# Analysis and reduction of models using Persalys

Claire-Eleuthèriane Gerrer[1] Hubert Blervaque[2] Julien Schueller[1] Daniel Bouskela[3] Sylvain Girard[1]

[1]Phimeca Engineering, France, {gerrer, schueller, girard}@phimeca.com
[2]Hubert Modelisation, France, hubertblervaque@gmail.com
[3]PRISME department, EDF, France, daniel.bouskela@edf.fr

## Abstract

Advanced computer experiments enable to understand and optimise a model. However, using these methods require skills in programming and a deep understanding of their underlying theory. Persalys is a software, based on OpenTURNS methods, which guides the user in the analysis and makes computer experiments accessible to non-programmers. This article aims to illustrate the use of Persalys on an intelligible use case : a solar collector from ThermoSysPro library.

We first performed a sensitivity analysis on the Modelica model exported as FMU. We then employed Persalys capabilities to optimize and metamodel the solar collector. We finally included the metamodel in the ThermoSysPro concentrated solar power plant to observe its performance.

Persalys is developed for both Windows and Linux. We succeeded in including metamodels on both OpenModelica Connection Editor (OMEdit) and Dymola, in the form of Modelica block or FMU.

*Keywords: FMI, FMU, OtFMI, Persalys, ThermoSysPro, sensitivity analysis, screening, metamodelling, model reduction, ExternalFunction.*

## 1 Introduction

Analysing Modelica models is necessary to understand their functioning and assert their reliability. Modellers do it naturally when they modify inputs or parameters and check the output variations. This verification step however demands lots of handwork when it is performed in modelling GUIs, be it OMEdit or Dymola.

Persalys[1] is a user-friendly Python-based GUI dedicated to the treatment of uncertainty and the management of variabilities. Persalys enables the analysis of Modelica models via the FMI standard. The software can be employed on all kinds of FMUs as well as with most finite elements models. FMUs (Functional Mockup Units) are models, written in Modelica or another 1D-modelling language, exported under the FMI standard. This standard provides an interface between different languages, such as Amesim, Simulink, Python.

Persalys is based on the Python library for the treatment of uncertainties, risks and statistics OpenTURNS[2](Baudin et al., 2015). The GUI is developed by part of the OpenTURNS consortium, EDF and Phimeca Engineering, to help and guide non-specialists in model analysis (see figure 1).

Aside from model analysis, Persalys enables to perform *metamodelling*, or *model reduction*, in a guided way. Metamodelling consists in learning the behaviour of the physical model using a mathematical function whose computation time is negligible (usually in milliseconds and below). Metamodels are thus a handy expedient to prohibitive computation times when many model runs are needed. Embedding metamodels in Modelica GUIs:

- permits to connect the metamodel with Modelica blocks for larger-scale modelling,
- enables the inclusion of (metamodelled) external models such as finite-element models.

Our purpose is to demonstrate how to, *using only GUIs*, thoroughly analyse a FMU, metamodel it and employ the metamodel in a Modelica environment.

The Modelica use case is detailed in section 2. The behaviour of a ThermoSysPro component, exported as FMU, using Persalys is explored in section 3. Section 4 shows how we computed the corresponding reduced model and exported it from Persalys. We finally discuss the use of the reduced model as FMU or Modelica component in section 5.

## 2 The Modelica use case

We consider a Concentrated Solar Power Plant (CSPP) transforming solar radiation to electricity. This industrial installation provides up to 1 MWe.

### 2.1 The physics of a CSPP

The most visible part of a CSPP is its large amount of surfaces reflecting the sun. The Parabolic Trough Solar Collector (PTSC) contains a parabolic reflective surface and a receiver tube, see figure 2. The reflected solar energy is transferred to the transparent receiver tube, which contains an absorber tube coated with blackened nickel to

