11th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems

San Antonio, Texas, USA, May 9, 2023

(Mini) **Tutorial:** Getting Started with Power System Modeling using Modelica and the OpenIPSL

Prof. Luigi Vanfretti
Professor @ http://ALSETLab.com
Rensselaer Polytechnic Institute
Troy, NY, USA

Content

Startup Checklist

- ☐ Get the slides, solution/model and notebook in the following link or use the QR code above to go to the directory: https://tinyurl.com/23-OpenIPSL-Tutorial
- ☐ Did you installed OpenModelica? (Y/N)
  - If "No", get a copy from one of my USB sticks (it takes a long time to download and install… ~ 1GB installer, yikes!)
- ☐ Where you able to install OpenIPSL in OpenModelica using OpenModelica's package manager? (Y/N)
  - If "No", go to the "Quick Start Guide" and follow the instructions in slides 7 and 8.
- ☐ Where you able to run the OpenIPSL model as described in the "Quick Start Guide"? (Y/N)
  - If "No", it is going to be hard to do the tutorial with your computer.
  - Find a partner to work with!

SCAN ME

Intended Learning Outcomes

- To gain a general understanding of Modelica.
- To obtain basic familiarity with the OpenModelica environment.
- To provide a brief introduction to the OpenIPSL and help you gain basic understanding of it's uses for power system simulation.
- To implement a basic power system model using OpenIPSL, to simulate it, and to analyze it using different methods available in the OpenModelica environment.

- *Milestones!*
  - Complete Example 1, using OMEdit.
  - **Stretch Goal:** interactive analysis using OMNotebook on the Model from Example 1, including simulation and linearization.

Content

- Preliminaries:
  - Startup checklist and set-up
  - Intended Learning Outcomes for this Tutorial
- Modelica in Brief
- OpenModelica Overview
- OpenIPSL Overview
- Tutorial:
  - **Example 1:** SMIB Model Implementation and Simulation
- Stretch Goal:
  - SMIB Model Analysis using OMNotebook
- Where to go from here?

Modelica:
a language, a community and much more!

- Non-proprietary, object-oriented, equation-based *modeling language* for cyber physical systems .
- Open access (no paywall) & standardized language specification (link), maintained by the Modelica Association
- Open source Modelica Standard Library with more than 1,600 components models.
- *Supported by 9 tools natively*, both proprietary (Dymola, Modelon Impact, etc.) and Open Source (OpenModelica)
- A vast number of proprietary and open-source Modelica Libraries

- Organizations supporting Modelica lang and community development:
- And the development of other sister open access standards for M&S with interoperability at their core:

fmi Functional Mock-up Interface

efmi Functional Mock-up Interface for embedded systems

ssp System Structure & Parameterization

dcp Distributed Co-Simulation Protocol

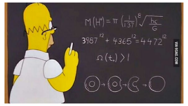
What is an Open Access Standard

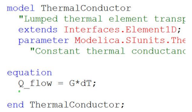
From Wikipedia: An open standard is a standard that is openly accessible and usable by anyone. It is also a prerequisite to use open license, non-discrimination and extensibility. Typically, anybody can participate in the development.

Modelica + Sister OA Stds (FMI, eFMI, etc):

Scope Separation Enables Multiple Uses of the Model

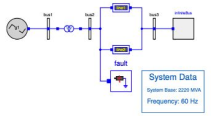
Modeling *by the Human*



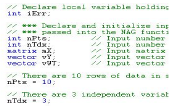


```
model ThermalConductor
  "Lumped thermal element transp
  extends Interfaces.Element1D;
  parameter Modelica.SIunits.The
    "Constant thermal conductanc

equation
  Q_flow = G*dT;

end ThermalConductor;
```

Equations



Diagrams



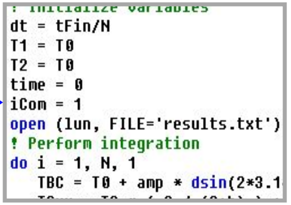
```
// Declare local variable holding
int iErr;

// *** Declare and initialize inp
// *** passed into the NAG functi
int nPts;          // Input number
int nTdx;          // Input number
matrix mX;         // Input matrix
vector vY;         // Input vector
vector vWT;        // Input vector

// There are 10 rows of data in s
nPts = 10;

// There are 3 independent variab
nTdx = 3;
```

Algorithms

Acausal, Transparent, Reusable Models in Modelica-Compliant Tools

Simulation Code Generation and Computation ***by Modelica-Compliant Software Tools (Computer)***

Code Generation

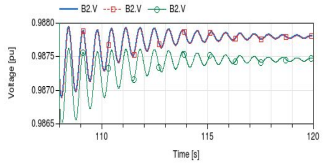


```
! Initialize variables
dt = tFin/N
T1 = T0
T2 = T0
time = 0
iCom = 1
open (lun, FILE='results.txt')
! Perform integration
do i = 1, N, 1
   TBC = T0 + amp * dsin(2*3.1
```

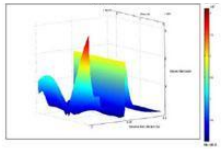
Procedural Code

Export

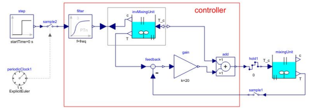
Exploit for Multiple Purposes ***And Applications***



Simulation and Linearization



Optimization



Control

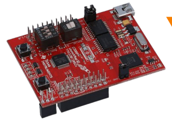
Embedded Systems

Within Modelica-Compliant SW (e.g., Dymola)

Interoperable and Portable Models in non-Modelica Environments via Sister Open-Access Standards (e.g. C/C++, Python, MATLAB)

fmi

efmi

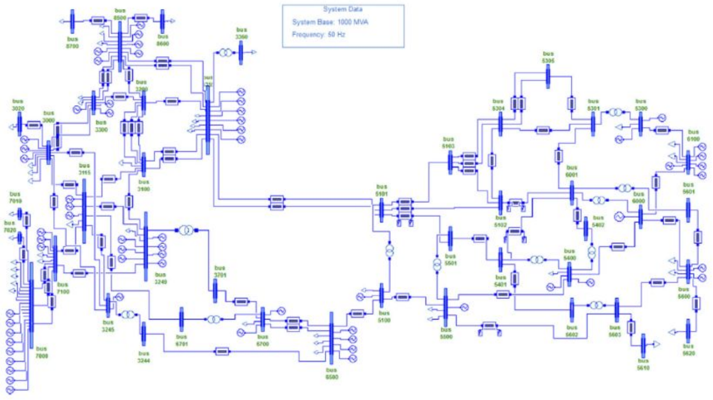


Other SW Env./HW

Modelica + Sister OA Stds (FMI, eFMI, etc):

One model portable across multiple tools!

*One **Open-Access** Standardized Modeling Language*



*With **Interoperability and Portability** in Modelica-Compliant and FMI-Compliant Tools*

Multiple Tools Natively Supporting the Modelica Language

DYMOLA

SIMULATIONX by ESI

MapleSim™

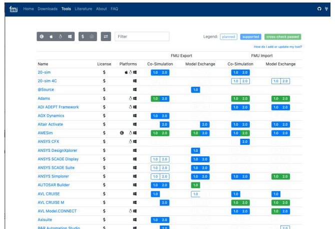
Wolfram SystemModeler™

OpenModelica

Multiple Tools Compliant with the FMI Model Exchange and Co-Simulation Standards

fmi Functional Mock-Up Interface

Supported by more than 170 tools!
https://fmi-standard.org/tools/



fmi Home Downloads Tools Literature About FAQ

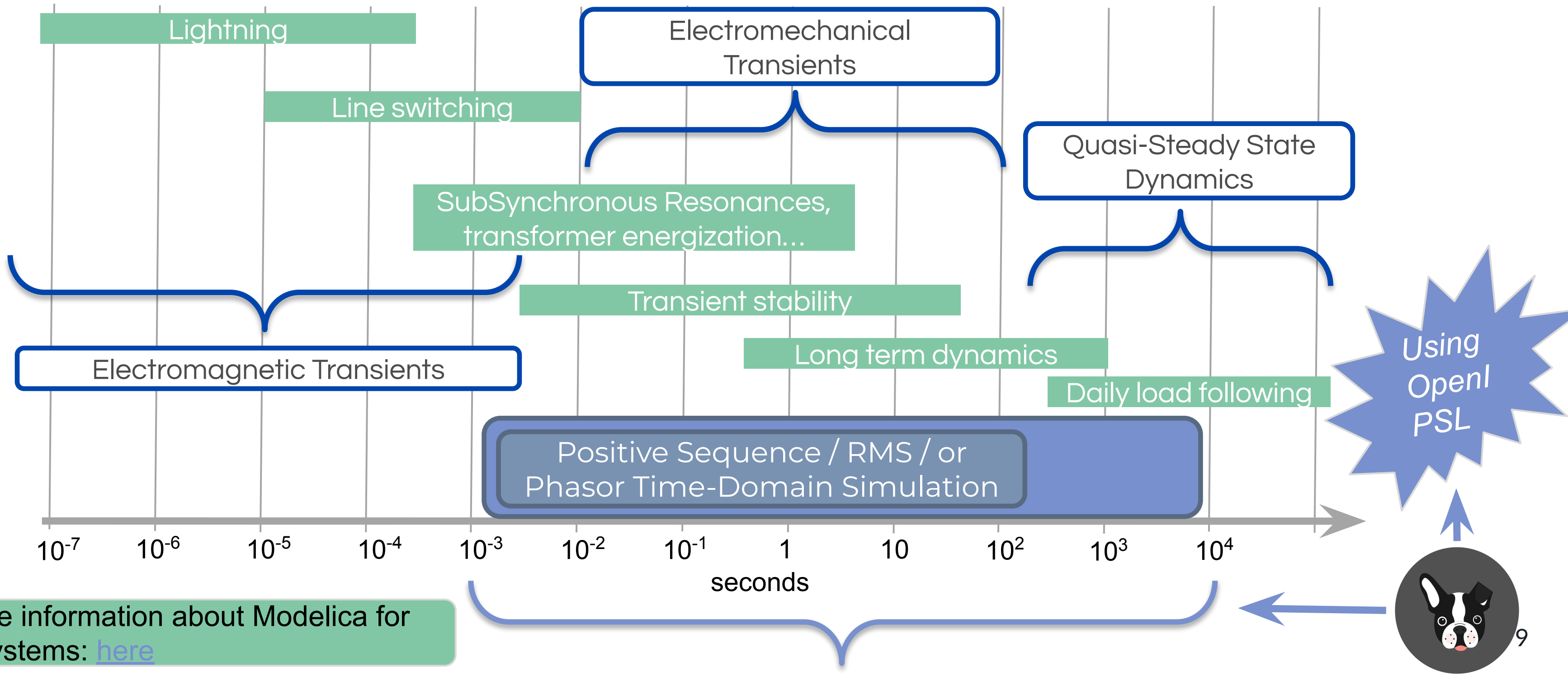
Filter

Legend: planned supported cross-check passed

How do I add or update my tool?

			FMU Export		FMU Import	
Name	License	Platforms	Co-Simulation	Model Exchange	Co-Simulation	Model Exchange
20-sim	$	🍎 🐧 ⊞	1.0 2.0		1.0 2.0	
20-sim 4C	$	⊞			1.0 2.0	1.0 2.0
@Source	$	⊞		1.0		
Adams	$	🐧 ⊞	1.0 2.0		1.0 2.0	1.0 2.0
ADI ADEPT Framework	$	🐧 ⊞			1.0 2.0	1.0 2.0
AGX Dynamics	$	⊞	1.0 2.0			
Altair Activate	$	⊞	2.0	2.0	1.0 2.0	1.0 2.0
AMESim	$	ⓒ 🐧 ⊞	1.0 2.0	1.0 2.0	1.0 2.0	1.0 2.0
ANSYS CFX	$	🐧 ⊞			2.0	
ANSYS DesignXplorer	$	⊞		1.0		
ANSYS SCADE Display	$	⊞	1.0 2.0	1.0 2.0		
ANSYS SCADE Suite	$	⊞	1.0 2.0	1.0 2.0		
ANSYS Simplorer	$	⊞	1.0 2.0	1.0 2.0	1.0 2.0	1.0 2.0
AUTOSAR Builder	$	⊞	1.0 2.0	1.0		
AVL CRUISE	$	⊞	1.0	1.0	1.0	1.0
AVL CRUISE M	$	⊞	2.0		1.0 2.0	1.0 2.0
AVL Model.CONNECT	$	🐧 ⊞			1.0 2.0	1.0 2.0
Axisuite	$	⊞	1.0 2.0			
B&R Automation Studio	$	⊞	2.0			2.0

Power system phenomena that can be modeled in the Modelica Language



See more information about Modelica for power systems: here

Content

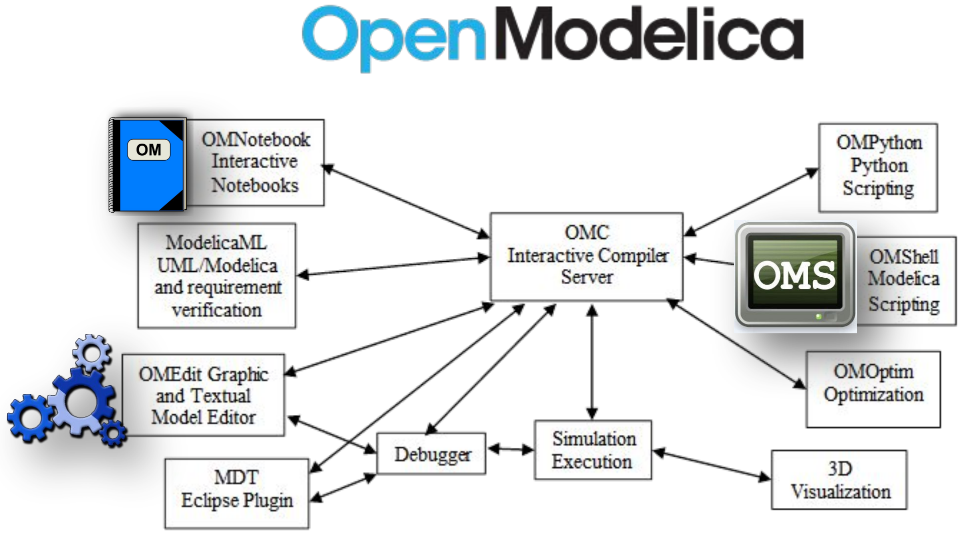
Understanding the OpenModelica Environment

- OpenModelica is an entire ecosystem, which at its core has the OpenModelica Compiler (OMC).

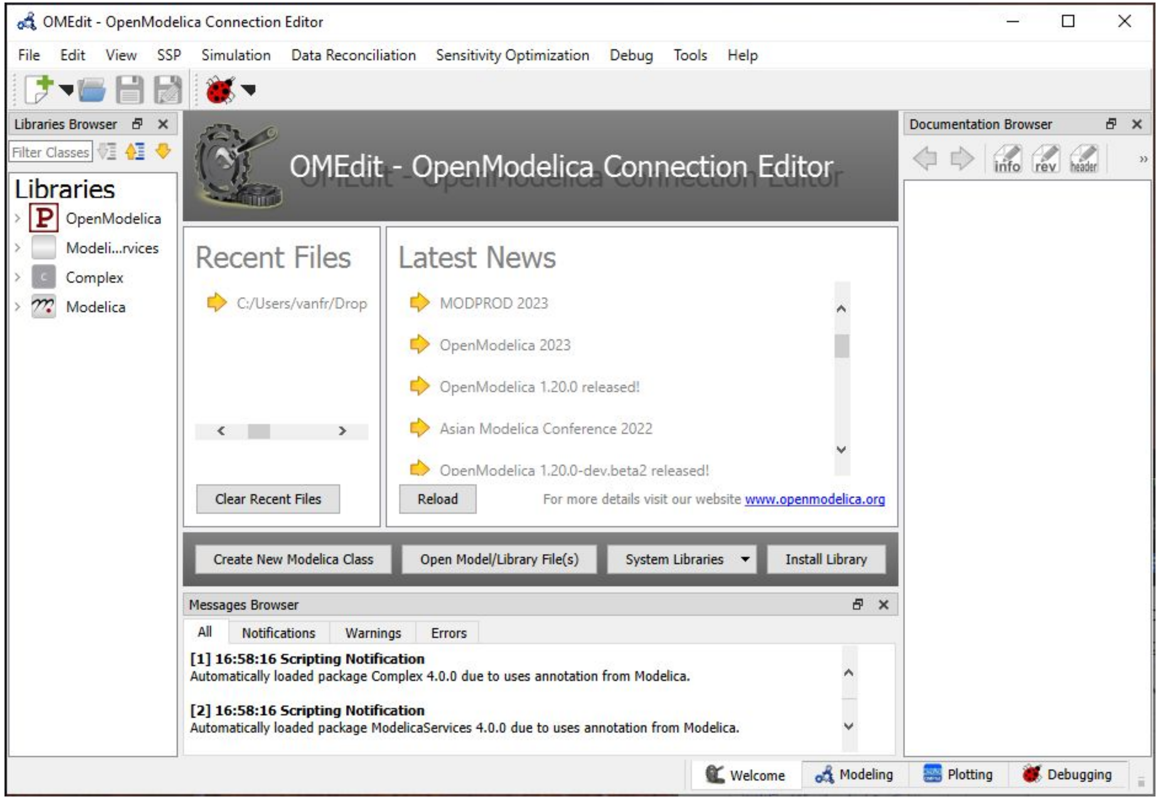
- There are many ways to interact with the OMC, and in this tutorial, we will use maily OMEdit.

- OMEdit:
    - Object-oriented graphical modeling.
    - To build, edit and simulate models.

- More about OpenModelica's Environment: https://openmodelica.org/doc/OpenModelicaUsersGuide/1.21/


OpenModelica
OM
OMNotebook Interactive Notebooks
ModelicaML UML/Modelica and requirement verification
OMEdit Graphic and Textual Model Editor
MDT Eclipse Plugin
OMC Interactive Compiler Server
Debugger
Simulation Execution
3D Visualization
OMPython Python Scripting
OMS
OMShell Modelica Scripting
OMOptim Optimization

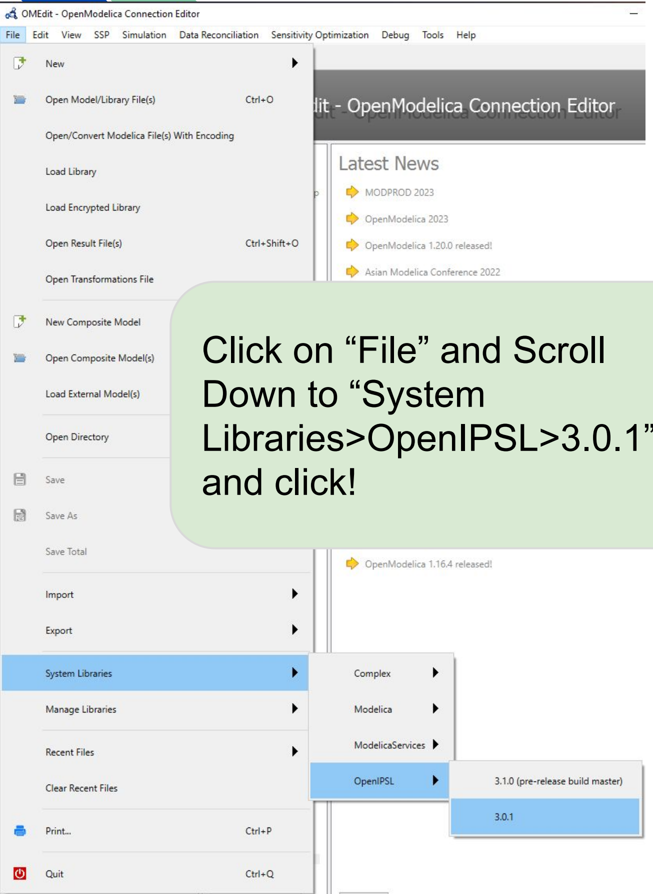
OMEdit - OpenModelica Graphical Modeling and Simulation Environment


OMEdit - OpenModelica Connection Editor
File Edit View SSP Simulation Data Reconciliation Sensitivity Optimization Debug Tools Help
Libraries Browser
Filter Classes
Libraries
OpenModelica
Modeli...rvices
Complex
Modelica
OMEdit - OpenModelica Connection Editor
Recent Files
C:/Users/vanfr/Drop
Clear Recent Files
Latest News
MODPROD 2023
OpenModelica 2023
OpenModelica 1.20.0 released!
Asian Modelica Conference 2022
OpenModelica 1.20.0-dev.beta2 released!
Reload
For more details visit our website www.openmodelica.org
Create New Modelica Class
Open Model/Library File(s)
System Libraries
Install Library
Documentation Browser
info
rev
header
Messages Browser
All
Notifications
Warnings
Errors
Welcome
Modeling
Plotting
Debugging

[1] 16:58:16 Scripting Notification
Automatically loaded package Complex 4.0.0 due to uses annotation from Modelica.

