



iTesla			
Innovative Tools for Electrical System Security within Large Area			
Grant agreement number		Funding scheme	
Start date		Duration	
Call identifier			

## RaPIde Toolbox User manual

A Quick start GUIde for the RaPIde toolbox

Dissemination level		
<b>PU</b>	Public.	
<b>TSO</b>	Restricted to consortium members and TSO members of ENTSO-E (including the Commission Services).	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission services).	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission services).	<b>X</b>

Document Name: Quick start GUIde for the RaPIId toolbox  
 Work Package: WP3  
 Task: WP3.4  
 Deliverable: D3.3?  
 Responsible Partner: KTH

Author		Approval	
Name	Visa	Name	Date
[KTH] Achour AMAZOUZ, and Luigi VANFRETTI	2013/06/04		

DIFFUSION LIST	
For action	For information
RTE WP2 leader and relevant partners (Pepite) WP7 leader (INESC Porto)	All partners

## DOCUMENT HISTORY

Index	Date	Author(s)	Main modifications
0.1	30 <sup>th</sup> May 2013	AMAZOUZ, Achour	Document creation
2.0	<b>4<sup>th</sup> June 2013</b>	VANFRETTI, Luigi	<b>Revision and addition of JModelica.org FMU generation</b>

# CONTENTS

## 1. CONTENTS

---

<b>2. INTRODUCTION .....</b>	<b>3</b>
2.1. GOAL AND RESTRICTIONS .....	3
2.2. THE WAY IT WORKS .....	5
2.3. RECOMMENDATIONS .....	6
<b>3. SETTING-UP THE TOOLBOX .....</b>	<b>6</b>
3.1. SIMULINK MODEL .....	6
3.2. SETTING PARAMETERS .....	9
3.3. CONTAINERS (SETTING AND RESULTS BACK UP) .....	11
<b>4. DEEPER DESCRIPTION .....</b>	<b>12</b>
4.1. PARAMETERS DESCRIPTION .....	12
4.2. METHODS DESCRIPTION .....	<b>ERROR! BOOKMARK NOT DEFINED.</b>
4.3. COMMAND LINE EXECUTION .....	18
4.4. REMARKS CONCERNING THE COMMAND LINE .....	19
4.5. GENERAL REMARKS .....	19
<b>5. APPENDIX – ADVANCED FMU GENERATION .....</b>	<b>20</b>
5.1. ABSTRACT AND DISCLAIMER .....	20
5.2. JMODELICA FOR FMU GENERATION .....	20
5.2.1. <i>Aims and limitations</i> .....	20
5.2.2. <i>Enviroment set-up</i> .....	20
5.2.3. <i>Compilation of FMUs from JModelica</i> .....	21
5.3. FMU SIMULATION WITH THE FMI TOOLBOX AND JMODELICA .....	22
5.3.1. <i>Simulation with the FMI Toolbox</i> .....	22
5.3.2. <i>Simulation with JModelica</i> .....	23

## 2. INTRODUCTION

### 2.1. Goal and restrictions

**RaPid** is a toolbox providing a framework for parameter identification applicable to any dynamic system (not only power systems). A Modelica model of a power system is compiled into a Flexible Mock-Up Unit and made available in the Simulink environment through the FMI Toolbox for MATLAB. The model is characterized by a certain number of parameters whose values can be independently chosen and set-up for an identification process. The model can be simulated and its simulation outputs “measured”. RaPid attempts to tune the parameters of the model so as to obtain the best curve fitting between the outputs of the Simulated and experimentally obtained measurements of the same outputs provided by the user. This is graphically depicted in the figure below.

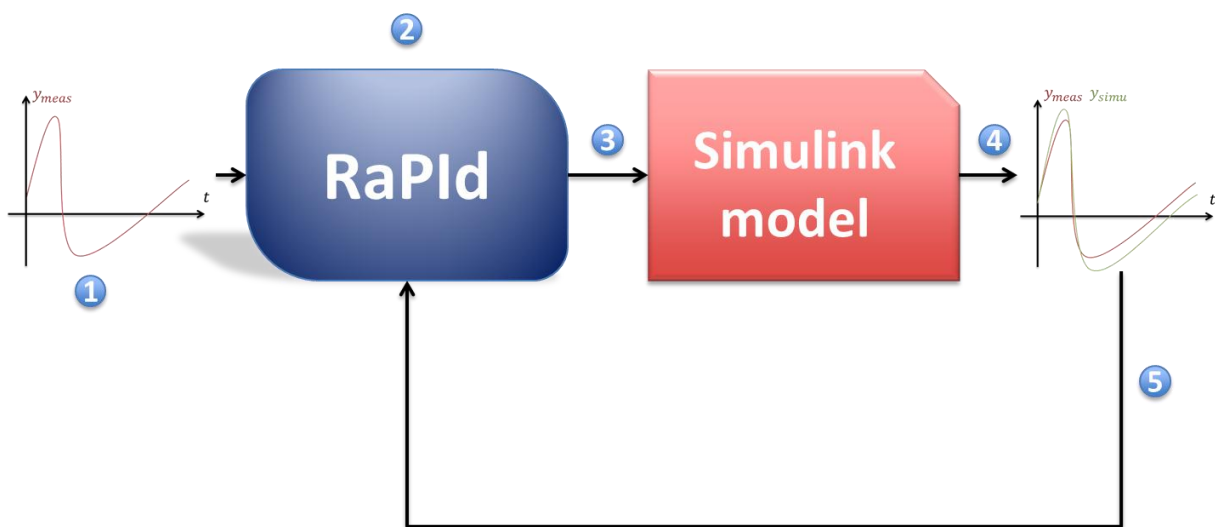


Figure 1: Toolbox iteration.

(1): Output (and optionally input) measurements are provided to RaPid by the user.

(2): At initialisation, a set of parameter is generated randomly or preconfigured in RaPid  
Iterations:

(3): The model is simulated with the parameter values given by RaPid

(4): The outputs of the model are recorded and compared to the user-provided measurements.

(5): A fitness function is computed to judge how close the measured data and simulated data are to each other

(2): Based on the fitness computed in (5) a new set of parameters is computed by RaPid

The iterations continue as long as a minimal fitness or a maximal number of iterations are reached.

The toolbox requires the users to comply with quite restrictive requirements:

1) The FMI Toolbox for Matlab 1.4 or 1.5<sup>1</sup> and Matlab R2012b must be installed in their 32bit versions.