[1]https://persalys.fr

[2]https://openturns.github.io/www/

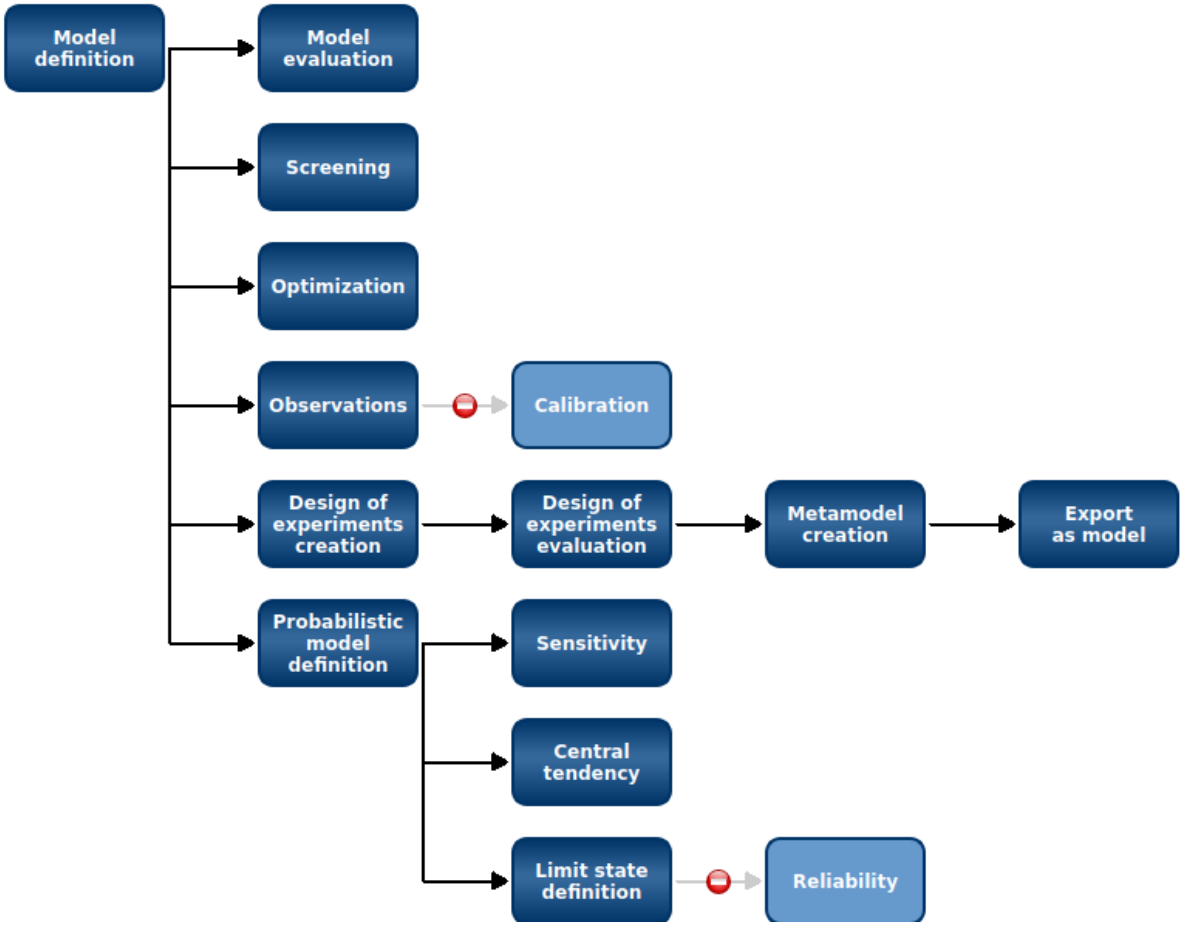



**Figure 1.** The analysis in Persalys is very guided. After defining the variables and outputs of interest, the user can evaluate the model, perform screening, optimization, etc. The no-entry sign on a method signals if a prior step was not completed. For instance, in the current analysis, the calibration method cannot be employed as observations must be provided first.

ensure high absorption. The absorber tubes heat up the synthetic oil it contains to nearly 400 °C.

The synthetic oil circuit exchanges heat with a circuit of water undergoing a Rankine cycle. The water is heated to dry steam and passes through a set of turbines in rotation, bringing an alternator in rotation to produce electricity. The water is then condensed and sent back to the heat exchangers, where it is heated again (Ferrière, 2008).

### 2.2 The modelling of a CSPP

The model *ConcentratedSolarPowerPlant_PTSC*[3] is an emblematic example of the opensource ThermoSysPro library [4]. ThermoSysPro provides components in the disciplines of thermal hydraulics and instrumentation and control for building nuclear, gas, coal or solar power plant models. The modelling choices concerning the concentrated solar power plant is detailed in (El Hefni, 2014) and (El Hefni and Bouskela, 2019). Figure 3 shows the entire power plant model, using ThermoSysPro v3.2.

The component in the upper part, called *SolarCollector*[5], models the PTSC. It is connected to the weather inputs: the sun radiation (direct normal incidence), sun incidence angle and atmospheric temperature. The *SolarCollector* component is connected to the primary circuit (which fluid is oil) exchanging heat with the secondary circuit. In this secondary circuit, the water is heated in three steps (by the economizer, heater and super-heater) before passing through the set of turbines.

### 2.3 A focus on the PTSC model

Before studying the entire solar power plant model, we focused on the PTSC model *SolarCollector*. Our purpose is to better apprehend its behaviour using statistical analysis and to replace it in the CSPP model by its metamodel.

We connected this component to ThermoSysPro interface component *HeatExchangerWall* and to a *HeatSource* setting the temperature value. We defined as output the heat flow produced by the solar collector.

This model, named *Env_PTSC*, is steady-state. We thus considered only the heat flow final value and constant weather inputs. We exported this model as FMU to study it using Persalys.

## 3 Apprehending the PTSC's behaviour

Modelica *parameters* and *inputs* have similar roles for statistical analysis as we consider a steady-state model. In the following, we call "variables" the model inputs and parameters under study.

We focus here on understanding the effect of the variables on the model output. We pre-selected 9 variables of the *SolarCollector*, with causalities *input* or *parameter*, to study their effects. We chose their variation bounds in accordance with the physics they represent, see table 1.

We loaded the PTSC model as FMU in Persalys. To gain insights in the model behaviour, we first ran screening and Sobol' sensitivity analyses (see subsection 3.1). We then checked if the optimal values of important variables correspond to our understanding of the physics (see subsection 3.2).

