RaPId : a Parameter Identification Toolbox

iTesla Model Validation SW Mock-Up Prototype for WP3.3 and WP3.4 used for Component Parameter Estimation and Aggregate Model Validation

RaPId Toolbox User Manual
A Quick start GUIde for the RaPId toolbox

Achour Amazouz and Prof. Dr.-Ing. Luigi Vanfretti

E-mail: luigiv@kth.se
Web: http://www.vanfretti.com

luigiv@kth.se
Associate Professor, Docent
Electric Power Systems Dept.
KTH
Stockholm, Sweden

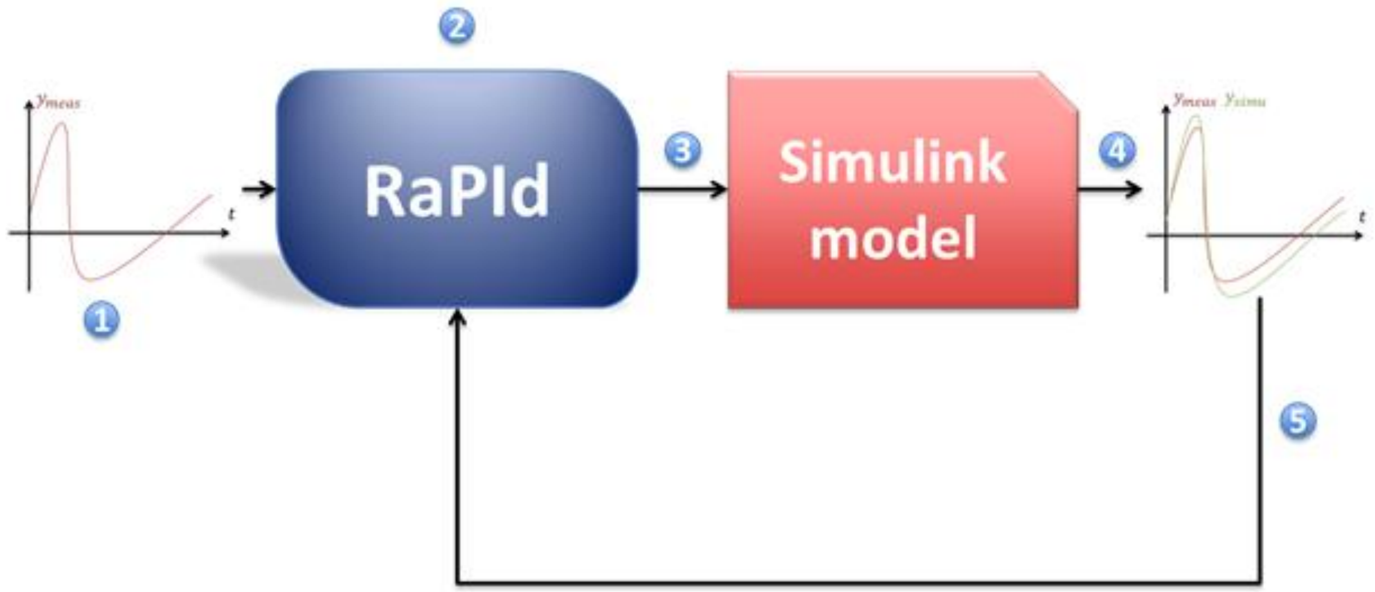
Luigi.Vanfretti@statnett.no
Special Advisor in Strategy and Public Affairs
Research and Development Division
Statnett SF
Oslo, Norway

June 2013

What is RaPId?

- **RaPId** is a toolbox providing a framework for parameter identification.
- A Modelica model, made available through a Flexible Mock-Unit (i.e. FMU) in the Simulink environment, is characterized by a certain number of parameters whose values can be independently chosen.
- The model is simulated and its outputs are measured.
- **RaPId** attempts to tune the parameters of the model in Simulink so as to obtain the best curve fitting between the outputs of the Simulation and the experimental measurements of the same outputs provided by the user.

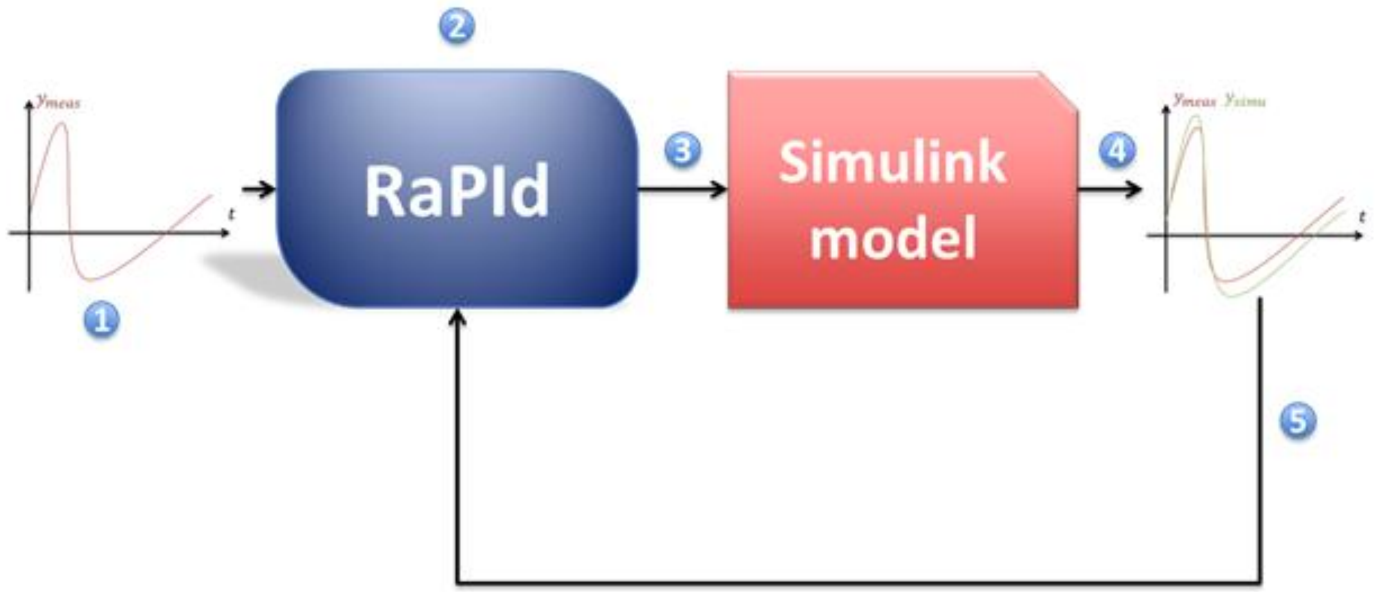
What is RaPId doing?

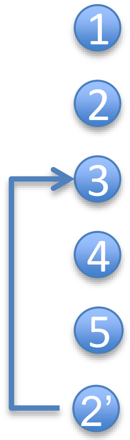


1. Output (and optionally input) measurements are provided to RaPId by the user.
2. At initialisation, a set of parameter is generated randomly or preconfigured in RaPId.
3. The model is simulated with the parameter values given by RaPId.
4. The outputs of the model are recorded and compared to the user-provided 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

What is RaPId doing?

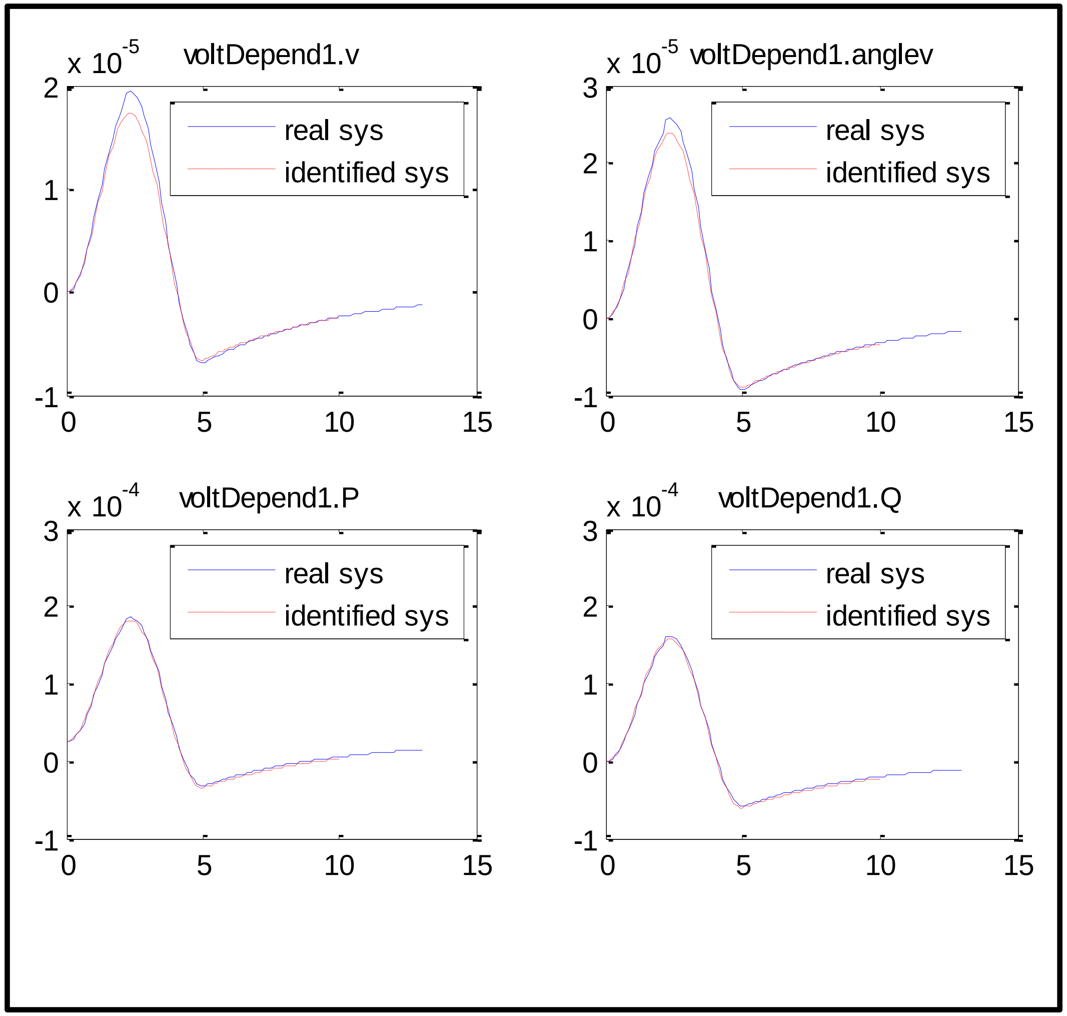




The iterations go on until a minimal fitness or a maximal number of iterations are reached.

Objective

- The model, written in Modelica, is characterised by a vector of parameters
- Find a vector allowing close matching of the Model’s output to outputs measured experimentally



Parameter Id for a load characterised by two parameters and four outputs

Requirements

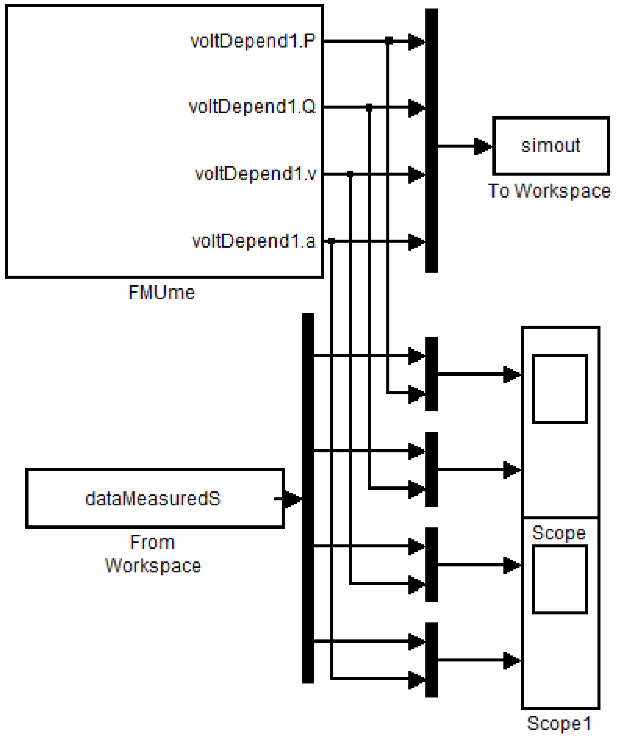
- FMI Toolbox and Matlab must be installed in the 32bit versions.
- The *.fmu container allowing import of the Modelica model in simulink needs to be compatible with Model Exchange (ME).
- If the *.fmu file is generated with Dymola
  - With local license or sharable license of Dymola, the license needs to be in the machine running the FMU
  - A runtime license of Dymola may be needed for FMI to be able to run the simulations in the machine excec the FMU
  - If FMU generated by a ”Binary model export license” of Dymola, the FMU will run anywhere
- The output (resp. input) user-provided data needs to be given in a *.mat file. Both time and value for every measured quantity needs to be provided.