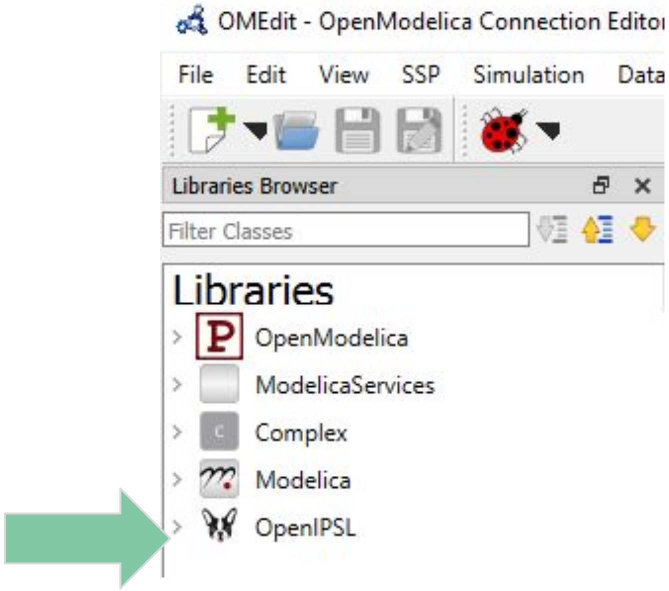
[2] 16:58:16 Scripting Notification
Automatically loaded package ModelicaServices 4.0.0 due to uses annotation from Modelica.

Loading OpenIPSL to OMEdit's Library Browser


OMEdit - OpenModelica Connection Editor
File Edit View SSP Simulation Data Reconciliation Sensitivity Optimization Debug Tools Help
New
Open Model/Library File(s) Ctrl+O
Open/Convert Modelica File(s) With Encoding
Load Library
Load Encrypted Library
Open Result File(s) Ctrl+Shift+O
Open Transformations File
New Composite Model
Open Composite Model(s)
Load External Model(s)
Open Directory
Save
Save As
Save Total
Import
Export
System Libraries
Manage Libraries
Recent Files
Clear Recent Files
Print... Ctrl+P
Quit Ctrl+Q
Complex
Modelica
ModelicaServices
OpenIPSL
3.1.0 (pre-release build master)
3.0.1
Latest News
MODPROD 2023
OpenModelica 2023
OpenModelica 1.20.0 released!
Asian Modelica Conference 2022
OpenModelica 1.16.4 released!

Click on "File" and Scroll Down to "System Libraries>OpenIPSL>3.0.1" and click!

OpenIPSL should be loaded now and ready to use!



Content

The *OpenIPSL* Project - *Origins*

- KTH SmarTS Lab (my former research team in Sweden) actively participated in the group or partners developing iPSL from 2012 until the end of the EU FP7 project *iTesla* project (March 2016)

- iPSL was a nice prototype, ***but we identified the following issues:***
  - Development: Need for compatibility with OpenModelica, (better) use of object orientation and proper use of the Modelica language features.
  - Maintenance: Poor harmonization, lack of code factorization, etc.
  - Human issues: The development workflow was complex
    - Different parties with disparate objectives, levels of knowledge, philosophy, etc.

  - These issues lead to a need of a different approach.

Fork: a software project going *in a different development direction*

- OpenIPSL *started as a **fork*** of iPSL in 2016, and has now largely evolved!
- OpenIPSL is hosted on GitHub at http://openipsl.org
- OpenIPSL is actively developed by ALSETLab members ***and friends***, as a research and education oriented library for power systems.

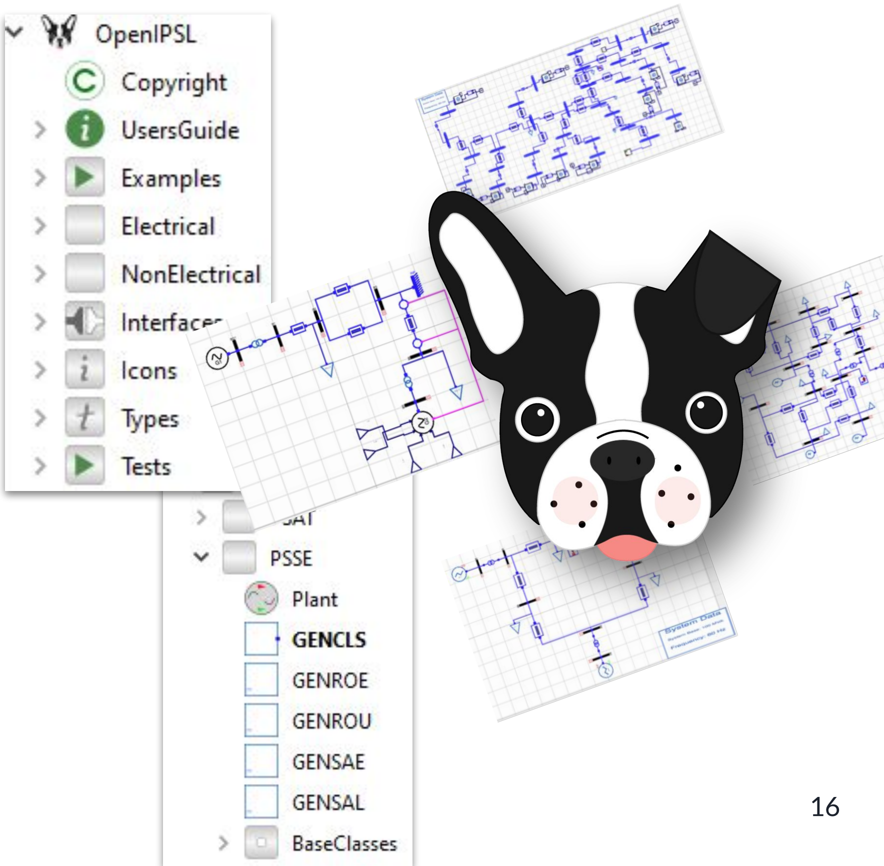
The *OpenIPSL* Library – Key Features

OpenIPSL is an open-source Modelica library for power systems that

- Contains a set of **power system components** for **phasor time domain** modeling and simulation of power systems
- Models have been **verified** against a number of reference tools (mainly PSS/E)

OpenIPSL enables:

- **Unambiguous** model exchange
- Formal **mathematical description** of models
- **Separation** of **models** from tools/IDEs and **solvers**
- Use of **object-oriented** paradigms



The *OpenIPSL* Library *Versions*

- Major updates to the library: https://github.com/OpenIPSL/OpenIPSL/releases
- What's different between versions - see previous publications of OpenIPSL:
  - [1] is pre-fork
  - [2] is for OpenIPSL v1.5.0
- Latest paper of OpenIPSL for Version 2.0.0:
  - Marcelo de Castro, Dietmar Winkler, Giuseppe Laera, Luigi Vanfretti, Sergio A. Dorado-Rojas, Tin Rabuzin, Biswarup Mukherjee, Manuel Navarro, Version [OpenIPSL 2.0.0] - [iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations], SoftwareX, Volume 21, 2023, 101277, ISSN 2352-7110, https://doi.org/10.1016/j.softx.2022.101277
- OpenIPSL Version 3.X.Y paper in preparation, will contain major updates of forthcoming PRs.

Development History

Year	MSL ver	Library ver	License
2012	MSL 3.2.1	Work starts! FP7 iTesla project funded for 4 years!	
5/2016	MSL 3.2.1	iPSL v1.0.0.	MPLv2.0
		Fork!	
12/2016	MSL 3.2.1	OpeniPSL v1.0.0.	MPLv2.0
11/2017	MSL 3.2.2	OpenIPSL v1.5.0	MPLv2.0
6/2022	MSL 3.2.3	OpenIPSL v2.0.0	3-Clause BSD
6/2022	MSL 4.0.0	OpenIPSL v3.0.1	3-Clause BSD

[1] L. Vanfretti, T. Rabuzin, M. Baudette, M. Murad, iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations, SoftwareX, Available online 18 May 2016, ISSN 2352-7110, http://dx.doi.org/10.1016/j.softx.2016.05.001

[2] M. Baudette, M. Castro, T. Rabuzin, J. Lavenius, T. Bogodorova, L. Vanfretti, "OpenIPSL: Open-Instance Power System Library — Update 1.5 to "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations"", SoftwareX, Volume 7, 2018, Pages 34-36, ISSN 2352-7110, https://doi.org/10.1016/j.softx.2018.01.002

Status and Coverage in Modelica Tools

Dymola

Modelon Impact

SIMULATION X by esi

MapleSim™

OpenModelica

Wolfram SystemModeler™

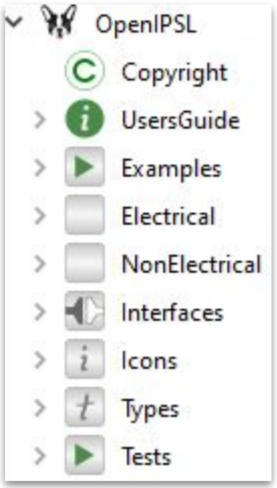
v.3.0.1 | Mostly Compatible | Partial Compatibility

Pre-release development build - v.3.0.1-dev | Improved Compatibility

Future: v.3.0.x

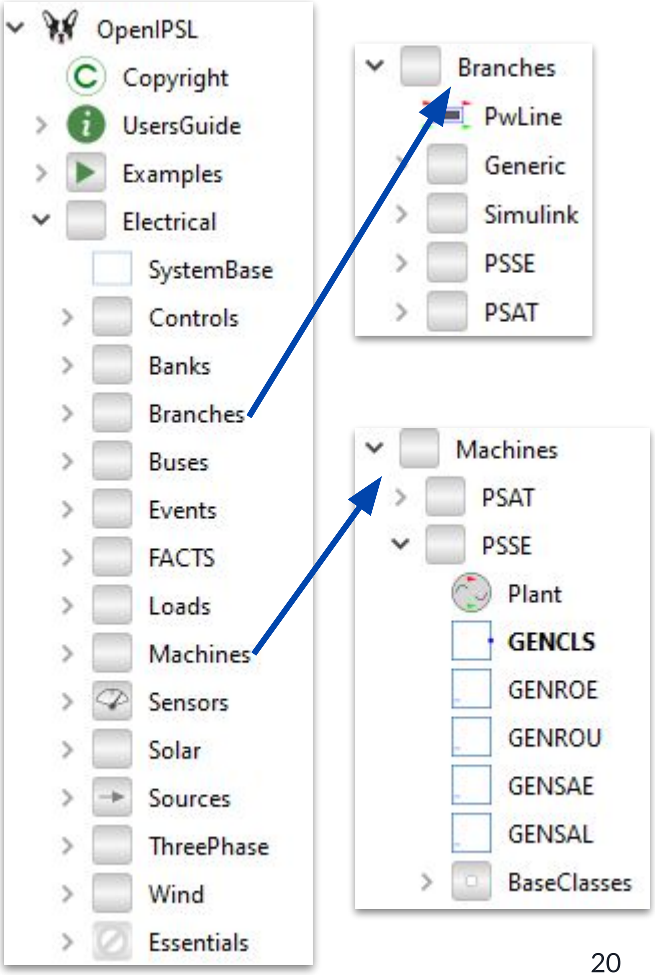
OpenIPSL - Library Structure

- The library is divided in the following sub-packages:
  - *Examples:* different types of power system models, from textbooks and the real-world.
  - *Electrical:* power system components (e.g. synchronous machines, excitation systems, loads, etc.)
  - *Non-electrical:* functions used by different components in the electrical package (e.g. saturation functions, specialized integrators, etc.)
  - *Interfaces:* specialized "pin" for acasual electrical coupling and other (future) interfaces.
  - *Icons:* defines icon for verified models (those checked against PSS/E).
  - *Types:* defines type units such as voltage, current, etc., and provides nominal values useful for scaling during initialization.
  - *Tests:* unit testing models for all library components. Meant for functional testing of implementation, not for illustration/analysis purposes.



OpenIPSL - Electrical

- The *Electrical* package contains most of the components that comprise an actual power network model.
- It includes electrical machines (synchronous generators, motros), transmission lines, loads, excitation systems, turbine+governors, etc.
  - These are used to build the power system network models.
- Under each category, you can find different types of models organized according to their original source, the main ones are:
  - PSAT: validated against Prof. Milano’s PSAT sw.
  - PSS/E: validated against PSS/E.
  - Generic: models taken from standards or literature without verification against another tool.



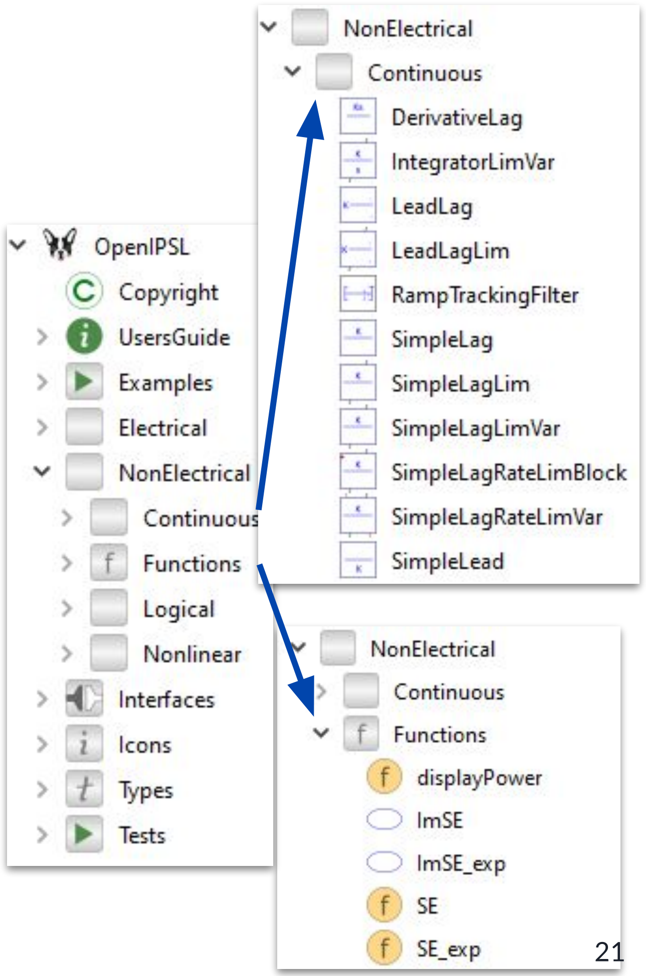
- OpenIPSL
  - Copyright
  - UsersGuide
  - Examples
  - Electrical
    - SystemBase
    - Controls
    - Banks
    - Branches
    - Buses
    - Events
    - FACTS
    - Loads
    - Machines
    - Sensors
    - Solar
    - Sources
    - ThreePhase
    - Wind
    - Essentials

- Branches
  - PwLine
  - Generic
  - Simulink
  - PSSE
  - PSAT

- Machines
  - PSAT
  - PSSE
    - Plant
    - GENCLS
    - GENROE
    - GENROU
    - GENSAE
    - GENSAL
    - BaseClasses

OpenIPSL - NonElectrical

- The ***NonElectrical*** sub-package is comprised by functions, blocks or models, which are used to build the the models in the ***Electrical*** sub-package:
    - *Transfer functions, logical operators, etc.*
- They perform specific operations which were not available in the Modelica Standard Library (MSL)
- Necessary to replicate the behavior of proprietary tools for basic functionalities, e.g. integrators with limiters..., all the way to complex functions (generator saturation model).
- Extremely important when aiming to reproduce PSS/E behavior.



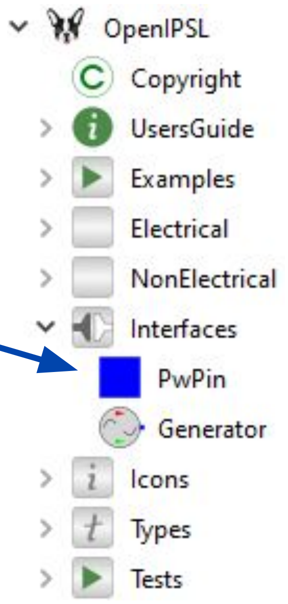
- OpenIPSL
  - Copyright
  - UsersGuide
  - Examples
  - Electrical
  - NonElectrical
    - Continuous
    - Functions
    - Logical
    - Nonlinear
  - Interfaces
  - Icons
  - Types
  - Tests

- NonElectrical
  - Continuous
    - DerivativeLag
    - IntegratorLimVar
    - LeadLag
    - LeadLagLim
    - RampTrackingFilter
    - SimpleLag
    - SimpleLagLim
    - SimpleLagLimVar
    - SimpleLagRateLimBlock
    - SimpleLagRateLimVar
    - SimpleLead

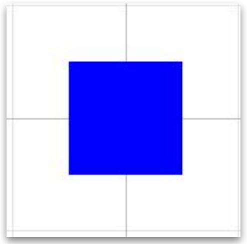
- NonElectrical
  - Continuous
  - Functions
    - displayPower
    - ImSE
    - ImSE_exp
    - SE
    - SE_exp

OpenIPSL - Interfaces

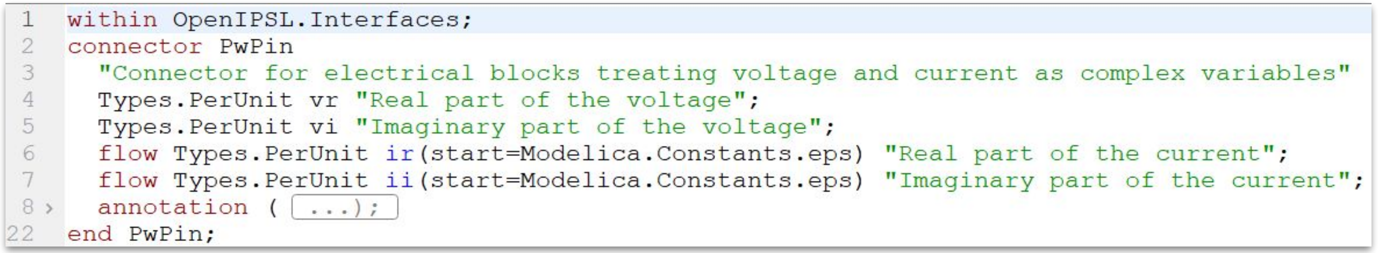
- The ***Interfaces*** package contains a set of specifically developed Modelica connectors to make acasual connections between the electrical models in this library
  - The most important is *PwPin* a connector, which contains voltage and current quantities in phasor representation (real and imaginary components of the complex number).
  - A container to build up a "source" or "sink" sub-system (e.g. a power plant with multiple machines, etc.) is also included.
- Other interfaces under development, e.g. multi-domain interface to couple mechanical+thermofluidic models of turbines, etc., will be included here in the future.



