Calibration Workflow for Mechanical and Thermal Applications

Tim Willert[1] Peter Sundström[2]

[1]Modelon K.K., Japan, tim.willert@modelon.com
[2]Modelon AB, Sweden, peter.sundstrom@modelon.com

Abstract

The calibration of models against measurement data is important to ensure model dynamics that are close to its real-world system. Derivative-free minimizing methods can be used for any model calibration regardless of continuous differentiability requirements, and find a (local) minimum in a reasonable number of iteration steps.

A user-friendly, python-based calibration Dash app to use with the cloud-based Modelica platform Modelon Impact is introduced. Basic calibration setup is done through the GUI of the app and graphical feedback (i.e. plots) is provided.

Two example calibrations are shown: A mechanical Furuta pendulum that only uses Modelica Standard Library components is calibrated against real-world measurement data, and a low-fidelity heat exchanger testbench model that uses Modelon's Air Conditioning Library is calibrated against a corresponding high-fidelity model.

Keywords: calibration, Modelon Impact, Dash, Nelder-Mead, Modelica, Air Conditioning Library, optimization

1 Motivation

Physical modeling can be used to optimize existing systems, predict the behaviour of a system under different boundary conditions, or design new systems.

When working with a system that exists in the real world, it is important that the system model shows the same dynamic behavior as the existing, real-world system with reasonable accuracy. This requires model calibration of relevant system parameters using real-world measurement data.

When working on the detailed design of a complex component, it is common to use testbenches with a high-fidelity model. The finalized model is then integrated in a larger system model. Often, this requires to switch from a high-fidelity to a low-fidelity model to allow for reasonable computation times. The uncalibrated low-fidelity model can behave differently from the high-fidelity model. Therefore, it is required to calibrate the low-fidelity model using high-fidelity model simulation data.

2 Optimization

Following Olsson, Mattsson, and Elmqvist (2006), we can mathematically describe the general problem with a number of measured/simulated inputs v_i, outputs w_i, and a relation between them

$$w_i = m(p, v_i) \tag{1}$$

where $m()$ describes our model function, and p describes system parameters. The goal of the optimization problem is to minimize the residuals by finding the optimal set of parameters p:

$$r_i(p) = m(p, v_i) - w_i \tag{2}$$

Typically, we are interested in more than one residual. Therefore, we can weight and combine all of the residuals into one scalar to be minimized:

$$f(p) = \sum_i k_i r_i^2(p) \tag{3}$$

where k_i is a weighting factor. Note that here the least squares formulation is used. The choice of good weighting factors is difficult. Finding good weighting factors has to be done with the problem formulation in mind and a good understanding of the underlying equations. It is also possible to formulate it using the square root or even custom error functions. Our optimization problem can then be represented as

$$f(p_{\text{opt}}) \leq f(p) \tag{4}$$

with f as our objective function, and p_{opt} as the optimal set of parameters. In this case, p can be the set of all possible design parameters and therefore the equation describes the global optimum. This is typically unfeasible to solve for. In the following, it is assumed to solve for a local optimum.

There are different methods and approaches to find the optimal solution for the problem at hand.

Normally, a sensitvity analysis is carried out preliminary to a calibration. This will not be discussed here.

In the following subsections, three optimization methods are described.

2.1 Parameter Sweep

A parameter sweep is a simple method where simulations are run repeatedly with a range of different parameter values. The batch of simulation results (outputs) can then be compared to the target data (inputs). Using a compare algorithm or visualization tools, parameters can be picked.

Since each simulation run with a set of parameters is independent from the others, each can be run completely in parallel. This method does not pose any model restrictions in terms of continuity, differentiability and/or events.

As picking the set of parameters is not based on an optimization algorithm, it is unlikely to find an optimal solution within a reasonable number of iteration steps.

2.2 Advanced Methods

For the advanced methods, picking a set of parameters follows a certain pattern. Typically, after a few initializing simulation runs, the next set of parameters is picked based on previous results and/or on (local) derivatives. Whether derivatives are used or not has different implications for the model architecture and end applications. This iterative process continues until a termination condition is met, e.g. the difference of the objective function results from consecutive simulation runs falls below a threshold.

