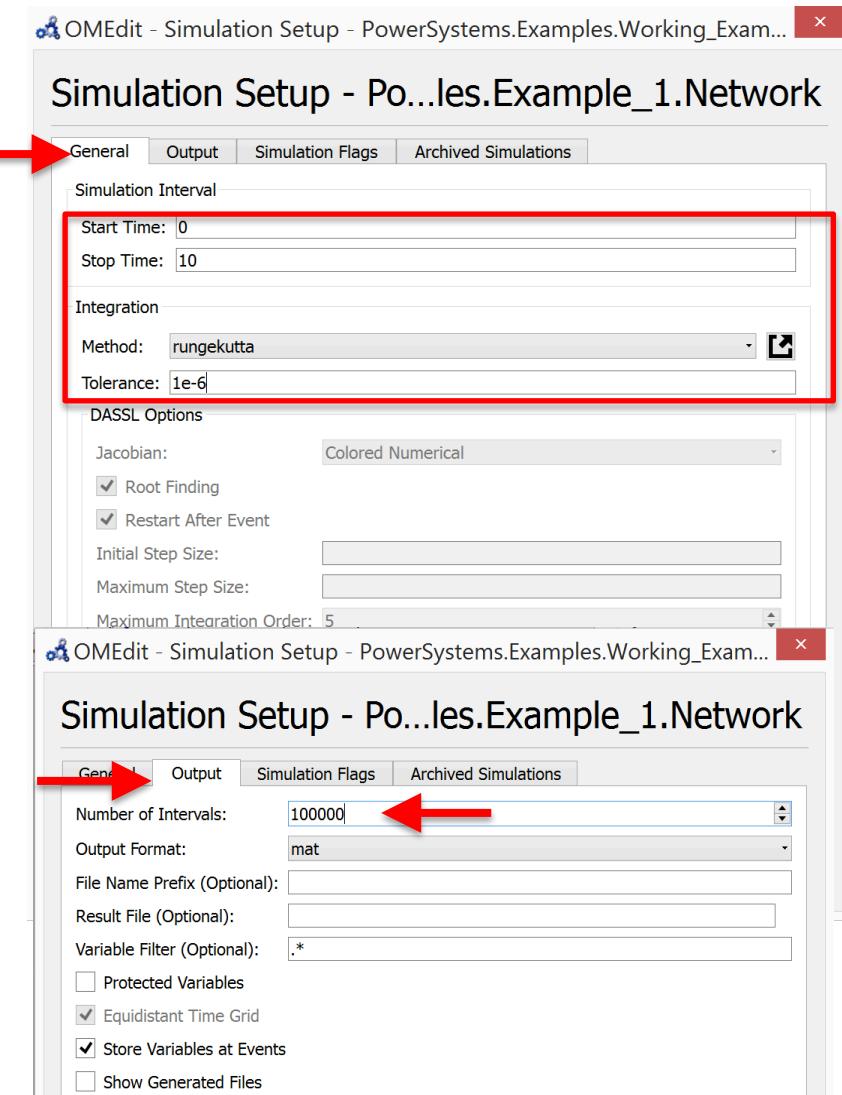


# Example 1

## Simulation

- Simulation time is set to 10s and the tolerance of the solver is set to 1e-6
- The size of the time step is set by the number of intervals
- Number of intervals:

$$n_{int} = \frac{\text{Simulation Time}}{\text{Time Step}} = \frac{10}{1e^{-4}} = 10^5$$



The image shows two side-by-side screenshots of the OMEdit software's "Simulation Setup" dialog for a PowerSystems example. Both screenshots focus on the "General" tab.

**Screenshot 1 (Top):** This screenshot shows the "Simulation Interval" section. It includes fields for "Start Time" (0), "Stop Time" (10), "Integration Method" (set to "rungekutta"), and "Tolerance" (set to "1e-6"). A red arrow points to the "General" tab at the top of the dialog.

Setting	Value
Start Time	0
Stop Time	10
Integration Method	rungekutta
Tolerance	1e-6

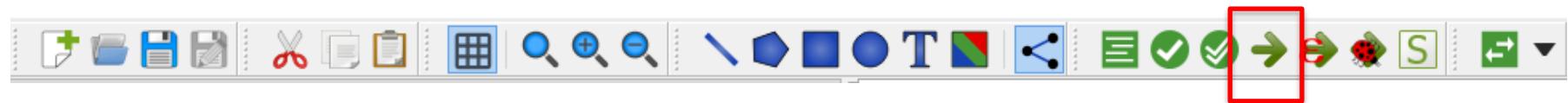
**Screenshot 2 (Bottom):** This screenshot shows the "Number of Intervals" field in the "Output" section, which is set to "100000". A red arrow points to the "General" tab at the top of the dialog.

Setting	Value
Number of Intervals	100000
Output Format	mat
File Name Prefix (Optional)	
Result File (Optional)	
Variable Filter (Optional)	*
Protected Variables	<input type="checkbox"/>
Equidistant Time Grid	<input checked="" type="checkbox"/>
Store Variables at Events	<input checked="" type="checkbox"/>
Show Generated Files	<input type="checkbox"/>

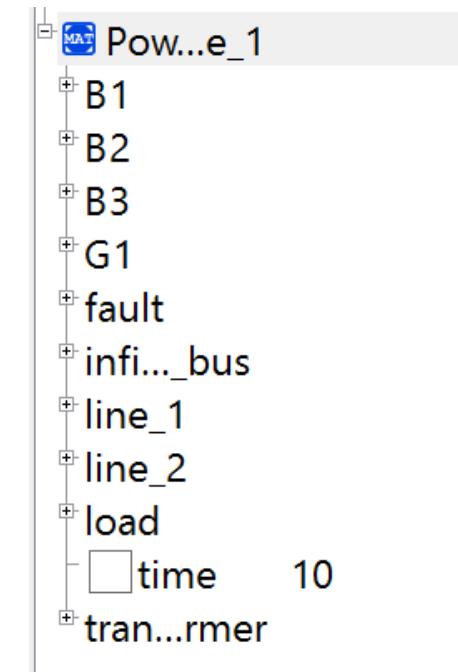
# Example 1

## Simulation

- By pressing the “Simulate” button on the toolbar, simulation of the model is executed



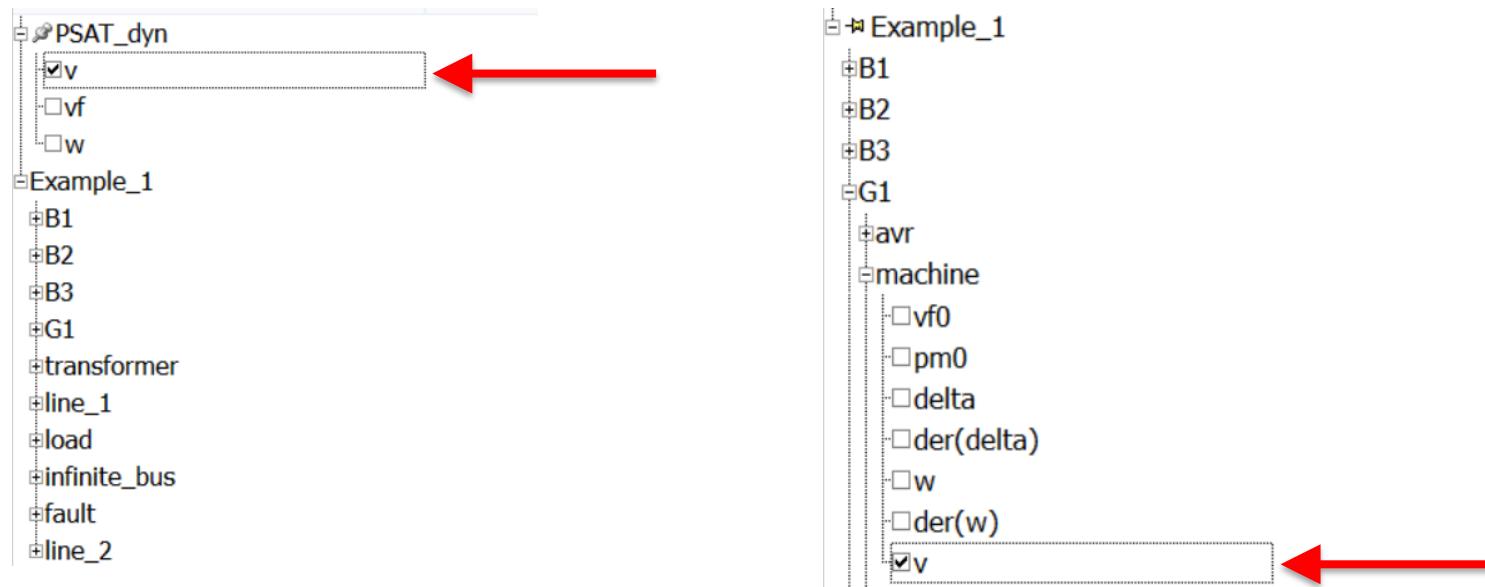
- Once the simulation is completed, Variable Browser is populated with the simulation results



# Example 1

## Simulation

- To display the simulation results or compare it with the results from PSAT, one can mark the check-box next to the variable which will be shown on the plot
- To show the terminal voltage of the generator in PSAT and modelica, variables “PSAT\_dyn.v” and “Example\_1.G1.machine.v” have to be selected



# Example 1

## Simulation

- To display the simulation results or compare it with the results from PSAT, one can mark the check-box next to the variable which will be shown on the plot
- To show the terminal voltage of the generator in PSAT and modelica, variables “PSAT\_dyn.csv.v” and “G1.machine.v” have to be selected

