OpenIPSL

A Modelica Library for Power Systems Simulation

Assoc. Prof. **Luigi Vanfretti**

luigiv@kth.se,

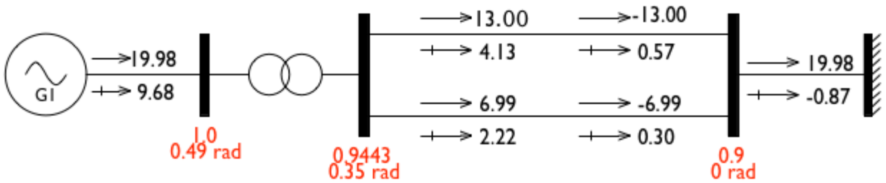
https://www.kth.se/profile/luigiv/

Hands-on Examples!

Please follow these slides to carry out the examples.

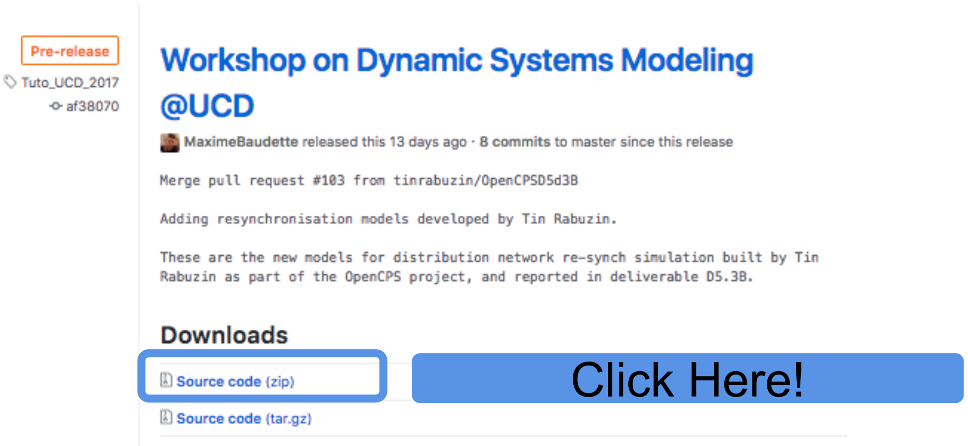
Workshop Agenda

- 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_UCD_2017



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.

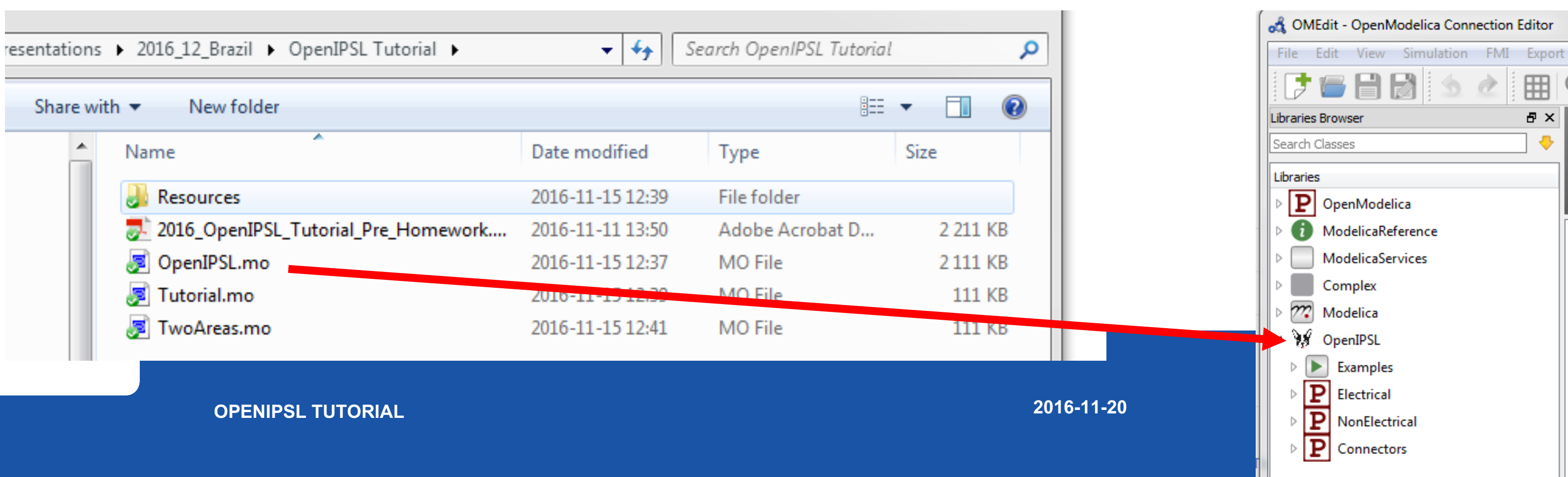
The dedicated package will also be available on a USB stick that we can circulate on the day of the tutorial.

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.

Note: In OM 1.11 beta, drag & drop does not work, use **File/Open**

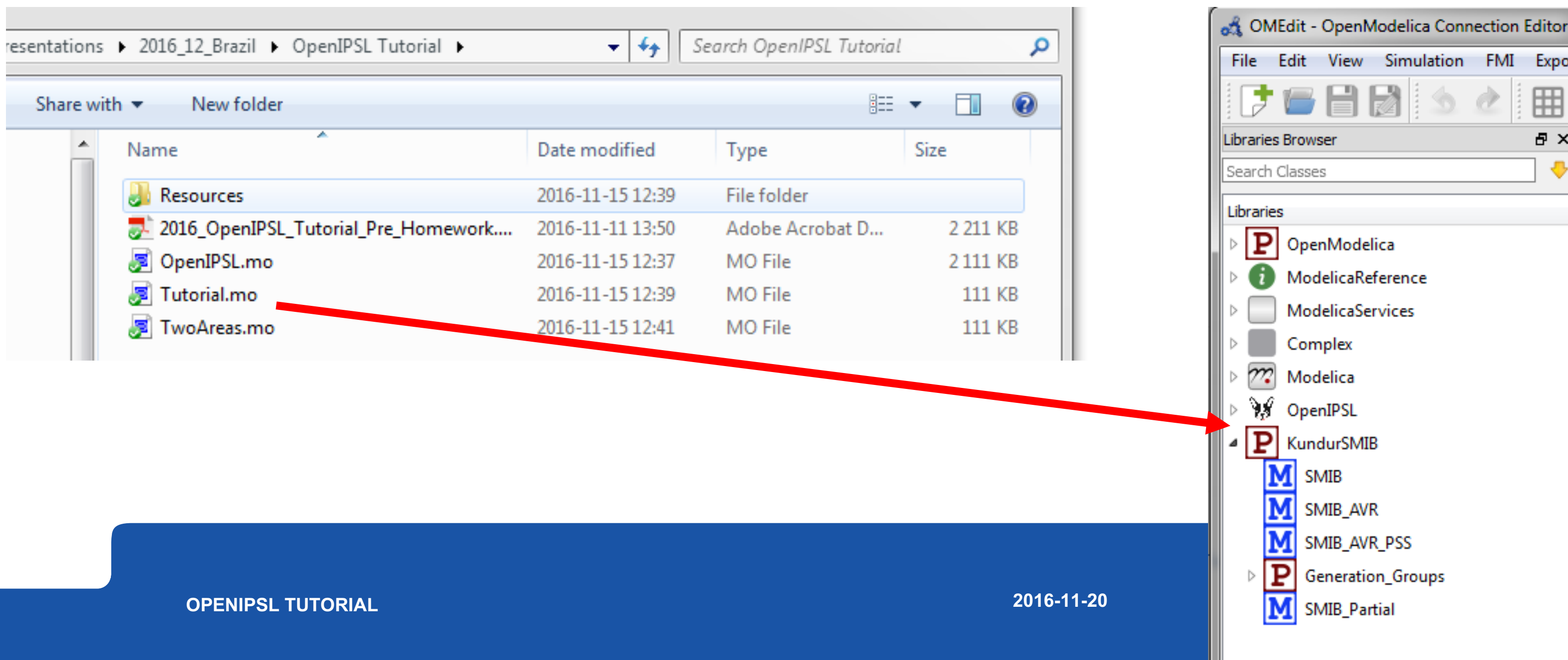


Load an Application Example to OMEdit

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

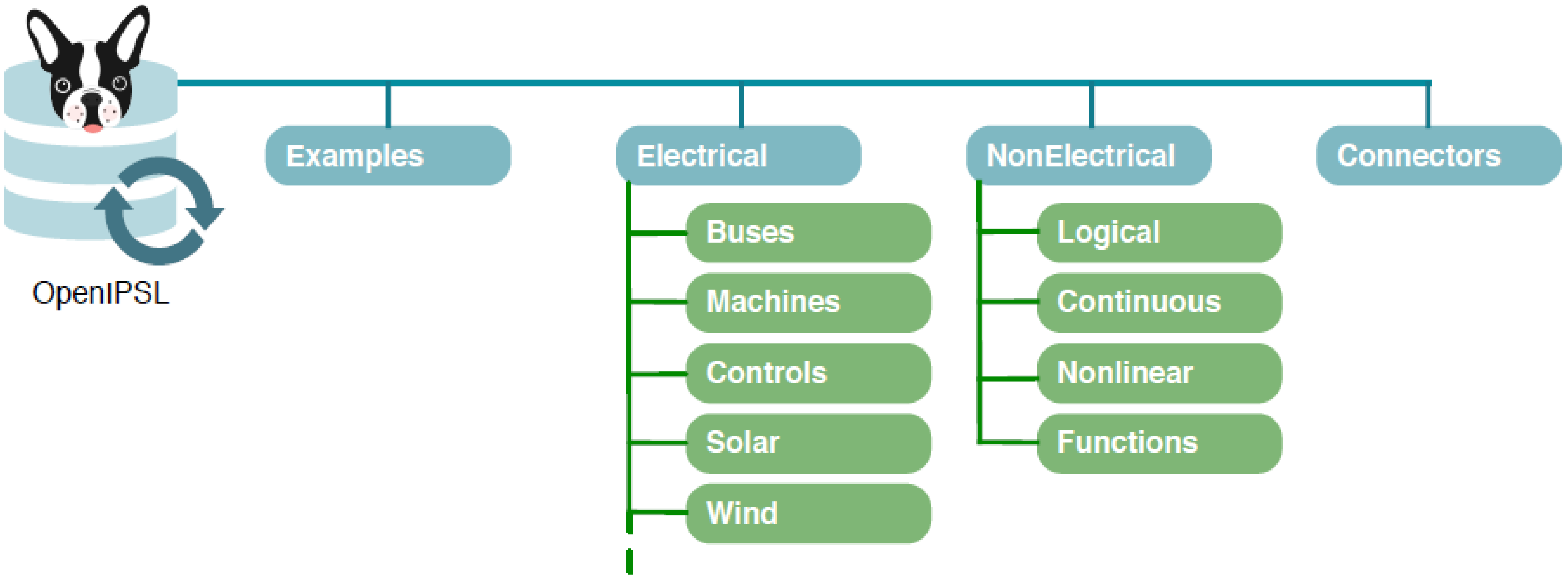
- Browse Windows Explorer to the location of the unzipped folder
- Drag & drop the **Tutorial.mo** file to the **Library Browser** in OMEdit.

Note: In OM 1.11 beta, drag & drop does not work, use **File/Open**



Library introduction

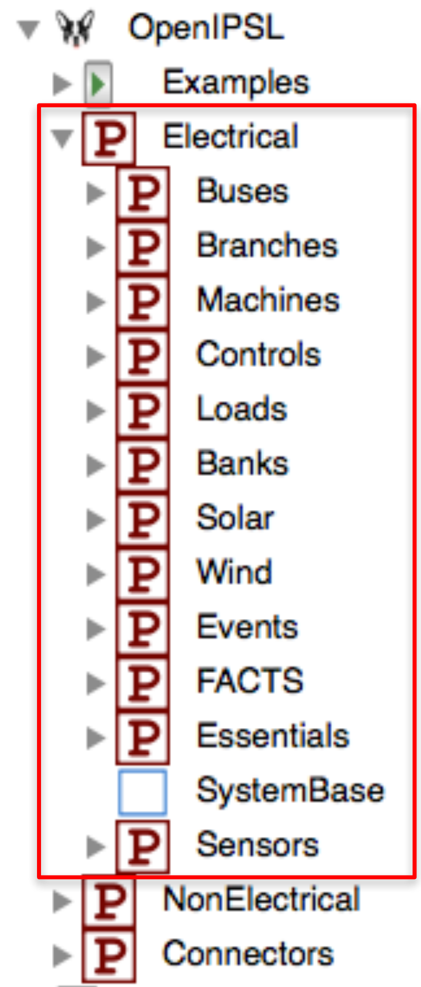
OpenIPSL is divided in four main categories:



Library introduction

Electrical

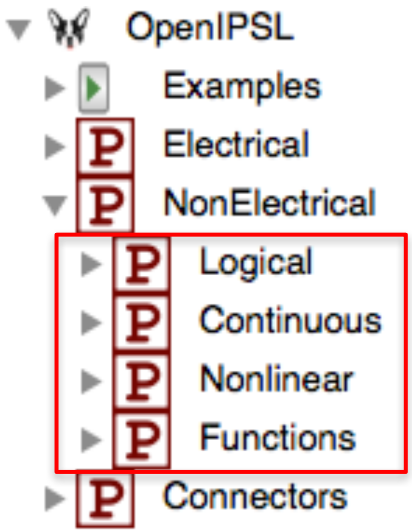
- The *Electrical* package contains most of the components that comprise an actual power network
- E.g., electrical machines, transmission lines, loads, excitation systems, turbine governors, etc.
- These are used to build the power system network models



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)



