Development of an open source multi-platform software tool for parameter estimation studies in FMI models

Javier Bonilla[1,3] Jose A. Carballo[1,3] Lidia Roca[1,3] Manuel Berenguel[2,3]

[1]CIEMAT-PSA, Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas - Plataforma Solar de Almería, Spain, {javier.bonilla,jose.carballo,lidia.roca}@psa.es
[2]Department of Informatics, University of Almería, Almería, Spain, beren@ual.es
[3]CIESOL, Solar Energy Research Center, Joint Institute University of Almería - CIEMAT, Almería, Spain

Abstract

This paper presents the current development of an open source multi-platform software tool intended for estimating or optimizing parameters of Functional Mock-up Interface (FMI) compliant models. Parameter estimation and optimization is a powerful tool in many engineering and science fields. Nevertheless, the effort and time that must be devoted to coupling and integrating complex modeling languages and tools together with analysis and optimization methods and algorithms sometimes is high. As a consequence of that, commonly the most convenient and easy-to-use optimization mechanisms are applied. Therefore, the focus on the development of this tool is in facilitating such coupling while being customizable. The main toolkit and libraries used in the development of the tool are presented, all of them are open source. Two application examples are also presented, one of them is a parameter optimization study considering a steady state model, while the other is a parameter estimation study of a dynamic model against experimental data. Finally, current tool limitations are presented, ongoing work and ideas for future features are also commented.

Keywords: parameter estimation, parameter optimization, model calibration, Functional Mock-up Interface (FMI), open source software tool

1 Introduction

The application of optimization to complex dynamic models has become recently more usual in industry, as well as in academia. Optimization can be online, such as optimal control in the form of Model Predictive Control (MPC) or offline, such as parameter estimation, state estimation or parameter optimization.

In parameter optimization, also known as design optimization, some parameters are optimized to improve the system dynamics or response according to some criteria. Parameter estimation, model calibration or parameter identification, comprises estimating some unknown parameters in a particular model. To that end, several simulations are performed and results are compared against experimental data. The unknown parameters are therefore determined by numerical optimization algorithms. This procedure is a powerful tool in many engineering and science fields and has its origin in the least squares method proposed by Gauss.

In order to apply optimization techniques to complex dynamic models, a suitable modeling language, that can deal with dynamic systems and the increasing complexity of research and engineering needs, is advisable. Modelica (Modelica Association, 2014b) is one of those modeling languages, easing the model development, maintenance and reuse thanks to the equation-based object-oriented paradigm and other useful features. Nevertheless, there are other commonly used modeling languages and simulation tools, being one of the most representative Matlab/Simulink (The MathWorks Inc., 2016). For this reason, the support of an independent standard devoted to Model Exchange (ME) and co-simulation, such as FMI (Modelica Association, 2014a), would be advisable when considering the model interface for an optimization software tool.

Even though there are modeling languages and tools for developing complex dynamic models, as well as a standard format to exchange those models, and advanced methods for optimization, sometimes the time required to couple all these tools is high. This task involves writing scripts for bindings and using several programming languages and tools.

With the aim of facilitating the integration of these tools and methods, an easy-to-use open source multi-platform software tool which performs parameter estimation studies in Functional Mock-up Units (FMUs) is currently being developed. This software tool performs parameter estimation studies using a global-search Multi-Objective Genetic Algorithm (MOGA). The tool also supports linear and non-linear equality and inequality constraints.

This paper is organized as follows. Section 1.1 is a brief summary of Modelica and FMI-based tools for parameter estimation. Section 2 describes the still under development software tool and its architecture. In Section 3, two examples are presented. Section 3.1 shows a steady-state parameter optimization study, whereas Section 3.2 presents a model calibration study against experimental data from a Thermal Energy Storage (TES) tank. This tank is used for research on solar thermal storage.

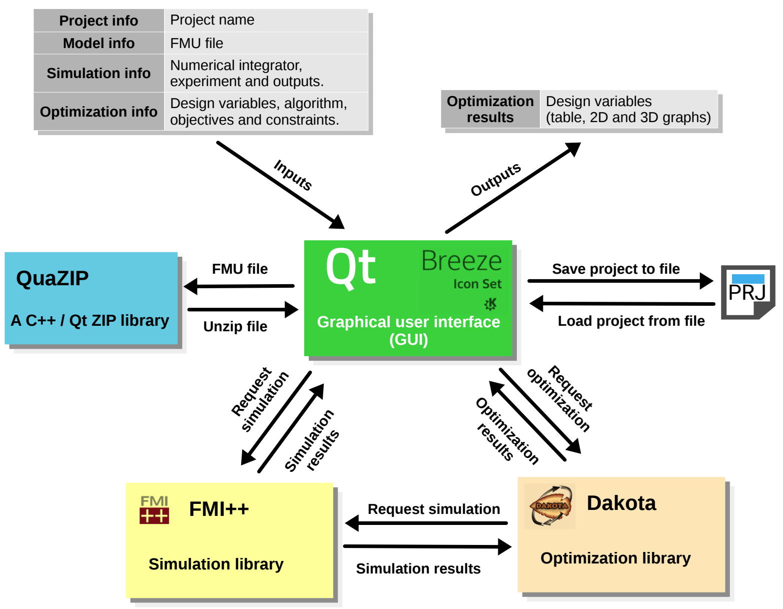


Figure 1. `Optifmus` information exchange

1.1 State of the art

Most commercial Modelica tools have parameter estimation and optimization libraries for Modelica models, i.e. Model Design Tools (Elmqvist et al., 2005; Pfeiffer, 2012) in Dymola (Dassault Systemes, 2016) and corresponding libraries in SimulationX (ESI ITI GmbH, 2016), MapleSim (MapleSoft, 2016) and SystemModeler (Wolfram, 2016), among others. GenOpt (Wetter, 2001) is an optimization program that can be coupled with Modelica models compiled in Dymola. BuildingsPy (Berkeley Lab, 2016) is a Python package that can run Modelica simulations using Dymola, additional Python packages for parameter estimation can be used by means of scripting, such as those from SciPy.org (SciPy developers, 2017). The OpenModelica tool (OSMC, 2016) includes the OMOptim tool (Thieriot et al., 2011) for parameter estimation of Modelica models in Windows platforms.

Modelica models can be also exported as FMUs and imported in commercial and open source numerical computational tools such as Matlab/Simulink and Scilab (Scilab Enterprises, 2015). Parameter estimation studies can be performed by means of scripting in these tools. JModelica.org (Åkesson et al., 2010) supports parameter estimation of Modelica and FMI models also by scripting. The RaPId Parameter Identification (RaPId) toolbox (Vanfretti et al., 2016) is a modular and extensible toolbox for parameter estimation of FMI models in Matlab/Simulink.

2 Optifmus Software Tool

The under development software tool is called `Optifmus`. Although it is at an early development stage, it is functional with respect to FMU simulations and parameter estimation studies using a MOGA. The tool is composed of the following main elements.

