



# OpenIPSL

A Modelica Library for Power Systems Simulation

Assoc. Prof. **Luigi Vanfretti**

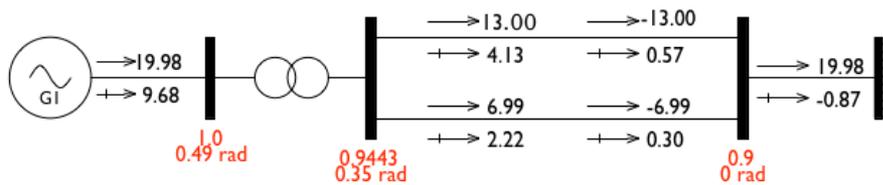
[luigiv@kth.se](mailto:luigiv@kth.se)

<https://www.kth.se/profile/luigiv/>

## **Hands-on Examples!**

Please follow these slides to carry out the examples.

- Very brief introduction to the Open-Instance Power System Library
- Modelling and simulation possibilities by using OpenIPSL and Modelica
- Comparison of the performance with a reference simulation software
- 3 use cases with a dynamic simulation and linearization





# Download the files for the tutorial:



Go to our Github repo:

[https://github.com/SmarTS-Lab/OpenIPSL/releases/tag/Tuto\\_TAMU\\_2017](https://github.com/SmarTS-Lab/OpenIPSL/releases/tag/Tuto_TAMU_2017)

The screenshot shows a web browser window with the URL [https://github.com/SmarTS-Lab/OpenIPSL/releases/tag/Tuto\\_TAMU\\_2017](https://github.com/SmarTS-Lab/OpenIPSL/releases/tag/Tuto_TAMU_2017). The page title is "OpenIPSL Tutorial @TAMU". It indicates that the release is a "Pre-release" and was released by "Ivanfretti" 6 days ago. Below the title is the logo for the "ELECTRICAL & COMPUTER ENGINEERING" department at "TEXAS A&M UNIVERSITY". The main text of the release states: "This release of OpenIPSL's 'Tutorial' was prepared for a Tutorial at Texas A&M University organised by Associate Prof. Le Xie on April 20th 2017." It then lists three items: "The preparatory work prior to the tutorial can be found in this .pdf:", "The slides with the hands-on examples, explained step by step, can be found here:", and "The presentation on OpenIPSL given will be made available herein:". Under the heading "Downloads", there are two buttons: "Source code (zip)" and "Source code (tar.gz)". A large blue button with the text "Click Here!" is overlaid on the right side of the download buttons.

**Note:** A dedicated package will be prepared for the tutorial and uploaded soon.

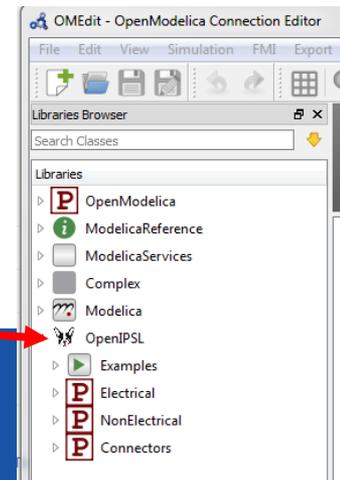
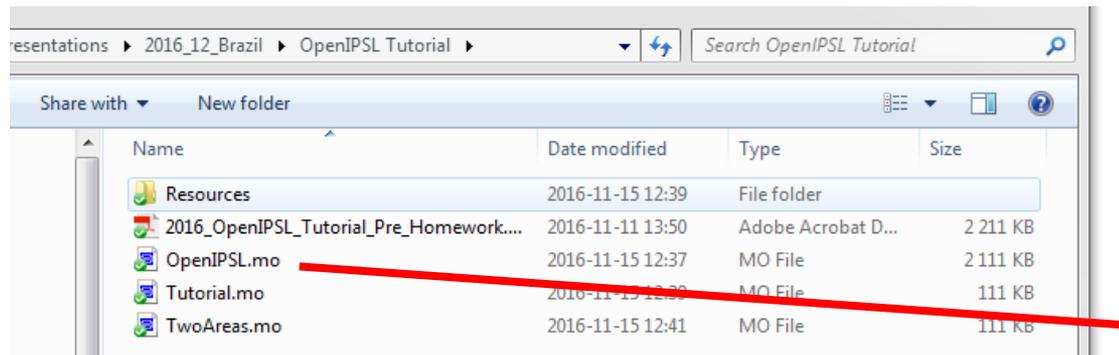
Please download (again!) the package on the day of the tutorial so that you have the most up to date files.

The dedicated package will also be available on a USB stick that we can circulate on the day of the tutorial.

# Load the OpenIPSL to OMEdit

External libraries, such as OpenIPSL, must be loaded in OMEdit to be used:

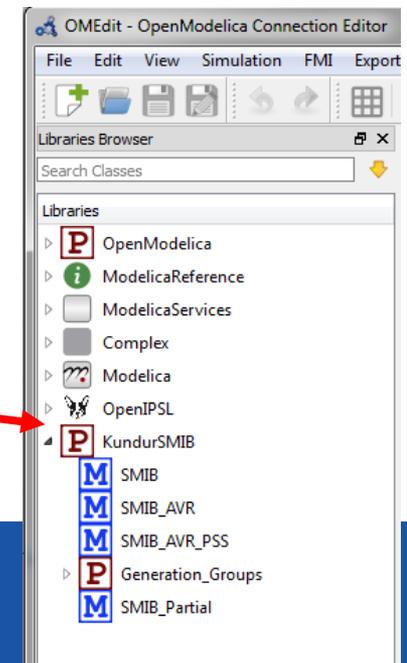
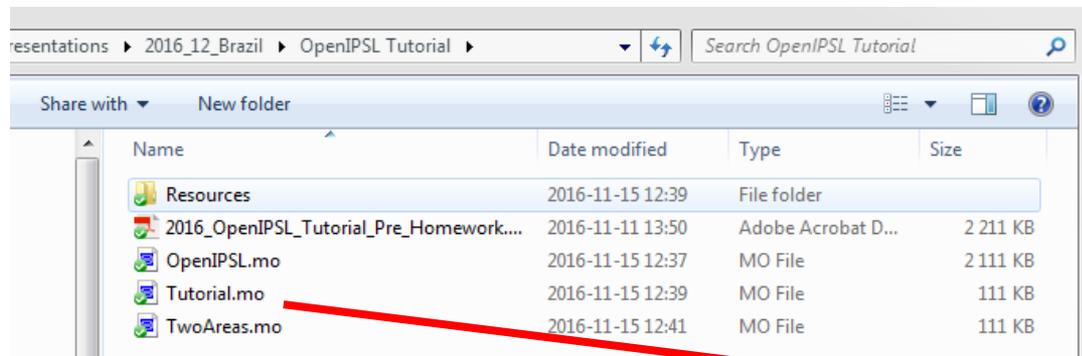
- Unzip the package downloaded at the previous step
- Open OpenModelica Connection Editor (OMEdit)
- Browse Windows Explorer to the location of the unzipped folder
- Drag & drop the **OpenIPSL.mo** file to the **Library Browser** in OMEdit.



# Load an Application Example to OMEdit

Once the OpenIPSL is loaded (see previous slide) in OMEdit, you can load the Tutorial package:

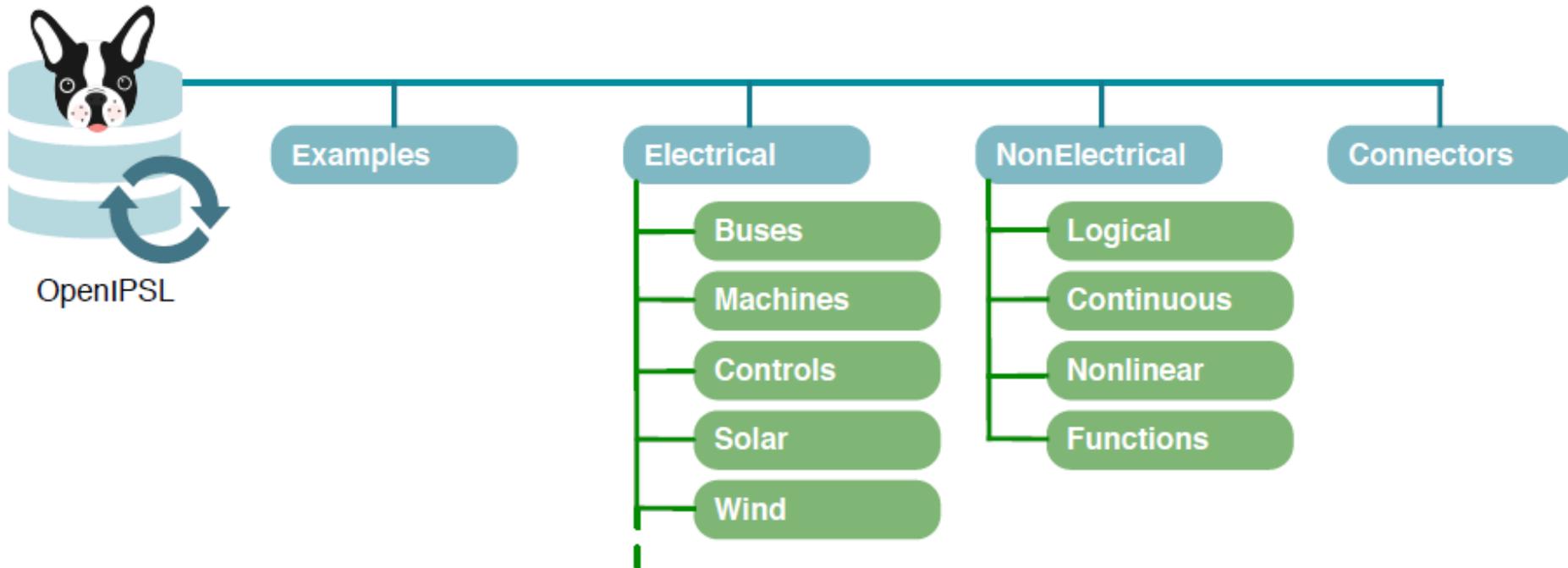
- Browse Windows Explorer to the location of the unzipped folder
- Drag & drop the **Tutorial.mo** file to the **Library Browser** in OMEdit.





# Library introduction

OpenIPSL is divided in four main categories:

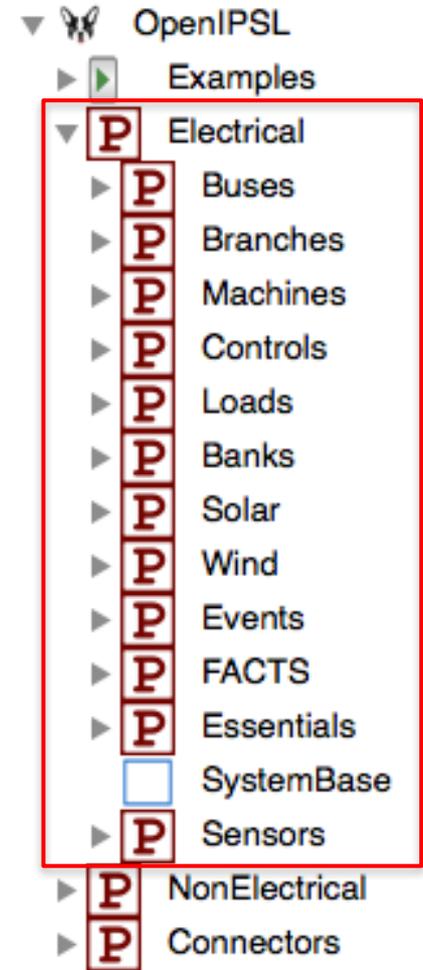




# Library introduction

## 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





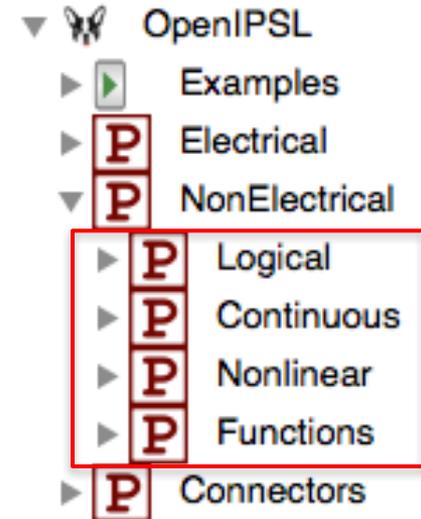
# Library introduction

## 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)

## Connectors

- The *Connectors* package contains a set of specifically developed Modelica connectors to harmonize the models in this library ( e.g. *PwPin* a connector, which contains voltage and current quantities in phasor representation)

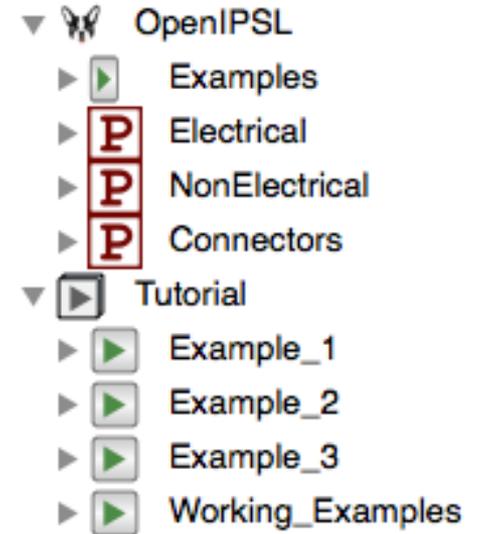




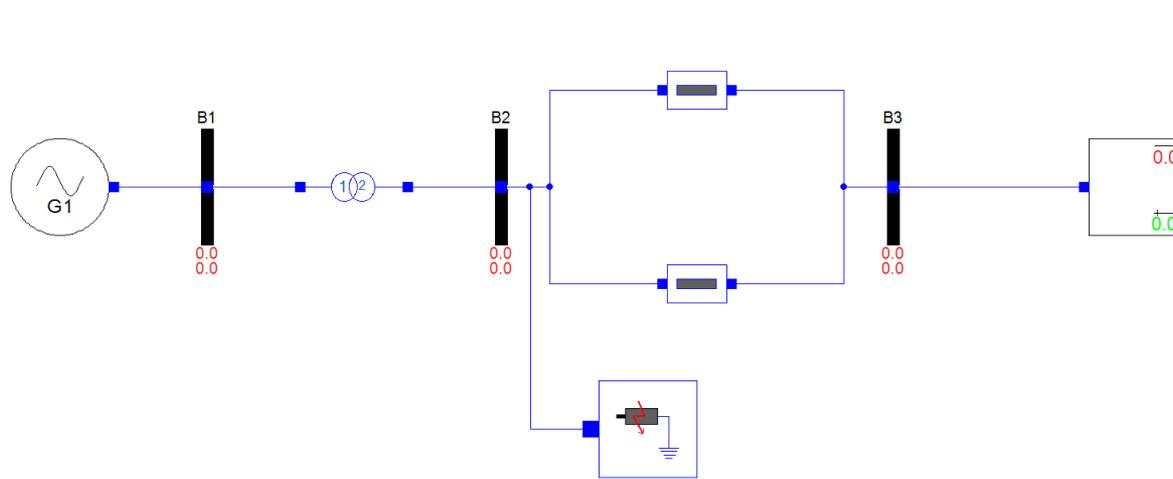
# Library introduction

## Examples

- In this workshop, the *Tutorial* 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 sub-packages will be used by attendees of the workshop to create use cases on their own



# Example 1\*

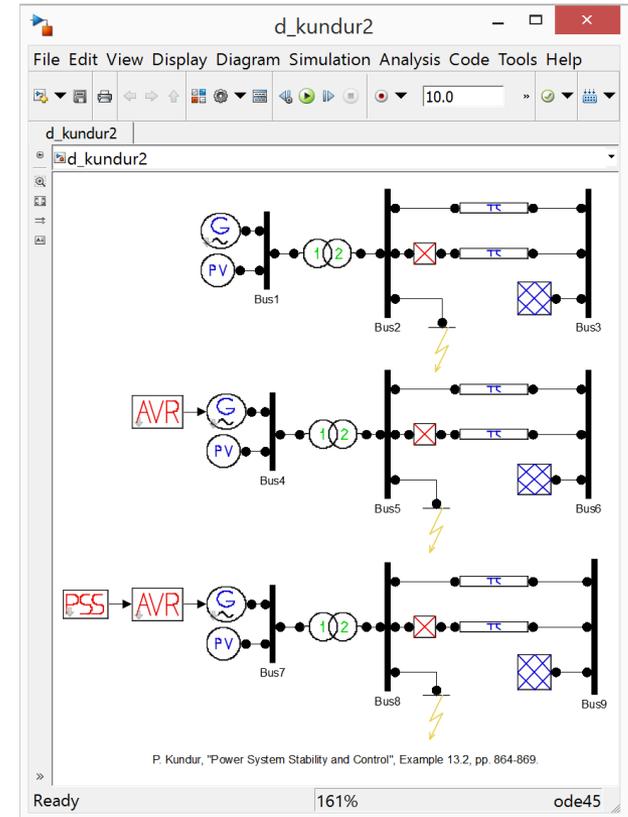


- Single Machine Infinite Bus (SMIB) system
- Analysis of the transient stability of the system including the effects of rotor circuit dynamics and excitation control
- Four machines represented by one connected via transformer and parallel lines to the infinite bus