Icon View

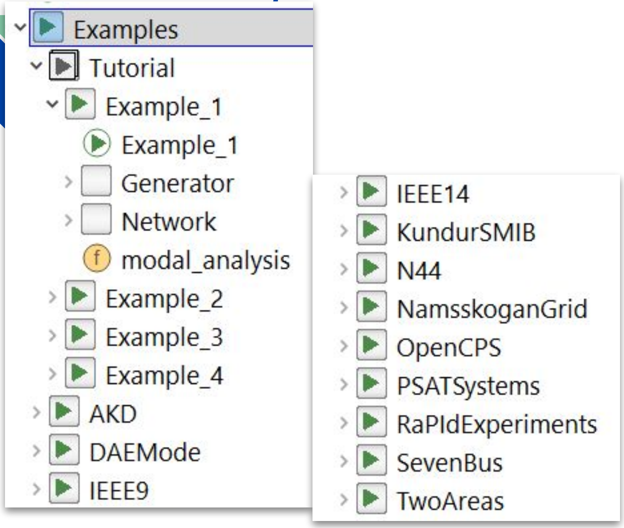


Text View



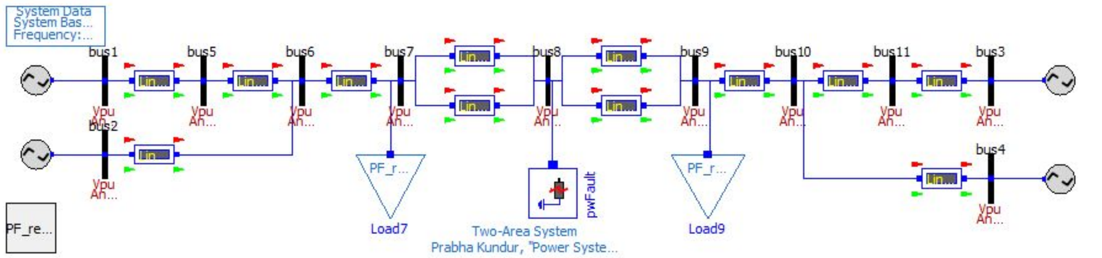
```
within OpenIPSL.Interfaces;
connector PwPin
  "Connector for electrical blocks treating voltage and current as complex variables"
  Types.PerUnit vr "Real part of the voltage";
  Types.PerUnit vi "Imaginary part of the voltage";
  flow Types.PerUnit ir(start=Modelica.Constants.eps) "Real part of the current";
  flow Types.PerUnit ii(start=Modelica.Constants.eps) "Imaginary part of the current";
  annotation (...);
end PwPin;
```

OpenIPSL – Examples

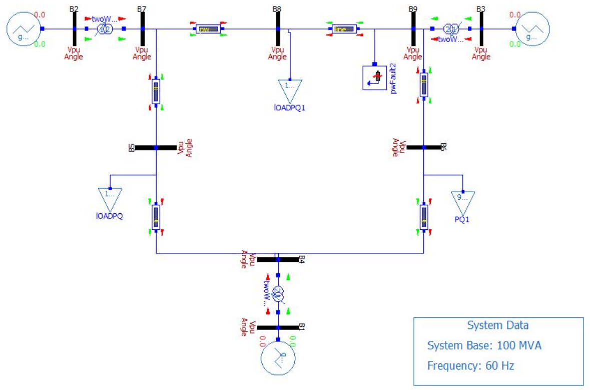


- Examples
  - Tutorial
    - Example_1
      - Example_1
      - Generator
      - Network
      - modal_analysis
    - Example_2
    - Example_3
    - Example_4
  - AKD
  - DAEMode
  - IEEE9
  - IEEE14
  - KundurSMIB
  - N44
  - NamsskoganGrid
  - OpenCPS
  - PSATSystems
  - RaPIdExperiments
  - SevenBus
  - TwoAreas

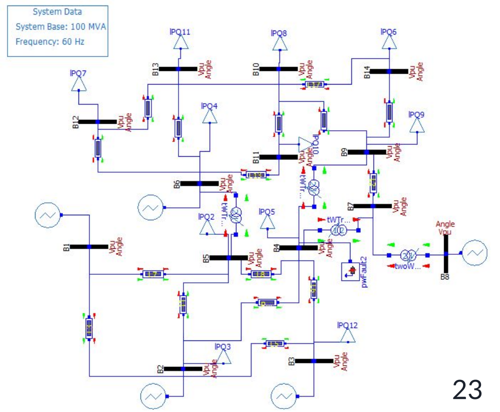
Klein-Rogers-Kundur 2-Area 4-Machine System



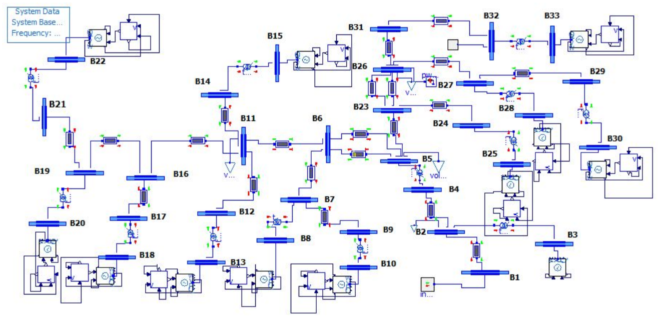
IEEE 9 Bus



IEEE 14 Bus

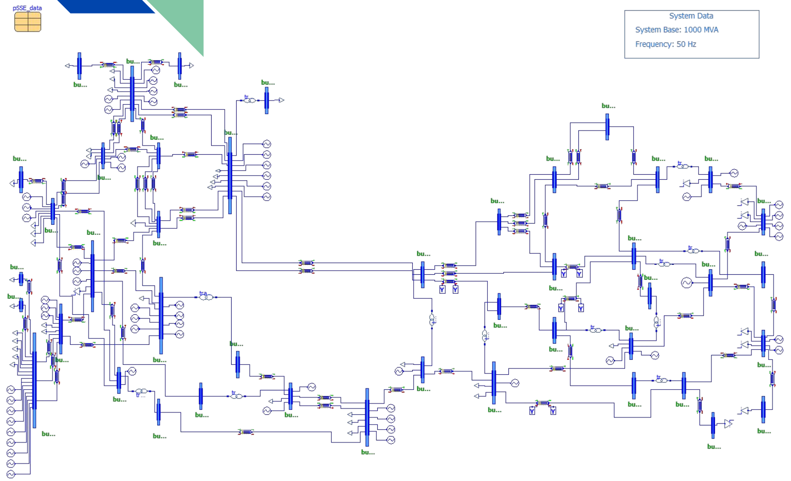


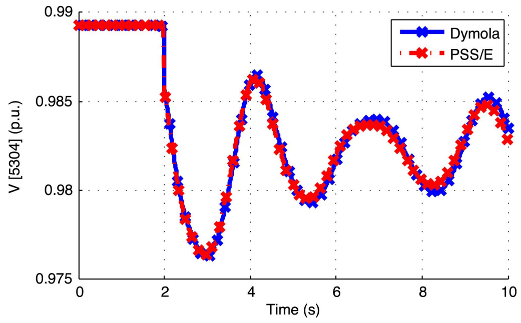
Namsskogan Distribution Network



OpenIPSL - Examples

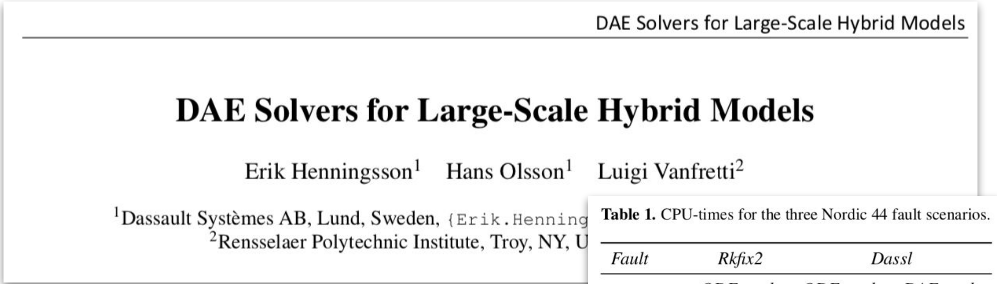
But can you model/simulate large networks?





Yes!

- Simulation performance depends on the simulation tool, not the model!
  - Example: OpenIPS.Examples.DAEMode.N44_Original_Systems
  - While you can simulate this model in several Modelica tools, some tools have better performance than others.
- Key: The Modelica standard enables model portability, and thus, *facilitates competition between software tools!*
- **Dymola 2019FD02 has shown to be faster than PSS/E,** as reported in the following paper: https://github.com/OpenIPSL/2019_Modelica_Conf_DAESolvers4LargeHybridModels



DAE Solvers for Large-Scale Hybrid Models

DAE Solvers for Large-Scale Hybrid Models

Erik Henningsson[1] Hans Olsson[1] Luigi Vanfretti[2]

[1]Dassault Systèmes AB, Lund, Sweden, {Erik.Henning
[2]Rensselaer Polytechnic Institute, Troy, NY, U

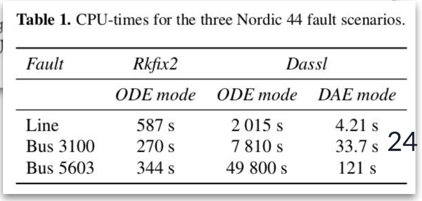
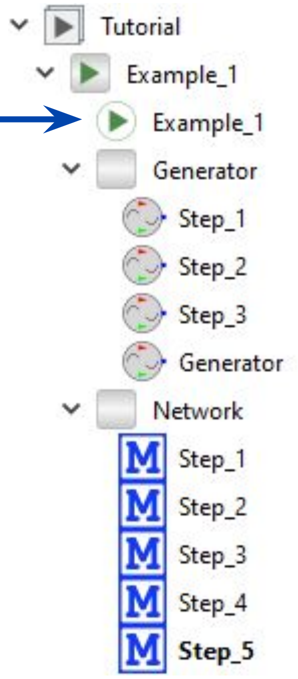


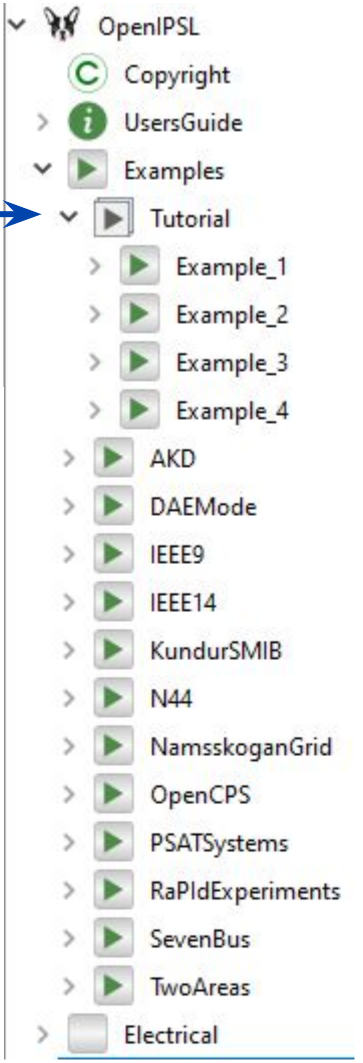
Table 1. CPU-times for the three Nordic 44 fault scenarios.

Fault	*Rkfix2*	*Dassl*	
	ODE mode	*ODE mode*	*DAE mode*
Line	587 s	2 015 s	4.21 s
Bus 3100	270 s	7 810 s	33.7 s
Bus 5603	344 s	49 800 s	121 s

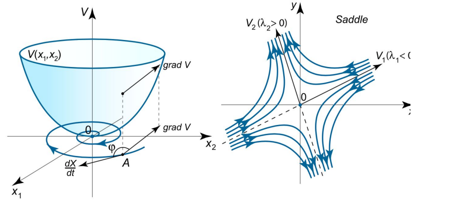
OpenIPSL - Examples.Tutorial

- ***Examples.Tutorial:***
- In this tutorial, the *Tutorial* package will be used to illustrate basic use examples of the library
- In the packages *Example_1*, …, all steps to build the models are provided.
  - The final ”answer” (i.e. model) is shown with a “Play” icon.


Tutorial
Example_1
Example_1
Generator
Step_1
Step_2
Step_3
Generator
Network
Step_1
Step_2
Step_3
Step_4
Step_5


OpenIPSL
Copyright
UsersGuide
Examples
Tutorial
Example_1
Example_2
Example_3
Example_4
AKD
DAEMode
IEEE9
IEEE14
KundurSMIB
N44
NamsskoganGrid
OpenCPS
PSATSystems
RaPIdExperiments
SevenBus
TwoAreas
Electrical

OpenIPSL - Providing a good *"initial guess"*

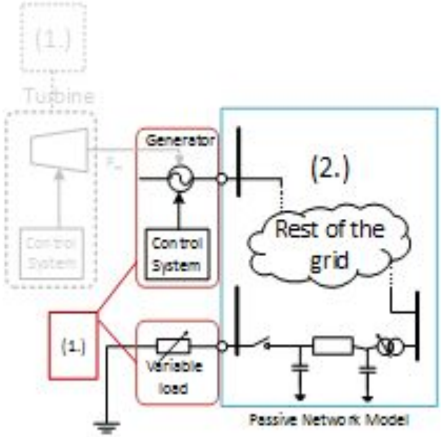


- The power system needs to be in equilibrium before running the simulation or if stable, after a disturbance is applied, it must converge to an equilibrium.
  - Q: How can we find this equilibrium?
  - A: Set derivatives to zero and solve for all unknown variables!

$$\begin{aligned}\dot{\mathbf{x}} &= f(\mathbf{x},\mathbf{y},\mathbf{u})\\ \mathbf{0} &= g(\mathbf{x},\mathbf{y},\mathbf{u})\end{aligned} \quad \Rightarrow \quad \begin{aligned}\mathbf{0} &= f(\mathbf{x},\mathbf{y},\mathbf{u})\\ \mathbf{0} &= g(\mathbf{x},\mathbf{y},\mathbf{u})\end{aligned}$$

Modelica –compliant tools attempt to solve this problem

- ***Need for a good initial guess:***
  - Let equation set g be separated in two sets, g_1 and g_2
    - (1) describes the dynamics (e.g. generators and their control systems, etc.), they depend on both $\boldsymbol{x}$ and $\boldsymbol{y}$, i.e. you have both differential and algebraic variables.
    - (2) Is the network model, consisting of transmission lines and other passive components which only depends on algebraic variables, $\boldsymbol{y}$.
- Finding the equilibrium for (1) and (2) is a difficult numerical problem, we address this by providing a good initial guess:
  - Our models in OpenIPSL derive these initial guesses from power flow input data, i.e. the solution of (2) which only solves for $\boldsymbol{y}$.
  - Note that a solution of (2) from a power flow solver helps, but does not guarantee an equilibrium, i.e. solution to **both** (1) and (2).



$$\begin{aligned}\dot{\mathbf{x}} &= f(\mathbf{x},\mathbf{y},\mathbf{u})\\ \mathbf{0} &= g_1(\mathbf{x},\mathbf{y},\mathbf{u})\end{aligned} \quad (1)$$

$$\mathbf{0} = g_2(\quad \mathbf{y},\mathbf{u}) \quad (2)$$

OpenIPSL - Providing a good "initial guess"

- **An initial guess for all algebraic, continuous and discrete variables need to be provided to solve a numerical problem!**

  → For example, when solving differential equations, one needs to provide the **initial guess** of the state variables at rest.

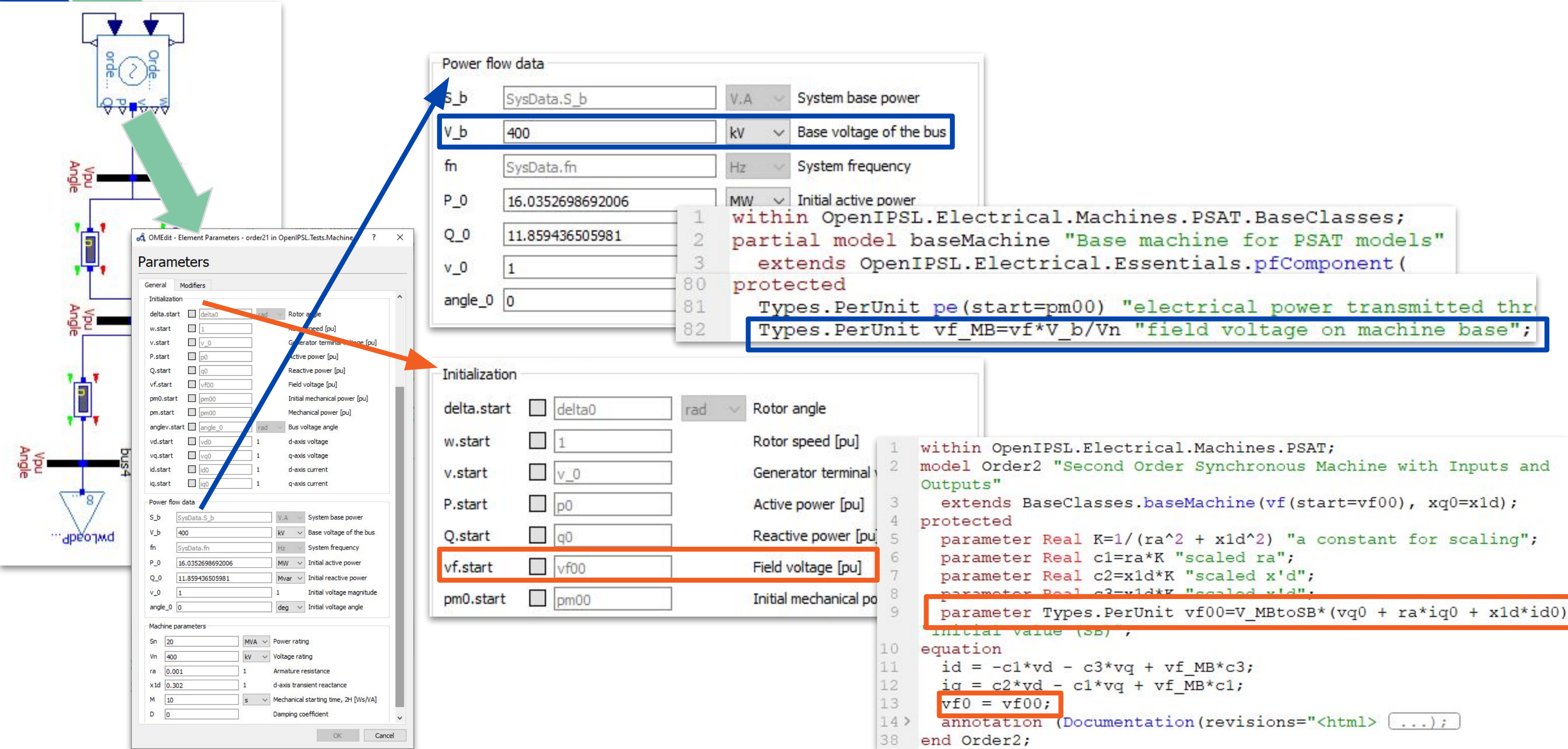
- In Modelica, the **initial values of states can be either solved or specified** in many ways:
  - Using `initial equation`

    `x = some_value OR x = expression to solve`
  - Setting the (`fixed=true, start=x0`) attribute when instantiating a model when the start value is known (or possible to calculate)
  - If nothing is specified, a default would be a guess value will be set by the Modelica-tool, such as (`start= 0, fixed=false`).

- In the OpenIPSL models we do the following:
  - **We compute an initial guess value for all required variables in the parameter section of the model definition.**
  - The initial guess value is then set with (`fixed = false`) for the solution of the initialization problem.

- In the OpenIPSL models we do the following:
  - We obtain these values from a power flow solution via an external tool (e.g. PSS/E).
  - This is used as a starting point to compute initial guess values through parameters within each model.
  - The power flow solution is NOT the initial guess value itself, it helps to provide a starting value to the Modelica-tool to solve the initialization problem.

OpenIPSL - Providing a good "initial guess"



OMEdit - Element Parameters - order21 in OpenIPSL.Tests.Machine

Parameters

General | Modifiers

Initialization

delta.start	☐	delta0	rad	Rotor angle
w.start	☐	1		Rotor speed [pu]
v.start	☐	v_0		Generator terminal voltage [pu]
P.start	☐	p0		Active power [pu]
Q.start	☐	q0		Reactive power [pu]
vf.start	☐	vf00		Field voltage [pu]
pm0.start	☐	pm00		Initial mechanical power [pu]
pm.start	☐	pm00		Mechanical power [pu]
anglev.start	☐	angle_0	rad	Bus voltage angle
vd.start	☐	vd0	1	d-axis voltage
vq.start	☐	vq0	1	q-axis voltage
id.start	☐	id0	1	d-axis current
iq.start	☐	iq0	1	q-axis current

Power flow data

S_b	SysData.S_b	V.A	System base power
V_b	400	kV	Base voltage of the bus
fn	SysData.fn	Hz	System frequency
P_0	16.0352698692006	MW	Initial active power
Q_0	11.859436505981	Mvar	Initial reactive power
v_0	1	1	Initial voltage magnitude
angle_0	0	deg	Initial voltage angle

Machine parameters

Sn	20	MVA	Power rating
Vn	400	kV	Voltage rating
ra	0.001	1	Armature resistance
x1d	0.302	1	d-axis transient reactance
M	10	s	Mechanical starting time, 2H [Ws/VA]
D	0		Damping coefficient

OK | Cancel

Power flow data

S_b	SysData.S_b	V.A	System base power
V_b	400	kV	Base voltage of the bus
fn	SysData.fn	Hz	System frequency
P_0	16.0352698692006	MW	Initial active power
Q_0	11.859436505981		
v_0	1		
angle_0	0		

Initialization

delta.start	☐	delta0	rad	Rotor angle
w.start	☐	1		Rotor speed [pu]
v.start	☐	v_0		Generator terminal
P.start	☐	p0		Active power [pu]
Q.start	☐	q0		Reactive power [pu]
vf.start	☐	vf00		Field voltage [pu]
pm0.start	☐	pm00		Initial mechanical po

```
within OpenIPSL.Electrical.Machines.PSAT.BaseClasses;
partial model baseMachine "Base machine for PSAT models"
  extends OpenIPSL.Electrical.Essentials.pfComponent(
protected
  Types.PerUnit pe(start=pm00) "electrical power transmitted thr
  Types.PerUnit vf_MB=vf*V_b/Vn "field voltage on machine base";
```

```
within OpenIPSL.Electrical.Machines.PSAT;
model Order2 "Second Order Synchronous Machine with Inputs and
Outputs"
  extends BaseClasses.baseMachine(vf(start=vf00), xq0=x1d);
protected
  parameter Real K=1/(ra^2 + x1d^2) "a constant for scaling";
  parameter Real c1=ra*K "scaled ra";
  parameter Real c2=x1d*K "scaled x'd";
  parameter Real c3=x1d*K "scaled x'd";
  parameter Types.PerUnit vf00=V_MBtoSB*(vq0 + ra*iq0 + x1d*id0)
  initial value (SB)";
equation
  id = -c1*vd - c3*vq + vf_MB*c3;
  iq = c2*vd - c1*vq + vf_MB*c1;
  vf0 = vf00;
  annotation (Documentation(revisions="<html> ...);
end Order2;
```

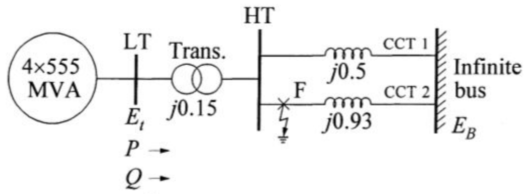
Content

Example 1 - Origins

- This example was originally presented in the reference book:
    - P. Kundur, “Power System Stability and Control”, McGraw-Hill Inc., Palo Alto, California, 1994. See: Example 13.2, pp. 864 – 869.
    - *IT is NOT exactly the same as in the book;* but can reproduce the same phenomena.