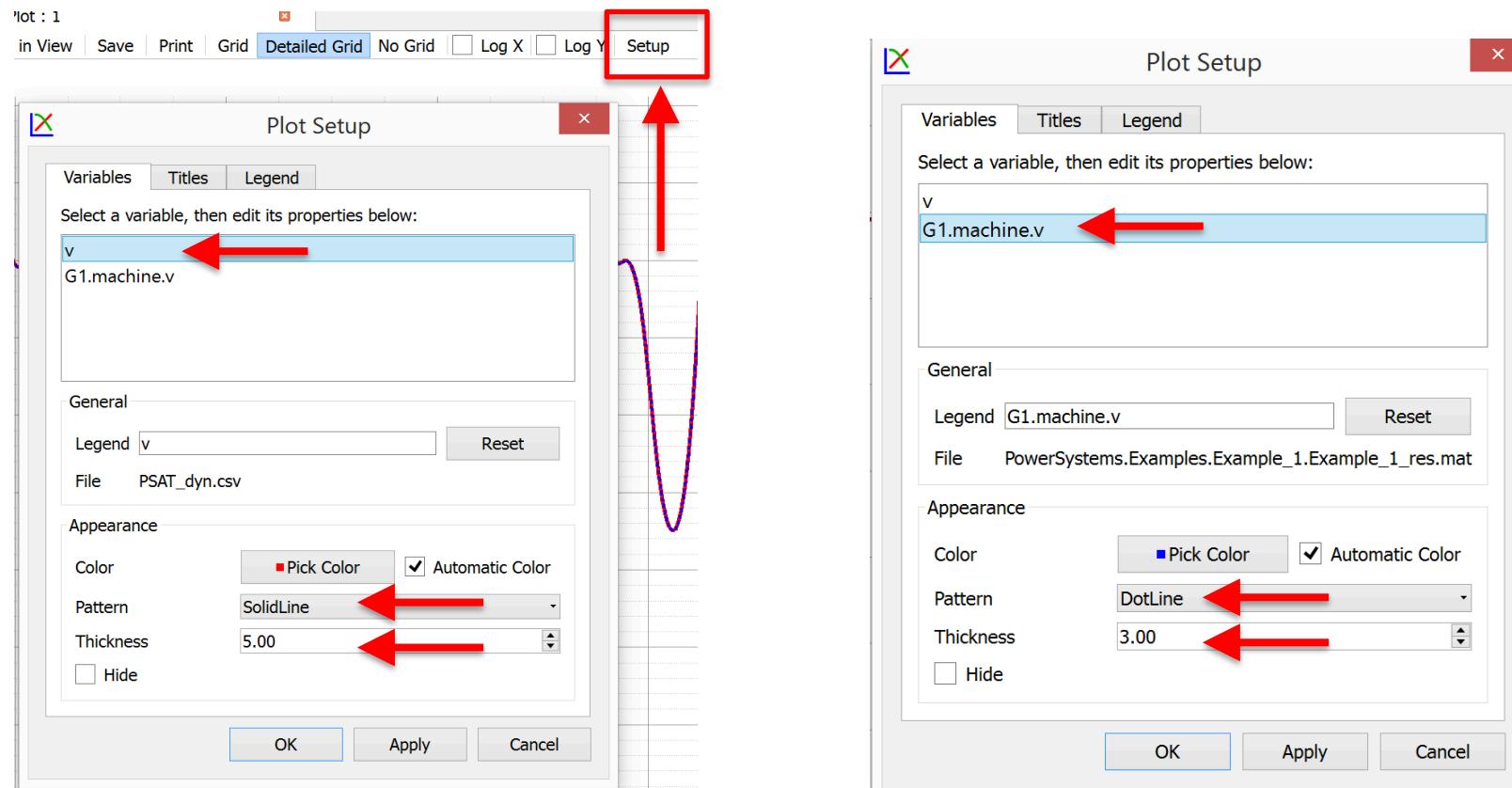


PowerSystems....e_1.Example_1	
+ B1	
+ B2	
+ B3	
+ G1	
+ avr	
+ machine	
<input type="checkbox"/> CoB	0.0...045
<input type="checkbox"/> CoB2	0.0...045
<input checked="" type="checkbox"/> v	1.0054
<input type="checkbox"/> vd	0.0...647
<input type="checkbox"/> vd0	0.6...549
<input type="checkbox"/> vf	-0....328
<input type="checkbox"/> vf0	2.42444
<input type="checkbox"/> vf00	1.08882

# Example 1

## Simulation

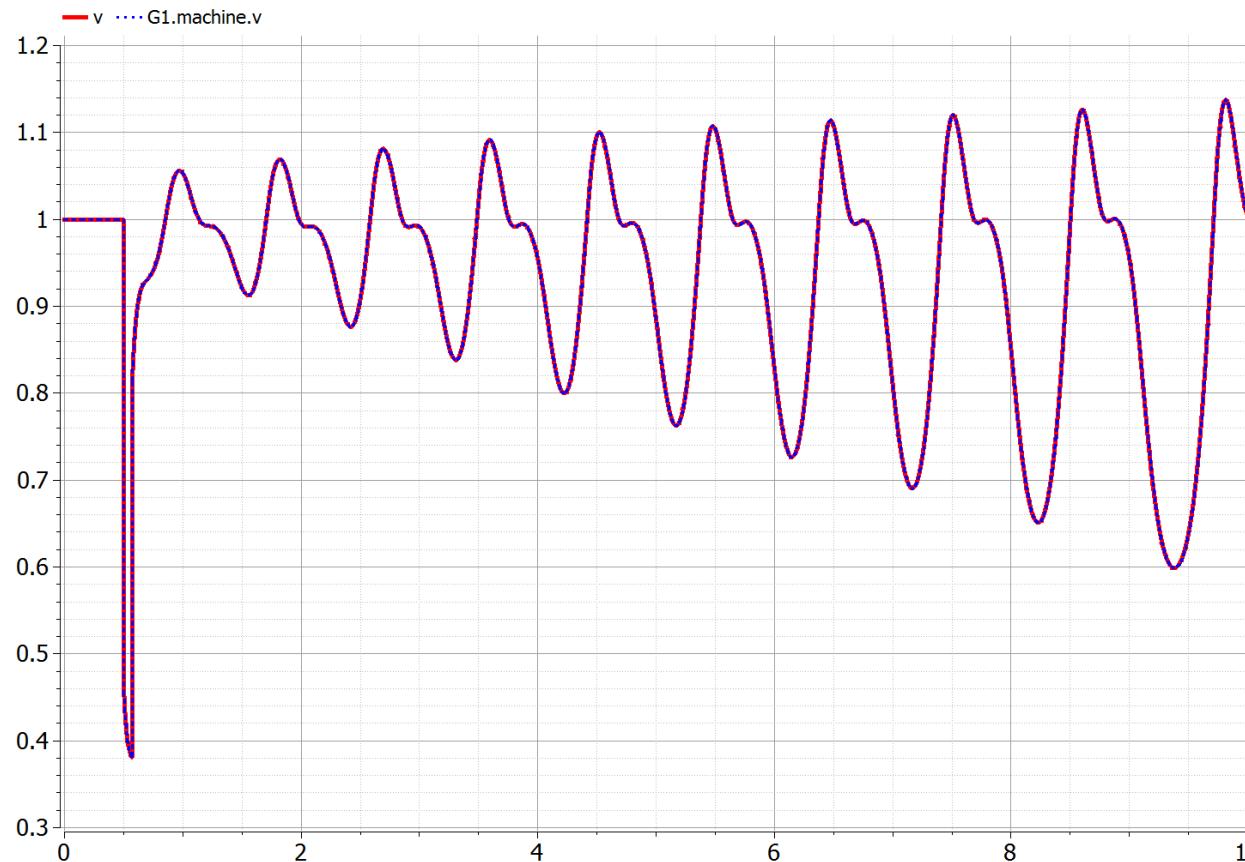
- To be able to distinguish different signals, one should adjust the thickness and the pattern of the signal line



# Example 1

## Simulation

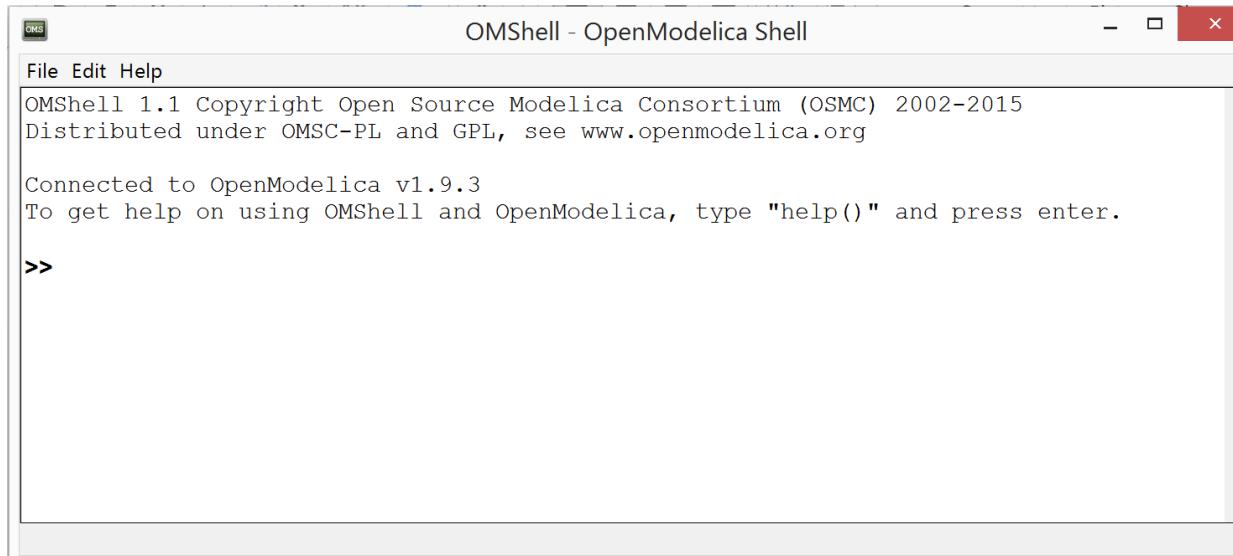
- Previous steps produce the plot shown in the figure below showing that the Modelica produces the same simulation results as the PSAT does



# Example 1

## Linearization

- To linearize the system, OpenModelica scripting will be needed
- Along with the library, script `linearize_iPSL.mos` was provided which performs the linearization procedure and provides the A matrix from the linearized state-space model
- The scripting shall be done in the command line interface of OpenModelica called OMShell



The screenshot shows a Windows-style application window titled "OMShell - OpenModelica Shell". The menu bar includes "File", "Edit", and "Help". The main window displays the following text:  
OMShell 1.1 Copyright Open Source Modelica Consortium (OSMC) 2002-2015  
Distributed under OMSC-PL and GPL, see [www.openmodelica.org](http://www.openmodelica.org)  
Connected to OpenModelica v1.9.3  
To get help on using OMShell and OpenModelica, type "help()" and press enter.  
>>

# Example 1

## Linearization

- First, open the script `linearize_iPSL.mos` with a text editor and make sure that the path in the second line points to the iPSL distribution on your hard drive

