LUND UNIVERSITY

Grey-Box Building Models for Model Order Reduction and Control

De Coninck, Roel; Magnusson, Fredrik; Åkesson, Johan; Helsen, Lieve

Published in:
Proceedings of the 10th International Modelica Conference

2014

Link to publication

Citation for published version (APA):
De Coninck, R., Magnusson, F., Åkesson, J., & Helsen, L. (2014). Grey-Box Building Models for Model Order Reduction and Control. In H. Tummescheit, & K.-E. Årzén (Eds.), *Proceedings of the 10th International Modelica Conference* (pp. 657-666). Linköping University Electronic Press.

Total number of authors:
4

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Grey-Box Building Models for Model Order Reduction and Control

Roel De Coninck[a,b], Fredrik Magnusson[c], Johan Åkesson[c,d], Lieve Helsen[b]
[a]3E nv., 1000 Brussels, Belgium,
[b]KU Leuven, Department of Mechanical Engineering, 3001 Heverlee, Belgium,
[c]Department of Automatic Control, Lund University, SE-221 00 Lund, Sweden,
[d]Modelon AB, Ideon Science Park, SE-223 70 Lund, Sweden

Abstract

As automatic sensing and Information and Communication Technology (ICT) get cheaper, building monitoring data is easier to obtain. The abundance of data leads to new opportunities in the context of energy efficiency in buildings. This paper describes ongoing developments and first results of data-driven grey-box modelling for buildings. A Python toolbox is developed based on a Modelica library with thermal building and Heating, Ventilation and Air-Conditioning (HVAC) models and the optimisation framework in JModelica.org. The tool chain facilitates and automates the different steps in the system identification procedure, like data handling, model selection, parameter estimation and validation. The results of a system identification and parameter estimation for a single-family dwelling are presented.

Keywords: buildings, grey-box models, parameter estimation, collocation method

1 Introduction

The continuous progress in ICT has lead to the availability of small and low-cost sensors, low power wireless data transfer protocols, cheap and accessible data storage and powerful servers. Applied to the building sector, these technologies can be used to collect massive amounts of building monitoring data at relatively low costs. The abundance of data gives rise to new opportunities and applications in existing buildings like fault detection, energy efficiency analysis and model-based building operation. A first step in many of these applications is the creation of a building energy model.

Building models can be classified in three categories: white-box, grey-box and black-box models [1–3]. White-box modelling bases the model solely on prior physical knowledge of the building. Most building simulation software falls under this category, like TRNSYS, EnergyPlus and many others [4]. Black-box modelling bases the model solely on response data (monitoring of the building) and a universal model set, including e.g. AR and ARMAX. No physical insight is required for making a black-box model. Grey-box identification methods and tools cater for the situation where prior knowledge of the object is not comprehensive enough for satisfactory white-box modelling and, in addition, purely empirical black-box methods do not suffice because the involved physical processes are too complex.

The difference between white- and grey-box modelling is not in the complexity of the model. A single-state model can be a white-box model if all parameters can be fixed based on physical knowledge only. However, when one or more parameters in a white-box model are estimated based on a fitting of the model to measurement data, the model becomes grey, no matter its complexity. Therefore, the distinction between white and grey cannot be made by only looking at the model structure; one has to know how the model parameters have been identified.

All three model types can be either deterministic or stochastic. A deterministic model cannot explain the differences between the model output and the true variations of the states (observations). Madsen and Holst [2] therefore introduced a Wiener process in the system equations to cope with the simplifications of the model and uncertainties in inputs and monitoring. The obtained model is a stochastic state-space model.

For existing buildings with available monitoring data, the grey-box approach is considered to combine the best of two worlds: physical insight and model structure from the white-box paradigm and parameter estimation and statistical framework from the black-box paradigm. This paper describes an approach to

grey-box modelling for buildings and the development of a toolbox combining Modelica and Python. The resulting framework will be referred to as *the toolbox* in the remainder of this paper.

The aim of the toolbox is to identify low-order models from (limited) building monitoring datasets. When the dataset is generated by a detailed building simulation model instead of an existing building, we speak of model order reduction. The obtained models can be used in order to set up Model Predictive Control (MPC) or to scale up simulations from single buildings to neighbourhoods and districts.

This paper describes the methodology of the toolbox and presents some results of the application to a model order reduction of a single-family dwelling.

2 Methodology

2.1 Overview

A high-level overview of the toolbox is shown in Figure 1. The toolbox is composed of four major components:

1. Modelica library *FastBuildings* with thermal zone models, HVAC components and building models;
2. different *.mop files* specifying the model components and which parameters to estimate;
3. *JModelica.org* as a middle layer for compilation of the *.mop files* as well as formulation and solution of the optimisation problem;
4. Python module *greybox.py* delivering the user interface and top-level functionality.