- **Graphical User Interface (GUI)**. The GUI allows the user to provide all the required information: model information (FMU file and parameters), simulation information (numerical solver and its configuration, inputs and simulation interval), optimization information (algorithm and its configuration, parameters, objective functions and constraints). The GUI also shows the obtained results. Results are presented in tables, 2D and 3D graphs.
- **Optimization toolkit**. This toolkit collects all the information introduced in the GUI and calls the FMU simulator to perform the needed simulation runs and carry the optimization out using the selected algorithm. Optimization results are presented to the user in the GUI.
- **FMU simulator**. The simulator performs the model simulation according to the suplied data (model and simulation information) and provides the results to the GUI or to the optimization toolkit.

2.1 Software Architecture

`Optifmus` is being developed in C++ using open source multi-platform libraries and tools. The following list briefly describes the main libraries. Figure 1 shows the information exchange between them.

- **Qt toolkit** (The Qt Company, 2016). It is a cross-platform application framework used for developing application software. The Qt Core and Qt Widgets modules are used for the GUI. The Qt Charts module is used for 2D graphs, whereas the Qt Data Visualization module is used for 3D graphs.
- **Breeze icons** (KDE Community, 2016). The GUI icons belong to this open source library.

- **FMI++** (Widl et al., 2013). The FMI++ library is a high-level utility package for FMI-based software development. It provides high-level features, which ease the handling and manipulation of FMU models: an eXtensible Markup Language (XML) parser and numerical integration capabilities. The FMI++ library relies on the Odeint library (Ahnert and Mulansky, 2011), and optionally on SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers (SUNDIALS) (Hindmarsh et al., 2005), for the numerical integration of FMUs.
- **QuaZIP** (Tachenov, 2016). It is a C++ wrapper for accessing ZIP files. This wrapper uses the Qt toolkit and therefore it is a multi-platform wrapper. It is used in `Optifmus` to handle the extraction of FMU files.
- **Dakota** (Adams et al., 2016). The Dakota toolkit is intended as a flexible, extensible interface between simulation codes and a variety of iterative systems analysis methods. Dakota is a powerful toolkit which provides the following functionality: optimization, uncertainty quantification, nonlinear least squares methods, and sensitivity/variance analysis. Dakota uses Sandia-developed libraries, as well as external optimization and design of experiments libraries. For further details consult Sandia Corporation (2016). Dakota can be used as a standalone application or as a C++ library.

Additionally, reading and writing operations of input and result files in Comma-separated Values (CSV) or trajectory mat format are performed by means of C functions available in the source code of the OpenModelica tool.

3 Examples

In order to show the `Optifmus` capabilities and illustrate how a parameter estimation study can be performed, the following sections introduce two examples. Section 3.1 shows a steady-state parameter optimization study, whereas Section 3.2 shows a model calibration study against experimental measurements from a real facility.

3.1 Parameter optimization

The general formulation for a optimization problem description is given by Equation 1. It can be formulated as optimize (minimize or maximize) several objective functions $f(x)$ that depend on some parameters or design variables x subject to several constraints: upper and lower bounds for design variables, x_l and x_u, equality constraints, $g(x)$ and inequality constraints, $h(x)$.

$$\begin{aligned} &\text{optimize} && f(x) \\ &\text{with respect to} && x \in \mathbb{R}^j \\ &\text{subject to} && x_l \le x \le x_u, \\ & && g(x) = 0, \\ & && h(x) \le 0, \end{aligned} \tag{1}$$

where,

$$\begin{aligned} f(x) &= \{f_1(x) \cdots f_i(x)\}, \\ x &= \{x_1 \cdots x_j\}, \\ x_l &= \{x_{l,1} \cdots x_{l,j}\}, \\ x_u &= \{x_{u,1} \cdots x_{u,j}\}, \\ g(x) &= \{g_1(x) \cdots g_k(x)\}, \\ h(x) &= \{h_1(x) \cdots g_n(x)\}. \end{aligned}$$

The example considered in this section, known as Srinivas' problem, can be found in the Dakota User's Manual (Adams et al., 2016), section *Additional examples → Multiobjective test problems → Multiobjective test problem 3*. The problem has two design variables, x_1 and x_2 with their respective upper and lower bounds,

$$\begin{aligned} -20 \le & x_1 \le 20, \\ -20 \le & x_2 \le 20, \end{aligned}$$

two objective functions, f_1 and f_2, which must be minimized,

$$\begin{aligned} f_1(x_1,x_2) &= (x_1-2)^2 + (x_2-1)^2 + 2, \\ f_2(x_1,x_2) &= 9x_1 - (x_2-1)^2, \end{aligned}$$

and two inequality constraints h_1 and h_2,

$$\begin{aligned} h_1(x_1,x_2) &= x_1^2 + x_2^2 - 225 \le 0, \\ h_2(x_1,x_2) &= x_1 - 3x_2 + 10 \le 0. \end{aligned}$$

The first step is to generate a FMU file of this model, most Modelica tools support exporting Modelica models to FMUs. The Modelica code of this model is as follows.

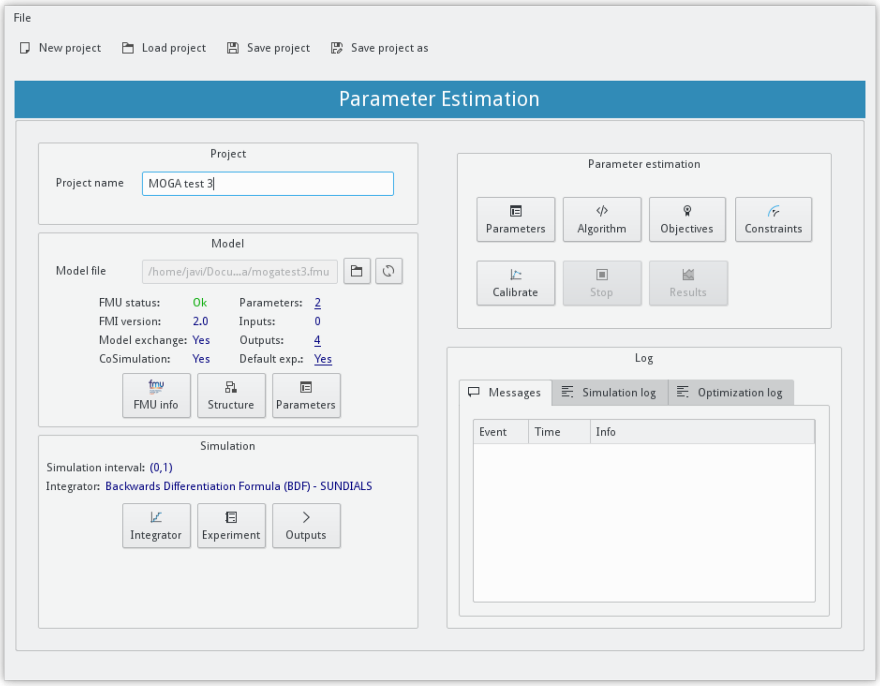
```
model mogatest3
  parameter Real x1 = 0 "Parameter x1";
  parameter Real x2 = 0 "Parameter x2";
  output Real f1 "Function f1";
  output Real f2 "Function f2";
  output Real h1 "Constraint h1";
  output Real h2 "Constraint h2";
equation
  f1 = (x1-2)^2 + (x2-1)^2 + 2;
  f2 = 9*x1 - (x2-1)^2;
  h1 = x1^2 + x2^2 - 225;
  h2 = x1 - 3*x2 + 10;
end mogatest3;
```

The next step is to create a new project in `Optifmus`. Currently, two kinds of projects can be created: simulation and parameter estimation. Projects can be saved to and loaded from files. Figure 2 shows the `Optifmus` GUI for parameter estimation. The information that must be completed is divided in groups in the GUI and it is described as follows.