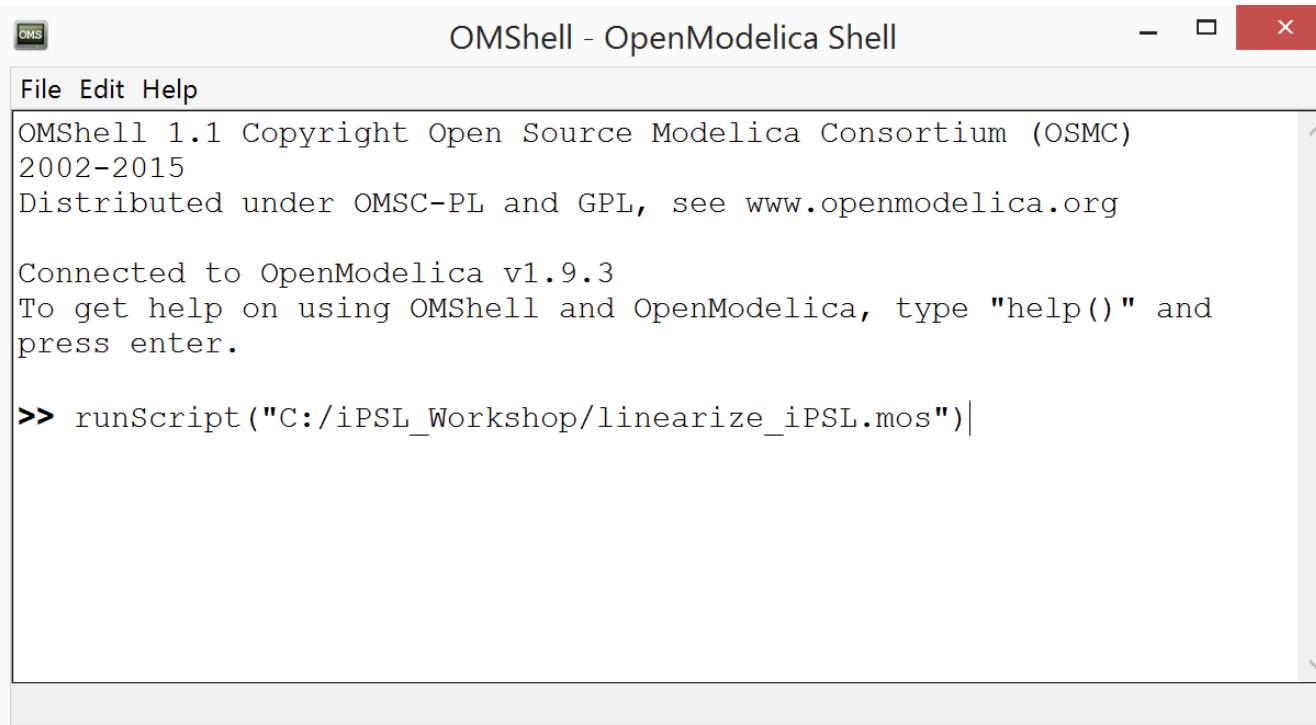
```
1 loadModel(Modelica);
2 loadFile("C:/iPSL_Workshop/iPSL_stripped.mo");
3 instantiateModel(iPSL;
4 instantiateModel(iPSL.Examples.Example_1.Example_1);
5
6 linearize(iPSL.Examples.Example_1.Example_1,stopTime=0.0);
7
8 loadFile("linear_iPSL.Examples.Example_1.Example_1.mo");
9
10 (a) := getParameterValue(linear_iPSL_Examples_Examp..._1.Example_1,"A");
```

# Example 1

## Linearization

- To linearize system, Modelica script `linearize_iPSL.mos` is run by executing a command

```
runScript("C:/iPSL_Workshop/linearize_iPSL.mos")
```



OMShell - OpenModelica Shell

File Edit Help

```
OMShell 1.1 Copyright Open Source Modelica Consortium (OSMC)
2002-2015
Distributed under OMSC-PL and GPL, see www.openmodelica.org

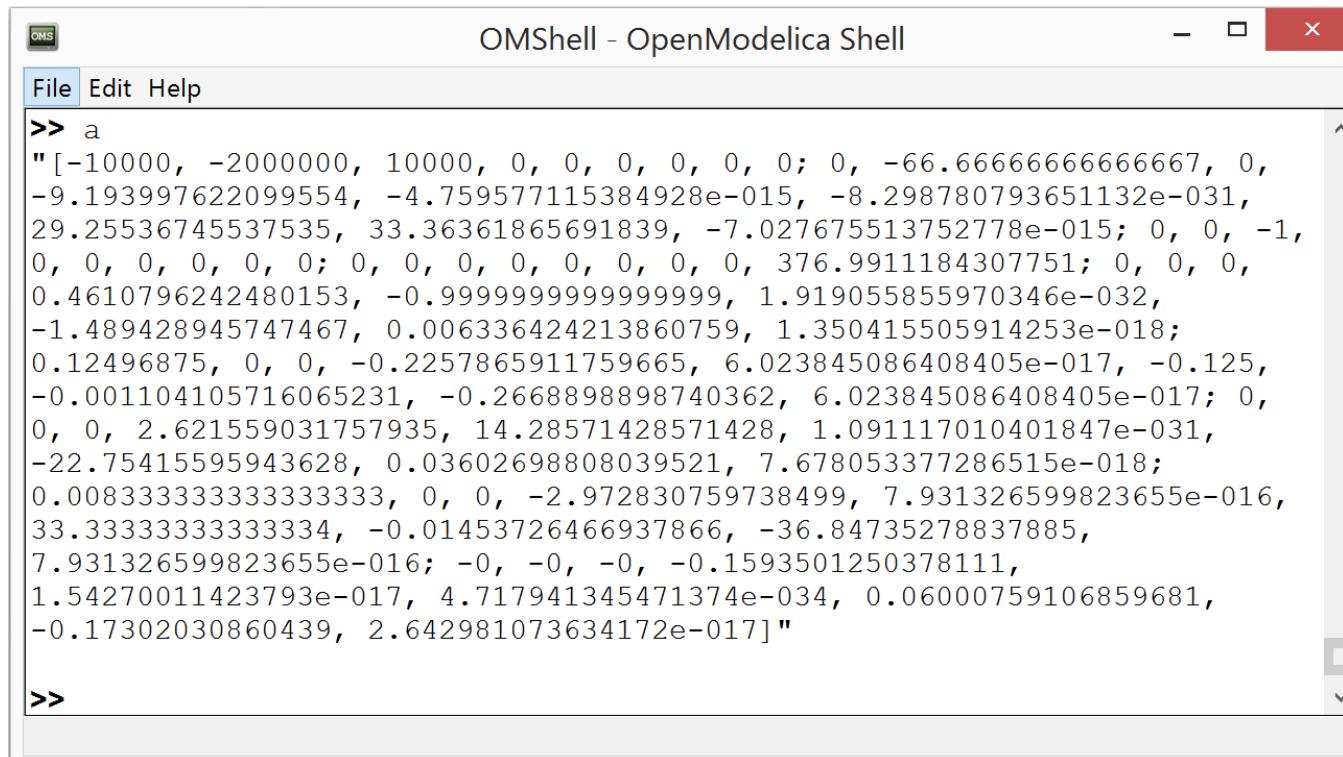
Connected to OpenModelica v1.9.3
To get help on using OMShell and OpenModelica, type "help()" and
press enter.

>> runScript("C:/iPSL_Workshop/linearize_iPSL.mos")|
```

# Example 1

## Linearization

- Upon the execution, the A matrix of the linearized state-space model will be saved in the variable a as a string
- By typing a and pressing Enter, it will be displayed

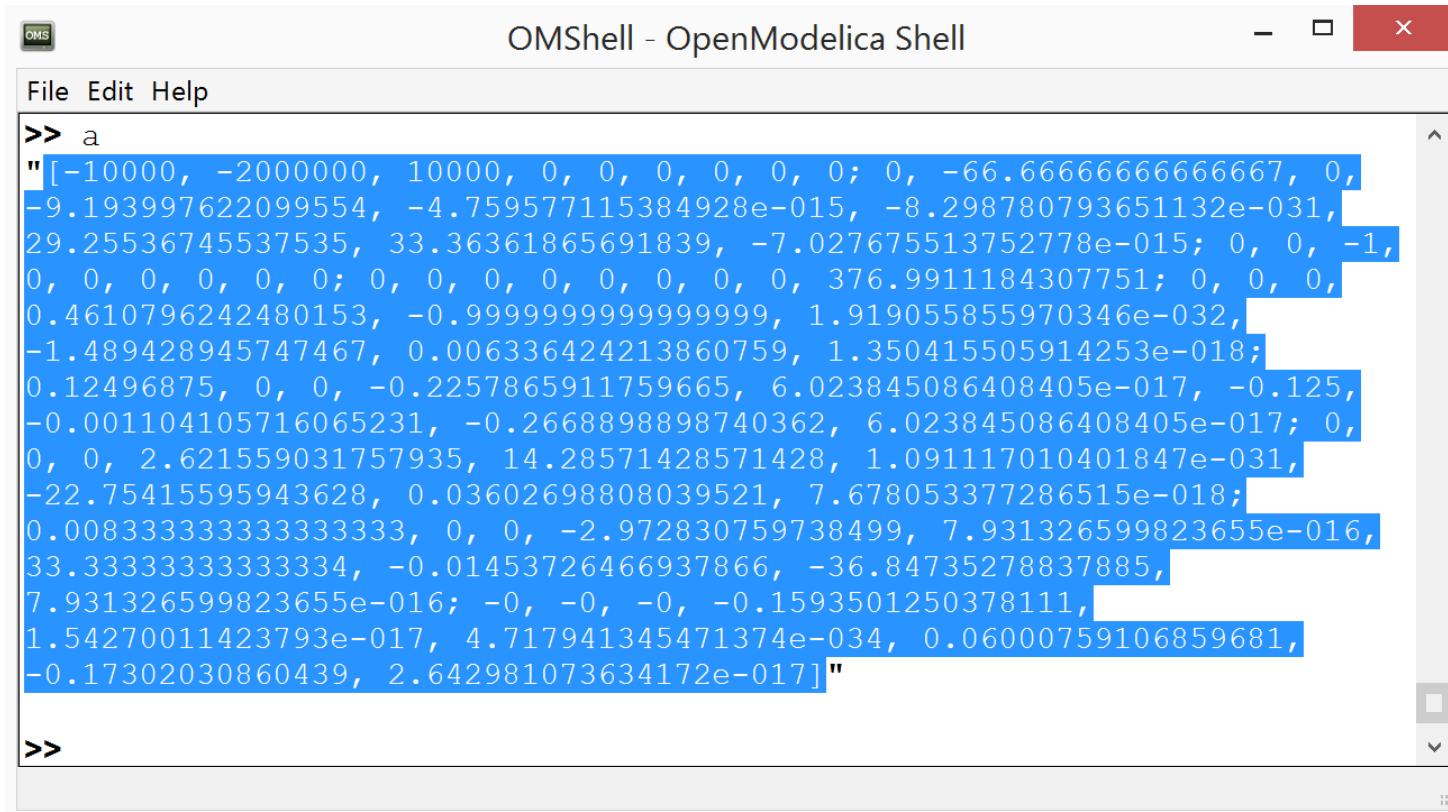


The screenshot shows a window titled "OMShell - OpenModelica Shell". The menu bar includes "File", "Edit", and "Help". The main area contains a command-line interface. The user has typed the command ">> a" followed by a long string of characters representing a matrix. The string starts with "[ -10000, -2000000, 10000, 0, 0, 0, 0, 0; 0, 0, -66.666666666667, 0, -9.193997622099554, -4.759577115384928e-015, -8.298780793651132e-031, 29.25536745537535, 33.36361865691839, -7.027675513752778e-015; 0, 0, -1, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 376.9911184307751; 0, 0, 0, 0.4610796242480153, -0.999999999999999, 1.919055855970346e-032, -1.489428945747467, 0.006336424213860759, 1.350415505914253e-018; 0.12496875, 0, 0, -0.2257865911759665, 6.023845086408405e-017, -0.125, -0.001104105716065231, -0.2668898898740362, 6.023845086408405e-017; 0, 0, 0, 2.621559031757935, 14.28571428571428, 1.091117010401847e-031, -22.75415595943628, 0.03602698808039521, 7.678053377286515e-018; 0.008333333333333333, 0, 0, -2.972830759738499, 7.931326599823655e-016, 33.33333333333334, -0.01453726466937866, -36.84735278837885, 7.931326599823655e-016; -0, -0, -0, -0.1593501250378111, 1.54270011423793e-017, 4.717941345471374e-034, 0.06000759106859681, -0.17302030860439, 2.642981073634172e-017]".