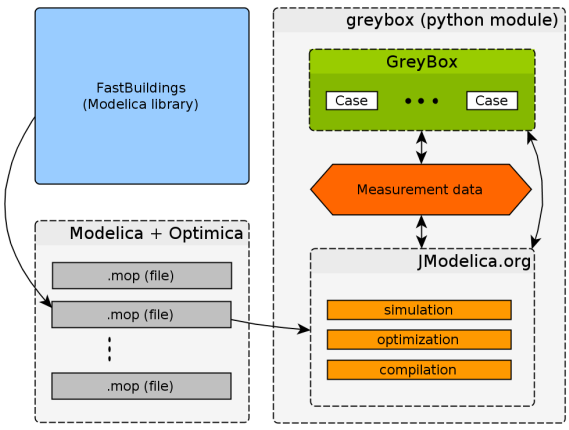

Figure 1: Overview of the grey-box buildings toolbox

2.2 Modelica

Modelica is gaining importance in the building simulation community [5, 6]. The choice of Modelica for the construction of the models is based on two major arguments. First, Modelica allows for linear, non-linear and hybrid model formulations and therefore it does not limit the model structure as such. Second, Modelica is equation-based, thus allowing efficient Newton-type solvers to be used as an alternative to for example genetic algorithms. Moreover, as shown in Section 3, the interfaces of the low-order models are identical to the detailed building model used in the *IDEAS* library [7], enabling easy model exchange.

2.3 Models

Every model structure for which the parameters have to be estimated is characterized by a different *.mop* file, of which the format is very similar to an ordinary Modelica (*.mo*) file. The *.mop* file extension is specified by JModelica.org, which is described in Section 2.4. Each *.mop* file has the same structure and has to define two models: one model for simulation, called `Sim`, and one for parameter estimation, called `Parest`. By default, the models are based on the *FastBuildings* Modelica library, which has been developed in conjunction with this toolbox. However, this is not required for the toolbox to work, as long as some naming conventions are followed. The *FastBuildings* library is introduced in Section 3. Any parameter present in the model can be estimated, including initial values of the states.

2.4 The JModelica.org platform

The toolbox relies heavily on the JModelica.org [8] platform, which is an open-source tool for modelling, simulation and optimisation of dynamic systems described by Modelica code. For simulation purposes, JModelica.org relies on the Functional Mockup Interface [9]. For optimisation purposes, JModelica.org offers various algorithms and also supports the Modelica language extension Optimica. Optimica allows for high-level formulation of dynamic optimisation problems of the type presented in Section 2.5. The file format *.mop* is used for Optimica code.

The optimization algorithm used by the toolbox to estimate the parameters is collocation-based and is presented in Section 2.6 and described in more detail in [10], where in particular optimal control is also treated. IPOPT [11], built with the MA27 solver of HSL [12], is used to solve the non-linear program (NLP) that arises from the collocation method.

Since IPOPT uses a gradient-based method, first- and second-order derivatives of all the expressions in the NLP with respect to all of the decision variables are needed. To this end, CasADi [13] is used to construct the NLP and then to compute the needed derivatives by algorithmic differentiation.

2.5 Problem formulation

Identification of the unknown model parameters is formulated as a dynamic optimization problem of the general form

$$\begin{aligned}
&\text{minimize} && \int_{t_0}^{t_f} e(t)^T \cdot Q \cdot e(t)\,\mathrm{d}t, && \text{(1a)}\\
&\text{with respect to} && \dot{x}, x, w, u, p, &\\
&\text{subject to} && F(t, \dot{x}(t), x(t), w(t), u(t), p) = 0, && \text{(1b)}\\
& && x(t_0) = x_0, && \text{(1c)}\\
& && \forall t \in [t_0, t_f]. &
\end{aligned}$$

The system dynamics are modelled by a single, possibly implicit, non-linear and time-variant, differential-algebraic equation (DAE) system of at most index one. That is, an equation system of the form (1b), where t is the time, x is the state, w is the vector-valued algebraic variable, u is the vector-valued system input, which includes both control variables and known disturbances, and p is the vector of parameters to be estimated. Since a gradient-based method will be applied to solve the dynamic optimization problem, F needs to be twice continuously differentiable with respect to all of its arguments except the first one (time). This disables the use of hybrid models. Initial conditions are given by specifying the initial state, that is, on the form of (1c), where t_0 is the start time. The initial state is usually unknown, in which case some, or all, elements of x_0 can also be introduced as elements of the vector p.

The objective (1a) of the optimisation is to minimise the integrated quadratic deviation e of the model output from the corresponding measurement data. The model output y is typically some of the states, but could also be some of the algebraic variables (and also inputs, as discussed below). The matrix Q, which typically is diagonal, is used to weight the different outputs. The measurement data is assumed to be a function of time, denoted by y_m. Since measurements are typically discrete in time, they are simply interpolated linearly to form y_m. The output deviation e is then given by

$$e(t) := y(t) - y_m(t). \tag{2}$$

The inputs can be treated in two different ways. The first is to assume that the inputs are known exactly by their measurement data and eliminate them from the problem. The second way is to have an error-in-variables approach where the inputs are kept as decision variables and treat them as model output, that is, include them in the vector y and penalize their deviation from the corresponding measurement data. The second way is useful for coping with uncertainties in the measurement data.