### 3.1 Screening and sensitivity analysis

Sensitivity analysis methods enable to determine the model variables which affect most the output. Morris method (also called "Morris screening") and Sobol' indices are two global sensitivity analysis methods (Iooss, 2011). Morris' qualifies the variables as *important*, *with nonlinear effect and/or interactions*, or *without effect*. Its results are rough, but the computation is rapid even with a large number of variables (Morris, 1991). Sobol' method quantifies the fraction of the output variation for which each variable is responsible. Sobol' indices are computationally expensive, but very precise.

We screened the important variables using Morris method, then studied more precisely their relative effect on the heat flow. Figure 5 shows the results of Morris screening for the 9 variables (based on 80 trajectories and 8 levels). The atmospheric temperature and the solar collector heat transfer coefficient, glass transmittivity and tube length variations have a negligible effect on the model output. We thus set these 4 variables "without effect" to their value in the model *ConcentratedSolarPowerPlant_PTSC*).

We refined the analysis of the effect of the 5 variables categorized as *important* using Sobol' sensitivity analysis. The results of Morris analysis can be used by Persalys to define the variables probabilistic model. Only the variables declared as significant are considered as uncertain: the others are fixed to their default value.

Figure 6 shows the results of Sobol' analysis in Persalys. The first-order index quantifies the effect of a variable *without interaction with any other variable* on the model output. The total index accounts for the effect of an variable alone and in interaction with other variables, on the output. The variables with the greatest Sobol' indices are the most influential (with respect to their given variation range). The heat flow produced by the solar panel thus mainly depends on the sun radiation and the incidence angle.

### 3.2 Variables optimal value

The value that maximises the model output is a complementary information to the relative influence of the variables. In theory, the heat flow produced is maximum when the sun radiation and mirror reflectivity are maximal, the angle incidence minimal. We checked the optimal combined value of the 5 variables in Persalys.

Figure 7 shows that the maximal heat flow is reached for maximal sun radiation and mirror reflectivity with