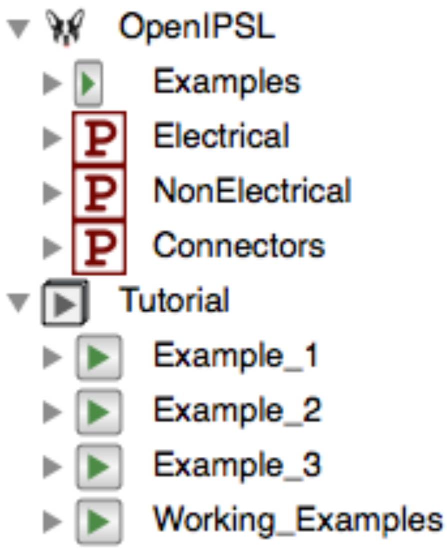
Connectors

- 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)

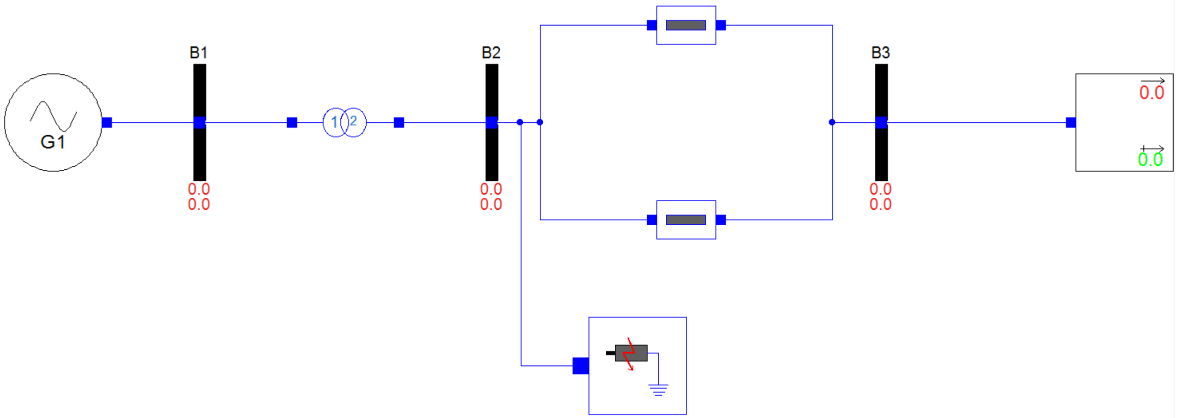
Library introduction

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*



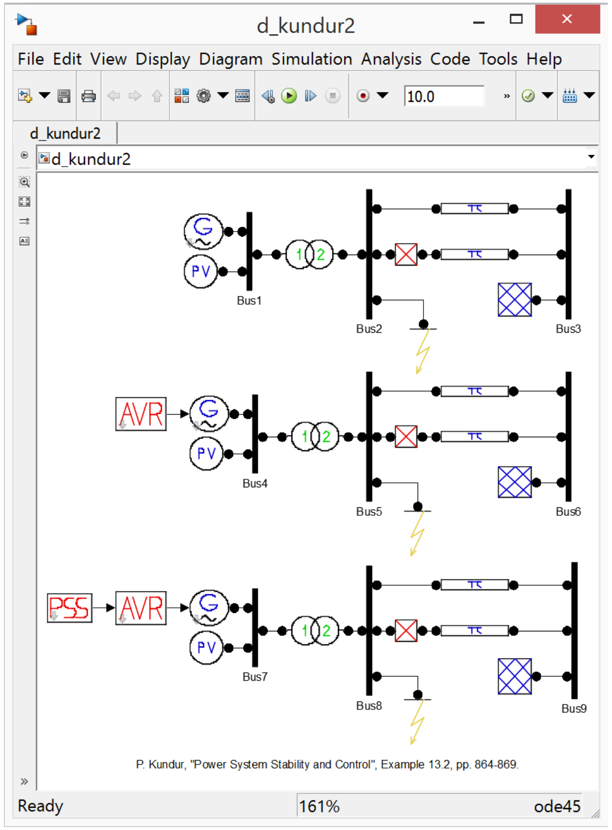
- Single Machine Infinite Bus (SMIB) system
- Analysis of the transient stability of the system including the effects of rotor circuit dynamics and excitation control
- Four machines represented by one connected via transformer and parallel lines to the infinite bus

* P. Kundur, “Power System Stability and Control”, Example 13.2

Example 1

Power flow

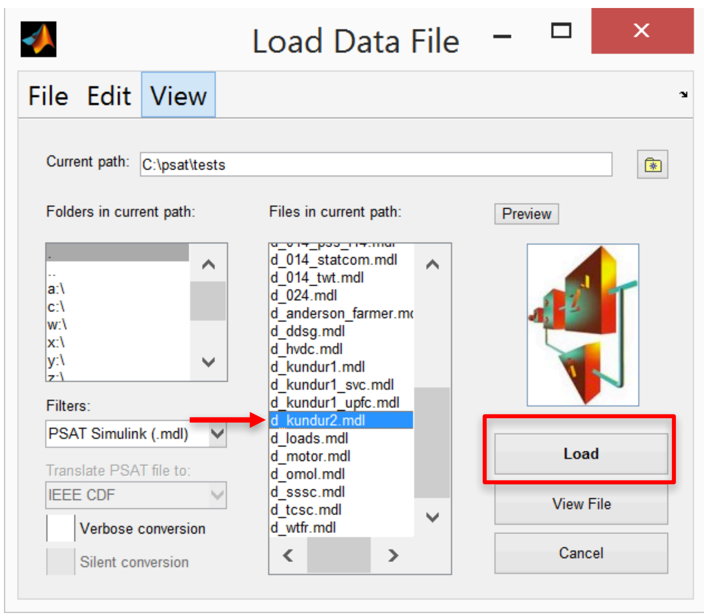
- Power flow results were obtained by PSAT
- Prepared Example 1 already exists in PSAT and can be used for power flow calculations and dynamic simulations

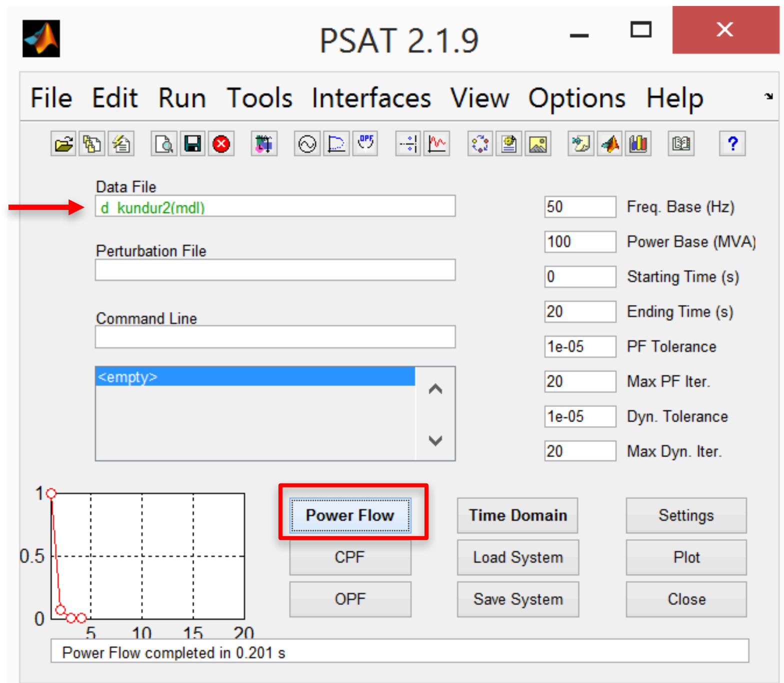


Example 1

Power flow

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

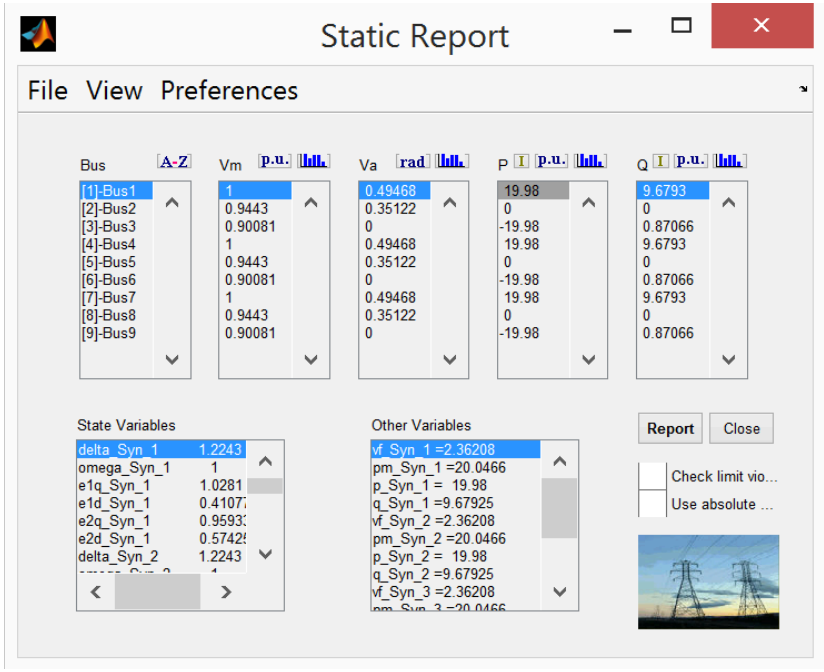




Example 1

Power flow

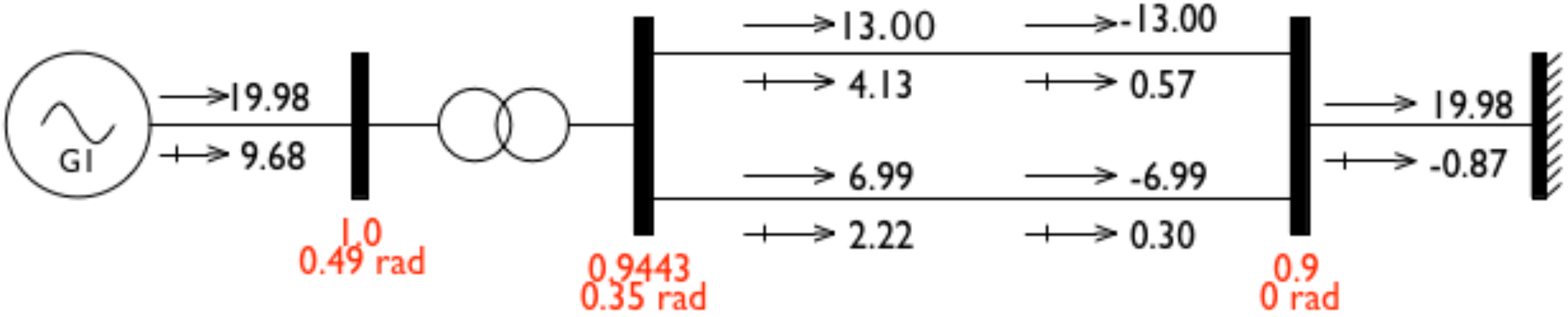
- Static Report can be access where all of the power flow results are listed along with the initial values of various state variables of the models
- In this tutorial, there is no need to run the power flow in PSAT since the data will be provided, but feel free to explore PSAT later



Example 1

Power flow

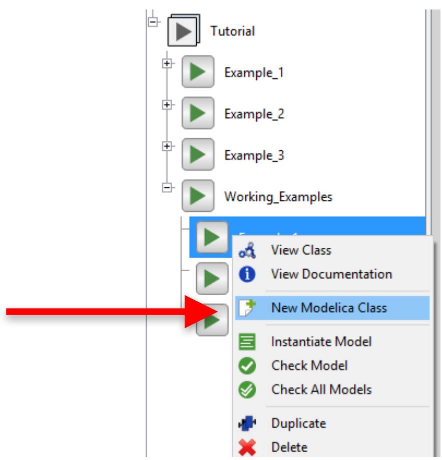
- The summary of all of the relevant data from the power flow is given on the figure below

