Parameter Estimation Methods for Fault Diagnosis using Modelica and FMI

Ahmad Alsaab[1] Morgan Cameron[2] Colin Hough[1] Purna Musunuru[3]

[1]ESI UK, UK, {ahmad.alsaab, colin.hough}@esi-group.com
[2]ESI Group, France, morgan.cameron@esi-group.com
[3]ESI US R&D, USA, purna.musunuru@esi-group.com

Abstract

We compare a number of different methods for estimating model parameters based on external stimuli. We examine the trade-offs between the different methodologies with respect to the modelling effort necessary to implement them and the granularity of the estimation obtained. In implementing these methods, we utilize Modelica and FMI.

As an application we show how these methods can be combined with component fault modes to provide effective real-time estimates of the health of a physical asset based on thermal sensor data. In particular, we contrast the effectiveness of the different estimators in predicting the degree and location of fault.

Keywords: Parameter estimation, parameter tuning, fault diagnosis, Modelica, FMI

1 Introduction

Parameter estimation methods are applicable to scenarios that are describable by a physical or data-driven models. Such scenarios include virtual testing, virtual commissioning, fault identification and control optimization. Parameter estimation is also a vital method for tuning parameters in control systems in order to produce robust control (Astrom, 1994).

Typically, physical products are designed to meet predefined specifications. Prototypes of these products are built and tested in an effort to confirm their behavior in standardized, well-defined conditions. Today, by building a virtual prototype, (virtual) testing can be done sooner in the design cycle and quicker. In this way, virtual testing supports faster iteration of the re-engineering of the physical product.

Beyond the design cycle, it is becoming increasingly important to leverage these methods in order to make informed decisions about the condition of the product in-operation. As the digital representation of the physical asset is updated in real-time using sensor data from instrumented components it becomes possible to optimize the asset's performance and identify possible sources of failure.

Likewise, we can imagine a scenario in an instrumented manufacturing plant where application of these methods would allow flexible adjustment of operating parameters to optimize production. In the same way, faults could be detected by comparing estimated operating parameters to expected (nominal) values corresponding to normal operating conditions.

We can divide methods used for condition estimation of a physical asset into two broad categories: those that use only data, and those that utilize a physical model of the system.

Data-driven methods are applicable when there is an abundance of labelled historical sensor data.

Model-based methods, such as state estimators, are particularly applicable when there is a paucity of available labelled historical sensor data. Furthermore, as they relate the fault to a physical component in the model, they allow us to identify more easily the root cause. The location of the fault in the model can be directly associated with a component. In contrast, with a purely data-driven approach we can detect the presence of a fault, but not its location.

In this paper, we present a comparison of different parameter estimation methods and exemplify how they can be implemented using Modelica and FMI.

2 Estimation methodology

A problem that is often encountered when employing standard parameter estimation methods is that they may require a mathematical model. With models constructed using Modelica, for example, the model of the physical system is typically constructed using connected components. In this context, using standard parameter estimation methods, requires that the mathematical equations of the system are modified to directly compute parameter values. Often this is difficult, especially if the user does not have access to such a model or the strong domain background to construct one. Moreover, some components may be black boxes, in which case this kind of modification is impossible.

Using a different approach such as the particle filter method (Liu, 1998), we can extend the scope to black box-type models without requiring knowledge of the mathematical details of the system. To avoid being required to modify a Modelica model in order to estimate its parameters, for example, we can use standardly-available tools to convert the model to an

FMU and then apply these methods to perform estimations.

Other approaches each have their own disadvantages. Kalman filters, for example, require knowledge of the governing equations of the system. Furthermore, the approach is limited to linear systems (Kalman, 1960). Extended Kalman filters do not require the system to be linear, but do require knowledge of the mathematical model/matrix of the system (Julier, 2004; Cocho *et al,* 2017).

Particle filters do not have these disadvantages. They can operate with nonlinear systems without knowledge of the governing equations and can be used to estimate parameters of the system without prior knowledge.