# Example 1

## Linearization

- Copy the output from the previous command without the quotation marks by pressing Ctrl+C



The screenshot shows a terminal window titled "OMShell - OpenModelica Shell". The window has a standard OS X style with a close button. The menu bar includes "File", "Edit", and "Help". The main text area contains the following code:

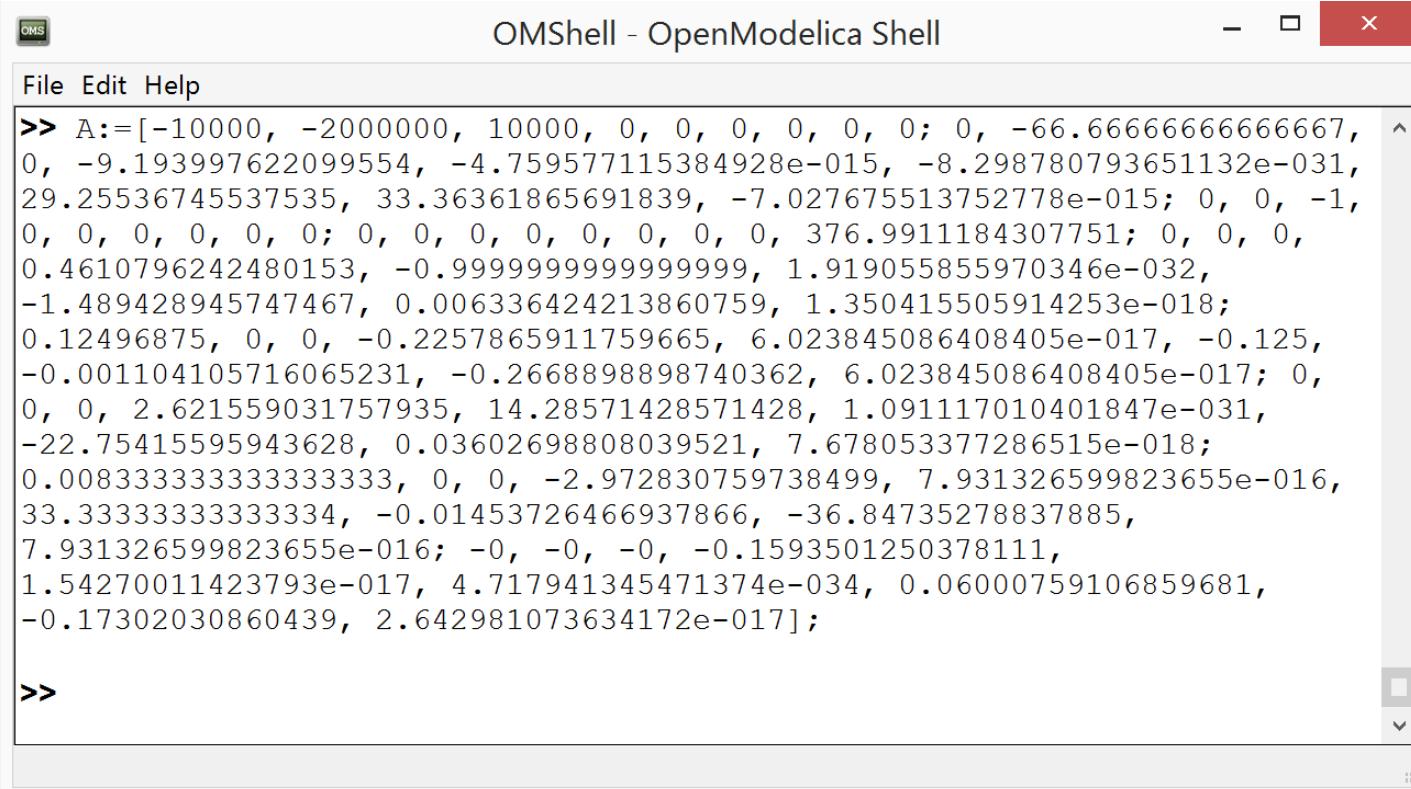
```
>> a
"[-10000, -2000000, 10000, 0, 0, 0, 0, 0, 0; 0, -66.66666666666667, 0,
-9.193997622099554, -4.759577115384928e-015, -8.298780793651132e-031,
29.25536745537535, 33.36361865691839, -7.027675513752778e-015; 0, 0, -1,
0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 376.991184307751; 0, 0, 0,
0.4610796242480153, -0.9999999999999999, 1.919055855970346e-032,
-1.489428945747467, 0.006336424213860759, 1.350415505914253e-018;
0.12496875, 0, 0, -0.2257865911759665, 6.023845086408405e-017, -0.125,
-0.001104105716065231, -0.2668898898740362, 6.023845086408405e-017; 0,
0, 0, 2.621559031757935, 14.28571428571428, 1.091117010401847e-031,
-22.75415595943628, 0.03602698808039521, 7.678053377286515e-018;
0.0083333333333333, 0, 0, -2.972830759738499, 7.931326599823655e-016,
33.33333333333334, -0.01453726466937866, -36.84735278837885,
7.931326599823655e-016; -0, -0, -0, -0.1593501250378111,
1.54270011423793e-017, 4.717941345471374e-034, 0.06000759106859681,
-0.17302030860439, 2.642981073634172e-017]"
```

The input line "a" is followed by a blank line, and the prompt ">>" is at the bottom.

# Example 1

## Linearization

- To save the matrix A as a matrix of Real values type A := and then press Ctrl+V to paste the copied matrix



OMShell - OpenModelica Shell

```
>> A:=[-10000, -2000000, 10000, 0, 0, 0, 0, 0, 0; 0, -66.66666666666667, ^  
0, -9.193997622099554, -4.759577115384928e-015, -8.298780793651132e-031,  
29.25536745537535, 33.36361865691839, -7.027675513752778e-015; 0, 0, -1,  
0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 0, 376.9911184307751; 0, 0, 0,  
0.4610796242480153, -0.999999999999999, 1.919055855970346e-032,  
-1.489428945747467, 0.006336424213860759, 1.350415505914253e-018;  
0.12496875, 0, 0, -0.2257865911759665, 6.023845086408405e-017, -0.125,  
-0.001104105716065231, -0.2668898898740362, 6.023845086408405e-017; 0,  
0, 0, 2.621559031757935, 14.28571428571428, 1.091117010401847e-031,  
-22.75415595943628, 0.03602698808039521, 7.678053377286515e-018;  
0.00833333333333333, 0, 0, -2.972830759738499, 7.931326599823655e-016,  
33.33333333333334, -0.01453726466937866, -36.84735278837885,  
7.931326599823655e-016; -0, -0, -0, -0.1593501250378111,  
1.54270011423793e-017, 4.717941345471374e-034, 0.06000759106859681,  
-0.17302030860439, 2.642981073634172e-017];  
  
>>
```

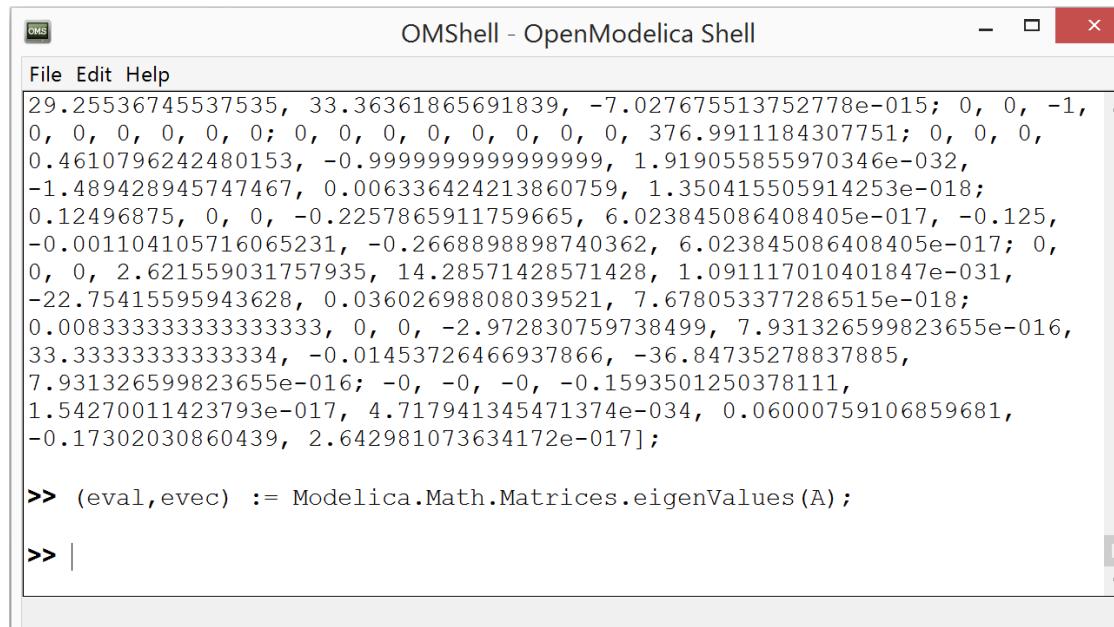
# Example 1

## Linearization

- It is known that the eigenvalues of the linearized system can be found by solving the following equation:

$$\det(A - \lambda I) = 0$$

- This can be done by executing command  
`(eval,evec) := Modelica.Math.Matrices.eigenValues(A);`



```

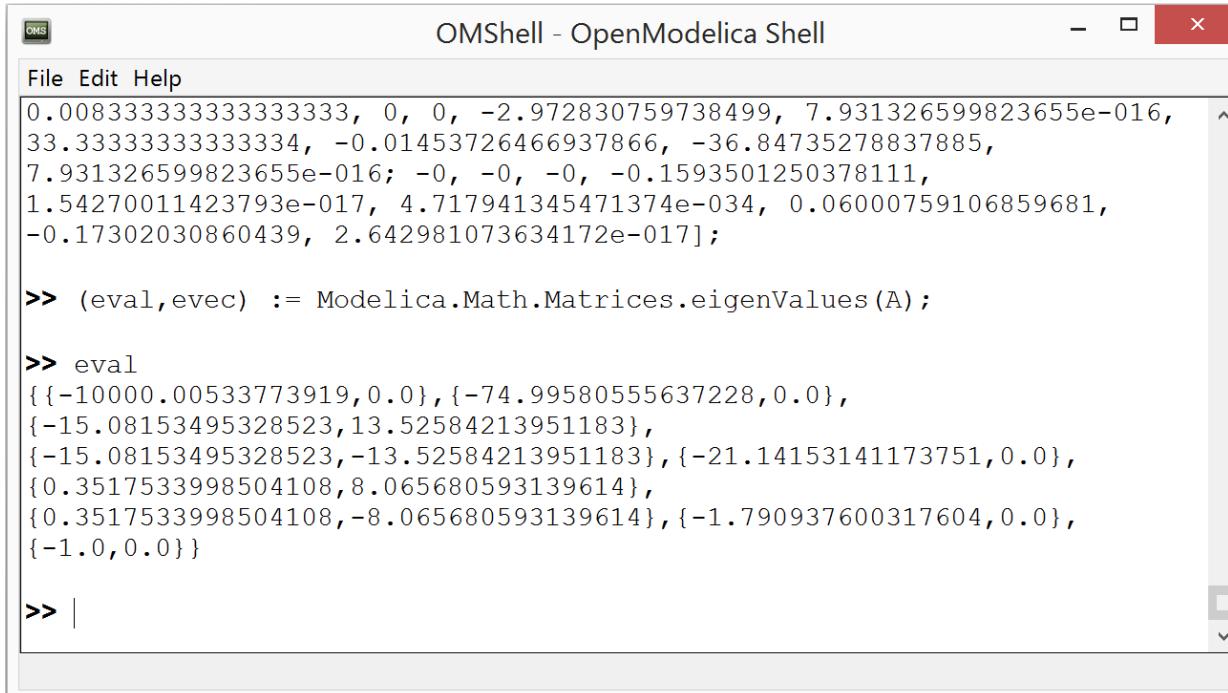
OMShell - OpenModelica Shell
File Edit Help
29.25536745537535, 33.36361865691839, -7.027675513752778e-015; 0, 0, -1, ^
0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 0, 376.9911184307751; 0, 0, 0,
0.4610796242480153, -0.999999999999999, 1.919055855970346e-032,
-1.489428945747467, 0.006336424213860759, 1.350415505914253e-018;
0.12496875, 0, 0, -0.2257865911759665, 6.023845086408405e-017, -0.125,
-0.001104105716065231, -0.2668898898740362, 6.023845086408405e-017; 0,
0, 0, 2.621559031757935, 14.28571428571428, 1.091117010401847e-031,
-22.75415595943628, 0.03602698808039521, 7.678053377286515e-018;
0.00833333333333333, 0, 0, -2.972830759738499, 7.931326599823655e-016,
33.33333333333334, -0.01453726466937866, -36.84735278837885,
7.931326599823655e-016; -0, -0, -0, -0.1593501250378111,
1.54270011423793e-017, 4.717941345471374e-034, 0.06000759106859681,
-0.17302030860439, 2.642981073634172e-017];
>> (eval,evec) := Modelica.Math.Matrices.eigenValues(A);
>> |