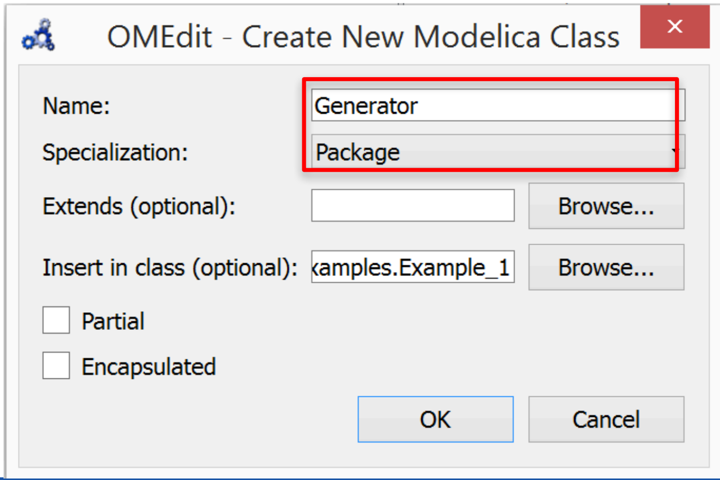


Example 1

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*

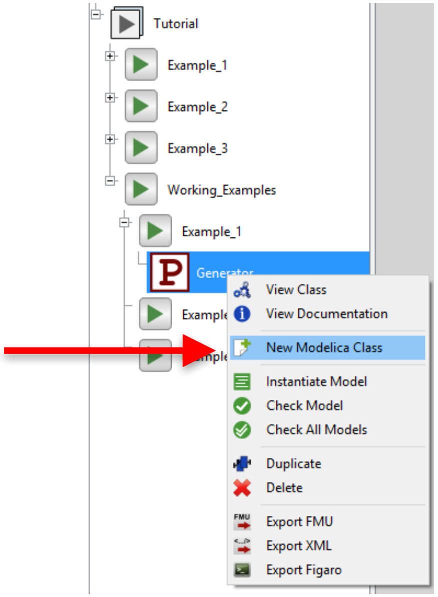


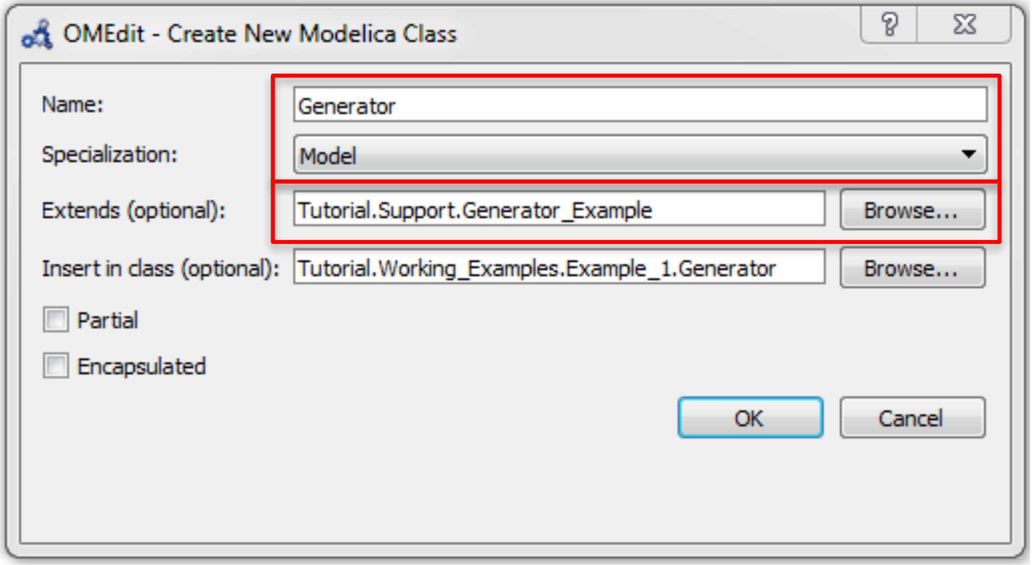


Example 1

Generator model – Step 1

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

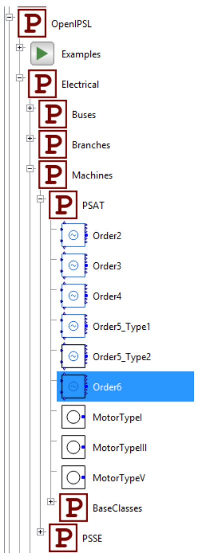




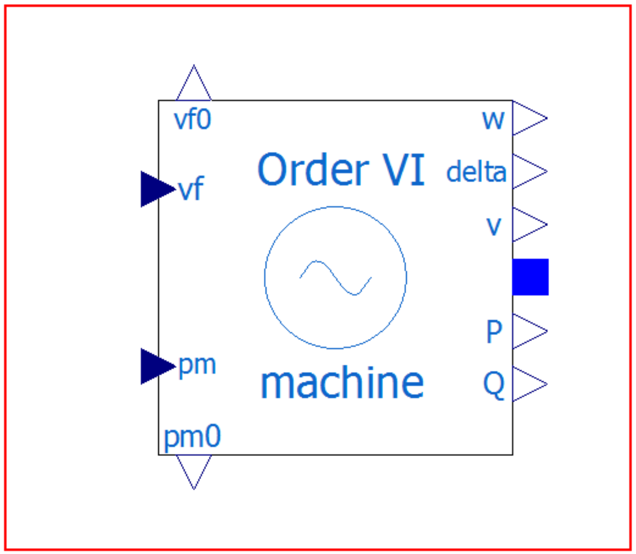
Example 1

Generator model – Step 1

- 6^{th} 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.Order6



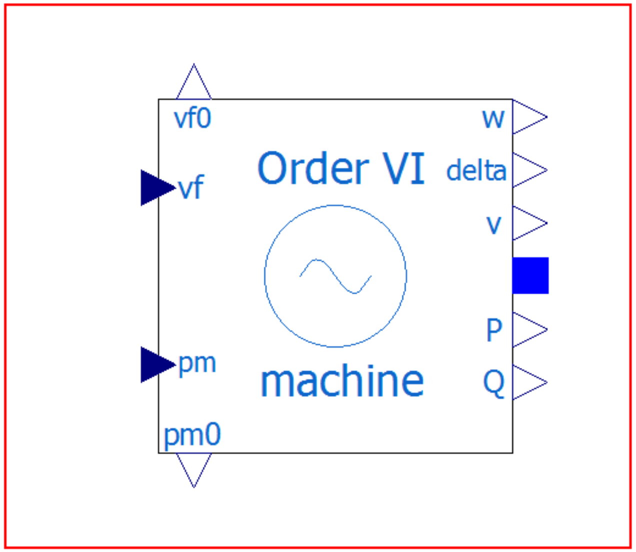
Example 1

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.PSATOrder6



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

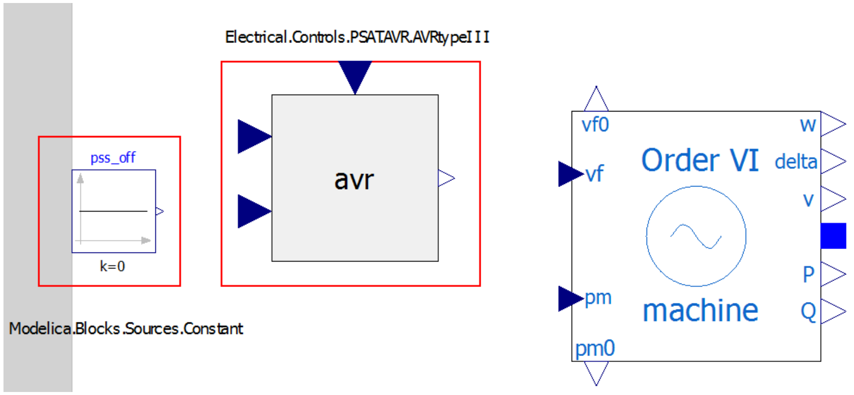
Note: Using the variables (V_0, angle_0, etc.) allow to propagate the parameters to the “upper layer” of the generator component

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

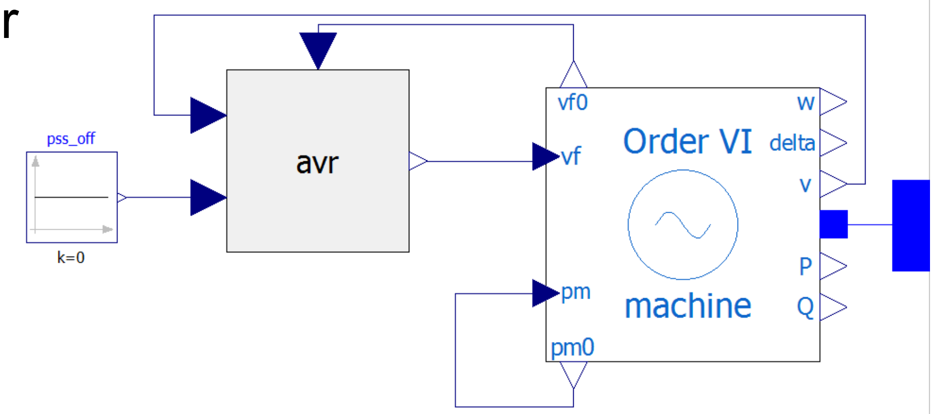


Example 1

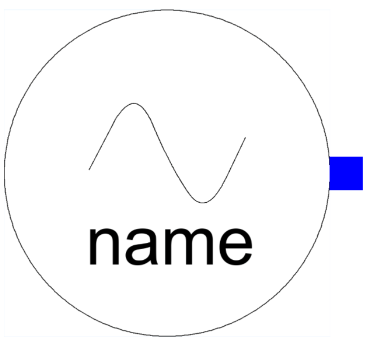
Generator model – Step 3

- To finish the generator model, different signals need to be connected

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



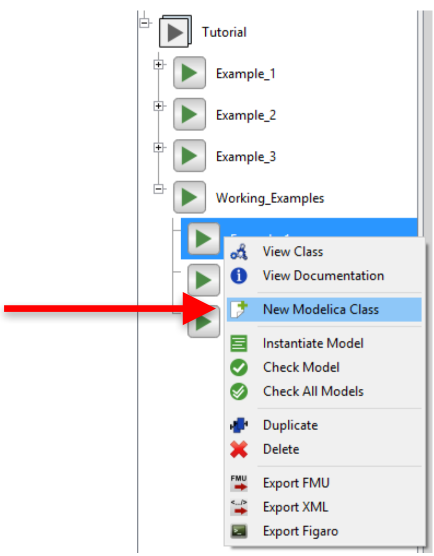
- Optionally, icon of the generator model can be altered