We can distinguish between the applicability of estimators by categorizing them as either "global" or "local". We use the term "local estimator" for methods that are applied inside the model and require knowledge of the mathematical equations of the components inside the system. Furthermore, each physical component in the system for which you want to estimate its parameters requires an instance of the estimator. In contrast, we can use the term "global" estimator to refer to a method such as a particle filter that is applied outside the model and can estimate all required parameters of the system, treating the model as a black box.

A number of studies have already been undertaken to develop parameter estimation methodologies using Modelica or FMI. An optimization library Ceres (Agarwal, 2012) was used to develop a parameter-estimation framework for FMUs, exemplified with the estimation of parameters of a delta robot. By posing the problem as a non-linear least squares problem depending on the error between the measured output and estimated output of a simulated FMU. An Extended Luenberger Observer (Bortoff, 2014) was implemented for estimating heat flows and heating load using FMUs. In both these studies, the estimation method required knowledge of the Jacobian of the system, unlike some of the methods described in the present paper, which require no a priori knowledge of the Jacobian

Similarly, FMI has been used in conjunction with the unscented Kalman filter (UKF) as a nonlinear parameter estimator. Unlike the non-linear least squares optimization methods, the UKF estimator does not require knowledge of the Jacobian of the model. Nevertheless, the UKF approximates the Gaussian distribution of the state by using a set of points called sigma points, while, the particle filter algorithm can approximate any arbitrary distribution (Bonvini, 2014).

In (Videla, 2008; Brembeck *el at*, 2014; Brembeck, 2019), three different variants of the Kalman filter (extended Kalman filter, unscented Kalman filter and ensemble Kalman filter) were implemented to estimate the system parameters of a Modelica model. In all cases, the estimators are "global" in nature, utilizing a stand-alone executable version of the model was used to represent the model.

A number of open source tools have been developed for parameter estimation using FMI such as ModestPy (Arendt *et al*, 2018), RaPId (Vanfretti *et al*, 2016) and Optifmus (Bonilla et al, 2017). These tools are configurable to allow the user to select the desired estimation algorithm, either by using one of the provided algorithms, or by incorporating new, user-implemented estimation algorithms. In the present paper, we do not focus on a particular toolkit, rather we describe a set of methods appropriate for state estimation. We note that the extensible nature of those toolkits means that any of the estimation methods described here could be incorporated into those toolchains.

An extension to RaPId was developed (Bogodorova *et al*, 2017) to implement an extended particle filter method. That method takes a similar approach to the "global" particle filter algorithm we present here. However, the architecture of the RaPId toolbox, isolates the estimation algorithm from the Modelica model itself, thereby precluding the implementation of a "local" variant of the algorithm such as the one we present in this paper.

2.1 Local estimators

In the present context, we use the term "local estimator" to refer to a method that can be implemented directly inside an individual Modelica component, without requiring the use of any other software tool for implementation. In a local estimator, the estimation takes place over the course of a single simulation.

Focusing on a single element of the whole system is easy because of the componentized nature of Modelica models; we can estimate parameters for a single component without having to construct an estimator for the whole system.

In Modelica, parameters cannot change over the course of a simulation. It is therefore desirable to restructure the model such that the parameters to be estimated are replaced by (unknown) variables. In turn, these variables can be defined to have a dependence on (time-varying) inputs to the system. In this way, the estimated parameters can be driven by external stimuli (e.g. sensor data). This, in turn, though, means that to implement the estimator it is necessary to modify the mathematical description of the individual components for which we would like to estimate parameters. For this reason, this technique is more suited to individual components rather than the whole system. Moreover, because of the correlations between components induced by connections in Modelica, it is difficult to implement a single estimator for connected sets of components.

One attraction of Modelica-based approaches is that we can imagine developing libraries of self-tuning

(individual) components where parameters will adjust automatically based on data from other physical components.