1. **Project information**. A descriptive name can be given to easily identify the project.
2. **Model information**. A FMU file must be specified. Once the file is loaded, some information is

Table 1. Numerical integrators

Numerical integrator	Library	Step size	Order
Forward Euler	Odeint	Constant	1
4^{th} order Runge-Kutta	Odeint	Constant	4
Adams-Bashforth-Moulton	Odeint	Constant	Adjustable
5^{th} order Runge-Kutta-Cash-Karp	Odeint	Controlled	5
5^{th} order Runge-Kutta-Dormand-Prince	Odeint	Controlled	5
8^{th} order Runge-Kutta-Fehlberg	Odeint	Controlled	8
Bulirsch-Stoer	Odeint	Controlled	Controlled
4^{th} Rosenbrock	Odeint	Controlled	4
Backwards Differentiation Formula (BDF)	SUNDIALS	Controlled	Controlled
Adams-Bashforth-Moulton	SUNDIALS	Controlled	Controlled



Project name: MOGA test 3

Model file: /home/javi/Docu...a/mogatest3.fmu

FMU status: Ok
FMI version: 2.0
Model exchange: Yes
CoSimulation: Yes
Parameters: 2
Inputs: 0
Outputs: 4
Default exp.: Yes

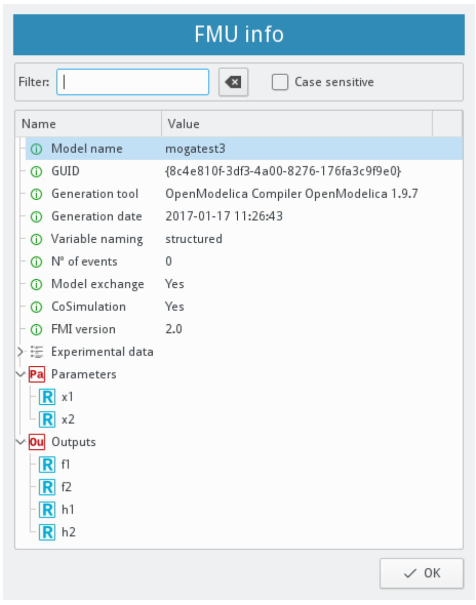
Simulation interval: (0,1)
Integrator: Backwards Differentiation Formula (BDF) - SUNDIALS

Figure 2. `Optifmus` GUI for parameter estimation.

displayed in the GUI. There are three buttons in this group. The FMU info button shows some information about the model, see Figure 3a. The structure button shows the model structure, see Figure 3b. The parameters button allows the user to give values to the model parameters, see Figure 3c. Since in our case, both parameters x_1 and x_2 are going to be calibrated, there is no need to give them values.

3. **Simulation information**. There are also three buttons in this group: numerical integrator, experiment and outputs. The first one allows us to select the numerical integrator, Figure 4a. The step size is only needed if it is not controlled by the integrator. If it is controlled, absolute and relative tolerance are used instead. Some integrators allows the users to specify the order whereas others have a fixed constant or a controlled order. The FMI++ library can use the numerical integrators given in Table 1 from the Odeint and SUNDIALS libraries. The numerical integrator (BDF) and tolerances (10^{-4}) are left by default in our example. The experiment window permits selecting the simulation interval, start and stop times, and matchs model inputs with data from files. Since our model does not have inputs and it is a steady-state model, this step can be omitted and thus leaving the simulation interval by default, [0,1] seconds. Section 3.2 shows how to use this window.

   The outputs window, see Figure 4b, shows the model outputs, furthermore allows us to specify the number of intervals, i.e. the number of points that will be sampled for the output trajectory. The number of points can be also set by a time step instead of by a fixed number. Since our model is a steady-state one, there is no need to sample more than one interval. One interval means that values at the beginning and end of the simulation are stored.



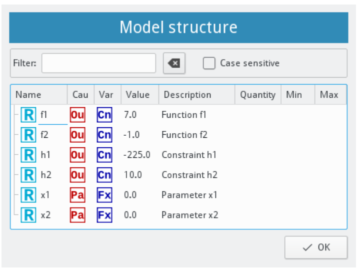
FMU info

Filter: | Case sensitive

Name	Value
Model name	mogatest3
GUID	{8c4e810f-3df3-4a00-8276-176fa3c9f9e0}
Generation tool	OpenModelica Compiler OpenModelica 1.9.7
Generation date	2017-01-17 11:26:43
Variable naming	structured
Nº of events	0
Model exchange	Yes
CoSimulation	Yes
FMI version	2.0
Experimental data	
Parameters	
x1	
x2	
Outputs	
f1	
f2	
h1	
h2	

OK

(**a**) FMU information



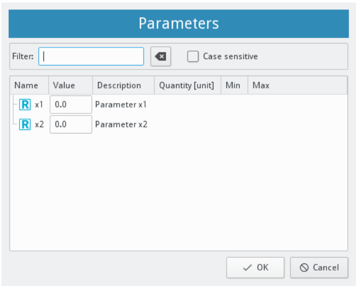
Model structure

Filter: Case sensitive

Name	Cau	Var	Value	Description	Quantity	Min	Max
R f1	Ou	Cn	7.0	Function f1			
R f2	Ou	Cn	-1.0	Function f2			
R h1	Ou	Cn	-225.0	Constraint h1			
R h2	Ou	Cn	10.0	Constraint h2			
R x1	Pa	Fx	0.0	Parameter x1			
R x2	Pa	Fx	0.0	Parameter x2			

OK

(**b**) Model structure



Parameters

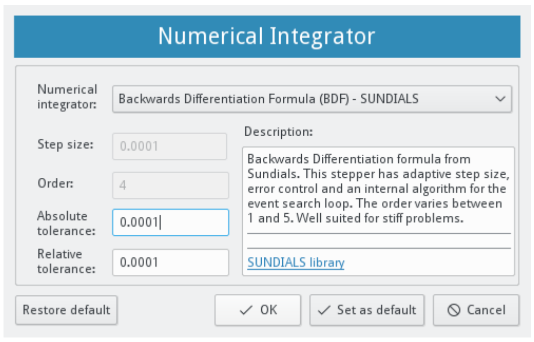
Filter: | Case sensitive

Name	Value	Description	Quantity [unit]	Min	Max
R x1	0.0	Parameter x1			
R x2	0.0	Parameter x2			