# Example 1

## Power flow

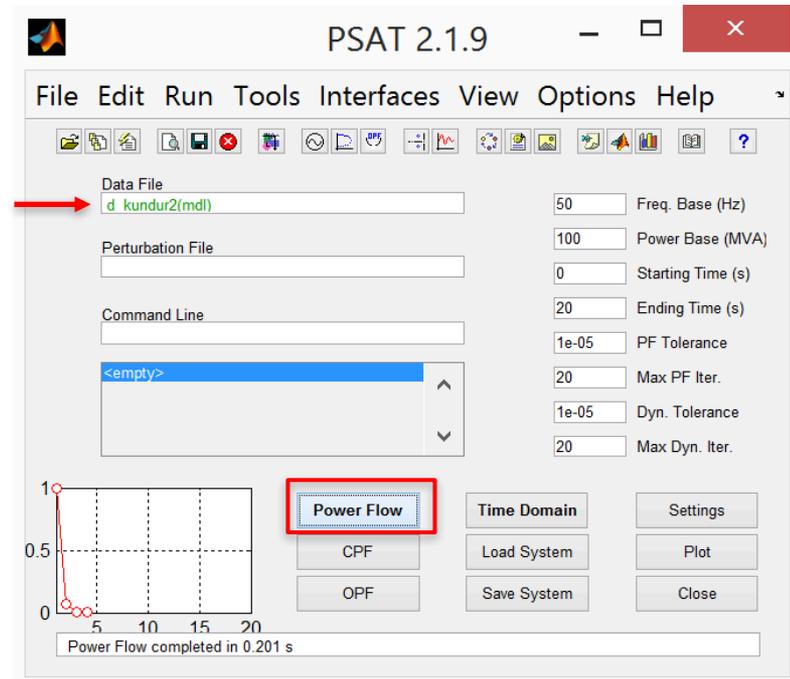
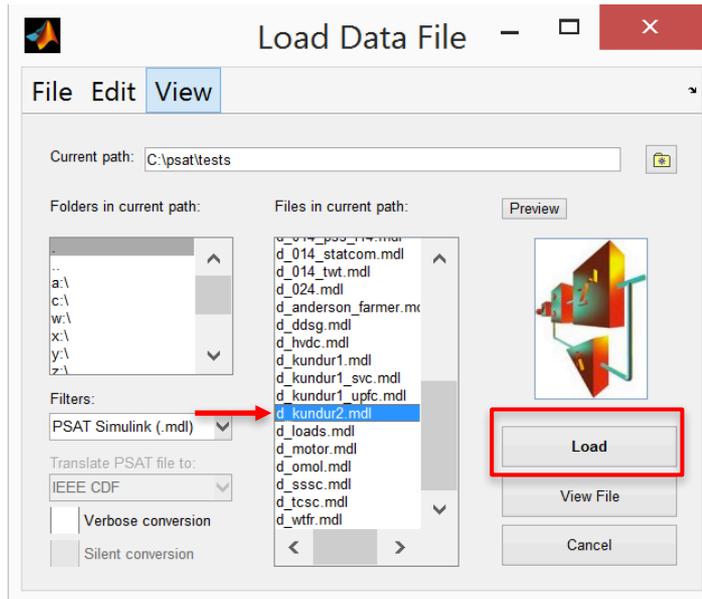
- Power flow results were obtained by PSAT
- Prepared Example 1 already exists in PSAT and can be used for power flow calculations and dynamic simulations



# Example 1

## Power flow

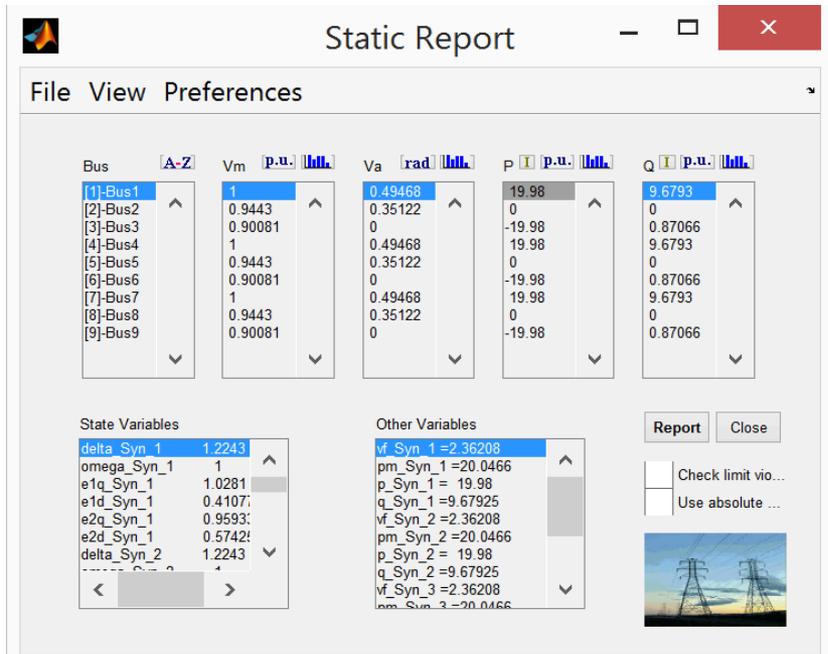
- Example 1 is loaded and the power flow calculations are executed



# 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



The screenshot shows the 'Static Report' window with the following data:

| Bus      | Vm [p.u.] | Va [rad] | P [p.u.] | Q [p.u.] |
|----------|-----------|----------|----------|----------|
| [1]-Bus1 | 1         | 0.49468  | 19.98    | 9.6793   |
| [2]-Bus2 | 0.9443    | 0.35122  | 0        | 0        |
| [3]-Bus3 | 0.90081   | 0        | -19.98   | 0.87066  |
| [4]-Bus4 | 1         | 0.49468  | 19.98    | 9.6793   |
| [5]-Bus5 | 0.9443    | 0.35122  | 0        | 0        |
| [6]-Bus6 | 0.90081   | 0        | -19.98   | 0.87066  |
| [7]-Bus7 | 1         | 0.49468  | 19.98    | 9.6793   |
| [8]-Bus8 | 0.9443    | 0.35122  | 0        | 0        |
| [9]-Bus9 | 0.90081   | 0        | -19.98   | 0.87066  |

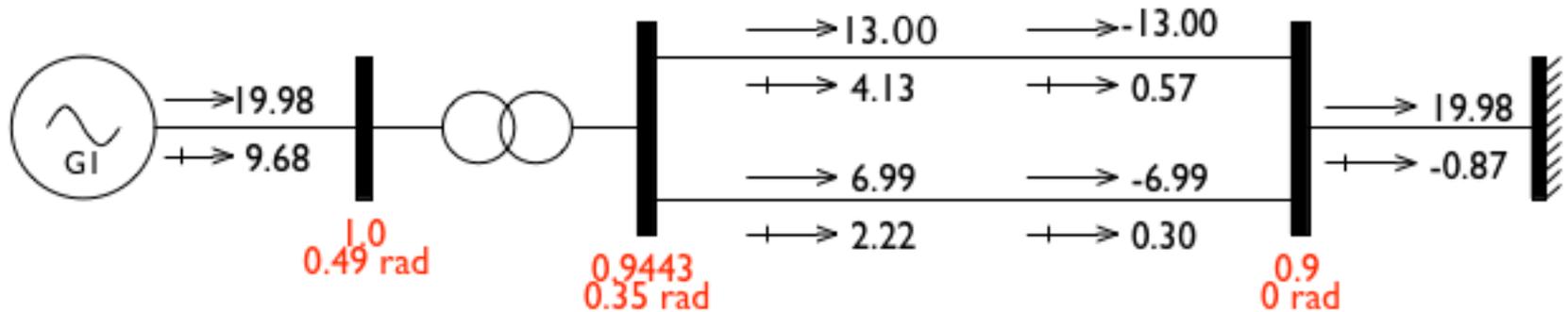
  

| State Variables | Other Variables |                  |
|-----------------|-----------------|------------------|
| delta_Syn_1     | 1.2243          | vf_Syn_1=2.36208 |
| omega_Syn_1     | 1               | pm_Syn_1=20.0466 |
| e1q_Syn_1       | 1.0281          | p_Syn_1=19.98    |
| e1d_Syn_1       | 0.4107          | q_Syn_1=9.67925  |
| e2q_Syn_1       | 0.9593          | vf_Syn_2=2.36208 |
| e2d_Syn_1       | 0.5742          | pm_Syn_2=20.0466 |
| delta_Syn_2     | 1.2243          | p_Syn_2=19.98    |
| omega_Syn_2     | 1               | q_Syn_2=9.67925  |
| e1q_Syn_2       | 1.0281          | vf_Syn_3=2.36208 |
| e1d_Syn_2       | 0.4107          | pm_Syn_3=20.0466 |
| e2q_Syn_2       | 0.9593          | p_Syn_3=19.98    |
| e2d_Syn_2       | 0.5742          | q_Syn_3=9.67925  |

# Example 1

## Power flow

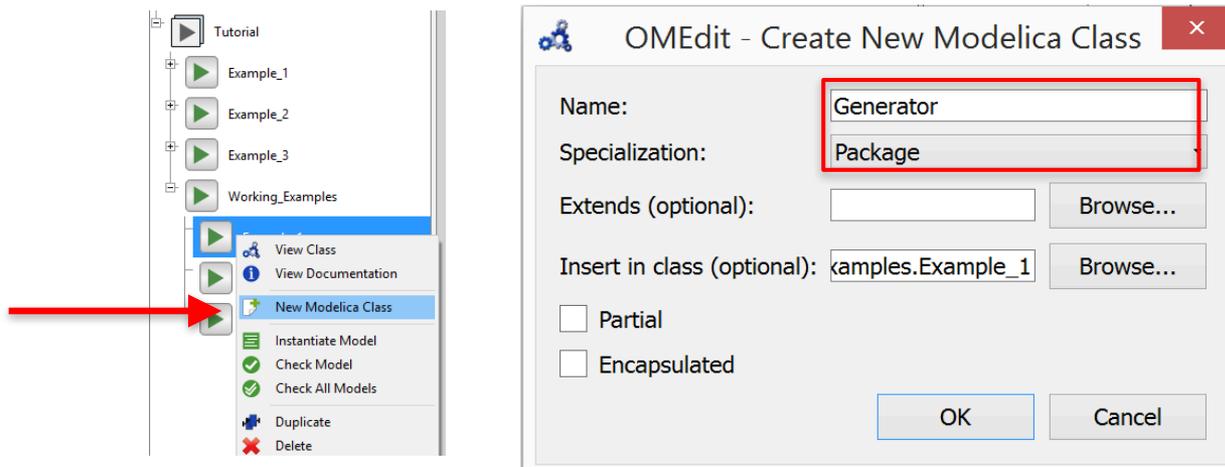
- The summary of all of the relevant data from the power flow is given on the figure below



# Example 1

## Generator model – Step 1

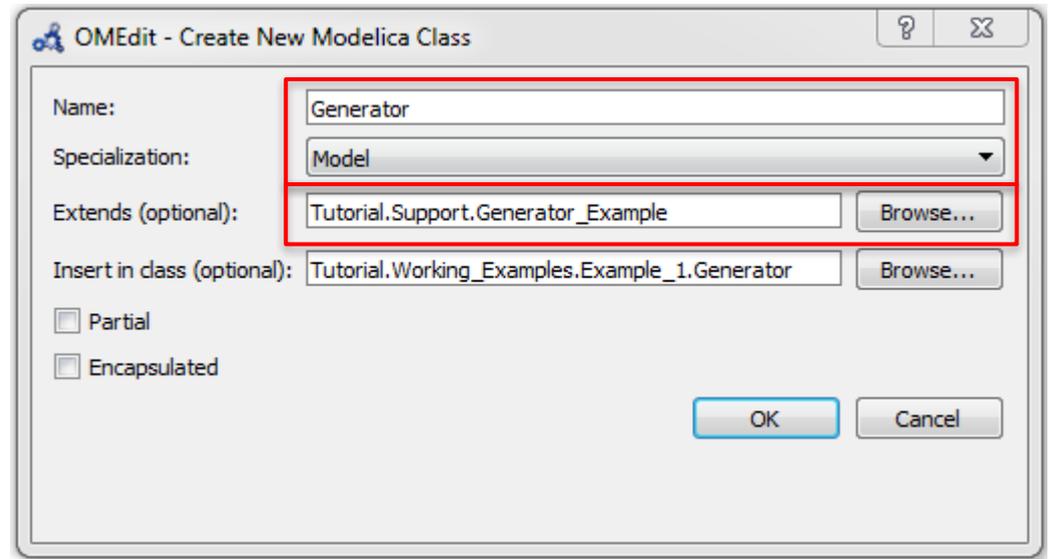
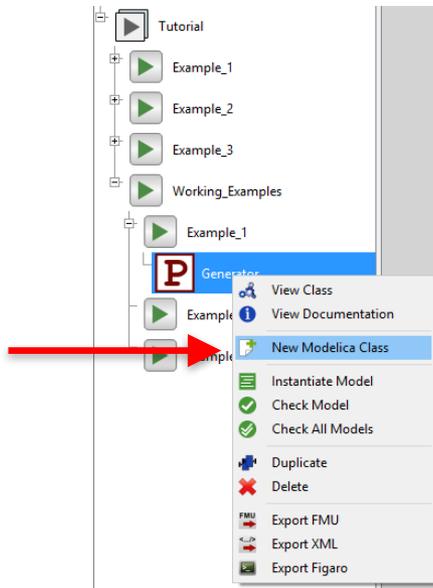
- First, the package where the generator model will be located has to be created
- This is done by right clicking on the *Example\_1* in the *Working\_Examples* package
- The package should be named *Generator*



# Example 1

## Generator model – Step 1

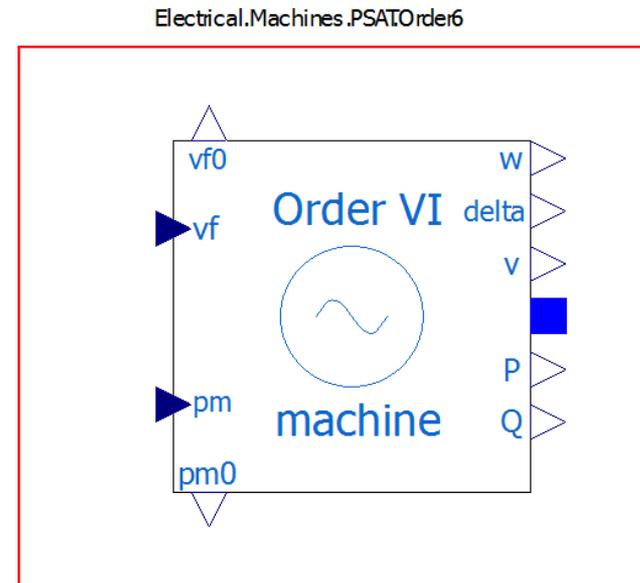
- Within the *Generator* package, model of the generator shall be created
- Extends from *Tutorial.Support.Generator\_Example*



# 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

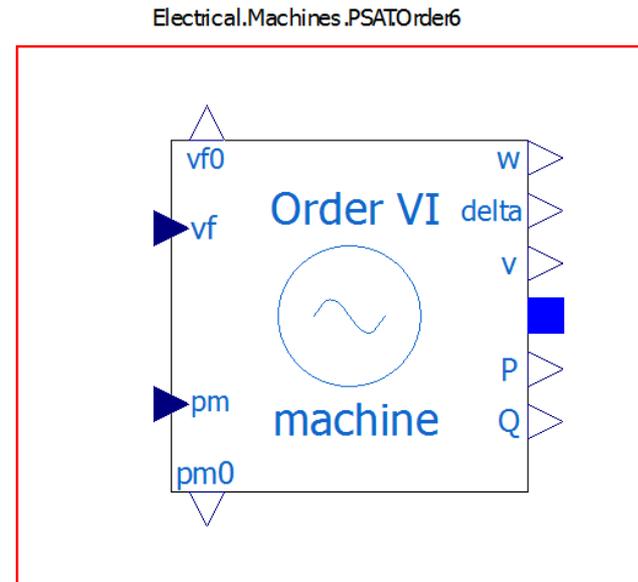


# Example 1

## 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     |



# Example 1

## Generator model – Step 1

- Power flow results:

|           |             |
|-----------|-------------|
| $V_0$     | V_0         |
| $angle_0$ | angle_0     |
| $P_0$     | P_0         |
| $Q_0$     | Q_0         |
| $V_b$     | V_b         |
| $S_b$     | Do not edit |
| $f_n$     | Do not edit |