2.2 Global estimators

Sometimes we may want to estimate multiple parameters (in multiple components) simultaneously, for example when estimating the degree of fault in multiple faulty components. In this case, we use the term "global estimator" to refer to any method which takes into account the whole system and does not require modifications to the governing equations. In fact, in some "global" estimation algorithms, no knowledge of the equations is even necessary.

A global estimator can run a simulation multiple times, taking as input the result of a previous iteration for initial values of state variables or parameters.

3 Tuning parameters

When a fault occurs in a physical system it can be interpreted as a change in the parameters of the model. The fault can be recognized by monitoring the difference between the output of the model and measured data. The value or the degree of fault can be determined by retuning the parameters of the system so that the new parameters give the same output response as the fault. In this work, we consider three kinds of parameter-tuning methods, specifically:

1. Direct Calculation
2. Least Square Method
3. Particle Filter Estimator

3.1 Direct calculation

In a Modelica model, the parameters define values which stay constant during the solution (simulation) of the model (Modelica Association, 2012). To solve the model, the number of equations must be equal to the number of (time-varying) variables. As an example, let us take a system of two springs and two masses (Figure 1).

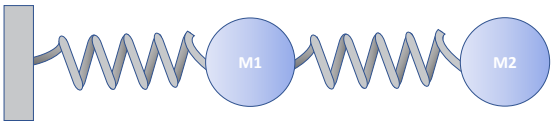


Figure 1. Two springs and masses system.

The model of the system is described by four parameters and six states or variables. The model parameters are the two masses M1 and M2 and the stiffnesses of the two springs K1, K2 respectively. The system states are the displacements of the masses x_1 and x_2, velocities $v_1$and v_2, and accelerations $a_1$and a_2. The outputs of the system are the displacement of the masses.

If the displacement of the M2 is measured, M1 can be calculated by declaring M1 as a time-varying state variable instead of a parameter and changing the declaration of x_2 to an input. In this way, the number of variables is kept equal to the number of the equations.

To test this method, a simulation was run from t=0s to t=10s, in which the value of the mass M1 was changed from 5kg to 2kg at t=5s (Figure 3). It can be seen from (Figure 2) that, as the vibration signal changes, the estimator was able to capture the change in the system parameter.

This estimator is very easy to implement, but it is noted that by construction, there needs to be a connected reference signal/external stimulus (e.g. sensor data source) per parameter to be estimated. Furthermore, it is very sensitive to noise in these signals.

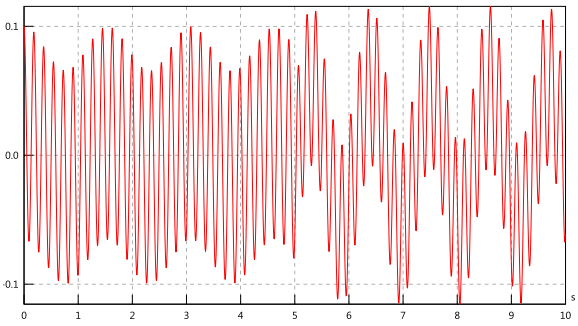
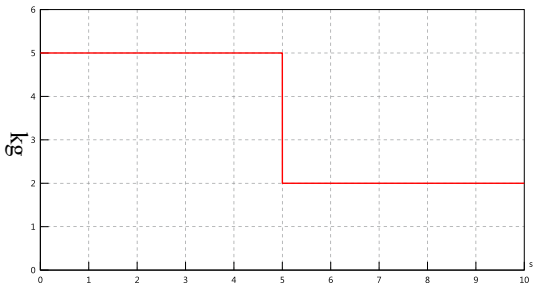


Figure 2. Displacement of M2.



Time (s)

Figure 3. Mass of M2 as a function of time (reference signal).