OK Cancel

(**c**) Model parameters

Figure 3. Model information



Numerical Integrator

Numerical integrator:	Backwards Differentiation Formula (BDF) - SUNDIALS
Step size:	0.0001
Order:	4
Absolute tolerance:	0.0001
Relative tolerance:	0.0001

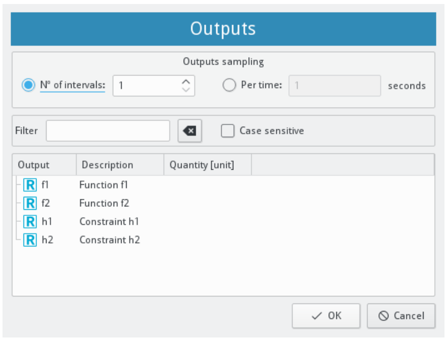
Description:

Backwards Differentiation formula from Sundials. This stepper has adaptive step size, error control and an internal algorithm for the event search loop. The order varies between 1 and 5. Well suited for stiff problems.

SUNDIALS library

Restore default | OK | Set as default | Cancel

(**a**) Numerical integrator



Outputs

Outputs sampling

[✓ Nº of intervals: 1] Per time: 1 seconds

Filter Case sensitive

Output	Description	Quantity [unit]
R f1	Function f1	
R f2	Function f2	
R h1	Constraint h1	
R h2	Constraint h2	

OK Cancel

(**b**) Model outputs

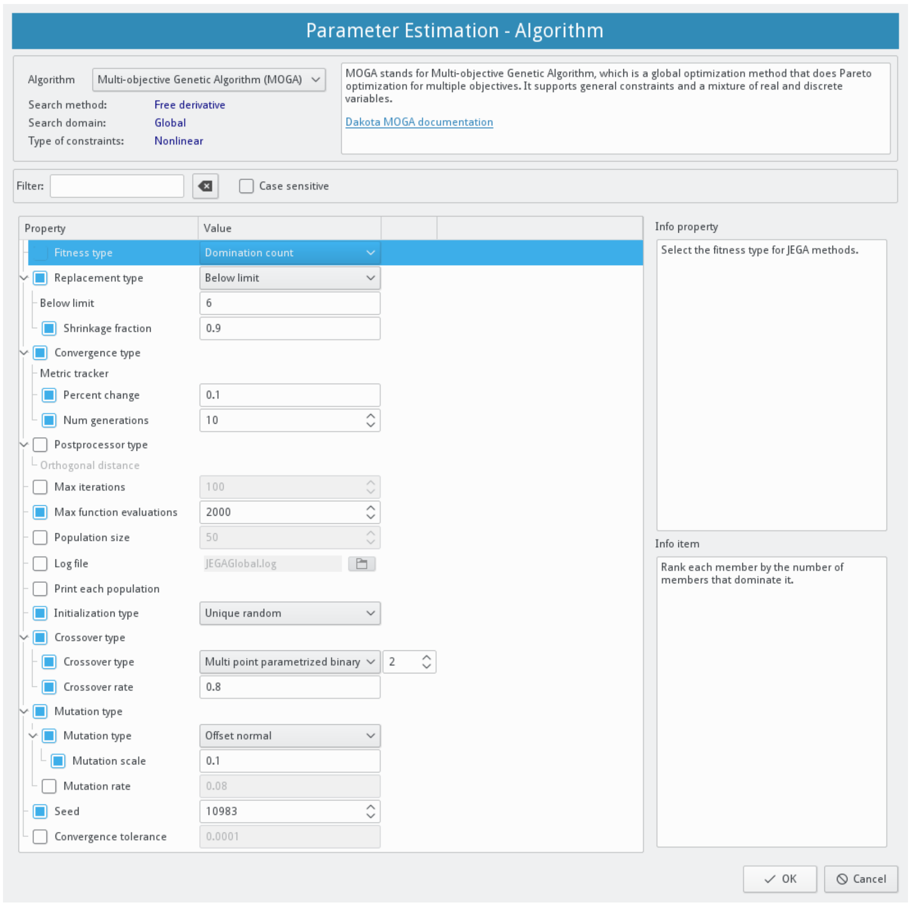
Figure 4. Simulation information

4. **Parameter estimation information**. This GUI group gathers all the information for the parameter estimation study: parameters or design variables, optimization algorithm, objective functions and constraints.

- **Parameters**. The parameters to be calibrated can be selected in the parameter window, see Figure 6. For each parameter, it can be specified its initial value, its lower and upper bounds, and if scaling is considered. If this is the case, the scaling type can be a fixed value, logarithm scale or automatic. The two first require a scale value. Consult Dakota documentation for further information about scaling of design variables (Adams et al., 2016).

- **Algorithm**. Currently, the only Dakota algorithm considered in the tool is the MOGA from the Sandia-developed JEGA library (Eddy and Lewis, 2001). This is a multi-objective algorithm which supports general constraints: bounded design variables, linear and nonlinear equality and inequality constraints. The algorithm is highly configurable. Figure 5 shows the window to configure the algorithm. The options selected in our case are those indicated in Dakota documentation for this example. The number of model evaluations is set to 2000.

- **Objectives**. The objectives can be selected in the objectives window, see Figure 7. For each objective the following options are available: criterion (maximize or minimize), trajectory reduction, weight, scaling type and value. The trajectory reduction option reduces the whole trajectory for each objective function to a single value in each simulation run. This is required because the optimization algorithm needs a single value per objective function and simulation run. The following options are available: root mean of squares, mean of absolute values, max, min or last value. The weight value is used if the option to use weights, and therefore transform the multi-objective problem to a single-objective problem, is checked.



Algorithm: Multi-objective Genetic Algorithm (MOGA)
Search method: Free derivative
Search domain: Global
Type of constraints: Nonlinear

MOGA stands for Multi-objective Genetic Algorithm, which is a global optimization method that does Pareto optimization for multiple objectives. It supports general constraints and a mixture of real and discrete variables.