[3] located in *ThermoSysPro.Examples.Book.PowerPlant*

[4] `https://thermosyspro.com`

[5] located in *ThermoSysPro.Solar.Collectors*

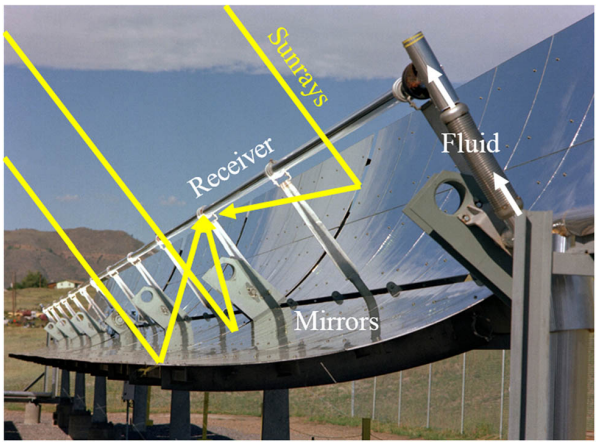



**Figure 2.** An example of PTSC. Source: (Tagle-Salazar et al., 2020)

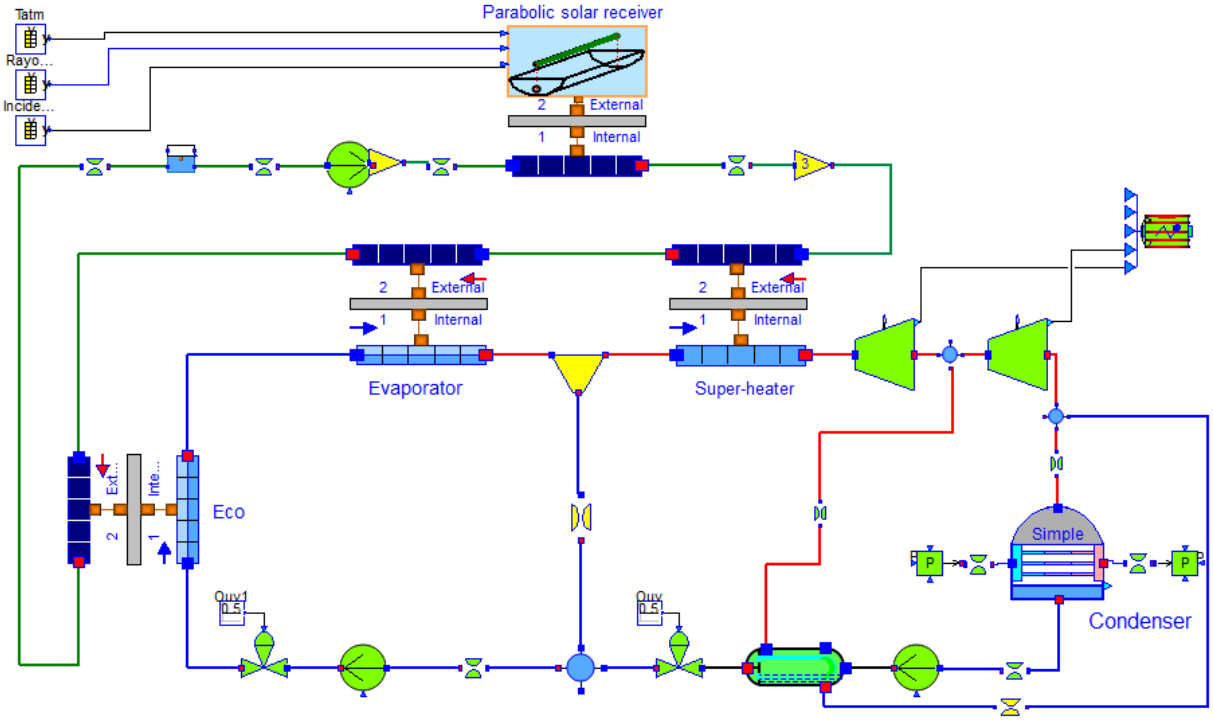



**Figure 3.** Modelica model of a concentrated solar power plant, designed by El Hefni and Bouskela (El Hefni, 2014).

| Variable group | Name | Unit | Role | Value | Bounds |
|---|---|---|---|---|---|
| Geometric parameters | L | m | Length of the PTSC tube | 450 | [400, 500] |
| | solarCollector.RimAngle | ° | Rim angle | 70 | [65, 75] |
| | solarCollector.f | m | Focal length | 1.79 | [1, 2] |
| | solarCollector.h | $W/m^2/K$ | Heat transfer coefficient between ambient air and glass envelope | 3.06 | [2.5, 5] |
| Cleanliness parameters | solarCollector.R | - | Mirror reflectivity | 0.93 | [0.6, 0.95] |
| | solarCollector.TauN | - | Glass transmittivity | 0.95 | [0.5, 0.95] |
| Weather inputs | T_atm.k | °K | Atmospheric temperature | 300 | [273, 303] |
| | angle_incidence.k | ° | Sun incidence angle (0° at zenith) | 0 | [0, 89] |
| | radiation.k | $W/m^2$ | Direct Normal Irradiance | 700 | [0, 1000] |

**Table 1.** Description of the 9 pre-selected variables of the PTSC models.

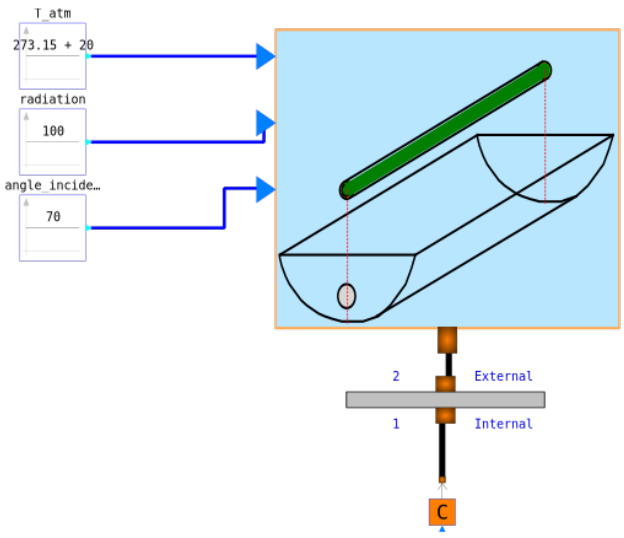



**Figure 4.** The *Env_PTSC* model relates the sun radiation, environment temperature and sun incidence to the heat flow produced by the *SolarCollector*.

minimal incidence angle as expected. The rim angle and the focal length are maximal, meaning that the mirrors have a large surface and "wrap" the tube. These results are to consider with care, as the optimal point for the 5 variables may not be the same for each variable individually.

After gaining insights in the PTSC behaviour, we statistically learned this behaviour, i.e. we metamodelled the *Env_PTSC* model. In existing solar power plants, the geometric parameters of the collectors can not be modified: only the weather inputs and the mirror cleanliness parameter evolve. Persalys enables us to reduce the PTSC model with respect to these 3 variables.

# 4 Training and export of the metamodel

Metamodelling, aka model reduction, consists in learning the model behaviour by a mathematical function. A general purpose for metamodelling is to lower the computational cost, especially if many model runs are required (e.g. to perform statistics or real-time simulations). We reduced the PTSC model (a steady-state model) by learning the *heatFlow* final value with respect to the 3 aforementioned variables with constant value.

Prior steps to metamodelling are the definition of a design of experiment and the simulation on these points, see figure 1. We considered a Full Factorial Design (Friedman et al., 2001) with 10 levels for each of the 3 variables, i.e. 1000 simulations. The metamodel is computed by Kriging with constant mean and squared exponential covariance. The figure 8 shows the metamodel validation by K-fold: the cross-validation coefficient $Q^2$ is equal to 0.996, which is very satisfying.

We saved the metamodel as a Python pickle object[6] from Persalys command line (see listing 1).

**Listing 1.** Save Persalys' metamodel as Python Pickle object

```
import persalys as pls
import pickle

study = pls.Study.GetInstances()[0]
metamodel = study.getPhysicalModels()[1].
    getFunction()

filename = "solar_collector.pkl"
with open(filename, 'wb') as f:
    pickle.dump(metamodel, f)
```

The pickled metamodel can be loaded in Python and exported as a Modelica graphical component and/or a FMU.