3.2 Recursive Least Squares Estimation

The recursive least squares method (RLS) can be used on-line and off-line to estimate parameters of static and dynamic linear systems (Watson, 1967) such as the two masses and two springs system seen above. If there is a linear system described as follows:

$$y = X^T A = a_1 x_1 + a_2 x_2 \dots\dots + a_n x_n \qquad (1)$$

where y is the output of the system, x is the state of the system and A is the parameters of the system.

The parameters of the system can be estimated in RLS using the following set of equations:

$$K_j = P_{j-1}X_j\left(1 + X_j^T P_{j-1}X_j\right)^{-1} \quad (2)$$

$$P_j = P_{j-1} - K_j X_j^T P_{j-1} \quad (3)$$

$$\hat{A}_j = \hat{A}_{j-1} - K_j\left(X_k \hat{A}_{j-1} - y_j\right) \quad (4)$$

Where j the current sample time, K is the estimator gain, P is the covariance matrix, $\hat{A}$ is the parameters of the system. The subscripts j denotes the value of that variable at that (time) iteration.

To estimate the value of the mass, when a connector class is used, the mass has two connectors *ctr1* and *ctr2* to connect with the springs. The connector *ctr1* has four variables: the force F, a flow variable between components, and the displacement x, a potential variable, as well as the velocity v and the acceleration a.

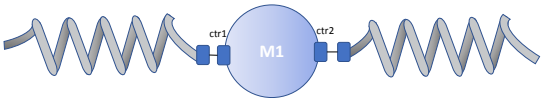


Figure 4. Mechanical connectors of mass.

$$a = \frac{d(v)}{dt} \quad (5)$$

$$v = \frac{d(x)}{dt} \quad (6)$$

$$ma = (F1 + F2) \quad (7)$$

So, to estimate the model mass of the object M1 using the RLS the relevant part of the model can be written in Modelica as:

```
ctr2.x=ctr1.x; // Displacement of ctr1 is
equal to displacement of ctr2
x=ctr1.x; // Displacement of the mass is
equal to displacement of ctr1
F=ctr1.F+ctr2.F; // Force on the mass is
equal to sum of the forces at connectors.

when sample(0, tSample) then
   P = pre(P) - pre(P)^2*ctr1.a^2(1+
pre(P)*ctr1.a^2)^-1;
   K=pre(P)*ctr1.a*(1+ pre(P)*ctr1.a^2)^-
1;
   m=pre(m)-1*K*(pre(m)*ctr1.a-F);
end when;
```

Where x is the displacement, F is the force, P is the covariance in (3), and ctr1 and ctr2 are the mechanical connectors shown in
Figure **4** and tSample is a (Real) parameter representing the sample rate at which to apply the estimation.

Figure 5 shows the estimated value of the mass, where the RLS algorithm took approximately two seconds to compute the real value of the mass.

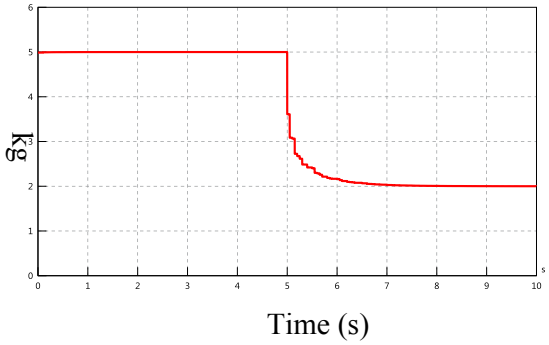


Figure 5. Estimated mass of M2 as computed by RLS algorithm.

This method is easy to implement, and is robust against sensor noise, but is only applicable to linear systems.

3.3 Particle Filter Estimation

As well as estimating the states of a system, the particle filter algorithm can also be used to estimate its parameters. Unlike the recursive least squares estimator, this algorithm is suitable for estimating the parameters of both linear and nonlinear systems (Moral *et al,* 2012; Gordon *et al,* 1993).

The particle filter uses the results of multiple concurrent simulations to produce an instantaneous estimate of the value(s) of parameters.