<sup>1</sup> See <http://www.modelon.com>

- 2) The model to be tuned must be written in Modelica, exported as a \*.fmu<sup>2</sup> (Flexible Mock-Up Unit) compatible with Model Exchange (ME).
  - a) In our tests, the files generated with Dymola could reliably be imported by the FMI toolbox but not those generated by OpenModelica.
  - b) The \*.fmu files generated through a standard installation of Dymola require:
    - i) A model compilation license (node-locked or sharable) of Dymola to be used when calling the FMI toolbox
    - ii) A runtime license of Dymola can be used in a computer without a model compilation license for the FMI Toolbox to run FMUs compiled from Dymola by third parties.
    - iii) In order to be free of any Dymola license concern when generating \*.fmu files, the computer generating the \*.fmu from Dymola needs a "Binary model export license". The \*.fmu will be able to run in any computer with an FMI Toolbox or other FMI-compliant simulation tool.
  - c) FMU's can be generated from JModelica.org<sup>3</sup>
    - i) This requires the user to have expertise in using Python scripting and to set up an environment for loading JModelica and other needed libraries.
    - ii) JModelica does not yet support the complete Modelica Standard definitions (e.g. the reinit function is not supported, it is used for example in the DFIG Wind Turbine model in the Modelica power systems library).
- 3) Measurements data (and input data if available) should be contained in a Matlab file (\*.mat), the file can include any kind of Matlab object. The toolbox will let the user specify of to get the appropriate data from a file to be loaded.
- 4) The names of the different blocks in the Simulink model must be given in RaPId or the default names should be used. The Simulink block-diagram cannot be built disregarding this configuration.
- 5) A number of parameters need to be specified for the toolbox to work in a correct way. Any misunderstanding in a parameter's value will generate exceptions. *No exception handling was integrated at this early stage of development.*
- 6) Some problems have been experienced when using the Simulink models packaged with the toolbox on different computers with different versions of the FMI toolbox. The models might need rebuilding by the user.

The toolbox was built as a proof of concept. It can present some instability. A model simulated at several occasions without problem may suddenly provoke bugs. In those cases the toolbox itself complains, errors are displayed in the Matlab console. A reboot of Matlab usually fixes the problems related to the toolbox. Similarly, the FMI toolbox on which the RaPId toolbox relies may cause problems after several iterations of the simulation process which can lead to the following exception:

`module = FMICAPI, log level = FATAL: Could not load the DLL: Not enough storage is available to process this command.`

A reboot of Matlab is then needed to use the toolbox.

---

<sup>2</sup> See <https://www.fmi-standard.org>

<sup>3</sup> This document provides an example on FMU generation with JModelica, however this should be considered an alternative only for advanced users.

## 2.2. The way it works

RaPIId is a set of Matlab scripts and a GUI helping towards performing a parameter identification process.

The user has access to measurements. These measurements are given to the toolbox as, a target file and two commands leading to evaluation of a time vector and a signal matrix.

The file should be given as *inputData file* in the main window of the user interface. The elements *time* and *outputs* are the strings evaluated to load the content of the so-called file. *time* is a row vector *outputs* is a matrix whose rows are the different output signals measured. If *inputData file* contains a *struct* with time named *simout* as generated by a *to workspace* component in Simulink, *time* will have to be given as `transpose(simout.time)`, *outputs* will be given `transpose(simout.signal.values)`.

The model representing the system which produced the measurement data is described using the Modelica language and compiled into a \*.fmu container. This container can be loaded in the Simulink environment through the FMI Toolbox.

The Modelica model is characterized by a number of parameters and number of variables. Among the variables, a few are the output of the model. They should represent the same physical quantities as found in the measured data. Some of the parameters are known, some are to be identified.

RaPIId allows an algorithm to change the value of the parameters to be identified, then triggers the simulation of the Simulink system containing the Modelica model; subsequently, it extracts the output of this simulation and evaluates the fitness of the parameters chosen and feedback this values to the algorithm. Repeating this process iteratively, the algorithm tries to minimize the difference between the measured data and the simulated data.

Different algorithms can be used for this identification process. Different algorithms are tuned differently and yield different results. The toolbox was designed to be easily extended with new algorithms.

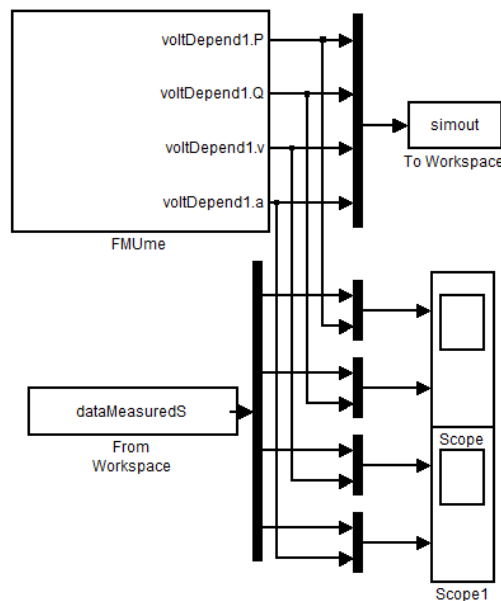
### 2.3. Recommendations

After extracting RaPIId to the desired location, a folder should be created inside the main folder of the toolbox. In this folder will be placed all the files related to one experience (certain parameter estimation procedure for a certain model).

Even on a modern computer, Simulink takes a long while to load for the first time. It is recommended to start by typing `Simulink` in the Matlab console before continuing. The RaPIId toolbox adds the FMI toolbox to your path based on the user-specified location, if a block is tagged with a *bad link* label in the Simulink diagram, this will be fixed later when the toolbox is launched.

## 3. SETTING-UP THE TOOLBOX

### 3.1. Simulink model



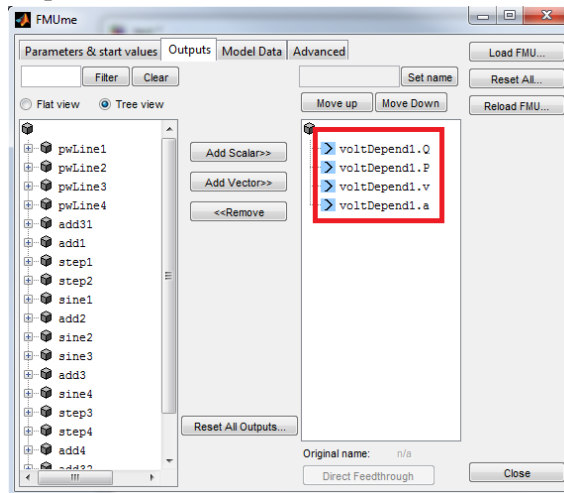
The Simulink model given in the folder *myTest* contains:

- The FMU me block where the modelica model is loaded
- A *To Workspace* component saving the simulated output. RaPIId will fetch the data saved by this component into workspace
- A *From Workspace* block allowing to include the measured outputs in the model
- Scopes plotting both the measured and simulated output while the computations are running. If the Simulink scopes are open before starting the toolbox every iteration of the optimization will be displayed
- The toolbox always loads in the workspace a struct with time dataMeasuredS based on the data provided by the user. This name should normally not be modified in the *From Workspace* component. The number of outputs (4 in this example) may vary with the model loaded. It's for the user to change the settings of the *scopes*, *mux* and *demux* components accordingly.