Preparation of the Simulink Model

- Warning:
  We have experienced some problems when using the Simulink models given as example in the RaPId toolbox on different computers with different versions of the FMI toolbox.
  The content describe here was tested with FMI Toolbox 1.5
- You might have to build the model yourself if a problem is encountered when simulating the Simulink model

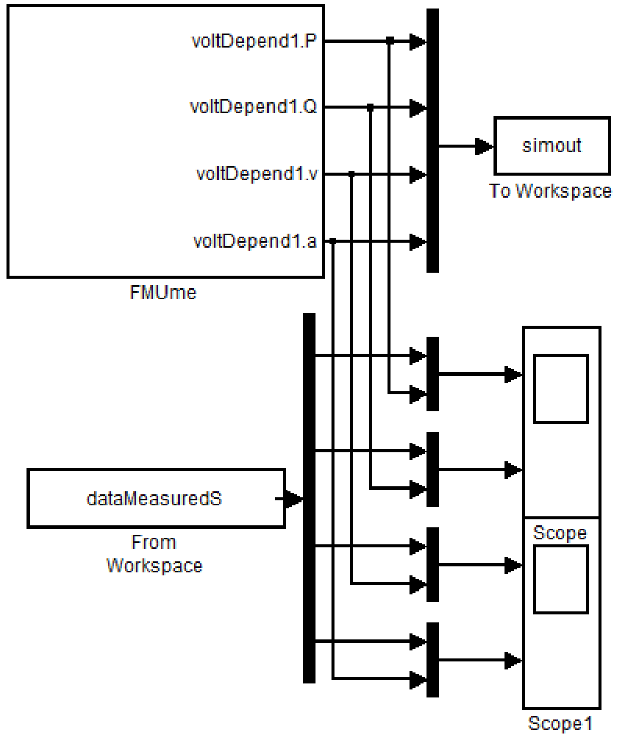
Preparation of the Simulink Model

- The simulink model given in example contains:
  - The FMU me block where the modelica model is loaded
  - A ”To Workspace” component saving the simulated output. RaPId will fetch the data saved by this component into workspace
  - A ”From Workspace” block allowing to include the measured outputs in the model
  - Scopes ploting both the measured and simulated output while the computation are running. If the scopes are open before starting the toolbox every iteration of the optimisation will be displayed



Preparation of the Simulink Model

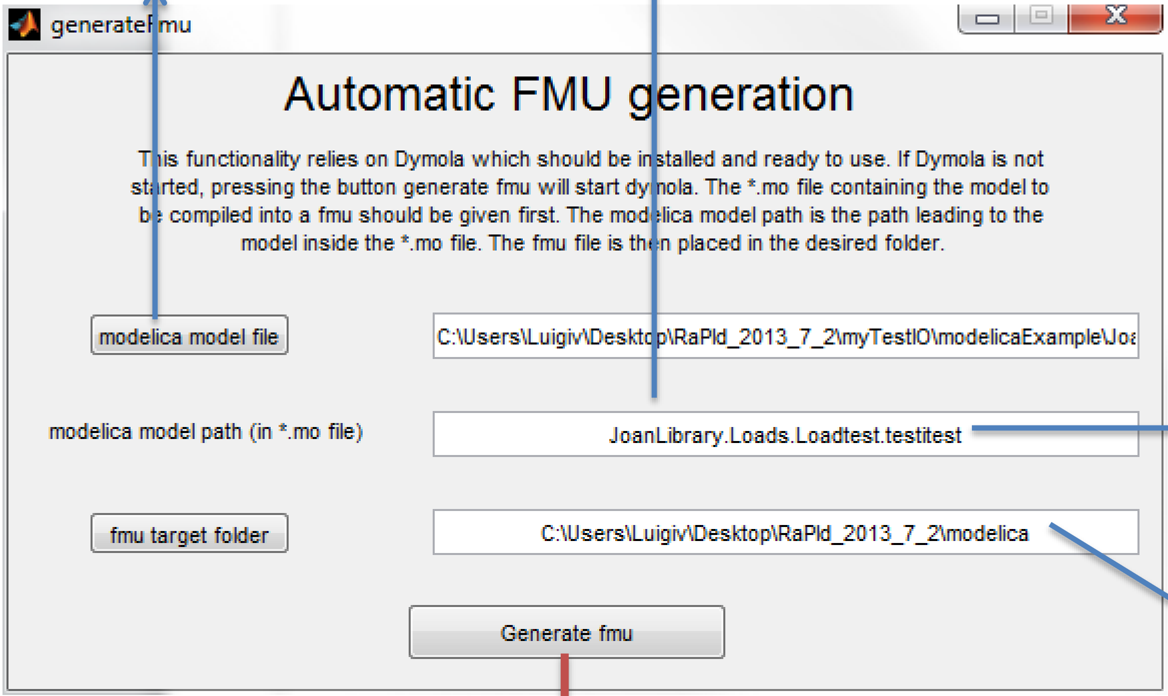
- The toolbox always loads in workspace a struct with time dataMeasuredS based on the data user-provided. This name should normally not be changed from the ”From Workspace” component.
- The name of the variable created by the ”To Workspace” component can be parametrised in the toolbox
  - i.e. the name of simout can be changed, but it must be modified through the GUI in the ”path” sub-GUI.
- The number of outputs (4 in this example) may vary. It's for the user to change the settings of the scopes, mux and demux components accordingly.



Generate the FMU

Select the .mo file containing the Modelica description for the model

Type the Modelica path to the model to be used



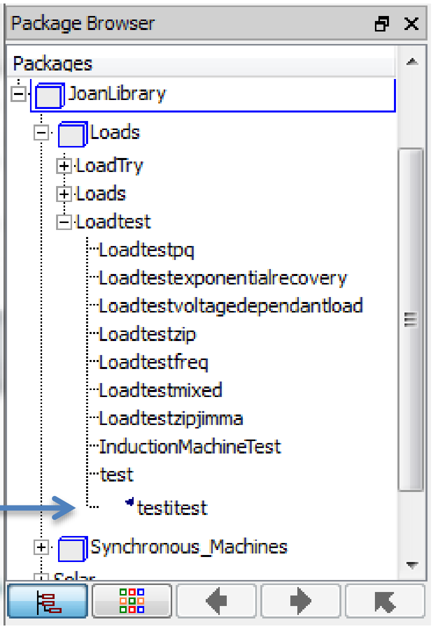
generateFmu

Automatic FMU generation

This functionality relies on Dymola which should be installed and ready to use. If Dymola is not started, pressing the button generate fmu will start dymola. The *.mo file containing the model to be compiled into a fmu should be given first. The modelica model path is the path leading to the model inside the *.mo file. The fmu file is then placed in the desired folder.

modelica model file	C:\Users\Luigiv\Desktop\RaPId_2013_7_2\myTestIO\modelicaExample\Jo
modelica model path (in *.mo file)	JoanLibrary.Loads.Loadtest.testitest
fmu target folder	C:\Users\Luigiv\Desktop\RaPId_2013_7_2\modelica

Generate fmu



Package Browser

Packages

```
JoanLibrary
  Loads
    LoadTry
    Loads
    Loadtest
      Loadtestpq
      Loadtestexponentialrecovery
      Loadtestvoltagedependantload
      Loadtestzip
      Loadtestfreq
      Loadtestmixed
      Loadtestzipjimma
      InductionMachineTest
      test
      testitest
  Synchronous_Machines
  Solar
```

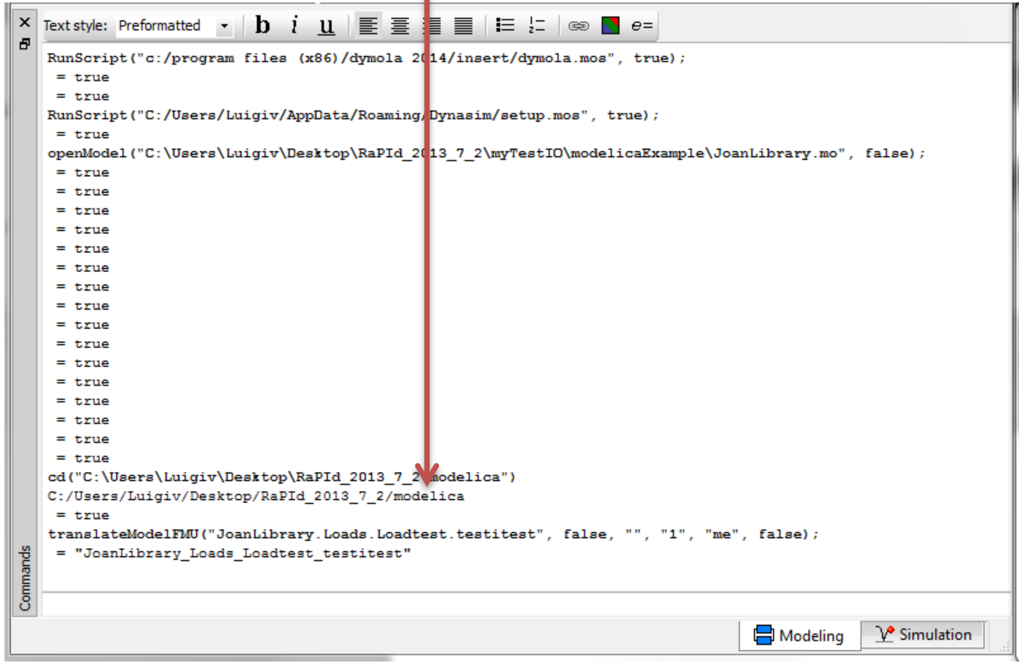
Select the folder where you want to place the FMU

Click to generate the FMU

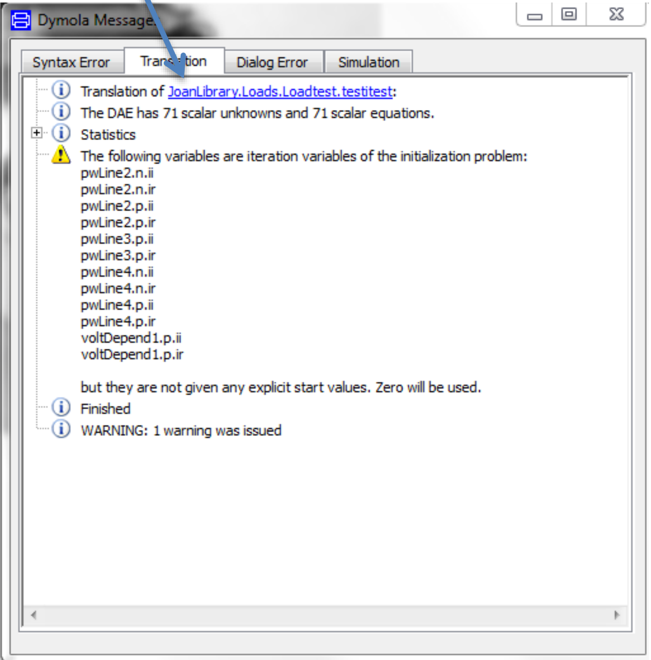
The FMI Toolbox will need to be provided (`Load FMU`) with the FMU generated

Generate the FMU

- Dymola will launch automatically and compile the FMU placing it in the selected folder.
- You can check this in the Dymola translation log and the Commands window.



```
RunScript("c:/program files (x86)/dymola 2014/insert/dymola.mos", true);
 = true
 = true
RunScript("C:/Users/Luigiv/AppData/Roaming/Dynasim/setup.mos", true);
 = true
openModel("C:\Users\Luigiv\Desktop\RaPId_2013_7_2\myTestIO\modelicaExample\JoanLibrary.mo", false);
 = true
 = true
 = true
 = true
 = true
 = true
 = true
 = true
 = true
 = true
 = true
 = true
 = true
 = true
 = true
 = true
cd("C:\Users\Luigiv\Desktop\RaPId_2013_7_2\modelica")
C:/Users/Luigiv/Desktop/RaPId_2013_7_2/modelica
 = true
translateModelFMU("JoanLibrary.Loads.Loadtest.testitest", false, "", "1", "me", false);
 = "JoanLibrary_Loads_Loadtest_testitest"
```



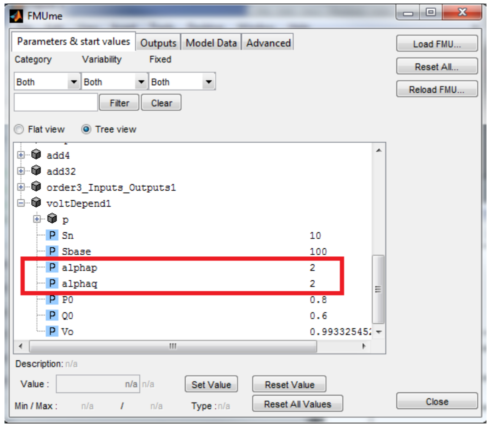
```
Translation of JoanLibrary.Loads.Loadtest.testitest:
The DAE has 71 scalar unknowns and 71 scalar equations.
Statistics
The following variables are iteration variables of the initialization problem:
pwLine2.n.ii
pwLine2.n.ir
pwLine2.p.ii
pwLine2.p.ir
pwLine3.p.ii
pwLine3.p.ir
pwLine4.n.ii
pwLine4.n.ir
pwLine4.p.ii
pwLine4.p.ir
voltDepend1.p.ii
voltDepend1.p.ir

but they are not given any explicit start values. Zero will be used.
Finished
WARNING: 1 warning was issued
```

Import the *.fmu in Matlab

- Once the FMI toolbox from Modelon is imported to the matlab path, two blocks are available in Simulink:
  - FMI ME: for model exchange
    - the fmu file only contains the system’s equation
    - The standard simulink solvers and settings are used
  - FMI CS: for co simulation
    - Include the solvers’ binaries for simulation purposes
    - Corresponds to an external model run with its own solver
- Here we use the ME block
- Import using the ”Load fmu” button after clicking on the FMU block in simulink

Setting up the *.fmu

In the first tab, note the complete names of the parameters that will be identified.
In this example, we note voltDepend1.alphap and voltDepend1.alphaq

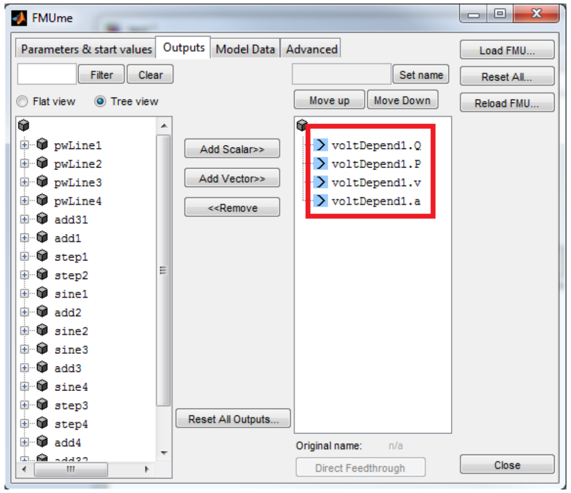


Category: Both; Variability: Both; Fixed: Both

```
add4
add32
order3_Inputs_Outputs1
voltDepend1
  p
  P Sn        10
  P Sbase     100
  P alphap    2
  P alphaq    2
  P P0        0.8
  P Q0        0.6
  P Vo        0.99332545:
```

Description: n/a
Value : n/a n/a
Min / Max : n/a / n/a
Type : n/a

In the second tab, select the outputs in the same order as provided in the measurement data. These outputs are selected among the all the variables of the modelica model.



```
pwLine1
pwLine2
pwLine3
pwLine4
add31
add1
step1
step2
sine1
add2
sine2
sine3
add3
sine4
step3
step4
add4
add32
```

```
voltDepend1.Q
voltDepend1.P
voltDepend1.v
voltDepend1.a
```

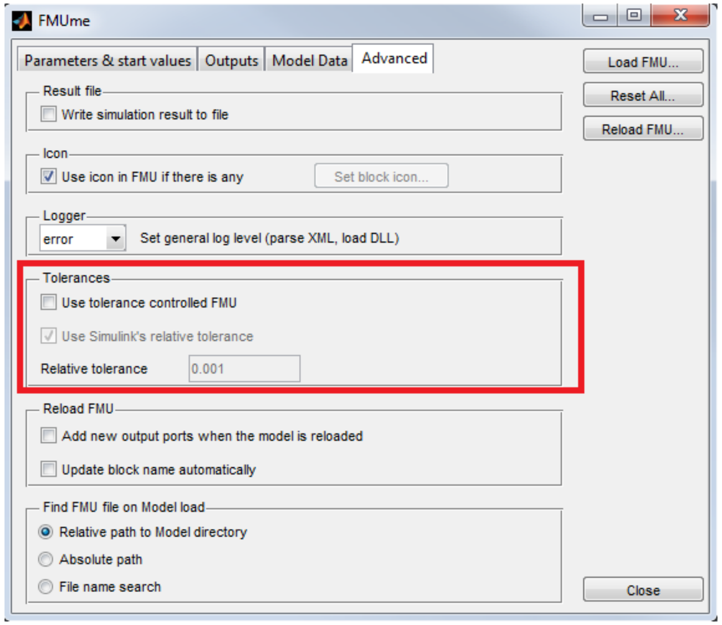
Original name: n/a

Setting up the *.fmu

Basic Solver Configuration

If you will simulate the system with a constant step solver (for example ’ode1’), you need to disable the use of tolerance controlled FMU in the fourth tab.