Example 13.2

In this example, we analyze the transient stability of the system of Figure E13.1 (considered in Example 13.1) including the effects of rotor circuit dynamics and excitation control. The system diagram is reproduced here as Figure E13.6 for reference.



Network reactances in pu on 2220 MVA base

Figure E13.6

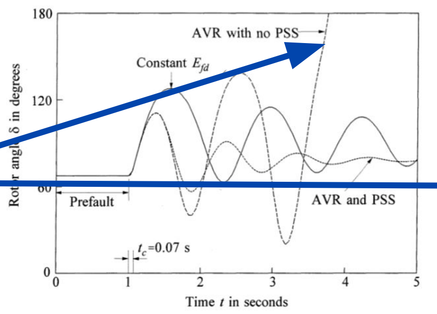


Figure E13.7 (a) Rotor angle response with fault cleared in 0.07 s

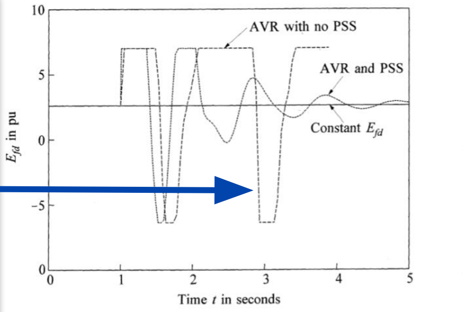


Figure E13.7 (d) Exciter output voltage response with fault cleared in 0.07 s

Example 1 - Background

- The model is used in senior/graduate courses for analysis of the so-called “transient stability” and “small-signal stability” (linearized analysis, eigenanalysis, etc.) of the system including the effects of rotor circuit dynamics and the excitation control system.

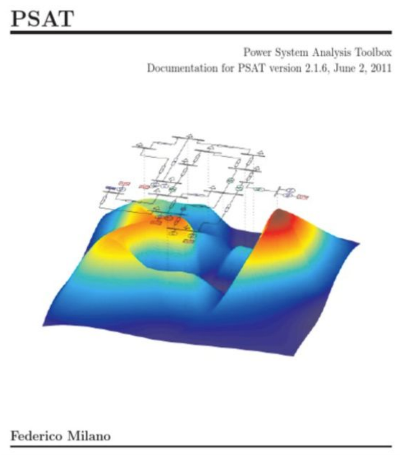
This phenomena observed in early days of “interconnections” (circa 1960’s), and first formally explained by de Mello and Concordia in 1969.

F. P. Dmello and C. Concordia, “Concepts of synchronous machine stability as affected by excitation control,” IEEE Trans. Power App. Syst., vol. PAS-88, pp. 316–329, 1969.

- It aims to represent a power plant (with many units) by using a single aggregate generating unit (with all it’s voltage control systems) connected via transformer and parallel lines to an “infinite bus”.
  - Hence the name Single Machine Infinite Bus (SMIB) system
  - In the French-speaking world they also call it OMIB (One-Machine …)

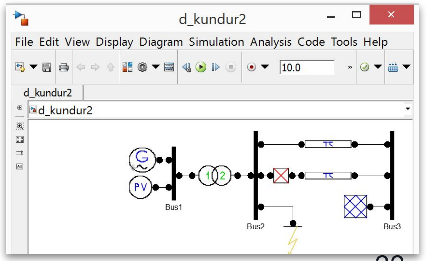
Example 1:
Gathering Parameter Data and Power Flow Results

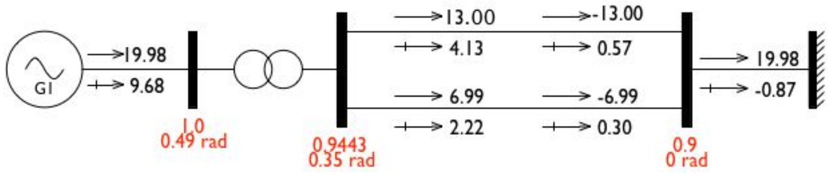
- We need parameter data for the models as well as a power flow solution: recall need for good initial guess!).
- To simplify the process, since we have models validated against PSAT, we will use data from PSAT as it is open source software.
- If you are interested in PSAT here is Prof. Milano's page: http://faraday1.ucd.ie
- You don't need PSAT to do anything right now!
  - The model used in this example exists as an example in PSAT and can be used for power flow calculations and dynamic simulations.
  - The parameter data used in the next slides, is that from PSAT's model implementation.
  - Power flow results were obtained using PSAT's power flow solution.
  - Summary power flow solution:


PSAT

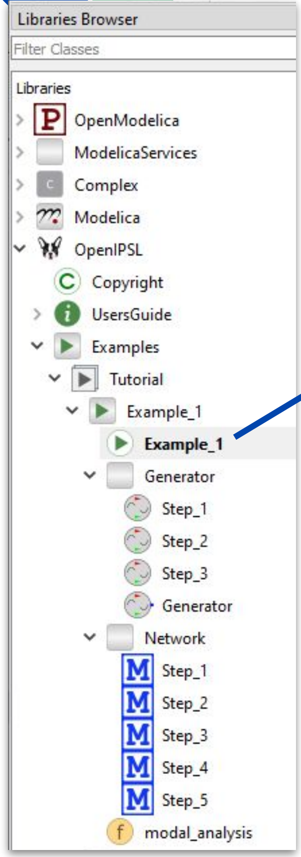
Power System Analysis Toolbox
Documentation for PSAT version 2.1.6, June 2, 2011

Federico Milano


d_kundur2
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
d_kundur2
d_kundur2
PV
Bus1
Bus2
Bus3

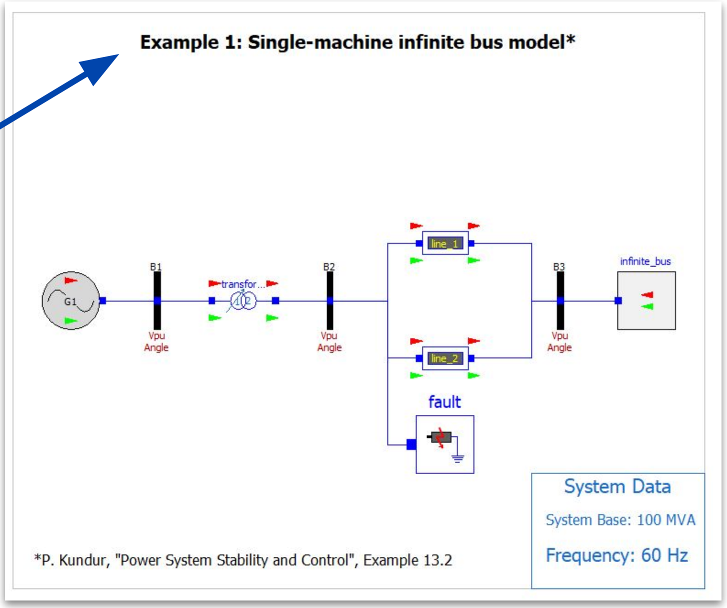


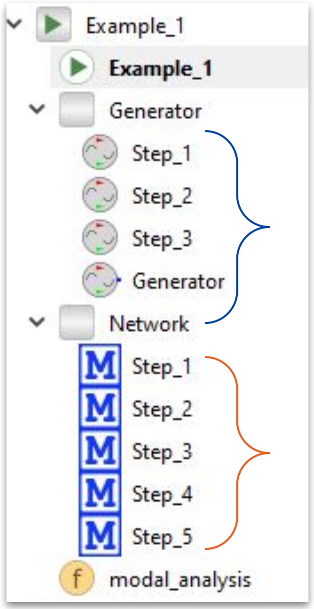
Example 1 – Let's Start Implementing



Our goal is to reproduce the model in OpenIPSL.Examples.Tutorial.Example_1.Example_1.

- The process is separated in two stages: (1) building the "Generator" and (2) building the "Network".
- Each stage has multiple steps, which we will do on our own here.
- The package with the library gives you the "solution", so you can open another instance of OpenModelica to verify what you are doing.





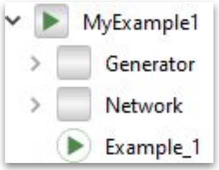
Stage (1), building the "Generator"

Stage (2), building the "Network"

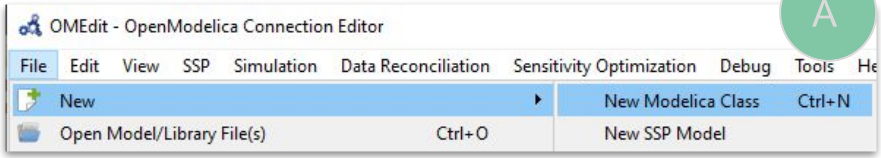
See solution in: OpenIPSL.Examples.Tutorial.Example_1

Example 1 – Creating the package structure

- First, we will setup a package structure as such:



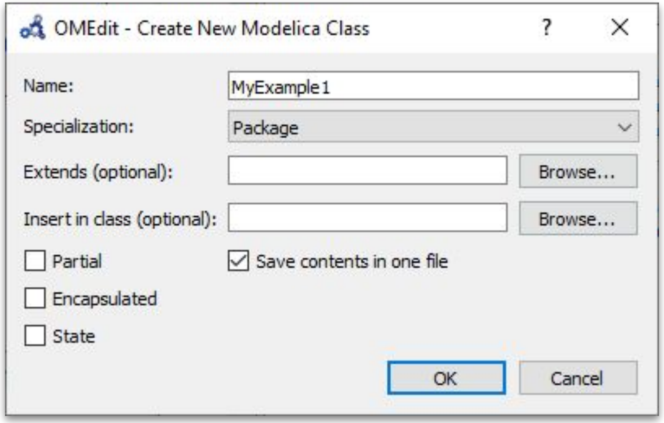
- To create the package structure for our example:


A
OMEdit - OpenModelica Connection Editor
File
Edit
View
SSP
Simulation
Data Reconciliation
Sensitivity Optimization
Debug
Tools
He
New
Open Model/Library File(s) Ctrl+O
New Modelica Class Ctrl+N
New SSP Model

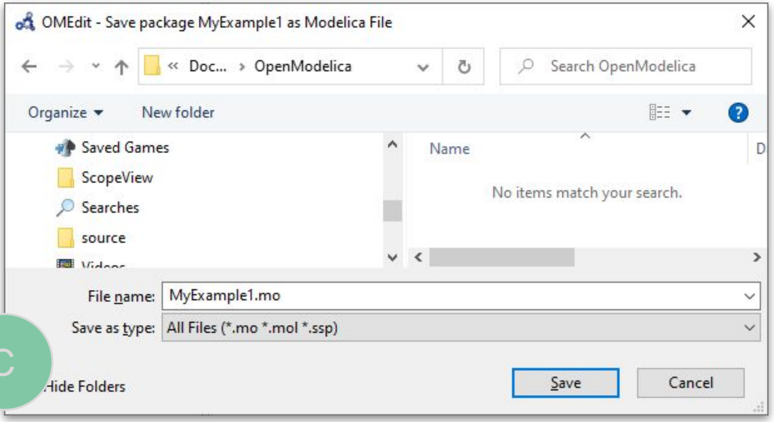
(A) In OM's toolbar go to File > New > New Modelica Class.

(B) This will open up the following window (1): In Name we enter "`MyExample1`", Specialization we select "`Package`" and make sure you click on the bottom check boxes the option "`Save contents in one file`"

(C) Press "Cntrl+S" so you can save it in an easy to find location, e.g., ./…/Documents/OpenModelica



Name: MyExample1
Specialization: Package
Extends (optional):
Insert in class (optional):
☐ Partial ☑ Save contents in one file
☐ Encapsulated
☐ State



File name: MyExample1.mo
Save as type: All Files (*.mo *.mol *.ssp)

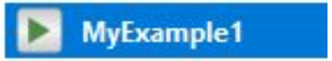
See solution in: OpenIPSL.Examples.Tutorial.Example_1

Example 1 – Creating the package structure

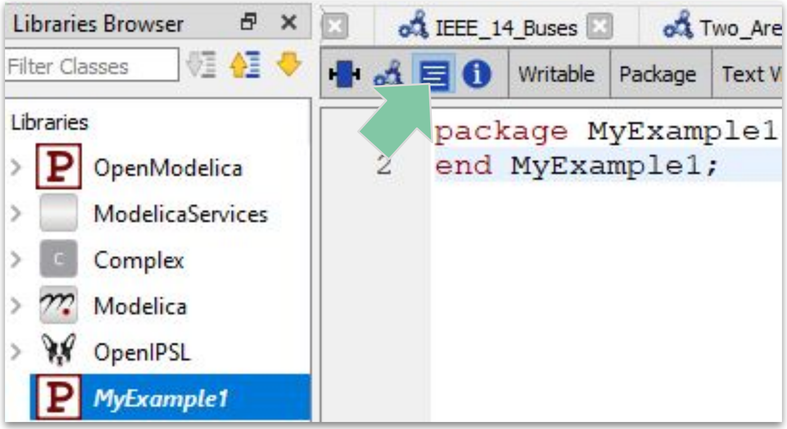
- Click on “MyExample1” and then select the “Text View”, enter the following instruction:

```
extends Modelica.Icons.ExamplesPackage;
```

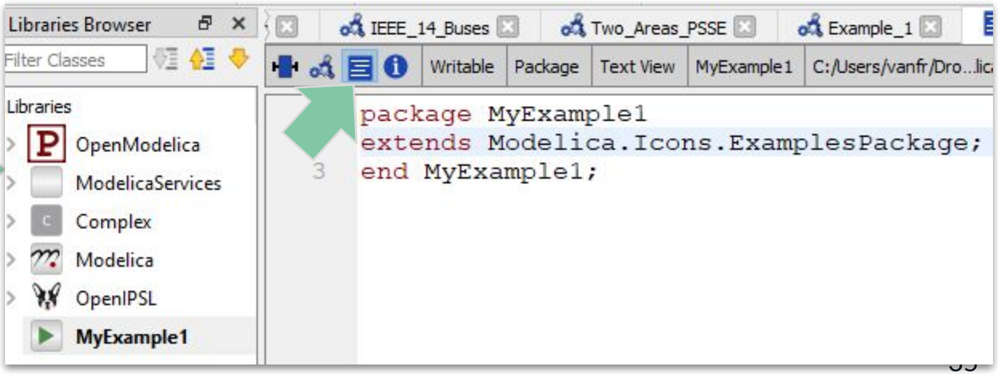
- Look at what happened to the icon of the package!



- You’ve just learned how to do inheritance in Modelica, yey!



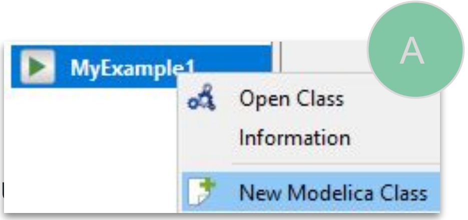
```
package MyExample1
end MyExample1;
```



```
package MyExample1
extends Modelica.Icons.ExamplesPackage;
end MyExample1;
```

See solution in: OpenIPSL.Examples.Tutorial.Example_1

Example 1 – Creating the package structure



- Now we continue building the package structure.

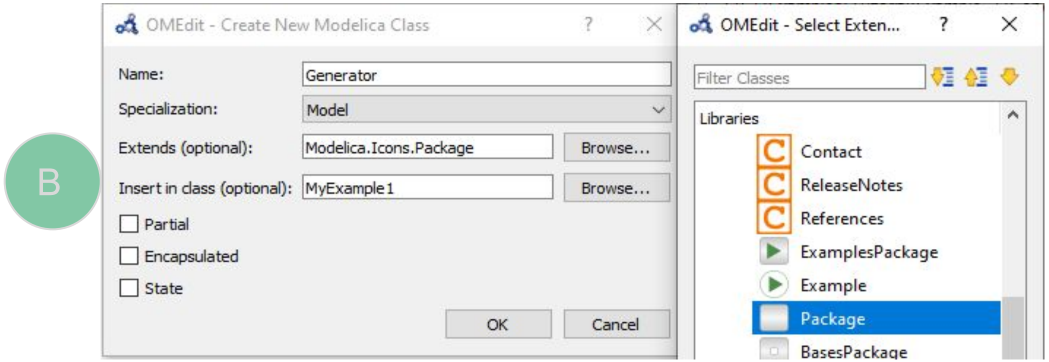
(A) Right click on "MyExample1" and select "New Modelica Class" from the context menu

(B) We now create a package called "Generator" and insert it in "MyExample1", we now choose to extend graphically the `Modelica.Icons.Package`icon by scrolling and finding the class through the window.

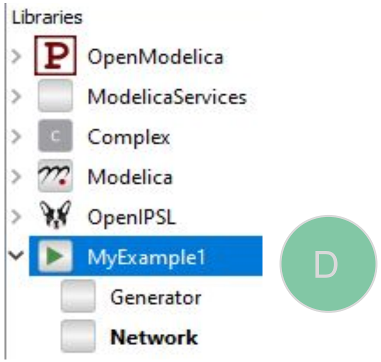
Yey! You just learned how to do inheritance graphically!

(C) We repeat the process to create a package called "Network"

(D) We now have a nice package structure like the one in the OpenIPSL.Examples.Tutorial.Example_1 package



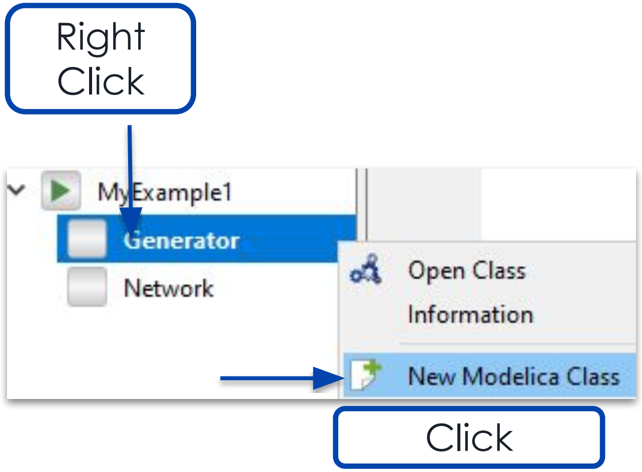
Name: Generator
Specialization: Model
Extends (optional): Modelica.Icons.Package
Insert in class (optional): MyExample1
☐ Partial
☐ Encapsulated
☐ State

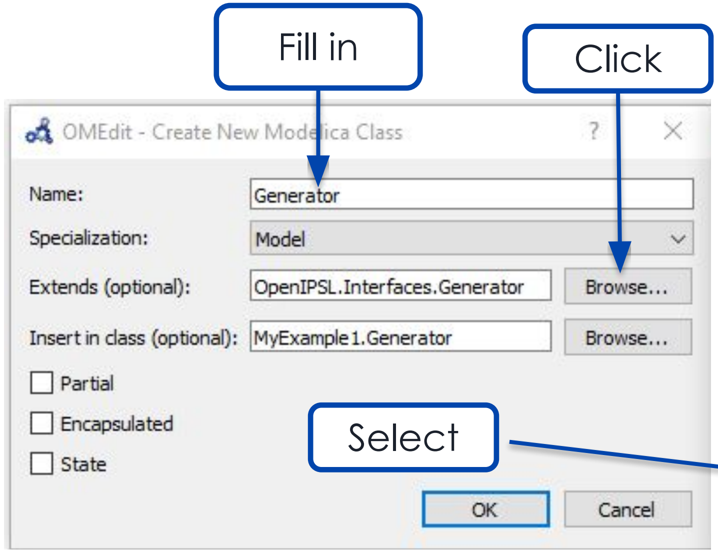


See solution in: OpenIPSL.Examples.Tutorial.Generator.Step_1

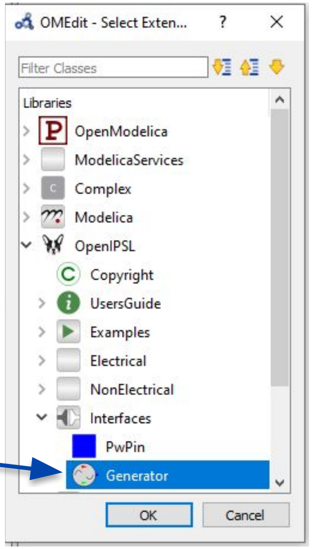
Example 1 – Generator Model and "**`extends`**"

- We now build a model for the power plant, which includes a synchronous machine and an excitation control system, we ignore the turbine and its controls in this example.
- Within the *Generator* sub-package all components related to the machine will be included within.
- We start by extending from the "Generator" interface from `OpenIPSL.Interfaces.Generator`





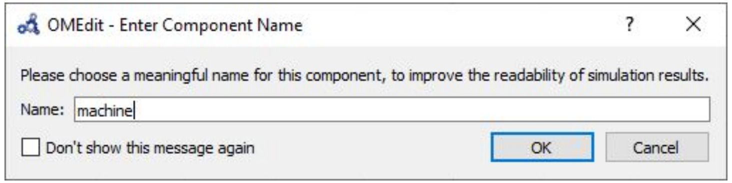
Name: Generator
Specialization: Model
Extends (optional): OpenIPSL.Interfaces.Generator
Insert in class (optional): MyExample1.Generator
☐ Partial
☐ Encapsulated
☐ State



See solution in: OpenIPSL.Examples.Tutorial.Generator.Step_1

Example 1 – Synchronous Machine Model

- We will use the 6th order model from PSAT: `OpenIPSL.Electrical.Machines.PSAT.Order6`
- Under the "Diagram View", drag the "Order6" model from the library and dropping it to the ***Generator*** sub-package that we just created
- Give the name "machine" to the component.