2.6 Solution algorithm

The approach taken to solve (1) is based on low-order direct collocation, see [14]. The idea is to divide the time horizon into a number of elements, n_e, and approximate the time-variant system variables $\dot{x}, x, w$ and u by a polynomial of time within each element, called a collocation polynomial. These polynomials are determined by enforcing the dynamic constraints at a certain number of points, n_c, within each element. These points are called collocation points and $t_{i,k}$ is used to denote collocation point number $k \in [1..n_c]$, where $[1..n_c]$ denotes the integer interval between 1 and n_c, in element number $i \in [1..n_e]$. The system variables' values at these points, denoted by

$$\begin{aligned}
&(\dot{x}_{i,k}, x_{i,k}, w_{i,k}, u_{i,k}, e_{i,k}) :=\\
&(\dot{x}(t_{i,k}), x(t_{i,k}), w(t_{i,k}), u(t_{i,k}), e(t_{i,k})),
\end{aligned}$$

are then interpolated based on Lagrange interpolation polynomials to form the collocation polynomials. There are different schemes for choosing the placement of collocation points with different numerical properties. In this paper we only consider Radau collocation. All collocation methods correspond to special cases of implicit Runge-Kutta methods and thus inherhit desirable stability properties making them suitable for stiff systems.

This approximation reduces the Problem (1), which is of infinite dimension, into a finite-dimensional non-

linear program of the form

$$\text{min.} \quad \sum_{i=1}^{n_e} \left(h_i \cdot \sum_{k=1}^{n_c} \omega_k \cdot e_{i,k}^T \cdot Q \cdot e_{i,k} \right), \tag{3a}$$

$$\text{w.r.t.} \quad \dot{x}_{i,k}, x_{i,l}, w_{i,k}, u_{i,k}, p,$$

$$\text{s.t.} \quad F(t_{i,k}, \dot{x}_{i,k}, x_{i,k}, w_{i,k}, u_{i,k}, p) = 0, \tag{3b}$$

$$x_{1,0} = x_0, \tag{3c}$$

$$x_{n,n_c} = x_{n+1,0}, \quad \forall n \in [1..n_e - 1], \tag{3d}$$

$$\dot{x}_{i,k} = \frac{1}{h_i} \sum_{l=0}^{n_c} \alpha_{l,k} \cdot x_{i,l}, \tag{3e}$$

$$\forall (i,k,l) \in ([1..n_e] \times [1..n_c] \times [0..n_c]).$$

The NLP objective (3a) is an approximation of the original objective (1a) based on a Gaussian-like quadrature formula, where the measurement error $e_{i,k}$ in each collocation point is summed and weighted by the corresponding element length h_i, which is fixed a priori, and the quadrature weight ω_k, which depends on the choice of collocation points. Notice that the decision variables are not only the unknown parameters p, but also the discretized system variables $\dot{x}_{i,k}, x_{i,l}, w_{i,k}$ and $u_{i,k}$.

Since the states need to be continuous (but not differentiable) with respect to time, the new continuity constraint (3d) needs to be introduced. Because we use Radau collocation, where no collocation point exists at the start of each element, this also requires the introduction of the new variables $x_{i,0}$, which represent the value of the state at the start of element i. With the introduction of $x_{1,0}$, the initial condition (1c) is straightforward to transcribe into (3c). The dynamic constraint (1b) is also straightforward to transcribe into (3b), by only enforcing it at the collocation points instead of during the entire time horizon.

Finally, we introduce the constraints (3e) to capture the dependency between x and $\dot{x}$. The state derivative $\dot{x}_{i,k}$ in a collocation point is approximated by a finite difference of the collocation point values of the state in that element. The finite difference weights $\alpha_{l,k}$ are related to the butcher tableau of the Runge-Kutta method that corresponds to the collocation method.

All that remains is to solve the NLP (3) in order to obtain an approximate solution to the original Problem (1). This can be done using dedicated NLP software; in our case IPOPT.

2.7 Toolbox functionality and workflow

The user interacts with the toolbox through the *greybox.py* Python module. This module defines two classes `GreyBox` and `Case`, as shown in Figure 1. The idea is to instantiate the `GreyBox` class once for the system identification of a given building. The `GreyBox` object will contain many different instances of the `Case` class. Every `Case` is an attempt (successful or not) to obtain a model for the given building. The `Case` therefore keeps track of the model structure, identification data, initial guess, solver settings and results of a single attempt. The functionality of the toolbox is packed in methods of the `GreyBox` class and can be grouped into different domains, according to the foreseen workflow. This is shown in Figure 2. This workflow is discussed in the following paragraphs.

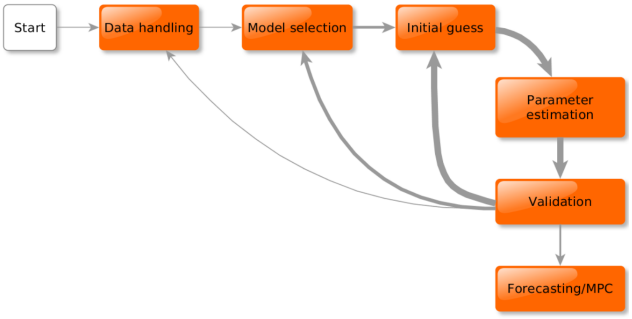

Figure 2: Workflow and high-level functionality in *greybox.py*

The methods under **data handling** are used to load the data files, resample the data if desired, create data slices of given lengths (for example one week, but can be any period) and show a plot of any data slice. Typically, one data slice is the training set, and the other slices can be used for cross-validation.