You can let it enabled as long as you specify a variable step solver in the toolbox (for example ode45). This setting will be showed later on.



Result file
- [ ] Write simulation result to file

Icon
- [✓] Use icon in FMU if there is any

Logger
error — Set general log level (parse XML, load DLL)

Tolerances
- [ ] Use tolerance controlled FMU
- [✓] Use Simulink's relative tolerance

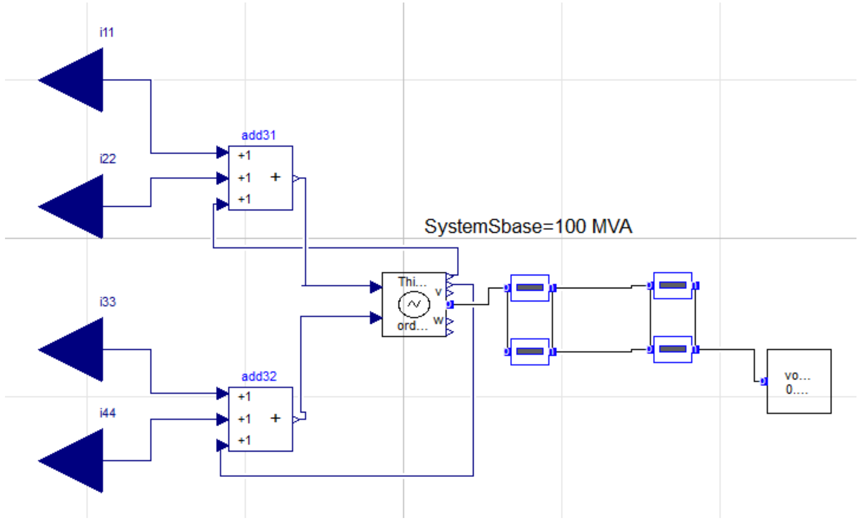
Relative tolerance 0.001

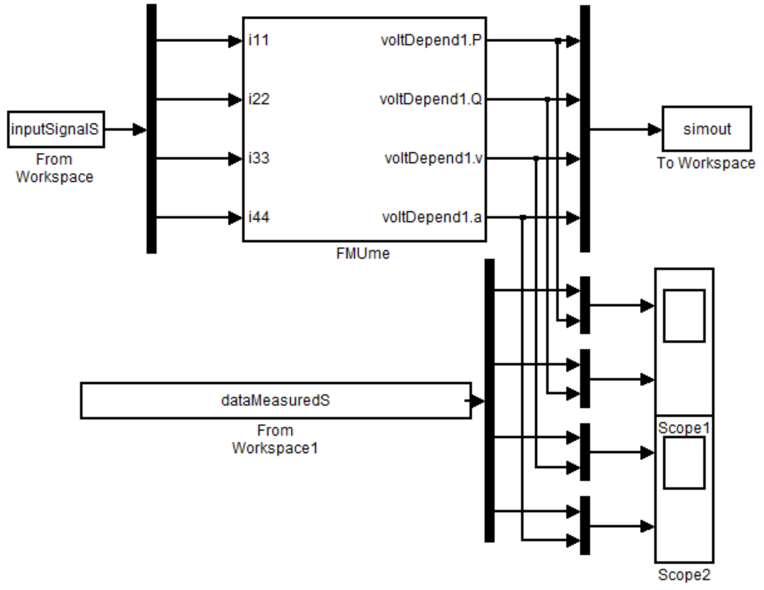
Reload FMU
- [ ] Add new output ports when the model is reloaded
- [ ] Update block name automatically

Find FMU file on Model load
- [✓ Relative path to Model directory]
- Absolute path
- File name search

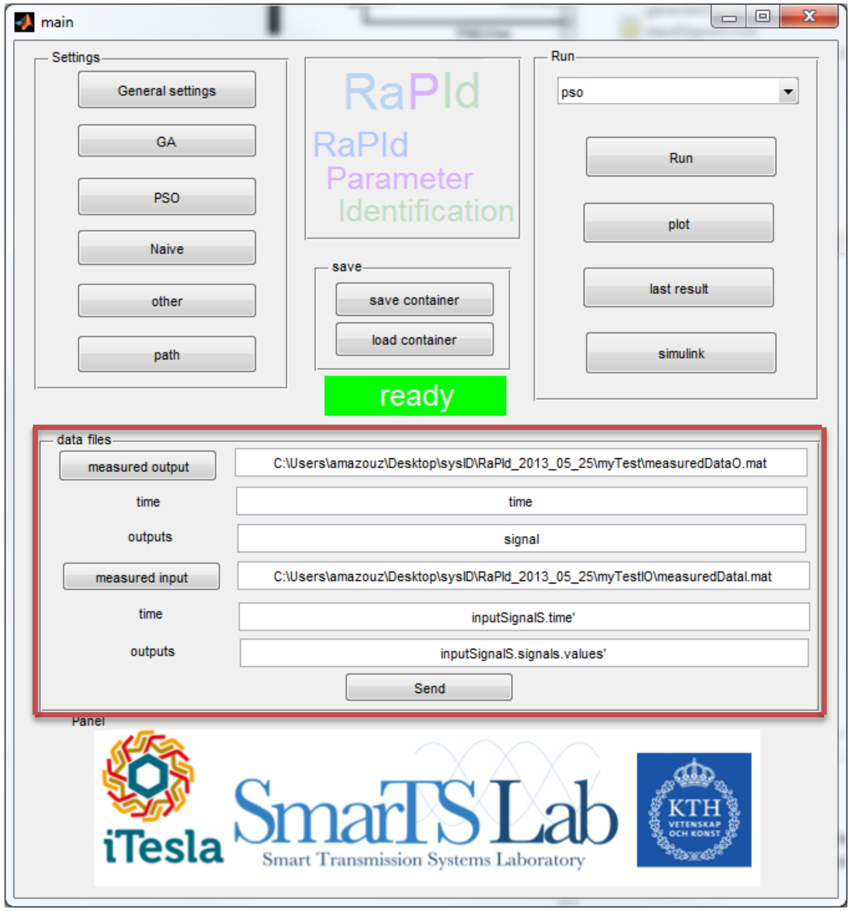
Input data

- If the Modelica model contains input ports left unconnected, they will automatically appear as inputs to the FMI block.
- In a second example included with the toolbox, a "From Workspace" is included. It loads a struct with time called InputSignalS, this struct is generated automatically by the GUI based on the data provided by the user.





User-provided data



Run: pso

data files

measured output	C:\Users\amazouz\Desktop\sysID\RaPId_2013_05_25\myTest\measuredDataO.mat
time	time
outputs	signal
measured input	C:\Users\amazouz\Desktop\sysID\RaPId_2013_05_25\myTestIO\measuredDataI.mat
time	inputSignalS.time'
outputs	inputSignalS.signals.values'

In the GUI, the user points at two files, one containing the measured Output, the second one containing the measured input.

The second one is optional and can be left empty (for output only identification).

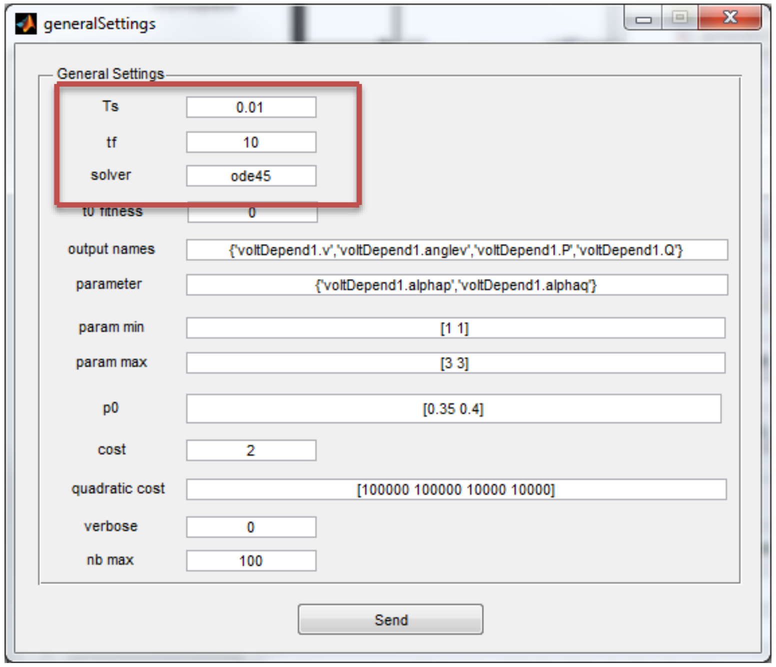
Both can also be the same file.

The ’time’ and ’output’ fields correspond to the command that should be typed to obtain the time vector as a row vector and the outputs a matrix whose rows are the different output signals.

In the example on the left, the file measuredDataO.mat contains a row vector called ’time’ and a matrix called ’signal’.

Simulink settings

- The sampling time, final time and solver to be used in simulink should not be entered in the configuration parameter of simulink but in the toolbox's general settings:



generalSettings

General Settings

Ts	0.01
tf	10
solver	ode45
t0 fitness	0
output names	{'voltDepend1.v','voltDepend1.anglev','voltDepend1.P','voltDepend1.Q'}
parameter	{'voltDepend1.alphap','voltDepend1.alphaq'}
param min	[1 1]
param max	[3 3]
p0	[0.35 0.4]
cost	2
quadratic cost	[100000 100000 10000 10000]
verbose	0
nb max	100

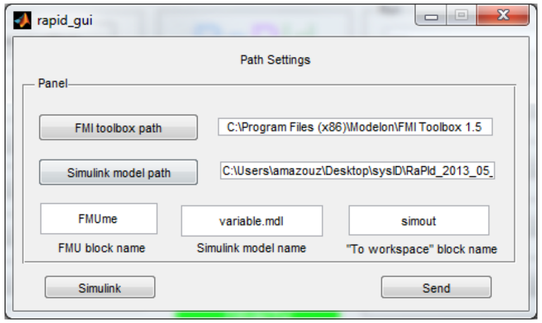
Simulink settings

- In the same menu, and in the path settings, some more information about the simulink model must be set.

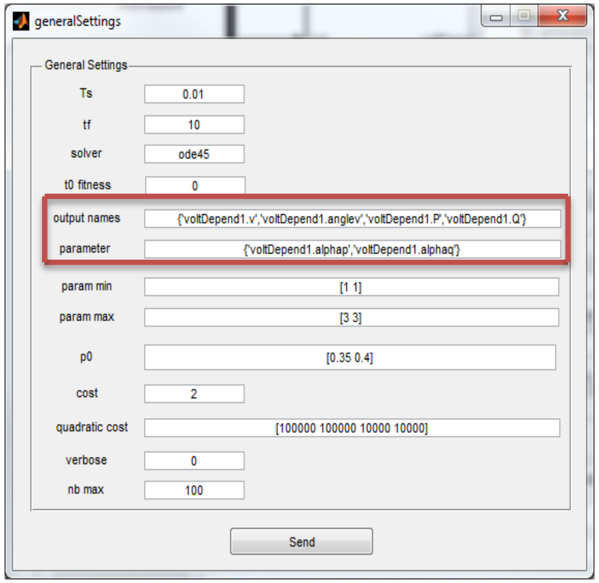
Paths to the FMI toolbox simulink model

Names of the FMU in the simulink model, name of the simulink file, name of the variable containing the output data after simulation

Complete names of the different outputs and parameters as they appear in the FMU



FMI toolbox path	C:\Program Files (x86)\Modelon\FMI Toolbox 1.5
Simulink model path	C:\Users\amazouz\Desktop\sysID\RaPId_2013_05_
FMU block name	FMUme
Simulink model name	variable.mdl
"To workspace" block name	simout



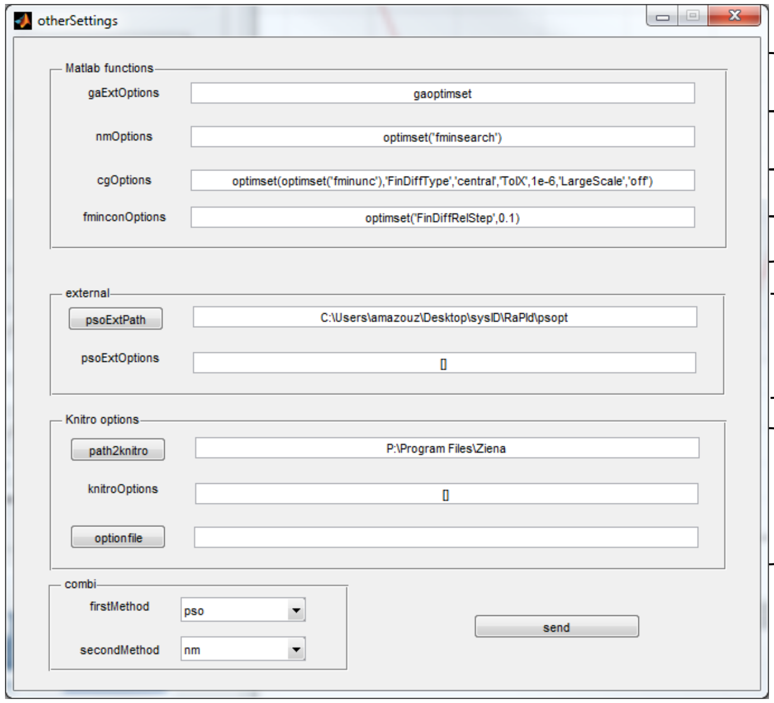
Ts	0.01
tf	10
solver	ode45
t0 fitness	0
output names	{'voltDepend1.v','voltDepend1.anglev','voltDepend1.P','voltDepend1.Q'}
parameter	{'voltDepend1.alphap','voltDepend1.alphaq'}
param min	[1 1]
param max	[3 3]
p0	[0.35 0.4]
cost	2
quadratic cost	[100000 100000 10000 10000]
verbose	0
nb max	100