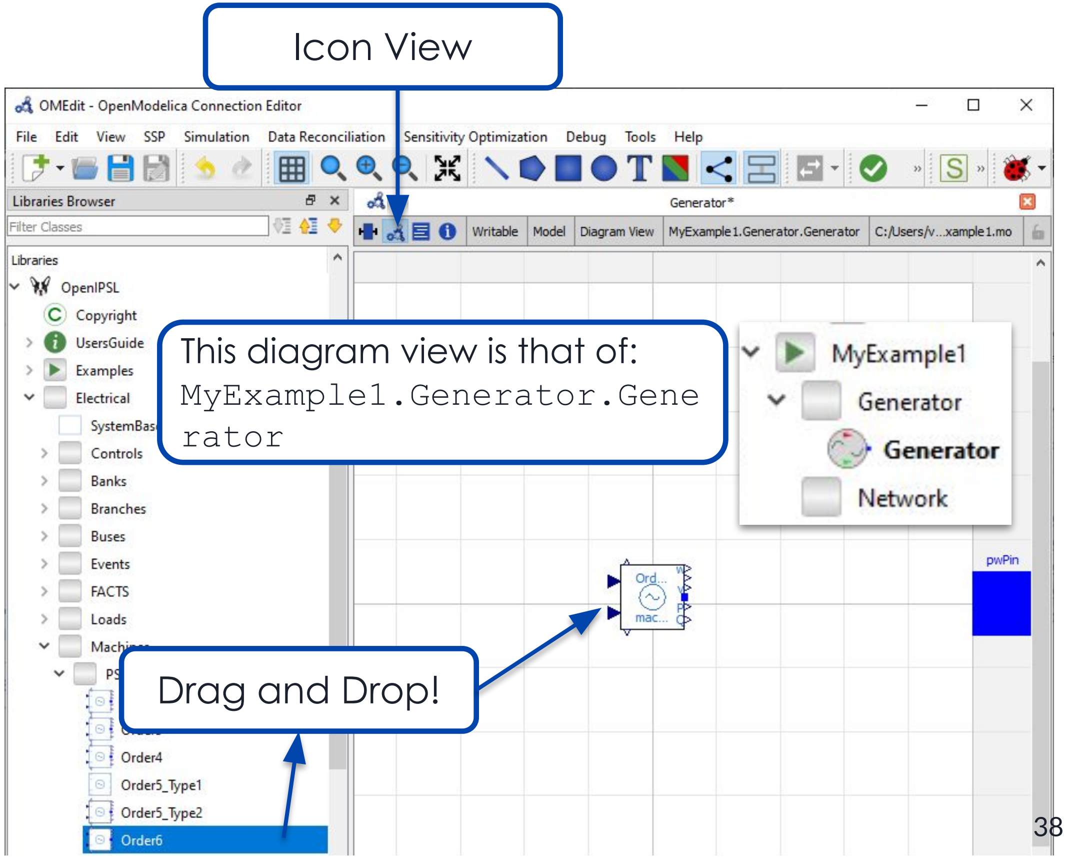

OMEdit - Enter Component Name

Please choose a meaningful name for this component, to improve the readability of simulation results.

Name: machine

☐ Don't show this message again

OK Cancel

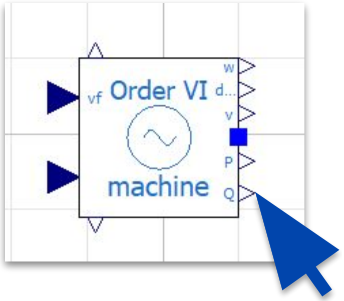


This diagram view is that of: `MyExample1.Generator.Generator`

See solution in: OpenIPSL.Examples.Tutorial.Generator.Step_1

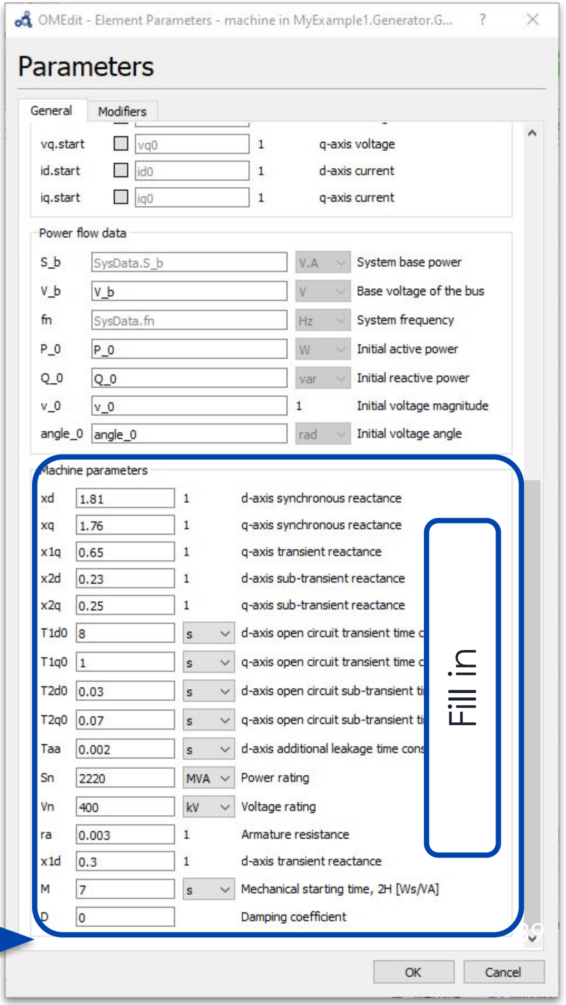
Example 1 – Synchronous Machine Model Parametrization

- Double click on the machine



- Parametrize only the highlighted area (for now) with the table on the right.

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


OMEdit - Element Parameters - machine in MyExample1.Generator.G...

Parameters

General | Modifiers

vq.start	vq0	1	q-axis voltage
id.start	id0	1	d-axis current
iq.start	iq0	1	q-axis current

Power flow data

S_b	SysData.S_b	V.A	System base power
V_b	V_b	V	Base voltage of the bus
fn	SysData.fn	Hz	System frequency
P_0	P_0	W	Initial active power
Q_0	Q_0	var	Initial reactive power
v_0	v_0	1	Initial voltage magnitude
angle_0	angle_0	rad	Initial voltage angle

Machine parameters

xd	1.81	1	d-axis synchronous reactance
xq	1.76	1	q-axis synchronous reactance
x1q	0.65	1	q-axis transient reactance
x2d	0.23	1	d-axis sub-transient reactance
x2q	0.25	1	q-axis sub-transient reactance
T1d0	8	s	d-axis open circuit transient time c
T1q0	1	s	q-axis open circuit transient time c
T2d0	0.03	s	d-axis open circuit sub-transient ti
T2q0	0.07	s	q-axis open circuit sub-transient ti
Taa	0.002	s	d-axis additional leakage time cons
Sn	2220	MVA	Power rating
Vn	400	kV	Voltage rating
ra	0.003	1	Armature resistance
x1d	0.3	1	d-axis transient reactance
M	7	s	Mechanical starting time, 2H [Ws/VA]
D	0		Damping coefficient

Fill in

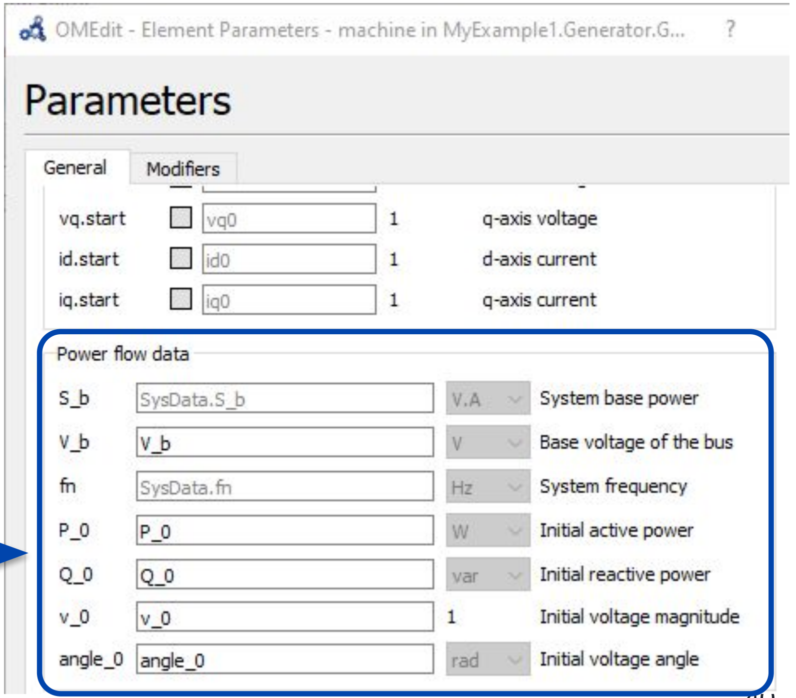
OK | Cancel

See solution in: OpenIPSL.Examples.Tutorial.Generator.Step_1

Example 1 – Synchronous Mach. Power Flow Input Data

- Next we learn to do parameter propagation.
- Although we already know the specific values from the power flow that we need to enter, we would like to be able to enter all of the data of the power flow of the "Generator" sub-system in a single window
  - In other words, we would like to propagate the parameters from the upper "Generator" layer, to each of the components inside the "Generator"
- Using the variables `(V_0, angle_0, etc.)` allows to propagate the parameters to the "upper layer" of the generator component.
- Notice that there are two greyed-out parameters, S_b and fn, as `SysData.S_b` and `SysData.fn`, don't modify them, we'll talk about the soon.

Fill in


OMEdit - Element Parameters - machine in MyExample1.Generator.G...

Parameters

General | Modifiers

vq.start	vq0	1	q-axis voltage
id.start	id0	1	d-axis current
iq.start	iq0	1	q-axis current

Power flow data

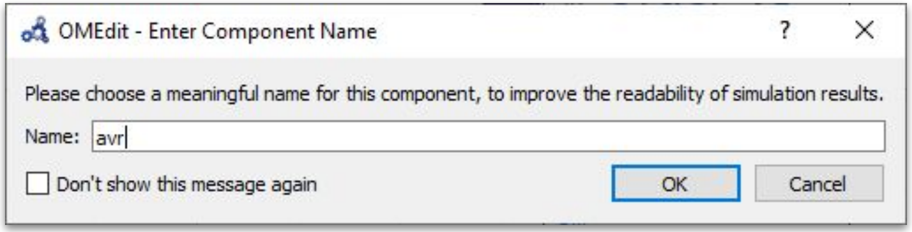
S_b	SysData.S_b	V.A	System base power
V_b	V_b	V	Base voltage of the bus
fn	SysData.fn	Hz	System frequency
P_0	P_0	W	Initial active power
Q_0	Q_0	var	Initial reactive power
v_0	v_0	1	Initial voltage magnitude
angle_0	angle_0	rad	Initial voltage angle

See solution in: OpenIPSL.Examples.Tutorial.Generator.Step_2

Example 1 – Excitation Control System Model

- The AC voltage at the terminals of a generator (stator) is controlled by varying the DC voltage of the field winding (rotor).
  - We model this through the so-called "Automatic Voltage Regulator" using **PSAT's AVR Type III model**, and parametrize it with the values in the table, drag and drop the component from and give it the name **`avr`**:

`OpenIPSL.Electrical.Controls.PSAT.AVR.AVRtypeIII`



OMEdit - Enter Component Name

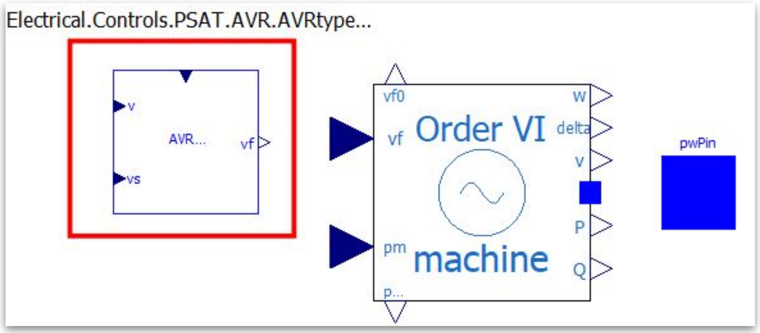
Please choose a meaningful name for this component, to improve the readability of simulation results.

Name: avr

☐ Don't show this message again

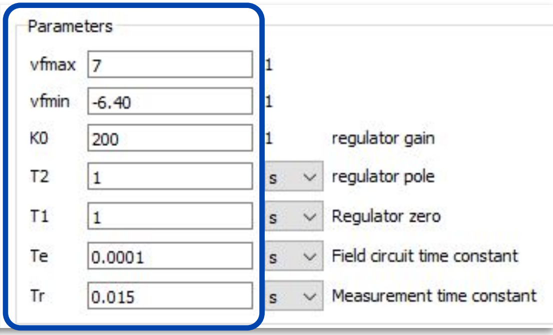
OK Cancel

  - Double click on the AVR model, and enter the parameter data in the table.



Fill in

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



Parameters

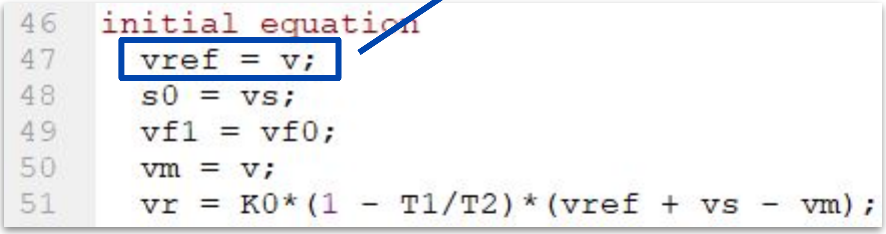
vfmax	7	1		
vfmin	-6.40	1		
K0	200	1		regulator gain
T2	1		s	regulator pole
T1	1		s	Regulator zero
Te	0.0001		s	Field circuit time constant
Tr	0.015		s	Measurement time constant

See solution in: OpenIPSL.Examples.Tutorial.Generator.Step_2

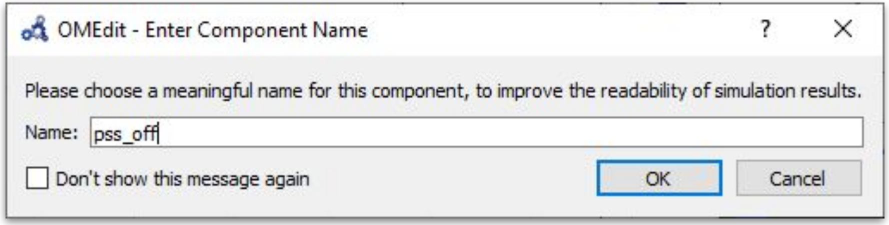
Example 1 – Set-point and External Input

- ***AVR Set-point:*** All control systems need a set-point, however, the value of the reference depends on the measured voltage and other variables.
  - In OpenIPSL, the set-point for all controllers, needs to during **during initialization** using the **initial equation** construct and receiving it as an input.
- ***External Input:*** AVRs have external input signal ports for different purposes, e.g. input from other sensors, other controls (e.g. PSS), etc.
  - In the Tutorial Example 2, this model is extended to add a PSS to the AVR, but for now, we will not model it.
  - A constant block from the MSL will be used as a zero, get that from `Modelica.Blocks.Sources.Constant` , set the value of k=0 and name it **`pss_off`**

The value of "v" is received as an input, it comes from the synchronous machine model as it has to correspond to the value of the required field voltage, we see in the next step how to interface them.



```
initial equation
  vref = v;
  s0 = vs;
  vf1 = vf0;
  vm = v;
  vr = K0*(1 - T1/T2)*(vref + vs - vm);
```

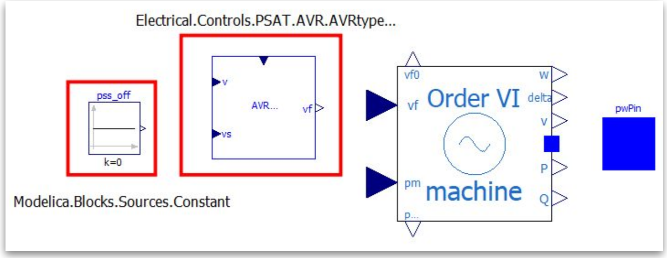


OMEdit - Enter Component Name

Please choose a meaningful name for this component, to improve the readability of simulation results.

Name: pss_off

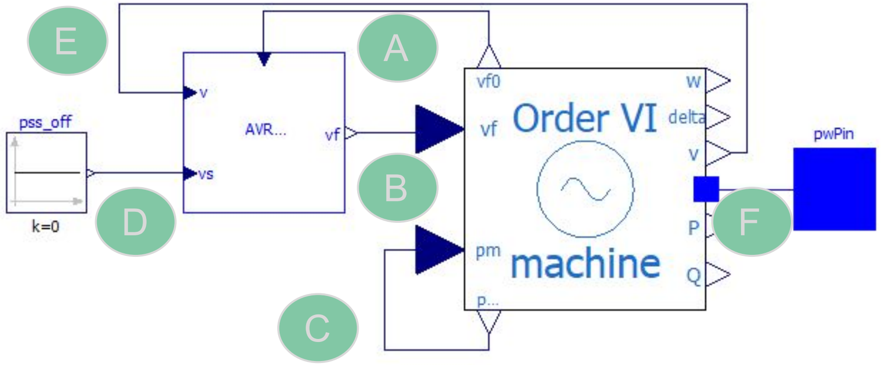
☐ Don't show this message again

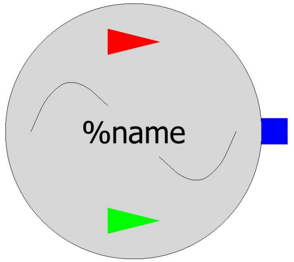

Electrical.Controls.PSAT.AVR.AVRtype...
pss_off
k=0
Modelica.Blocks.Sources.Constant
v
vs
AVR...
vf
vf0
vf
Order VI
machine
pm
p...
w
delta
v
P
Q
pwPin

See solution in: OpenIPSL.Examples.Tutorial.Generator.Step_3

Example 1 – Interfacing all components

- To finish the generator model, different signals need to be connected as shown on the figure on the right.
- Optionally, the "**icon**" generator model can be altered from the "Icon View" to change the visual appearance.



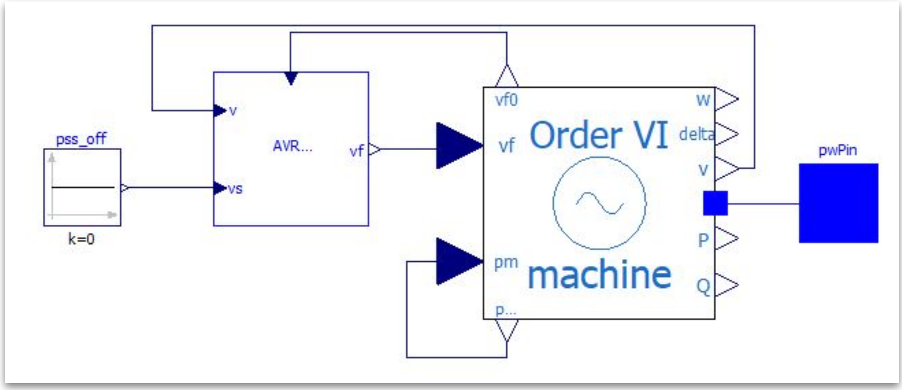


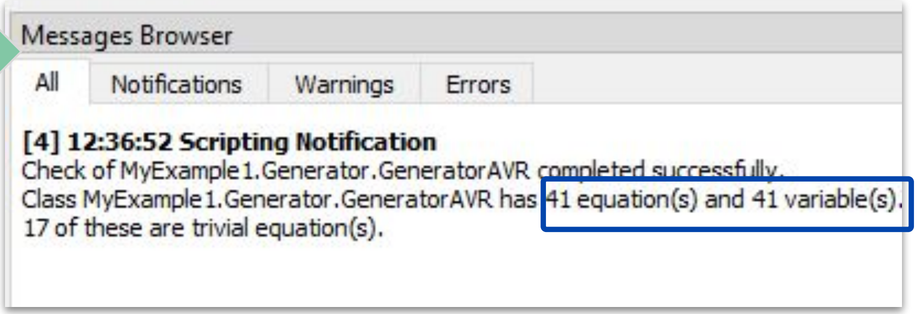
A.	1. Machine's terminal voltage to AVR's input signal
B.	2. AVR's output field voltage to machine's input field voltage
C.	3. Initially calculated mechanical power to input signal of the machine's mechanical power input pm.
D.	4. Constant pss_off to the PSS input at the AVR
E.	5. Initial generator field voltage to initial AVR field voltage.
F.	6. Generator pin to External pin

See solution in: OpenIPSL.Examples.Tutorial.Generator.Step_3

Example 1 – Check!