Two kinds of \*.fmu files exist. FMUcs (for co-simulation) include binaries for the solvers to be used in simulation. **FMUme (for model exchange) only contains a description of the model, when using FMUme, we rely on the solvers embedded in Simulink. RaPid only works with the later.**

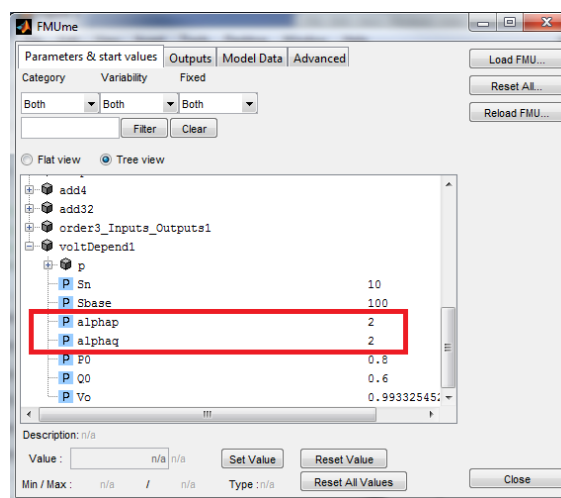
The choice of the solver and the solver options used in Simulink has to be configured in the toolbox. This implies that the FMU block chosen from the FMI toolbox has to be taken as Model Exchange compatible (ME and CS blocks both exist in the FMI library). Similarly, at creation of the \*.fmu file, the user has to make sure that the \*.fmu generated will be compatible with the ME standard (see compilation options in Dymola).

Double-clicking on the FMU block in the Simulink model allows the user to select the outputs of the model among all the variables present in the Modelica model:



**The order of appearance of these outputs matters. It should be identical to the order of the outputs in the measurement data provided by the user.** The outputs will indeed be compared one by one. If two outputs are switch in the FMU block, the identification will be made more difficult and might never succeed.

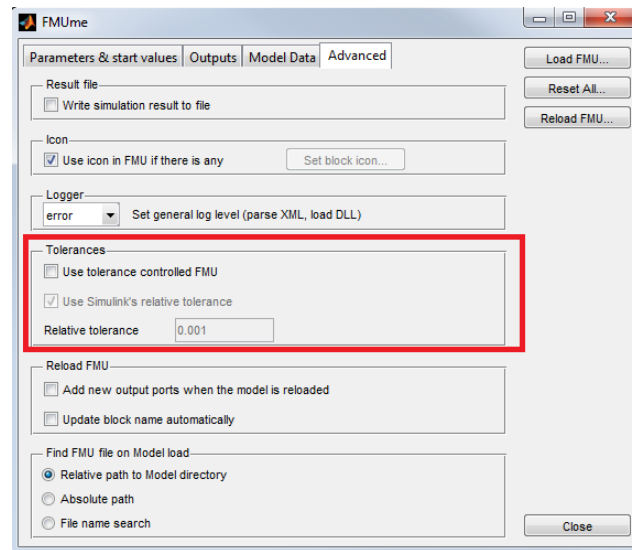
Besides, it is recommended to the user to note the complete name of all parameters whose values will be identified by RaPid. The user will indeed have to provide this information to the toolbox.



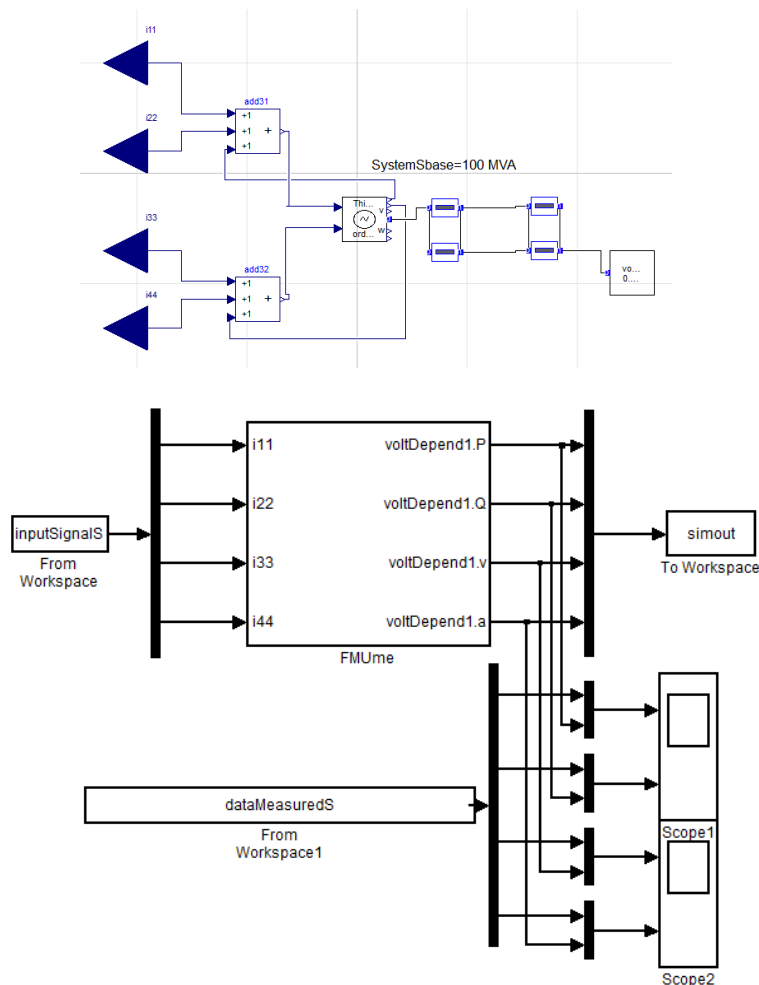
In this example, we note that the variable to be identified have the complete names **voltDepend1.alphap** and **voltDepend1.alphaq**.



If the user desires to use a solver with fixed step (for example 'ode1'), the use of tolerance controlled FMU in the fourth tab of the FMU block needs to be disabled:



