#### RaPId ‘User Manual’

# 

[Introduction](#h.aqfu1olf7bp5)

[The way it works](#h.ujmk4rr22e86)

[Setting up RaPId](#h.sq344d45yygb)

[What does what?](#h.j5gvsb1v0aqi)

[Recommendations](#h.3zdjgjpuchvp)

[Parameter description](#h.q3ypwh5idtn)

[Setting parameters](#h.1qr0bf7s22g2)

[Containers (setting and results back up)](#h.prhxj9d0fhxa)

[Visualise the result of every iteration](#h.nawvqj5pd6st)

[Command line execution](#h.lllu3vlfjnn7)

Introduction

**RaPId** is a toolbox providing a framework for parameter identification. As of today, the toolbox requires the users to comply with quite restrictive requirements.

1. The FMI toolbox (see modelon.com) must be installed in a 32bit version
2. The model to be tuned must be written in Modelica, exported as a \*.fmu container. In our tests, the files generated with Dymola could reliably be imported by the FMI toolbox but not those generated by OpenModelica.
3. Measurements data should be contained in a matlab file (\*.mat), the file can include any kind of matlab object
4. The user should rigorously follow the instruction given in the third section of this document

The ToolBox is a proof of concept. Unfortunately, the accent was not put on building a failsafe tool providing seamless user experience.

Exception handling was not yet taken care of. Any wrong parameter setting will generate numerous and hardly intelligible exceptions in the matlab console.

Besides the FMI Toolbox, on which repose RaPId can be characterised of unstable. For different reason, a model simulated without problem may suddenly cause problems. In those cases the toolbox itself complains (see errors in console). A reboot of matlab usually fixes the problems related to the toolbox.

The  symbol represents the operation to get the toolbox working. Feel free to avoid reading the boring text and stick to these operations only.

# The way it works

RaPId is a set of matlab scripts and a gui helping towards performing a parameter identification experience.  
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. 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 signal 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.

RaPId 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 minimise the difference between the measured data and the simulated data.

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

# Setting up RaPId

### What does what?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Menu in the Gui** | **Name in the GUI** | **Name of the field in the settings struct** | **Functionality** | **Typical value** |
| General Settings | 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.output2}  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 maters, has to math 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 maters, has to math 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 cost function which is quadratic | put 2, might change if you implement new objective functions |
|  | 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. |

### 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Menu in the Gui** | **Name in the GUI** | **Name of the field in the settings struct** | **Functionality** | **Typical value** |
| GA | nbChromosome | settings.ga\_options.nbCromosomes | 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 ar 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 phy\_now/phy\_start < fitnessStopRatio  meaning when we consider the fitness sufficiently low (weirdly enough, close match means small fitness index) | very small real positive number |
|  | 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 |
|  | nbResurections | 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 |
|  | nRandMin | settings.ga\_options.nRandMin | 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. nRandMin 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 |

### 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PSO | 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 of the speed directed towards the personal optimum of the particle | same as alpha1 |
|  | nbIterations | 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 |  |

### 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Naive | 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 |

### 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| other | 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: <https://code.google.com/p/psomatlab/> |  |
|  | psoExtOptions | settings.psoExtOptions | optimset for the toolbox psopt | see options in  “doc pso” |
|  | path2knitro | settings.kn\_options.path2Knitro | path to the knitro toolbox  found at: <http://www.ziena.com/knitro.htm> |  |
|  | 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 |

### 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Path | 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 |  |

### 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| main | inputData file | settings.path2data | path to the mat file containing the measurement data, can contain any kind of data handled by matlab |  |
|  | 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 |  |

#### Unless explicitly required to be a string, you should not input the data as strings. The field for the parm min vector should be [x1 x2] not ‘[x1 x2]’

### Recommendations

After extracting RaPId 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 (a certain parameter estimation for a certain model).

Even on 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 toolbox adds the FMI toolbox to your path based on the user-specified location.

**Setting up the modelica model**

 Extract RaPId

 Create a Folder, we’ll call it working directory

 Start Matlab

 Add to path the FMI toolbox main directory

 Start Simulink

 Create a new simulink model iin the working directory

 Open the component library and introduce an FMU ME bloc, a mux and a to workspace component.