When the data has been pre-processed, a model structure has to be specified in the **model selection** step. This is accomplished by specifying the path to a *.mop* file. There are two models in the *.mop* file: a Modelica model for simulation, called `Sim`, and a Modelica + Optimica model for parameter estimation, called `Parest`. The main difference is that in the model `Parest`, the value of the Optimica attribute `free` is set to `true` for each parameter to be estimated. The compilation of both models happens automatically by invoking the corresponding JModelica.org functionality. This includes extracting information from the model (state vector, parameter vector and required inputs) and getting solver options.

Before the parameter estimation can be started, an **initial guess** has to be specified for each element in the parameter vector. These can be obtained by default, by inheritance, by Latin hypercube sampling or manually.

When the default initial guesses are used, an appropriate value is chosen for each parameter, based on its

name. For example, the naming convention in *FastBuildings* forces all parameter names for thermal resistances to start with 'r' (like `rWal`), for thermal capacities with 'c' (like `cZon`), for fractions with 'fra' (like `fraRad`), etc. Based on the first letter(s) of a parameter to be estimated, a default initial value can be set.

An alternative for obtaining the initial guess is to start from the optimized parameter vector from a previous case. This is especially useful when a new *.mop* file is selected that has similarities with a previously processed *.mop* file. Due to the naming conventions in *FastBuildings*, the corresponding parameters will have the same name. Therefore, the best initial guess for the parameters in the new model will be the optimal value from the previous estimation. For new parameters, the default initial guess method described above is used.

The last automated option to obtain initial guesses is based on Latin hypercube sampling. Due to the non-convexity of the problem, there can potentially exist many local minima. To investigate the parameter search space more systematically and increase the chances of finding a global minimum, a Latin hypercube sampling method has been implemented. This method will take a single initial guess as well as lower and upper bounds for each parameter and derive a univariate beta distribution from these three values. The distribution can be symmetric or asymmetric, as shown in Figure 3.

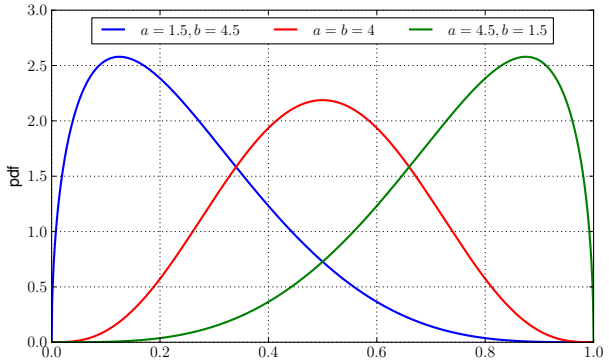

Figure 3: Symmetric and asymmetric beta distributions

The Latin hypercube sampling will then derive n stratified samples from each distribution and combine them randomly to obtain n different initital guesses. Each of these guesses will be copied to a new `case` to keep track of the results.

When a `case` has an initial guess for the parameter vector, the **parameter estimation** can be started. However, the NLP described in Equation (3) requires good initial guesses for each of the decision variables in the collocation problem (including all collocated states and algebraic variables). This is handled by doing a simulation first with the `Sim` model and the initial guess of the parameter vector. The resulting simulation trajectories are used as initial guesses for the decision variables in Problem (3). Numerical scaling factors for each system variable are also computed as the infinity norm of the corresponding trajectory.

The solution time and the number of iterations can vary a lot depending on the initial guess and the ability of the model to represent the measurement data. IPOPT allows the specification of a maximum solution time and/or a maximum number of iterations after which it will interrupt the optimisation. Both can be set in the toolbox, but in practice, specifying a maximum solution time is more intuitive. It is also more effective when the iterations become very slow. Moreover, experience shows that long execution times often lead to solutions far away from the global optimum or even divergence.

The estimation will add the optimized parameter vector to the `case`, as well as the IPOPT solver statistics.

The **validation** of the results is always based on a post-simulation with the `Sim` model and the optimized values of the parameter vector. There are both visual and quantitative validation methods. The visual methods contain for example time series plots of the resulting trajectories and corresponding residuals, scatter plots of the residuals with monitoring data and a plot of the autocorrelation function of the residuals. This also implies a check on the weights of the matrix Q from (1a) in case the error-in-variables method is used. When a full Latin hypercube sample has been estimated, a visual check of the different local optima is implemented. This can whether the sample was large enough to suppose the global optimum to be found. The quantitative methods are based on a computation of the root-mean-square error (RMSE) for each trajectory in the vector e from Equation (2). This can be done on the training data (auto-validation) or on any other dataset (cross-validation). As the RMSE is computed based on the post-simulation, possible discretisation errors in the collocation method are disregarded in the model validation process.

A computation of the confidence interval for each of the estimated parameters is implemented. This will give an indication of the accuracy of the estimation and

the parameter's influence on the model's input-output behaviour. The standard deviation of the estimated parameters $\hat{p}$ is computed according to Englezos and Kalogerakis [15]. The standard deviation for parameter i is the square root of the diagonal element on (i,i) in the covariance matrix $\text{cov}(\hat{p})$ of the estimated parameters, which is given by

$$\text{cov}(p^*) = \hat{\sigma}^2 (J^T J)^{-1},$$

where J is the Jacobian of the trajectories with respect to the estimated parameters and $\hat{\sigma}^2$ is the estimated variance of the output deviation e.