Dakota MOGA documentation

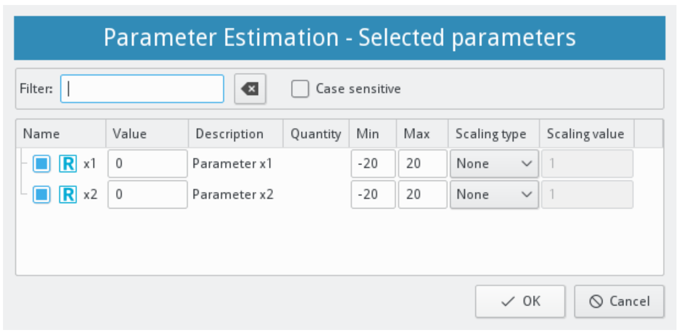
Filter: ☐ Case sensitive

Property	Value
Fitness type	Domination count
☑ Replacement type	Below limit
Below limit	6
☑ Shrinkage fraction	0.9
☑ Convergence type	
Metric tracker	
☑ Percent change	0.1
☑ Num generations	10
☐ Postprocessor type	
Orthogonal distance	
☐ Max iterations	100
☑ Max function evaluations	2000
☐ Population size	50
☐ Log file	JEGAGlobal.log
☐ Print each population	
☑ Initialization type	Unique random
☑ Crossover type	
☑ Crossover type	Multi point parametrized binary, 2
☑ Crossover rate	0.8
☑ Mutation type	
☑ Mutation type	Offset normal
☑ Mutation scale	0.1
☐ Mutation rate	0.08
☑ Seed	10983
☐ Convergence tolerance	0.0001

Info property: Select the fitness type for JEGA methods.

Info item: Rank each member by the number of members that dominate it.

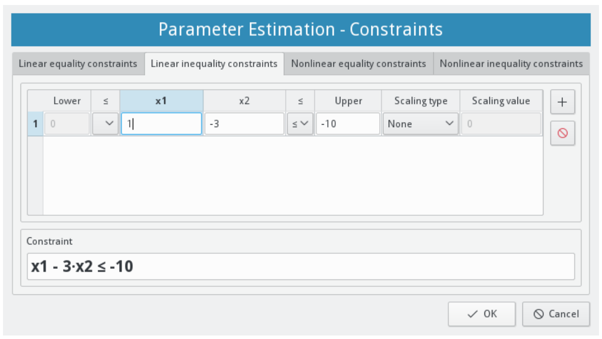
Figure 5. Optimization algorithm configuration



Filter: ☐ Case sensitive

Name	Value	Description	Quantity	Min	Max	Scaling type	Scaling value
☑ R x1	0	Parameter x1		-20	20	None	1
☑ R x2	0	Parameter x2		-20	20	None	1

Figure 6. Selected design variables

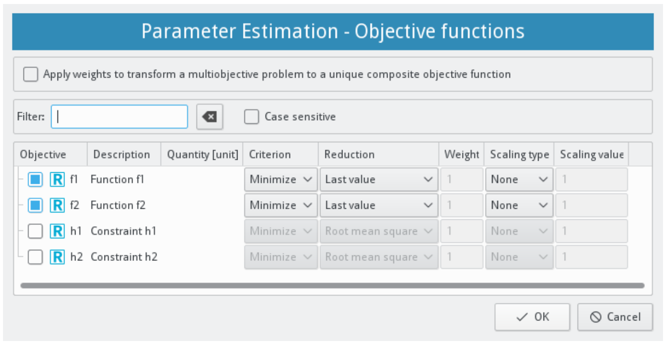


Parameter Estimation - Constraints
Linear equality constraints
Linear inequality constraints
Nonlinear equality constraints
Nonlinear inequality constraints

	Lower	≤	x1	x2	≤	Upper	Scaling type	Scaling value
1	0		1	-3	≤	-10	None	0

Constraint: **x1 - 3·x2 ≤ -10**

Figure 8. Linear inequality constraints

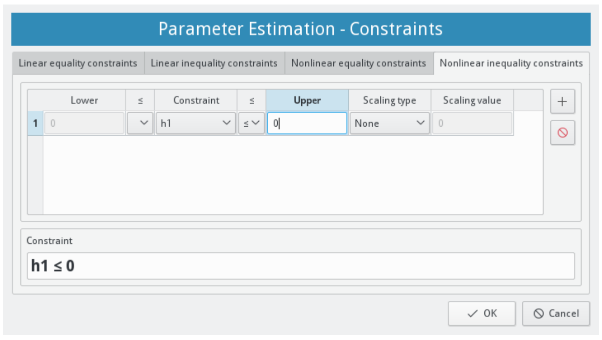


☐ Apply weights to transform a multiobjective problem to a unique composite objective function

Filter: ☐ Case sensitive

Objective	Description	Quantity [unit]	Criterion	Reduction	Weight	Scaling type	Scaling value
☑ R f1	Function f1		Minimize	Last value	1	None	1
☑ R f2	Function f2		Minimize	Last value	1	None	1
☐ R h1	Constraint h1		Minimize	Root mean square	1	None	1
☐ R h2	Constraint h2		Minimize	Root mean square	1	None	1

Figure 7. Objective functions



Parameter Estimation - Constraints
Linear equality constraints
Linear inequality constraints
Nonlinear equality constraints
Nonlinear inequality constraints

	Lower	≤	Constraint	≤	Upper	Scaling type	Scaling value
1	0		h1	≤	0	None	0

Constraint: **h1 ≤ 0**

Figure 9. Nonlinear inequality constraints

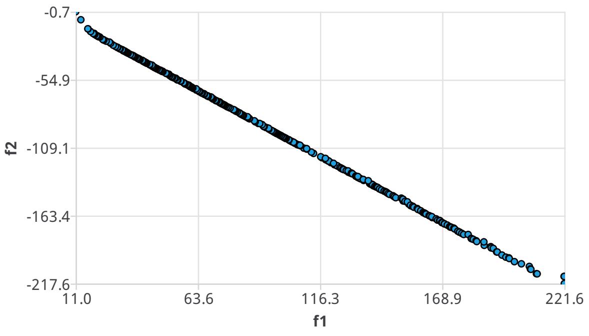


Figure 10. Srinivas' problem - 2D graph

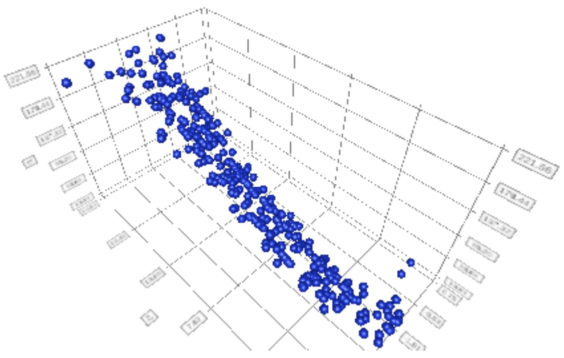
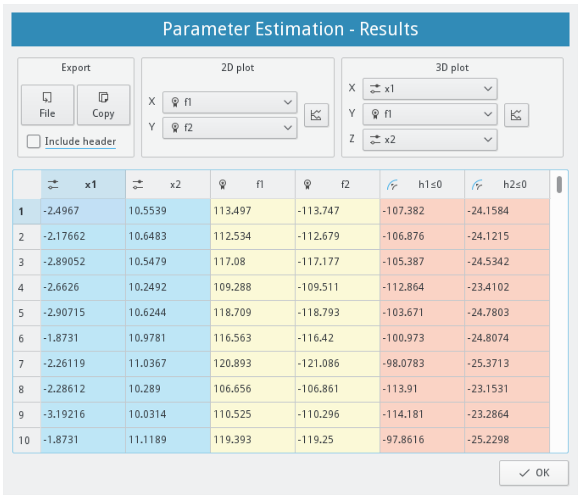


Figure 11. Srinivas' problem - 3D graph

The scaling option has the same meaning than for design variables, but there are some limitations for objective functions, consult Dakota documentation for further details. In our example, for both objective functions the criterion is minimize. Since this is a steady-state model, reductions are set to last simulation values. Scaling is not used and weight are not enabled because this is a multi-objective optimization problem.