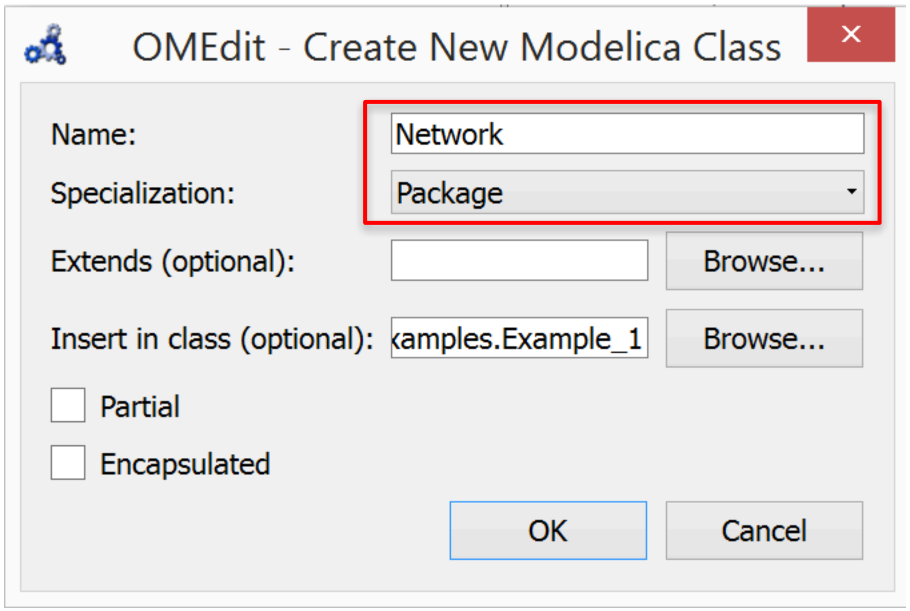


Example 1

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

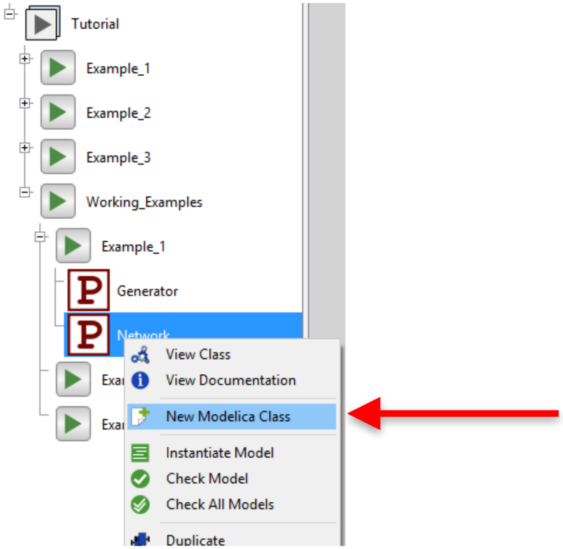


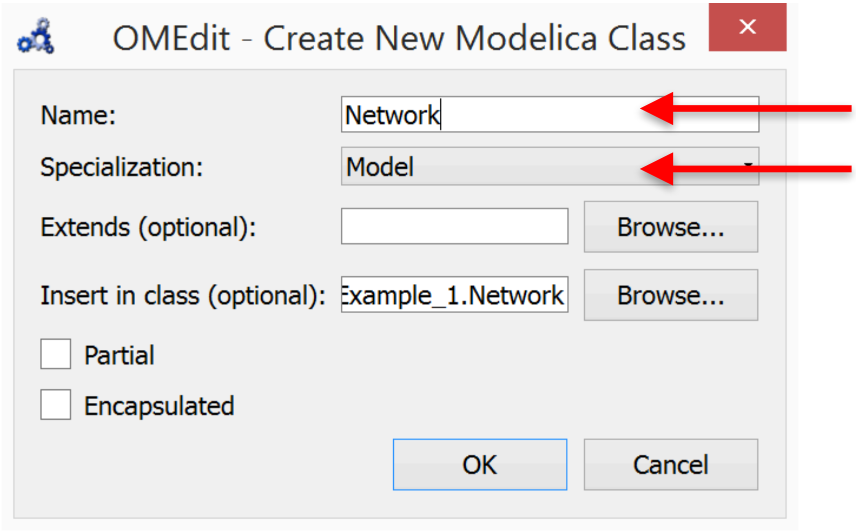


Example 1

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*



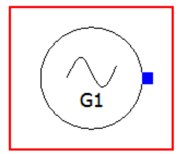


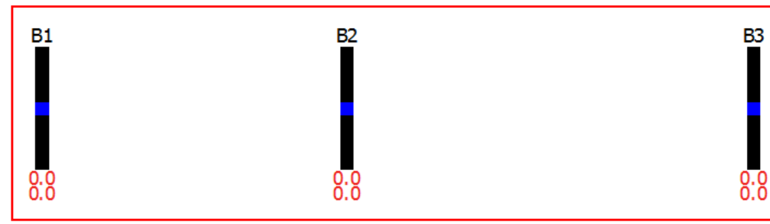
Example 1

Network model – Step 1

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

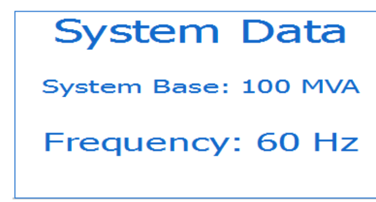
Electrical.Buses.Bus





- 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



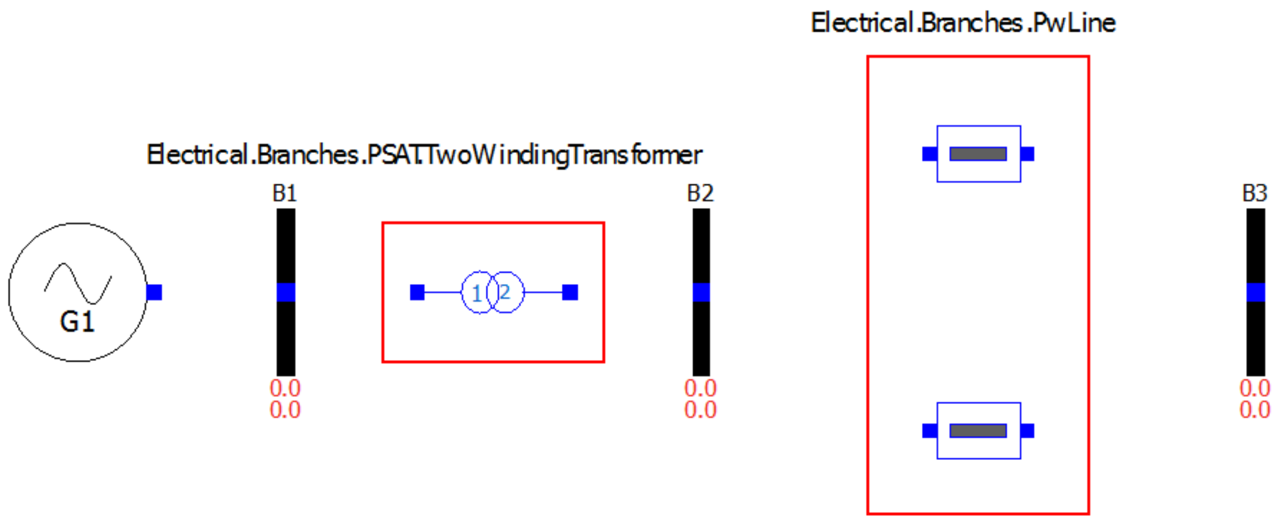
- In *text view* add the **inner** keyword in front of the component declaration

```
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

Line 1

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

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 2

R	0.0	G	0.0
X	0.93*100/222 0	B	0.0
S_b	100		

Example 1

Network model – Step 3

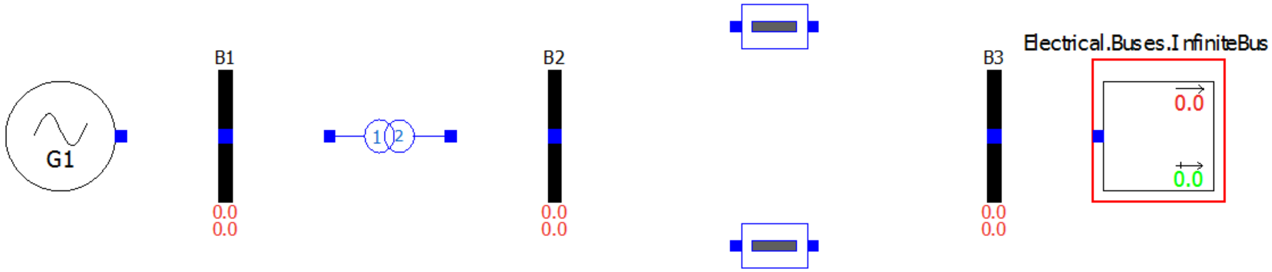
- Infinite bus is added
- Power Flow results are implemented

G1

V	1	$angle$	0.4946
P	1997.999	Q	967.92

Infinite bus

V	0.90081	$angle$	0
P	-1998	Q	87.066



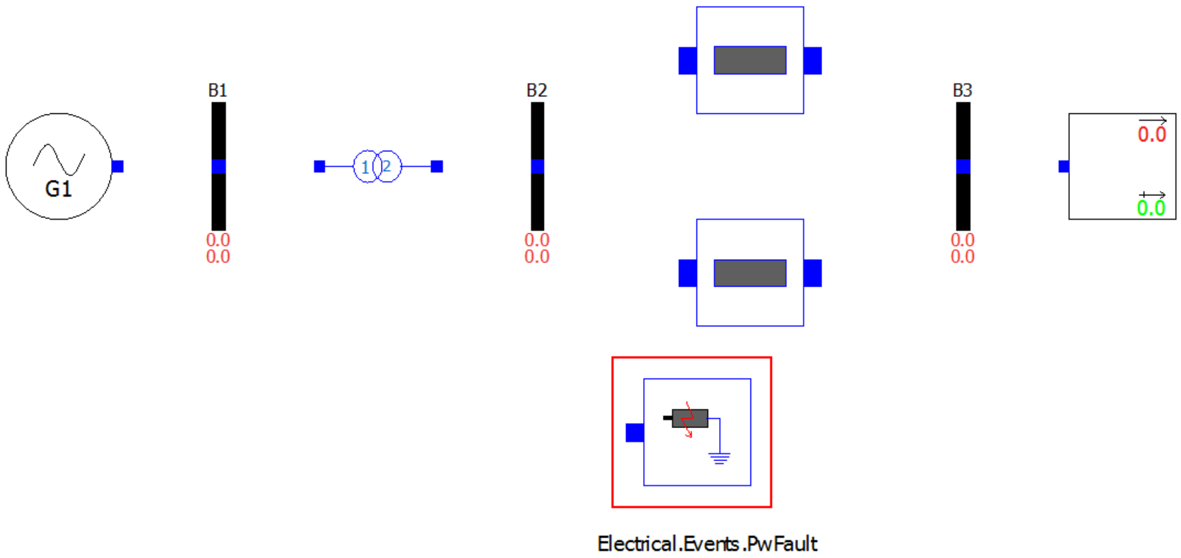
Example 1

Network model – Step 4

- 3-phase-to-ground fault is added

Fault

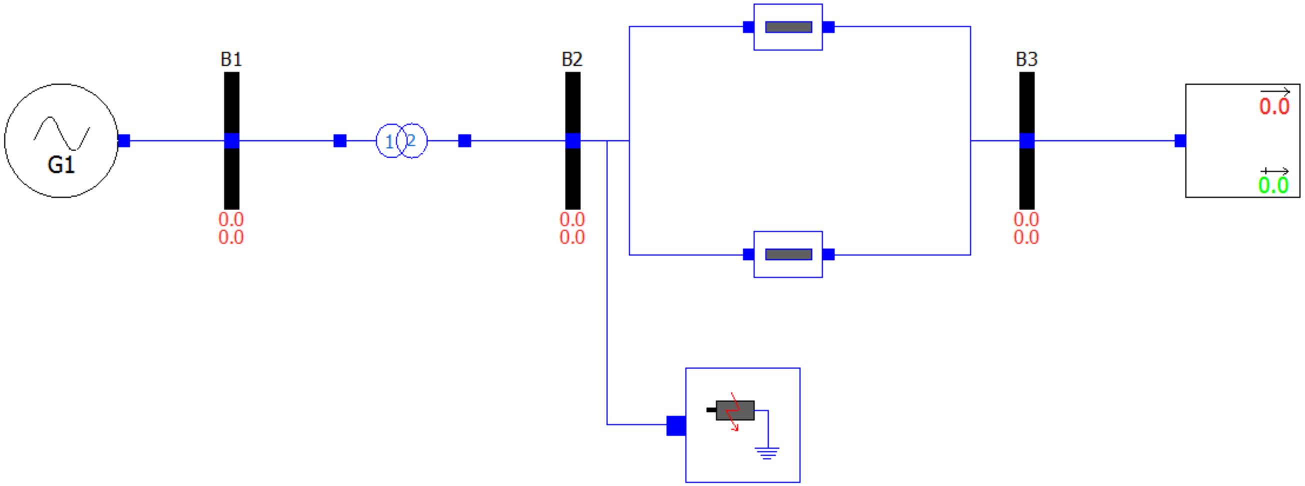
R	0	t_1	0.5
X	0.01*100/2220	t_2	0.57



Example 1

Network model – Step 5

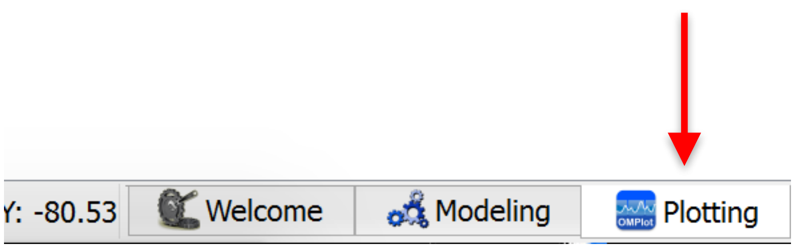
- The network model is completed by connecting all of the components
- Now, the model can be simulated and linearized



Example 1

Simulation

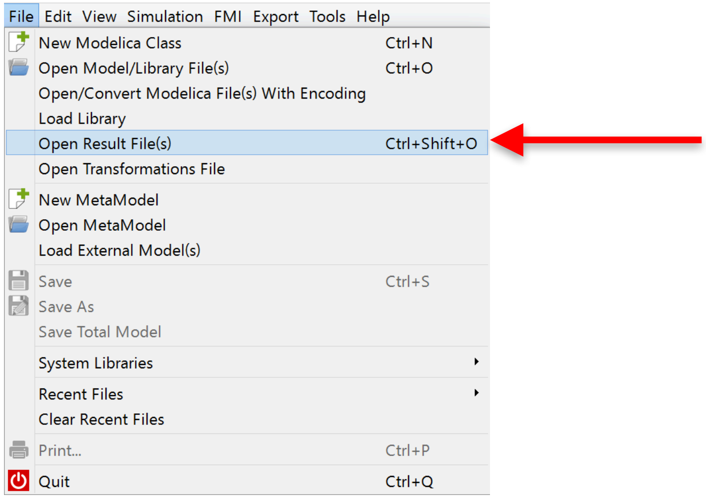
- System will be simulated with 3-phase-to-ground fault at t=0.5s with a duration of 70ms
- Simulation results will be compared with the reference results from the PSAT that will be loaded first
- PSAT results are provided in a file “PSAT_dyn.csv”
- To load the file, the view should be switched to “Plotting”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

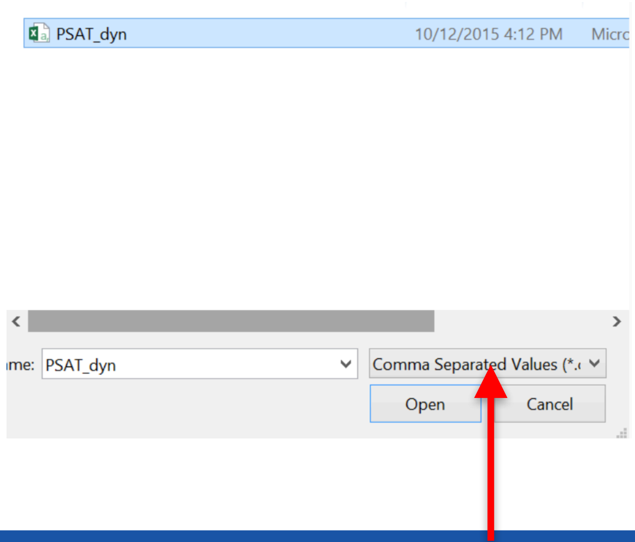


Example 1