Configure the methods

- Once the simulink model is set up, the different algorithms to be used need to be parametrised
- The menus entitled GA, PSO and Naive correspond three methods we implemented with the toolbox
- In the menu Other can be found the settings to the matlab methods that can be called in the toolbox
- In the main window, a drop-down menu allows selection of the methods:
- pso, ga, naive are implemented in the toolbox, the commented source code can be found in \core\functions\algoFunctions
- cg, nm, gaExt, fmincon, respectivelly call the matlab functions fminunc, fminsearch, ga, fmincon
- psoExt and knitro are based on external toolboxes that need to be installed, see https://code.google.com/p/psomatlab/ and http://www.ziena.com/knitro.htm
- Combi uses a combination of two of the methods mentioned here, they are ran one after the other

Settings the algo

- Pso, ga and naive are set-up directly in the gui.
- The external methods (these using matlab or other toolboxes' functions) are set up through the formulation of an optimset
- The comand to get to the so called optimset has to be input in the "other settings" window.



gaExtOptions	gaoptimset
nmOptions	optimset('fminsearch')
cgOptions	optimset(optimset('fminunc'),'FinDiffType','central','TolX',1e-6,'LargeScale','off')
fminconOptions	optimset('FinDiffRelStep',0.1)
psoExtPath	C:\Users\amazouz\Desktop\sysID\RaPId\psopt
psoExtOptions	[]
path2knitro	P:\Program Files\Ziena
knitroOptions	[]
optionfile	
firstMethod	pso
secondMethod	nm

ga matlab function

fminsearch matlab function

fminunc matlab function

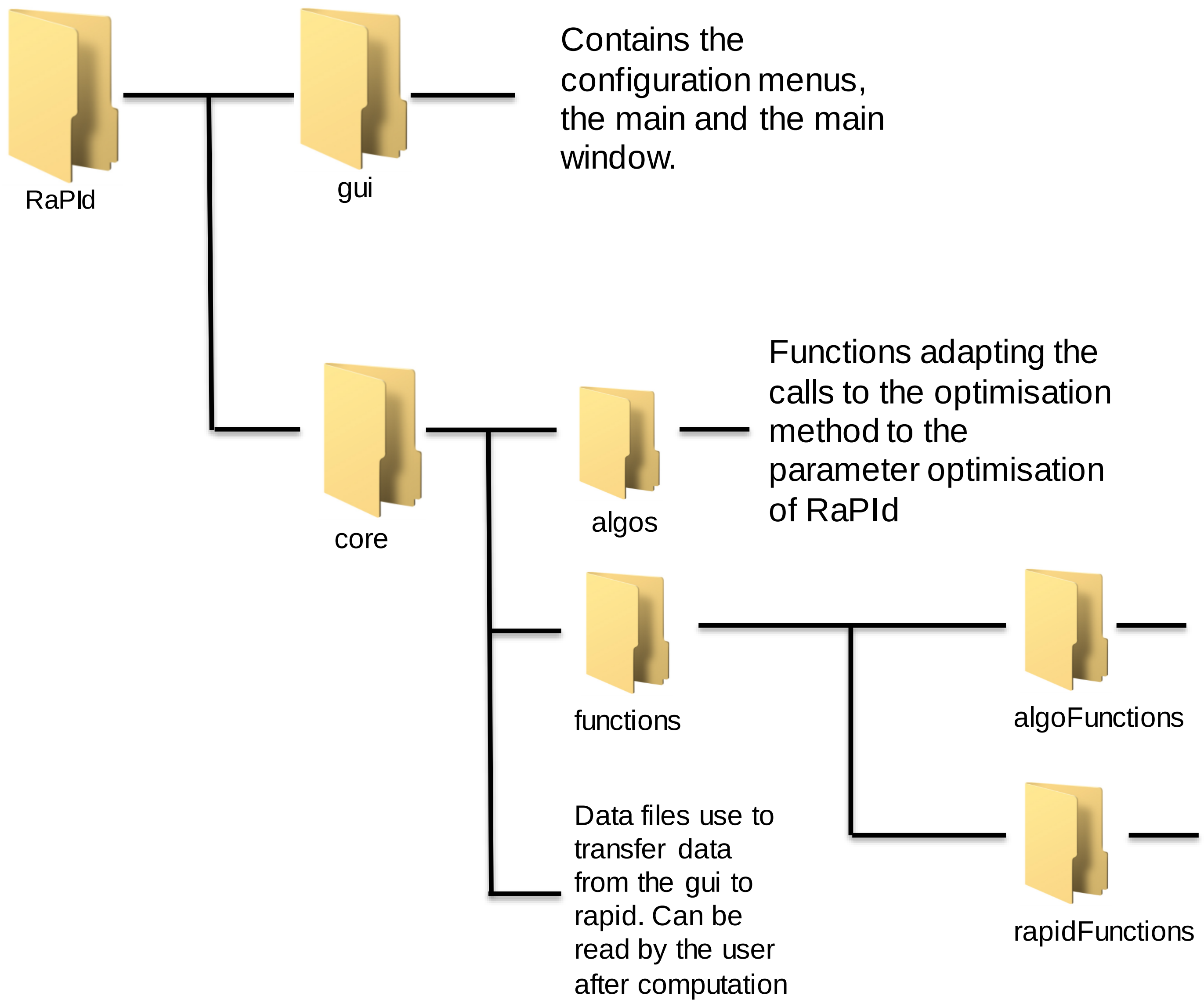
fmincon matlab function

pso function from the toolbox posomatlab

ktrlink function from the toolbox Knitro

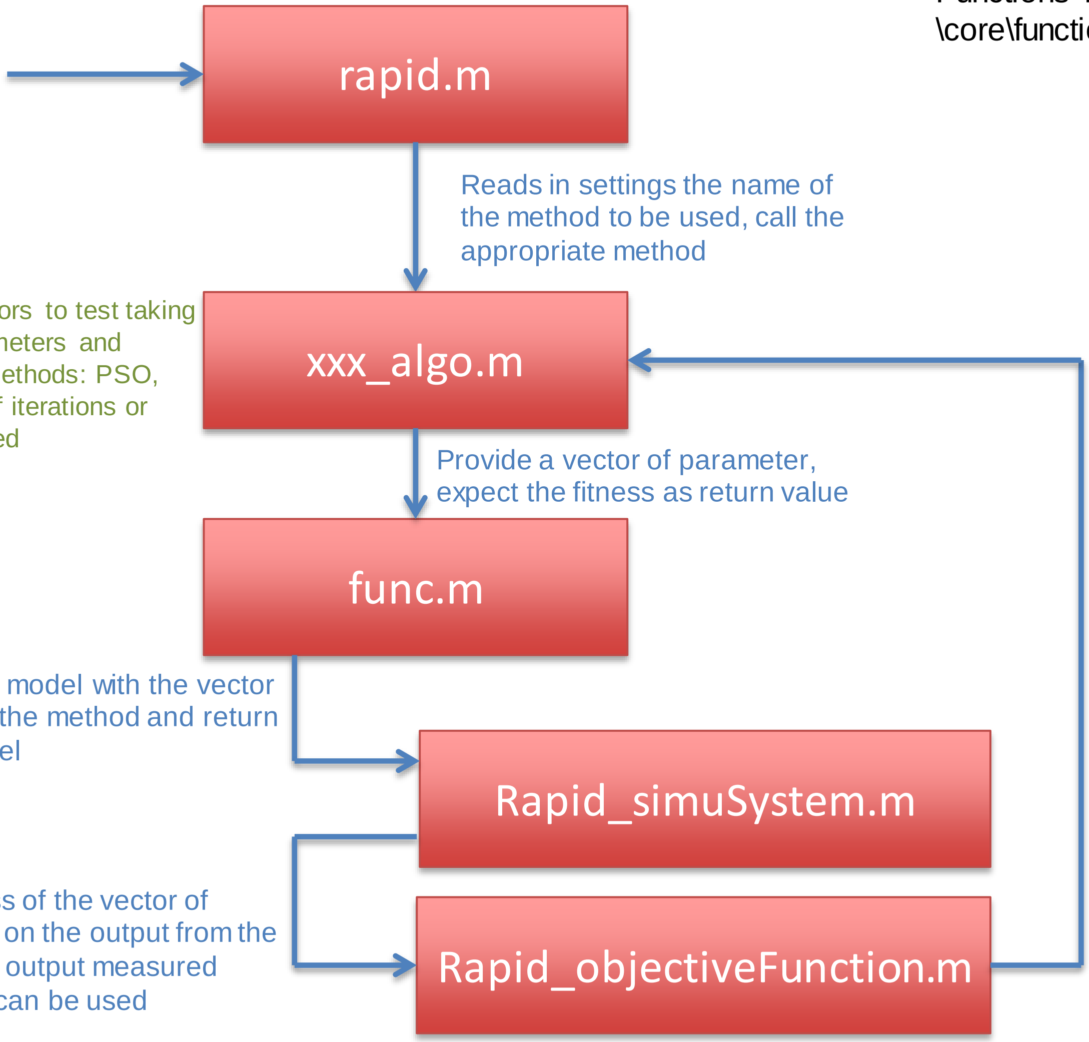
Choice of the two methods in the combination

File Structure


RaPId
gui
Contains the configuration menus, the main and the main window.
core
algos
Functions adapting the calls to the optimisation method to the parameter optimisation of RaPId
functions
algoFunctions
Functions used by the algorithm implemented with the toolbox (pso, ga, naive method)
rapidFunctions
Base functions of the toolbox
Data files use to transfer data from the gui to rapid. Can be read by the user after computation

Details of implementation

Functions from :
\core\functions\rapidFunctions



Called from the gui or the command line, needs to be fed the settings struct and the measured data

Reads in settings the name of the method to be used, call the appropriate method

Generates iteratively parameter vectors to test taking into account past values of the parameters and corresponding fitness (example of methods: PSO, GA, Gradient) stop after a number of iterations or when a condition on fitness is reached

Provide a vector of parameter, expect the fitness as return value

Simulate the simulink model with the vector of parameters fed by the method and return the output of the model

Evaluate the fitness of the vector of parameters based on the output from the simulation and the output measured Several criterions can be used

Return the fitness

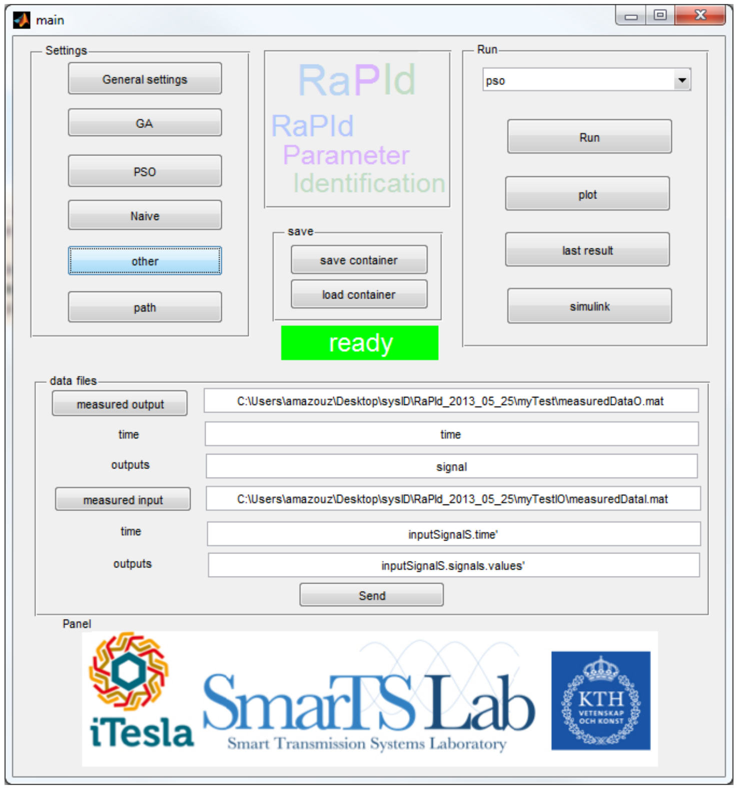
Running the toolbox

- The user should go through every single menu and make sure all the fields are correctly filled.
- A description of every field can be found in the user manual.
- For more information, every script of the toolbox was commented. Reading the code should help understanding the basic usage.

Investigating the code

- The user should go through every single menu and make sure all the fields are correctly filled.
- A description of every field can be found in the user manual.
- For more information, every script of the toolbox was commented. Reading the code should help understanding the basic usage.
- The gui only save the data entered in the menus inside a struct called settings and saves it to a file, when the ”run” from the main window is pressed, the settings object is loaded and given to the main toolbox function called ”rapid”.
- Everything can be done in command line. The gui code is quite heavy and large, doing things comand line is the right way if the user wants to dig in how things work.
- After changing directory to the RaPId directory, to run the gui, execute run_gui.m to run comand line execute run_cmd.m
- run_cmd.m can be used with the same philosophy as the gui: let it be like it is, just change the parameters present in the file according to your needs

Running the gui



Run: pso

measured output	C:\Users\amazouz\Desktop\sysID\RaPId_2013_05_25\myTest\measuredDataO.mat
time	time
outputs	signal
measured input	C:\Users\amazouz\Desktop\sysID\RaPId_2013_05_25\myTestIO\measuredDataI.mat
time	inputSignalS.time'
outputs	inputSignalS.signals.values'

When all menus are filled, press Run.

The green ”ready” turns to red. If it goes back to green, the simulation has ended, you can press plot to visualize the simulated and measured outputs.

Press last result to see what were the values found for the parameters.

If a problem occurs, the green space turns to yellow. The message displayed in the console should help finding what went wrong. Most likely a parameter was not entered correctly. *Nothing in the toolbox checks that the parameters you entered are valid.*

Save container allows you to save all the settings and the last solution (if available) in a file that can be loaded later. When a setting is changed. The last result is erased and the settings struct is modified irreversibly. It’s good to save a container every time a new kind of simulation is performed.