In general, it is recommended to give bounds to the system parameters. Otherwise, it is possible to find a mathematically optimal solution which is unphyscial (e.g. negative joint friction) or economically not feasible (e.g. heat exchanger with the size of a space ship). Since the minimization can be sensitive to the choice of initial parameter sets, it is common to run the optimization method of choice several times with different initial parameters.

2.2.1 Derivative-free Methods

Derivative-free methods involve algorithms that work without using derivatives. The methods can be categorized depending on the type of method and shape of objective function. For an extensive overview see Larson, Menickelly, and Wild (2019).

A prominent example is the Nelder-Mead method (Nelder and Mead 1965) which uses $n+1$ test points arranged as a simplex for an optimization problem with n parameters. By reflection, expansion, inner contraction and shrinkage of the simplex (see Figure 1), a local minimum can be found. For a description with an example

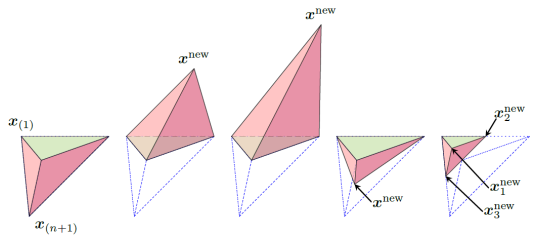

Figure 1. Nelder-Mead simplex operations: original simplex, reflection, expansion, inner contraction, and shrinkage (Larson, Menickelly, and Wild 2019).

implementation in C see Press et al. (1992).

Other examples of derivative-free optimization methods are the Powell method (see Powell (1964)) and the further developed COBYLA method (see Powell (1994)).

Since the objective function is not required to be continuous and differentiable, derivative-free optimization methods can be applied to any function. This is especially useful for complex models in Modelica that involve model discontinuities (such as media phase changes) or events. As opposed to the parameter sweep, with these methods it is more likely to find a (local) optimal solution with a limited number of iteration steps. Depending on the used algorithm, parts of the implementation can be parallelized, e.g. the calls to the objective function within one iteration of the Nelder-Mead algorithm.

For (two times) continuously differentiable objective functions, these methods can be inefficient to find the (local) minimum.

2.2.2 Derivative-based Methods

Derivative-based methods involve algorithms that use derivatives of the objective function. This means our underlying system model needs to be continuously differentiable or have even stricter requirements. Depending on the used method it might also be necessary to explicitly give the derivative in form of the Jacobian as an extra input.

This sub-category of optimization methods can for example be used for techno-economic assessments of well formulated system models (Köppen et al. 2022). Since our focus is the calibration of general mechanical and thermal applications, regardless of differentiability and continuity, further explanation of derivative-based methods is left out.

2.3 Existing Frameworks

In this subsection, some existing frameworks are introduced. One of them has been tested with the thermal application described below. The others serve as a reference for an interested reader to test and compare.

All of the introduced frameworks use functional mock-up units (FMU) and work in a python environment. This makes the user independent from Modelica tools, but is as a stand-alone workflow less integrated in the modeling process.

ModestPy

ModestPy (see Arendt et al. (2018)) is a discontinued open-source python package for parameter estimation in FMUs. Available algorithms are genetic algorithm, (legacy) single-process genetic algorithm, pattern search and some SciPy solvers. These methods can be used in a sequence within the framework, which makes it a powerful framework. It outputs several plots that help to analyze the results and parameter interdependencies. The whole package is command line based and changes to the setup are done in the python code. There is no direct graphical interface.

For an inexperienced user, the framework can be difficult to use even for simple applications. Without the knowledge of the different algorithms and how to effectively use them sequentially, this framework can be too complicated.

EstimationPy

Similarly to ModestPy, EstimationPy (see *EstimationPy* (2023)) is a Python package for the estimation of state and parameters of dynamic systems that conform to FMI standard. It uses the packages NumPy and SciPy for the computation , is compatible with Pandas DataFrames and DataSeries and relies on PyFMI and Assimulo to run the model simulations. It goes beyond typical calibration problems, such as model-based fault detection. It also gives the user the option to use Kalman filter to solve state estimation problems. The whole package is command line based and changes to the setup are done in the python code. There is no direct graphical interface.