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

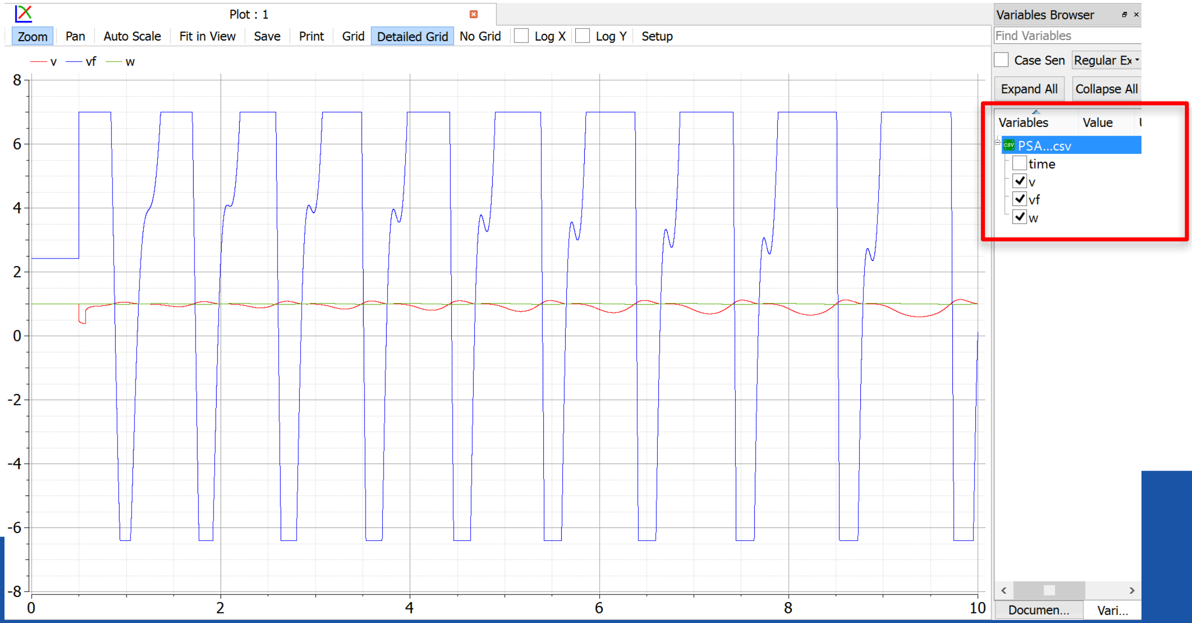




Example 1

Simulation

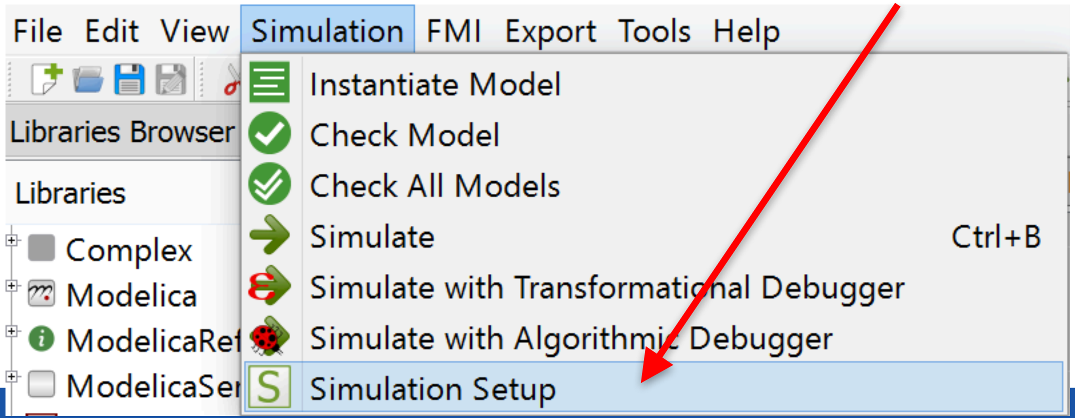
- In the variable browser, three waveforms from the PSAT results are loaded which can be displayed on the plot as it is shown in the figure below
- Loaded waveforms are generator terminal voltage, excitation field voltage and the generator speed



Example 1

Simulation

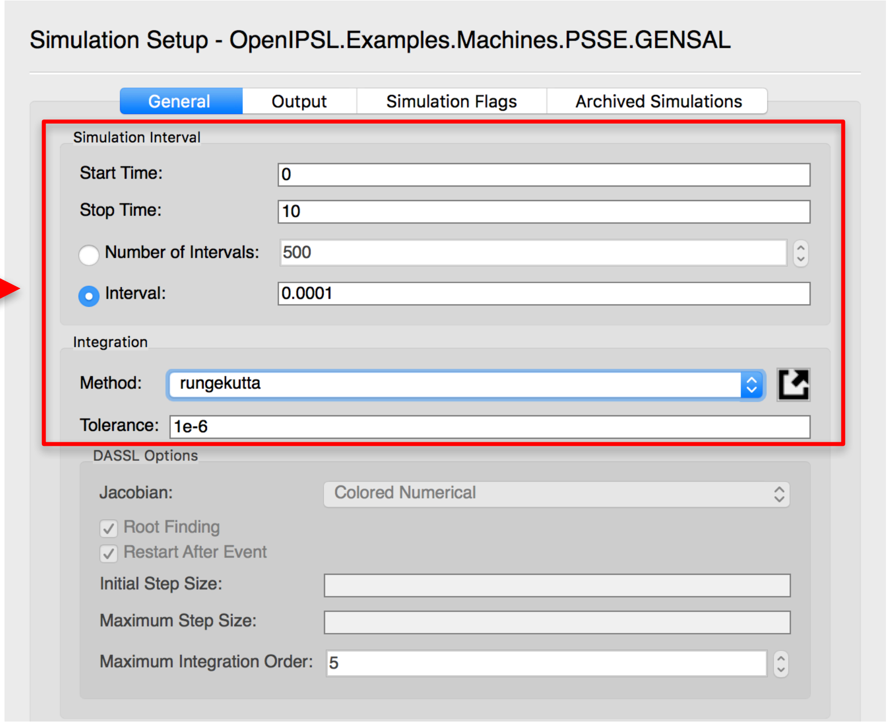
- Before the simulation, solver and its parameters are set to be the same as in the PSAT
- Solver is chosen to be Runge-Kutta with a fixed step
- More solvers can be chosen in Modelica (depending on the tool), however, to match the model's response with the one in PSAT choice of the solver is limited



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

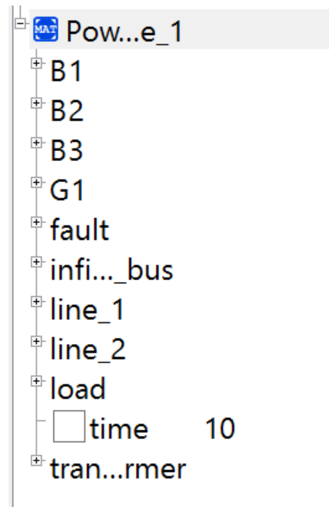


Example 1

Simulation

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

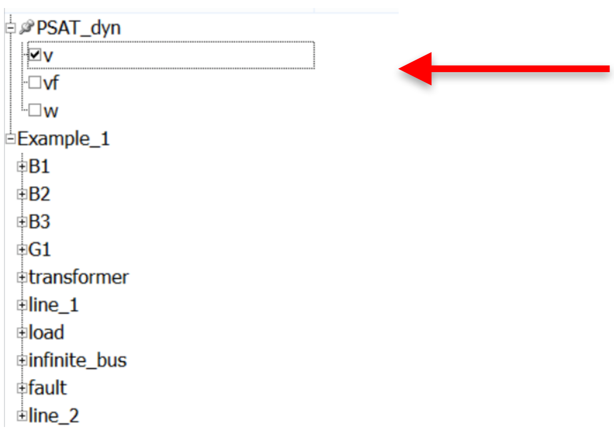
- Once the simulation is completed, the Variable Browser is populated with the simulation results

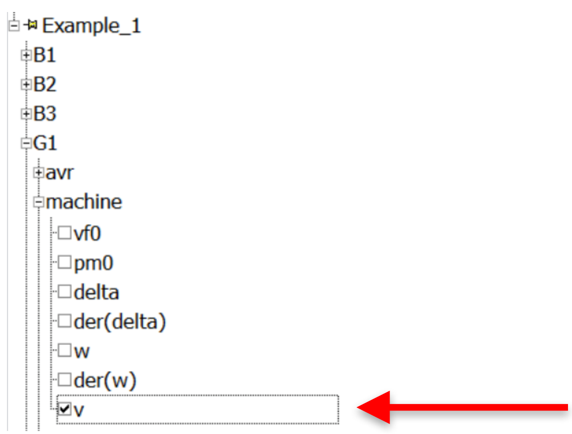


Example 1

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

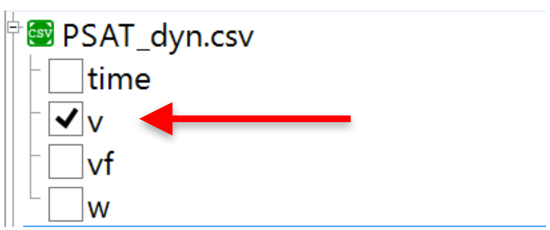


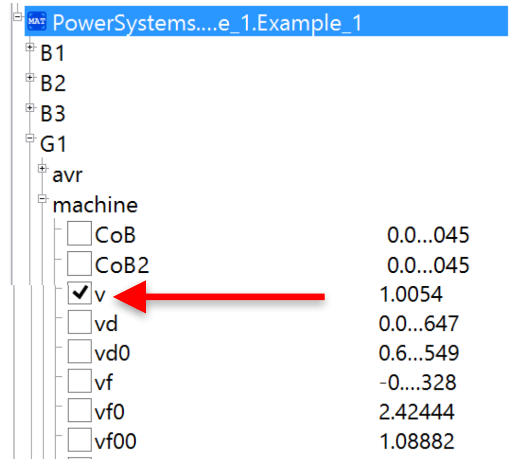


Example 1

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

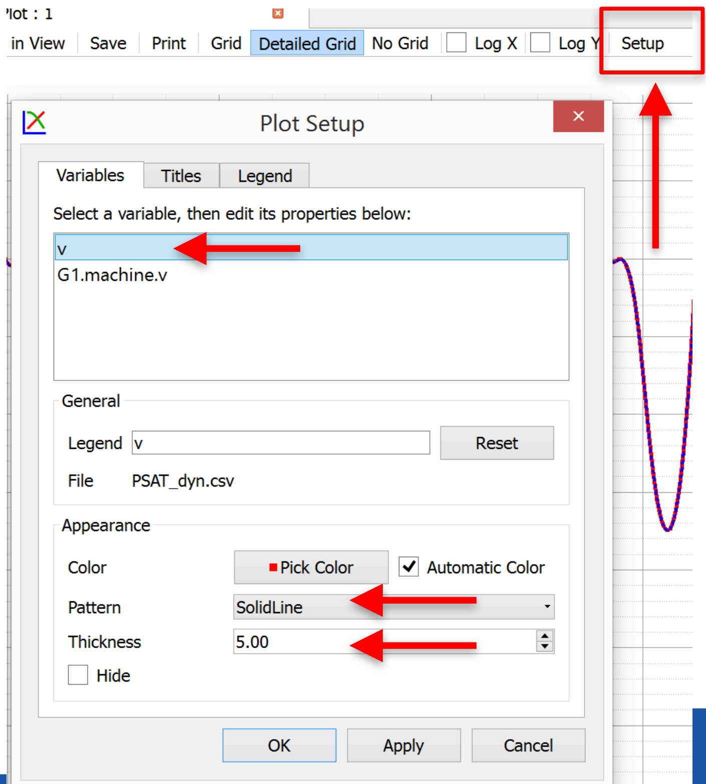


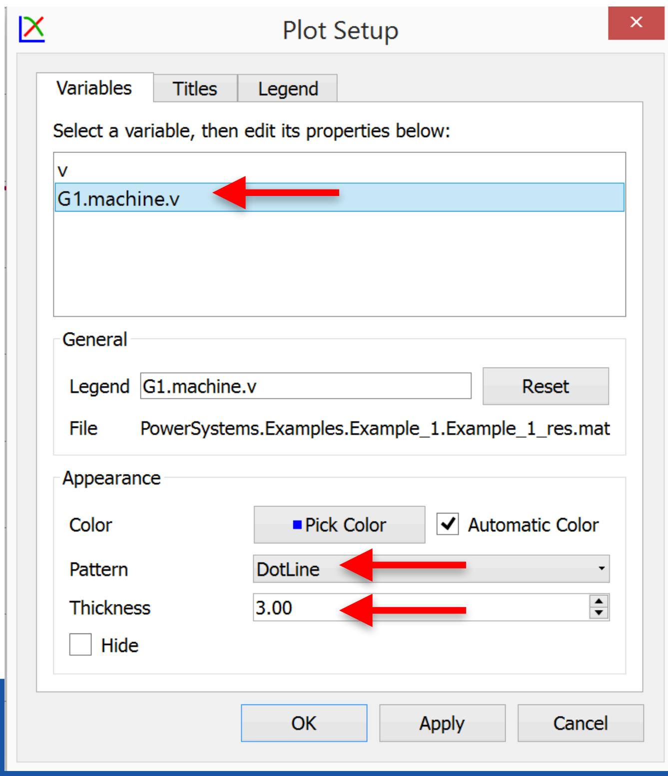


Example 1

Simulation

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

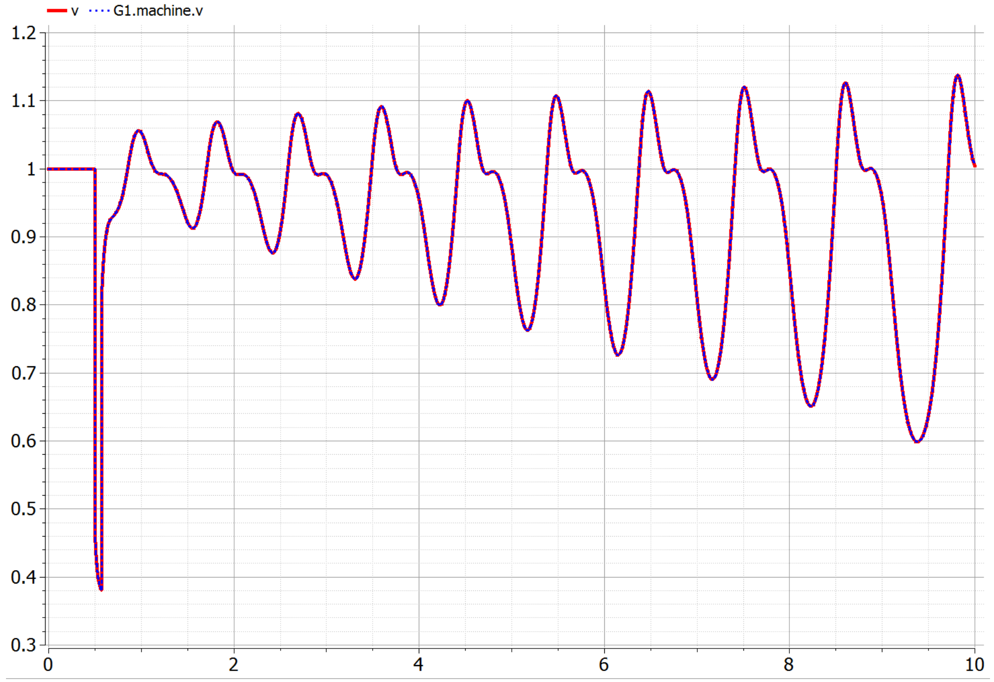




Example 1

Simulation

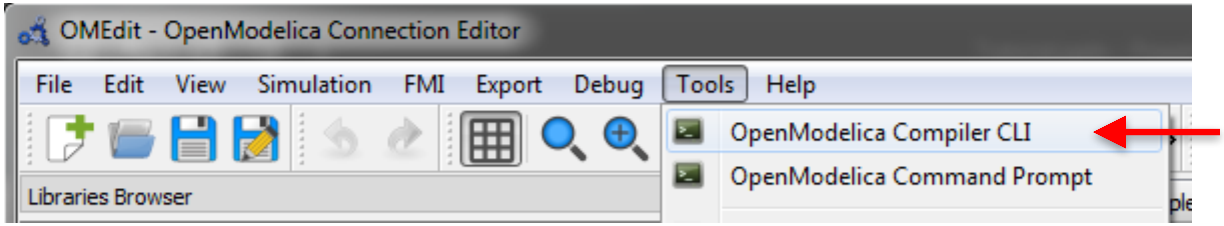
- Previous steps produce the plot shown in the figure below showing that the Modelica produces the same simulation results as the PSAT does



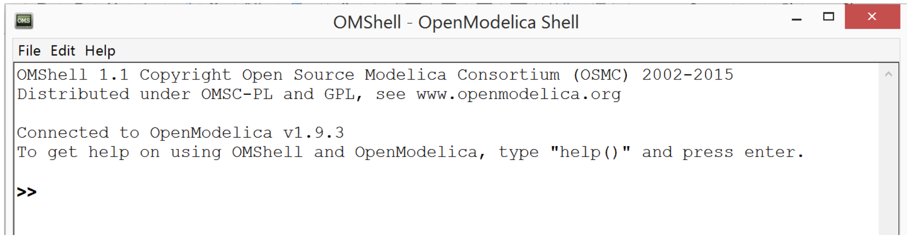
Example 1

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



Example 1

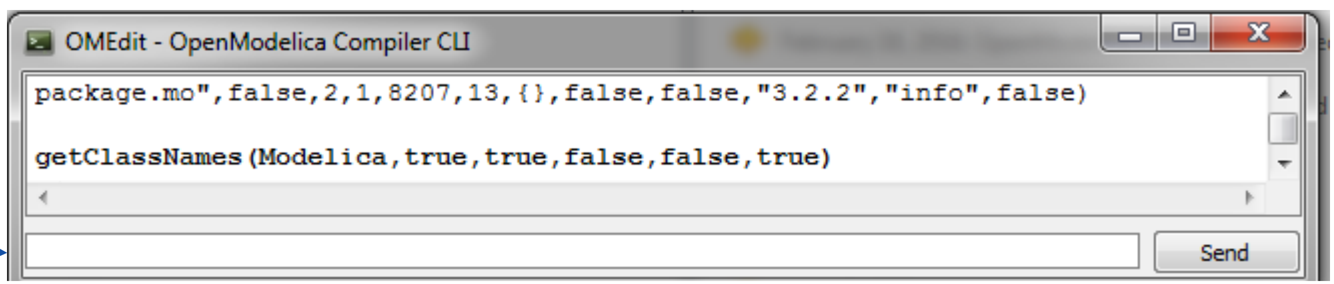
Linearization

- 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,evec) := Modelica.Math.Matrices.eigenValues(A);
```