# 5 Using a Persalys metamodel in Modelica GUI

Using the metamodel in a modelling environment enables to connect it graphically to other components. We inserted the metamodel as a power source in the concentrated solar power plant model to show the validity of the interface between Persalys and Modelica GUIs. Following the same methodology, metamodels reproducing the behaviour of models *from other environments (Excel, Amesim, Matlab...)* can be employed as blocks or FMUs in Modelica GUIs.

The metamodel is a mathematical function, computed by Persalys using OpenTURNS. The Python library OTFMI interfaces FMUs with OpenTURNS objects, and reciprocally (Girard and Yalamas, 2017)[7].

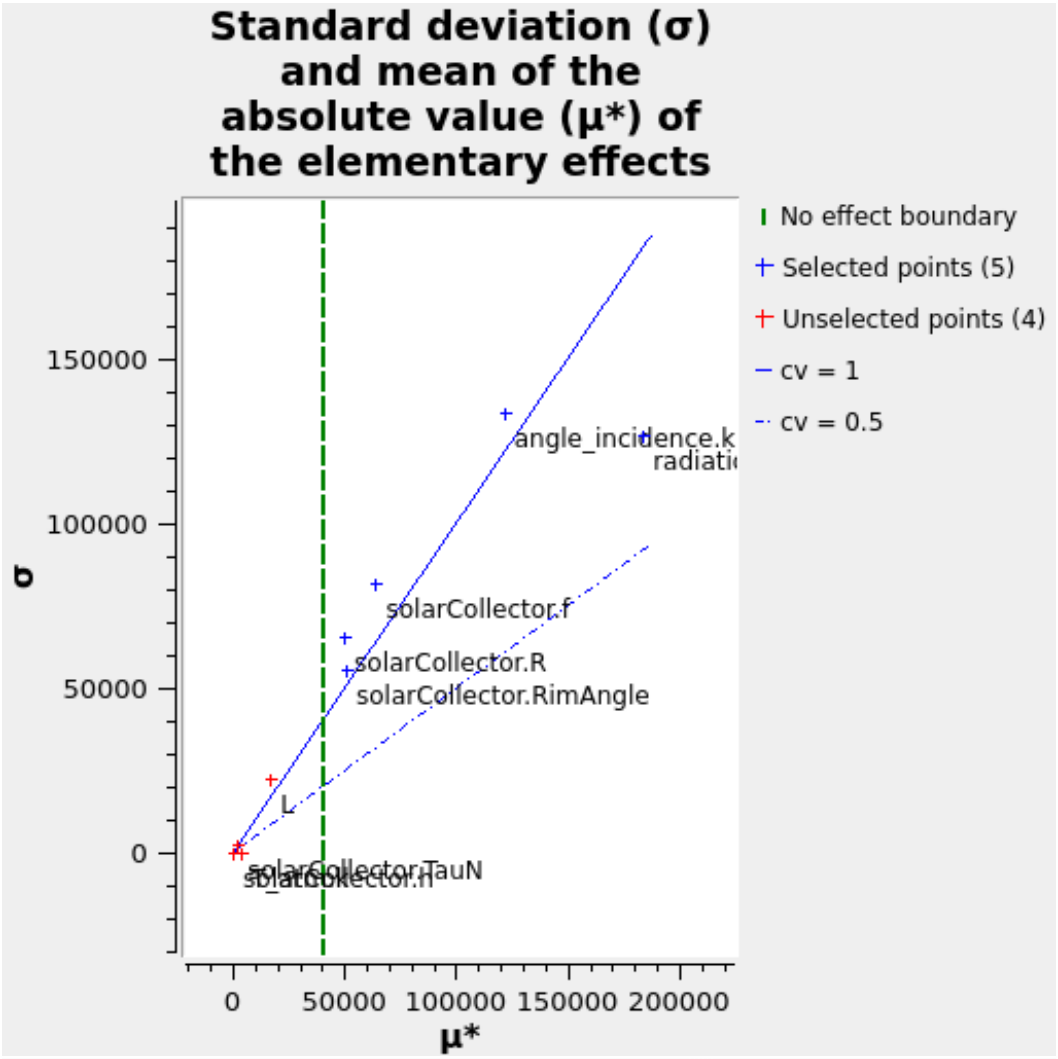
We compare the advantages of exporting Persalys metamodels as *Modelica blocks* or *FMUs* in subsection 5.1. We show the insertion of the metamodel in a Modelica model using a GUI in subsection 5.2.

## 5.1 Modelica block versus FMU

The metamodel can be used in Modelica as a block or as a FMU. The FMU has the advantage to be standalone: it embeds the contents required to run the metamodel, i.e. a C function and C libraries. The "block" is a Modelica wrapper of the C functions and XML file in which the metamodel is stored. The simulation is faster, as unzipping the FMU to access the underlying C is not necessary. However, the C files need to be all conserved and the wrapper path must be updated manually if the user moves the C files.