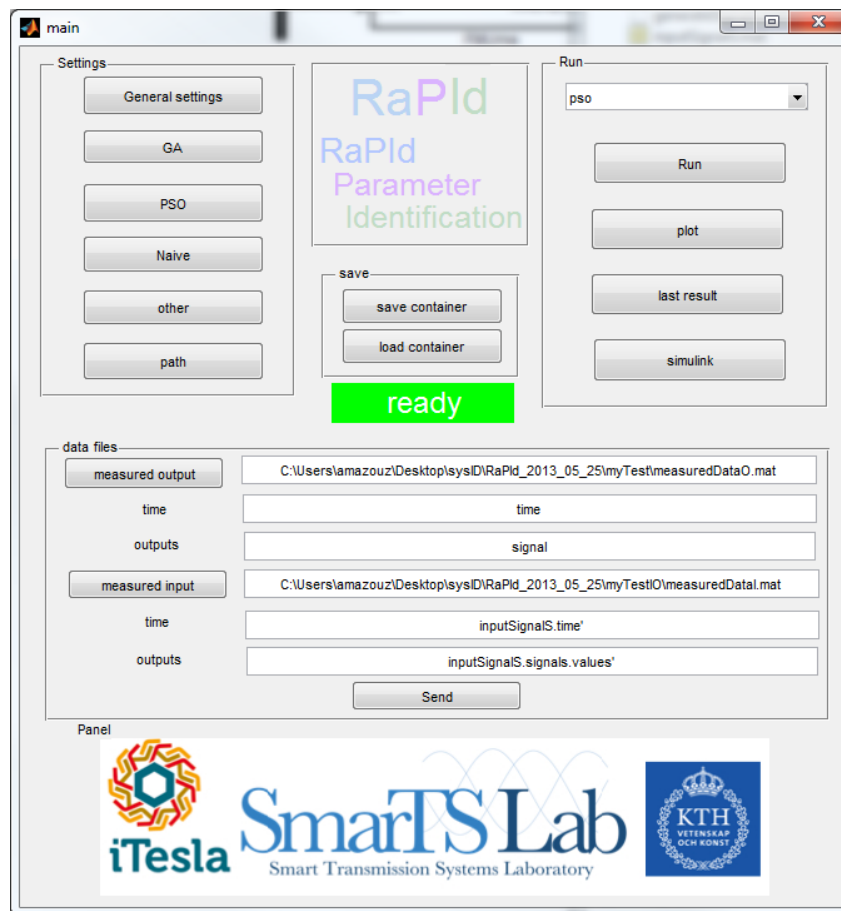
If the Modelica model contains input ports left unconnected, they will automatically appear as inputs to the FMI block.



The toolbox loads a struct called `inputSignals` based on the input data provided by the user.

The same happen with the struct called `dataMeasuredS` which represents the measured outputs. In the GUI, the user provides the path to the Matlab files containing the input and output measured data and a pair of keywords allowing to evaluate the time vector and signal matrix of the measurement.

The idea is that the toolbox loads the files given as *measured output* and *measure input* into the workspace and evaluates the content of the *time* and *outputs* fields. The field *time* should give a row vector containing all the samples at which the outputs were samples. The field *outputs* should give a matrix whose rows are the output signals themselves. The same can be transposed to the input data. The structs `inputSignals` and `outputMeasuredS` are constructed by the GUI based on the data aforementioned.



The simulation settings should not be configured in the Configuration Parameters of Simulink. These configuration parameters are overridden by the settings of the toolbox. In the GUI, these settings appear in the `generalSettings` menu.

### 3.2. Setting parameters

The first time the toolbox is open, the settings pre-defined in the GUI are for the most part incorrect.

They need to be set one after the other.

This section describes the settings to change every time a new model is simulated. They depend on your model and/or on your computer (e.g. path).

➡ cd to the RaPIId main directory (the GUI do not run correctly if the current directory is not the good one)

➡ Start the GUI by running the script run\_GUI.m

The script adds the appropriate folders of RaPIId in the path.  
The main window of the toolbox pops up.

➡ Click on *path* to set the path to the FMI toolbox main directory, set the path to the Simulink model built at the previous section. Using the button *Simulink model path* for that purpose will also give the value to the *Simulink model name* field. *FMU block name* should be the one given in the Simulink model, the same apply to “*To Workspace block name*”.

➡ Press *send*

The *Simulink* button brings up the model if the *Simulink model path* was given correctly. Looking into the Simulink model allows to take a peek to the different blocks' names.

➡ Back to the main window. Click on *General Settings*.

➡ Set the values for *solver*, *Ts*, *tf* just like you would do it in a Simulink model. They are the parameters of the simulation. Do not input this information directly in Simulink, they will be overridden by the settings set in *General Settings*.

➡ Set the values for *output names*, *parameter*, *param min*, *param max*, *quadratic cost* and *p0*. The field *output names* correspond to a struct with strings containing the names appearing on the FMU block once the corresponding outputs were selected in the *output* tab of the *FMU ME* component. It's composed the following way: subBlockName.variableName. The field *parameter* contains the name of all parameters to be identified by the toolbox. The content of these two components should be entered while reading the name of the components inside the Simulink model to minimise the risk of errors. The fields *param min*, *param max* and *p0* need to be of the appropriate size.

The field *quadratic cost* represents a vector containing the weights put on the square of the difference of every simulated output when compared to the measured output. If we write the fitness function

$$\text{fitness} = (\mathbf{Y}_{\text{meas}} - \mathbf{Y}_{\text{simu}})' \mathbf{Q} (\mathbf{Y}_{\text{meas}} - \mathbf{Y}_{\text{simu}})$$

*quadratic cost* represents the diagonal of  $\mathbf{Q}$ .

➡ Give the path to the \*.mat object containing the measurement data in *inputData file*


➡ Give in *time* and *outputs* the expression which, if evaluated in the Matlab command line after loading the data from *inputData file*, gives the values of the time row vector and the signal array (rows corresponds to different outputs.)

➡ Choose the optimisation method to be used via the drop down menu. Warning some methods rely on external methods (it is the case of Knitro, psoExt) be sure to add the path to these toolboxes on the corresponding emplacements in the menu *other* if you want to use one of these methods. Note that Knitro requires a license.

➡ Press the button *send*

➡ A press on *run* will now launch the toolbox and perform multiple iterations of the optimisation process

The simulation takes time. An indicator marked **running**/**ready**/**error** helps figuring out if the simulation is still running. An eye should be kept on the console in case an error is detected.

 After the computation has finished, plots can be showed and the final solution can be given via pressing the appropriate buttons.

### 3.3. Containers (setting and results back up)

The folder 'core' contains the file **data.mat**. This file contains all the parameters of the toolbox and of the parameter identification method to be used, but also the data generated at the last use of the toolbox.

Initially, the file doesn't exist. The fields of the GUI are filled with default values that need to be changed, for the most part. A parameter set with an invalid default value will generate errors.

The toolbox contains menus allowing setting the values for every needed parameter. These menus should be closed with the button labelled **send**, if another method lead to shutting the menu, all change will be discarded. When the button **send** is pressed, the data is saved in the file **data.mat**, any former settings are then discarded, only the new parameters will be taken into consideration.

This file may be exported/imported using the **save container** and **load container** buttons in the main window of the graphical interface. We advise the user to save a file for every single model on which to perform the system identification. Many of the parameters change when the model to be identified changes.

The file **data.mat** is a container transferred between the different elements of the toolbox. The user should normally not effect any change on this object outside the toolbox.