Example 1

Linearization

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

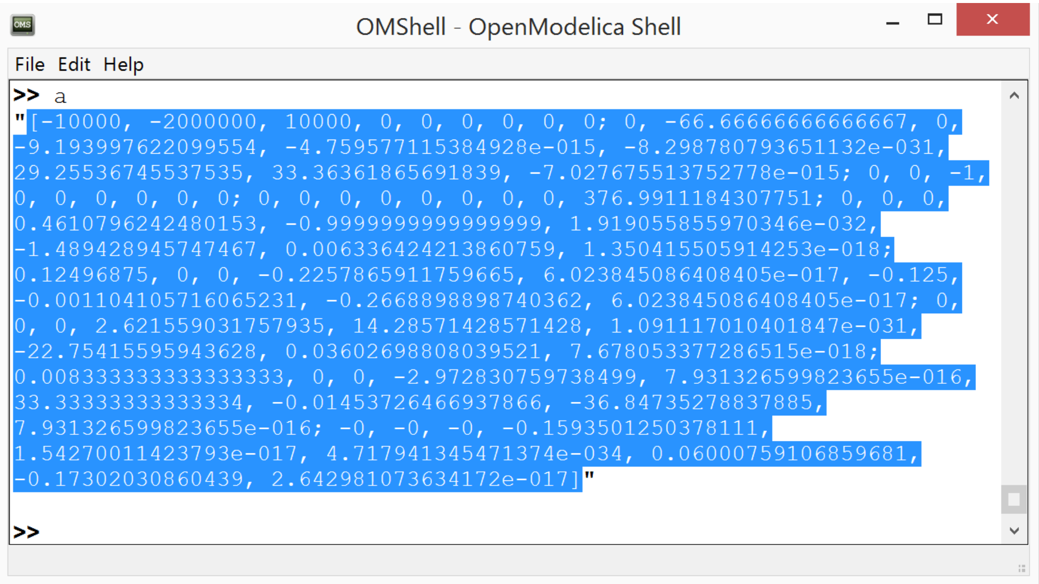
```
(a) := getParameterValue(linear_Tutorial_Example__1_Example__1,"A")
"[-10000.00001883345, -1999999.999376403, 10000.00001354202, 0, 0, 0, 0, 0,
0; 0, -66.66666664588011, 0, -9.19399840693, 0, 0, 29.25536833302091,
33.36361960418314, 0; 0, -0, -1, 0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0,
376.9911183132299; 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]"
```

Send

Example 1

Linearization

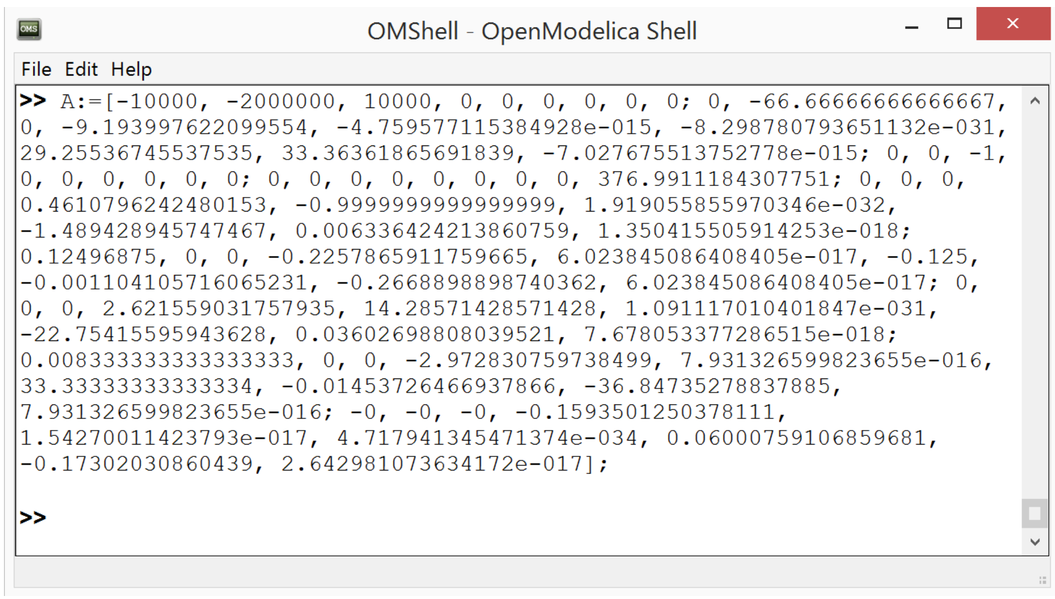
- Copy the output from the previous command without the quotation marks by pressing Ctrl+C



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



Example 1

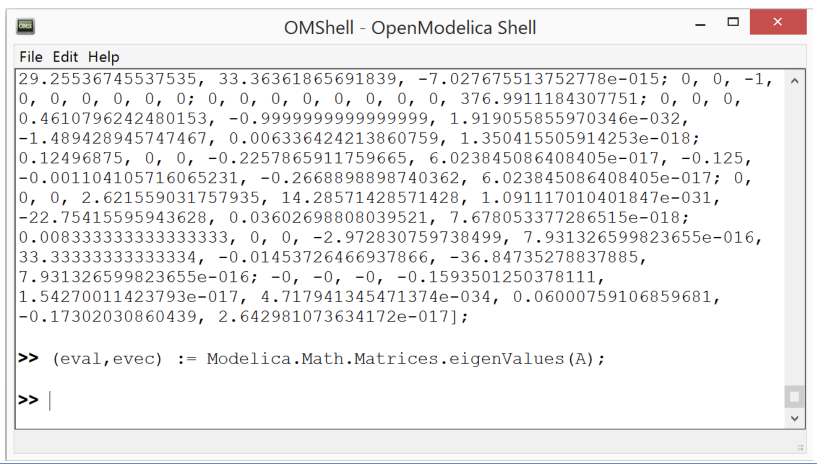
Linearization

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

$$det(\boldsymbol{A} - \lambda \boldsymbol{I}) = \boldsymbol{0}$$

- This can be done by executing the last command

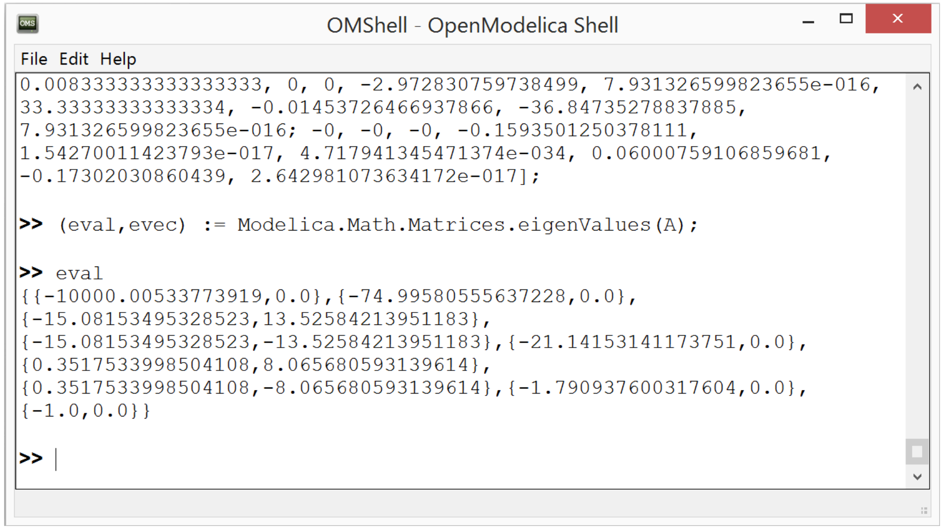
```
(eval,evec) := Modelica.Math.Matrices.eigenValues(A);
```



Example 1

Linearization

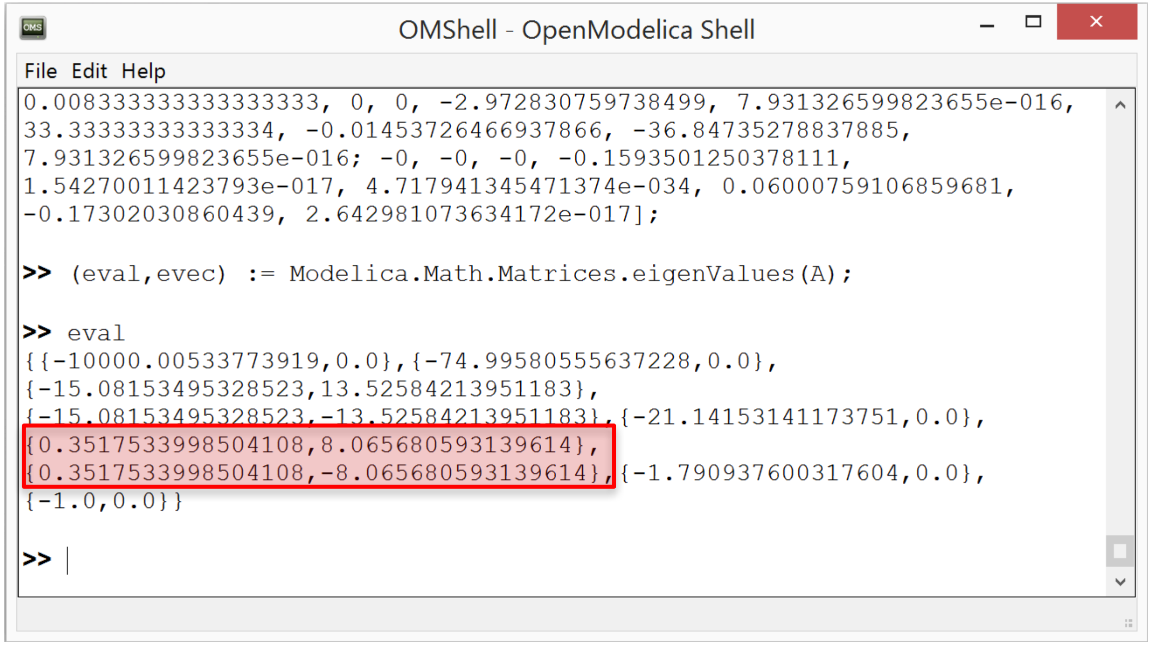
- The eigenvalues are now stored in the eval variable and they can be listed by executing `eval`
- Groups of numbers are listed where the first number is real part of the system's pole and the second one is the imaginary part