For an inexperienced user, the package can be difficult to use even for simple applications.

AixCaliBuHA

AixCaliBuHA (see Wüllhorst et al. (2022)) is a framework that aims at automatizing the process of calibrating models used in Building and HVAC simulations. As opposed to the aforementioned python packages, this framework has a focus on calibrating models. It includes the capability to perform a sensitivity analysis and several visualization options that help to analyze the results. For the optimization method, it is possible to choose between SciPy's differential evolution method, SciPy's minimize methods, and dlib's minimize (implemented in C++). The whole package is command line based and changes to the setup are done in the python code. There is no direct graphical interface.

For an inexperienced user, the package can be difficult to use even for simple applications.

This framework was tested with the thermal application described here and delivered reasonable results.

3 Technical setup

In the following, the general calibration setup for two different models - one mechanical and one thermal - is outlined. The model specific setup is described in section section 4.

On the modeling side, the cloud-based platform Modelon Impact is used. It is not necessary to manually convert the model into a FMU. There are no specific requirements to the model itself. A python-based dashboard app created in Dash (https://dash.plotly.com) connects to Modelon Impact using the Impact Python Client (https://github.com/modelon-community/impact-client-python). The dashboard uses the client to set up the workspace, get and set parameter values and compile and run the model. Dash is low-code frameowrk to build data apps in Python. The used optimization algorithm is the Nelder-Mead algorithm that is one method included in `scipy.optimize.minimize`. Note that it is also possible to use the Powell or Cobyla method from the same python module or `differential_evolution`, as these are derivative-free methods as well. The underlying objective function is constructed as the sum of the squared error over the time window for the calibration. A simplified graph of the base setup can be seen in Figure 2.

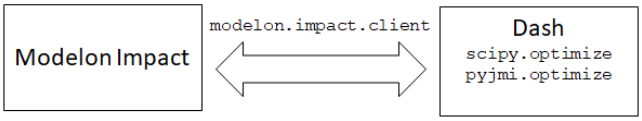

Figure 2. Simplified graph of software setup.

4 Applications

The calibration app is structured with three tabs: Model, Measurement, and Calibration.

The Model tab is used to pick a workspace that is stored in the user's Modelon Impact account, and the model of interest inside that workspace. The picked model is then to be compiled and simulated for an adjustable stop time. For verification, model variables can be plotted.

The Measurement tab is used to load in the measurement data. Allowed formats are `.mat` and `.csv`. For verification, the data can be plotted.

The Calibration tab is used for the actual calibration process. Model variables of interest (output w_i as in section 2) are assigned manually to the corresponding measurement variable (input v_i as in section 2). The variables can be weighted (weight k_i in section 2). The relevant system parameters (p in section 2) are picked and the bounds can be assigned as well as a nominal value which serves as start value. It is possible to pick multiple system parameters for a single calibration. In the following examples, only two parameters are picked for each application. Note that with each extra parameter, the number of iteration steps will increase and thus can become unfeasible to solve the calibration within a reasonable amount of time. The calibration time interval can be picked and finally the calibration algorithm is started. When the calibration is finished, a plot shows the nominal model variables, measurement variables and the calibrated model variables. The nominal and calibrated simulation run results are stored in the model in Modelon Impact.

4.1 Mechanical

A real-world furuta pendulum is modeled in Modelica. For the model to show the same behavior as the real-world system a calibration is necessary.

The Furuta pendulum consists of an arm rotating in the horizontal plane and a pendulum which is free to rotate in the vertical plane. The construction has two degrees of freedom, the angle of the arm, φ, and the angle of the pendulum, θ. The real-world system is shown in Figure 3.