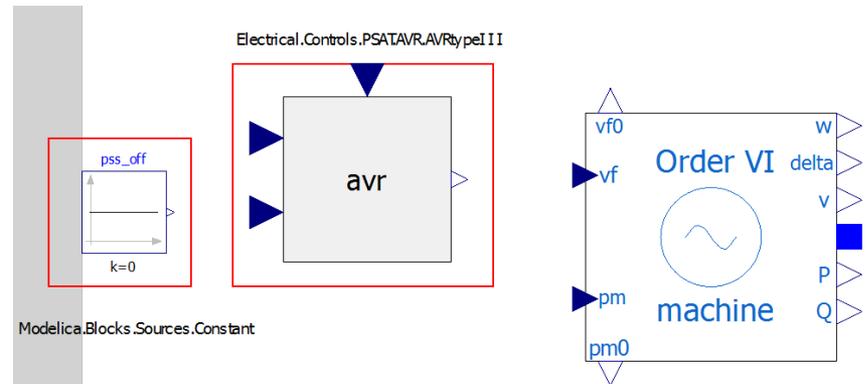
Note: Using the variables ( $V_0$ ,  $angle_0$ , etc.) allow to propagate the parameters to the “upper layer” of the generator component

# 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:

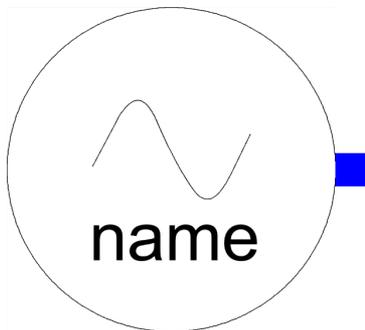
|             |        |
|-------------|--------|
| $v_{f,max}$ | 7      |
| $v_{f,min}$ | -6.4   |
| $K_0$       | 200    |
| $T_2$       | 1      |
| $T_1$       | 1      |
| $T_e$       | 0.0001 |
| $T_r$       | 0.015  |



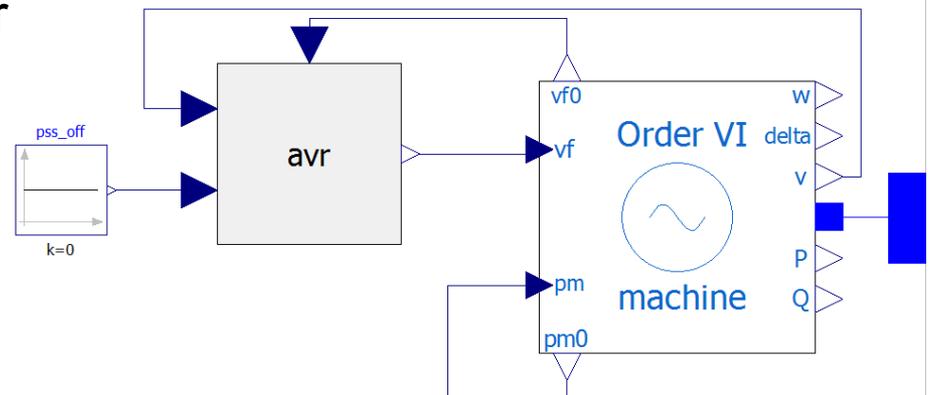
# Example 1

## Generator model – Step 3

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



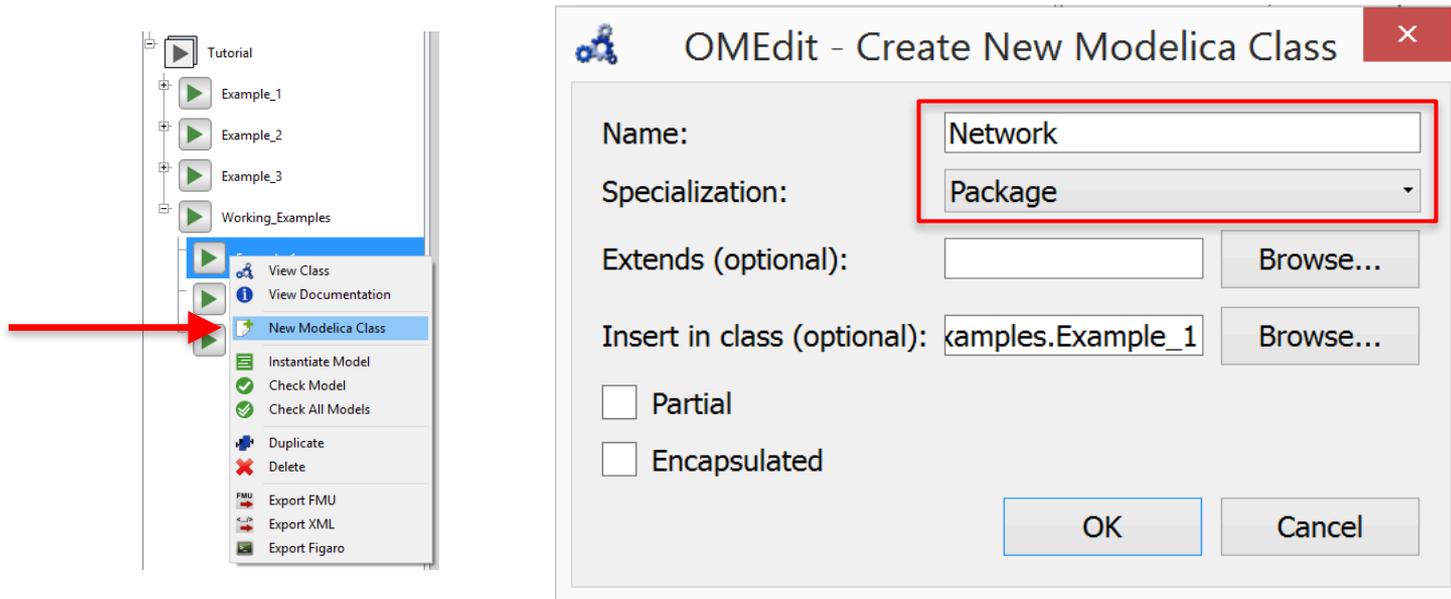
1. Machine's terminal voltage to AVR's input signal
2. AVR's output field voltage to machine's input field voltage
3. Initially calculated mechanical power to input signal of the machine's mechanical power
4. Machine's power terminal to the generator model power terminal
5. Constant `pss_off` to the PSS input at the AVR
6. Initial generator field voltage to initial AVR field voltage



# Example 1

## Network model – Step 1

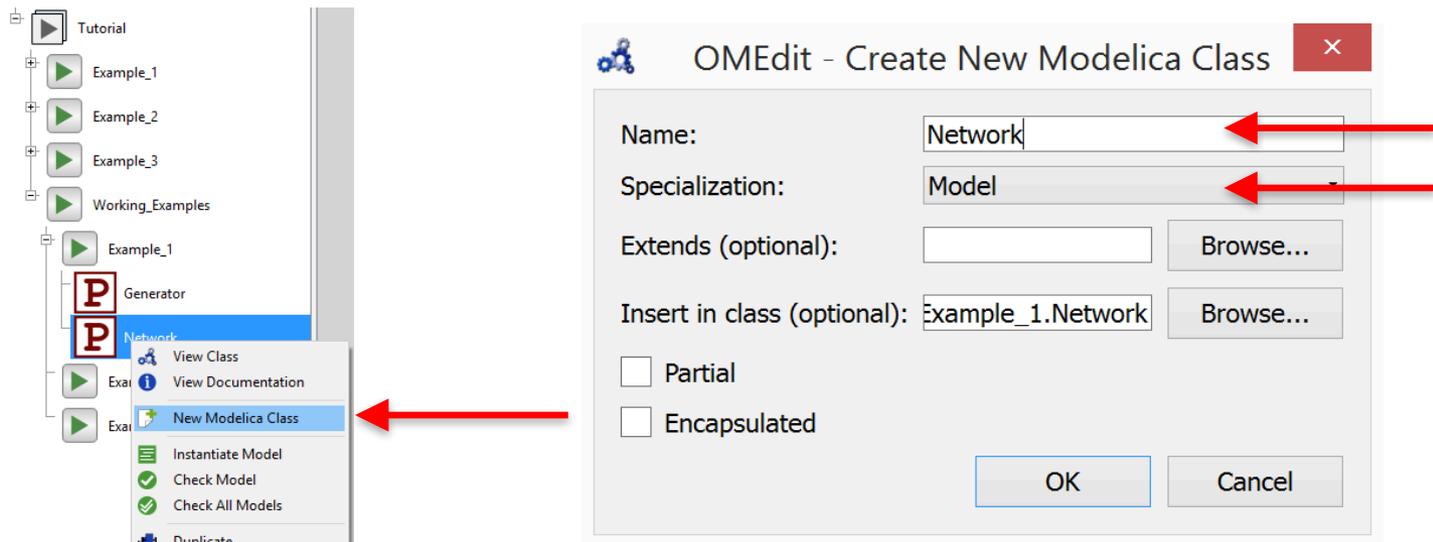
- Network package will be created in the *Example\_1* package
- This package is created by right clicking on the *Example\_1* in the *Working\_Examples* package



# Example 1

## Network model – Step 1

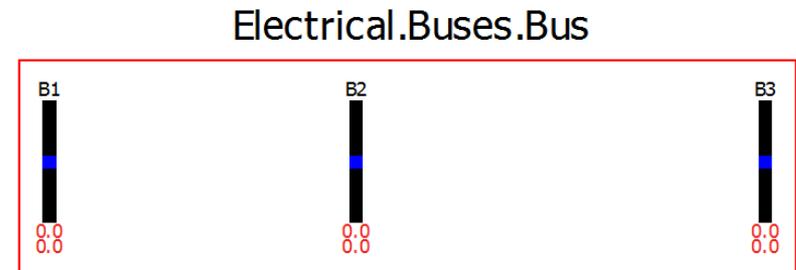
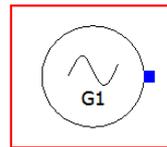
- Network model will be created in the *Network* package
- This package is created by right clicking on the *Network* package
- The name of the network model will be *Example\_1*



# Example 1

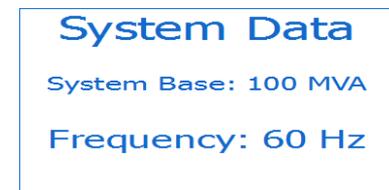
## Network model – Step 1

- Created generator model (name it machine) and three bus models are added to the network model



- Also, model *OpenIPSL.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  |



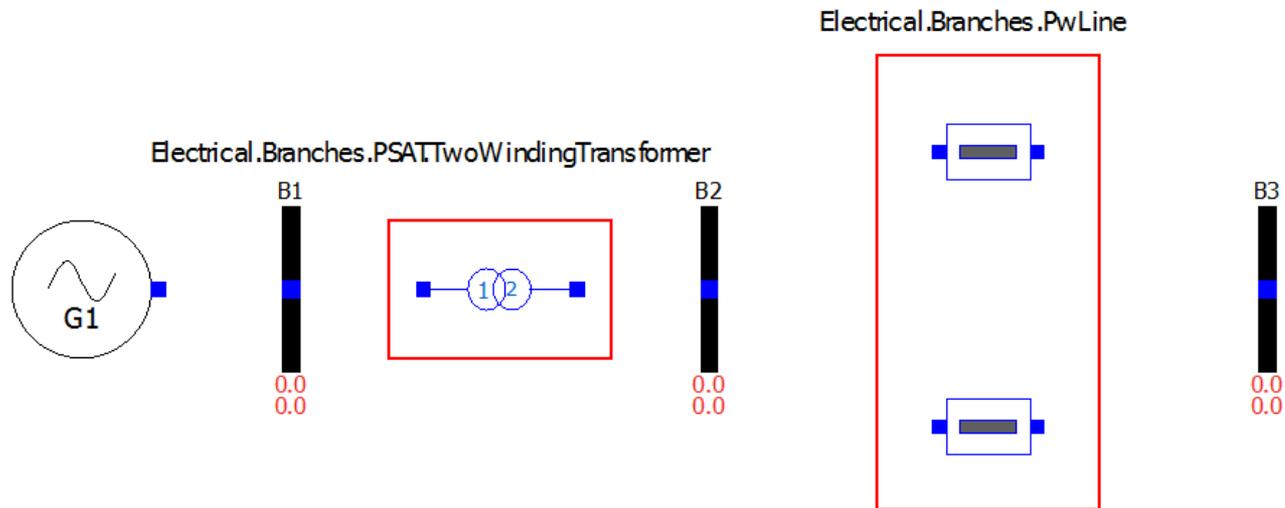
- In *text view* add the **inner** keyword in front of the component declaration

```
inner OpenIPSL.Electrical.SystemBase SysData
```

# Example 1

## Network model – Step 2

- Transformer and line models are added



# Example 1

## Network model – Step 2

- Transformer and line parameters

### Transformer

|       |             |       |             |
|-------|-------------|-------|-------------|
| $S_b$ | Do not edit | $f_n$ | Do not edit |
| $S_n$ | 2220        | $kT$  | 1           |
| $V_b$ | 400         | $x$   | 0.15        |
| $V_n$ | 400         | $r$   | 0           |

### Line 1

|       |                        |     |     |
|-------|------------------------|-----|-----|
| $R$   | 0.0                    | $G$ | 0.0 |
| $X$   | $0.5 \cdot 100 / 2220$ | $B$ | 0.0 |
| $S_b$ | 100                    |     |     |

### Line 2

|       |                             |     |     |
|-------|-----------------------------|-----|-----|
| $R$   | 0.0                         | $G$ | 0.0 |
| $X$   | $0.93 \cdot 100 / 222$<br>0 | $B$ | 0.0 |
| $S_b$ | 100                         |     |     |

# Example 1

## Network model – Step 3

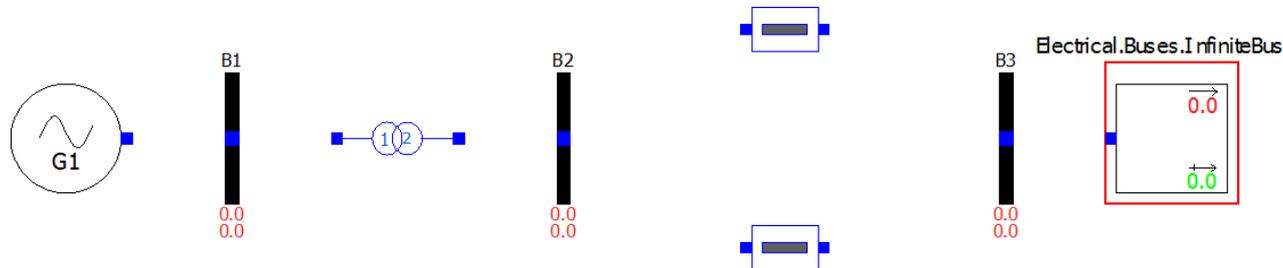
- Infinite bus is added
- Power Flow results are implemented

G1

|   |          |       |        |
|---|----------|-------|--------|
| V | 1        | angle | 0.4946 |
| P | 1997.999 | Q     | 967.92 |

Infinite bus

|   |         |       |        |
|---|---------|-------|--------|
| V | 0.90081 | angle | 0      |
| P | -1998   | Q     | 87.066 |