Adding an objective function

```
function fitness = rapid_objectiveFunction( realRes, simuRes,settings)
%OBJECTIVEFUNCTION compute quadratic criterion
%   realRes is data used for matching, from measurments
%   simuRes is the data we want to have matching realRes
%   the two of them are under the shape [[x[1];y[1]] [x[2];y[2]] ...
[x[nStep];y[nStep]]]
%
%   settings has to contain the fields
%       cost: integer, for quadratic cost the value is 2
%       objective: struct adapted to the method chose
%           if cost = 2, objective contains the field:
%               Q: nb_parameters*nb_parameters weight matrix, should take
%               into account the respective outputs scaling

switch settings.cost
    case 2
        nStep = size(realRes,2);
        nx = size(settings.objective.Q,1);
        Q_ = repmat({settings.objective.Q},1,nStep);
        Q_ = blkdiag(Q_{:});
        fitness = (realRes(:)-simuRes(:))'*Q_*(realRes(:)-simuRes(:));
    otherwise
        errorWrongCost
end

end
```

Here the fitness is characterised by one unique matrix Q : $\varphi = \Sigma_k (y[k] - y_{mes}[k])^T Q (y[k] - y_{mes}[k])$

1. In rapid_objectiveFunction, choose a keyword: case 2. Add a case to the switch, in the body of the case, compute the fitness of the vector of parameters based on realRes and simuRes, matrices whose rows are respectivelly the measured output signals and simulated output signals.
2. realRes is in the toolbox an interpolation of the result of the simulation at the time instants given by the measurement data. This allows you to compare measured and simulated data point to point.
3. *Your method will be used for fitness computation if the approriate keyword is given in* ***settings.cost***
4. You can define as many elements as needed inside **settings.objective**, in the example here a weight matrix is provided in settings.objective

Adding an optimization method

```
function [ sol, historic ] = rapid( settings, systemDescription )
%RAPID main function of the rapid toolbox RaPId

global nbIterations

nbIterations = 0;

% Set-up the simulink model to run between 0 and tf with a step of Ts and
% with the solver given by settings.methodName, the function
% rapid_simuSystem call the simulation in the workspace of rapid.m after
% changing the parameters value through fmuSetValueSimulink
addpath(settings.rapidFolder);
settings.realData = rapid_interpolate(systemDescription,settings);

% choose and launch the desired method for parameter estimation
switch settings.methodName
    case 'pso'
        [ sol, historic, settings ] = pso_algo(settings);
    case 'ga'
        [ sol, historic, settings ] = ga_algo(settings);
    case 'naive'
        sol = naive_algo(settings);
        historic = [];
    case 'cg'
        [ sol, historic, settings ] = cg_algo(settings);
    case 'nm'
        [ sol, historic, settings ] = nm_algo(settings);
    case 'combi'
        settings2 = settings;
        settings2.methodName = settings2.combiOptions.firstMethod;
        [sol1,historic1, settings2] = rapid(settings2,systemDescription);
        settings2.methodName = settings2.combiOptions.secondMethod;
        settings2.p0 = sol1;
        [sol,historic2, settings] = rapid(settings2,systemDescription);
        historic.sol1 = sol1;
        historic.historic1 = historic1;
        historic.historic2 = historic2;
    case 'psoExt'
        [ sol, historic, settings ] = psoExt_algo(settings);
    case 'gaExt'
        [ sol, historic, settings ] = gaExt_algo(settings);
    otherwise
        errorWrongMethodName
end



end
```

Add a case with a new keyword in the function rapid.m
Call your method *methodName_algo*, place it in . *It takes* the struct *settings* as argument and return the vector solution, an historic containing the data you wish to have as output (useful for debugging) and the setting struct in which you might have added data. (The methods of the toolbox add to the struct the parameter vector as settings.xsol)

Your method can only call the function func*(v)*
func takes a vector of parameter and returns the fitness associated to this vector.

This assumes that the good settings.cost and settings.objective were declared to use the objective function you desire to evaluate.

func care of simulating the model, saving its output, interpolating the output data and computing the fitness. All fields of the settings struct must however be filled correctly.

Adding an optimization method

What you need to write:

```
function [ sol, other ] = yourMethod_algo(settings)
%KNITRO_ALGO applying ktrlink from the KNITRO toolbox to compute the minimum of the
objective
%function defined by the parameter identification problem
%   settings must have as a field the initial guess for the parameter
%   vector p0, lower and upper bound for the vector of parameters and possibly
%   options generated by optimset
    options = settings.yourMethod_options;
    sol = yourMethod(@func,settings.p0,settings.p_min,settings.p_max,[],options);
end
```

Call to your function

1) The function may output any number of parameters, you can store them in other, they will be saved and accessible after computation
2) If your function is on the main folder of RaPid, nothing needs to be done for the path management

The *settings* struct

```
settings =

           rapidFolder: 'C:\Users\amazouz\Desktop\sysID\RaPId'
    path2simulinkModel: 'C:\Users\amazouz\Desktop\sysID\RaPId\model\'
       path2fmiToolbox: 'C:\Program Files (x86)\Modelon\FMI Toolbox 1.4.4'
             modelName: 'test'
             blockName: 'test/FMUme'
             scopeName: 'data_sim'
                    Ts: 0.1000
                    tf: 0.2000
            fmuOutData: {'voltDepend1.v'  'voltDepend1.anglev'  'voltDepend1.P'  'voltDepend1.Q'}
        parameterNames: {'voltDepend1.alphap'  'voltDepend1.alphaq'}
           plotOutputs: 1
                 p_min: [-1 -1]
                 p_max: [3 3]
               verbose: 1
                  cost: 2
             objective: [1x1 struct]
             intMethod: 'ode1'
            ga_options: [1x1 struct]
           pso_options: [1x1 struct]
         naive_options: [1x1 struct]
          gaExtOptions: []
             nmOptions: []
             cgOptions: [1x1 struct]
            psoExtPath: 'C:\Users\amazouz\Desktop\sysID\RaPId\psopt'
         psoExtOptions: [1x1 struct]
           combiOptions: [1x1 struct]
            methodName: 'pso'
                    p0: [1 1]
                 dataT: 't'
                 dataY: 'y'
             path2data:
'C:\Users\amazouz\Desktop\sysID\RaPId\\dataMeasured.mat'
               strings: [1x1 struct]
              realData: [4x3 double]
```

Characterise the simulation, the measured data will also be interpolated to fit this specification

Used to label the plots

Restrict the values allowed to the vector of parameters

Objective function identifier and struct containing it's parameters

Settings specific to each method

Determine the method to be used

Initial guess, used for example for NM or CG

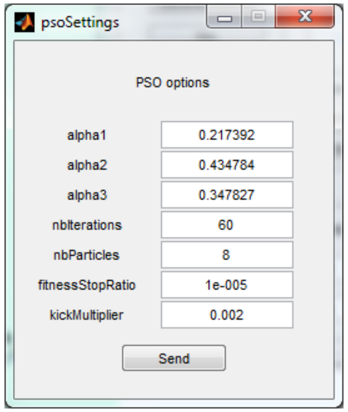
Some gui string generated automatically

Data put here automatically after interpolation and filtering

Path and variable names needed to locate the simulink model and the file data.mat used to store and communicate data in between the gui's components

The *settings* struct

- The content of *settings* is either set manually or through the gui:

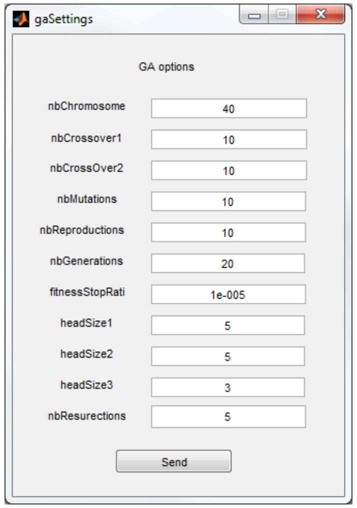


psoSettings

PSO options

alpha1	0.217392
alpha2	0.434784
alpha3	0.347827
nbIterations	60
nbParticles	8
fitnessStopRatio	1e-005
kickMultiplier	0.002

```
settings.pso_options
ans =

             alpha1: 0.2174
             alpha2: 0.4348
             alpha3: 0.3478
              limit: 60
       nb_particles: 8
   fitnessStopRatio: 1.0000e-005
    kick_multiplier: 0.0020
```



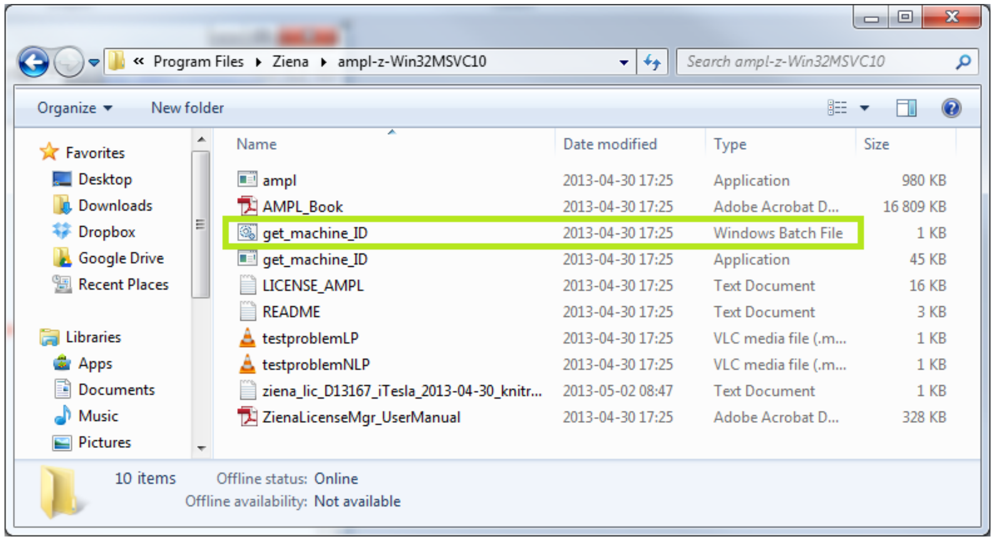
gaSettings

GA options

nbChromosome	40
nbCrossover1	10
nbCrossOver2	10
nbMutations	10
nbReproductions	10
nbGenerations	20
fitnessStopRati	1e-005
headSize1	5
headSize2	5
headSize3	3
nbResurections	5

```
settings.ga_options

ans =

       nbCromosomes: 40
      nbCroossOver1: 10
      nbCroossOver2: 10
        nbMutations: 10
     nbReproduction: 10
              limit: 20
   fitnessStopRatio: 1.0000e-005
          headSize1: 5
          headSize2: 5
          headSize3: 3
      nbReinjection: 5
```

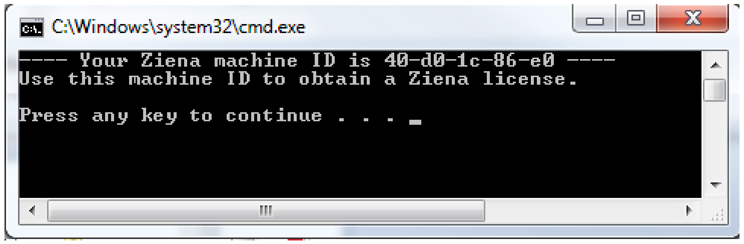
KNITRO: license installation



Name	Date modified	Type	Size
ampl	2013-04-30 17:25	Application	980 KB
AMPL_Book	2013-04-30 17:25	Adobe Acrobat D...	16 809 KB
get_machine_ID	2013-04-30 17:25	Windows Batch File	1 KB
get_machine_ID	2013-04-30 17:25	Application	45 KB
LICENSE_AMPL	2013-04-30 17:25	Text Document	16 KB
README	2013-04-30 17:25	Text Document	3 KB
testproblemLP	2013-04-30 17:25	VLC media file (.m...	1 KB
testproblemNLP	2013-04-30 17:25	VLC media file (.m...	1 KB
ziena_lic_D13167_iTesla_2013-04-30_knitr...	2013-05-02 08:47	Text Document	1 KB
ZienaLicenseMgr_UserManual	2013-04-30 17:25	Adobe Acrobat D...	328 KB

10 items Offline status: Online
Offline availability: Not available

In the knitro folder (that you either took from the toolbox or downloaded), there is a get_machine_ID batch file.
Run it, you get this:

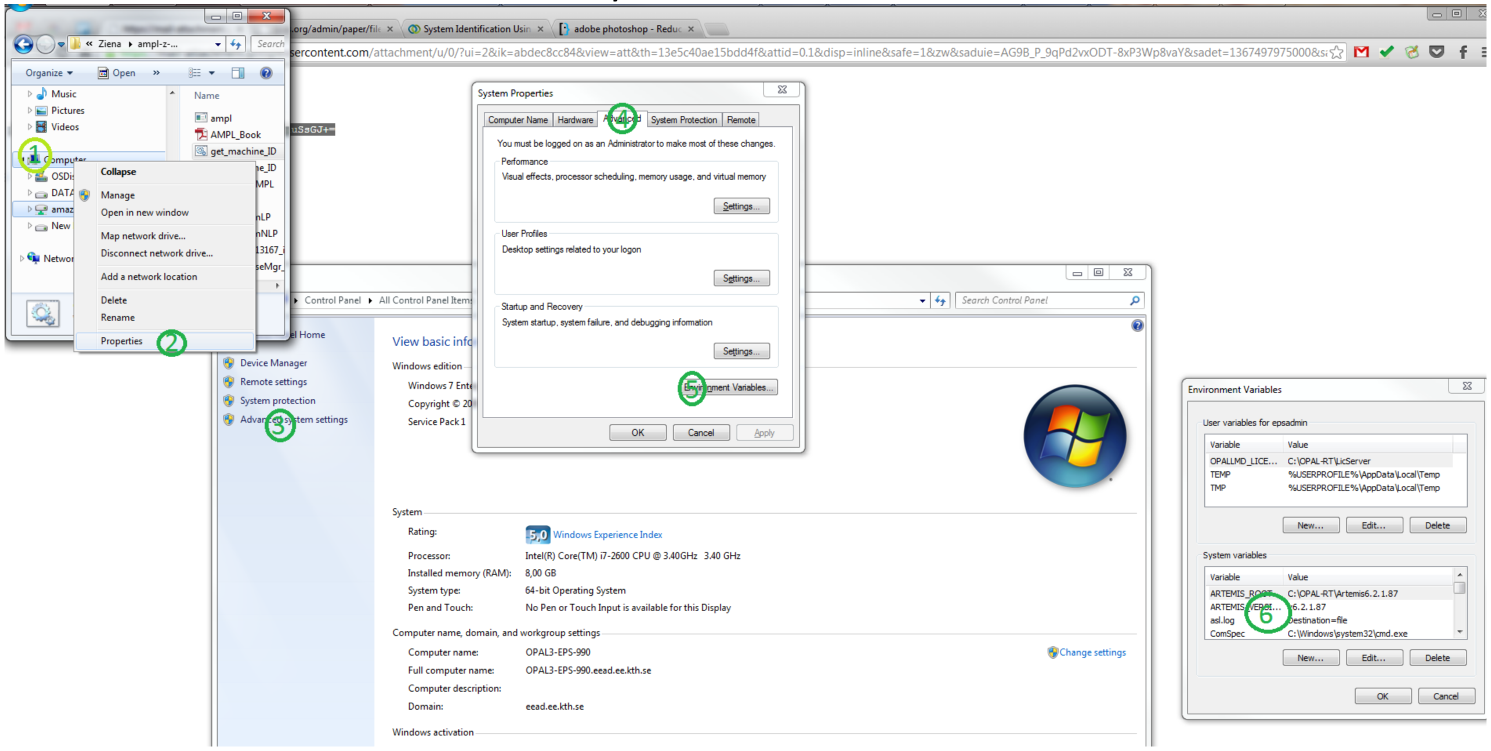


```
---- Your Ziena machine ID is 40-d0-1c-86-e0 ----
Use this machine ID to obtain a Ziena license.

Press any key to continue . . .
```

The picture needs to be sent to :
thibaut.brejon@artelys.com

KNITRO

- This guy at Artleys will send you a *.txt containing something funky
- E.g. : =Pblaetle19HUlZwSGZkSczyHMNRRBj-MF2yzQHVriPpnuSsGJ+=
- We need to change two environment variables
  - rightClickOnComputer>Properties>AdvancedSystemSettings>loginAsAdministrator>Advanced>EnvironmentVariables>SystemVariables



You must be logged on as an Administrator to make most of these changes.