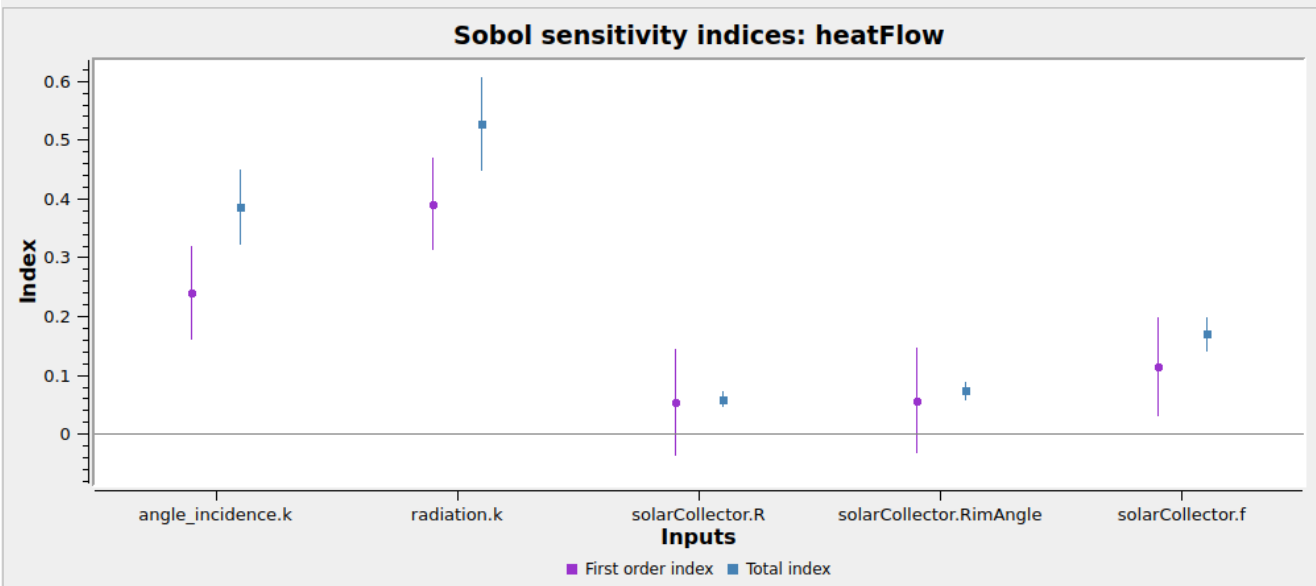
The metamodel can be exported in both formats using OTFMI, see listing 2. The function *export_model* exports the given metamodel in a Modelica wrapper. If the gui option is prescribed, input and output connectors are generated, which enables to connect the wrapper to other Modelica blocks. Otherwise inputs and outputs are defined using Modelica language, which enables to simulate the

[6] https://docs.python.org/3/library/pickle.html
[7] https://github.com/openturns/otfmi




| Variable | No effect | Linear effect | Non-linear effect or interaction | Non-monotonic effect |
|---|---|---|---|---|
| L | X | | | X |
| T_atm.k | X | | | |
| ☑ angle_incidence.k | | | X | |
| ☑ radiation.k | | | X | |
| ☑ solarCollector.R | | | X | |
| ☑ solarCollector.RimAngle | | | X | |
| solarCollector.TauN | X | | | |
| ☑ solarCollector.f | | | X | |
| solarCollector.h | X | | | X |

**Figure 5.** Top: Graph $\mu^*/\sigma$ drawn in Persalys GUI. The "no effect boundary" must be set by the user.
Bottom: Interpretation of the graphs $\mu^*/\sigma$ and $\mu^*/\mu$ in Persalys GUI.




| Input | First order index | First order index confidence interval | Total index | Total index confidence interval |
|---|---|---|---|---|
| angle_incidence.k | 0.239918 | [0.160932, 0.318904] | 0.38588 | [0.322624, 0.449135] |
| radiation.k | 0.391587 | [0.31463, 0.468544] | 0.526854 | [0.44882, 0.604888] |
| solarCollector.R | 0.0549183 | [-0.0347596, 0.144596] | 0.0595337 | [0.0464649, 0.0726025] |
| solarCollector.RimAngle | 0.0574346 | [-0.0314311, 0.1463] | 0.0734052 | [0.0589891, 0.0878213] |
| solarCollector.f | 0.115092 | [0.0321582, 0.198026] | 0.170077 | [0.142031, 0.198123] |

**Figure 6.** Up: the first-order and total Sobol' indices of the 5 variables considered.
Bottom: the computed Sobol' indices. Insufficient convergence of the estimates is signalled by warning panels.