The corresponding Modelica model is modeled using Modelica Standard Library components only. The revolute joint `armJoint` is connected to the world component and the horizontal arm, and allows rotational

Figure 3. Photograph of Furuta pendulum.

movement in the global y-axis. Another revolute joint, `pendulumJoint`, connects the horizontal arm with the vertical pendulum, and allows rotational movement in the local x-axis. Both revolute joints are connected through their axis flange to a bearing friction component to allow friction modeling. The model can be seen in Figure 4.

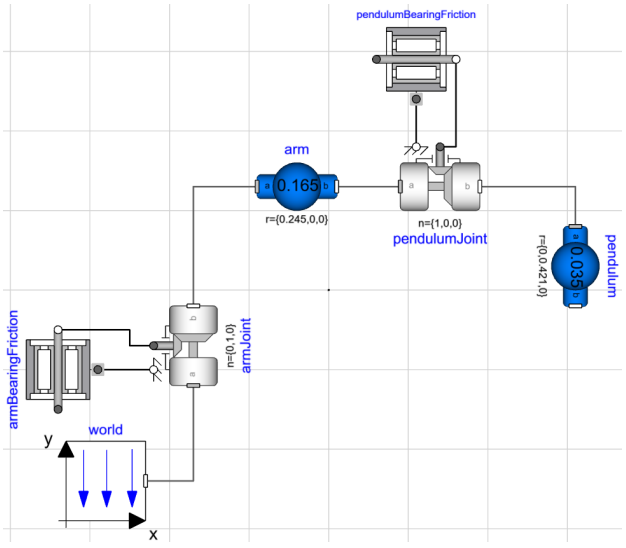

Figure 4. Modelica model of Furuta pendulum.

Experimental data of the real-world pendulum is used for the calibration (input v_i as in section 2). Variables of interest are arm angle φ, and the pendulum angle θ.

Correspondingly, model variables of interest (output w_i as in section 2) are the angle of the arm joint `armJoint.phi`, and the angle of the pendulum joint `pendulumJoint.phi`. Both variables are weighted equally with factor 1.

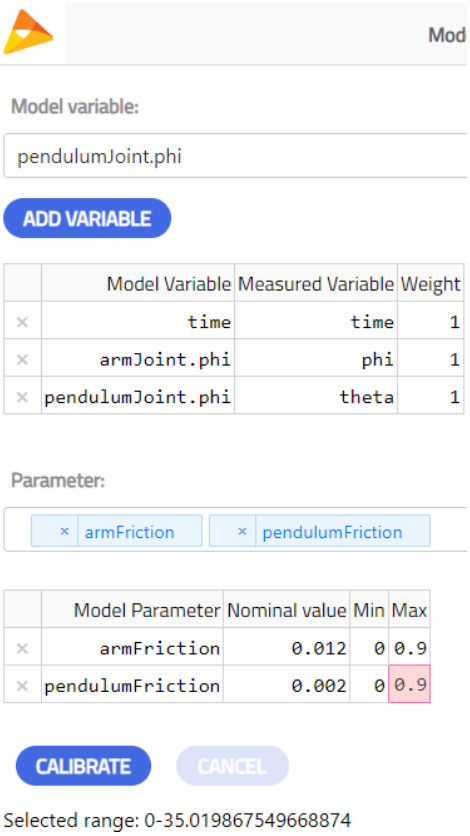

Figure 5. Furuta calibration setup in the calibration app's GUI.

The parameters of interest for the calibration process are the bearing friction coefficient for the arm joint `armFriction`, and the bearing friction coefficient for the pendulum joint `pendulumFriction`. For `armFriction`, we use 0.012 as nominal value, and for `pendulumFriction` we use 0.002 as nominal value. For both parameters, we set the bounds as 0 and 0.9.

Calibration time interval is set from approximately 0 to 35 seconds.

An overview of all the settings in the calibration app can be seen in Figure 5.

It took 27 iteration steps to find a local minimum at

$$\texttt{armFriction} = 0.010125 \tag{5}$$

$$\texttt{pendulumFriction} = 0.00117 \tag{6}$$

Figure 6 and Figure 7 respectively show the the arm angle and the pendulum angle over time for the experimental data, and the nominal and calibrated case of the model.