- **Constraints**. The tool supports linear and nonlinear equality and inequality constraints. Scaling options are also available. Linear constraints with respect to design variables can be directly specified in the appropriate tab in the constraint window. In our example, the linear constraint h_2 was defined in the Modelica code, but this was not necessary since it can be directly defined in the GUI, see Figure 8. On the other hand, it can be also defined in the model as in our example. Nonlinear constraint functions must be defined in the model, in our example h_1, then both or only one limit per constraint must be set in the GUI, see Figure 9.

Once all previous information is defined, the optimization process can be performed hitting the calibration button, see Figure 2. The stop button allows us to stop the current optimization process. When the calibration process is completed, the results button will be enabled to show



	x1	x2	f1	f2	h1≤0	h2≤0
1	-2.4967	10.5539	113.497	-113.747	-107.382	-24.1584
2	-2.17662	10.6483	112.534	-112.679	-106.876	-24.1215
3	-2.89052	10.5479	117.08	-117.177	-105.387	-24.5342
4	-2.6626	10.2492	109.288	-109.511	-112.864	-23.4102
5	-2.90715	10.6244	118.709	-118.793	-103.671	-24.7803
6	-1.8731	10.9781	116.563	-116.42	-100.973	-24.8074
7	-2.26119	11.0367	120.893	-121.086	-98.0783	-25.3713
8	-2.28612	10.289	106.656	-106.861	-113.91	-23.1531
9	-3.19216	10.0314	110.525	-110.296	-114.181	-23.2864
10	-1.8731	11.1189	119.393	-119.25	-97.8616	-25.2298

Figure 12. Srinivas' problem - optimization results

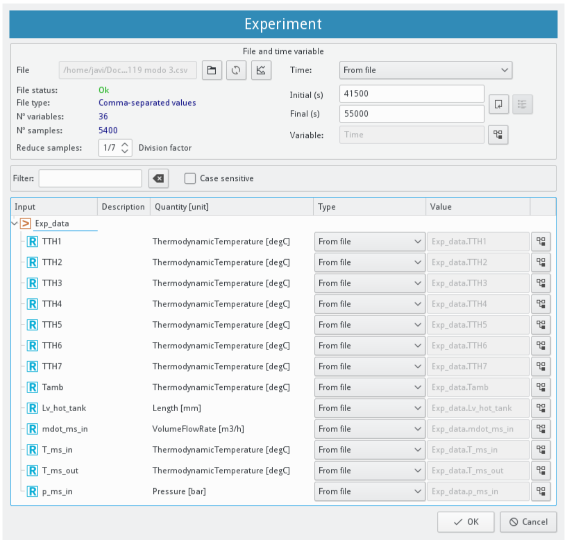
information about the optimization results. Our optimization example took less than 5 seconds for 2000 model evaluations in a conventional laptop (4 x Intel Core i5 2.60 GHz, 8 Gbytes of RAM). During the process, messages and log information are shown in the GUI.

Optimization results are shown in the results window, see Figure 12. The MOGA algorithm provides all the solutions found in or close to the Pareto front. In our case, the MOGA algorithm found 421 solutions after 2000 model evaluations. Design values, objective functions and constraints for each solution are shown in a table. In the result window, any design variable, objective function or constraint can be selected to be plotted in 2D or 3D graphs. Figure 10 shows a 2D graph of f_2 with respect to f_1, which is the Pareto front of our problem. Figure 11 shows a 3D graph of f_1 with respect to x_1 and x_2.

3.2 Model calibration

This section presents the calibration of a TES tank dynamic model that it is under development. This kind of tanks is used in solar thermal power plants in order to store thermal energy and dispatch it at night or under unfavorable meteorological conditions. A complete description of the model is out of the scope of this paper, but a brief summary is given in the following lines. The storage fluid is commonly molten salts, which can reach high temperatures.

The dynamic model considers two control volumes and dynamic mass and energy balances for molten salt and the the inert gas in the facility, nitrogen. Tank geometry, slope and dimensions are considered in the model. The pump inside the tank is also modeled, assuming a simplified geometrical form. The position of the level meter and thermocouples in the tank are also taken into account. The model considers different kinds of heat transfer processes: convection, conduction and radiation between molten salt, gas, tank walls, roof, floor, pump, insulation and foundation. The variables of interest are tank level, together with molten salt and gas temperatures.



File	/home/javi/Doc...119 modo 3.csv
File status:	Ok
File type:	Comma-separated values
Nº variables:	36
Nº samples:	5400
Reduce samples:	1/7 Division factor
Time:	From file
Initial (s)	41500
Final (s)	55000
Variable:	Time
Filter:	
Case sensitive	☐

Input	Description	Quantity [unit]	Type	Value
Exp_data				
TTH1		ThermodynamicTemperature [degC]	From file	Exp_data.TTH1
TTH2		ThermodynamicTemperature [degC]	From file	Exp_data.TTH2
TTH3		ThermodynamicTemperature [degC]	From file	Exp_data.TTH3
TTH4		ThermodynamicTemperature [degC]	From file	Exp_data.TTH4
TTH5		ThermodynamicTemperature [degC]	From file	Exp_data.TTH5
TTH6		ThermodynamicTemperature [degC]	From file	Exp_data.TTH6
TTH7		ThermodynamicTemperature [degC]	From file	Exp_data.TTH7
Tamb		ThermodynamicTemperature [degC]	From file	Exp_data.Tamb
Lv_hot_tank		Length [mm]	From file	Exp_data.Lv_hot_tank
mdot_ms_in		VolumeFlowRate [m3/h]	From file	Exp_data.mdot_ms_in
T_ms_in		ThermodynamicTemperature [degC]	From file	Exp_data.T_ms_in
T_ms_out		ThermodynamicTemperature [degC]	From file	Exp_data.T_ms_out
p_ms_in		Pressure [bar]	From file	Exp_data.p_ms_in

Figure 13. Tank calibration experiment

Molten salt tank foundation designs are commonly outside of standards for foundations, since these standards do not cover the temperature range were TES systems operate. The modeled tank has several foundation layers made of concrete with steel fibers and a compacted light expanded clay aggregate. The thermal insulation is usually made of several layers from different materials. The outer thermal insulation layer is frequently covered with an aluminum jacket for weather protection. For all these reasons, thermal conductivities in the insulation and foundation are difficult values to estimate. In this model, those thermal conductivities are assumed as mean constant values and have been calibrated using the `Optifmus` tool. Needed measurements are available from experimental campaigns carried out in the facility. It is located at CIEMAT - Plataforma Solar de Almería (PSA). Therefore, our model calibration problem can be formulated as minimizing the differences between molten salt and gas experimental and simulated temperatures by tuning the mean insulation and foundation thermal conductivities.