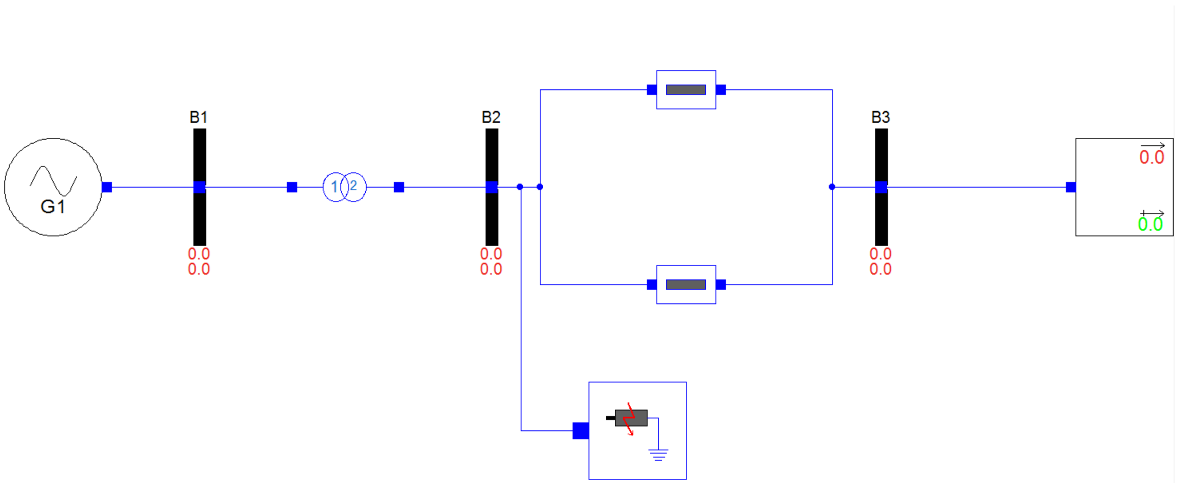
Example 1

Linearization

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



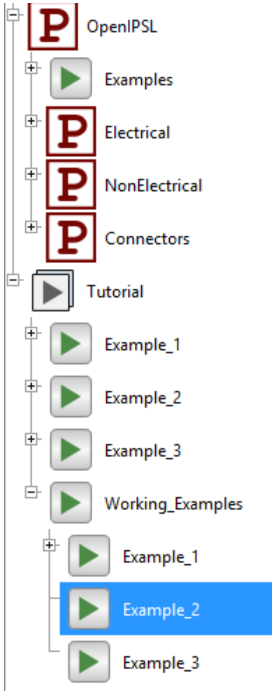
Example 2



- 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

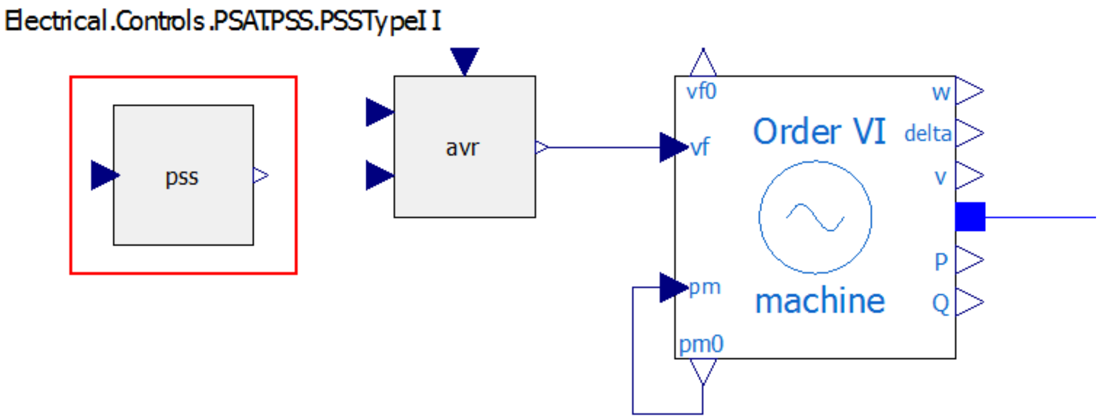
- The work on Example 2 should continue with the files prepared in a package Tutorial.Working_Examples.Example_2



Example 2

Generator model – Step 1

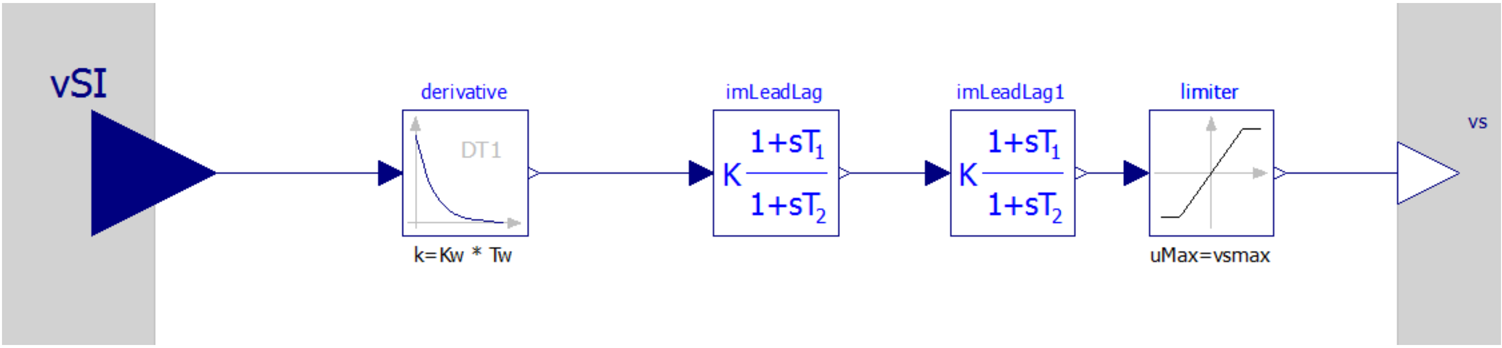
- The first step is to add the model of the PSS Type II and the summation block to the model of the generator



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”*



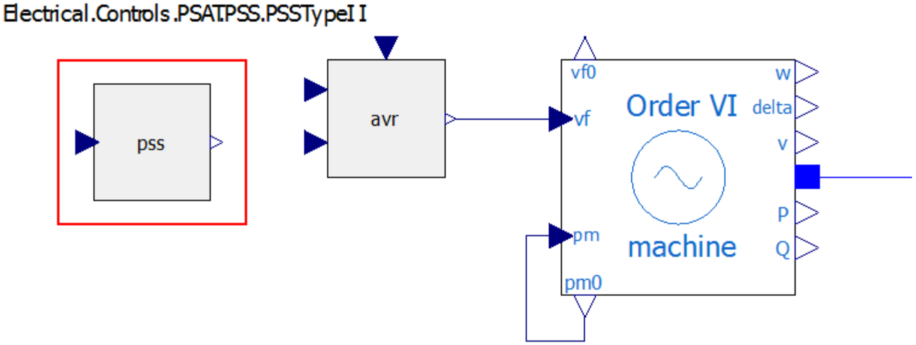
Example 2

Generator – Step 1

- PSS should be parameterized as shown in the table

PSS

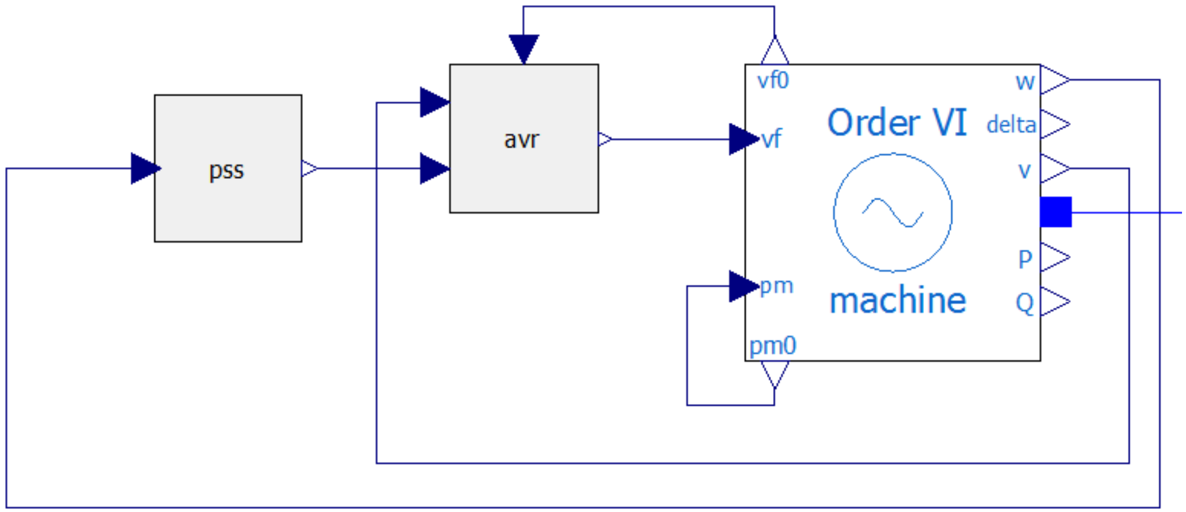
$v_{s,max}$	0.2	T_1	0.154
$v_{s,min}$	-0.2	T_2	0.033
K_w	1.41	T_3	1
T_w	0.001	T_4	1



Example 2

Generator – Step 2

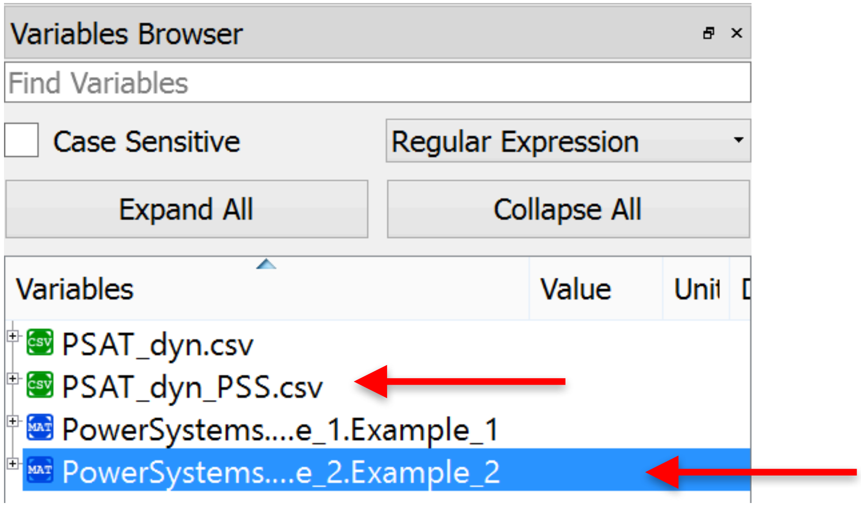
- When the signals of the generator model are connected as shown, model of the generator is completed



Example 2

Simulation

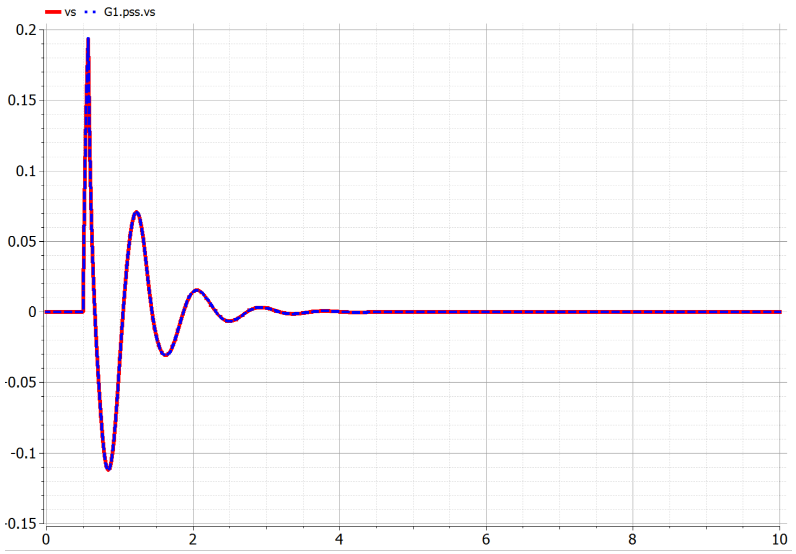
- Simulation steps can be repeated as it was shown in the Example 1
- This time, reference simulation results from the PSAT can be found in the file “PSAT_dyn_PSS.csv”
- After the simulation is executed, variable browser should look as it is shown below



Example 2

Simulation

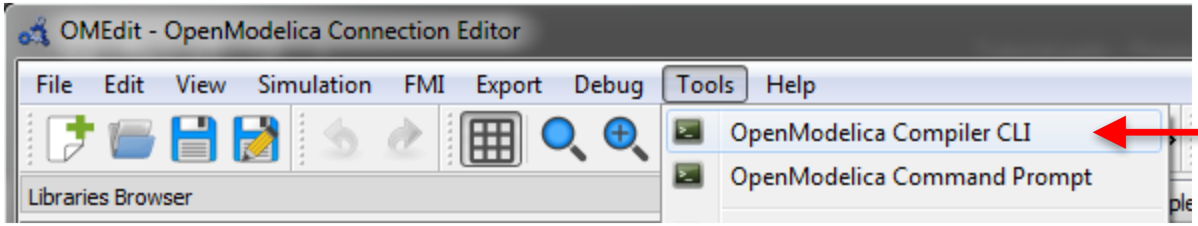
- Simulation results can be plotted again
- Comparison of the PSAT and Modelica simulation results of the PSS signal is shown on the figure below



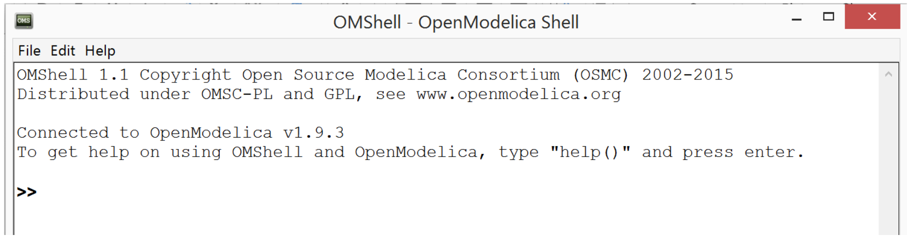
Example 2

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



Example 1

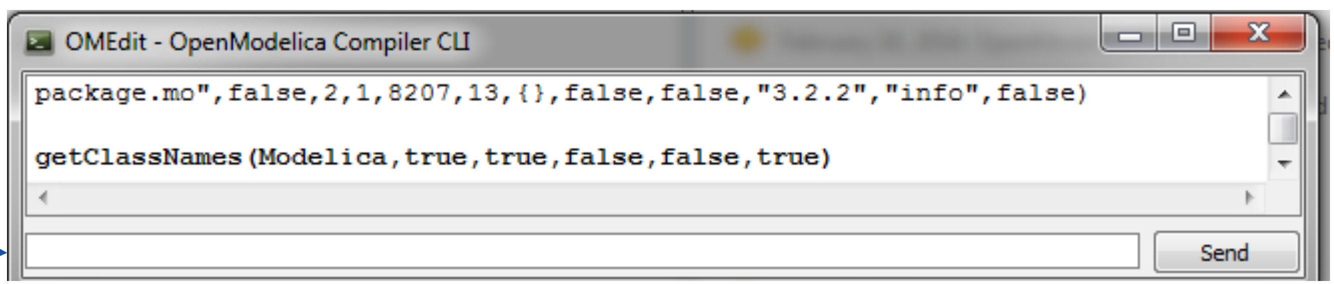
Linearization

- 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

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,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},
{-1.800161321638047,7.229416233496 77},
{-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