In comparison to the uncalibrated model, we can see that the result of the calibrated model is in good agreement with the experimental data. The value of the objective function has decreased from $1.645 \cdot 10^2$ to 2.729.

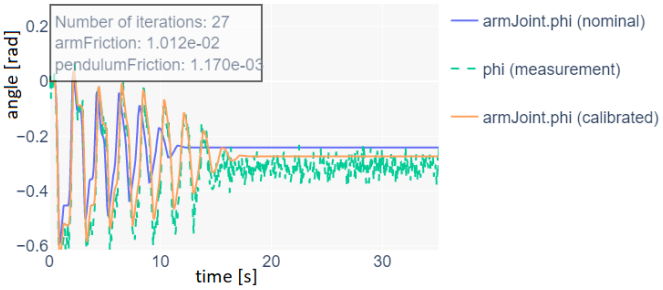

Figure 6. Arm angle over time curves of experimental data (green), and nominal (blue) and calibrated case (orange) for system model.

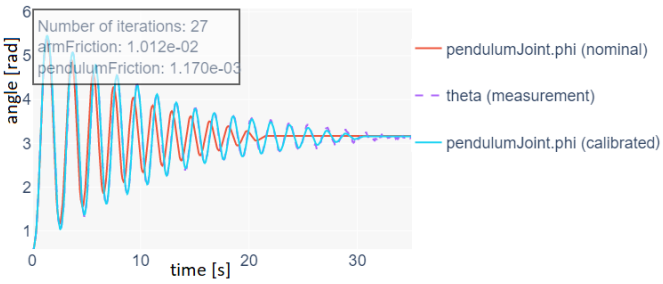

Figure 7. Pendulum angle over time curves of experimental data (purple), and nominal (red) and calibrated case (light blue) for system model.

4.2 Thermal

A heat exchanger was designed as a high-fidelity model. This heat exchanger is now to be used in a complex system model requiring fast respond times. This requires a computationally less expensive model, a low-fidelity model. For the low-fidelity model to show the same behavior as the high-fidelity model, a calibration is necessary.

In our example, we are using a testbench model for microtube heat exchanger models from Modelon's Air Conditioning Library (see Figure 8). A testbench is a type of

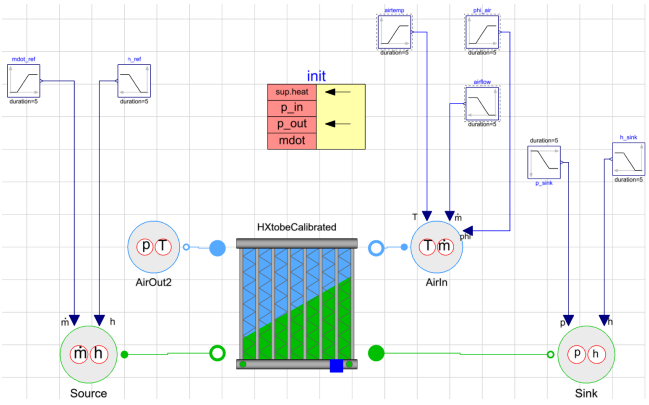

Figure 8. Testbench model for a microtube heat exchanger model using Modelon's Air Conditioning Library.

model that focuses on a single complex component. Adequate boundary conditions in the form of sources and sinks are connected to complex model that is to be tested. In this case, a heat exchanger is connected to a source and sink for the air and refrigerant side respectively. Except for the air sink, each source and sink is connected to a ramp signal for each variable. Since we are interested in the dynamic behavior, the boundary conditions (i.e. all ramp signals) are configured to change within 170 seconds starting with the first boundary condition after 60 seconds.

The low-fidelity heat exchanger testbench model extends the testbench model for the high-fidelity heat exchanger. The difference between the two testbench models is the pressure correlation of the refrigerant side in the heat exchanger model. The high-fidelity model uses `PlossCommon`, which is a Reynolds-based two-phase pressure loss model for laminar and turbulent flow based on Friedel (1979). The low-fidelity model uses `DensityProfilePressureLossHX`, which is a distributed pressure loss model for two-phase fluids using a density profile.