The final step in the system identification is **model selection**. Model selection should be carried out on two levels: for a single model, and between different models. For a single model, generally a Latin hypercube sampling is executed and the resulting global optimum is taken if it is a valid solution. Valid means that:

- the parameters do not lie on the specified minimum or maximum bounds,
- the parameter values are physically reasonable,
- the confidence intervals are within reasonable bounds.

The normal procedure for an inter-model selection procedure starts by a parameter estimation on a very simple (first-order) model. The obtained parameter values are often a good indication of the order of magnitude of the parameters for the more detailed models later on. Then, different models of higher order and complexity are estimated, and only those for which the single model validation is satisfactory are retained. Among all the retained models, the best model is the one that leads to the lowest RMSE value for cross-validation. This approach avoids overfitting of the model, as will be demonstrated in Section 4. A more focused forward selection procedure as described by Bacher and Madsen [16] can also be applied.

3 FastBuildings library

The *FastBuildings* library targets low-order building modelling. The library has sub-packages for thermal zone models (including windows), HVAC, user behaviour, inputs, buildings and examples. Single and multi-zone building models can be created easily by instantiating one of the predefined templates in the `Building` sub-package and redeclaring the desired submodels, like the thermal zone, HVAC or window model. The following design principles are applied throughout the library.

- The thermal connectors are `HeatPorts` from the `Modelica.Thermal` package.
- Thermal resistors and capacitances are not used from the *Modelica Standard Library* (MSL). Simplified versions with less auxiliary variables are implemented. They have exactly the same interface and connectors for compatibility with the MSL.
- A strict naming convention is used for consistency and to enable the *greybox.py* toolbox to automate certain tasks.
- The library heavily relies on the `extends` construct in order to avoid code duplication. This is specifically useful for the thermal zone models that have increasing complexity as a function of their order.
- An `inner/outer` component `simFasBui` passes all inputs like weather data, occupancy etc. from the top most level to all sublevels.
- The models for thermal zones, HVAC and user behaviour have exactly the same interface as their equivalents in the *IDEAS* library [7]. Therefore, it is very easy to replace one or more detailed models from an *IDEAS*-based model by a low-order equivalent from the *FastBuildings* library.

Currently, the thermal zone models available in the library are based on a resistor-capacitance (RC) network analogy which is often used for the modelling of thermal processes. This is however not required; any model that specifies a relationship between the heat flows and temperatures at the interface of a thermal zone can be implemented. An example of one of the third-order models in the library is given in Figure 4.

The library is distributed with the *Modelica license 2* and can be found in the *open-IDEAS* source code repository on Github [17].

4 Results

The toolbox is applied on a case study for model order reduction on a single family dwelling. A low-order model is derived from the simulated trajectories that are obtained with a detailed model. Prior knowledge about the dwelling is not taken into account in the