## 4. DIGGING DEEPER INTO RAPID: DETAILED SPECIFICATIONS AND FUNCTIONALITIES

### 4.1. Parameters description

Menu in the GUI	Name in the GUI	Name of the field in the <code>settings</code> struct	Functionality	Typical value
<b>General Settings</b>	Ts	settings.Ts	Sampling time for the simulated model whose parameter are tuned, only used if the integration method is a fixed step method.	positive real
	tf	settings.tf	final time of the simulation	real positive
	t0 fitness	settings.t0_fitness	start time for the computation of the fitness function (allow to remove the influence of the first second of the simulation on the computation of the cost function)	real positive, by default 0
	output names	settings.fmuOutData	names of the output measured, should correspond to both the measurement data and the output you set to the FMU block in the Simulink model	struct of strings every string is the name of the output as presented in the FMU block, e.g.: {'subbloc.output1',subbloc.output 2} the order matters, it has to match the order of the signals in the measured data and the order of the outputs in the Simulink model
	parameter	settings.parameterNames	names of the parameters whose values are going to be identified.	struct of strings, every string should contain the name of the appropriate parameter as stated in the FMU bloc e.g.: {'subbloc.parameter1','subbloc.parameter2'}
	param min	settings.p_min	vector containing the min acceptable value for all the parameter to be identified	Depends on the parameter, has to be real though... The order matters, has to match the order specified when you input the parameters' names
	param max	settings.p_max	vector containing the min acceptable value for all the parameter to be identified	Depends on the parameter, has to be real though... The order matters, has to match the order specified when you input the parameters' names
	verbose	settings.verbose	can trigger more data in the console, used in debugging of the methods	just put it to 0
	cost	settings.cost	identifiant of the cost function to be used, the function is shipped with only 1	put 2, might change if you implement new objective functions

			cost function which is quadratic	
quadratic cost	settings.objective.vect		weights describing the importance of every single output on the quadratic cost	vector of real positive numers, each weight corresponds to one output, they must hence be given in the same order as the outputs
solver	settings.intMethod		name of the integration method used in Simulink	string given the exact name of the integration method in Simulink, for example 'ode1' or 'ode45'
p0	settings.p0		initial guess for the vector of parameters to be identified	must be compatible with the upper and lower bounds specified in p_min and p_max. Some methods don't take into account this parameter.
Nb max	settings.nbMaxIterations		Number max of iteration before stopping the algorithm	Must be large, stopping the algorithm that way will not provide an optimal solution. Usefull for debugging purposes.

Menu in the GUI	Name in the GUI	Name of the field in the <b>settings</b> struct	Functionality	Typical value
<b>GA</b>	nbChromosome	settings.ga_options.nbChromosomes	number of vectors of parameters tested at every iteration of the genetic algorithm	natural integer remark: nbCrossover1+nbCrossover2+nbMutations+nbReproductions + nbResurrection = nbChromosome
	nbCrossover1	settings.ga_options.nbCrossover1	the crossover is a mean to reach better values for the vectors of parameters searched, here is the number of crossover of first type to be performed at every iteration	natural integer
	nbCrossover2	settings.ga_options.nbCrossover2	the crossover is a mean to reach better values for the vectors of parameters searched, here is the number of crossover of second type to be performed at every iteration	natural integer
	nbMutations	settings.ga_options.nbMutations	mutations are a mean to reach better values for the vectors of parameters searched, here is the number of mutations to be performed at every iteration	natural integer
	nbReproductions	settings.nbReproduction	when you reproduce a chromosome, it just remains the same	natural integer
	nbGenerations	settings.ga_options.limit	number of iterations of the genetic algorithm	natural integer
	fitnessStopRatio	settings.ga_options.fitnessStopRatio	the algorithm will stop before nbGenerations if the ratio $\text{phy\_now}/\text{phy\_start} < \text{fitnessStopRatio}$	very small real positive number

			meaning when we consider the fitness sufficiently low (weirdly enough, close match means small fitness index)	
headSize1	settings.ga_options.headSize1		the crossovers of type 1 are effected on the headSize1 chromosome with best fitness	natural integer < nbChromosome
headSize2	settings.ga_options.headSize2		the crossovers of type 2 are effected on the headSize2 chromosome with best fitness	natural integer < nbChromosome
headSize3	settings.ga_options.headSize3		the mutations are effected on the headSize1 chromosome with best fitness	natural integer < nbChromosome
nbResurrections	settings.ga_options.nbReinjection		number of random chromosomes that should have been deleted given their fitness but that are kept to add a random parameter	natural integer
nRandomMin	settings.ga_options.nRandomMin		the initial set of parameter vectors is determined as a set determined geometrically (vertices of the hypercube + regularly spaced vectors) to which we add a certain number of randomly generated vectors. nRandomMin is the minimum number of randomly generated particles desired	natural integer < nbChromosomes
p0s	settings.ga_options.p0s		matrix to be used to specify certain initial guesses of the parameters and to include them in the initial cell of chromosomes	matrix whose rows are different initial guesses, rows should respond to the same requirements as p0 from the general settings
storeData	settings.ga_options.storeData		boolean allowing to log data can break the memory for systems bigger than "tiny"	0

Menu in the GUI	Name in the GUI	Name of the field in the settings struct	Functionality	Typical value
<b>PSO</b>	alpha1	settings.pso_options.alpha1	inertia parameter (used to compute the particle's speed at next time instant )	real, alpha1 + alpha2 + alpha3 = 1
	alpha2	settings.pso_options.alpha2	parameter used to compute the speed at the next time instant, relates the component of the speed directed towards the global optimum of the swarm	same as alpha1
	alpha3	settings.pso_options.alpha3	parameter used to compute the speed at the next time instant, relates the component	same as alpha1

			of the speed directed towards the personal optimum of the particle	
nbliterations	settings.pso_options.limit		number maximal of iteration for the whole PSO algorithm	natural integer
nbParticles	settings.pso_options.nb_particles		number of particles in the swarm	natural integer
fitnessStopRatio	settings.pso_options.fitnessStopRatio		see description in the same parameter on GA	
kickMultiplier	settings.pso_options.kick_multiplier		the speed decreases as the number of iteration increases,when the particle is stuck we give it an impulse to start it back the kick multiplier helps determining when to consider the particle stuck it's the minimum ratio between current speed of the particle and maximum allowed speed of the particle	positive real number smaller than 1
nRandMin	settings.pso_options.nRandMin		same as in GA	
p0s	settings.pso_options.p0s		same as in GA	
storeData	settings.pso_opt.storeData		same as in GA	

Menu in the GUI	Name in the GUI	Name of the field in the settings struct	Functionality	Typical value
<b>Naive</b>	tolerance1	settings.naive_options.tolerance1	at every iteration, the optimisation consists of stepping in the opposite direction to the gradient with decreasing steps the tolerance is the smallest size admitted for the step	positive real
	tolerance2	settings.naive_options.tolerance2	at every iteration, the optimisation consists of stepping in the opposite direction to the gradient with decreasing steps the tolerance is the smallest size admitted for the step	positive real
	iterations	settings.naive_options.iterations	elementary number of optimization in orthogonal directions of the parameter space	natural integer
	iterations2	settings.naive_options.iterations2	number of time the elementary iterations are repeated for every orthogonal directions	natural integer
	iterations3	settings.naive_options.iterations3	third level of iteration	natural integer