Performance: Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles: Desktop settings related to your logon

Startup and Recovery: System startup, system failure, and debugging information

System

Rating: 5,0 Windows Experience Index
Processor: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz 3.40 GHz
Installed memory (RAM): 8,00 GB
System type: 64-bit Operating System
Pen and Touch: No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name: OPAL3-EPS-990
Full computer name: OPAL3-EPS-990.eead.ee.kth.se
Computer description:
Domain: eead.ee.kth.se

Windows activation

User variables for epsadmin

Variable	Value
OPALLMD_LICE...	C:\OPAL-RT\LicServer
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

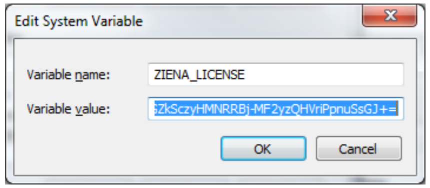
System variables

Variable	Value
ARTEMIS_ROOT	C:\OPAL-RT\Artemis6.2.1.87
ARTEMIS_VERSI...	v6.2.1.87
asl.log	Destination=file
ComSpec	C:\Windows\system32\cmd.exe

iTesla
Innovative Tools for Electrical System Security within Large Areas

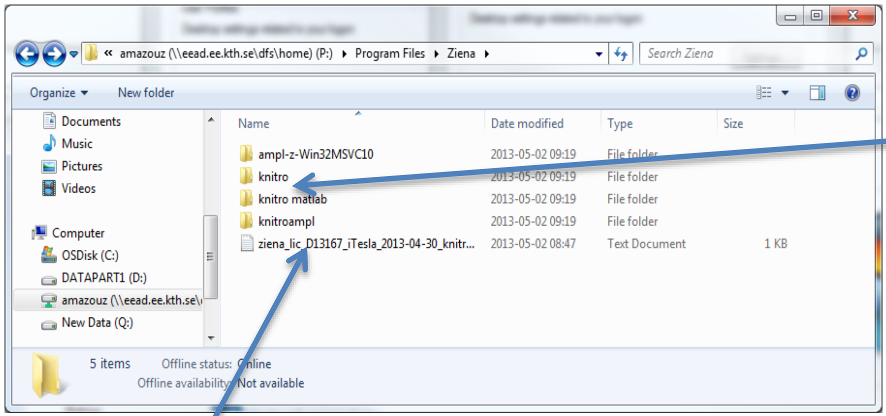
KNITRO

- Add the content of the file you received as ZIENA_LICENSE


Edit System Variable

Variable name: ZIENA_LICENSE
Variable value: ;ZkSczyHMNRRBj-MF2yzQHVriPpnuSsGJ+=

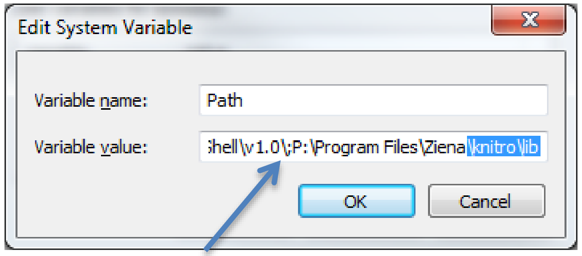
- Add the path to the folder \knitro\lib to the Path variable


« amazouz (\\eead.ee.kth.se\dfs\home) (P:) ▸ Program Files ▸ Ziena ▸

Name	Date modified	Type	Size
ampl-z-Win32MSVC10	2013-05-02 09:19	File folder	
knitro	2013-05-02 09:19	File folder	
knitro matlab	2013-05-02 09:19	File folder	
knitroampl	2013-05-02 09:19	File folder	
ziena_lic_D13167_iTesla_2013-04-30_knitr...	2013-05-02 08:47	Text Document	1 KB

5 items Offline status: Online
Offline availability: Not available

”knitro” is the one of the 4 folders
should have downloaded and un
somewhere
In my case: P:\Program Files\Zie


Edit System Variable

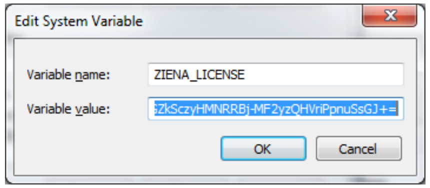
Variable name: Path
Variable value: ;hell\v1.0\;P:\Program Files\Ziena\knitro\lib

This is the license file they sent

Notice the semicolon
Don’t erase the line, just add to it

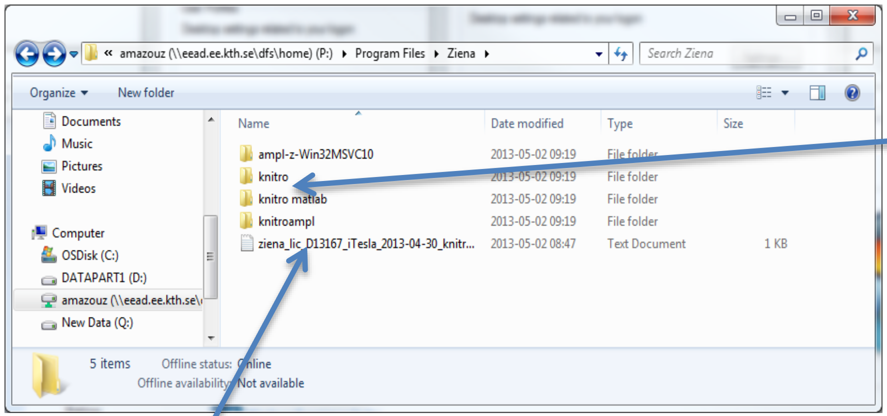
KNITRO

- Add the content of the file you received as ZIENA_LICENSE



Variable name: ZIENA_LICENSE
Variable value: ZkSczyHMNRRBj-MF2yzQHVriPpnuSsGJ+=

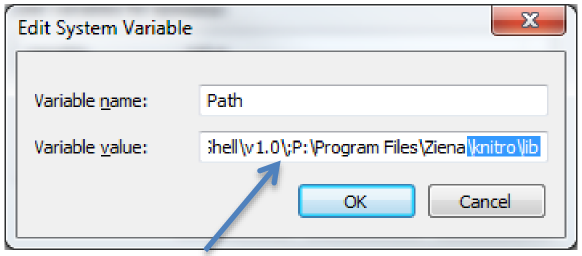
- Add the path to the folder \knitro\lib to the Path variable



Name	Date modified	Type	Size
ampl-z-Win32MSVC10	2013-05-02 09:19	File folder	
knitro	2013-05-02 09:19	File folder	
knitro matlab	2013-05-02 09:19	File folder	
knitroampl	2013-05-02 09:19	File folder	
ziena_lic_D13167_iTesla_2013-04-30_knitr...	2013-05-02 08:47	Text Document	1 KB

5 items Offline status: Online
Offline availability: Not available

”knitro” is the one of the 4 folders
should have downloaded and un
somewhere
In my case: P:\Program Files\Zie



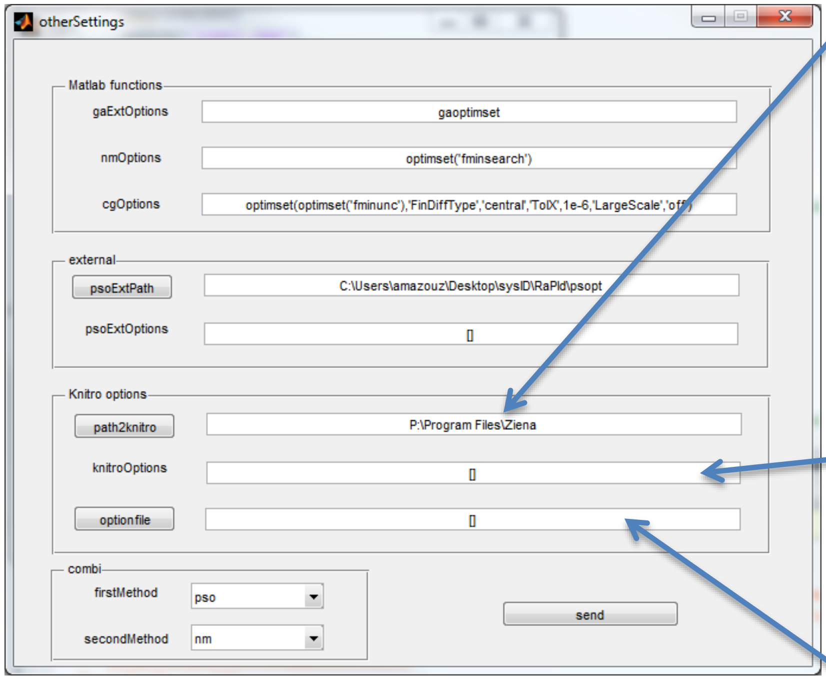
Variable name: Path
Variable value: shell\v1.0\;P:\Program Files\Ziena\knitro\lib

This is the license file they sent

Notice the semicolon
Don’t erase the line, just add to it

KNITRO

- The toolbox asks for a path2knitro in ”other”



gaExtOptions: gaoptimset
nmOptions: optimset('fminsearch')
cgOptions: optimset(optimset('fminunc'),'FinDiffType','central','TolX',1e-6,'LargeScale','off')
psoExtPath: C:\Users\amazouz\Desktop\sysID\RaPId\psopt
psoExtOptions: []
path2knitro: P:\Program Files\Ziena
knitroOptions: []
option file: []
firstMethod: pso
secondMethod: nm

You should select the folder ”knitro” that you downloaded

You can give something of the kind optimset(’parameter’,’value’)

Name of a file, that you add yourself to the path or, better, put in the Knitro folder, the kind of file is given in the documentation of KNITRO

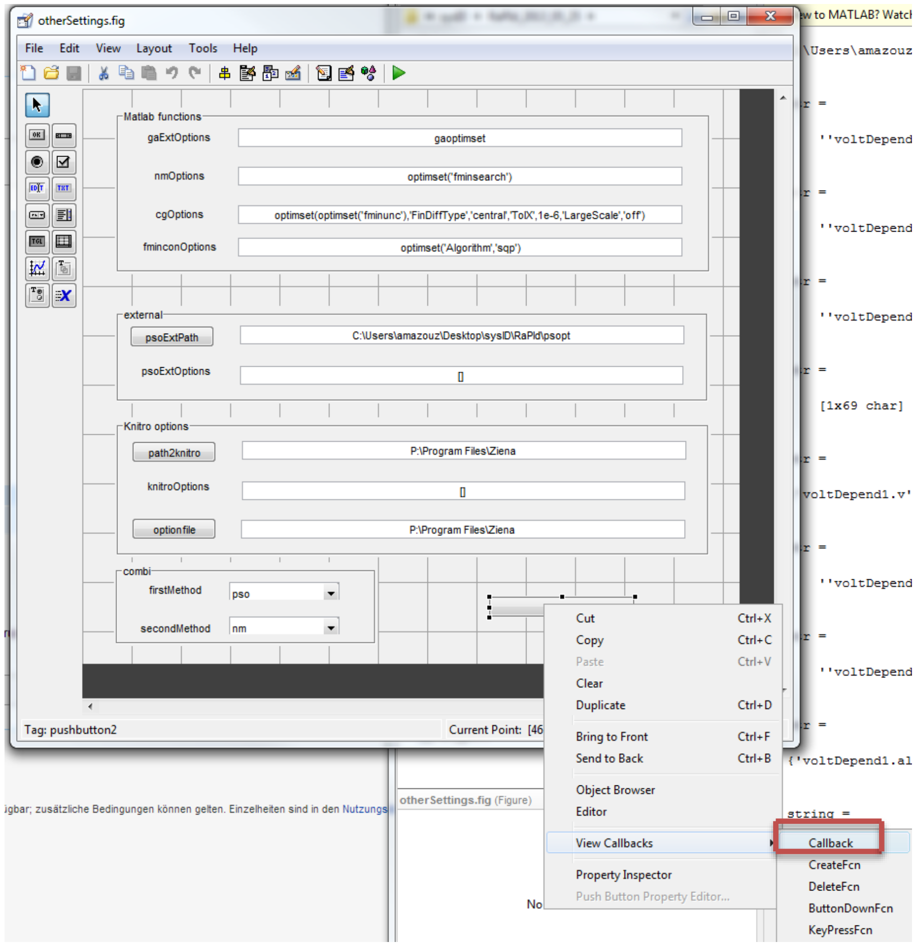
Some words on the GUI

- A file named *data.mat* is used to communicate data to the different elements of the gui and finally to the toolbox

- data.mat contains :
  - The struct *settings*
  - *sol ,* the output of the toolbox at the last use

- **Inside the GUI:** a gui element (a window) is
  - Characterised by fields with an initial value and,
  - Read the file – fill in window: Automatically initialised in the function *windowName_OpeningFcn*
    - *This function reads the settings struct stored in core\data.mat and copy the values of parameters inside the window's fields*
  - Read the window – fill in the file: Updated and closed in the call back of a push button (send or next) pushbuttoni_Callback
    - This does the opposite of an *OpeningFcn, it reads the content of the window, save it in the settings struct and saves the struct in core\data.mat*

GUI



Matlab functions
gaExtOptions gaoptimset
nmOptions optimset('fminsearch')
cgOptions optimset(optimset('fminunc'),'FinDiffType','central','TolX',1e-6,'LargeScale','off')
fminconOptions optimset('Algorithm','sqp')

external
psoExtPath C:\Users\amazouz\Desktop\sysID\RaPId\psopt
psoExtOptions []