- Now we can see if the "Generator" sub-system has been assembled correctly.
- In Modelica-tools we use the operation `check(modelName)` to determine if the model is "balanced" (same number of equations as unknowns), and perform other checks about your model (e.g. missing or broken connections, etc).
- Right click on your model and then select the icon ✓
- If you see a similar message in the "Messages Browser" then that means the model has passed the check.



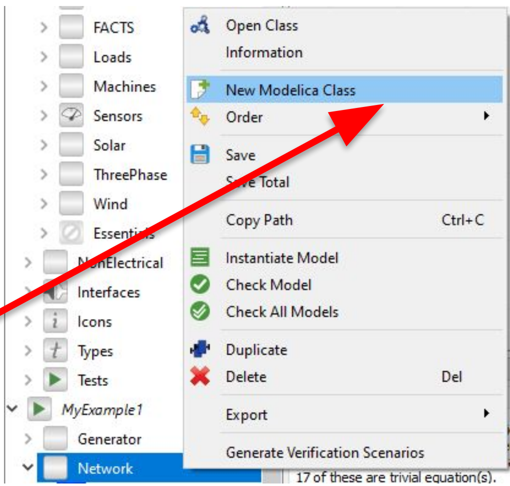


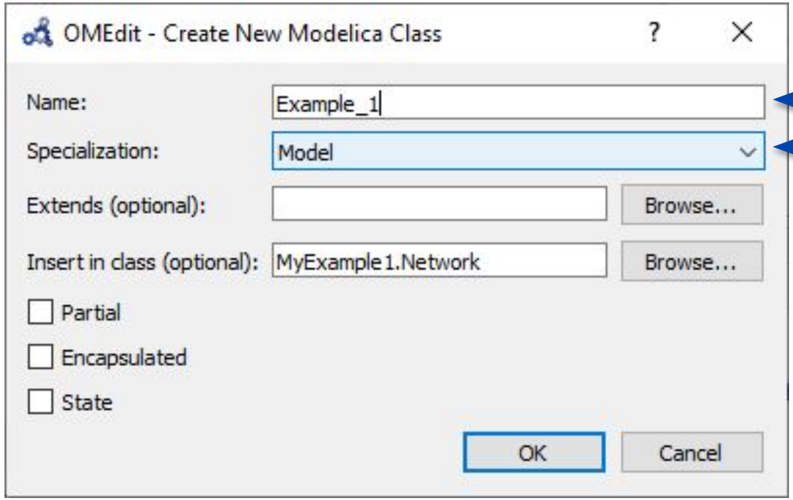
```
[4] 12:36:52 Scripting Notification
Check of MyExample1.Generator.GeneratorAVR completed successfully.
Class MyExample1.Generator.GeneratorAVR has 41 equation(s) and 41 variable(s).
17 of these are trivial equation(s).
```

See solution in: OpenIPSL.Examples.Tutorial.Network.Step_1

Example 1 – Network Model Setup

- The power network model will be implemented in the *Network* package we created.
- Right clicking on the *Network* package, and select "New Modelica Class"
- The name of the network model will be *"Example_1"* and you should select the specialization "Model".




OMEdit - Create New Modelica Class

Name: Example_1

Specialization: Model

Extends (optional):

Insert in class (optional): MyExample1.Network

- [ ] Partial
- [ ] Encapsulated
- [ ] State

Fill in: "Example_1"

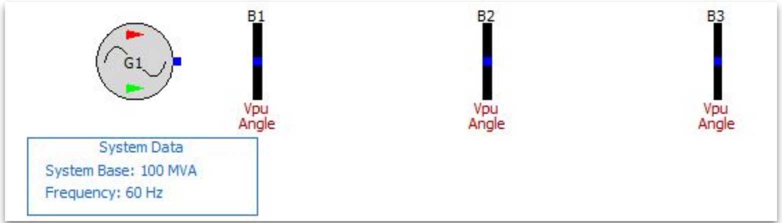
See solution in: OpenIPSL.Examples.Tutorial.Network.Step_1

Example 1 – Network Components

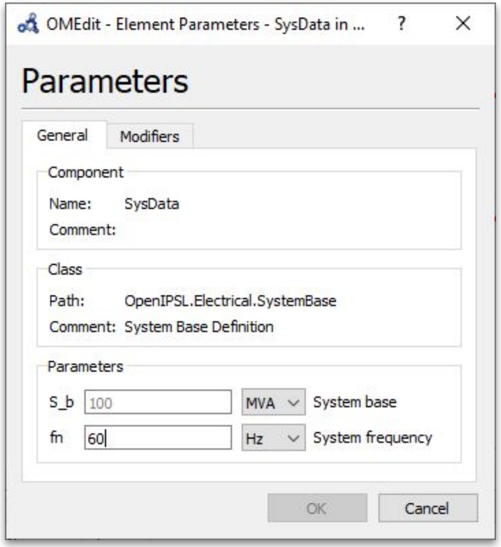
- Drag and drop the generator model that we created and name it G1.
- Drag and drop a bus model three times, and name them B1, B2 and B3; use: `OpenIPSL.Electrical.Buses.Bus`
- Drag and drop a system data block from: OpenIPSL.Electrical.SystemBase and name it SysData
    - Double click on SysData and enter: fn = 60.
    - Look at the text view, and you will find the following statement:

```
inner OpenIPSL.Electrical.SystemBase SysData(fn = 60)
```

-
    - This will allow all the components that use variables `S_b` and `fn` to access the value defined in the top layer of the model.
    - Note that `S_b` is not in parenthesis, this is because we will use the default value of 100 MVA, so we don't need to modify it.



System Data
System Base: 100 MVA
Frequency: 60 Hz


OMEdit - Element Parameters - SysData in ...

Parameters

General | Modifiers

Component
Name: SysData
Comment:

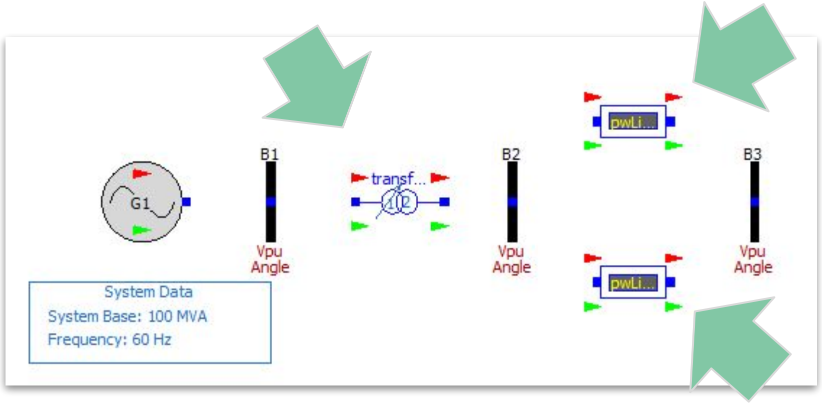
Class
Path: OpenIPSL.Electrical.SystemBase
Comment: System Base Definition

Parameters
S_b 100 MVA System base
fn 60 Hz System frequency

See solution in: OpenIPSL.Examples.Tutorial.Network.Step_2

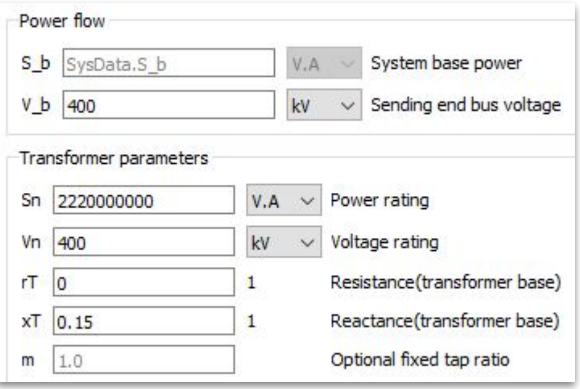
Example 1 – Network Branches

- Add the following transmission lines and transformer models (leave default names):
  - `OpenIPSL.Electrical.Branches.PSAT.TwoWindingTransformer`
  - `OpenIPSL.Electrical.Branches.PwLine`
- Parametrize the new components:



System Data
System Base: 100 MVA
Frequency: 60 Hz

Transformer



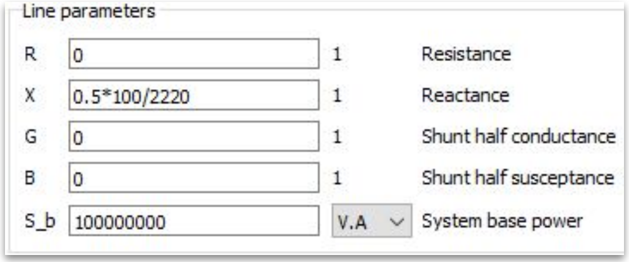
Power flow

S_b	SysData.S_b	V.A	System base power
V_b	400	kV	Sending end bus voltage

Transformer parameters

Sn	2220000000	V.A	Power rating
Vn	400	kV	Voltage rating
rT	0	1	Resistance(transformer base)
xT	0.15	1	Reactance(transformer base)
m	1.0		Optional fixed tap ratio

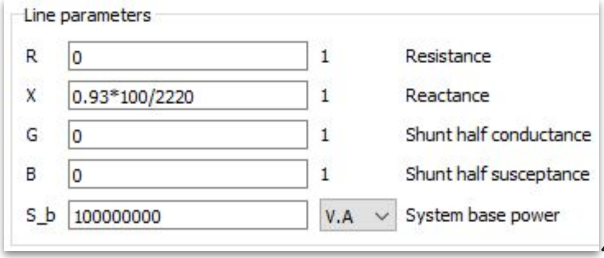
Line 1



Line parameters

R	0	1	Resistance
X	0.5*100/2220	1	Reactance
G	0	1	Shunt half conductance
B	0	1	Shunt half susceptance
S_b	100000000	V.A	System base power

Line 2



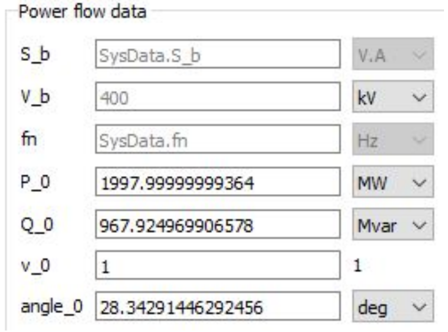
Line parameters

R	0	1	Resistance
X	0.93*100/2220	1	Reactance
G	0	1	Shunt half conductance
B	0	1	Shunt half susceptance
S_b	100000000	V.A	System base power

Example 1 – Network External Grid and G1 Dispatched Power

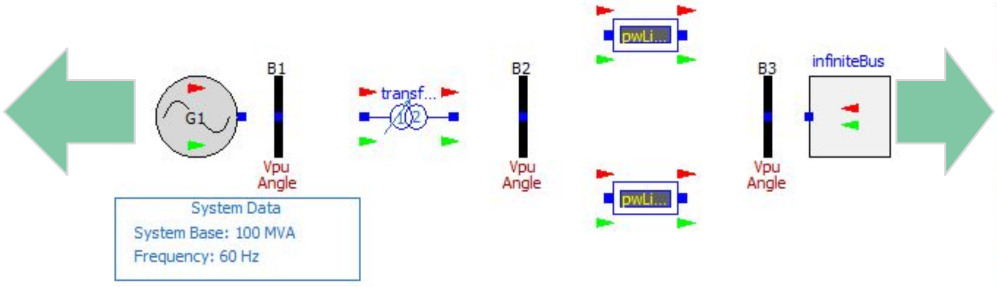
- It is quite common to use an “Infinite Bus” to model the connection of a plant to the rest of the power transmission network.
  - Drag and drop the ”Infinite Bus” model from `OpenIPSL.Electrical.Buses.InfiniteBus` and call it `infiniteBus`
- Parametrize **both** the generator and infinite bus using the power flow data!

G1

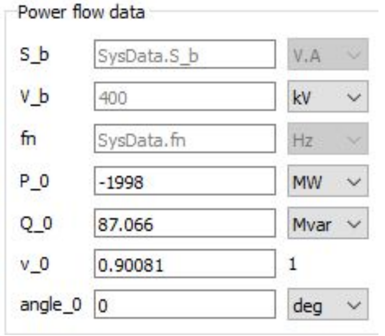


Power flow data

S_b	SysData.S_b	V.A
V_b	400	kV
fn	SysData.fn	Hz
P_0	1997.99999999364	MW
Q_0	967.924969906578	Mvar
v_0	1	1
angle_0	28.34291446292456	deg



Infinite bus



Power flow data

S_b	SysData.S_b	V.A
V_b	400	kV
fn	SysData.fn	Hz
P_0	-1998	MW
Q_0	87.066	Mvar
v_0	0.90081	1
angle_0	0	deg

Note: such type of power flow would be unfeasible in an actual network. Observe the voltage magnitude and powers – this is truly a textbook example.

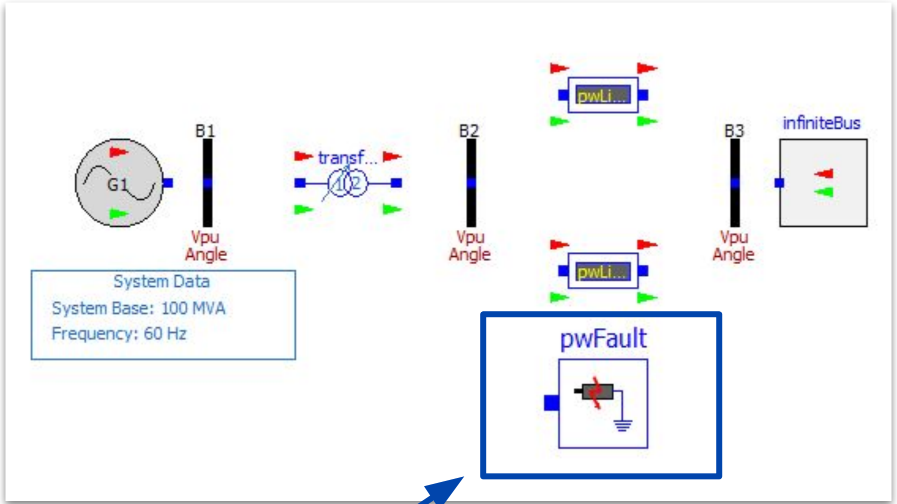
See solution in: OpenIPSL.Examples.Tutorial.Network.Step_4

Example 1 – Fault Event

- We would like to simulate how the system behaves when exposed to a large disturbance.
- The disturbances that are most critical for power networks are 3-phase-to-ground faults.
- To model this, we add the block from

```
OpenIPSL.Electrical.Events.PwFault
```

- Parametrize the fault as follows

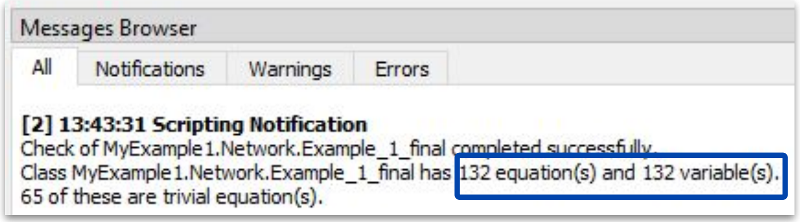


R	0	t1	0.5
X	0.01*100/2220	t2	0.57

See solution in: OpenIPSL.Examples.Tutorial.Network.Step_5

Example 1 – **The Final Model!**

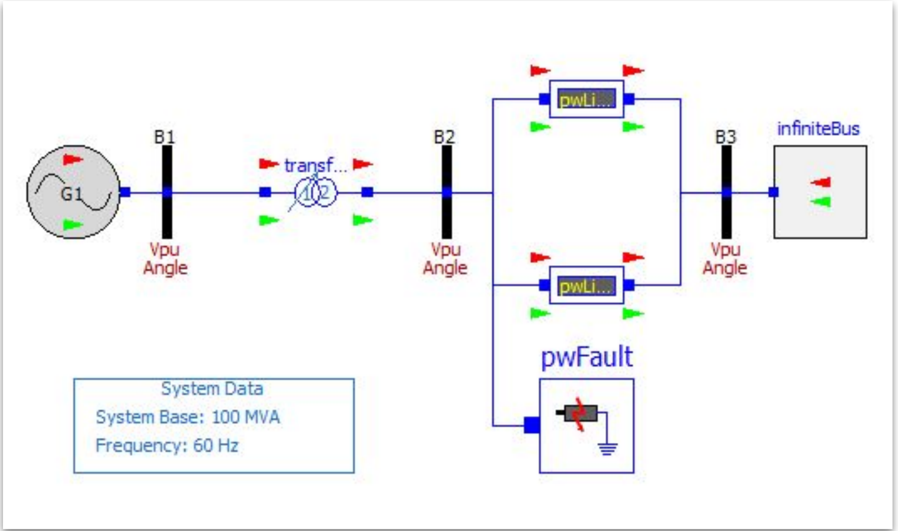
- The network model is completed by connecting all of the components together as shown in the diagram.
- Before simulating, we should "check" the model, you should get something similar if the model is balanced.
- Right click on your model and then the check icon ✓
- If you see a similar message in the "Messages Browser" then that means the model has passed the check.



Messages Browser

All | Notifications | Warnings | Errors

```
[2] 13:43:31 Scripting Notification
Check of MyExample1.Network.Example_1_final completed successfully.
Class MyExample1.Network.Example_1_final has 132 equation(s) and 132 variable(s).
65 of these are trivial equation(s).
```

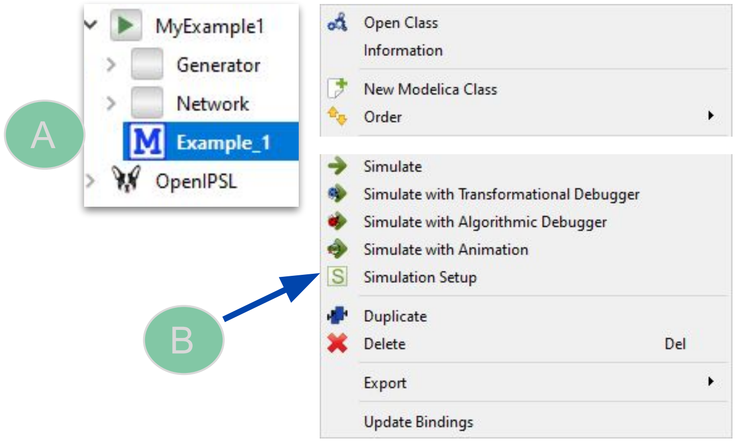


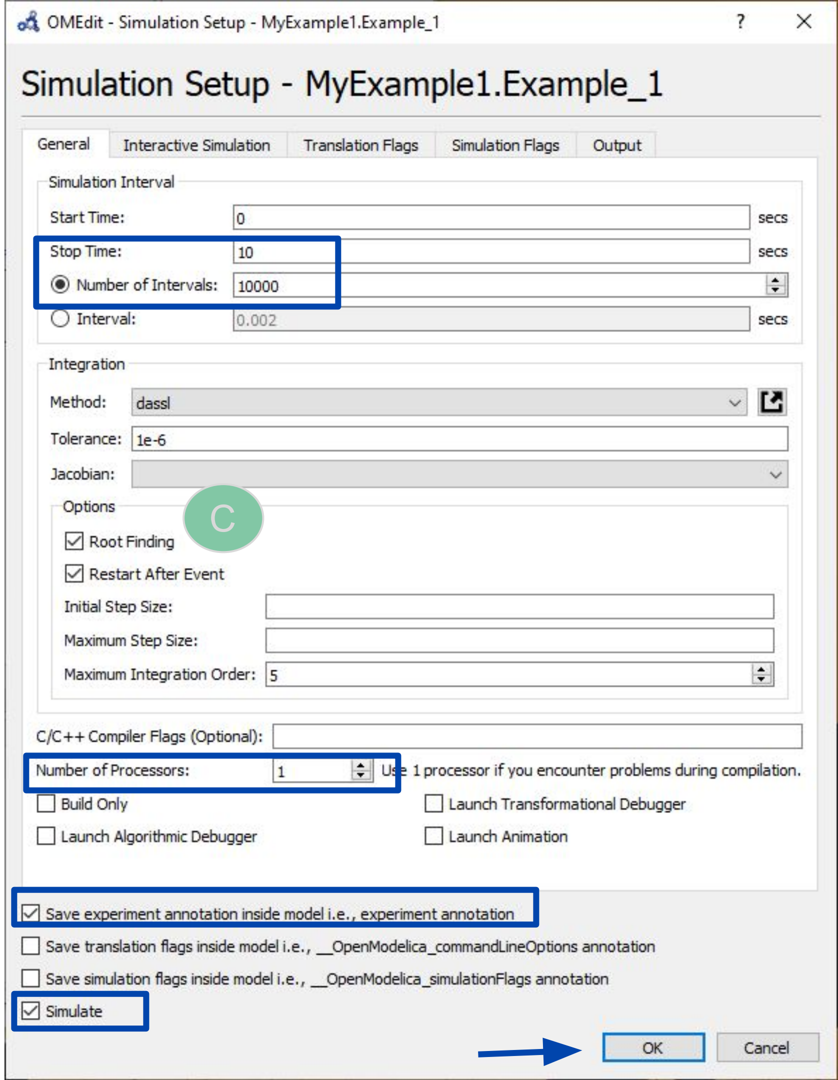
Example 1
Let’s simulate!