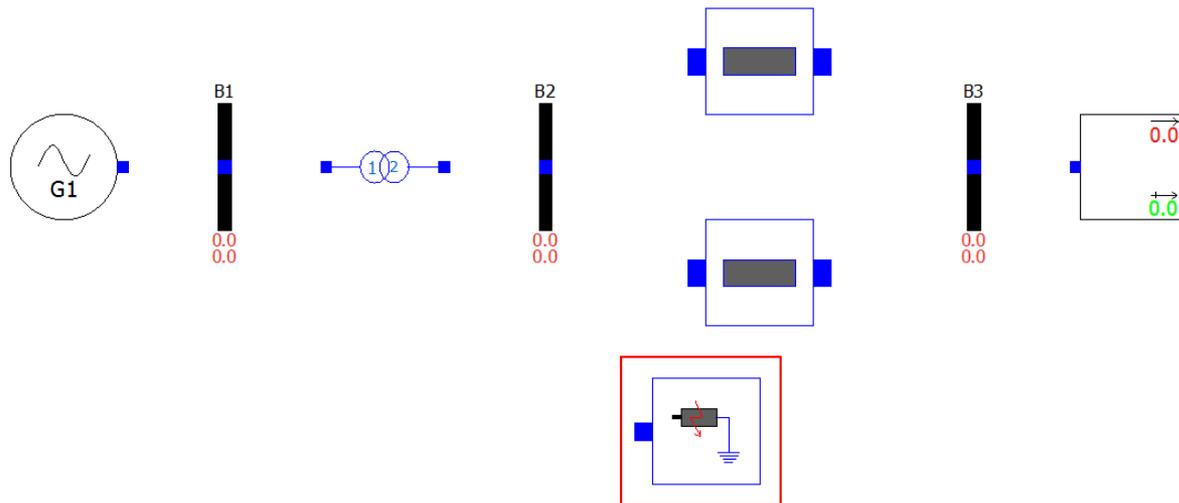
# Example 1

## 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 |

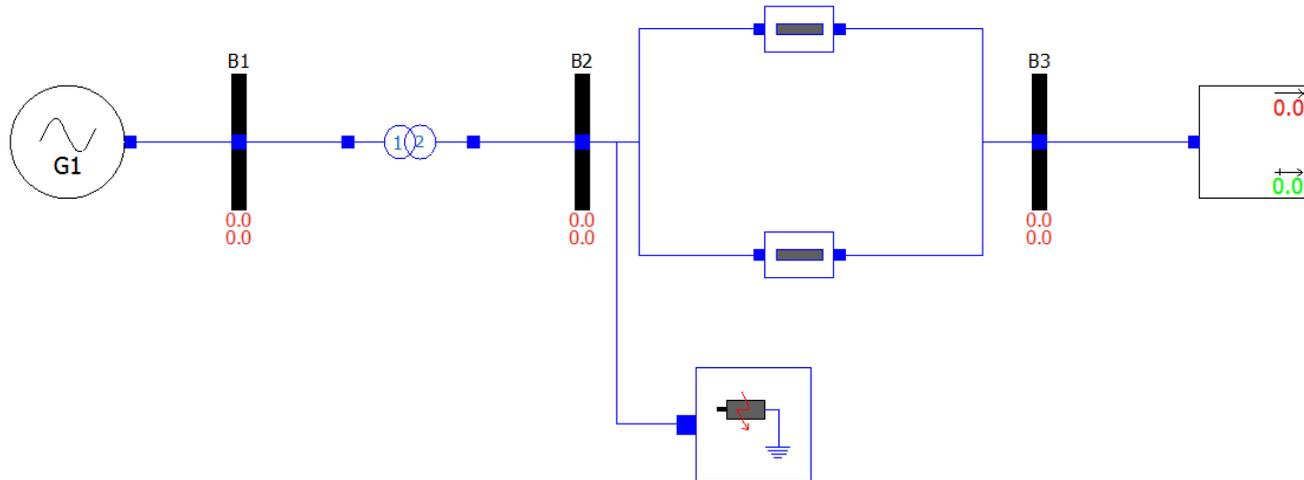


Electrical.Events.PwFault

# 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

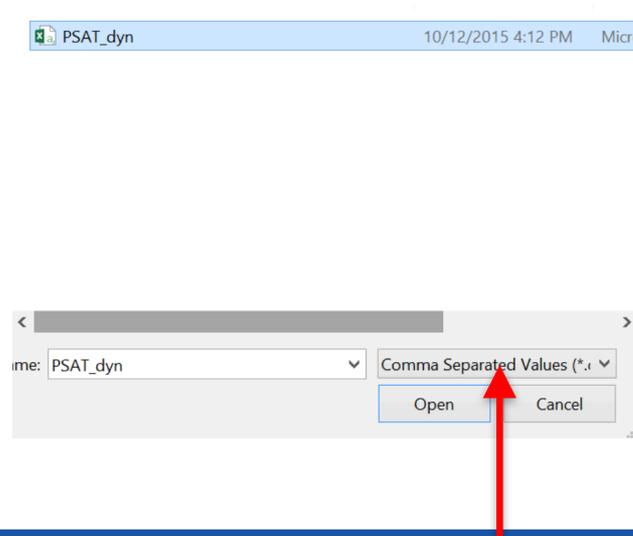
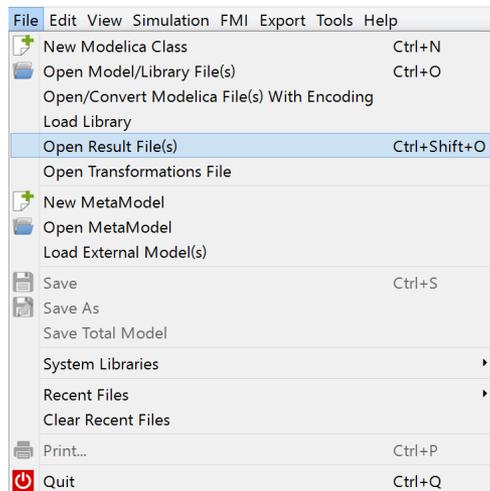
- System will be simulated with 3-phase-to-ground fault at  $t=0.5s$  with a duration of 70ms
- Simulation results will be compared with the reference results from the PSAT that will be loaded first
- PSAT results are provided in a file “PSAT\_dyn.csv”
- To load the file, the view should be switched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



# Example 1

## Simulation

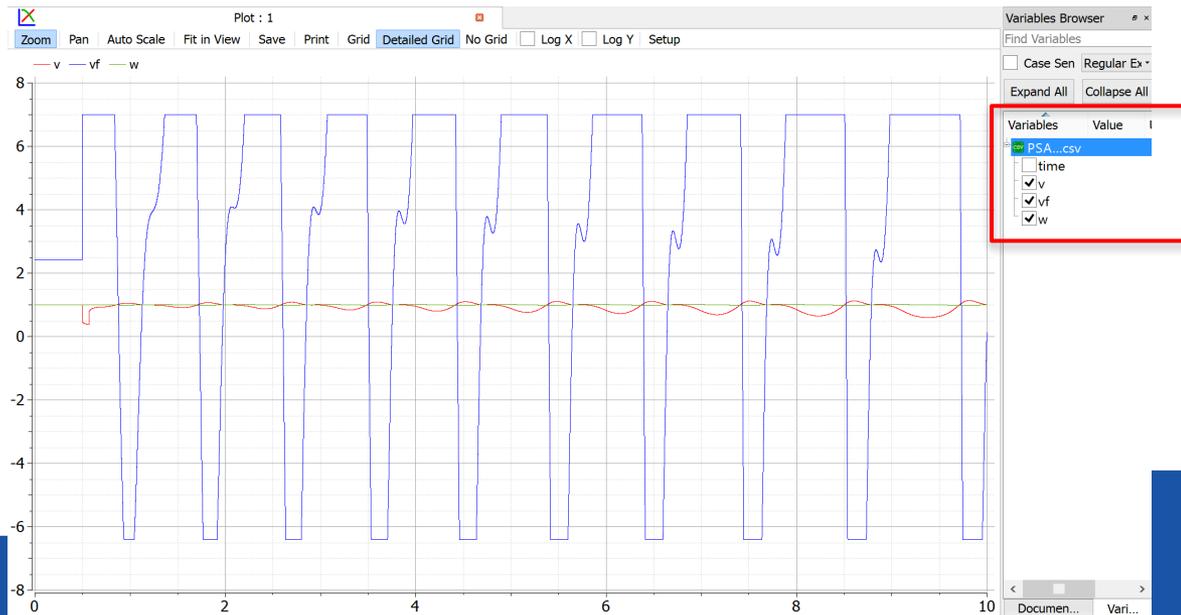
- Result file can be opened by navigating the menu to File->Open Result File(s)
- In the pop-up menu, one has to select “Comma Separated Values” as a file type, navigate to the directory where the file is located and open it



# Example 1

## Simulation

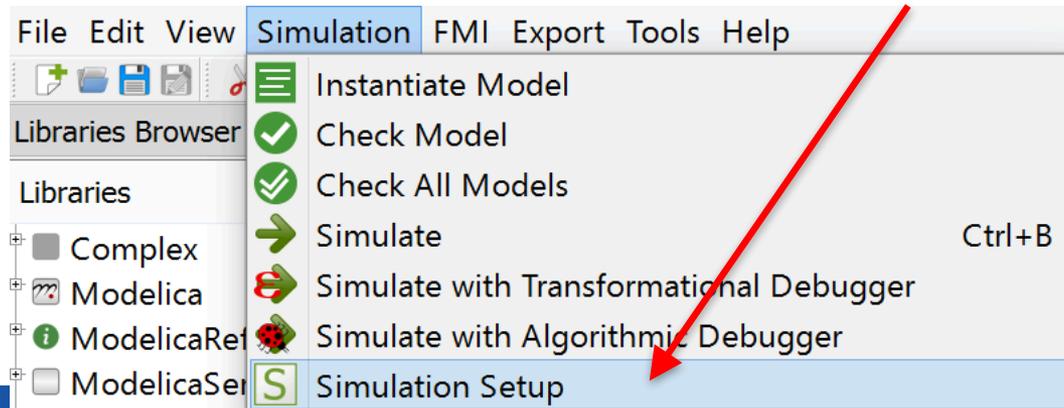
- In the variable browser, three waveforms from the PSAT results are loaded which can be displayed on the plot as it is shown in the figure below
- Loaded waveforms are generator terminal voltage, excitation field voltage and the generator speed



# 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 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 time step is set to 0.0001



Simulation Setup - OpenIPSL.Examples.Machines.PSSE.GENSAL

General Output Simulation Flags Archived Simulations

Simulation Interval

Start Time: 0

Stop Time: 10

Number of Intervals: 500

Interval: 0.0001

Integration

Method: rungekutta

Tolerance:  $1e-6$

DASSL Options

Jacobian: Colored Numerical

Root Finding

Restart After Event

Initial Step Size:

Maximum Step Size:

Maximum Integration Order: 5

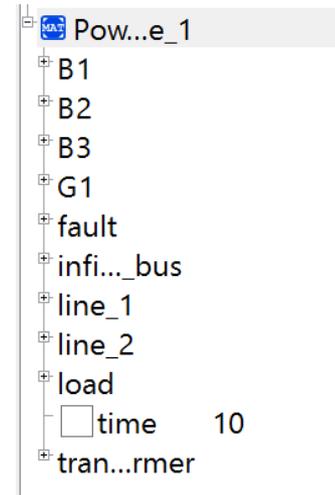
# Example 1

## Simulation

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



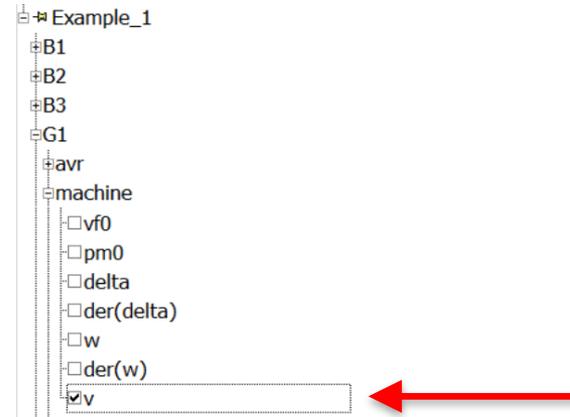
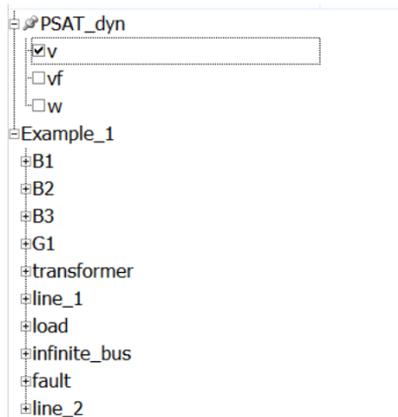
- Once the simulation is completed, the Variable Browser is populated with the simulation results



# Example 1

## Simulation

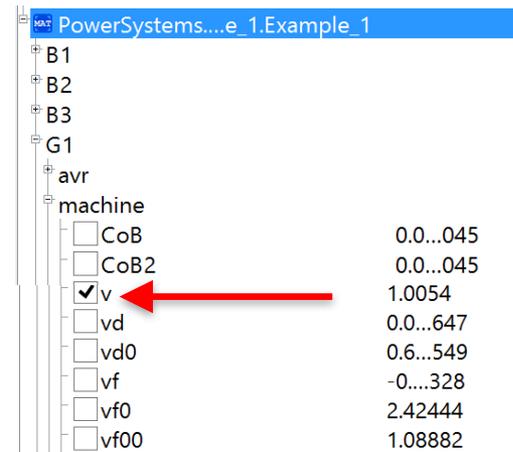
- To display the simulation results or compare it with the results from PSAT, one can mark the check-box next to the variable which will be shown on the plot
- To show the terminal voltage of the generator in PSAT and modelica, variables “PSAT\_dyn.v” and “Example\_1.G1.machine.v” have to be selected



# Example 1

## Simulation

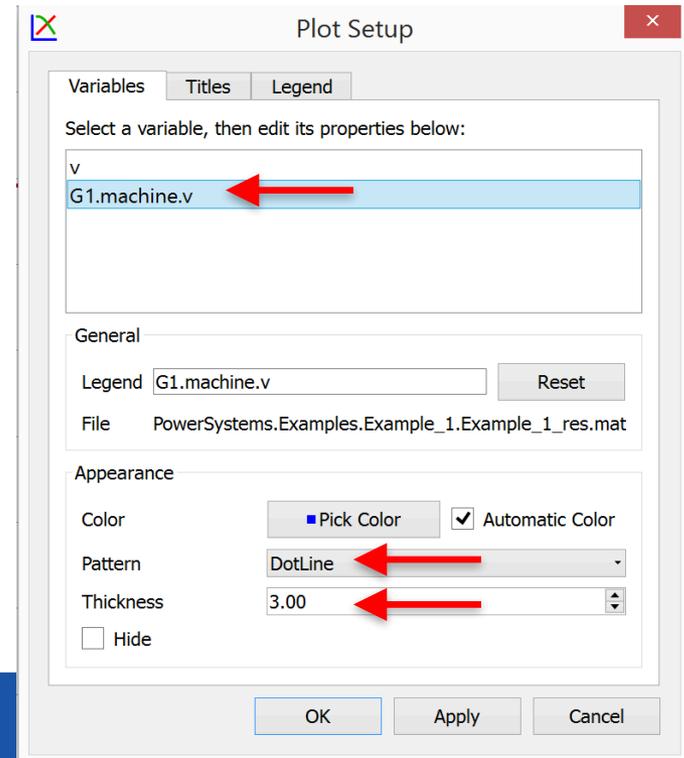
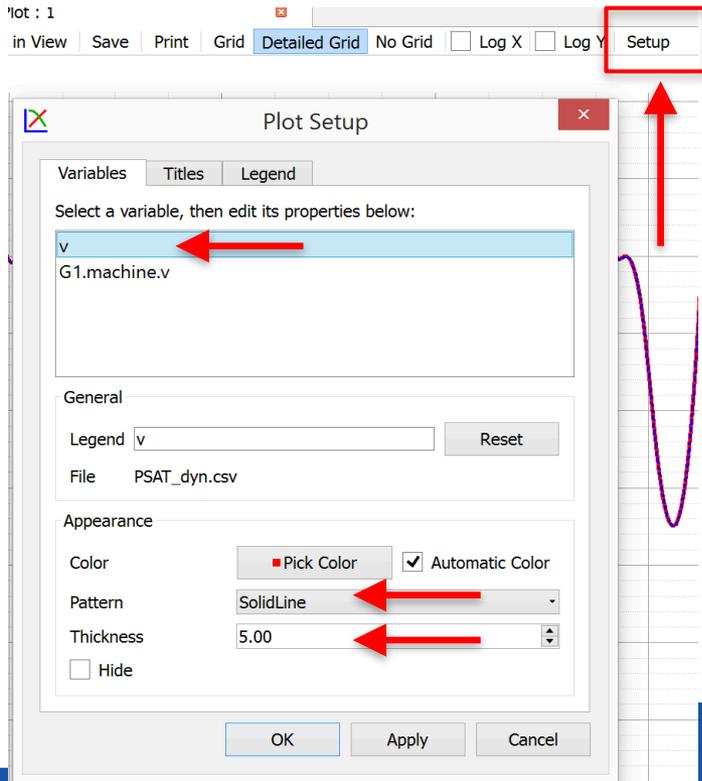
- To display the simulation results or compare it with the results from PSAT, one can mark the check-box next to the variable which will be shown on the plot
- To show the terminal voltage of the generator in PSAT and modelica, variables “PSAT\_dyn.csv.v” and “G1.machine.v” have to be selected