Menu in the GUI	Name in the GUI	Name of the field in the settings struct	Functionality	Typical value
<b>other</b>	gaExtOptions	settings.gaExtOptions	optimset to give the genetic algorithm from Matlab	see options in "doc ga"
	nmOptions	settings.nmOptions	optimset to give the nelder-meade method	see options in "doc fminsearch"
	cgOptions	settings.cgOptions	optimset to give the conjugate gradient method	see options in "doc fminunc"
	psoExtPath	settings.psoExtPath	path to the toolbox download here: <a href="https://code.google.com/p/psomatlab/">https://code.google.com/p/psomatlab/</a>	
	psoExtOptions	settings.psoExtOptions	optimset for the toolbox psot	see options in "doc pso"
	path2knitro	settings.kn_options.path2Knitro	path to the knitro toolbox found at: <a href="http://www.ziena.com/knitro.htm">http://www.ziena.com/knitro.htm</a>	
	knitroOptions	settings.kn_options.knOptions	optimset for the knitro toolbox	see options in "doc ktrlink"
	option file	settings.kn_options.knOptionsFile	path to option file to be given the knitro toolbox	see knitro documentation
	firstMethod	settings.combiOptions.firstMethod	the method combi executes to methods one after the other this is the name of the first method	see drop down menu
	secondMethod	settings.combiOptions.secondMethod	name of the second method to be executed in combi	see dropdown menu

Menu in the GUI	Name in the GUI	Name of the field in the <b>settings</b> struct	Functionality	Typical value
<b>Path</b>	FMI toolbox path	settings.path2fmiToolbox	path to the FMI toolbox	C:\Program Files (x86)...
	Simulink model path	settings.path2fmiToolbox	path to the Simulink model containing the FMU block and the to workspace component	
	FMUme	settings.blockName	name of the FMU block in the Simulink diagram, must be simply "blockname" in the GUI, must be "modelname\blockname" in the command line	
	Simulink model name	settings.modelName	string containing the name of the Simulink model	
	to workspace block name	settings.scopeName	name of the To Workspace component in the Simulink model	
<b>main</b>	<b>inputData file</b>	<b>settings.path2data</b>	<b>path to the mat file containing the measurement data, can contain any kind of data handled by Matlab</b>	
	time	settings.dataT	command to be executed to obtain a row vector containing the time vector of the measurement when the inputData file is loaded in workspace	
	signal	settings.dataY	command to be executed to obtain a matrix whose row vectors contain the output vectors of the measurement when the inputData file is loaded in workspace	
	method	settings.methodName	name of the method to be used in the optimisation	

## 4.2. Optimization Methods and Tools

**pso**: particle swarm algorithm, a swarm of particles. The dimension of the vectorial space containing the particles is equal to the number of parameter to identify, a particle is a vector of parameter. The position of the global optimum of the whole swarm and this of every particle are memorized. A speed is given to every particle depending to the distance this particle has to it's personal optimum and the distance to the global optimum.

**ga** : A cell contains a number of chromosomes. A chromosome represent a vector of parameters. A certain number of chromosomes will be subject of a crossovers (they exchange part of their genes, a gene is a parameter), other will undergo a mutation (one of their gene is altered randomly). The process is repeated a great number of times, at every iterations we keep the best chromosomes and some random of the not best performing chromosomes.

**naive**: iteratively solve one-dimensional gradient methods in orthogonal directions taken randomly. Requires an initial guess.

**cg**: conjugate gradient method from Matlab (fminunc). Carefull! this method doesn't take into account the min max for the different parameters, requires an initial guess

**nm**: nelder-mead method, lies on fminsearch from Matlab, requires an initial guess

**combi**: choose two methods to apply one after the other, the second one uses the result of the first one as initial guess. The idea is to first use a machine learning based method (less prone to sticking to local minima) and then use an accurate gradient based method: typically NM

**psoExt**: like **pso** but taken from an external toolbox, <https://code.google.com/p/psoMatlab/>

**gaExt**: like **ga** but using the embedded Matlab function

**knitro**: powerful external optimisation function, <http://www.ziena.com/Matlabknitro.html>

### 4.3. Command line execution

The command line execution of the toolbox requires the user to create the *settings* struct containing all the parameters of the toolbox. A function to *rapid* call with all the appropriate settings definition for one specific method is rather compact and much easier to comprehend than going through all the menus of the GUI. Doing things in a command line is also a much less error prone approach.

The advantage of the command line approach is the control given to the user over all the elements coming to and from the toolbox.

The user should refer to the help of the function *rapid.m* and look into the function *run\_cmd.m* which gives an example of command line definition for every single method included.

The user should only change the *settings.methodName* flag to switch from one method to the other.

The parameters to be set for every method are the same as in the GUI, the list of parameters given in the present document is then still valid for the command line approach.

Using the *run\_cmd.m* file is pretty easy. The user needs to go through all fields and adapt them to his own example. The fields to be changed are in most cases:

- settings.path2SimulinkModel	- settings.p_min
- settings.modelName	- settings.p_max
- settings.blockName	- settings.p0
- settings.scopeName	- settings.objective.vect
- settings.fmuOutData	- settings.path2data
	- settings.parameterNames

#### 4.4. Remarks concerning the command line

The GUI takes care of loading the different toolbox needed to the path from the fields of the struct called `settings`. `Settings` is built as the users fills the different menus of the GUI. In command line, the user will then have to include the following lines to reproduce this mechanism.

```
addpath(genpath(settings.path2fmiToolbox))
addpath(genpath(settings.psoExtPath)) {this only if the external pso toolbox is used}
addpath(genpath(settings.kn_options.path2Knitro))
```

The array of settings given formerly contains all the data entered by the user in the GUI. When using the toolbox in command line, all these fields need to be input manually by the user. You may however decide not to input the settings relative to an optimisation method that you won't use, which makes the process shorter. However, the GUI includes to `settings` some fields that don't appear in the table here above. These measurement data needs to be included in the settings struct following:

```
load(settings.path2data)
settings.realData = eval(settings.dataY);
settings.realTime = eval(settings.dataT);
```

These lines are valid if you filled all the fields of the struct as specified in the table here above.

Otherwise, just arrange yourself to have `settings.realData` a row vector of time samples, and `settings.realTime` a matrix whose rows represent the measurement of every output at the corresponding sample times.

#### 4.5. General remarks

The core folder contains a file named `data.mat` that should never be modified by the user. The user also need to make sure no other file is ever called `data.mat` within the Matlab path or the current directory.

The optimsets of the different functions from Matlab (`fmincon`, `fminsearch`, `fminunc`, `ga` and so on) can be generated through the optimization toolbox obtained with the command `optimtool`.

This provide a GUI that allows to export into the workspace a struct that can then be fed to the toolbox in the appropriate fields of the menu 'other'.

## 5. APPENDIX – ADVANCED FMU GENERATION

### *JModelica FMU Compilation and Simulation*

#### 5.1. Abstract and Disclaimer

This Appendix describes the use of JModelica.org for the generation of Flexible Mock-Up units and to carry out simulation using FMUs both in JModelica.org and with the FMI Toolbox for Matlab using Simulink blocks.