```

# Example 1

## Linearization

- The eigenvalues are now stored in the eval variable and they can be listed by executing eval
- Groups of numbers are listed where the first number is real part of the system's pole and the second one is the imaginary part



```
OMShell - OpenModelica Shell
File Edit Help
0.008333333333333333, 0, 0, -2.972830759738499, 7.931326599823655e-016,
33.33333333333334, -0.01453726466937866, -36.84735278837885,
7.931326599823655e-016; -0, -0, -0, -0.1593501250378111,
1.54270011423793e-017, 4.717941345471374e-034, 0.06000759106859681,
-0.17302030860439, 2.642981073634172e-017];

>> (eval,evec) := Modelica.Math.Matrices.eigenValues(A);

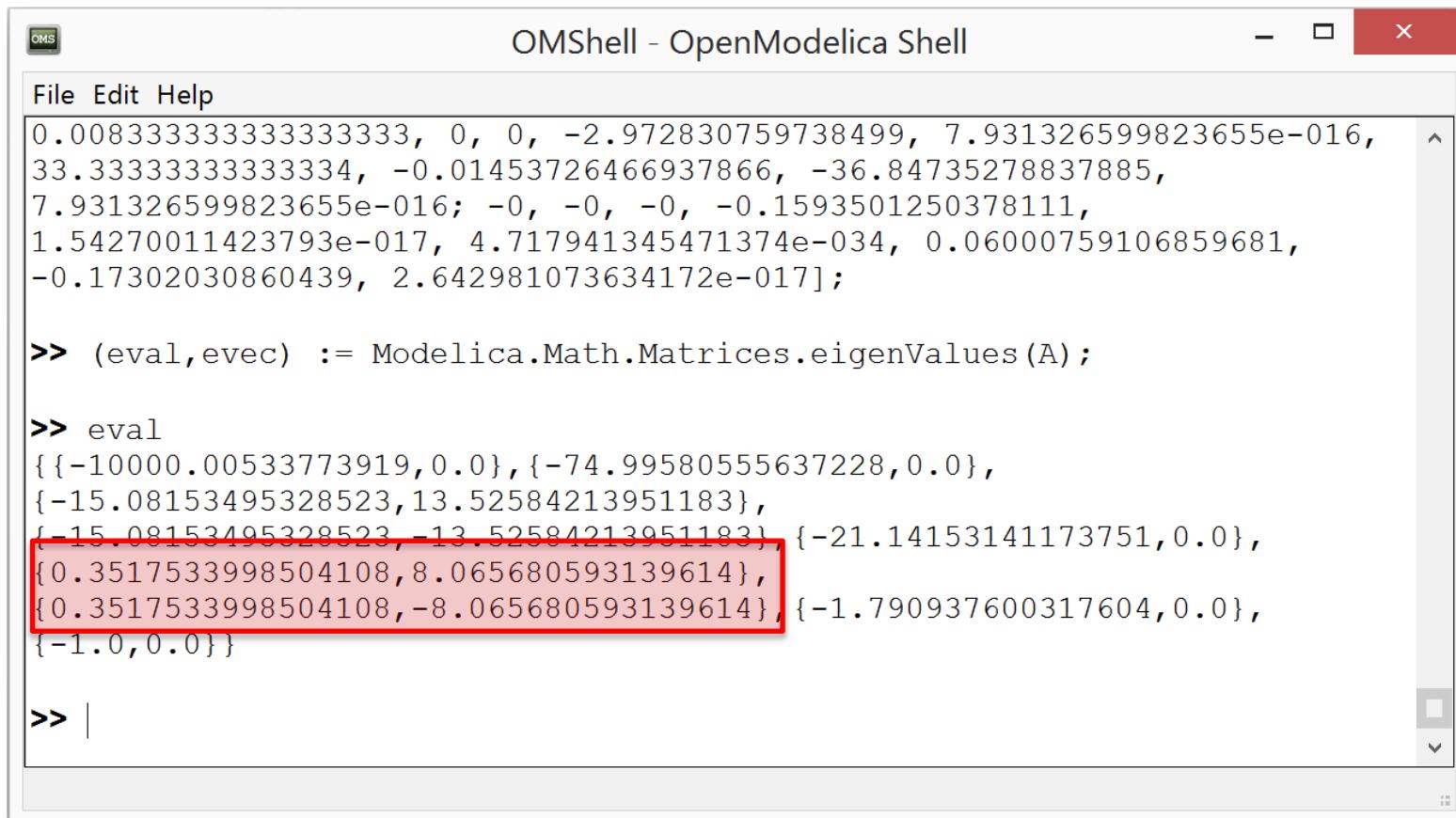
>> eval
{{-10000.00533773919,0.0}, {-74.99580555637228,0.0},
{-15.08153495328523,13.52584213951183},
{-15.08153495328523,-13.52584213951183}, {-21.14153141173751,0.0},
{0.3517533998504108,8.065680593139614},
{0.3517533998504108,-8.065680593139614}, {-1.790937600317604,0.0},
{-1.0,0.0} }

>> |
```

# Example 1

## Linearization

- It can be seen that the pair of conjugate poles exists on the right side of the stability plane and thus, the behaviour of the system is unstable



The screenshot shows the OMShell interface with the title "OMShell - OpenModelica Shell". The window contains the following MATLAB-style code and its execution results:

```
OMS
OMShell - OpenModelica Shell
File Edit Help
0.008333333333333333, 0, 0, -2.972830759738499, 7.931326599823655e-016,
33.33333333333334, -0.01453726466937866, -36.84735278837885,
7.931326599823655e-016; -0, -0, -0, -0.1593501250378111,
1.54270011423793e-017, 4.717941345471374e-034, 0.06000759106859681,
-0.17302030860439, 2.642981073634172e-017];

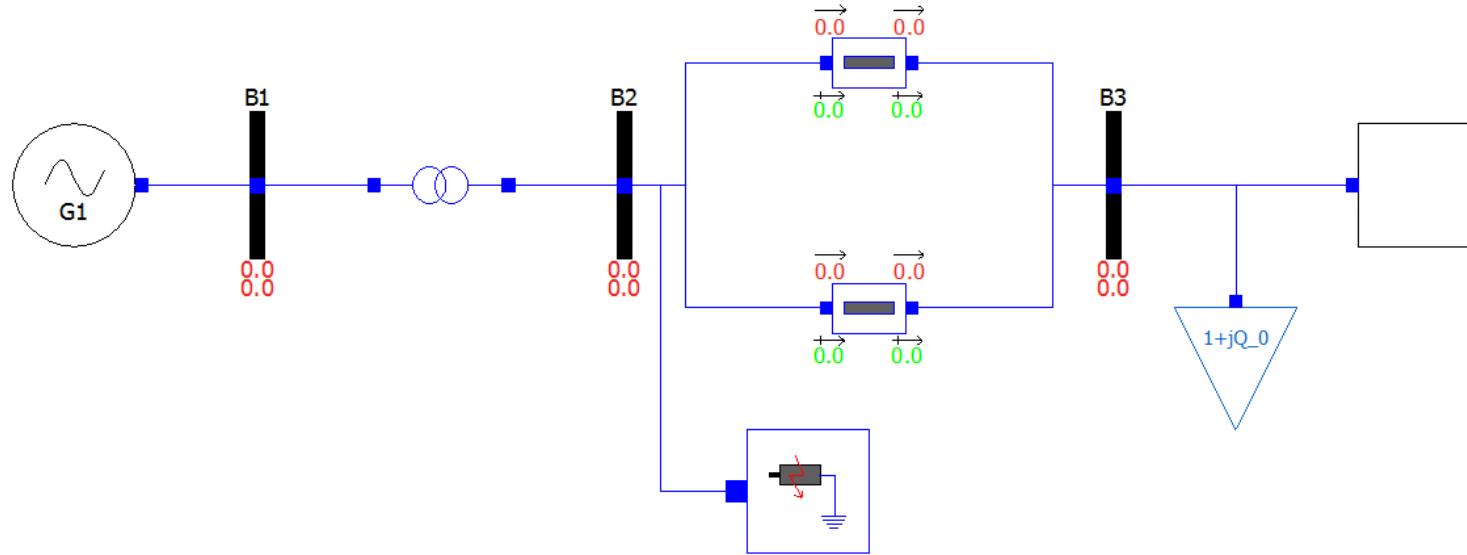
>> (eval,evec) := Modelica.Math.Matrices.eigenValues(A);

>> eval
{{-10000.00533773919,0.0}, {-74.99580555637228,0.0},
{-15.08153495328523,13.52584213951183},
{-15.08153495328523,-13.52584213951183}, {-21.14153141173751,0.0},
{0.3517533998504108,8.065680593139614},
{0.3517533998504108,-8.065680593139614}, {-1.790937600317604,0.0},
{-1.0,0.0} }

>> |
```

The output shows the eigenvalues of matrix A. Two eigenvalues are highlighted with a red box:  $0.3517533998504108 \pm 8.065680593139614i$ . These are the conjugate poles mentioned in the text.