Measurement data (input v_i in section 2) is generated from the high-fidelity testbench model. Variables of interest are the pressure drop on the refrigerant side `dp_ref`, and the total heat flow on the refrigerant side `Qdot_refTotal`. Correspondingly, low-fidelity model variables of interest (output w_i in section 2) are the same: Pressure drop on the refrigerant side `dp_ref`, and the total heat flow on the refrigerant side `Qdot_refTotal`. Both variables are weighted equally with factor 1. Note that with the given problem, we could also give the pressure drop a higher weighting factor as the heat flow.

The parameters of interest for the calibration process are four calibration factors for the corresponding correlation equations:

1. `CF_RefHT`, for heat transfer correlation on refrigerant side.
2. `CF_AirHT`, for heat transfer correlation on air side.
3. `CF_Refdp`, for pressure loss correlation on refrigerant side.
4. `CF_Airdp`, for pressure loss correlation on air side.

A preliminary investigation showed that only the calibration factors for the refrigerant side are relevant for this particular case. This makes sense, as we are changing the pressure loss correlation only on the refrigerant side, and our variables of interest are on the refrigerant side. For both parameters, we use 1.0 as nominal value. We set the bounds to 0.1 and 5.

Calibration time interval is set from approximately 50 to 200 seconds. This is to not capture the initial transient, and to allow enough time for the system to reach a steady-state before the ramp signal steps are activated.

An overview of all the settings in the calibration app can be seen in Figure 9.

	Model Variable	Measured Variable	Weight
×	time	Result 1|time [s] Case 1	1
×	HXtobeCalibrated.summary.dp_ref	Result 1|HXtobeCalibrated.summary.dp_ref [Pa] Case 1	1
×	HXtobeCalibrated.summary.Qdot_refTotal	Result 1|HXtobeCalibrated.summary.Qdot_refTotal [W] Case 1	1

Parameter:

× CF_RefHT × CF_Refdp

	Model Parameter	Nominal value	Min	Max
×	CF_RefHT	1	0.1	5
×	CF_Refdp	1	0.1	5

CALIBRATE CANCEL

Selected range: 49.82638888888889-200

Figure 9. Heat exchanger calibration setup in the calibration app's GUI.

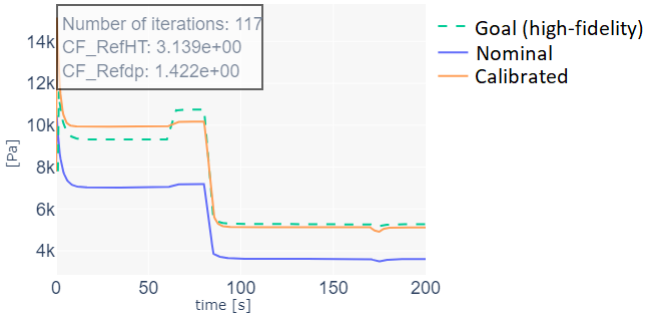

Figure 10. Pressure drop over time curves for high-fidelity model (green), and nominal (blue) and calibrated case (orange) for low-fidelity model.

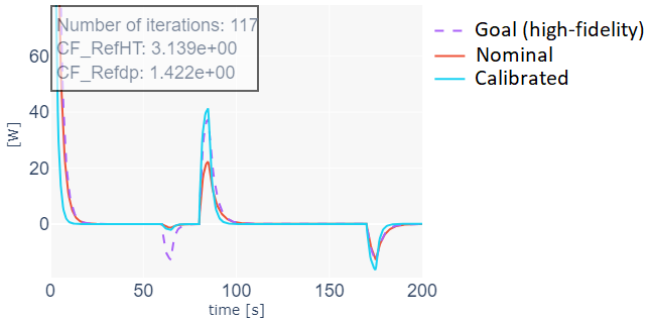

Figure 11. Heat transfer over time curves for high-fidelity model (purple), and nominal (red) and calibrated case (light blue) for low-fidelity model.