(A) Double click on the “Example_1” model.

(B) Next, right click on the model and click on the simulation setup icon: S

(C) Configure the solver as shown in the window on the right and then click on “OK”




OMEdit - Simulation Setup - MyExample1.Example_1

Simulation Setup - MyExample1.Example_1

General | Interactive Simulation | Translation Flags | Simulation Flags | Output

Simulation Interval

Start Time:	0	secs
Stop Time:	10	secs
● Number of Intervals:	10000	
○ Interval:	0.002	secs

Integration

Method:	dassl
Tolerance:	1e-6
Jacobian:	

Options

[✓] Root Finding

[✓] Restart After Event

Initial Step Size:	
Maximum Step Size:	
Maximum Integration Order:	5

C/C++ Compiler Flags (Optional):

Number of Processors: 1 Use 1 processor if you encounter problems during compilation.

[ ] Build Only [ ] Launch Transformational Debugger

[ ] Launch Algorithmic Debugger [ ] Launch Animation

[✓] Save experiment annotation inside model i.e., experiment annotation

[ ] Save translation flags inside model i.e., __OpenModelica_commandLineOptions annotation

[ ] Save simulation flags inside model i.e., __OpenModelica_simulationFlags annotation

[✓] Simulate

Example 1
Let’s plot!

(A) After clicking “OK”, OpenModelica entered the plotting mode.

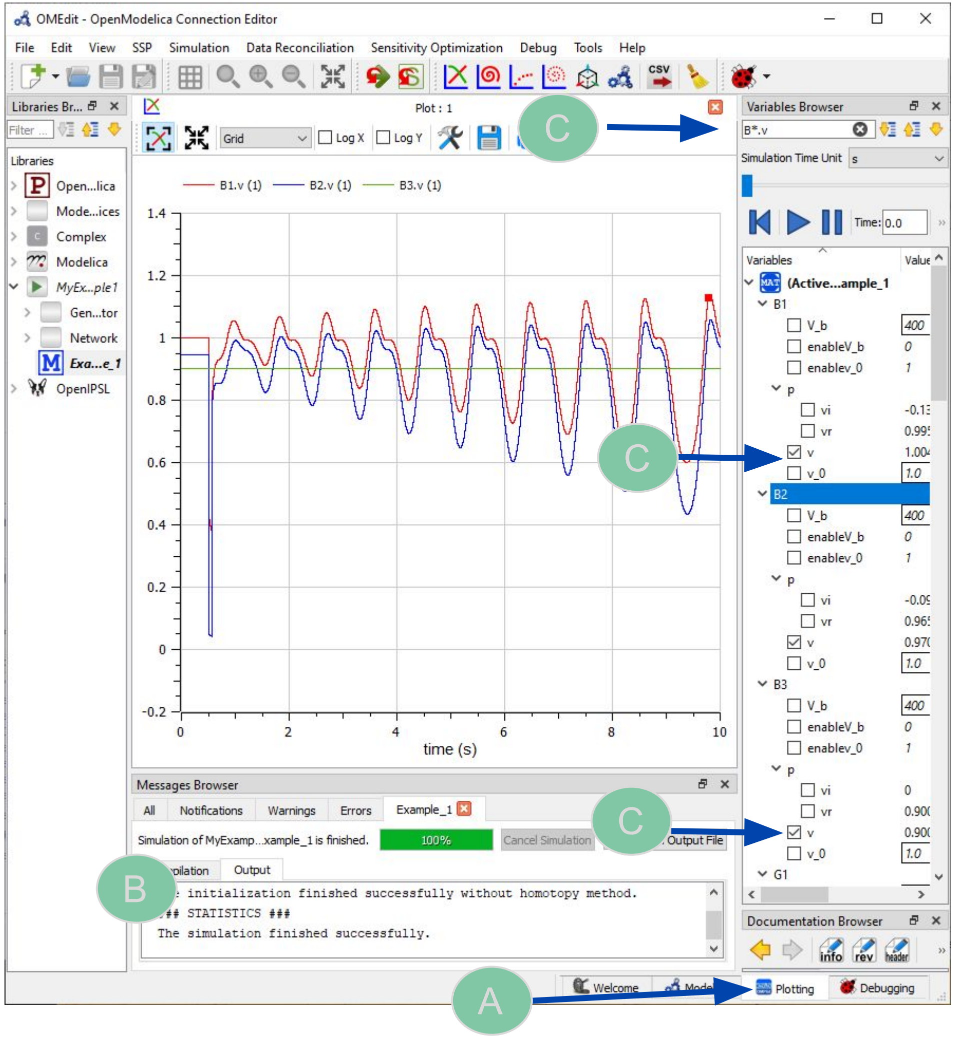
(B) If the simulation was successful, the “Messages Browser” will indicate that:

```
The simulation finished successfully.
```

(C) You can now use wildcards to search for specific variables to plot.

- In the “`Variables Browser`” you can enter “`B*.v`” to search for all variables that contain the structure.
- Select `B1.v`, `B2.v`, and `B3.v`, these correspond to the generator bus, the high side of the transformer bus and the infinite bus voltage.

This example shows that the model is unstable when the fault is applied, this is the behavior that we wanted to reproduce.



Simulation of MyExamp...xample_1 is finished. 100%

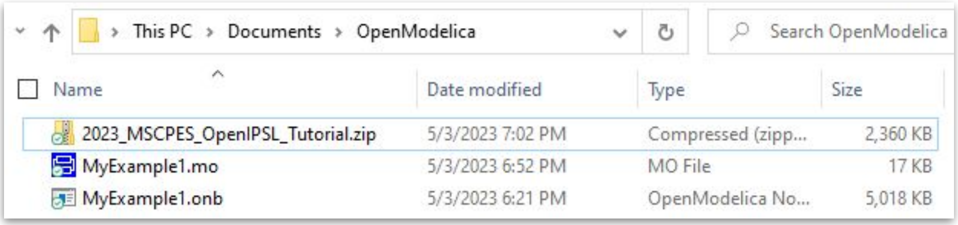
```
...: initialization finished successfully without homotopy method.
### STATISTICS ###
The simulation finished successfully.
```

Content

Using OMNotebook for Interactive Analysis

- The OpenModelica installation includes a very handy tool called OMNotebook.
- OMNotebook allows you to interact with the OpenModelica Compiler (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).**
    - Similar to Jupyter Notebooks or Mathematica Notebooks.
- We use OMNotebook next to load the model developed, and simulate it interactively.

Open the Notebook

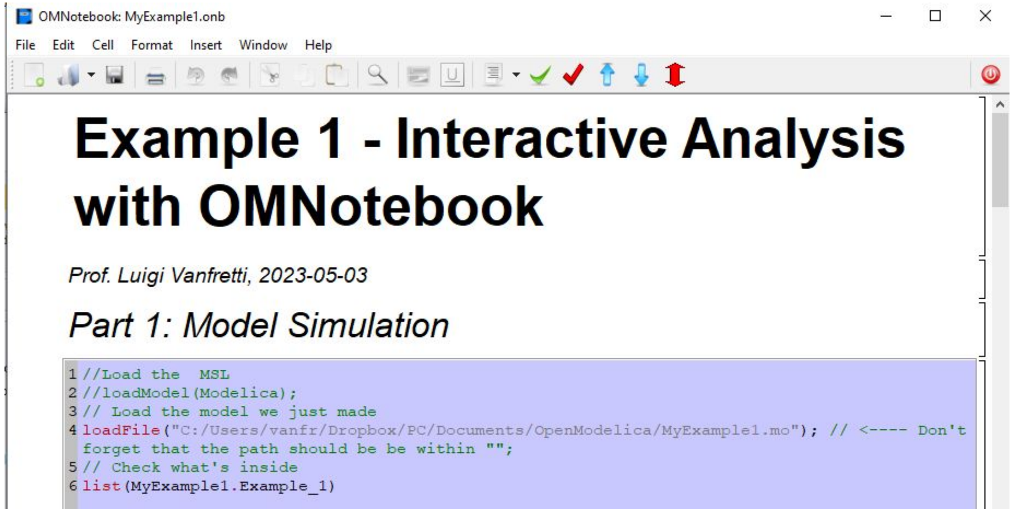


This PC › Documents › OpenModelica

Search OpenModelica

Name	Date modified	Type	Size
2023_MSCPES_OpenIPSL_Tutorial.zip	5/3/2023 7:02 PM	Compressed (zipp...	2,360 KB
MyExample1.mo	5/3/2023 6:52 PM	MO File	17 KB
MyExample1.onb	5/3/2023 6:21 PM	OpenModelica No...	5,018 KB

- Uncompress the files in the .zip file distributed for this tutorial in a directory that you can find easily (e.g., “Documents/OpenModelica”)
- The OpenModelica Notebook file you need is called: `MyExample1.onb`
    - Double click it to open it using OMNotebook.
    - The OMN window will appear as shown below.



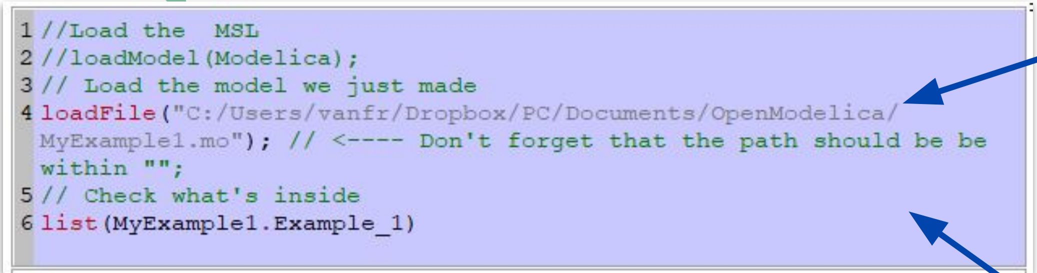
Example 1 - Interactive Analysis with OMNotebook

Prof. Luigi Vanfretti, 2023-05-03

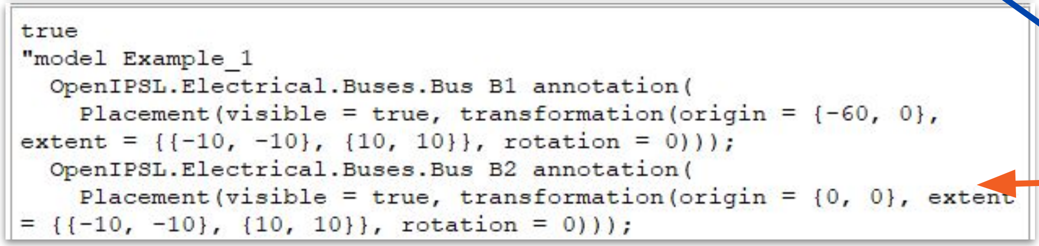
Part 1: Model Simulation

```
//Load the  MSL
//loadModel(Modelica);
// Load the model we just made
loadFile("C:/Users/vanfr/Dropbox/PC/Documents/OpenModelica/MyExample1.mo"); // <---- Don't forget that the path should be be within "";
// Check what's inside
list(MyExample1.Example_1)
```

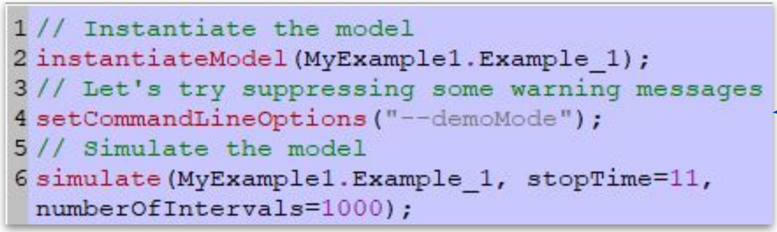
Set-up your path & evaluate cells!



```
//Load the  MSL
//loadModel(Modelica);
// Load the model we just made
loadFile("C:/Users/vanfr/Dropbox/PC/Documents/OpenModelica/
MyExample1.mo"); // <---- Don't forget that the path should be be
within "";
// Check what's inside
list(MyExample1.Example_1)
```



```
true
"model Example_1
  OpenIPSL.Electrical.Buses.Bus B1 annotation(
    Placement(visible = true, transformation(origin = {-60, 0},
extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  OpenIPSL.Electrical.Buses.Bus B2 annotation(
    Placement(visible = true, transformation(origin = {0, 0}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
```



```
// Instantiate the model
instantiateModel(MyExample1.Example_1);
// Let's try suppressing some warning messages
setCommandLineOptions("--demoMode");
// Simulate the model
simulate(MyExample1.Example_1, stopTime=11,
numberOfIntervals=1000);
```

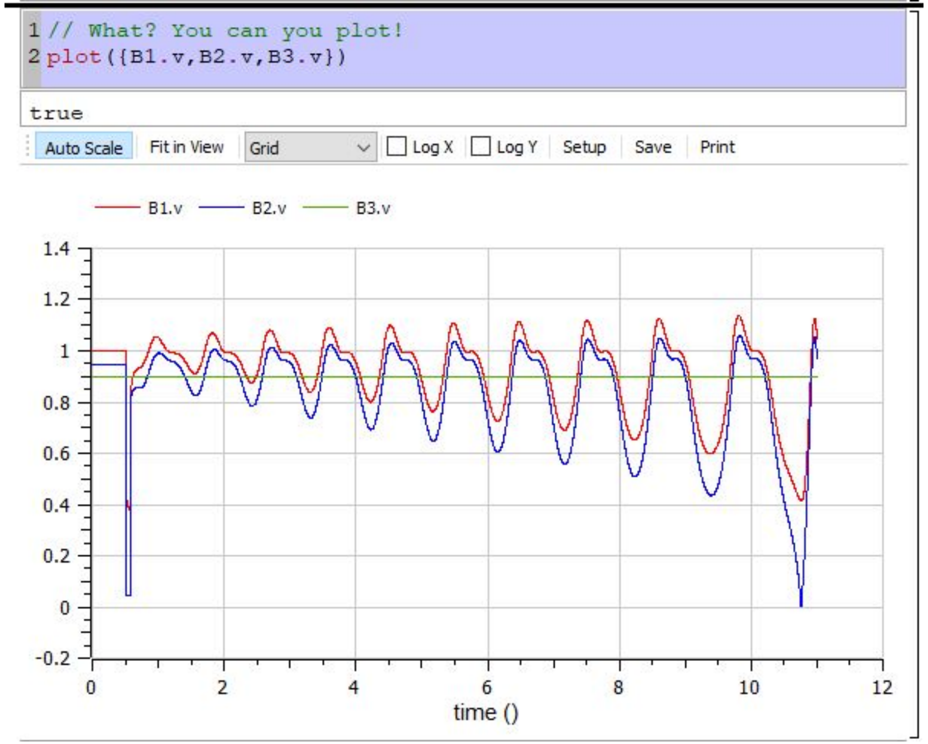
Set your path to where "MyExample1.mo" is located.

Evaluate each cell using "Shift+Enter"

This should give you as output the model's contents.

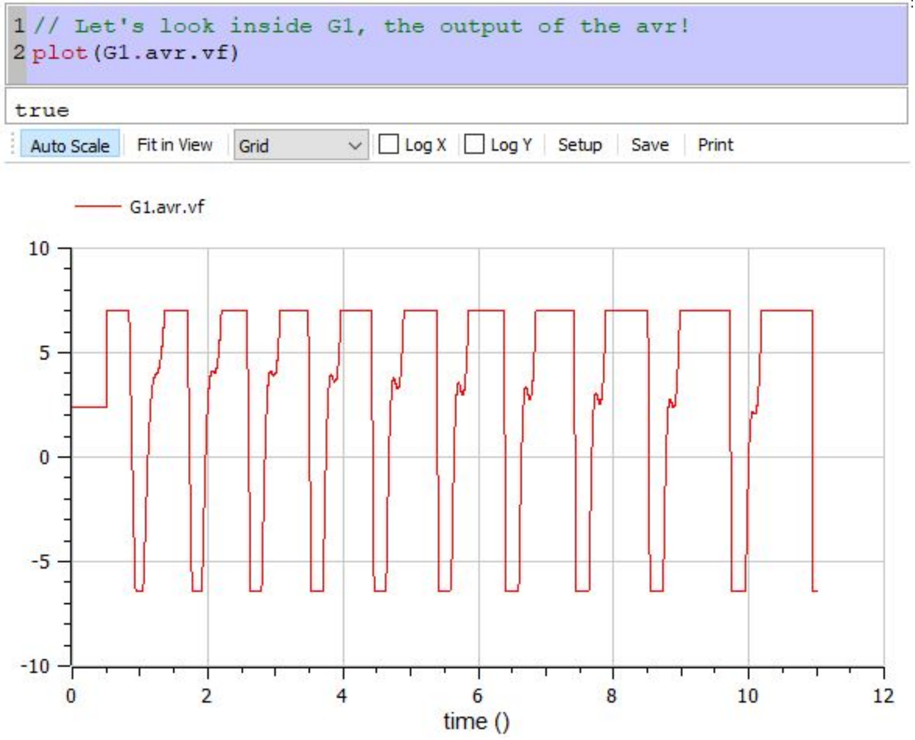
Instantiate and Simulate the Model!

Plotting Results



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

```
true
```

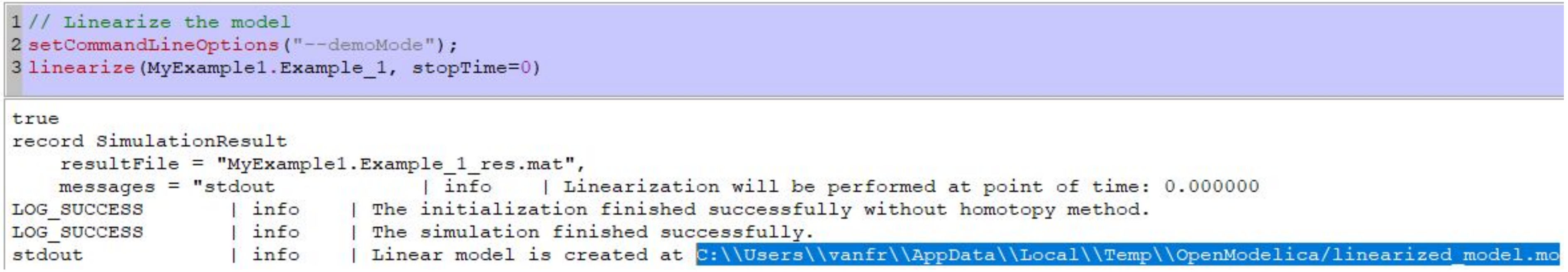


```
// Let's look inside G1, the output of the avr!
plot(G1.avr.vf)
```

```
true
```

Linear Model

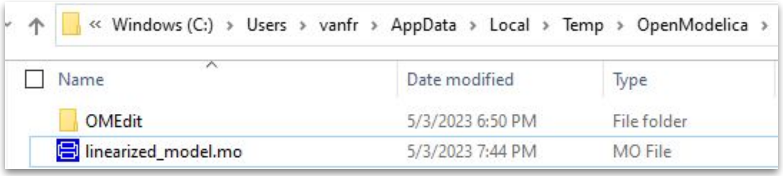
- Let's now do a bit more analysis on the system's stability by using linear analysis.
- This will allow us to determine the system's damping with the current value of the AVR's gain (K=200).



```
// Linearize the model
setCommandLineOptions("--demoMode");
linearize(MyExample1.Example_1, stopTime=0)
```

```
true
record SimulationResult
    resultFile = "MyExample1.Example_1_res.mat",
    messages = "stdout            | info    | Linearization will be performed at point of time: 0.000000
LOG_SUCCESS       | info    | The initialization finished successfully without homotopy method.
LOG_SUCCESS       | info    | The simulation finished successfully.
stdout            | info    | Linear model is created at C:\\Users\\vanfr\\AppData\\Local\\Temp\\OpenModelica/linearized_model.mo
```

- After evaluating the cell, you can see the model is linearized successfully.
- OpenModelica will provide you with not only the linear model's matrices, but also with a linear model called: `linearized_model.mo`
- This can be found in the temporary directory created by OM & can be opened in OMEdit too.