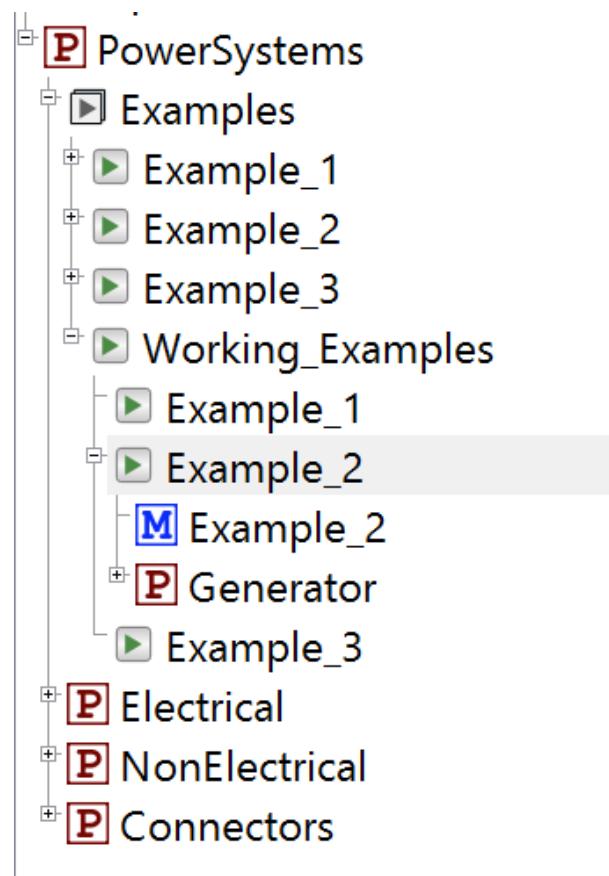
# Example 2



- In the Example 1, it was shown that the system was unstable with a pair of poles on the right side of the stability plane
- In the Example 2, Power System Stabilizer (PSS) will be added to the generator in order to stabilize the system

# Example 2

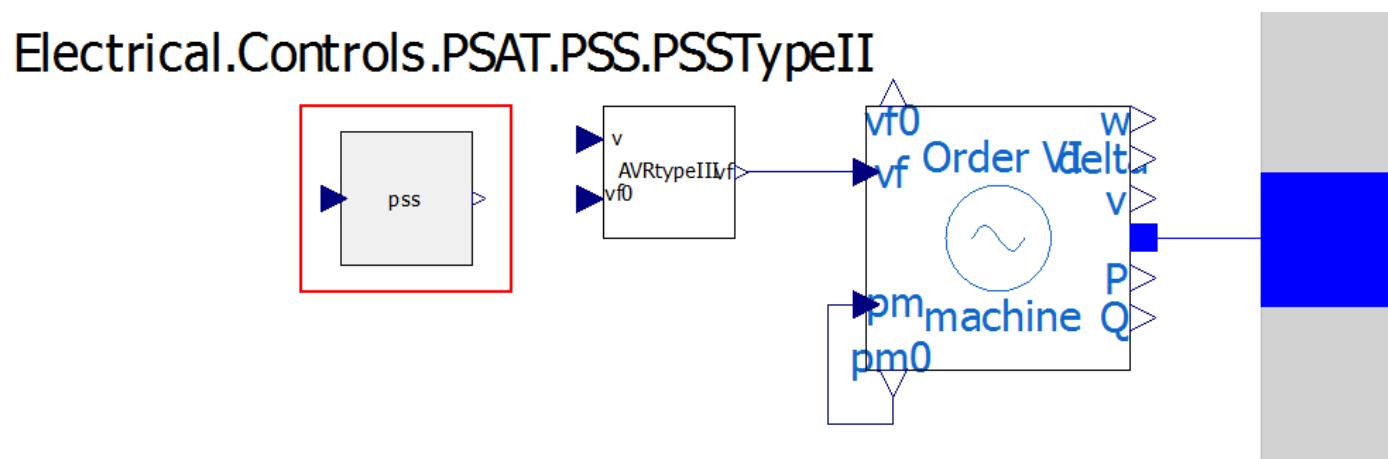
- The work on Example 2 should continue with the files prepared in a package PowerSystems.Examples.Working\_Examples.Example\_2



# Example 2

## Generator model – Step 1

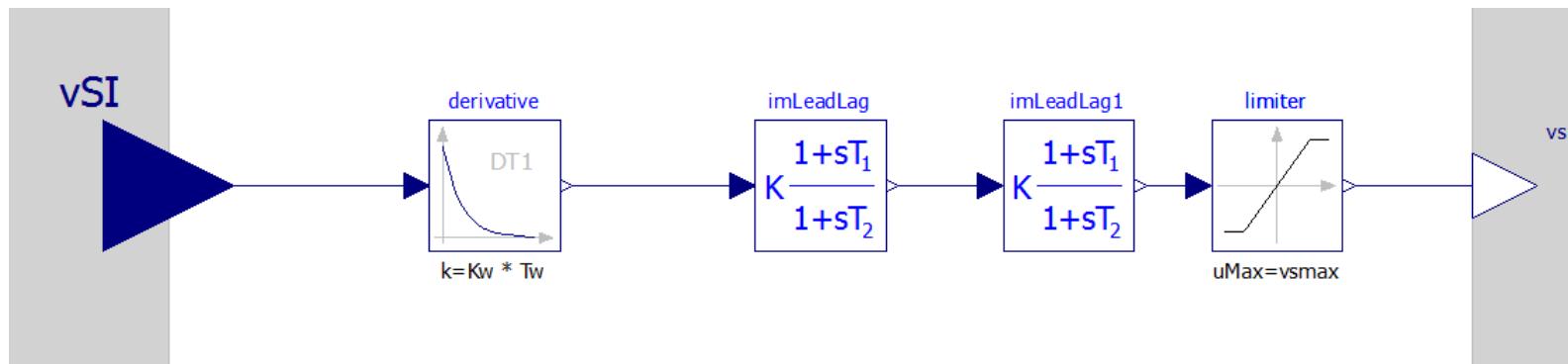
- The first step is to add the model of the PSS Type II and the summation block to the model of the generator



# Example 2

## Generator model – Step 1

- The internal control structure of the PSS can be accessed by right-clicking on the PSS block and selecting “View Class”



# Example 2

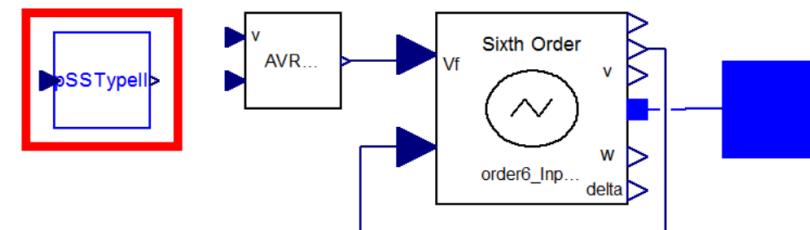
## Generator – Step 1

- PSS should be parameterized as shown in the table

PSS

$v_{s,max}$	0.2	$T_1$	0.154
$v_{s,min}$	-0.2	$T_2$	0.033
$K_w$	1.41	$T_3$	1
$T_w$	0.001	$T_4$	1

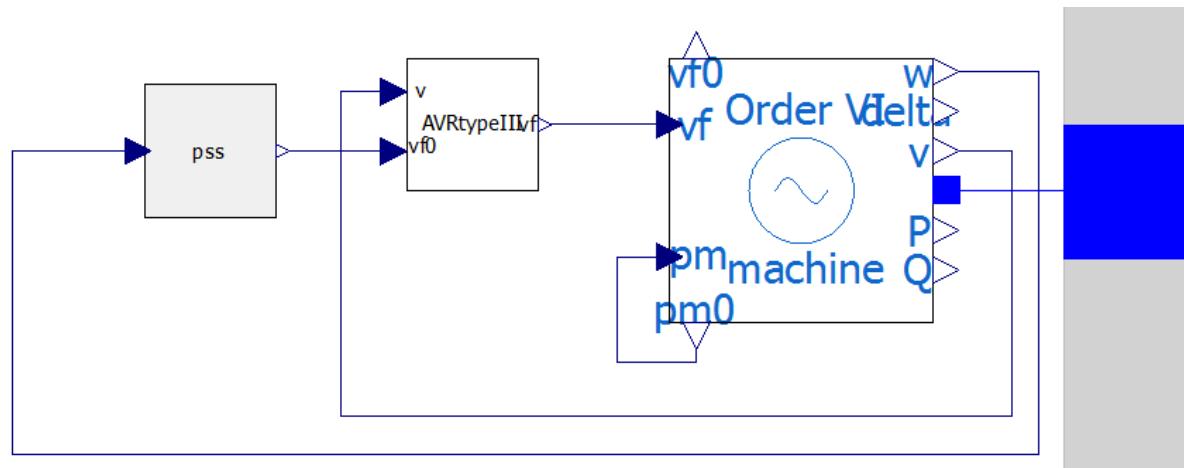
Electrical.Controls.PSAT.PSS.PSSTypell



# Example 2

## Generator – Step 2

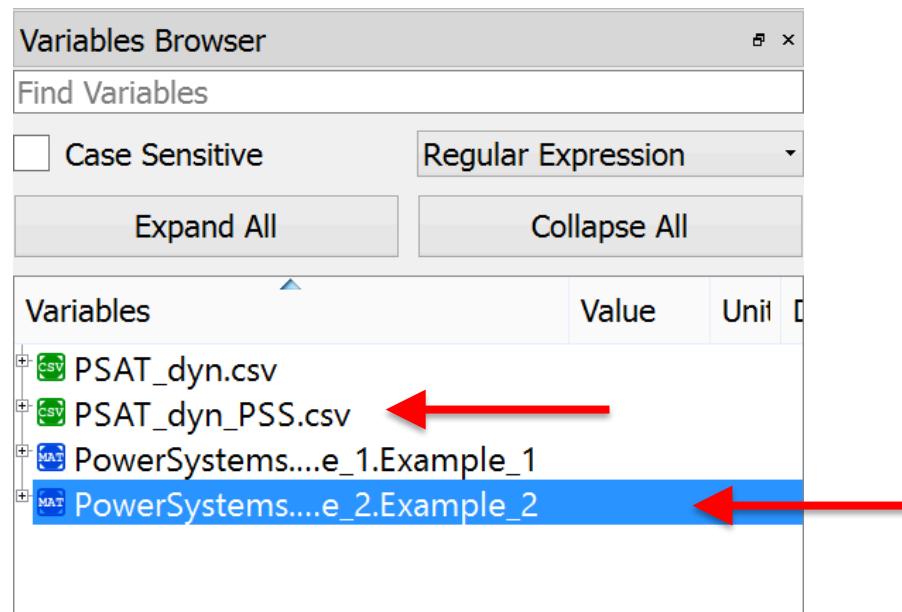
- When the signals of the generator model are connected as shown, model of the generator is completed