# Example 1

## Simulation

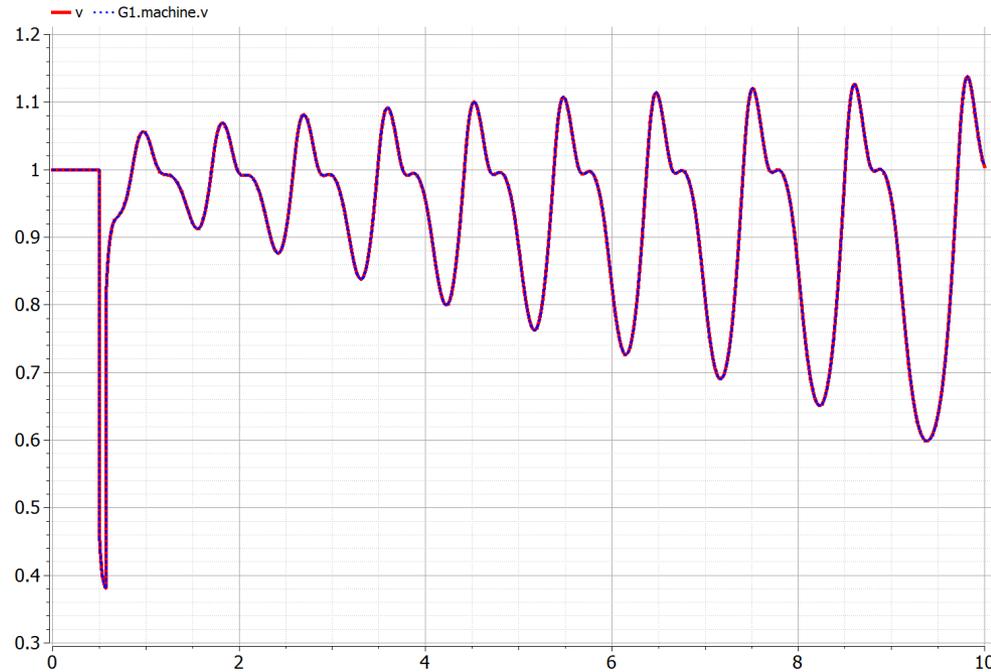
- To be able to distinguish different signals, let's adjust the thickness and the pattern of the signal line



# 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





# Using OMNotebook for Interactive Analysis

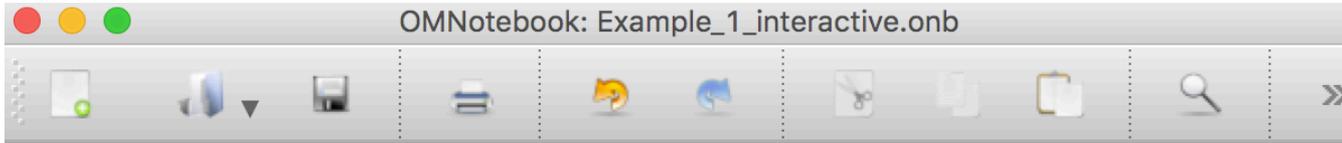
The OpenModelica installation includes a very handy tool called OMNotebook.

OMNotebook allows you to interact with the OpenModelicaCompiler (OMC), and at the same time analyze the model. This gives you a basic “lab notebook” where you do your tests before you create a script for automated analysis (called .mos scripts).

We use OMNotebook next to load the model developed, and simulate it interactively.



# Loading and Simulating your Model



*Example 1 - Interactive Analysis using OMNotebook*  
*Prof. Luigi Vanfretti, 2017-04-19*

```
//Set the current working directory to your installation
cd("/Users/luigiv/Downloads/OpenIPSL-Tuto_TAMU_2017/");
//Load the standard MSL
loadModel(Modelica);
//Load the OpenIPSL
loadFile("../OpenIPSL/package.mo");
// list(OpenIPSL) //enable to check if the library is being loaded
[done]
```

```
// Load the tutorial
cd("../ApplicationExamples/_Tutorial/Tutorial/");
loadFile("package.mo");
//Check everything is in the model
list(Tutorial.Example_1.Example_1);
[done]
```

```
// Instantiate the model of the network
instantiateModel(Tutorial.Example_1.Example_1);
// Simulate the model
simulate(Tutorial.Example_1.Example_1, stopTime=5);
```

Set your patch to where your tutorial .zip is available.

Evaluate each cell using "Shift+Enter"

Instantiate and Simulate the Model



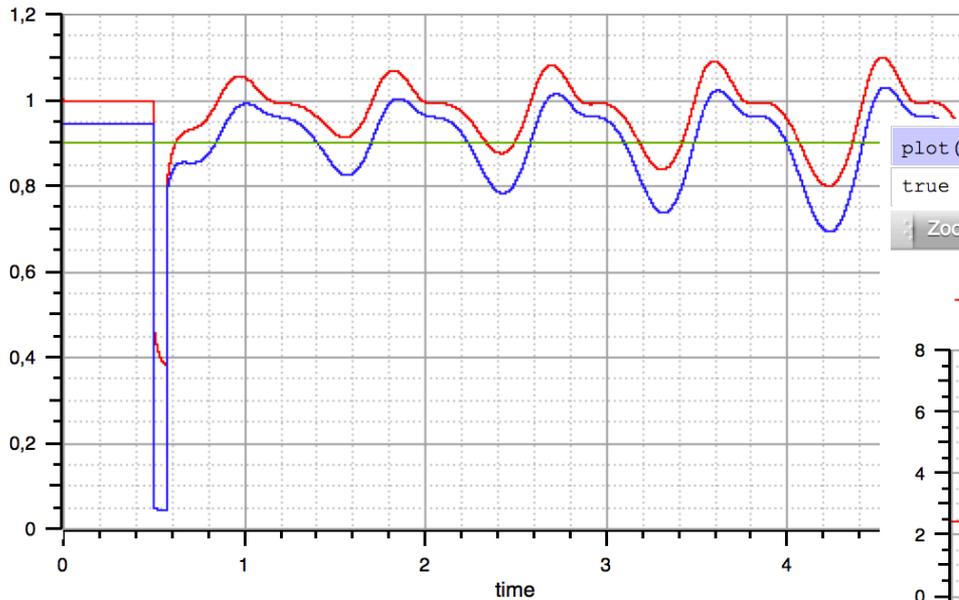
# Plotting in OMNotebook

```
// What can you plot!  
plot({B1.V,B2.V,B3.V})
```

true

Zoom Pan Auto Scale Fit in View Save Print Grid >>

— B1.V — B2.V — B3.V

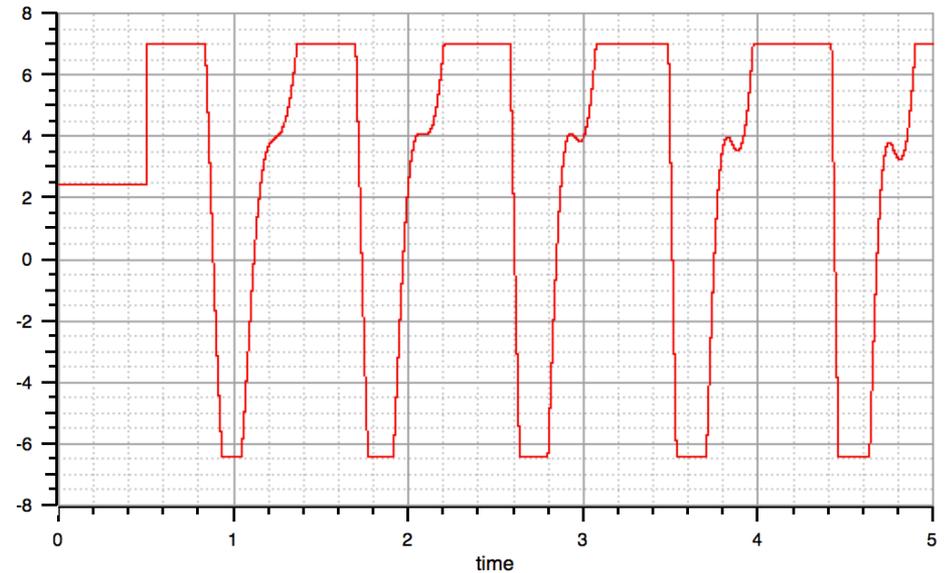


```
plot(G1.avr.vf)
```

true

Zoom Pan Auto Scale Fit in View Save Print Grid >>

— G1.avr.vf





# Parametric Sweep and Simulation using OMNotebook

Using OMNotebook, we can interact with the OMC in order to analyze the behavior of the system.

The AVR's output looks saturated.

The type of instability that we are observing it is typically due to negative feedback from the AVR. This was shown in the early 1900's by Concordia and De Mello!

A parametric sweep is used in OMC to simulate the system's response for different values of the gain of the AVR, namely,  $K_0$ .



# Parametric Sweep and Simulation using OMNotebook

Example 1 - Parametric Sweep  
Prof. Luigi Vanfretti, 2017-04-19

```
//Set the current working directory to your installation
cd("/Users/luigiv/Downloads/OpenIPSL-Tuto_TAMU_2017/");
//Load the standard MSL
loadModel(Modelica);
//Load the OpenIPSL
loadFile("../OpenIPSL/package.mo");
// list(OpenIPSL) //enable to check if the library is being loaded
// Load the tutorial
cd("../ApplicationExamples/_Tutorial/Tutorial/");
loadFile("package.mo");
// Build the model once
buildModel(Tutorial.Example_1.Example_1);
```

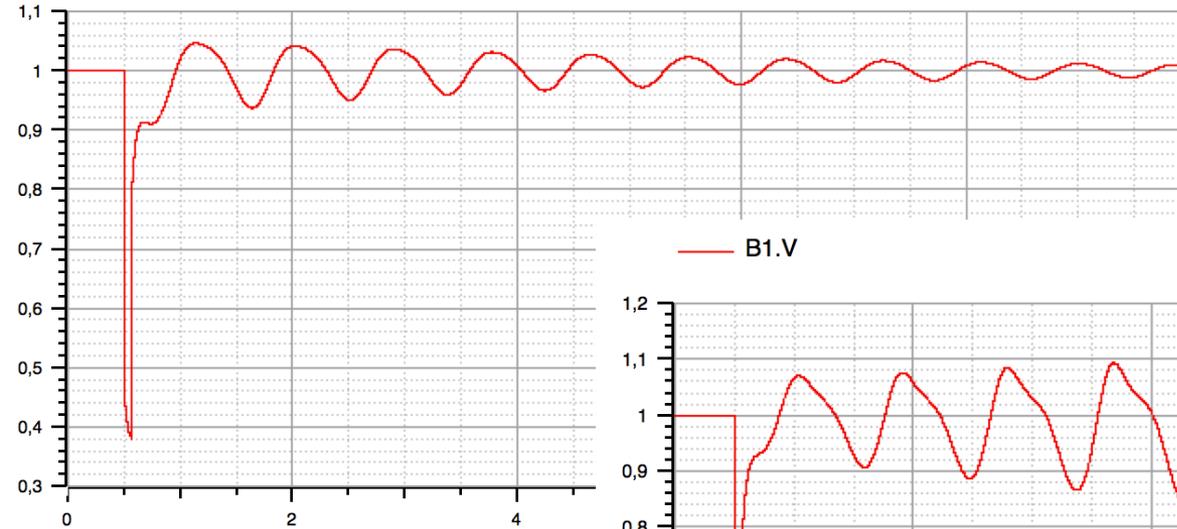
```
// Do the parametric sweep using a loop
getErrorString();
for i in 1:10 loop
  // We update the parameter using value
  value := i*10;
  // call the generated simulation code to produce a result files %i%_res.mat
  system("../Tutorial.Example_1.Example_1 -override=G1.avr.K0="+String(value)+" -r=Example_1_parametric"
+ String(i) + "_res.mat");
  getErrorString();
end for
```

```
""
```

```
plot({B1.V}, fileName="Example_1_parametric1_res.mat")
```

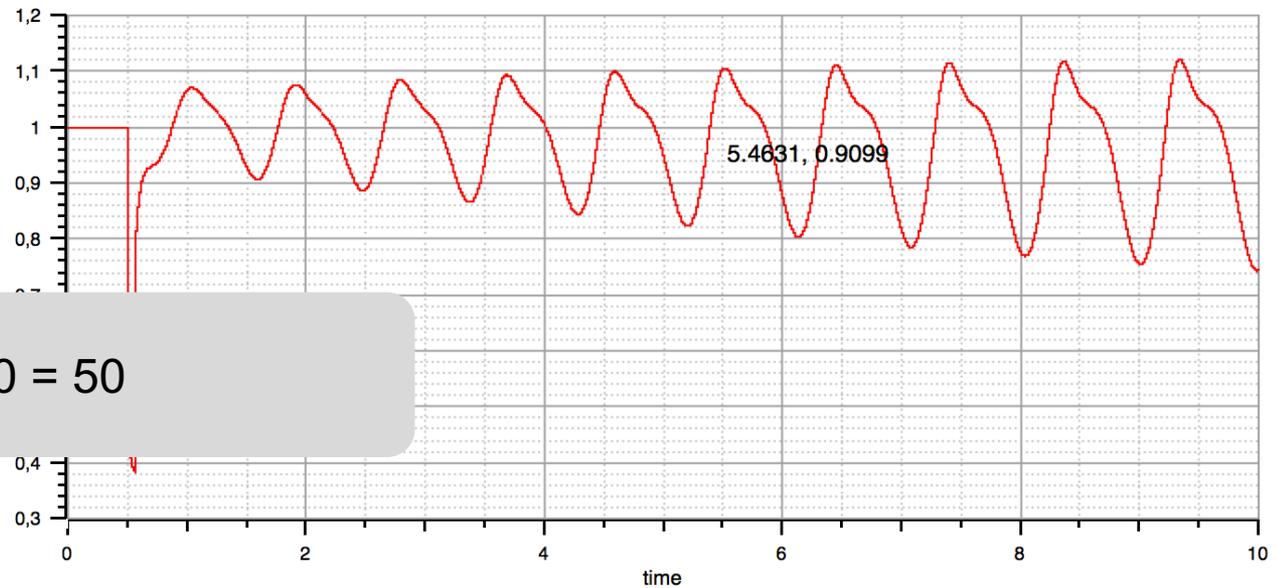
# Parametric Sweep and Simulation using OMNotebook

B1.V



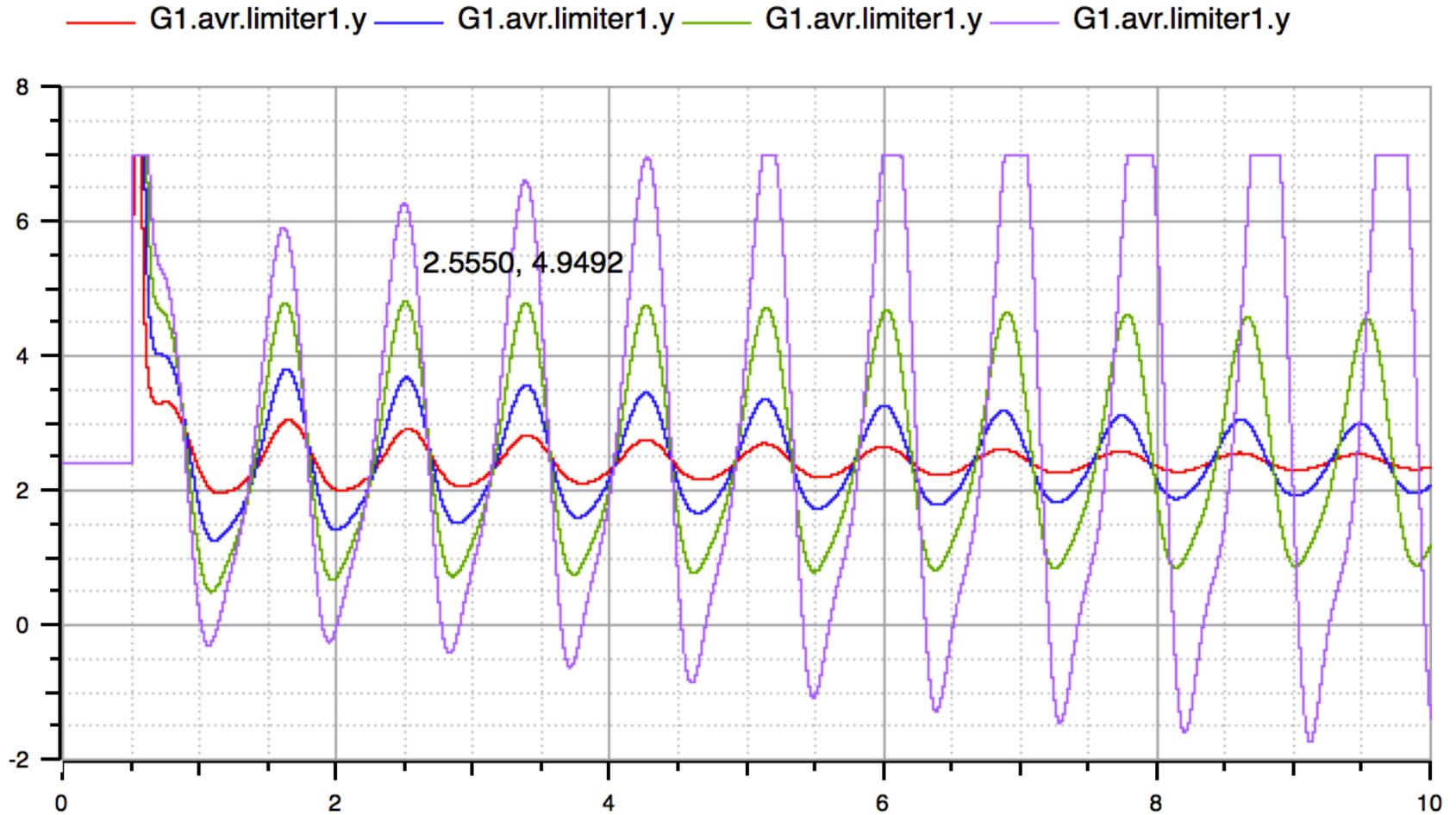
$K0 = 10$

B1.V



$K0 = 50$

# Parametric Sweep and Simulation using OMNotebook





# Linear Analysis

Let's now do a bit more analysis on the system's stability by using linear analysis.