The steps to perform the calibration are similar to those in Section 3.1, for example setting the project (step 1) and model (step 2) information. The remaining steps are briefly summarized in what follows.

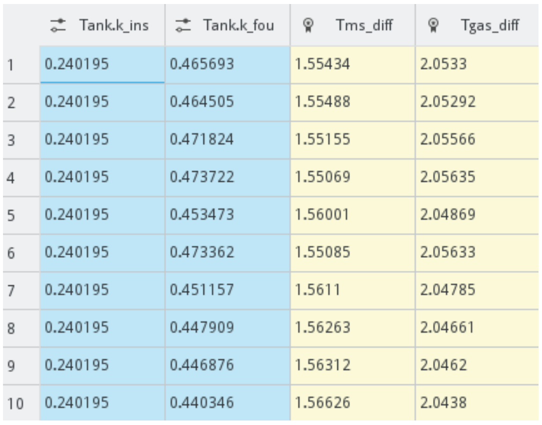
3. **Simulation information**. The experiment window is used to match model inputs with experimental data stored in files, see Figure 13. The values in the input file can be visualized thanks to a plotting tool included in `Optifmus`. In case the input file is oversampled, a factor can be used to reduce the number of samples. Model inputs can be also fixed to constant values in this window if needed. Time values can be read from the loaded file or a number of time intervals between the start and stop times can be specified in the GUI. In our example, the simulation interval is set to $[41500, 55000]$ seconds. The number of samples is reduced by $1/7$ in order to reduce the calibration time, since samples were taken each five seconds. Model inputs and time are matched to file data. The numerical integrator is left by default. The number of intervals is set to 500 in the output window in order to capture several points in the output trajectories.

4. **Parameter estimation information**. There are no constraints in our example, the remaining configuration options are described as follows.

- **Parameters**. Insulation and foundation mean thermal conductivities (k_{ins}, k_{fou}) are the design variables, guess initial values are $0.15\,\mathrm{W/(m\,K)}$. They are bounded in the $[0, 1]$ interval.
- **Algorithm**. The MOGA algorithm is used with its default values, besides the number of model evaluations (1000) and the seed (1) in order to obtain reproducible results.
- **Objectives**. The differences between experimental and simulated molten salt and gas temperatures are the two objective functions ($T_{ms,diff}, T_{gas,diff}$), they must be minimized. The trajectory reduction was set to root mean square values, therefore both objective functions provide the mean temperature difference in the trajectory. There is no need to use scaling since both objective functions represent temperature differences. Weights are not used because we are considering a multi-objective minimization problem.

The calibration process took 21 minutes and found 157 solutions in or close to the Pareto front for 1000 model evaluations. Figure 14 shows 10 of those solutions, where the objective functions give the mean temperature difference between experimental data and simulation results. Figure 15 shows the Pareto front.

The first solution in Figure 14, and pointed out by the arrow in Figure 15, was used to compare the model results against a different set of experimental data. All figures shown in this section were created in the `Optifmus` plotting tool. The system was exposed to several mass flow rate steps in this experiment, see Figure 16. Figure 17 shows experimental and simulated tank levels. Horizontal lines point out the position in height of the thermocouples. The gas experimental temperature corresponds to that from the highest-placed thermocouple, whereas the molten salt temperature is obtained from the highest-placed thermocouple immersed in molten salts. Figure 18 shows the experimental and simulated molten salt temperatures. Notice that there is no experimental molten salt temperature until the tank level reaches the first thermocouple position. This is why the experimental temperature is set to a constant value at the beginning of the simulation. Figure 19 shows the experimental and simulated gas temperatures. Notice that the molten salt level reaches the last thermocouple at the end of the simulation, therefore there are no available measurements for gas temperature.



	Tank.k_ins	Tank.k_fou	Tms_diff	Tgas_diff
1	0.240195	0.465693	1.55434	2.0533
2	0.240195	0.464505	1.55488	2.05292
3	0.240195	0.471824	1.55155	2.05566
4	0.240195	0.473722	1.55069	2.05635
5	0.240195	0.453473	1.56001	2.04869
6	0.240195	0.473362	1.55085	2.05633
7	0.240195	0.451157	1.5611	2.04785
8	0.240195	0.447909	1.56263	2.04661
9	0.240195	0.446876	1.56312	2.0462
10	0.240195	0.440346	1.56626	2.0438

Figure 14. Tank model calibration results

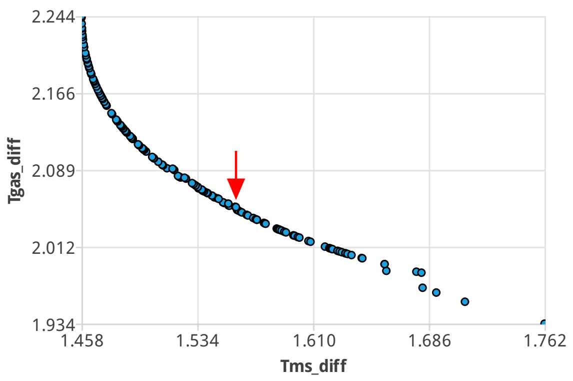


Figure 15. Tank model calibration Pareto front

4 Optifmus limitations

The current main `Optifmus` limitations are listed here.

- The software tool has been tested only in Linux. It is planned to be tested in Windows and Mac platforms.
- Only ME FMUs version 1.0 and 2.0 are supported.
- Only continuous real design parameters, objectives functions and constraints are supported.
- All the MOGA options are configurable from the GUI besides the niching type and the use of surrogate models which are not supported.

5 Ongoing work and future ideas

Ongoing work is summarized in the following list.

- Optimize the code to improve speed.
- Unit support when setting parameter values.
- Load FMI++ logs in the `Optifmus` GUI.
- More graphic configuration options.
- Include an option to simulate the model with the parameters of the selected row in the result table.