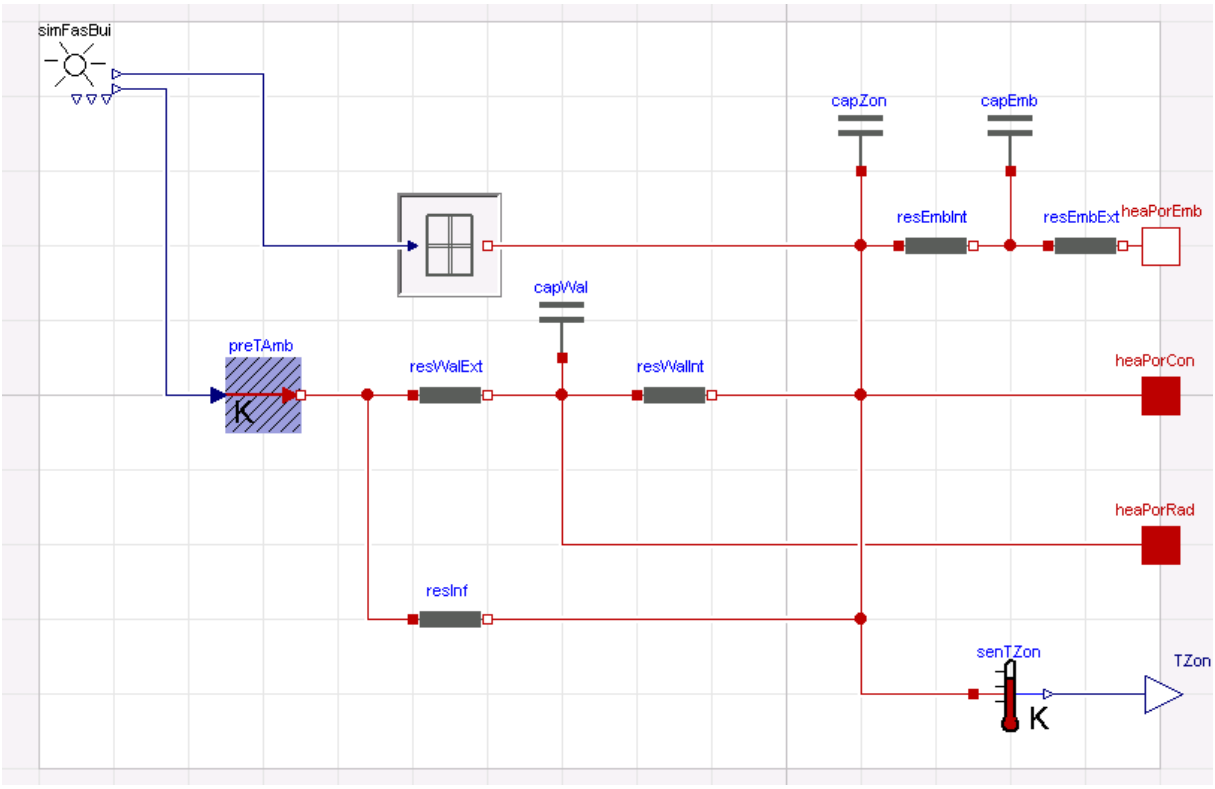

Figure 4: Example of a third-order thermal zone model in the FastBuildings library

model selection or parameter estimation, but is given here. The dwelling has two construction layers with a total heated floor surface of 196 m^2. The dwelling is designed according to a low-energy standard and has massive walls and floor heating. A total of 33 m^2 of windows with a g-value of 0.6 is integrated as follows: 9.5 m^2 on east, 10.4 m^2 on south and 11.6 m^2 on west.

Figure 5 shows the training and validation datasets. Both sets consist of one week of hourly data. Although the 'measurement' data comes from a (detailed) simulation, only a few variables that could easily be measured in a real dwelling are used for this case study. These include: the ambient temperature $TAmb$, global solar radiation on a horizontal surface $IGloHor$, electricity consumption $powEle$, thermal power of the heating system $qHeaCoo$, and zone temperature $TZon$. Hereafter the detailed simulation data is called measurement data. All of the inputs are eliminated from the problem, as discussed in Section 2.5, except for $TZon$ which is the fitting variable.

Nine models with different order, equations, and parameters have been identified using Latin hypercube sampling. After validation on single-model level, three models have been eliminated because some of their parameters were positioned on a specified minimum or maximum boundary. The RMSE values for auto and cross-validation of the six remaining models are shown in Figure 6. This figure shows that a lower RMSE on auto-validation does not necessarily imply a lower RMSE on cross-validation. The best model is the one with the lowest RMSE on cross-validation. For this case it is the model with 11 parameters, we'll call it model$_{11}$. This is the model for which the structure is shown in Figure 4. The models with 12 and 13 parameters are overfitted.

We will now analyse the results for model$_{11}$ in more detail. The sample size was 38 and a maximum CPU time of 5.5 seconds was set. The default solver settings were not changed. This implies that the number of collocation elements corresponds to the number of data points ($n_e = 168$), and each collocation element has two collocation points ($n_c = 2$). From the 38 cases, 10 converged to a solution within the limit of 5.5 seconds. Figure 7 shows the RMSE (auto-validation) compared to the IPOPT objective function for each of these 10 solutions. This plot reveals that 4 local optima have been found; the (supposed) global optimum has been found 3 times. The results also show that the discretisation in the collocation method did not lead to significant errors: there is a strong consistency between the optima found according to Equation (3) and the RMSE of the post-simulation.