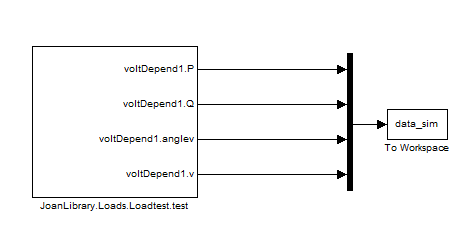
 Set the to workspace block to output a structure with time

 In the FMU ME block, go to the output tab and select all the outputs to be measured

 In the advanced tab of the FMU ME block, uncheck the setting ‘Use tolerance controlled FMU

 The name of the FMU block will most likely be quite long, rename it, this name, along with the to workspace component’s name will have to be given the toolbox.

 Save the model



### Methods description

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>

### Setting parameters

The first time the toolbox is open, the settings entered 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 RaPId main directory  
 Start the GUI by running the script gui\_run.m

The script adds the appropriate folder in the path, it won’t work if the current directory wasn’t changed as asked.

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 next

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. output names is the name appearing on the FMU block once the corresponding output was selected in the output tab of the FMU ME component. It’s composed the following way: subBlockName.variableName . 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.

param min, param max, p0 need to be of the appropriate size.

quadratic cost represent 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

fitness = (Ymeas - Ysimu)’ Q (Ymeas - Ysimu)’

quadratic cost represents the diagonal of 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 relly 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 optimiasation 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.

# 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.

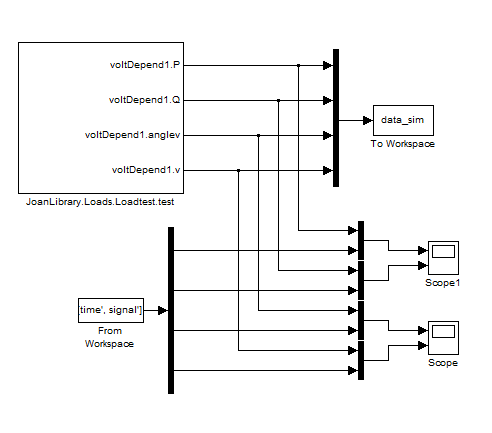
The toolbox contains menus allowing to set the values for every needed parameter. These menus should be closed with the button labelled send, if another method lead to shutting the menu, all change will be discarded. When the button send is pressed, the data is saved in the file data.mat, any former 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.

# Visualise the result of every iteration

In order to visualise the result of every iteration and get some insight into the way the algorithm works, scopes can be added in the simulink model. The following figure is an example of this. The idea is to include in the Simulink model the data used as measurement data and compare the result of iteration with the current computation.  
Many iterations of the optimisation routine will follow each other, if the simulation only takes a fraction of a second, the manouver might be pointless, Otherwise, it should be able to observe the simulated data getting closer to the measurement data as the toolbox iterates the simulations.

:

The From Workspace component helps importing the measured data. The idea is to visualise with the scopes the difference between the data coming from the measurements and the data simulated at every step.

Note that the From Workspace component requires the first column of the data input to be the time vector, the following vectors are the signals themselves.

If the modelica model expects an input to be fed, the FMU block will have an input port. That’s for you to generate the input signal the way you want it basing yourself on the simulink tools.  
A way to do it is to use a From Workspace block and loading a time series. For that purpose the time series must be loaded into the Matlab Workspace beforehand.

# Command line execution

The command line execution of the toolbox requires the user to create the struct containing all the parameters of the toolbox by himself. A function call with all the appropriate settings for one specific method is rather compact.  
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 rapid\_run which gives an example of command line definition for every single method included.  
  
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.

Remark: The GUI takes care of loading the different toolbox needed to the path from the fields of the struct called settings built as the users fills the different menus of the gui.

The user will then have to include the following lines to repproduce 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.

### Important 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 be generated through the optiization toolbox obtained with the command optimtool.  
This a 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’.

There exist an unsolved problem with the FMI toolbox, after a certain time, the toolbox may produce the following exception:

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

When this occurs, the toolbox stops, clear all needs to be called in the console.