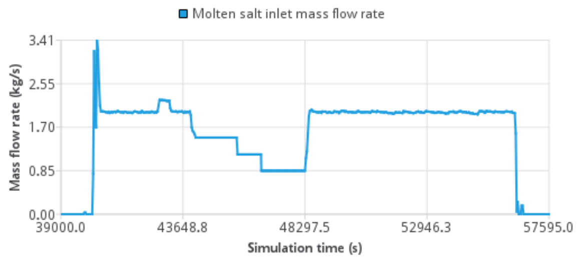


Figure 16. Tank simulation - mass flow rate

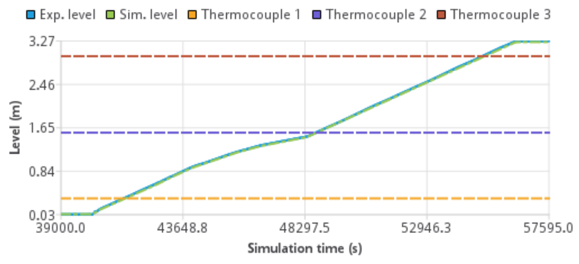


Figure 17. Tank simulation - levels

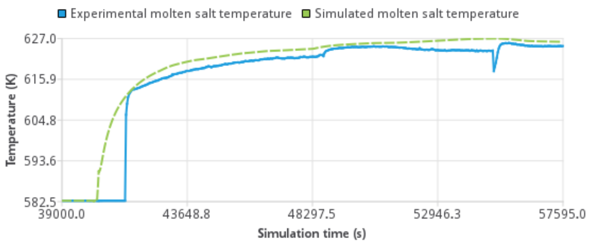


Figure 18. Tank simulation - molten salt temperatures

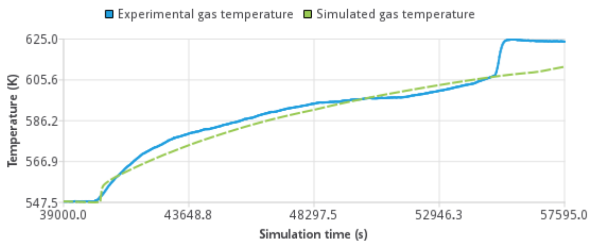


Figure 19. Tank simulation - gas temperatures

- Dakota offers many more algorithms for parameter estimation and optimization. One of the ongoing tasks is to implement several of those algorithms.

As future ideas to improve the software tool, the following will be considered and studied.

- Dakota is a powerful toolkit, other features that could be added to the tool are: parameter studies, sensitivity analyses, design of experiments, uncertainty quantification, and model simplification by means of surrogate models.
- The development of an Application Programming Interface (API) independent of the GUI could be useful for integrating FMI optimization capabilities in other

tools. It could be applied to offline optimization, as well as to online optimization, for example in MPC.

- Parallelization of the software tool could drastically reduce the optimization time. Dakota offers capabilities for parallelization. If we consider parallelization, as well as an API, executable programs could be generated and executed in high-performance clusters to further reduce the computational time.

Acknowledgments

This work has been funded by the National Plan Project DPI2014-56364-C2-1/2-R (ENERPRO-EFFERDESAL) of the Spanish Ministry of Economy, Industry and Competitiveness and ERDF funds.

References

Brian M. Adams, Mohamed S. Ebeida, Michael S. Eldred, Gianluca Geraci, John D. Jakeman, Kathryn A. Maupin, Jason A. Monschke, Laura P. Swiler, J. Adam Stephens, Dena M. Vigil, and Timothy M.Wildey. Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.5 User's Manual, 2016.

Karsten Ahnert and Mario Mulansky. Odeint - Solving ordinary differential equations in C++. In *AIP Conference Proceedings*, volume 1389, pages 1586–1589, 2011. ISBN 9780735409569. doi:10.1063/1.3637934.

Johan Åkesson, Karl-Erik Årzén, Magnus Gäfvert, Tove Bergdahl, and Hubertus Tummescheit. Modeling and Optimization with Optimica and JModelica.org—Languages and Tools for Solving Large-Scale Dynamic Optimization Problems. *Computers and Chemical Engineering*, 34(11):1737–1749, 2010.

Berkeley Lab. BuildingsPy - Modelica Buildings Library, 2016. URL `http://simulationresearch.lbl.gov/modelica/buildingspy/`.

Dassault Systemes. Dymola 2017 FD01, 2016. URL `http://www.dymola.com`.

John Eddy and Kemper Lewis. Effective Generation of Pareto Sets Using Genetic Programming. In *ASME 2001 Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, number 1, pages 1–9, Pittsburgh, PA, 2001.

Hilding Elmqvist, Hans Olsson, Sven Erik Mattsson, Dag Brück, Christian Schweiger, Dieter Joos, and Martin Otter. Optimization for Design and Parameter Estimation. In *Proc. 4th International Modelica Conference*, 2005.

ESI ITI GmbH. SimulationX 3.8, 2016. URL `http://www.simulationx.com/`.

Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, 2005. ISSN 0098-3500. doi:10.1145/1089014.1089020.

KDE Community. Breeze icons, 2016. URL `https://github.com/KDE/breeze-icons`.

MapleSoft. MapleSim 2016, 2016. URL `https://www.maplesoft.com/products/maplesim/`.

Modelica Association. Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0, 2014a. URL `https://www.fmi-standard.org/downloads`.

Modelica Association. Modelica Specification, version 3.3 Revision 1, 2014b. URL `http://www.modelica.org/documents`.

OSMC. OpenModelica 1.9.7, 2016. URL `http://www.openmodelica.org/`.

Andreas Pfeiffer. Optimization Library for Interactive Multi-Criteria Optimization Tasks. In *Proc. 9th International Modelica Conference*, pages 669–680, Munich, Germany, nov 2012.

Sandia Corporation. Dakota Packages, 2016. URL `https://dakota.sandia.gov/content/packages`.

Scilab Enterprises. Scilab: Open Source software for numerical computation, 2015. URL `http://www.scilab.org/`.

SciPy developers. SciPy.org - Python-based ecosystem of open-source software for mathematics, science, and engineering, 2017. URL `http://scipy.org/`.

Sergey A. Tachenov. QuaZIP - Qt/C++ wrapper for ZIP/UNZIP package, 2016. URL `http://quazip.sourceforge.net/`.

The MathWorks Inc. MATLAB R2016b, 2016. URL `http://www.mathworks.es/products/matlab/`.

The Qt Company. Qt - Cross-platform software development for embedded & desktop, 2016. URL `https://www.qt.io`.

Hubert Thieriot, Maroun Nemer, Mohsen Torabzadeh-Tari, Peter Fritzson, Rajiv Singh, and John John Kocherry. Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms. In *Proc. 8th International Modelica Conference*, pages 756–762, 2011.

Luigi Vanfretti, Maxime Baudette, Achour Amazouz, Tetiana Bogodorova, Tin Rabuzin, Jan Lavenius, and Francisco José Goméz-López. RaPId: A modular and extensible toolbox for parameter estimation of Modelica and FMI compliant models. *SoftwareX*, 5:144–149, 2016. ISSN 23527110. doi:10.1016/j.softx.2016.07.004.

Michael Wetter. GenOpt - A Generic Optimization Program. *Seventh International IBPSA Conference*, (1):601–608, 2001.

Edmund Widl, Wolfgang Muller, Atiyah Elsheikh, Matthias Hortenhuber, and Peter Palensky. The FMI++ library: A high-level utility package for FMI for model exchange. *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, MSCPES 2013*, 2013. doi:10.1109/MSCPES.2013.6623316.

Wolfram. SystemModeler 4.3, 2016. URL `http://www.wolfram.com/system-modeler/`.