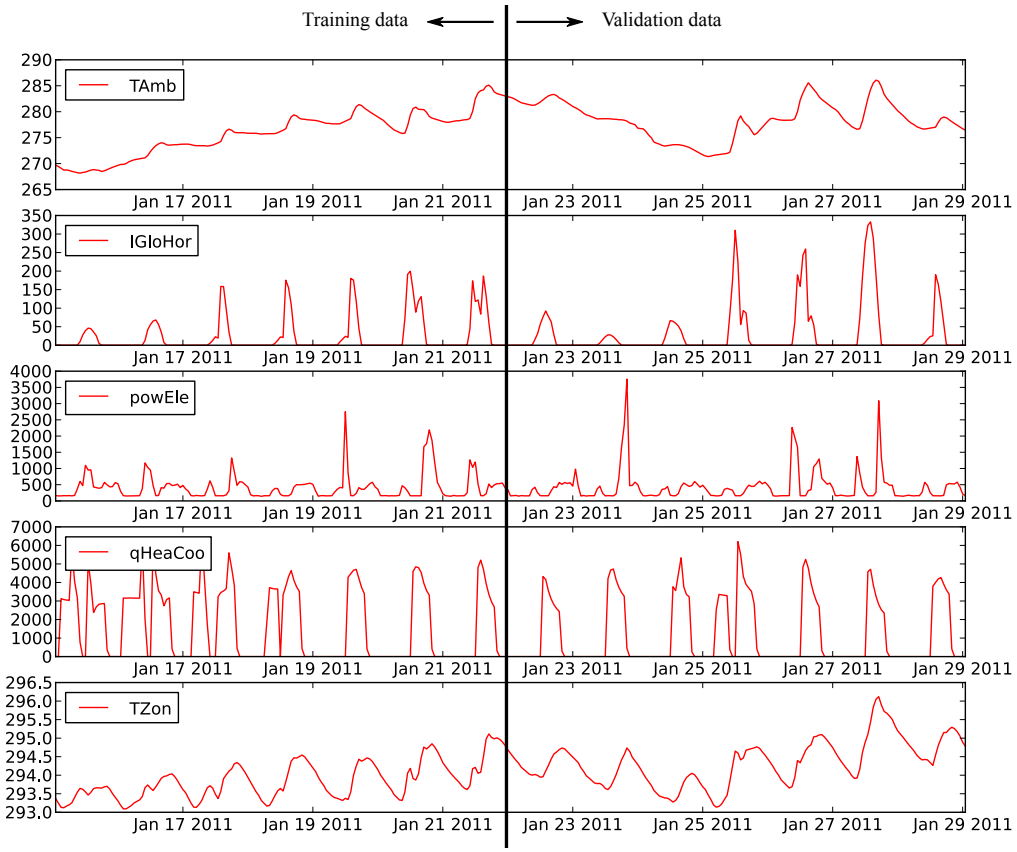

Figure 5: Measurement data for two weeks (first week as training data, second week as validation data)

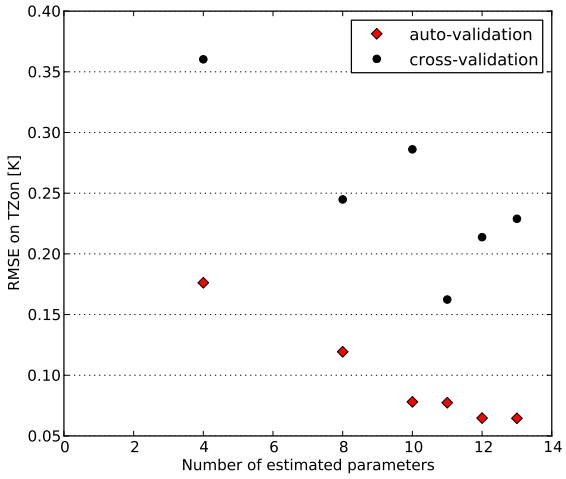

Figure 6: RMSE values for auto and cross-validation for the different models as a function of the number of estimated parameters

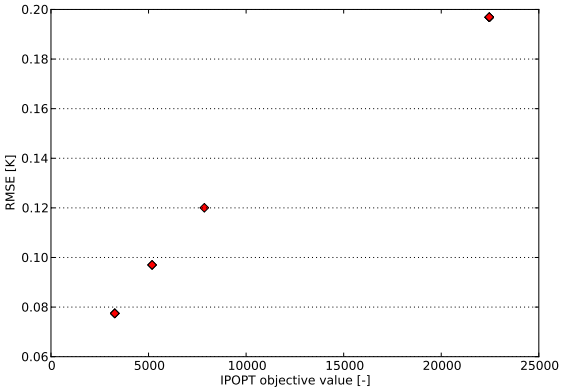

Figure 7: Relation between the IPOPT objective value and RMSE for auto-validation (obtained by post-simulation)

A time series plot of the cross-validation is shown in Figure 8. It is important to note that this plot shows an open-loop simulation over one week. The prediction power of the model is very good, with absolute deviations of the zone temperature always below 0.5 K.

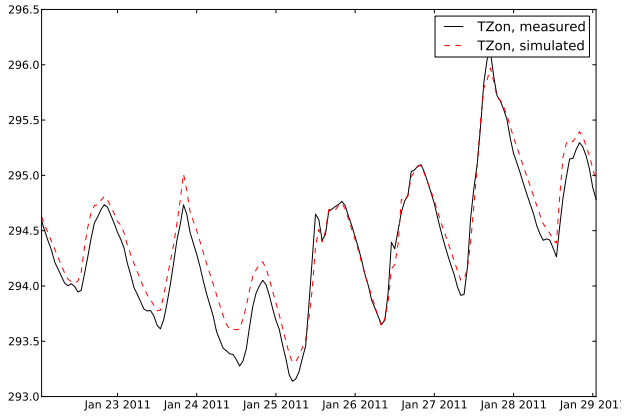

Figure 8: Comparison of the model output with the measurement data for an open loop simulation on the validation dataset

The normalized confidence intervals are shown in Figure 9. All parameters seem to have well defined confidence intervals. The most unconfident parameter values are found for the thermal resistance of the infiltration (resInf in Figure 4) and for the thermal capacity of the zone air (capZon in Figure 4).

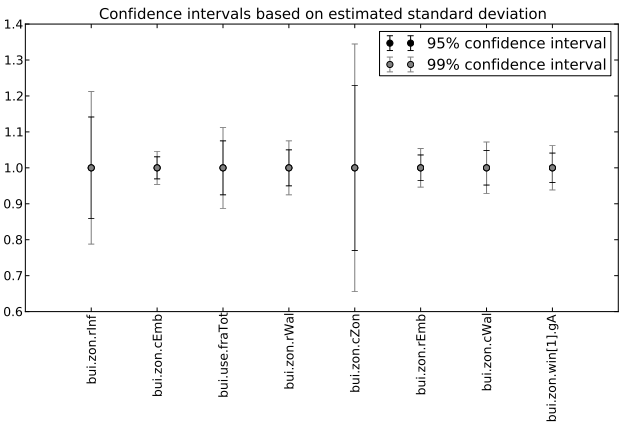

Figure 9: Normalized confidence interval for the parameters in the selected model

5 Conclusion

Inverse modelling is gaining attention in the building simulation community. More specifically grey-box modelling is considered as a strong framework for the creation of low-order models for analysis and control of monitored buildings. This paper presents an approach to obtain useful grey-box models in a largely automated way.