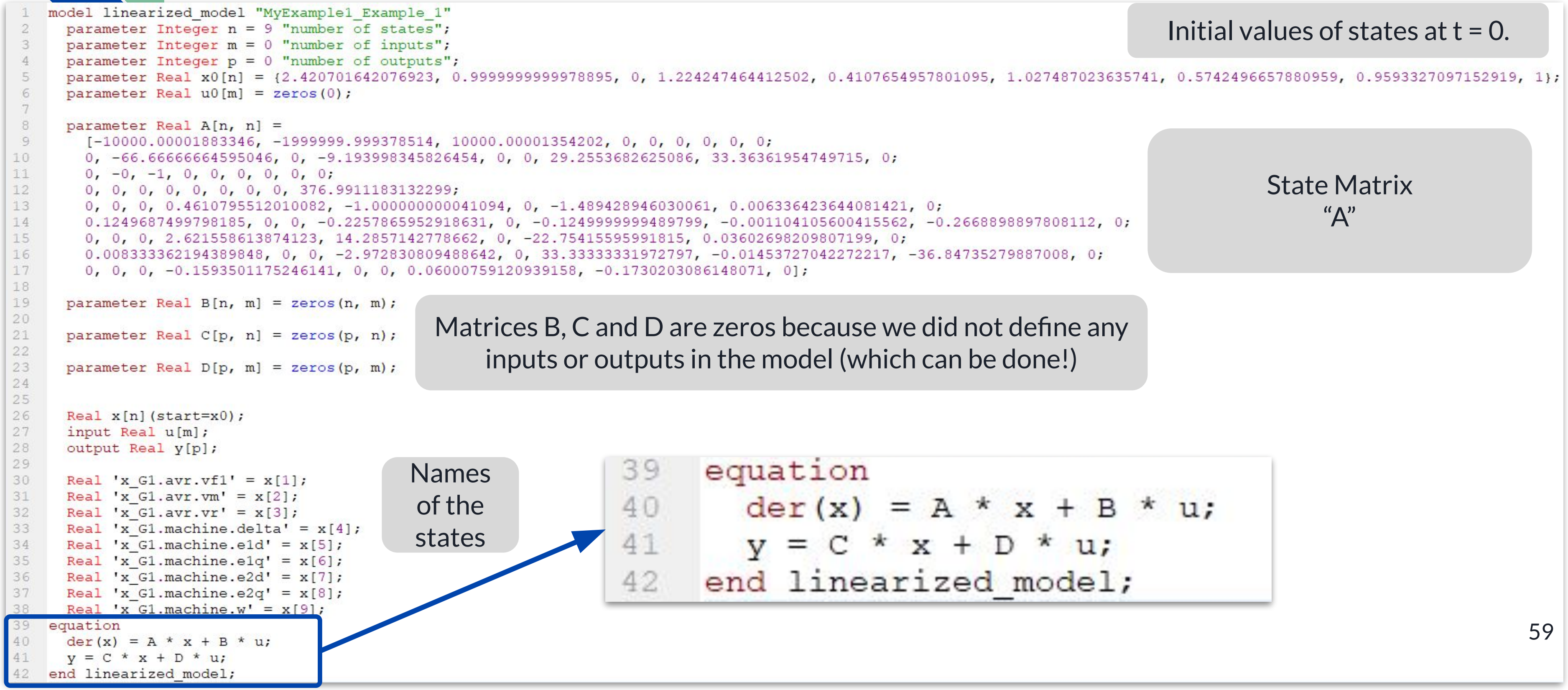


« Windows (C:) › Users › vanfr › AppData › Local › Temp › OpenModelica ›

Name	Date modified	Type
OMEdit	5/3/2023 6:50 PM	File folder
linearized_model.mo	5/3/2023 7:44 PM	MO File

Linear Model:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$



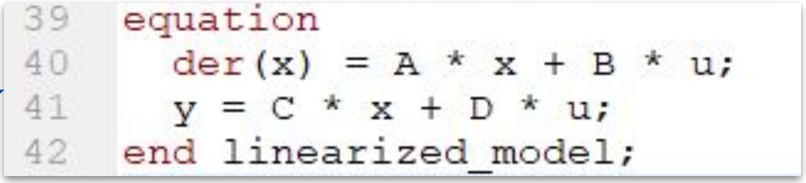
```
model linearized_model "MyExample1_Example_1"
  parameter Integer n = 9 "number of states";
  parameter Integer m = 0 "number of inputs";
  parameter Integer p = 0 "number of outputs";
  parameter Real x0[n] = {2.420701642076923, 0.9999999999978895, 0, 1.224247464412502, 0.4107654957801095, 1.027487023635741, 0.5742496657880959, 0.9593327097152919, 1};
  parameter Real u0[m] = zeros(0);

  parameter Real A[n, n] =
    [-10000.00001883346, -1999999.999378514, 10000.00001354202, 0, 0, 0, 0, 0, 0;
    0, -66.66666664595046, 0, -9.193998345826454, 0, 0, 29.2553682625086, 33.36361954749715, 0;
    0, -0, -1, 0, 0, 0, 0, 0, 0;
    0, 0, 0, 0, 0, 0, 0, 0, 376.9911183132299;
    0, 0, 0, 0.4610795512010082, -1.000000000041094, 0, -1.489428946030061, 0.006336423644081421, 0;
    0.1249687499798185, 0, 0, -0.2257865952918631, 0, -0.1249999999489799, -0.001104105600415562, -0.2668898897808112, 0;
    0, 0, 0, 2.621558613874123, 14.2857142778662, 0, -22.75415595991815, 0.03602698209807199, 0;
    0.008333362194389848, 0, 0, -2.972830809488642, 0, 33.33333331972797, -0.01453727042272217, -36.84735279887008, 0;
    0, 0, 0, -0.1593501175246141, 0, 0, 0.06000759120939158, -0.1730203086148071, 0];

  parameter Real B[n, m] = zeros(n, m);

  parameter Real C[p, n] = zeros(p, n);

  parameter Real D[p, m] = zeros(p, m);


  Real x[n](start=x0);
  input Real u[m];
  output Real y[p];

  Real 'x_G1.avr.vf1' = x[1];
  Real 'x_G1.avr.vm' = x[2];
  Real 'x_G1.avr.vr' = x[3];
  Real 'x_G1.machine.delta' = x[4];
  Real 'x_G1.machine.e1d' = x[5];
  Real 'x_G1.machine.e1q' = x[6];
  Real 'x_G1.machine.e2d' = x[7];
  Real 'x_G1.machine.e2q' = x[8];
  Real 'x_G1.machine.w' = x[9];
equation
  der(x) = A * x + B * u;
  y = C * x + D * u;
end linearized_model;
```

Initial values of states at t = 0.

State Matrix
"A"

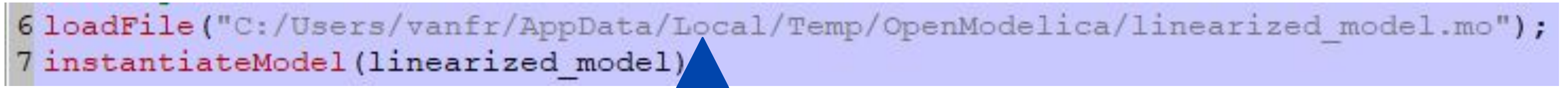
Matrices B, C and D are zeros because we did not define any inputs or outputs in the model (which can be done!)

Names of the states



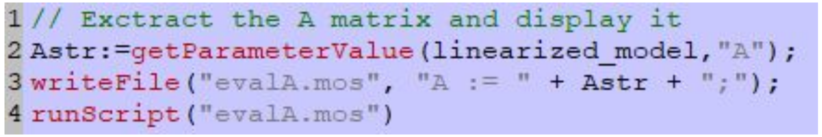
```
equation
  der(x) = A * x + B * u;
  y = C * x + D * u;
end linearized_model;
```

Linear Analysis

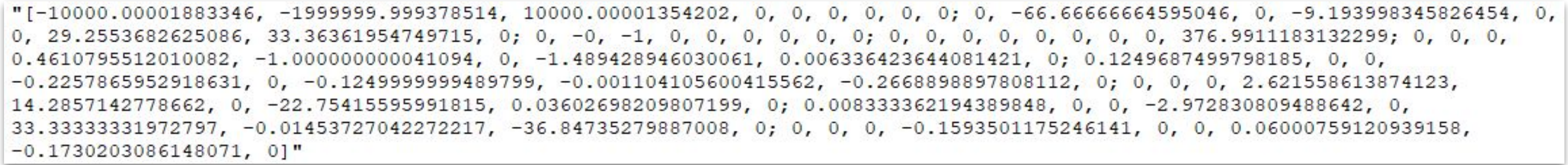


```
loadFile("C:/Users/vanfr/AppData/Local/Temp/OpenModelica/linearized_model.mo");
instantiateModel(linearized_model)
```

- We now load and instantiate the linear model:
- Then we extract the A-matrix and display it:

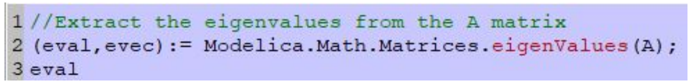


```
// Exctract the A matrix and display it
Astr:=getParameterValue(linearized_model,"A");
writeFile("evalA.mos", "A := " + Astr + ";");
runScript("evalA.mos")
```

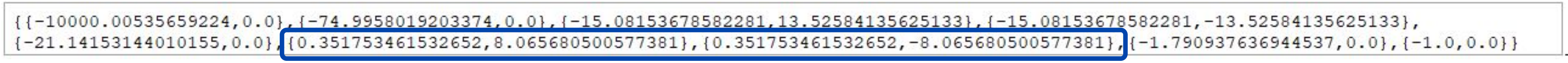


```
"[-10000.00001883346, -1999999.999378514, 10000.00001354202, 0, 0, 0, 0, 0, 0; 0, -66.66666664595046, 0, -9.193998345826454, 0,
0, 29.2553682625086, 33.36361954749715, 0; 0, -0, -1, 0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0, 376.9911183132299; 0, 0, 0,
0.4610795512010082, -1.000000000041094, 0, -1.489428946030061, 0.006336423644081421, 0; 0.1249687499798185, 0, 0,
-0.2257865952918631, 0, -0.1249999999489799, -0.001104105600415562, -0.2668898897808112, 0; 0, 0, 0, 2.621558613874123,
14.2857142778662, 0, -22.75415595991815, 0.03602698209807199, 0; 0.008333362194389848, 0, 0, -2.972830809488642, 0,
33.33333331972797, -0.01453727042272217, -36.84735279887008, 0; 0, 0, 0, -0.1593501175246141, 0, 0, 0.06000759120939158,
-0.1730203086148071, 0]"
```

- Now we can extract the eigenvalues:



```
//Extract the eigenvalues from the A matrix
(eval,evec):= Modelica.Math.Matrices.eigenValues(A);
eval
```



```
{{-10000.00535659224,0.0},{-74.9958019203374,0.0},{-15.08153678582281,13.52584135625133},{-15.08153678582281,-13.52584135625133},
{-21.14153144010155,0.0},{0.351753461532652,8.065680500577381},{0.351753461532652,-8.065680500577381},{-1.790937636944537,0.0},{-1.0,0.0}}
```

This is the complex unstable mode we were looking for!

Parametric Sweep

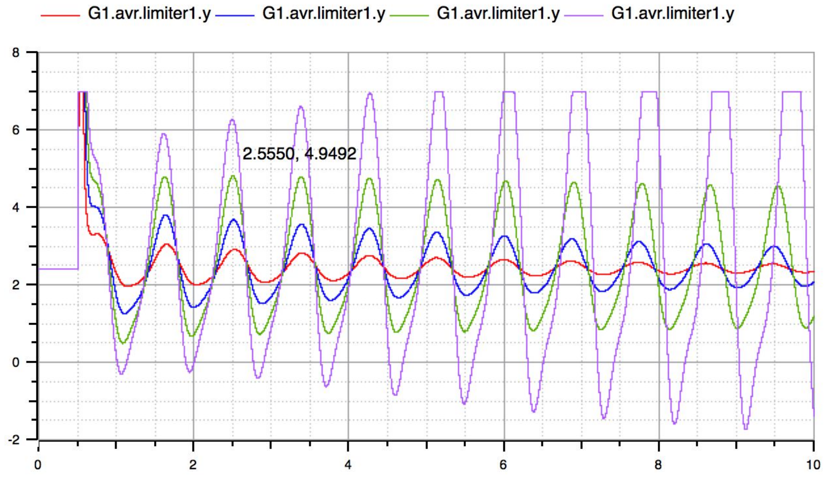
- Using OMNotebook, we can interact with the OMC in order to analyze the behavior of the system.

- 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!

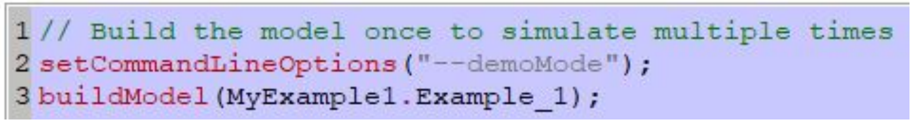
- A parametric sweep is used in OMC to simulate the system's response for different values of the gain of the AVR, namely, K0.

Results from K=10 to K=50

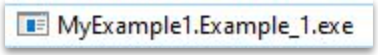


Parametric Sweep

- When performing several simulations that only require a parameter change, “building” the model will avoid generating new C-code for every new parameter value.
    - This creates an executable called:



```
// Build the model once to simulate multiple times
setCommandLineOptions("--demoMode");
buildModel(MyExample1.Example_1);
```



MyExample1.Example_1.exe

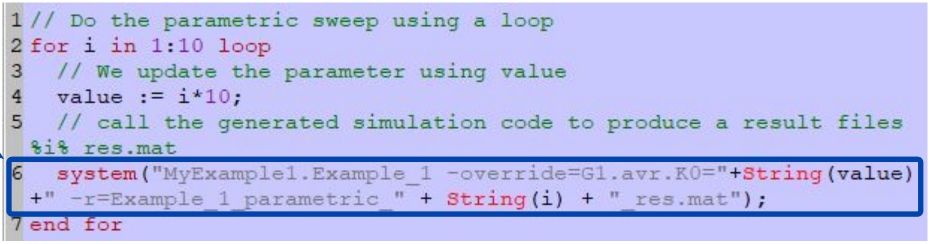
- The executable can be run the system command, i.e. equivalent to running from a CMD console in Windows
- We pass the data we want to modify using the flags:

  `-override = parameter.name= value`

  will set a new value to the AVR gain, G1.avr.K0”

  `-r = result_file_name.mat`

  gives a name to the result file at each new iteration.



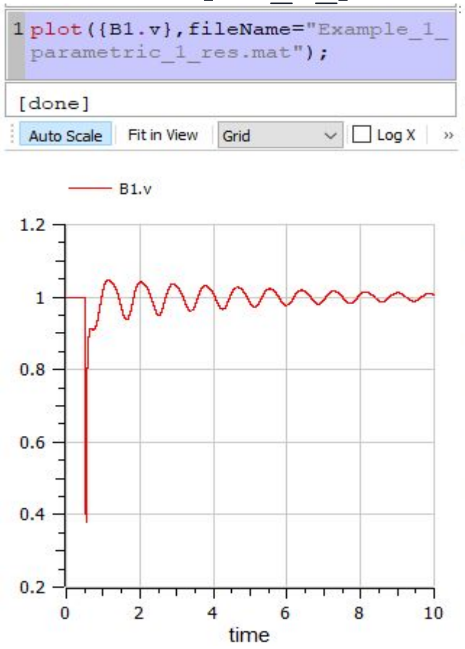
```
// Do the parametric sweep using a loop
for i in 1:10 loop
  // We update the parameter using value
  value := i*10;
  // call the generated simulation code to produce a result files
%i% res.mat
  system("MyExample1.Example_1 -override=G1.avr.K0="+String(value)
+" -r=Example_1_parametric_" + String(i) + "_res.mat");
end for
```

Parametric Sweep - Plots

- When plotting, we now need to refer to the specific output file we want to display results for:

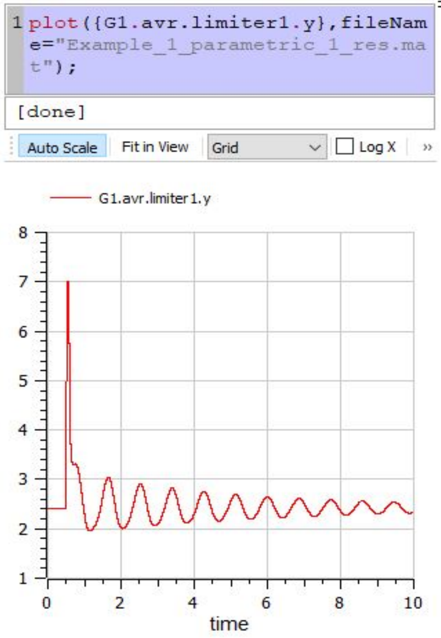
Example_1_parametric_**1**_res.mat

Example_1_parametric_**5**_res.mat



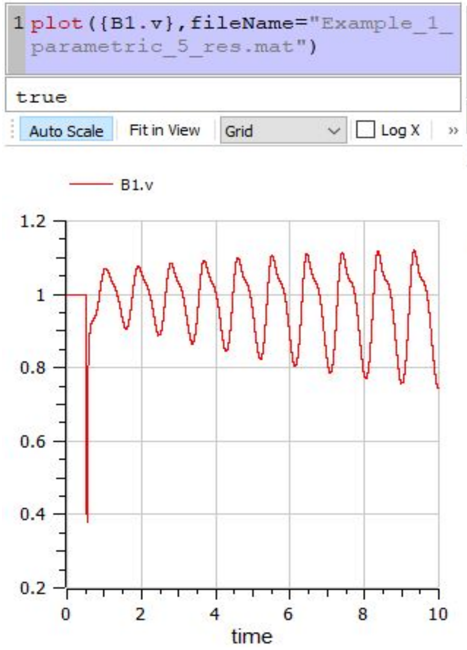
```
plot({B1.v},fileName="Example_1_parametric_1_res.mat");
```

```
[done]
```



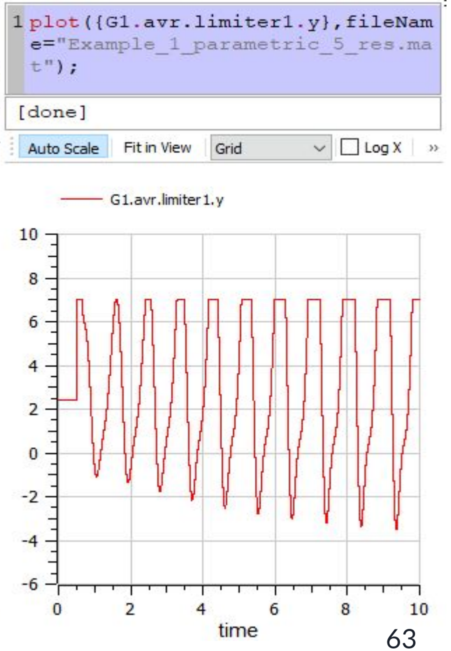
```
plot({G1.avr.limiter1.y},fileName="Example_1_parametric_1_res.mat");
```

```
[done]
```



```
plot({B1.v},fileName="Example_1_parametric_5_res.mat")
```

```
true
```



```
plot({G1.avr.limiter1.y},fileName="Example_1_parametric_5_res.mat");
```

```
[done]
```

Content

Where to go from here?

- Learn more about OpenIPSL
  - OpenIPSL overview video from 2020: https://youtu.be/2i3fvgFtcYA
  - Recent overview presentation on OpenIPSL and related efforts: https://youtu.be/IKdECHSO9wc

- What should I learn to build some power system simulation models on my own?
  - Learn more about how to use OMEdit:
    - Tutorial, courses, etc: https://openmodelica.org/useresresources/modelica-courses/
  - Go through the OpenIPSL full tutorial!
    - The instructions are outdated (form 2017!), but it should be good enough for you to learn how to put together some system models.
    - Link to Github: https://github.com/OpenIPSL/OpenIPSL/releases/tag/Tutorial_CURENT_ERC_2017_08_22
  - If you have Dymola, a recent tutorial also shows how to populate your model with the power flow solution form GridCal (or PSS/E):
    - Link to Github: https://github.com/ALSETLab/SMIB_Tutorial

- What about developing new models?
  - Learn about power system modeling, you can read these books:
    - Federico Milano, Power System Modelling and Scripting, 2010.
    - J. Chow and J. Sanchez-Gasca,Power System Modeling, Computation, and Control, 2019.
  - Learn about Modelica:
    - Dr. Michael M. Tiller, Modelica by Example: https://mbe.modelica.university/
    - P. Fritzson, Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach, 2014.
  - Learn about GIT and Github: https://git-scm.com/book/en/v2

- ***Learn More about Modelica:***
  - Modelica Association Website: https://modelica.org/

Thank you for your attention!

Beware of Pygmalion!

Happy Modeling!

http://Openipsl.org

Prof. Luigi Vanfretti
Professor @ http://ALSETLab.com
Rensselaer Polytechnic Institute
Troy, NY, USA