# Example 2

## Simulation

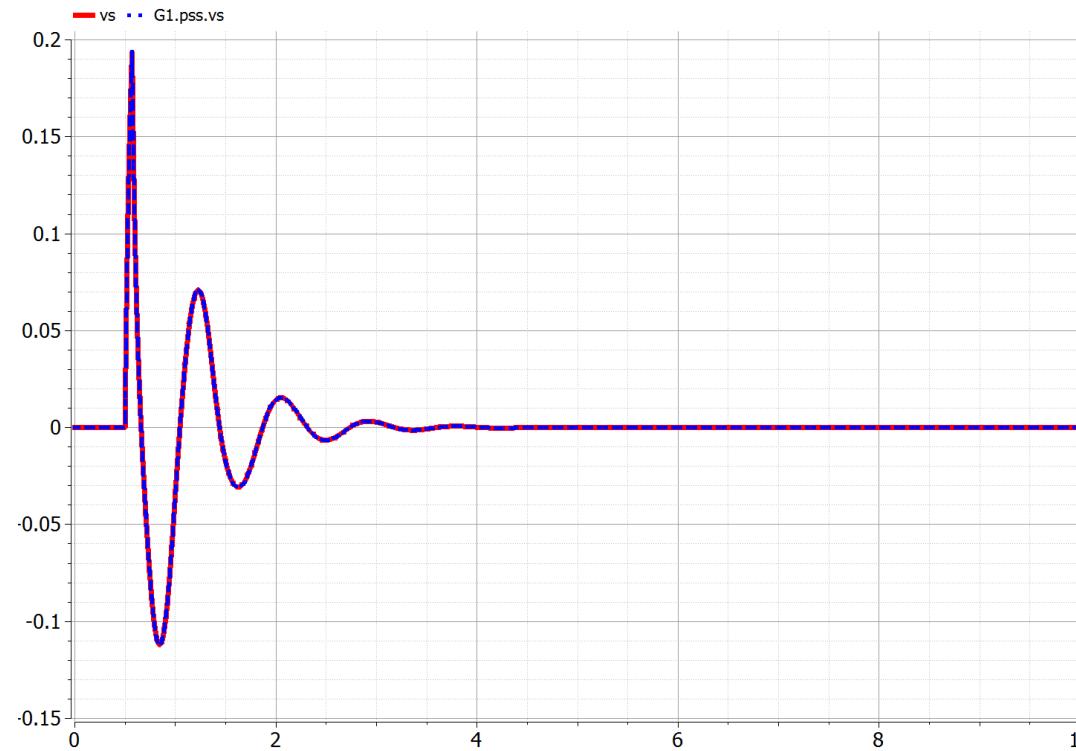
- Simulation steps can be repeated as it was shown in the Example 1
- This time, reference simulation results from the PSAT can be found in the file “PSAT\_dyn\_PSS.csv”
- After the simulation is executed, variable browser should look as it is shown below



# Example 2

## Simulation

- Simulation results can be plotted again
- Comparison of the PSAT and Modelica simulation results of the PSS signal is shown on the figure below



# Example 2

## Linearization

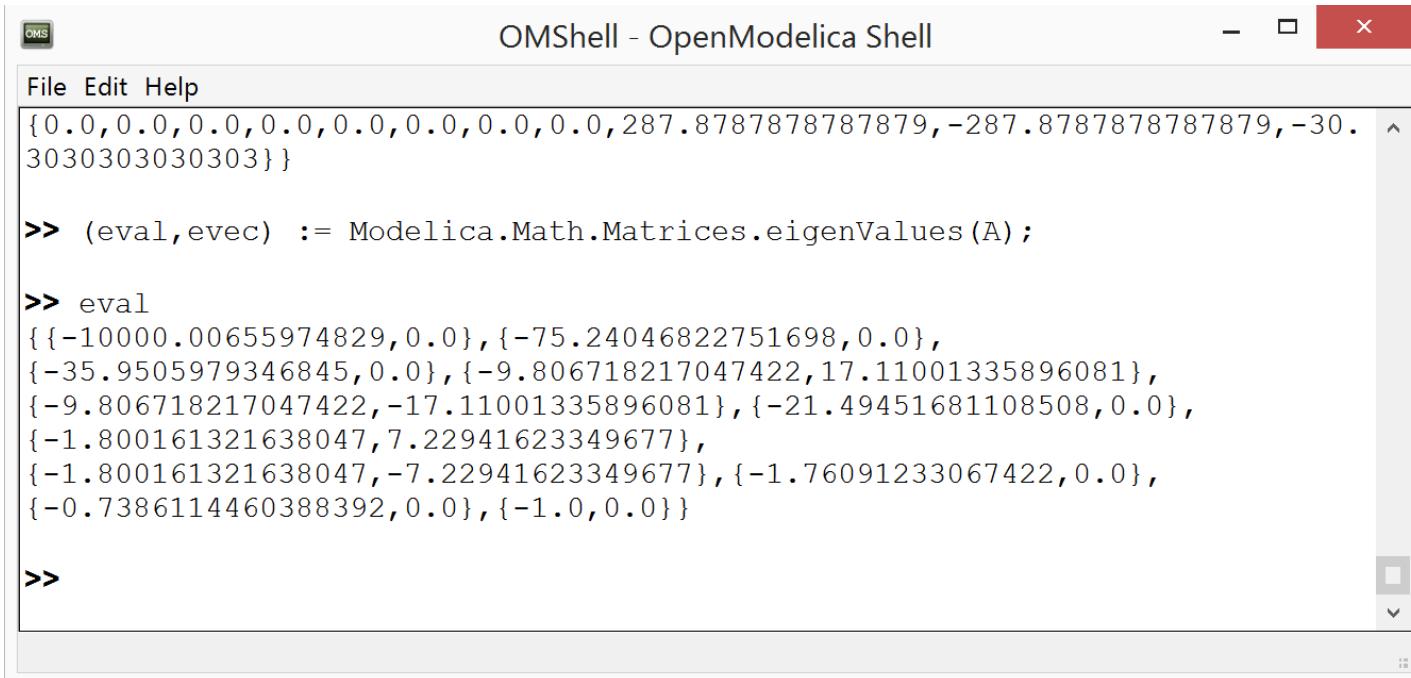
- Before the system is linearized again, changes to the `linearize_iPSL.mos` script shall be made
- Make sure to change the example number from 1 to 2 in lines 4-10
- The script should look as it is shown on the figure below

```
1 loadModel(Modelica);
2 loadFile("C:/iPSL_Workshop/iPSL_stripped.mo");
3 instantiateModel(iPSL;
4 instantiateModel(iPSL.Examples.Example_2.Example_2);
5
6 linearize(iPSL.Examples.Example_2.Example_2,stopTime=0.0);
7
8 loadFile("linear_iPSL.Examples.Example_2.Example_2.mo");
9
10 (a) := getParameterValue(linear_iPSL_Examples_Examp..._2.Example_2,"A");
```

# Example 2

## Linearization

- The rest of the steps shall be repeated as it was shown in Example 1
- The same procedure with a linearized system from Example 2 results in the new set of eigenvalues



OMShell - OpenModelica Shell

```
File Edit Help
{0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,287.87878787879,-287.87878787879,-30.
30303030303} }

>> (eval,evec) := Modelica.Math.Matrices.eigenValues(A);

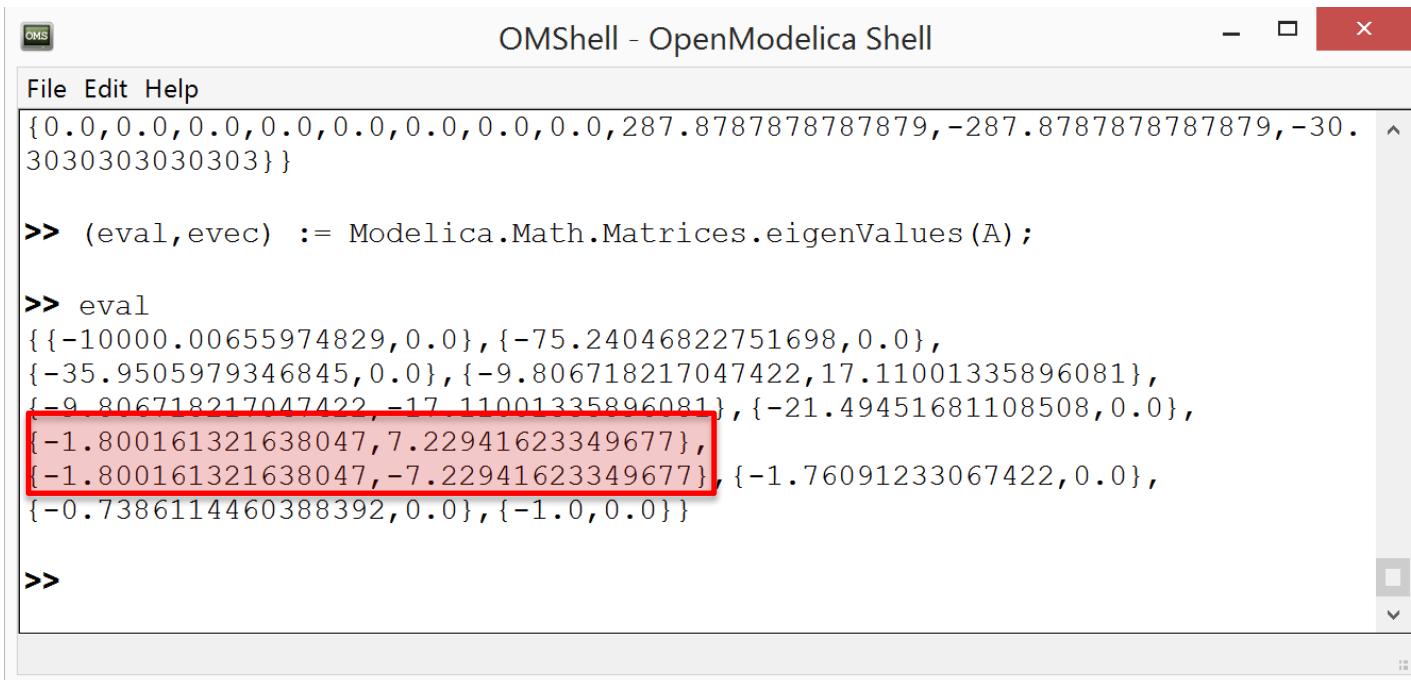
>> eval
{{-10000.00655974829,0.0}, {-75.24046822751698,0.0},
{-35.9505979346845,0.0}, {-9.806718217047422,17.11001335896081},
{-9.806718217047422,-17.11001335896081}, {-21.49451681108508,0.0},
{-1.800161321638047,7.22941623349677},
{-1.800161321638047,-7.22941623349677}, {-1.76091233067422,0.0},
{-0.7386114460388392,0.0}, {-1.0,0.0} }

>>
```

# Example 2

## Linearization

- The conjugate pair of poles that was on the right side of the plane in Example 1 was, by introducing the PSS, moved to the left side of the stability plane and, thus, the system is now stable