The first step is the creation of a building library with many potential model candidates. The Modelica package *FastBuildings* contains low-order models for thermal zones, HVAC, users, single and multi-zone buildings.

Next, a toolbox is presented that largely automates the parameter estimation of the *FastBuildings* models. It is implemented as a Python module that wraps the functionality of *JModelica.org* and presents the user a high-level interface for all common operations. The use of a gradient-based method allows an efficient numerical solution of the estimation problems. Specific attention is paid to robustness and ease-of-use. A Latin hypercube sampling of the parameter search space overcomes issues related to the non-convexity of the optimization problem.

The toolbox is applied to a model order reduction case study for a single-family dwelling. Only variables that can easily be measured in a real building are used. The selected model has 11 parameters and is able to predict the indoor temperature in an open-loop simulation (with a priori knowledge about weather and electricity consumption) with an RMSE of 0.16 K.

The real value of the toolbox can only be assessed by using the obtained models for model predictive control or for large-scale district simulations. Both applications are foreseen in future work.

Acknowledgement

Roel De Coninck wishes to acknowledge the EU ITEA2 project *Enerficiency* for supporting his work on behalf of 3E and the EU FP7 project *PerformancePlus* (contract nb. 308991) for supporting his work on behalf of KU Leuven. Fredrik Magnusson and Johan Åkesson gratefully acknowledge support from the Lund Center for Control of Complex Engineering Systems (LCCC) and the ELLIIT Excellence Center at Lund University.

References

[1] T. Bohlin, "Editorial - Special issue on grey box modelling," *International journal of adaptive control and signal processing*, vol. 9, pp. 461–464, 1995.

[2] H. Madsen and J. Holst, "Estimation of continuous-time models for the heat dynamics of a building," *Energy and Buildings*, vol. 22, pp. 67–79, 1995.

[3] N. R. Kristensen, H. Madsen, and S. B. Jorgensen, "Parameter estimation in stochastic grey-box models," *Automatica*, vol. 40, pp. 225–237, Feb. 2004.

[4] D. B. Crawley, J. W. Hand, M. Kummert, and B. T. Griffith, "Contrasting the capabilities of building energy performance simulation programs," *Building and Environment*, vol. 43, pp. 661–673, Apr. 2008.

[5] M. Wetter, "A view on future building system modeling and simulation," in *Building performance simulation for design and operation* (J. L. M. Hensen and R. Lamberts, eds.), no. i, p. 28, 2011.

[6] M. Wetter and C. Van Treeck, "IEA EBC Annex 60 - New generation computational tools for building and community energy systems based on the Modelica and Functional Mockup Interface standards." http://iea-annex60.org/about.html, 2013.

[7] R. Baetens, R. De Coninck, J. Van Roy, B. Verbruggen, J. Driesen, L. Helsen, and D. Saelens, "Assessing electrical bottlenecks at feeder level for residential net zero-energy buildings by integrated system simulation," *Applied Energy*, no. (Special issue on Smart Grids, Renewable Energy Integration, and Climate Change Mitigation - Future Electric Energy Systems), 2012.

[8] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit, "Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problems," *Computers and Chemical Engineering*, vol. 34, pp. 1737–1749, Nov. 2010.

[9] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, *et al.*, "The functional mockup interface for tool independent exchange of simulation models," in *Modelica'2011 Conference, March*, pp. 20–22, 2011.

[10] F. Magnusson and J. Åkesson, "Collocation methods for optimization in a Modelica environment," in *9th International Modelica Conference*, (Munich, Germany), Sept. 2012.

[11] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[12] HSL, "A collection of Fortran codes for large scale scientific computation." http://www.hsl.rl.ac.uk, 2013.

[13] J. Andersson, J. Åkesson, and M. Diehl, "CasADi – A symbolic package for automatic differentiation and optimal control," in *Recent Advances in Algorithmic Differentiation* (S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, eds.), Lecture Notes in Computational Science and Engineering, (Berlin), Springer, 2012.

[14] L. T. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. MOS-SIAM Series on Optimization, Mathematical Optimization Society and the Society for Industrial and Applied Mathematics, 2010.

[15] P. Englezos and N. Kalogerakis, *Applied Parameter Estimation for Chemical Engineers*, vol. 81 of *Chemical Industries*. CRC Press, Oct. 2000.

[16] P. Bacher and H. Madsen, "Identifying suitable models for the heat dynamics of buildings," *Energy and Buildings*, vol. 43, pp. 1511–1522, Feb. 2011.

[17] KU Leuven and 3E, "open-IDEAS source code repository." https://github.com/open-ideas, 2014.