This will allow us to determine the system's damping with the current value of the AVR's gain ( $K=200$ ).

Two methods are shown in the following slides, using OMNotebook and the OMSHELL (optional).

```
// Instantiate the model of the network  
instantiateModel(Tutorial.Example_1.Example_1);  
// Linearize the model  
linearize(Tutorial.Example_1.Example_1, stopTime=0);
```



# Linear Analysis

```
list(linear_Tutorial_Example__1_Example__1)
```

```
"model linear_Tutorial_Example__1_Example__1
  parameter Integer n = 9;
  // states
  parameter Integer k = 0;
  // top-level inputs
  parameter Integer l = 0;
  // top-level outputs
  parameter Real x0[9] = {2.420701642076922, 1, 0, 1.224247464404018, 0.4107654957816429, 1.02748702363574,
0.5742496657902401, 0.9593327097152915, 1};
  parameter Real u0[0] = {i for i in 1:0};
  parameter Real A[9, 9] = [-10000, -2000000, 10000, 0, 0, 0, 0, 0, 0; 0, -66.66666666666667, 0,
-9.193997621986762, 6.549112611627166e-15, 0, 29.25536745530336, 33.36361865698515, -5.06776165969379e-16;
0, 0, -1, 0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 376.9911184307751; 0, 0, 0, 0.4610796242587042, -1, 0,
-1.489428945747467, 0.006336424213860586, -1.215373955322828e-17; 0.12496875, 0, 0, -0.2257865911752657,
6.023845086408405e-17, -0.125, -0.001104105716065041, -0.266889889874036, 3.011922543204202e-17; 0, 0, 0,
2.62155903181871, 14.28571428571428, 0, -22.75415595943628, 0.03602698808039422, -6.910248039557863e-17;
0.008333333333333333, 0, 0, -2.972830759729272, 7.931326599823655e-16, 33.33333333333334,
-0.01453726466937616, -36.84735278837885, 3.965663299911827e-16; -0, -0, -0, -0.1593501250388591,
8.249174533413525e-17, -0, 0.06000759107001079, -0.1730203086038577, 1.487405919582988e-17];
  parameter Real B[9, 0] = zeros(9, 0);
  parameter Real C[0, 9] = zeros(0, 9);
  parameter Real D[0, 0] = zeros(0, 0);
```



# Linear Analysis

Now we copy and paste the A matrix of the model defined above, the assignment `A:= [...]`; makes the A matrix available to the OpenModelicaCompiler.

Once in the OMC, we use the MSL mathematical functions to compute the eigenvalues and eigenvectors!

```
A := [-10000, -2000000, 10000, 0, 0, 0, 0, 0, 0; 0, -66.66666666666667, 0, -9.193997621986762,
6.549112611627166e-15, 0, 29.25536745530336, 33.36361865698515, -5.06776165969379e-16; 0, 0, -1, 0, 0, 0,
0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 376.9911184307751; 0, 0, 0, 0.4610796242587042, -1, 0, -1.489428945747467,
0.006336424213860586, -1.215373955322828e-17; 0.12496875, 0, 0, -0.2257865911752657, 6.023845086408405e-17,
-0.125, -0.001104105716065041, -0.266889889874036, 3.011922543204202e-17; 0, 0, 0, 2.62155903181871,
14.28571428571428, 0, -22.75415595943628, 0.03602698808039422, -6.910248039557863e-17;
0.008333333333333333, 0, 0, -2.972830759729272, 7.931326599823655e-16, 33.33333333333334,
-0.01453726466937616, -36.84735278837885, 3.965663299911827e-16; -0, -0, -0, -0.1593501250388591,
8.249174533413525e-17, -0, 0.06000759107001079, -0.1730203086038577, 1.487405919582988e-17];
```

```
[done]
```

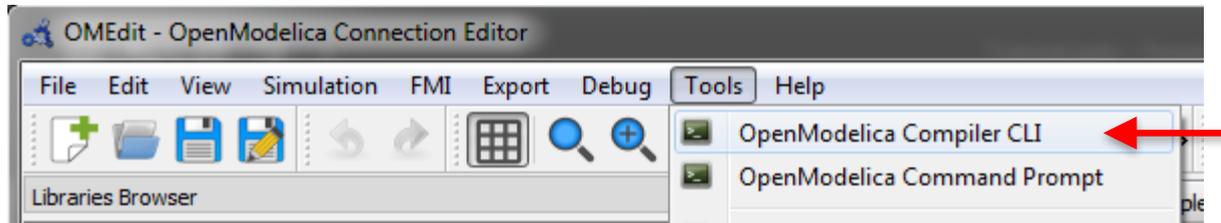
```
(eval, vec) := Modelica.Math.Matrices.eigenValues(A);
eval
```

```
{{-10000.00533773918, 0.0}, {-74.9958055563845, 0.0}, {-15.08153495326777, 13.52584213953},
{-15.08153495326777, -13.52584213953}, {-21.1415314117239, 0.0}, {0.351753399829141, 8.06568059314465},
{0.351753399829141, -8.06568059314465}, {-1.790937600311332, 0.0}, {-1.0, 0.0}}
```

# Example 1 – Linearization using OMShell

## Linearization

- To linearize the system, OpenModelica scripting will be needed



- Along with the library, a set of commands was provided (Command\_List.txt) to linearize the model and extract the A matrix

```
OMShell - OpenModelica Shell
File Edit Help
OMShell 1.1 Copyright Open Source Modelica Consortium (OSMC) 2002-2015
Distributed under OSMC-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.

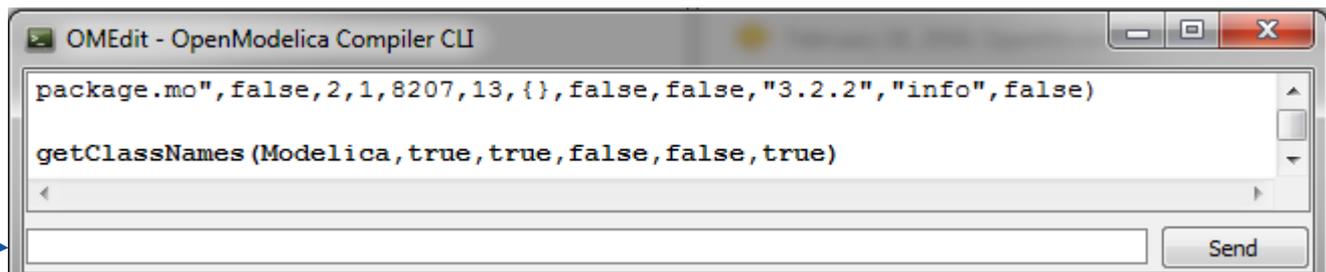
>>
```

# Example 1 – Linearization using OMShell

## Linearization

- Copy and paste each line from the Command\_List.txt for Example 1 to the command prompt in OpenModelica

```
# Example 1  
linearize(Tutorial.Example_1.Example_1,stopTime=0.0)  
loadFile("linear_Tutorial.Example_1.Example_1.mo")  
(a) := getParameterValue(linear_Tutorial_Example__1_Example__1,"A")  
  
(eval,eval) := Modelica.Math.Matrices.eigenValues(A);
```

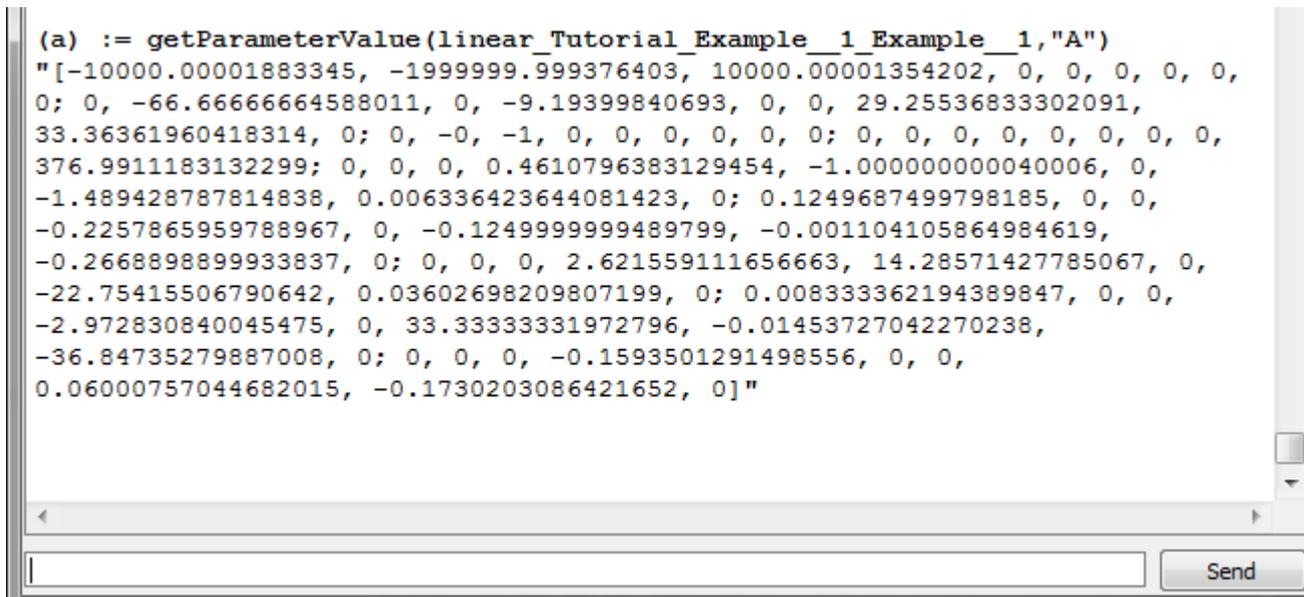


# Example 1 – Linearization using OMShell

## Linearization

- The third command will save the A matrix of the linearized state-space model in the variable `a` as a string

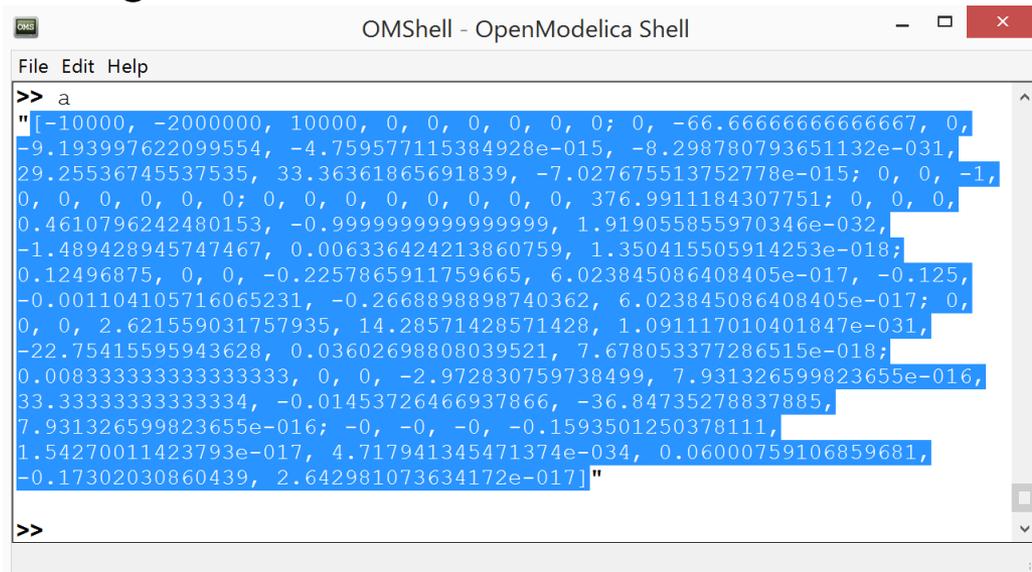
```
(a) := getParameterValue(linear_Tutorial_Example_1_Example_1,"A")
"[-10000.00001883345, -1999999.999376403, 10000.00001354202, 0, 0, 0, 0, 0, 0,
0; 0, -66.66666664588011, 0, -9.19399840693, 0, 0, 29.25536833302091,
33.36361960418314, 0; 0, -0, -1, 0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0,
376.9911183132299; 0, 0, 0, 0.4610796383129454, -1.000000000040006, 0,
-1.489428787814838, 0.006336423644081423, 0; 0.1249687499798185, 0, 0,
-0.2257865959788967, 0, -0.1249999999489799, -0.001104105864984619,
-0.2668898899933837, 0; 0, 0, 0, 2.621559111656663, 14.28571427785067, 0,
-22.75415506790642, 0.03602698209807199, 0; 0.008333362194389847, 0, 0,
-2.972830840045475, 0, 33.33333331972796, -0.01453727042270238,
-36.84735279887008, 0; 0, 0, 0, -0.1593501291498556, 0, 0,
0.06000757044682015, -0.1730203086421652, 0]"
```

A screenshot of the OMShell terminal window. The terminal displays a command to retrieve the A matrix of a linearized state-space model. The output is a long string representing the matrix elements, enclosed in double quotes. The window has a scroll bar on the right and a 'Send' button at the bottom right.

# Example 1 – Linearization using OMShell

## Linearization

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

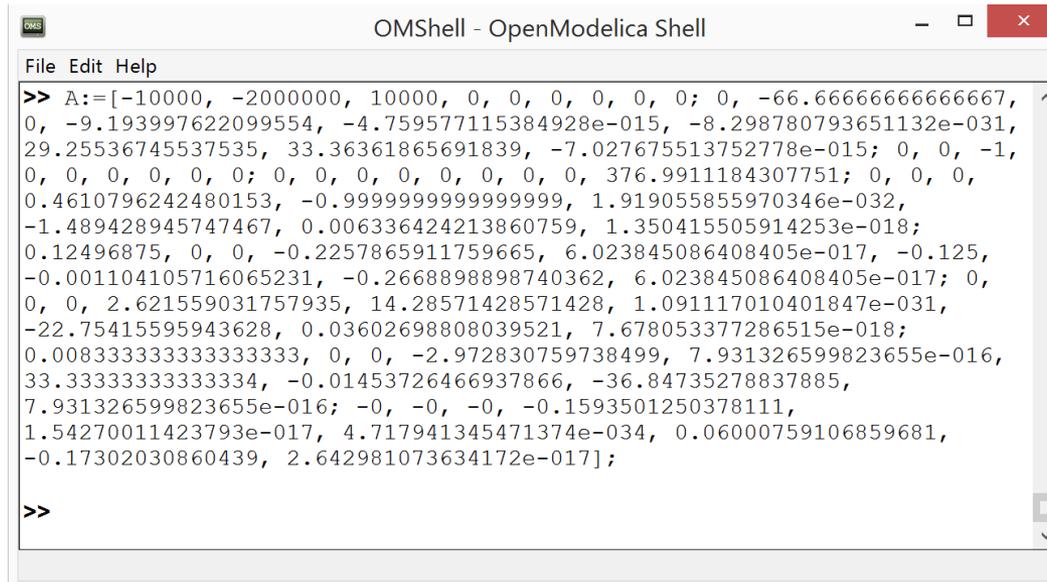


```
OMShell - OpenModelica Shell
File Edit Help
>> a
"[-10000, -2000000, 10000, 0, 0, 0, 0, 0, 0; 0, -66.66666666666667, 0,
-9.193997622099554, -4.759577115384928e-015, -8.298780793651132e-031,
29.25536745537535, 33.36361865691839, -7.027675513752778e-015; 0, 0, -1,
0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 0, 376.9911184307751; 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;
0.008333333333333333, 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 using OMShell

## Linearization

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

A screenshot of the OMShell (OpenModelica Shell) window. The window title is 'OMShell - OpenModelica Shell'. It has a menu bar with 'File', 'Edit', and 'Help'. The main text area contains a long list of numerical values in a matrix format, starting with '>> A:=[-10000, -2000000, 10000, 0, 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, 0, 0, 376.9911184307751; 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; 0.008333333333333333, 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];'. The prompt '>>' is visible at the bottom of the text area.