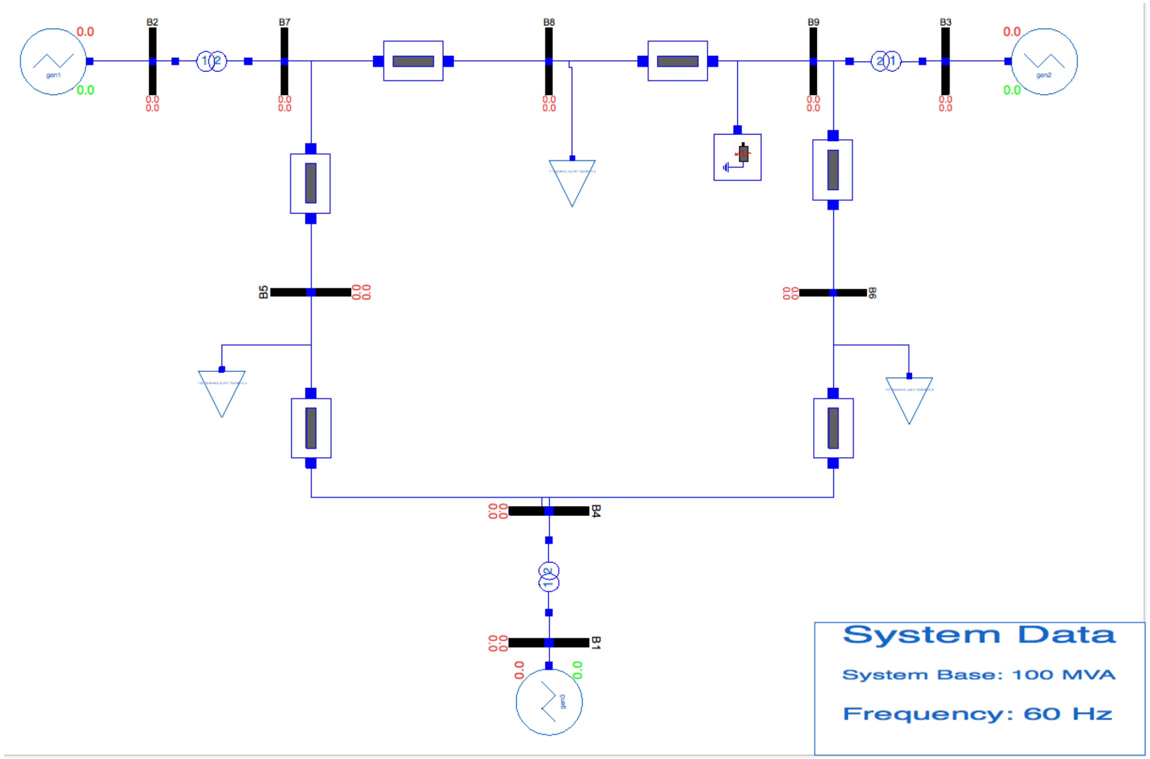
```
{0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,287.8787878787879,-287.8787878787879,-30.3030303030303}}

>> (eval,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

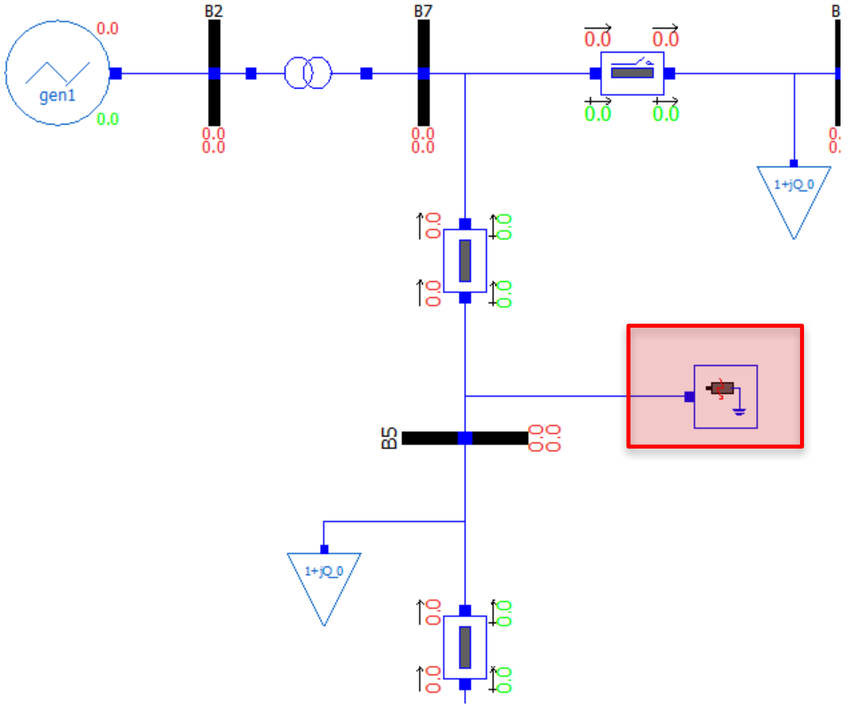

B2
B7
B8
B9
B3
B5
B6
B4
B1
System Data
System Base: 100 MVA
Frequency: 60 Hz

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

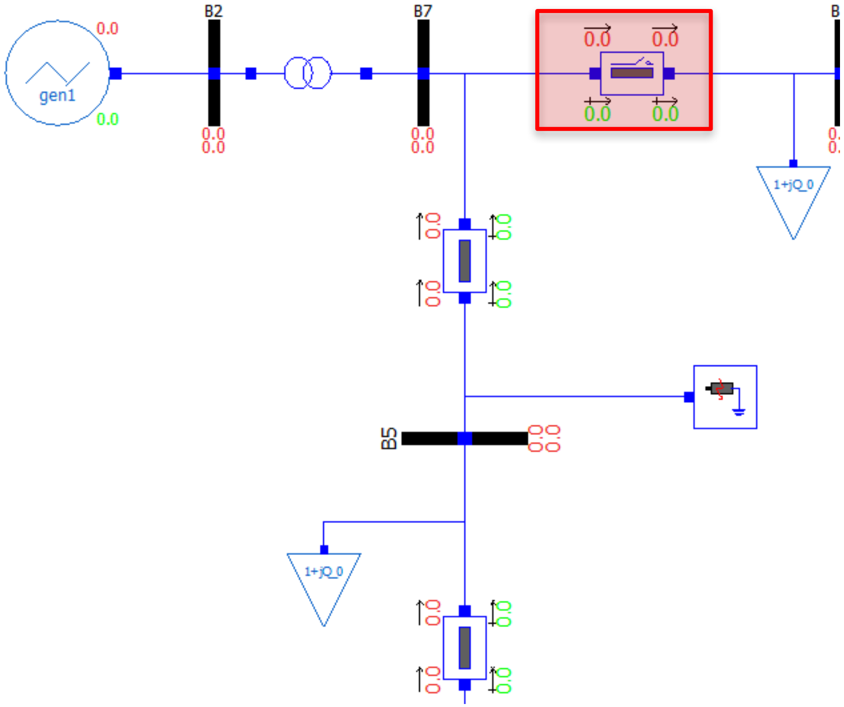
Example 3

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



Example 3

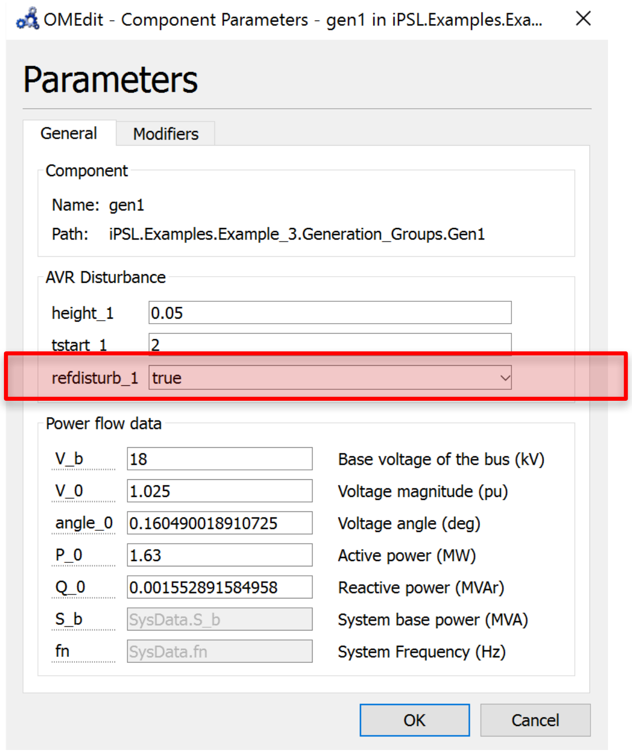
... or open the line at the given time instant*



*Model of the line with opening is OpenIPSL.Electrical.Branches.PwLine2Openings

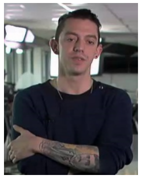
Example 3

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



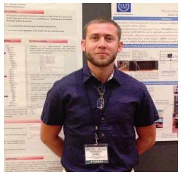
Questions?

Thanks to all current and former OpenIPSL Developers @ KTH


Luigi Vanfretti

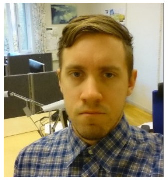

Achour Amazouz


Mohammed Ahsan Adib Murad


Francisco José Gómez

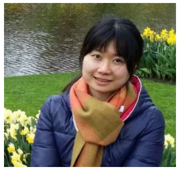

Giusseppe Laera


Tin Rabuzin


Jan Lavenius


Le Qi


Maxime Baudette


Mengjia Zhang


Tetiana Bogodorova


Joan Russiñol Mussons

Join us!

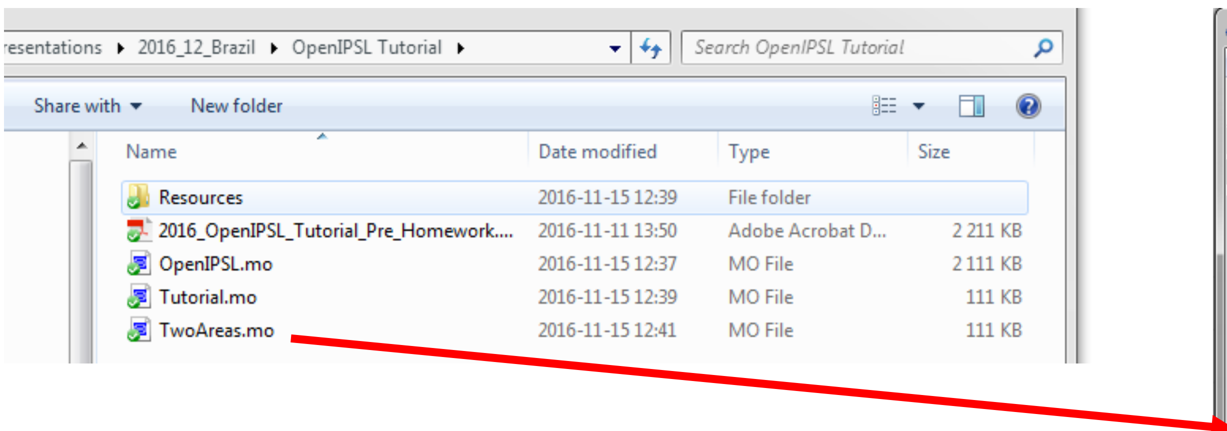
Bonus Stuff!

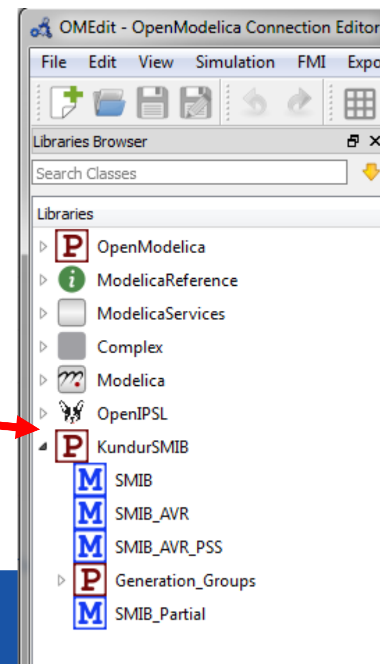
Other goodies and topics…

Bonus: KRK 2-Area model

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 **TwoAreas.mo** file to the **Library Browser** in OMEdit.





Demo of Modelica and Other Tools

Modelica and Python

Python opens countless new applications for OpenIPSL.

In this demo, the integration of Modelica and Python will be leveraged to perform a root locus.

OpenModelica

JMODELICA.ORG
OPEN SOURCE MODELICA PLATFORM
FOR MODELING, SIMULATION
AND OPTIMIZATION

python™

Demo of Modelica and Other Tools

Modelica and FMI (Functional Mockup Interface)

FMI is a standard for **model exchange** between different tools. Modelica is FMI compliant. Python opens countless new applications for OpenIPSL.

In this demo, FMI and the FMI toolbox will be leveraged to simulate an OpenIPSL model in Matlab

FUNCTIONAL MOCK-UP INTERFACE

MATLAB