Optimal point

| | Variable | Value |
|---|---|---|
| Output | heatFlow | 60449.7 |
| Inputs | angle_incidence.k | 33 |
| | radiation.k | 63 |
| | solarCollector.R | 0.837 |
| | solarCollector.RimAngle | 63 |
| | solarCollector.f | 1.611 |

Optimization result

| | |
|---|---|
| Number of function evaluations | 145 |
| Absolute error | 5.19903e-06 |
| Relative error | 8.25243e-08 |
| Residual error | 6.85487e-08 |
| Constraint error | 0 |

**Figure 7.** Display of optimisation results in Persalys GUI.

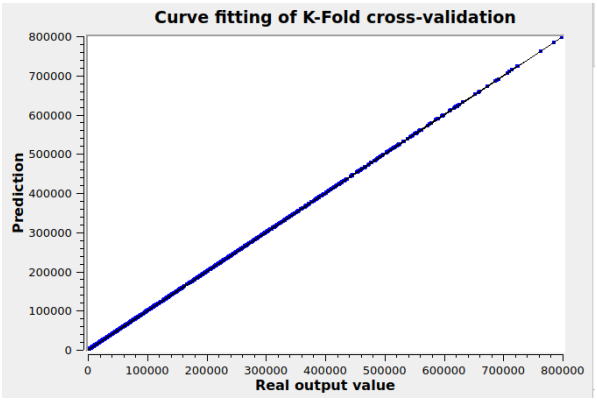



**Figure 8.** Metamodel validity using the K-fold method.

model via OMCompiler. The function *export_fmu* generates a FMU from a temporary Modelica wrapper. Both ModelExchange or CoSimulation formats are supported.

**Listing 2.** Export the metamodel as component or FMU

```
fe = otfmi.FunctionExporter(metamodel,
    point)
fe.export_model(path_model, verbose=False,
    gui=True)
fe.export_fmu(path_fmu, verbose=False)
```

These functionalities are supported on Windows and Linux platforms. The generated component can be included in OpenModelica Connection Editor (OMEdit) as well as in Dymola.

### 5.2 Inserting the metamodel in a Modelica model

We wrapped the metamodel in a Modelica block with graphical input and output connectors. The figure 9 shows the ThermoSysPro *ConcentratedSolarPowerPlant_PTSC* model, which solar panel has been replaced by heat source whose value is prescribed by the metamodel. We call this model *PartlyReducedCSPP*. We successfully calibrated and ran the *PartlyReducedCSPP* model in Dymola.

At every time step of the simulation, the metamodel value is solicited by the solver (it is not co-simulation). The initialization and integration steps last longer for the partly reduced CSPP model than for the original CSPP, see table 3. The table 2 shows that, even though the metamodel is computationally less expensive than the Modelica model, its inclusion in the Modelica environment increases the simulation time.

| | Metamodel | | *Env_PTSC* |
|---|---|---|---|
| Simulation tool | Python | Modelica | Modelica |
| Run time (s) | $5 \times 10^{-5}$ | 0.5 | $3 \times 10^{-3}$ |

**Table 2.** The metamodel simulates in Python up to 100 times faster than the corresponding model in Modelica.

We have two hypotheses to explain this difference. First, the computation of the metamodel derivatives (necessary to set the solver time steps) is possibly difficult. Each derivative must be estimated by the Modelica solver as the metamodel analytical function is not accessible. This increase in simulation time could thus be due to a larger number of calls to the metamodel.

Second, three different languages (Python, C and Modelica) are employed for one computation of the metamodel in the Modelica environment. This could be improved in the following by employing the C implementation of OpenTURNS library instead of its Python implementation.

## 6 Conclusion and perspectives

Asserting the behaviour of a model and exploiting its results are very important tasks for a modeller. We presented in this paper technological solutions to meet these needs while staying (mostly) in graphical interface.

The Persalys software enables the statistical analysis of Modelica (and others) models. Its strength resides in the guidance for the analysis, performed in a very visual way. We showed the use of Persalys for selecting the influential variables of the PTSC by Morris and Sobol' sensitivity analyses. We leveraged the results of these analyses and our physical knowledge to successfully reduce the model. We showed that the reduced model can be easily imported in a Modelica GUI, be it OMEdit or Dymola, and connected to other Modelica blocks. This technological bridge can be used in a large scope of applications, for instance to introduce reduced finite-elements models in the Modelica environment.

The implementation in Persalys of statistical methods for time-dependent outputs is underway. This will open new possibilities to study models which output(s) evolve over time without reaching a steady state. We also intend to increase the simulation speed of the reduced model by suppressing the underlying call to Python and directly using the C-implementation of OpenTURNS.

## Acknowledgement

We thank warmly Arnaud Barthet, Pan Zhou and Li Xiao (M4G team, EDF Chine) for sharing with us their experience of solar collectors.

This work was partially supported by the Paris region through the FUI research project "Modeliscale", a collaboration with Dassault Systèmes, Inria, EDF, Engie, CEA INES, DPS, Eurobios and Phimeca Engineering.