```
>> A:=[-10000, -2000000, 10000, 0, 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, 0, 0, 376.9911184307751; 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; 0.008333333333333333, 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];

>>
```

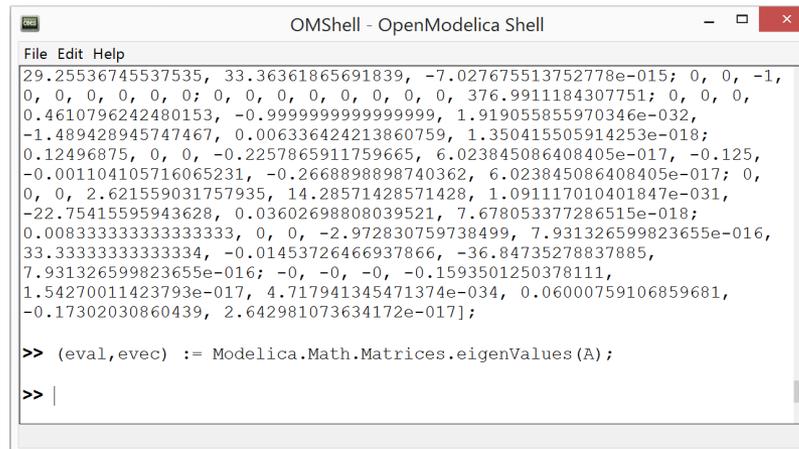
# Example 1 – Linearization using OMShell

## 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 the last command  
(eval, evec) := Modelica.Math.Matrices.eigenValues(A);



```

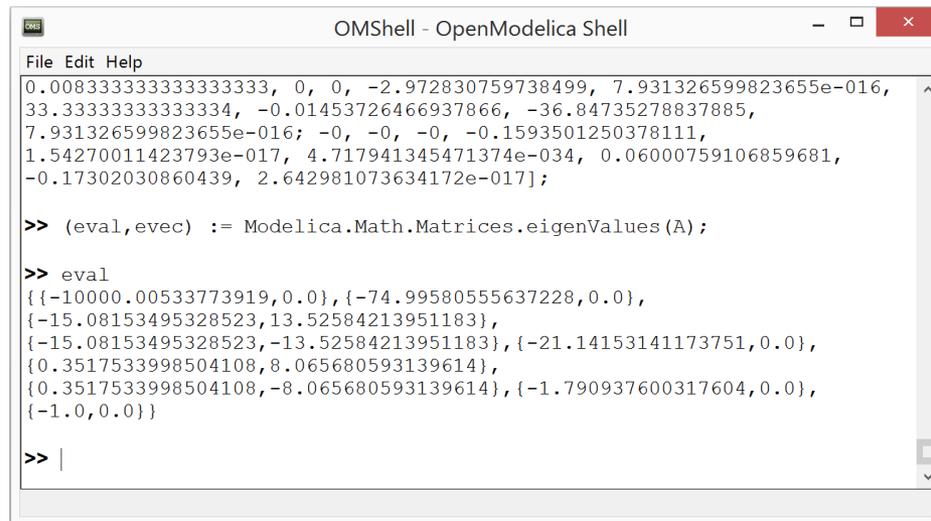
OMShell - OpenModelica Shell
File Edit Help
29.25536745537535, 33.36361865691839, -7.027675513752778e-015; 0, 0, -1,
0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 376.9911184307751; 0, 0, 0,
0.4610796242480153, -0.9999999999999999, 1.919055855970346e-032,
-1.489428945747467, 0.006336424213860759, 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;
0.008333333333333333, 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);
>> |
  
```

# Example 1 – Linearization using OMShell

## 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

A screenshot of the OMShell (OpenModelica Shell) terminal window. The window title is 'OMShell - OpenModelica Shell'. The terminal shows the execution of the `eval` command, which outputs a list of eigenvalues. The output is a list of pairs of numbers, where the first number is the real part and the second is the imaginary part of the pole. The terminal text is as follows:

```
File Edit Help
0.008333333333333333, 0, 0, -2.972830759738499, 7.931326599823655e-016,
33.333333333333334, -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,vec) := Modelica.Math.Matrices.eigenValues(A);

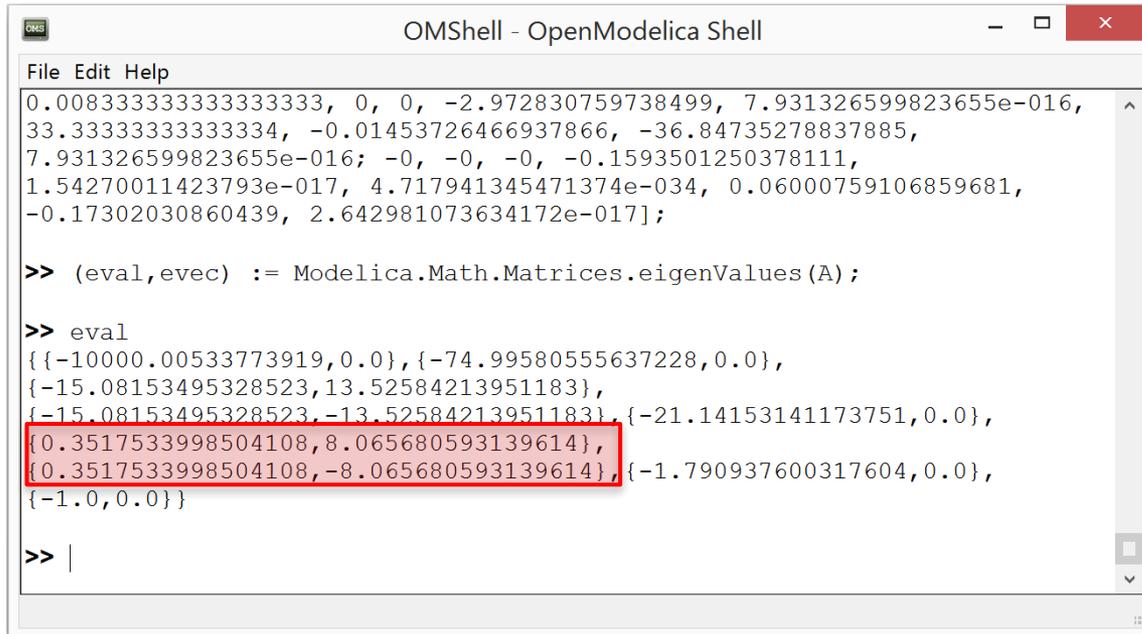
>> eval
{{-10000.00533773919,0.0},{-74.99580555637228,0.0},
{-15.08153495328523,13.52584213951183},
{-15.08153495328523,-13.52584213951183},{-21.14153141173751,0.0},
{0.3517533998504108,8.065680593139614},
{0.3517533998504108,-8.065680593139614},{-1.790937600317604,0.0},
{-1.0,0.0}}

>> |
```

# Example 1 – Linearization using OMShell

## Linearization

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



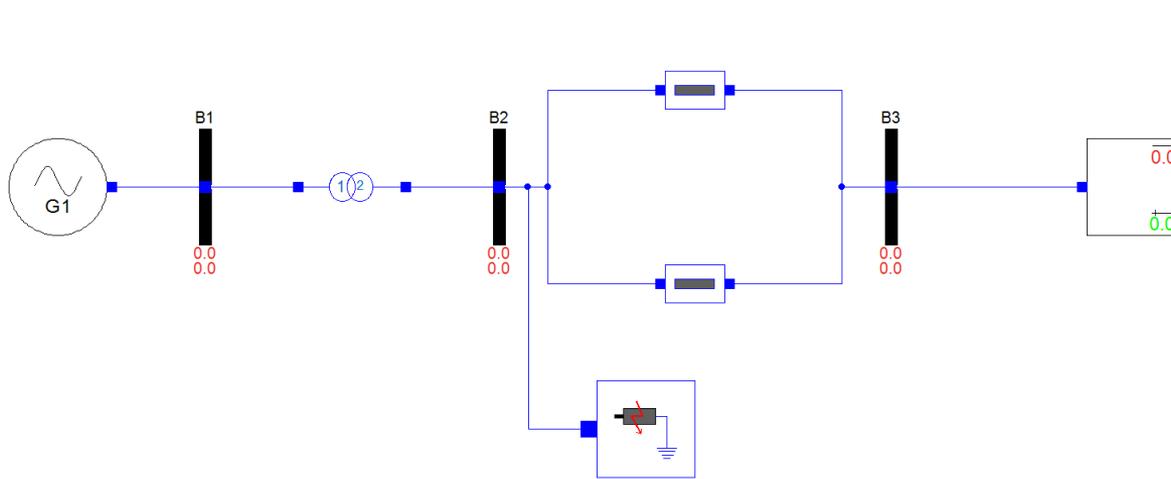
```
OMShell - OpenModelica Shell
File Edit Help
0.008333333333333333, 0, 0, -2.972830759738499, 7.931326599823655e-016,
33.333333333333334, -0.01453726466937866, -36.84735278837885,
7.931326599823655e-016; -0, -0, -0, -0.1593501250378111,
1.54270011423793e-017, 4.717941345471374e-034, 0.06000759106859681,
-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}}

>> |
```

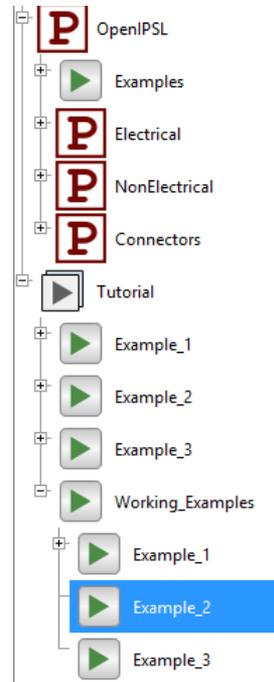
## Example 2



- 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

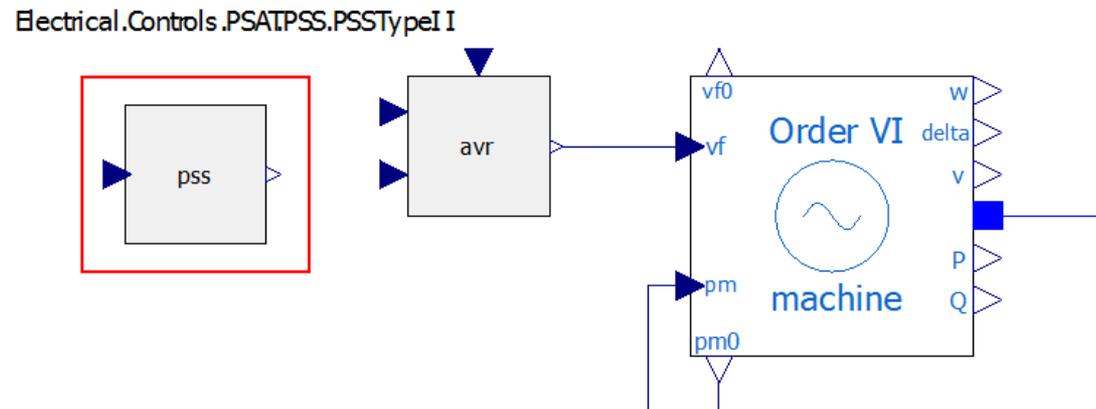
- The work on Example 2 should continue with the files prepared in a package `Tutorial.Working_Examples.Example_2`



# Example 2

## Generator model – Step 1

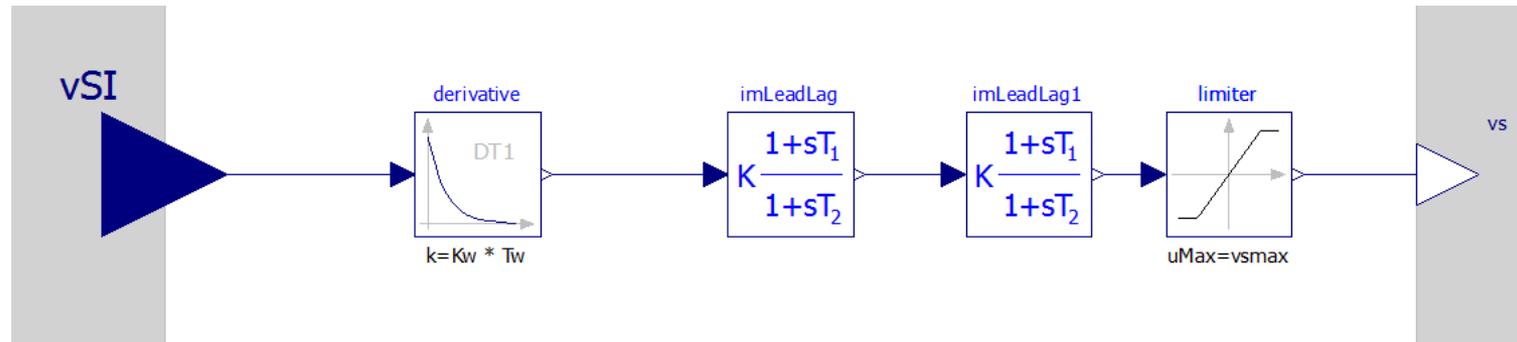
- The first step is to add the model of the PSS Type II and the summation block to the model of the generator



# 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

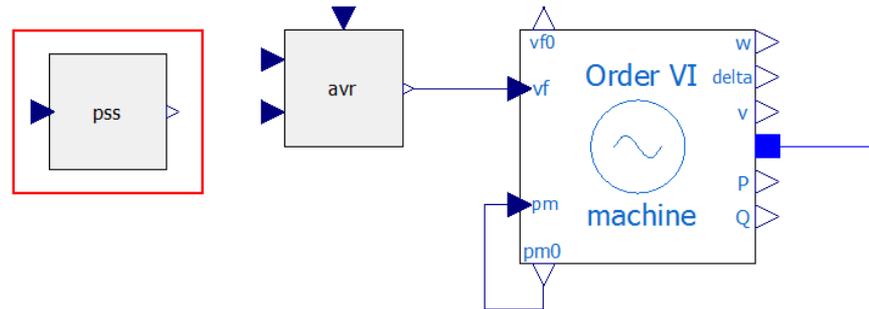
## Generator – Step 1

- 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 |
| $K_w$       | 9.5  | $T_3$ | 1     |
| $T_w$       | 1.41 | $T_4$ | 1     |

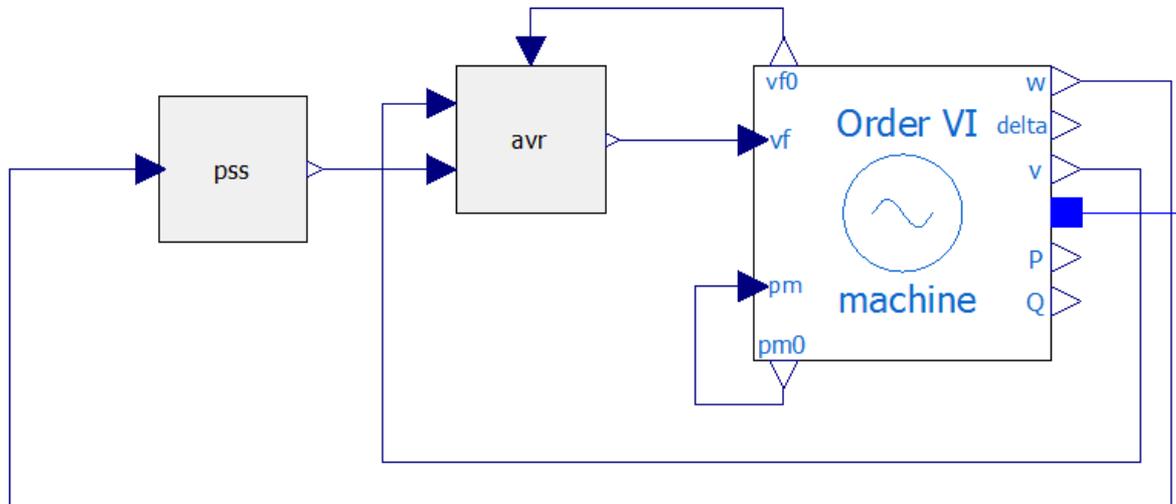
Electrical.Controls.PSATPSS.PSSTypeI I



## Example 2

### Generator – Step 2

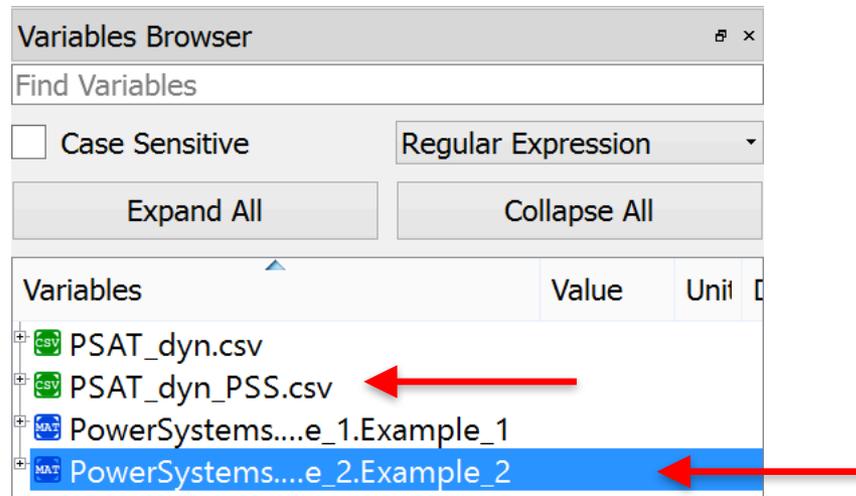
- When the signals of the generator model are connected as shown, model of the generator is completed



## Example 2

### Simulation

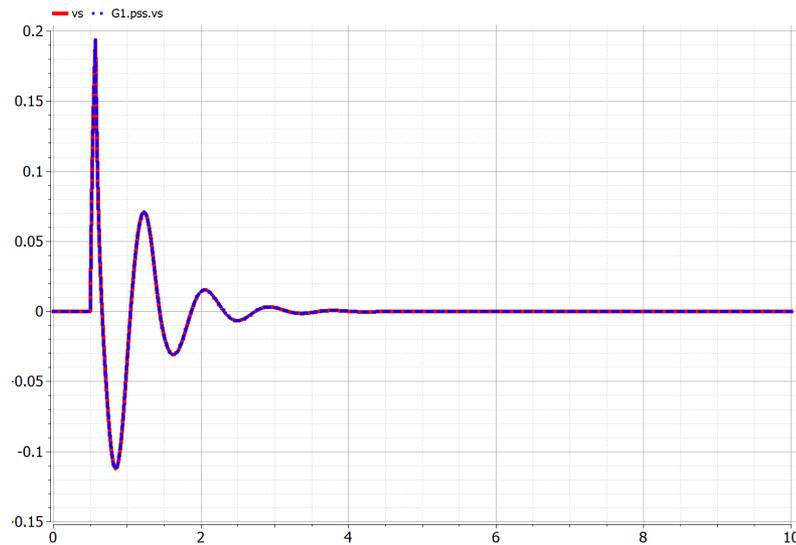
- Simulation steps can be repeated as it was shown in the Example 1
- This time, reference simulation results from the PSAT can be found in the file “PSAT\_dyn\_PSS.csv”
- After the simulation is executed, variable browser should look as it is shown below



## Example 2

### Simulation

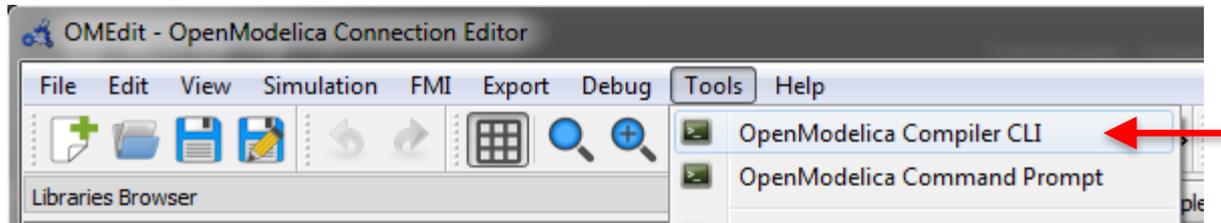
- Simulation results can be plotted again
- Comparison of the PSAT and Modelica simulation results of the PSS signal is shown on the figure below



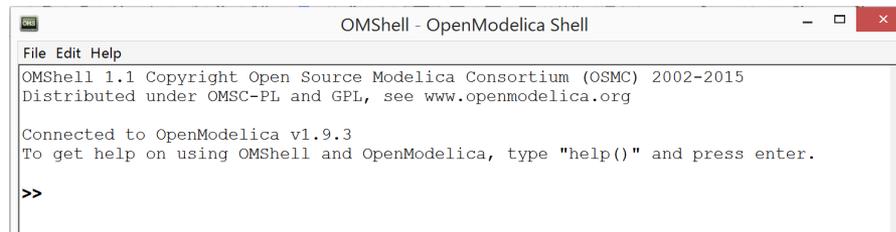
## Example 2

### Linearization

- To linearize the system, OpenModelica scripting will be needed



- Along with the library, a set of commands was provided (Command\_List.txt) to linearize the model and extract the A matrix