Knitro options
path2knitro P:\Program Files\Ziena
knitroOptions []
optionfile P:\Program Files\Ziena

combi
firstMethod pso
secondMethod nm

- To summarize, the GUI code operates following the scheme:
  - At initialisation, the *data.mat* file is loaded, the *settings* struct is read and the different fields of the windows are filled with the corresponding values in the struct
  - Upon press event on the *send* button, all the fields of the GUI are read and the settings struct is updated with the new values

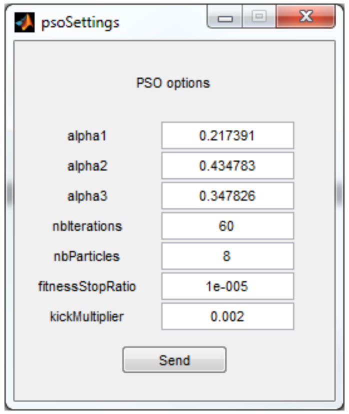
- If you want to study what is done in a particular window, open the fig related to this window inside GUIDE (graphical GUI editor) right click on the element you would like to control and click on the callback, this will open the part of the script launched upon interraction with the investigated element (for a button, when the user presses the button…)

iTesla
Innovative Tools for Electrical System Security within Large Areas

Some words on the GUI

Example:

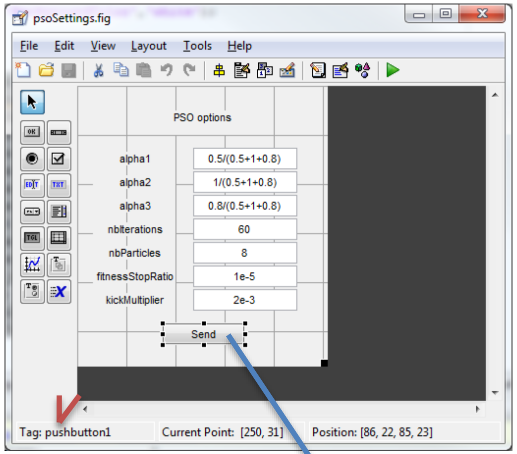
One of the option windows



PSO options

alpha1	0.217391
alpha2	0.434783
alpha3	0.347826
nbIterations	60
nbParticles	8
fitnessStopRatio	1e-005
kickMultiplier	0.002

In the GUIDE Graphical editor

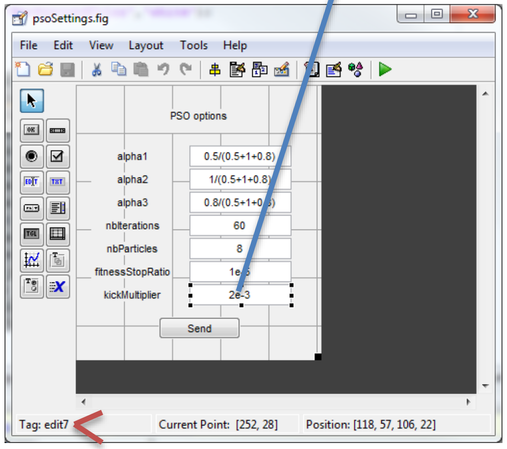


PSO options

alpha1	0.5/(0.5+1+0.8)
alpha2	1/(0.5+1+0.8)
alpha3	0.8/(0.5+1+0.8)
nbIterations	60
nbParticles	8
fitnessStopRatio	1e-5
kickMultiplier	2e-3

Tag: pushbutton1 Current Point: [250, 31] Position: [86, 22, 85, 23]

Clicking on an element will show you its name



PSO options

alpha1	0.5/(0.5+1+0.8)
alpha2	1/(0.5+1+0.8)
alpha3	0.8/(0.5+1+0.8)
nbIterations	60
nbParticles	8
fitnessStopRatio	1e-5
kickMultiplier	2e-3

Tag: edit7 Current Point: [252, 28] Position: [118, 57, 106, 22]

Some words on the GUI

Initialisation:

```
% --- Executes just before psoSettings is made visible.
function psoSettings_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to psoSettings (see VARARGIN)

% Choose default command line output for psoSettings
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes psoSettings wait for user response (see UIRESUME)
% uiwait(handles.figure1);
load('core\data.mat');
if isfield(settings,'pso_options')
    set(handles.edit1,'String',settings.pso_options.alpha1);
    set(handles.edit2,'String',settings.pso_options.alpha2);
    set(handles.edit3,'String',settings.pso_options.alpha3);
    set(handles.edit4,'String',settings.pso_options.limit);
    set(handles.edit5,'String',settings.pso_options.nb_particles);
    set(handles.edit6,'String',settings.pso_options.fitnessStopRatio);
    set(handles.edit7,'String',settings.pso_options.kick_multiplier);
end
```

Each of the text fields is given the value of the corresponding parameter in the *settings* struct

Some words on the GUI

Update:

This function is called when the button is clicked

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
pso_options.alpha1 = eval(get(handles.edit1,'String'));
pso_options.alpha2 = eval(get(handles.edit2,'String'));
pso_options.alpha3 = eval(get(handles.edit3,'String'));
pso_options.limit = eval(get(handles.edit4,'String'));
pso_options.nb_particles = eval(get(handles.edit5,'String'));
pso_options.fitnessStopRatio = eval(get(handles.edit6,'String'));
pso_options.kick_multiplier = eval(get(handles.edit7,'String'));
load('core\data.mat');
settings.pso_options = pso_options;
save('core\data.mat','settings');
close(gcf)
```

The values from the GUI are copied in the settings struct

The data file is loaded

The new data is appended to the loaded struct

The data is saved at the same emplacement

Bug list

- If you give to the GUI bad names for the parameters to fetch in the simulink diagram, FMItoolbox will make simulink shut down without notice
- Some jpeg files are included in the main window, warnings are displayed in console…
- The fmi toolbox sometimes requires the FMU block to be re-initialised. If that’s the case the model won’t simulate even out of the toolbox. When this happens, we usually need to restart matlab.
- In the simulink model, if a variable step method is used, set manually a value to simulation>configuration parameters>solve> Max Step Size, otherwise matlab complains a lot in the console

Advanced

EXTRA INFO FOR DEVELOPERS AND ADVANCED USERS

Adding new functionalities

- If a new functionality comes with settings to be parametrised or has to be launched by the user (to display the value of a parameter for example), a new window has to be written.
- The files main.fig and main.m have to be edited to call this new window
- If any data needs to be stored, be sure to do it in the struct settings in core\data.mat
- Here every window looks into the settings struct to set it’s initial values, if you load a different object data.mat, the setting windows will be updated automatically and the next time you run the optimisation the parameters of the new file are used and not some defaults
- If you delete or empty data.mat, the interface will be filled with default values, be sure to go through every menus and set good parameters everywhere

Getting Technical

- If you want to get technical. Use the /core/ files. Show the scripts:
- <open the script run_cmd.m>
  - in run_cmd, all parameters are declared and given a value. They are all stored in a struct called settings. This struct is provided to the main function of the toolbox rapid.
- <open the script core\functions\rapid.m>
  - this script contains a switch which choses a xxx_algo.m script to run according to the value of settings.methodName

Getting Technical

- <open fmincon_algo.m>
  - in the xxx_algo.m script, we adapt the call to the algorithm function to the toolbox's use. To do that we use the function func.m which yields the fitness of the parameter vector given in input, This is the function of several variables to be minimised in R
- <open func.m>
  - this function calls two of the rapid functions
- <show rapid_simuSystem.m>
  - Sets the values of the parameters that are currently tested by the algorithm, inside the fmu and simulates the system, it takes as an output the result of the simulation

Getting Technical

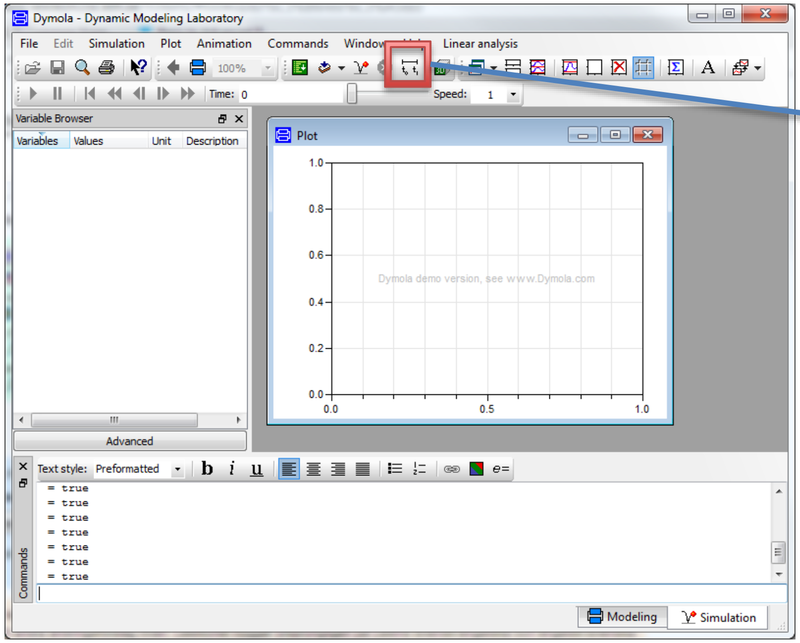
- <show rapid_objectiveFunction>
  - this result is used for the computation of the objective function.
  - Before the computation, the simulated output data is interpolated at the time instant of the measured output data using rapid_interpolate.m and then the fitness is computed according to the method chosen through the switch.
  - Then you show the diagram in the slides to see were is the reccursion.

- **The files contain quite a bit of comments. Somebody who has a some experience programming should spend a bit of time thinking about it looking at the files.**

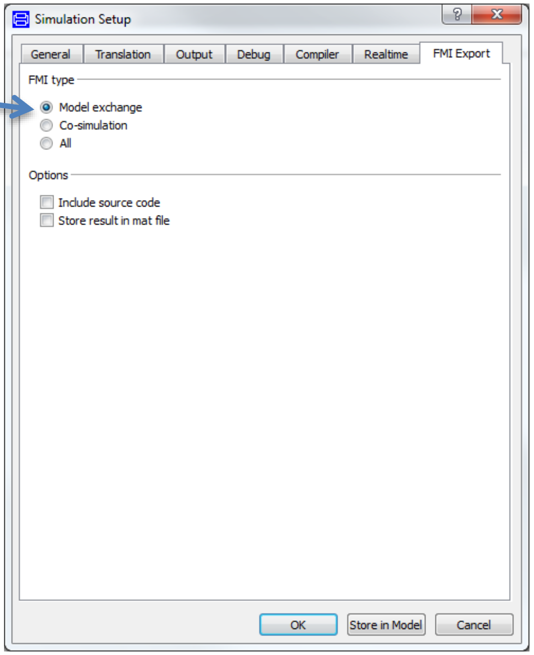
Advanced

OPTIONS FOR GENERATING FMUS

Generate the *fmu from Dymola



```
= true
= true
= true
= true
= true
= true
= true
```



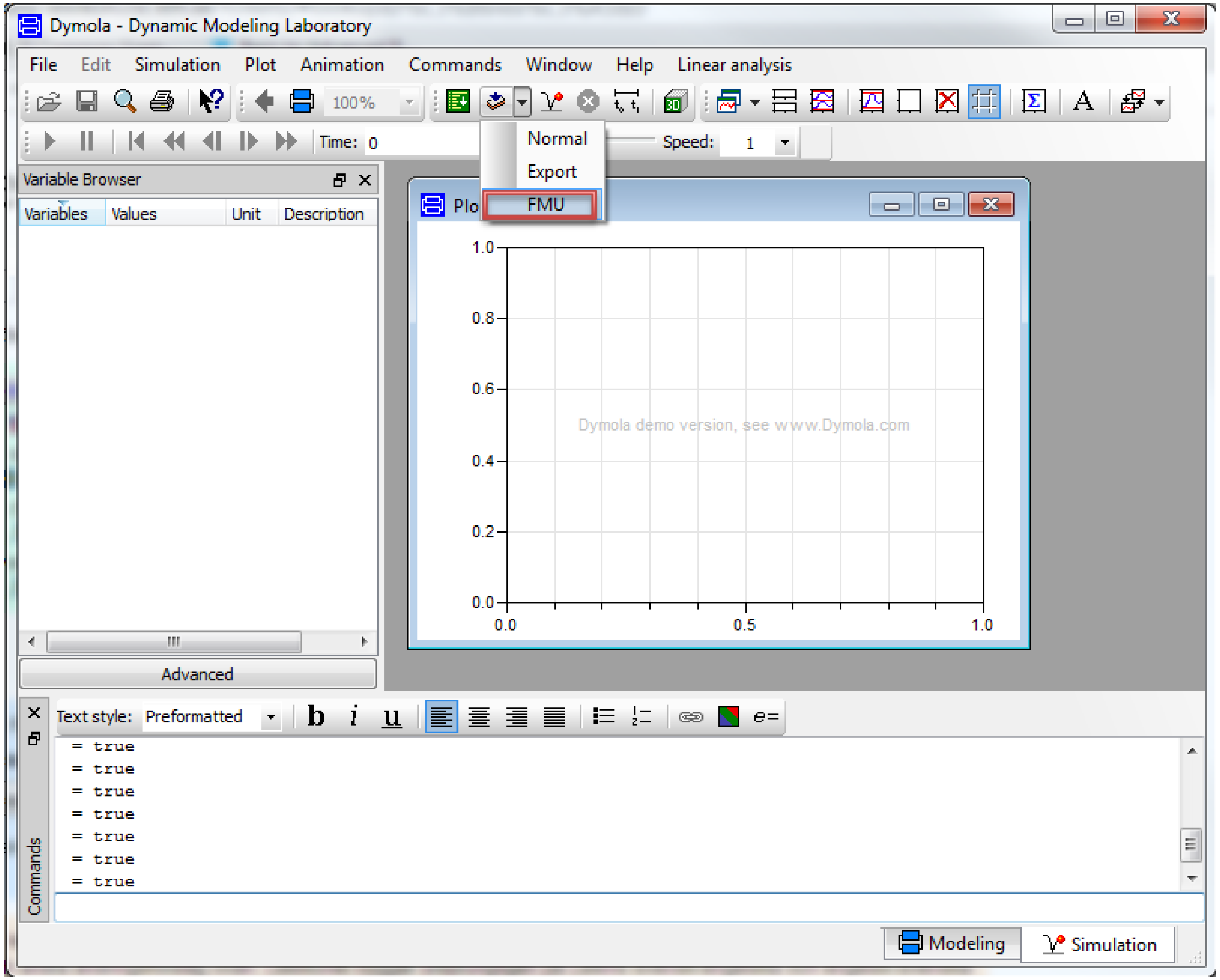
FMI type

[✓ Model exchange] Co-simulation All

Options

- [ ] Include source code
- [ ] Store result in mat file

Generate the *fmu from Dymola


Dymola - Dynamic Modeling Laboratory
File Edit Simulation Plot Animation Commands Window Help Linear analysis
Normal
Export
FMU
Time: 0
Speed: 1
Variable Browser
Variables Values Unit Description
Advanced
Dymola demo version, see www.Dymola.com
Text style: Preformatted
Commands
Modeling
Simulation