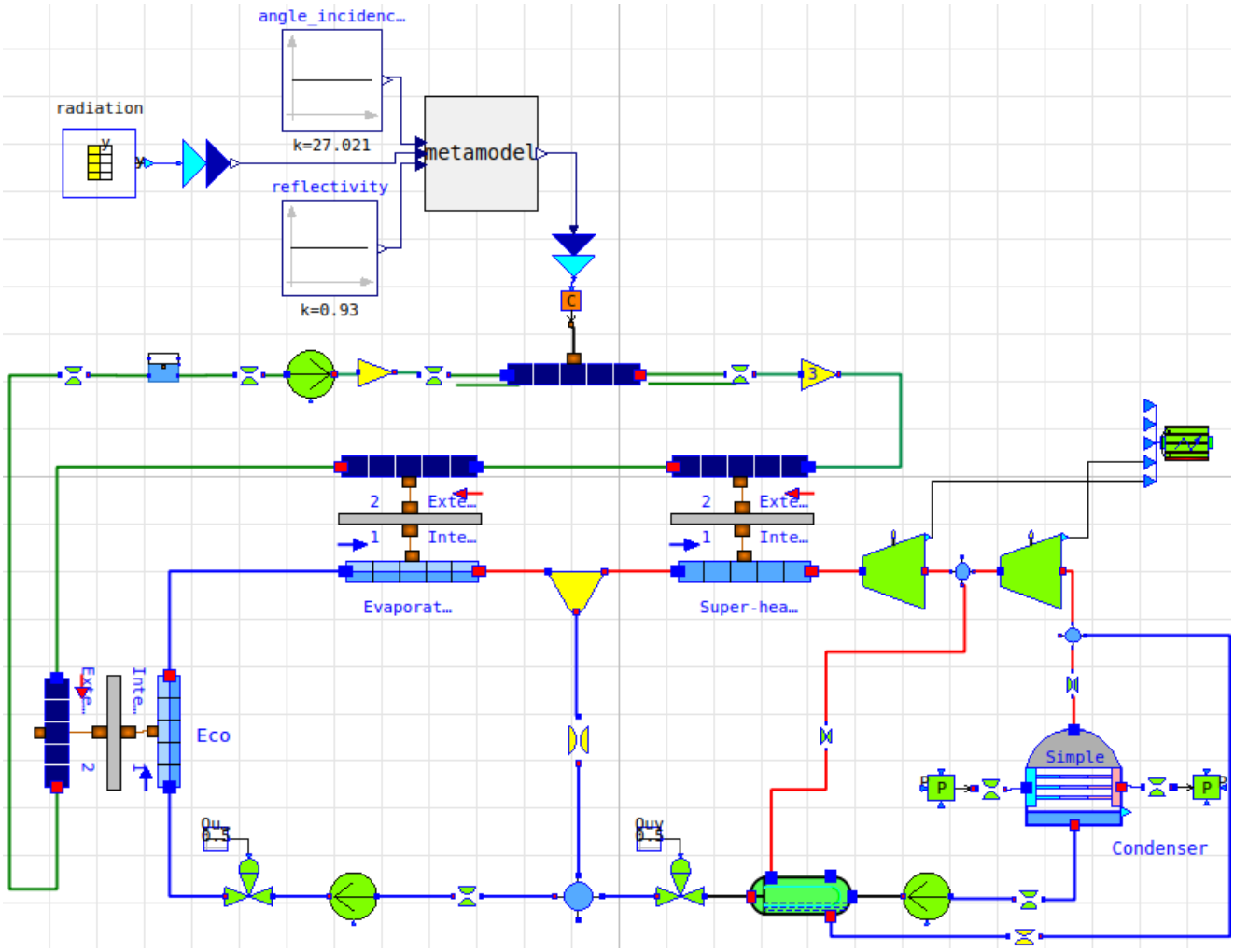



**Figure 9.** The *PartlyReducedCSPP* model, based on ThermoSysPro emblematic *ConcentratedSolarPowerPlant_PTSC*.

| | *PartlyReducedCSPP* | *ConcentratedSolarPowerPlant_PTSC* |
|---|---|---|
| CPU time for initialization (s) | 0.345 | 0.179 |
| CPU time for integration (s) | 993 | 109 |

**Table 3.** The initialization and integration times of the partly reduced CSPP are larger than for the original ThermoSysPro model.

# References


Michaël Baudin, Anne Dutfoy, Bertrand Iooss, and Anne-Laure Popelin. Openturns: An industrial software for uncertainty quantification in simulation. 2015.

Baligh El Hefni. Dynamic modeling of concentrated solar power plants with the thermosyspro library (parabolic trough collectors, fresnel reflector and solar-hybrid). *Energy Procedia*, 49: 1127–1137, 2014.

Baligh El Hefni and Daniel Bouskela. *Modeling and Simulation of Thermal Power Plants with ThermoSysPro*. Springer, 2019.

Alain Ferrière. Centrales solaires thermodynamiques, 2008. URL `https://www.techniques-ingenieur.fr/base-documentaire/energies-th4/energies-renouvelables-42594210/centrales-solaires-thermodynamiques-be8903/technologies-solaires-a-concentration-be8903niv10001.html`.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Sylvain Girard and Thierry Yalamas. A probabilistic take on system modeling with modelica and python. Technical report, Phimeca Engineering, 2017. `https://www.researchgate.net/publication/321624302_A_Probabilistic_take_on_system_modeling_with_Modelica_and_Python`.

Bertrand Iooss. Revue sur l'analyse de sensibilité globale de modèles numériques. *Journal de la Société Française de Statistique*, 152(1):1–23, 2011.

Max D Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

Pablo D Tagle-Salazar, Krishna DP Nigam, and Carlos I Rivera-Solorio. Parabolic trough solar collectors: A general overview of technology, industrial applications, energy market, modeling, and standards. *Green Processing and Synthesis*, 9(1):595–649, 2020.