It took 117 iteration steps to find a local minimum at

$$\texttt{CF\_RefHT} = 3.139 \tag{7}$$

$$\texttt{CF\_Refdp} = 1.422 \tag{8}$$

Figure 10 and Figure 11 respectively show the pressure drop and total heat flow over time for the high-fidelity model, and the nominal and calibrated case of the low-fidelity model.

In comparison to the uncalibrated model, we can see that the result of the calibrated low-fidelity model is closer to the results of the high-fidelity model. The value of the objective function has decreased from $1.495 \cdot 10^9$ to $4.029 \cdot 10^7$.

However, it is also visible that the calibrated low-fidelity model parameterization requires more optimization. Including other system parameters can help to find a parameterization that follows the results of the high-fidelity model more closely. The underlying correlation of the low-fidelity model uses 4 extra parameters (pressure, enthalpy, mass flow, pressure drop) that can be adjusted to match the steady-state solution of the high-fidelity model. This way, the nominal solution is closer to the high-fidelity solution and potentially less iteration steps are required to get a more optimized solution.

In a further step, different start values could be picked and/or different minimizing methods (i.e. Cobyla or Powell) could be picked, to verify whether a global minimum within the bounds was found. These next steps are not discussed here.

5 Conclusion

Derivative-free minimization methods are a good way to calibrate system models, as these methods do not set strict requirements on the model architecture and find a (local) optimal solution in a reasonable number of iteration steps. To verify that an optimal solution is found with little computational effort, it is recommended to do a sensitivity analysis, run several calibration runs with different start values and use different minimization methods, such as Nelder-Mead, Powell, and Cobyla.

Using a Dash app and `scipy.optimize.minimize`, it is possible for a user with limited knowledge to cal-

ibrate Modelica models against measured or simulation data with the Nelder-Mead method. This can be used independently from the physics domain. The reliability of the results can be improved by implementing an automization and parallelization of the calibration runs with different start values and minimization methods.

References

Arendt, K. et al. (2018). *ModestPy: An Open-Source Python Tool for Parameter Estimation in Functional Mock-up Units*. Tech. rep. Modelica Association. URL: https://github.com/sdu-cfei/modest-py.

EstimationPy (2023). URL: https : / / github . com / lbl - srg / EstimationPy (visited on 2023-08-14).

Friedel, L. (1979). "Improved Friction Pressure Drop Correlation for Horizontal and Vertical Two-Phase Pipe Flow". In: *Proc. of European Two-Phase Flow Group Meet., Ispra, Italy, 1979*. URL: https://cir.nii.ac.jp/crid/1570572701507538176.

Köppen, Arne et al. (2022-11). *Techno-economic assessment of an industrial project towards carbon neutrality*. DOI: 10.3384/ecp19325.

Larson, Jeffrey, Matt Menickelly, and Stefan M. Wild (2019). *Derivative-free optimization methods*.

Nelder, J. A. and R. Mead (1965-01). "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4, pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

Olsson Hans, Jonas Eborn, Sven Erik Mattsson, and Hilding Elmqvist (2006-09). "Calibration of Static Models using Dymola". In: *Modelica 2006*. The Modelica Association, pp. 615–620.

Powell, M. J. D. (1964-01). "An efficient method for finding the minimum of a function of several variables without calculating derivatives". In: *The Computer Journal* 7.2, pp. 155–162. DOI: 10.1093/comjnl/7.2.155.

Powell, M. J. D. (1994). "A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation". In: *Advances in Optimization and Numerical Analysis*. Ed. by Susana Gomez and Jean-Pierre Hennart. Dordrecht: Springer Netherlands, pp. 51–67. DOI: 10.1007/978-94-015-8330-5_4.

Press, William H. et al. (1992). *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. USA: Cambridge University Press. ISBN: 0521431085.

Wüllhorst, Fabian et al. (2022). "AixCaliBuHA: Automated calibration of building and HVAC systems". In: *Journal of Open Source Software* 7.72, p. 3861. DOI: 10.21105/joss.03861. URL: https://doi.org/10.21105/joss.03861.