```
= true
= true
= true
= true
= true
= true
= true
```

Optional – Advanced

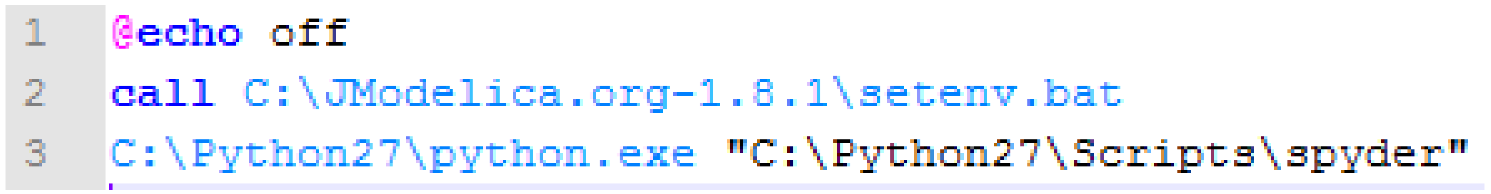
Generate the FMU from JModelica.org

- 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` ***commnad used by the DFIG WT is not support -> no FMU can be generated)***
- We illustrate one example here, carried out with JModelica.org-1.8.1

Optional – Advanced

Generate the FMU from JModelica.org

- Setting 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 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:

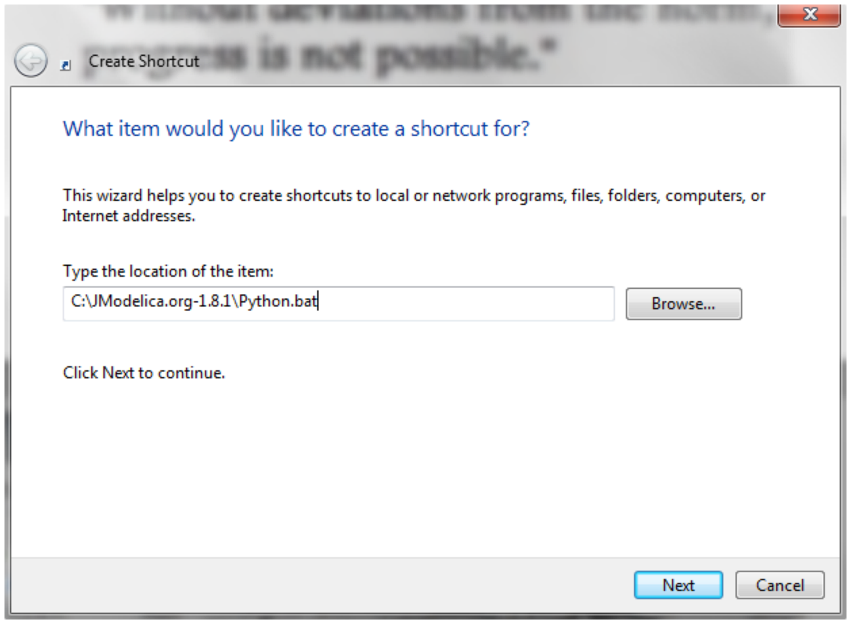


```
@echo off
call C:\JModelica.org-1.8.1\setenv.bat
C:\Python27\python.exe "C:\Python27\Scripts\spyder"
```

Optional – Advanced

Generate the FMU from JModelica.org

- Shortcut and automatic loading of JModelica
  - 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



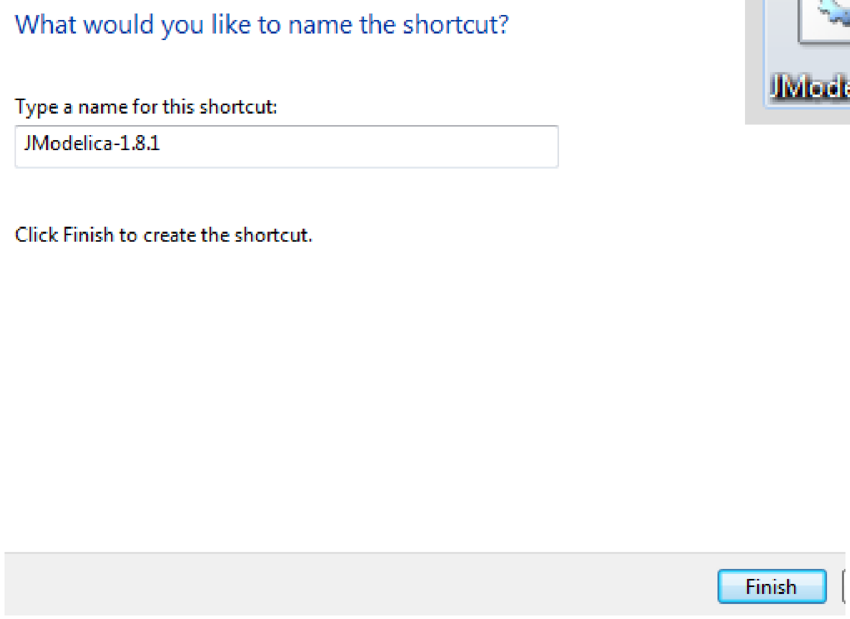
What item would you like to create a shortcut for?

This wizard helps you to create shortcuts to local or network programs, files, folders, computers, or Internet addresses.

Type the location of the item:

C:\JModelica.org-1.8.1\Python.bat

Click Next to continue.


What would you like to name the shortcut?

Type a name for this shortcut:

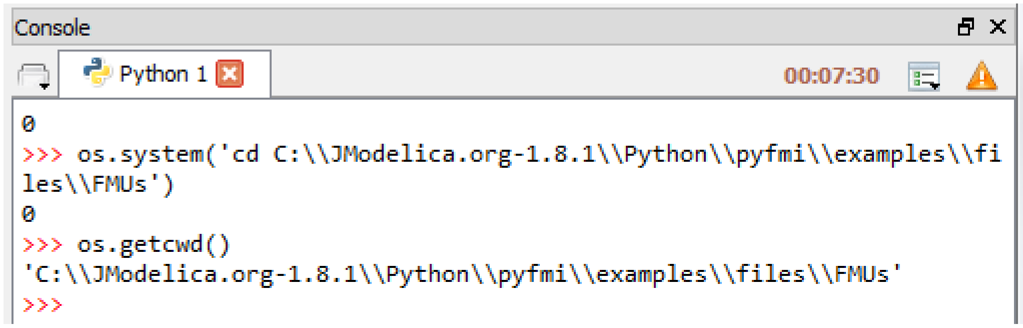
JModelica-1.8.1

Click Finish to create the shortcut.

Optional – Advanced

Generate the FMU from JModelica.org

- Compile FMUs from JModelica
  - Go to the directory where you have put the Modelica model you whish to compile through the command prompt in Spyder, and check that you are there



Console

Python 1

00:07:30

```
0
>>> os.system('cd C:\\JModelica.org-1.8.1\\Python\\pyfmi\\examples\\fi
les\\FMUs')
0
>>> os.getcwd()
'C:\\JModelica.org-1.8.1\\Python\\pyfmi\\examples\\files\\FMUs'
>>>
```

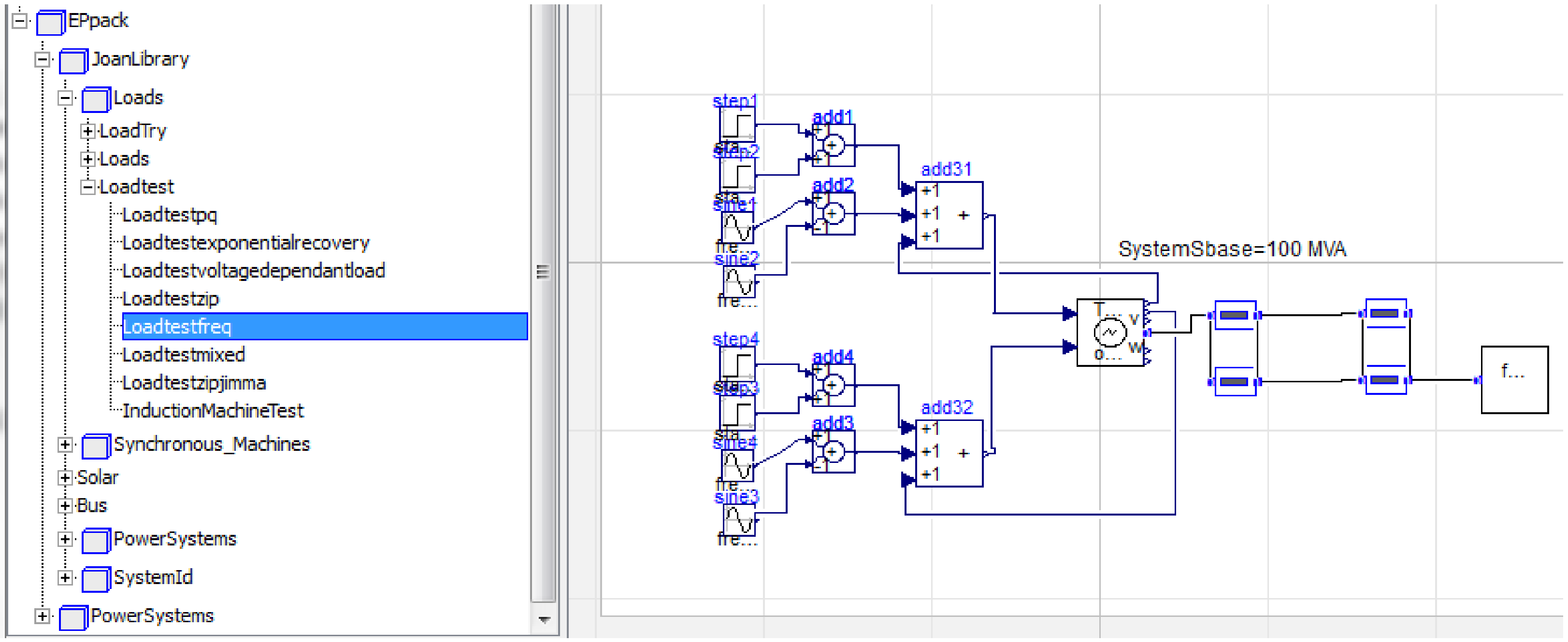
  - The following syntax can now be used to compile your FMU, we use it in the next example:

```
# 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)
```

Optional – Advanced

Generate the FMU from JModelica.org

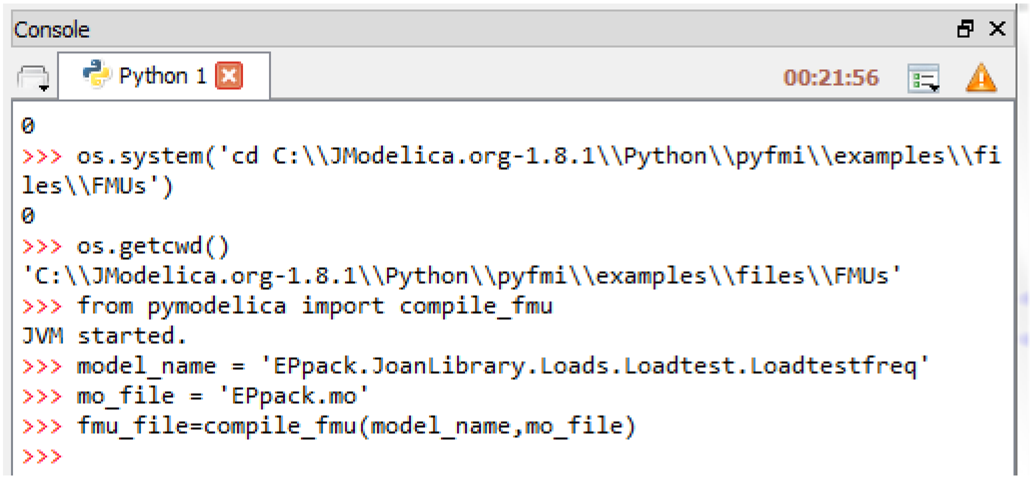
- Compile FMUs from Jmodelica: locating the model name and the file name.
  - We will compile a small test system including a third order generator, lines and a frequency dependent load that can be used for load parameter estimation
  - This model is contained the ‘`EPpack.mo`’ package**, this is the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



Optional – Advanced

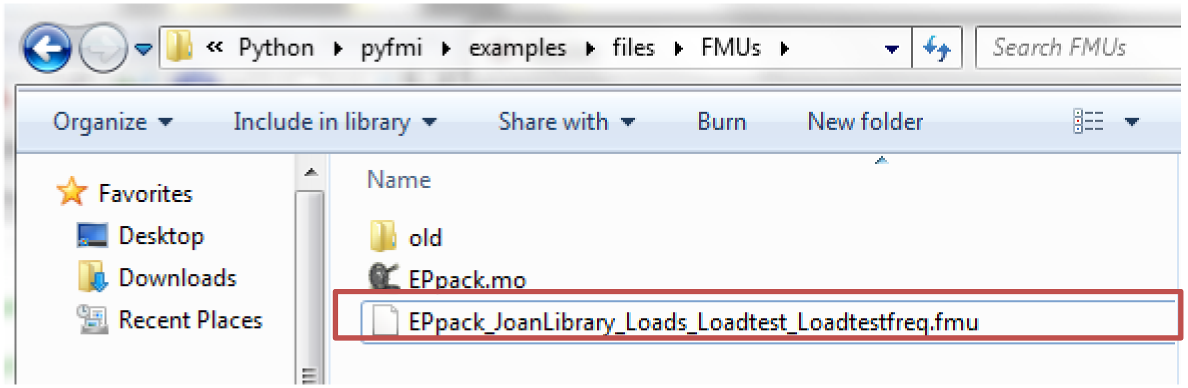
Generate the FMU from JModelica.org

- Compile FMUs from Jmodelica: Example, part 1 – compiling!
  - Go to the console of Syper and type the commands as shown in the screenshot



```
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, you will find the compiled FMU there



```
old
EPpack.mo
EPpack_JoanLibrary_Loads_Loadtest_Loadtestfreq.fmu
```