This is considered an advanced option, and the user should be aware that relevant experience with programming is necessary. *If you are not comfortable with working with source code or to program using Python this option is not recommended and instead you should refer to the documentation above regarding the use of Dymola.*

#### 5.2. JModelica for FMU Generation

##### 5.2.1. Aims and limitations

JModelica.org is an extensible Modelica-based OSS platform, providing facilities for simulation and analysis of Modelica models.

However, it has the following disadvantages for novice users:

- It does not provide a user interface
- It requires some expertise working with Python scripting
- It requires a correct set up of JModelica
- It does not support the complete Modelica standard definition ( e.g. `reinit` command used by the DFIG WT is not support -> no FMU can be generated)

We illustrate one example for FMU here, carried out with JModelica.org-1.8.1.

##### 5.2.2. Enviroment set-up

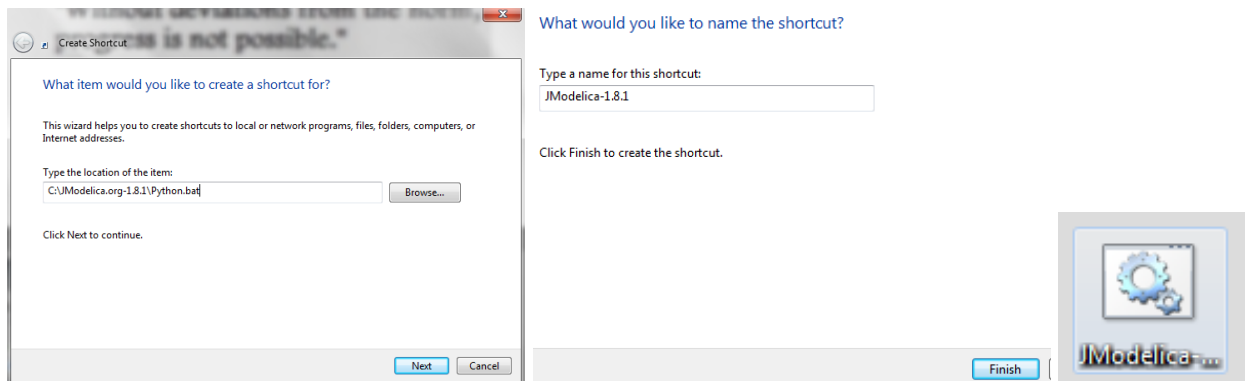
Follow the next instructions to set-up an environment to compile FMUs with JModelica:

- Download JModelicaorg-1.8.1:  
<http://www.jmodelica.org/downloads/JModelica.org-1.8.1.exe>
- Download Python(x,y) – an environment for numerical computations using Python  
<https://code.google.com/p/pythonxy/>
- Python(x,y) will include Spyder, a development environment for the Python language.
- Open the file C:\JModelica.org-1.8.1\Python.bat with Notepad++ or similar. Make sure you have the following statements:

```
1 @echo off
2 call C:\JModelica.org-1.8.1\setenv.bat
3 C:\Python27\python.exe "C:\Python27\Scripts\spyder"
```

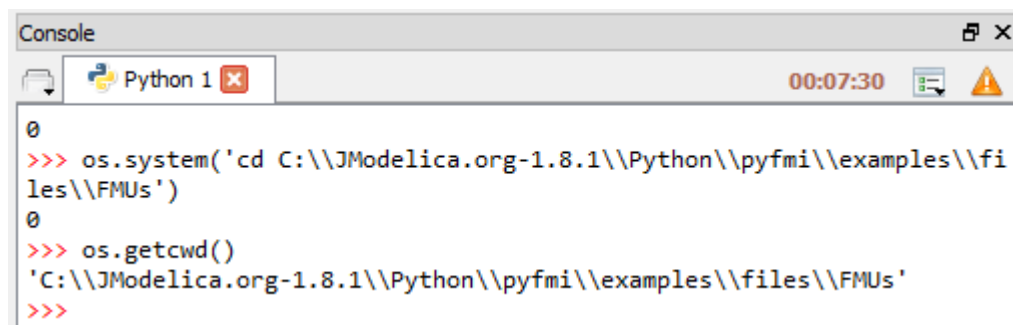
- Create a shortcut for JModelica running in Spyder automatically:
  - Create a shortcut icon in your desktop, and point it to the "C:\JModelica.org-1.8.1\Python.bat" file.
  - Give a name to the shortcut and click on "Finish"

- You should get a shortcut to launch Spyder with the Jmodelica libraries loaded and enabled



### 5.2.3. Compilation of FMUs from JModelica

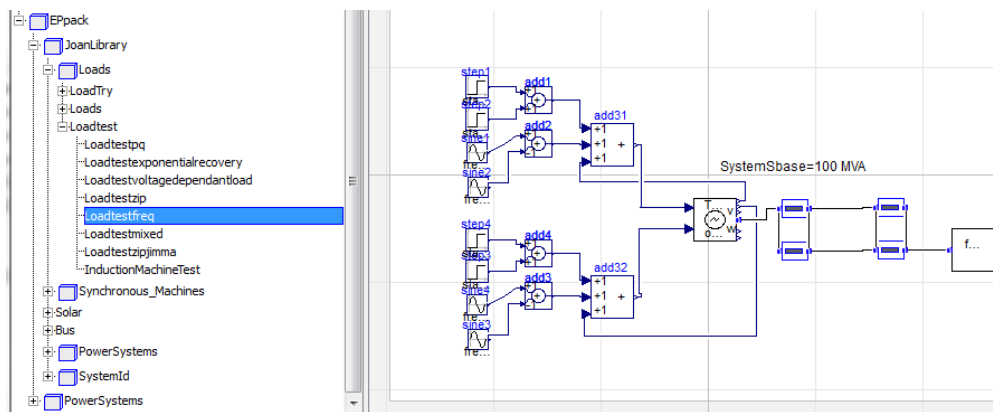
- Go to the directory where you have put the Modelica model you wish to compile through the command prompt in Spyder, and check that you are there:



- The following syntax can now be used to compile your FMU, we use it in the next example:

```
# Import the compiler function
from pymodelica import compile_fmu
# Specify Modelica model and model file (.mo or .mop)
model_name = 'myPackage.myModel'
mo_file = 'myModelFile.mo'
# Compile the model and save the return argument, which is the file name of
the FMU
fmu_file = compile_fmu(model_name, mo_file)
```

- Compile FMUs from Jmodelica: locating the model name and the file name.
  - We will compile a small test system including a third order generator, lines and a frequency dependent load that can be used for load parameter estimation
  - This model is contained the 'EPpack.mo' package, this is the file name.
  - From the hierarchy illustrated in the figure below, one can obtain the model name: EPpack.JoanLibrary.Loads.Loadtest.Loadtestfreq