```
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.

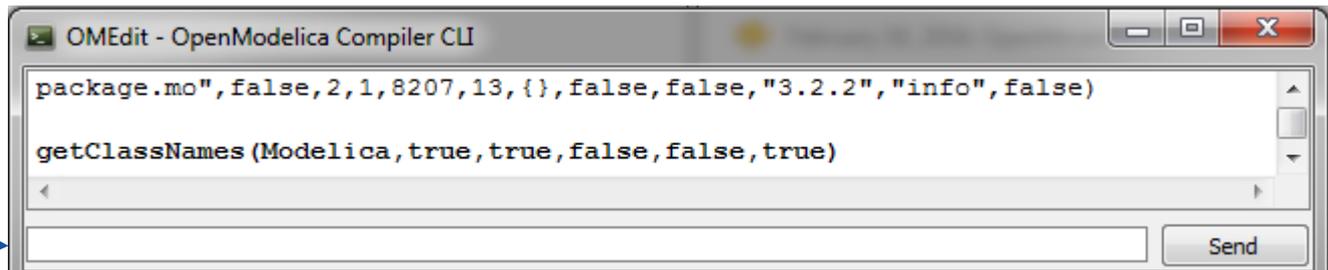
>>
```

# Example 1

## Linearization

- Copy and paste each line from the Command\_List.txt for Example 1 to the command prompt in OpenModelica

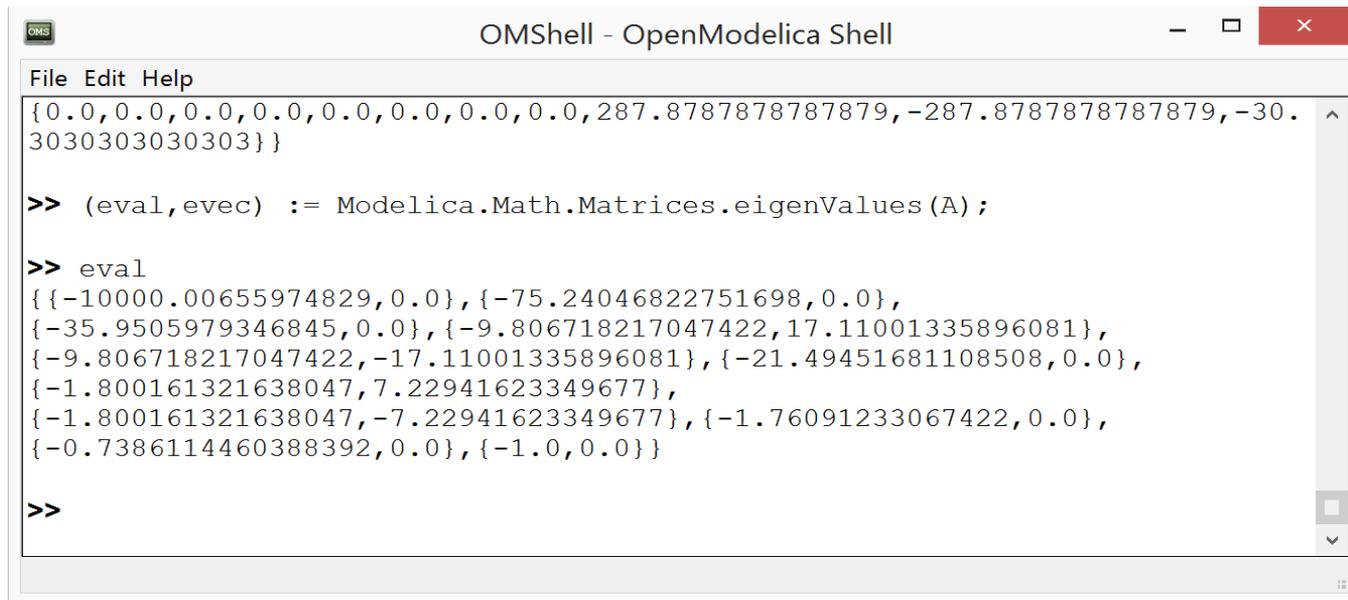
```
# Example 2  
linearize(Tutorial.Example_2.Example_2, stopTime=0.0)  
loadFile("linear_Tutorial.Example_2.Example_2.mo")  
(a) := getParameterValue(linear_Tutorial_Example__2_Example__2, "A")  
(eval, evec) := Modelica.Math.Matrices.eigenValues(A);
```



# 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

A screenshot of the OMS Shell (OpenModelica Shell) terminal window. The window title is 'OMShell - OpenModelica Shell'. The terminal shows a list of eigenvalues and the execution of a command to calculate the eigenvalues of a matrix A. The output shows a list of complex eigenvalues.

```
File Edit Help
{0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,287.8787878787879,-287.8787878787879,-30.
303030303030303}}

>> (eval,eval) := 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

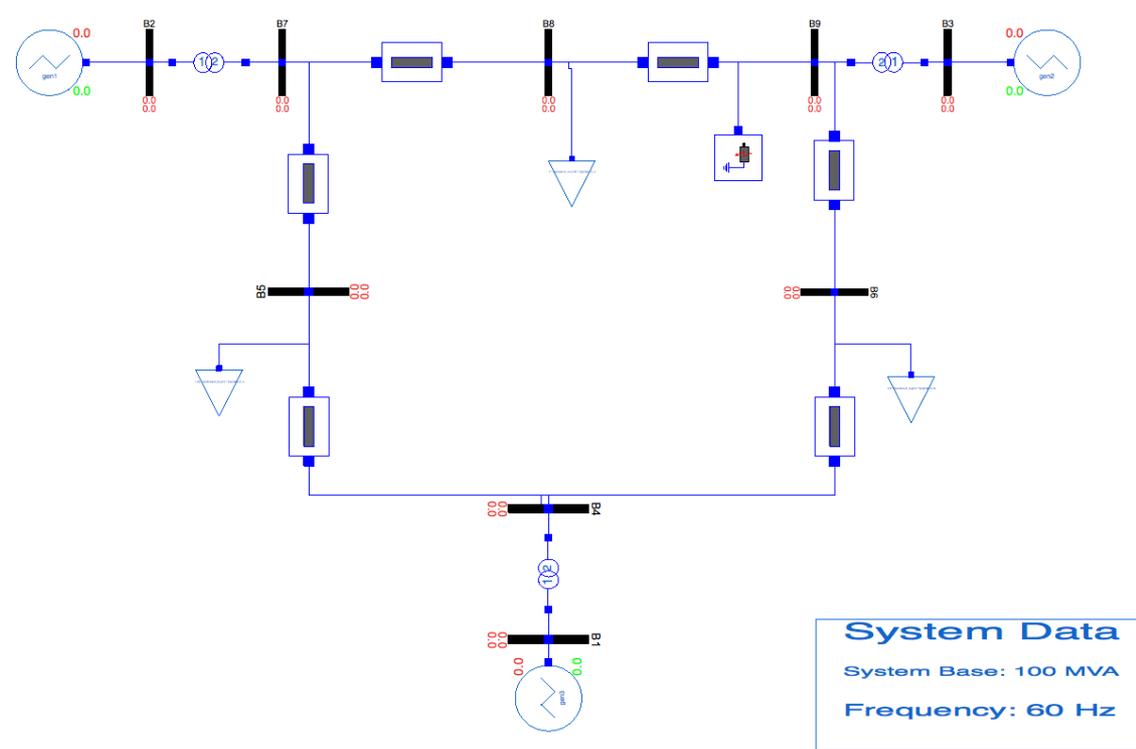
```
OMShell - OpenModelica Shell
File Edit Help
{0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,287.8787878787879,-287.8787878787879,-30.3030303030303}}

>> (eval,vec) := 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



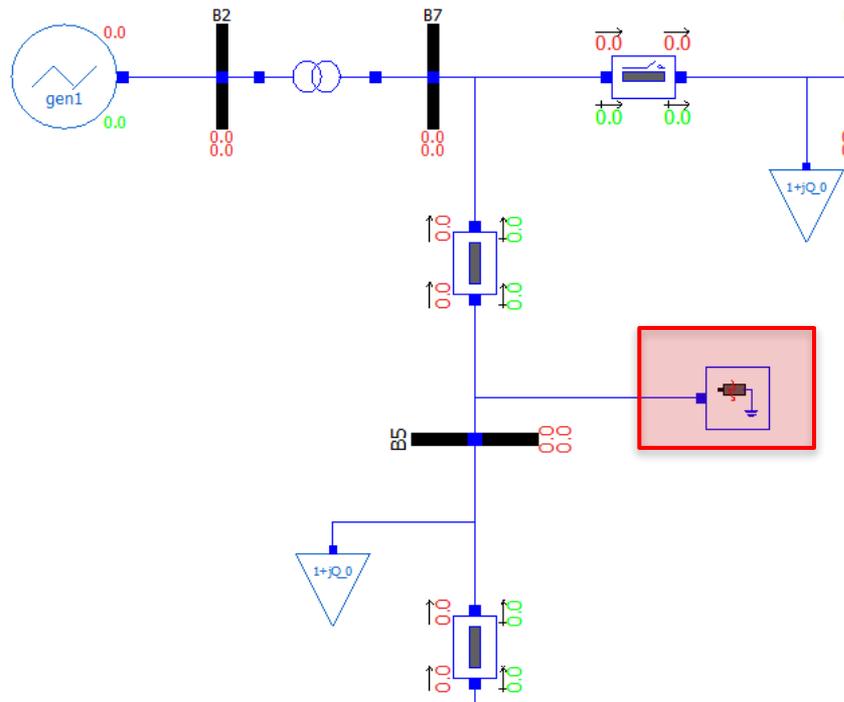


## Example 3

- Example 3 contains the model of the IEEE 9 Bus system
- 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

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

OMEdit - Component Parameters - gen1 in iPSL.Examples.Exa... X

### Parameters

General Modifiers

Component

Name: gen1  
Path: iPSL.Examples.Example\_3.Generation\_Groups.Gen1

AVR Disturbance

height\_1 0.05  
tstart\_1 2  
refdisturb\_1 true

Power flow data

|         |                   |                              |
|---------|-------------------|------------------------------|
| V_b     | 18                | Base voltage of the bus (kV) |
| V_0     | 1.025             | Voltage magnitude (pu)       |
| angle_0 | 0.160490018910725 | Voltage angle (deg)          |
| P_0     | 1.63              | Active power (MW)            |
| Q_0     | 0.001552891584958 | Reactive power (MVar)        |
| S_b     | SysData.S_b       | System base power (MVA)      |
| fn      | SysData.fn        | System Frequency (Hz)        |

OK Cancel



# Questions?

Thanks to all current and former OpenIPSL Developers @ KTH



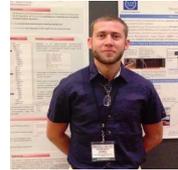
Luigi Vanfretti



Achour  
Amazouz



Mohammed  
Ahsan Adib Murad



Francisco José  
Gómez



Giuseppe  
Laera



Tin Rabuzin



Jan Lavenius



Le Qi



Maxime  
Baudette



Mengjia Zhang



Tetiana  
Bogodorova



Joan Russiñol  
Mussons

Join us!