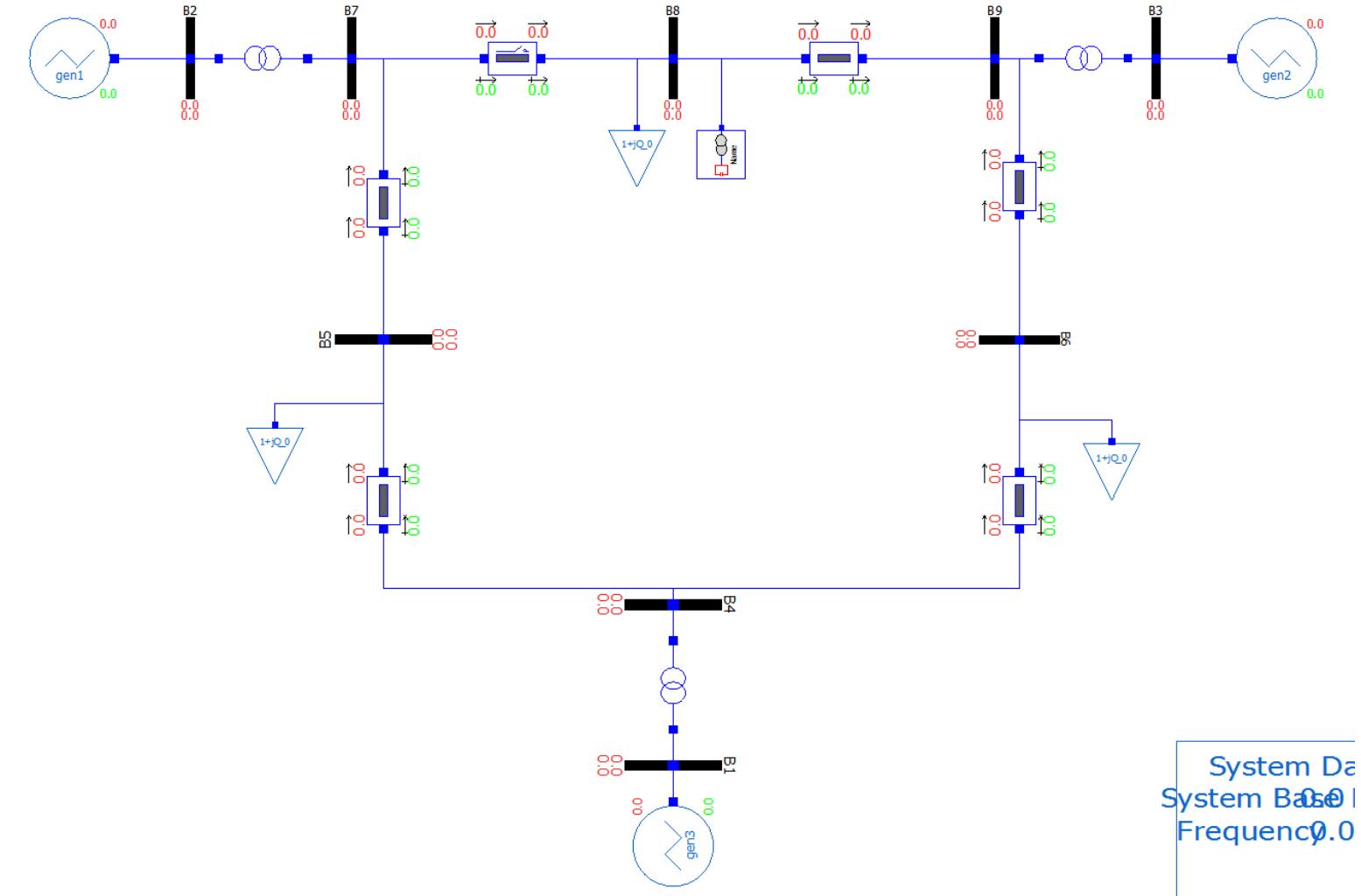
```
OMShell - OpenModelica Shell
File Edit Help
{0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,287.87878787879,-287.87878787879,-30.
3030303030303} }

>> (eval,evec) := Modelica.Math.Matrices.eigenValues(A);

>> eval
{{{-10000.00655974829,0.0}, {-75.24046822751698,0.0},
{-35.9505979346845,0.0}, {-9.806718217047422,17.11001335896081},
{-9.806718217047422,-17.11001335896081}, {-21.49451681108508,0.0},
{-1.800161321638047, 7.22941623349677},
{-1.800161321638047,-7.22941623349677}, {-1.76091233067422,0.0},
{-0.7386114460388392,0.0}, {-1.0,0.0}}}

>>
```

# Example 3



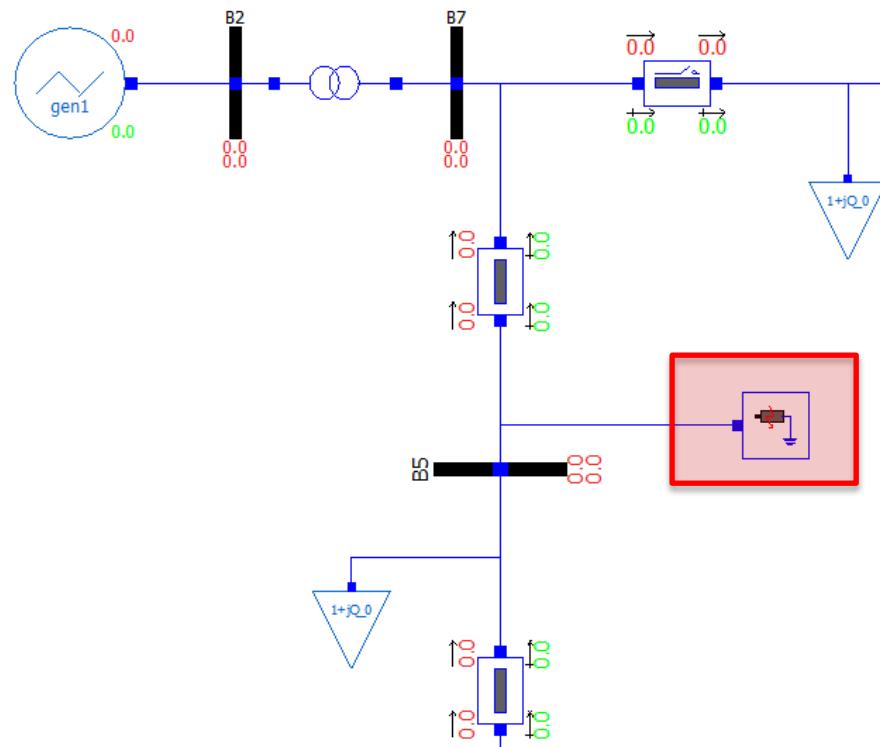
System  $\Delta$   
System  $\beta$   
 $f$   
 $f_0$

# Example 3

- Example 3 contains the model of the IEEE 9 Bus system with the STATCOM at bus 8
- It is pre-configured with all of the power flow and dynamic data
- In the previous two examples, you learned how to build the models of the power system, introduce the faults, run the dynamic simulations and perform the linearization of the model
- In Example 3 you are free to explore the model and introduce various faults

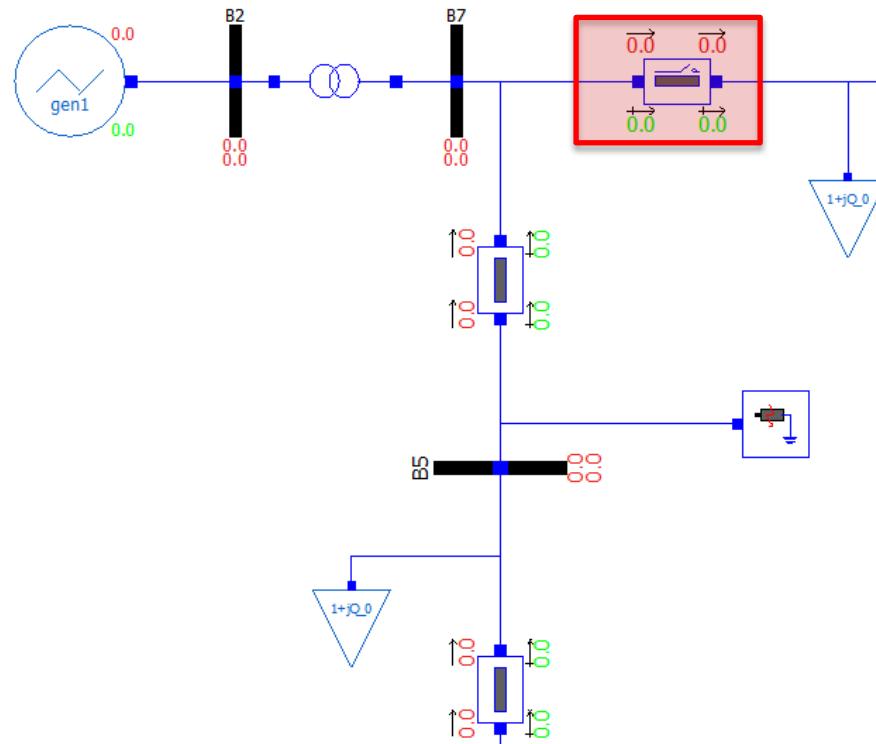
# Example 3

- You can, for instance, introduce the bus fault ...



# Example 3

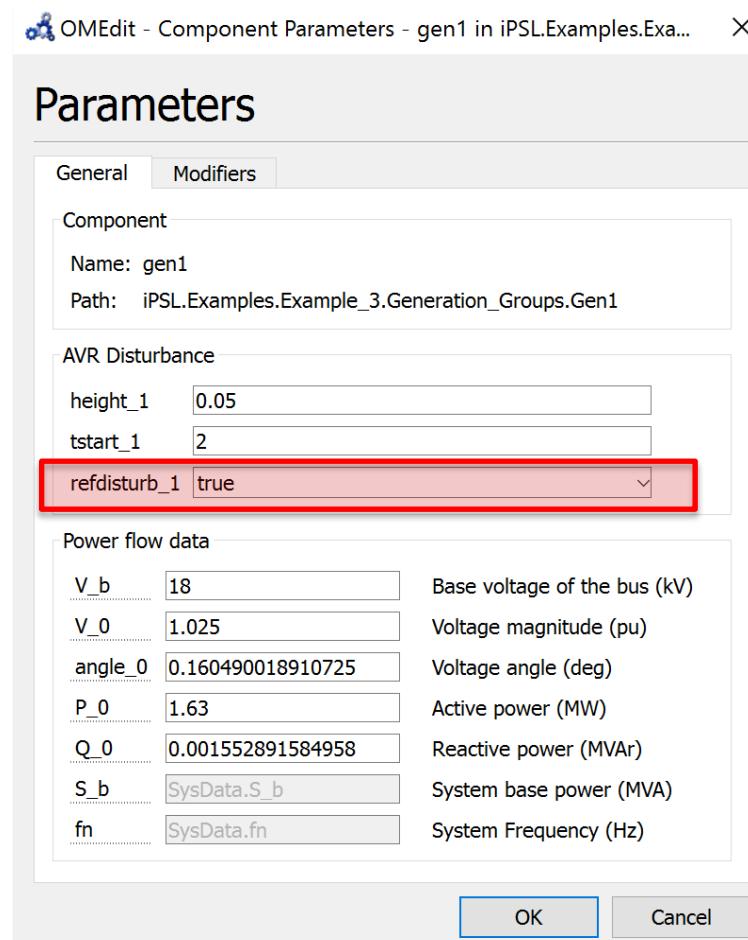
- ... or open the line at the given time instant\*



\*Model of the line with opening is PowerSystems.Electrical.Banches.PwLine2Openings

# Example 3

- Step disturbance to the voltage reference of the generators can be introduced by setting the desired `refdisturb_x` parameter to `true`



# iPSL Developers



GRUPO AIA



## KTH Developers



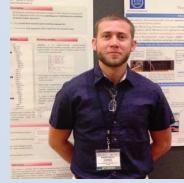
Luigi Vanfretti



Achour  
Amazouz



Mohammed  
Ahsan Adib Murad



Francisco José  
Gómez



Giusseppe  
Laera



Jan Lavenius



Le Qi



Maxime  
Baudette



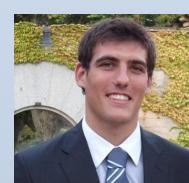
Mengjia Zhang



Tetiana  
Bogodorova



Tin Rabuzin



Joan Russiñol  
Mussons