- Go to the console of Syper and type the commands as shown in the screenshot

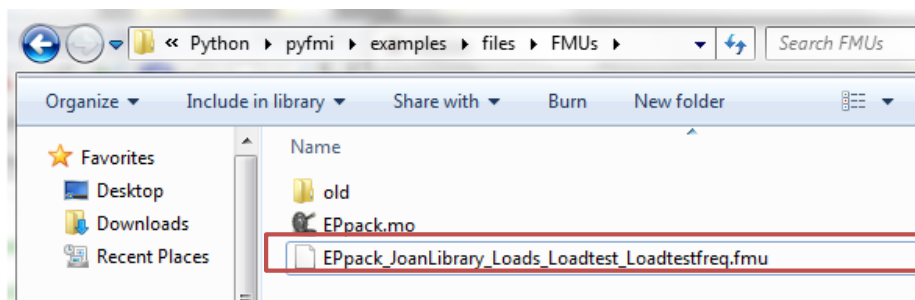
```

Console
Python 1 x
00:21:56

0
>>> os.system('cd C:\\JModelica.org-1.8.1\\Python\\pyfmi\\examples\\files\\FMUs')
0
>>> os.getcwd()
'C:\\JModelica.org-1.8.1\\Python\\pyfmi\\examples\\files\\FMUs'
>>> from pymodelica import compile_fmu
JVM started.
>>> model_name = 'EPpack.JoanLibrary.Loads.Loadtest.Loadtestfreq'
>>> mo_file = 'EPpack.mo'
>>> fmu_file=compile_fmu(model_name,mo_file)
>>>

```

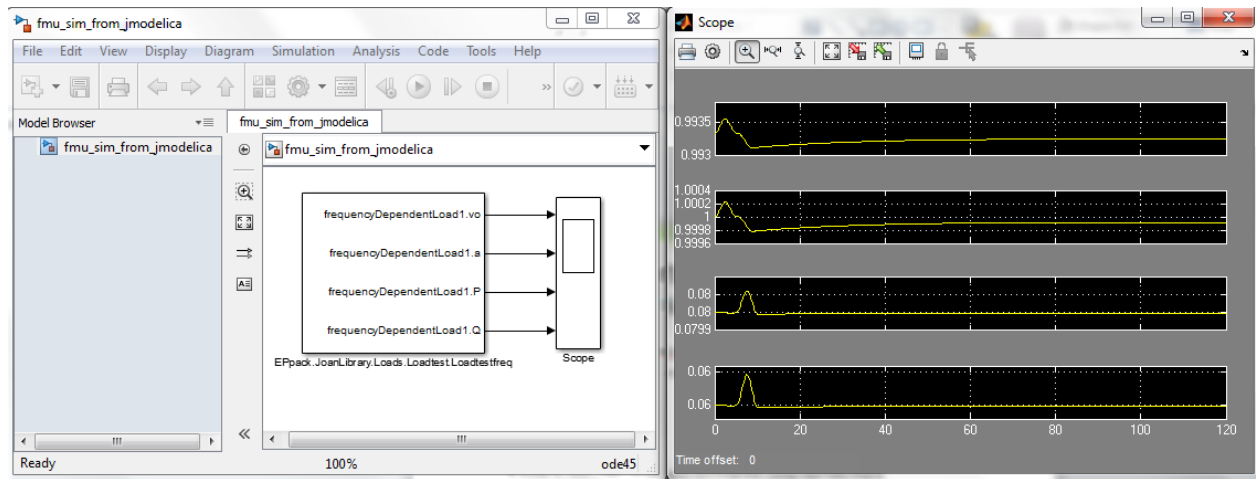
- Browse to your FMU folder and you will find the compiled FMU there!



## 5.3. FMU Simulation with the FMI Toolbox and JModelica

### 5.3.1. Simulation with the FMI Toolbox

- Create a new Simulink file and load the FMU block for Model Exchange
- Load the FMU and select the outputs to display during simulation
- Put a scope to display the results
- Click on play to see if the FMU is working properly, you should get the results below, the inspected values are at the load bus (V, angle, P and Q):



- You might have the situation that the simulation results make no sense. This is not an issue of Matlab/Simulink nor the FMI Toolbox
- It is simply due to the fact that JModelica does not support the full definition of the Modelica standard, and thus, its compiler may not be able to give a “correct” FMU (or even it may create an error creating the FMU).

### 5.3.2. Simulation with JModelica

JModelica supports simulation of Modelica models using standard Modelica files, FMUs or JMUs. The script below can be used for simulating the FMU of the example above in JModelica.

```
# -*- coding: utf-8 -*-
"""
Created on Friday May 31 00:22:56 2013

@author: luigiv
"""

import os as O
import pylab as P
import numpy as N

from pyfmi import FMUModel

curr_dir = O.path.dirname(O.path.abspath(__file__));
path_to_fmus = O.path.join(curr_dir, 'files', 'FMUs')

def run_demo(with_plots=True):
    """
    Demonstrates how to use JModelica.org for simulation of FMUs.
    """

    fmu_name = O.path.join(path_to_fmus, 'EPpack_JoanLibrary_Loads_Loadtest_Loadtestfreq.fmu')
    model = FMUModel(fmu_name)

    opts = model.simulate_options()
    opts['CCode_options']['atol'] = 1.0e-6 # specific tolerance
    opts['CCode_options']['atol'] = 1.0e-6 # specific tolerance
    opts['ncp'] = 5000 #changing the number of communication points

    res = model.simulate(options=opts, final_time=100.)
    print("FMU succesfully simulated")

    #Retrieve the result for the variables
    output_vo_0 = res['frequencyDependentLoad1.vo']
    output_vo_1 = res['frequencyDependentLoad1.a']
    output_p = res['frequencyDependentLoad1.P']
```



```

output_q = res['frequencyDependentLoad1.Q']
t         = res['time']

#Plot the solution
if with_plots:
    fig = P.figure()
    P.clf()
    P.subplot(2,1,1)
    P.plot(t, output_vo_0)
    P.ylabel('Voltage Magnitude at Load (pu)')
    P.xlabel('Time (s)')
    P.subplot(2,1,2)
    P.plot(t, output_vo_1)
    P.ylabel('Voltage Angle at Load (rad)')
    P.xlabel('Time (s)')
    P.suptitle('FMU Simulation using JModelica')
    P.show()

    fig2 = P.figure()
    P.clf()
    P.subplot(2,1,1)
    P.plot(t, output_p)
    P.ylabel('Active Power Response (pu)')
    P.xlabel('Time (s)')
    P.subplot(2,1,2)
    P.plot(t, output_q)
    P.ylabel('Reactive Power Response (pu)')
    P.xlabel('Time (s)')
    P.suptitle('FMU Simulation using JModelica')
    P.show()
if __name__ == "__main__":
    run_demo()

```

The results are shown in the plots below, being identical to the results produced with the FMI Toolbox for Matlab, the inspected values are at the load bus (V, angle, P and Q):