In this scheme, selection criteria are used that weight the uncertainty in the model versus the uncertainty in a reference signal. These selection criteria are used to weight the outputs of each of the concurrent simulations. These outputs are in turn used to run another set of simulations, updating parameter values and initial conditions accordingly.

The particle filter algorithm mainly consists of four steps namely initialization, prediction, updating and resampling. In the initialization step, the set of particles is created, where each particle includes initiation guess of the parameters. In the prediction step, for each particle, the FMU is run after assigning the parameters of the system with the particle's parameters. The updating step uses the residual, i.e. the error between the outputs of the summation and the measurement, to give an importance weight to each particle. In the last step, the resampling step, the parameters stored in each particle are redistributed depending on their weights. This algorithm is represented as a flow chart in Figure 6.

These stages were implemented in Modelica for the mass-estimation example previously described.

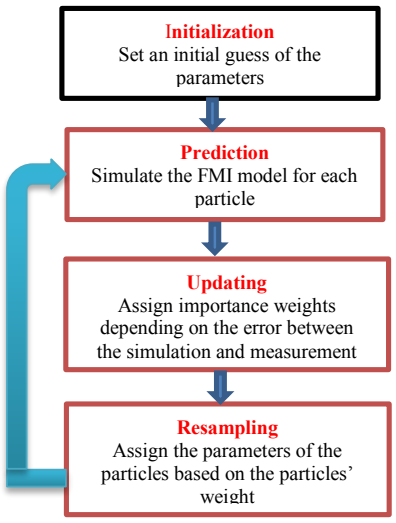


Figure 6. Particle filter algorithm.

To test this implementation, the particle filter algorithm was executed using 50 particles. The results of the estimation of the mass are shown in Figure 7. It was observed that the particle filter took less than one second to adapt the estimate to the correct value of the mass.

It is noted that the particle filter algorithm is an iterative method involving multiple simulations of the whole model. In the case of large models, or indeed large parameter sets, this could result in CPU-intensive calculations that could limit the method's applicability in real-time applications. A detailed investigation of this aspect is considered to be outside of the scope of this paper.

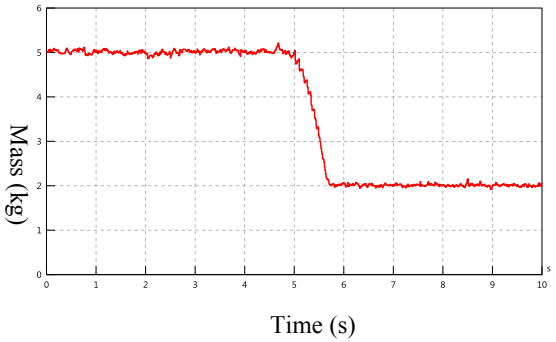


Figure 7. Estimated mass of M2 as computed by the particle filter algorithm

So far, the three tuning methods considered were implemented using Modelica by explicitly changing the mathematical model of the component and replacing it with the respective estimator algorithm.

While the Modelica implementation of the particle filter algorithm was seen to work well for simple models, we observed that the computation time increased significantly for more complex models. There is not a standardized framework defined in the Modelica specification for the parallel execution of Modelica models. This makes the parallelization of models a proprietary solution, that is tool-dependent (Modelica Association, 2012). For this reason, we developed an alternative implementation in Python that utilizes the easy conversion of a Modelica model to an FMU, to allow execution of simulations concurrently. Our implementation makes use of the PyFMI framework to manage interaction with the FMUs (Andersson *et al*, 2016). An additional advantage of utilizing FMI, is that we can apply the implementation to FMUs generated by non-Modelica tools. Furthermore, with this implementation, we realize a global estimator that can be used to estimate parameters without changing the mathematical structure of the system.

To demonstrate this approach, we consider the model of a thermo-mechanical system (Figure 8). Here, the input mechanical energy, *Pin,* is converted to heating energy according to a loss rate *U*, while the heating energy is converted to an (effective) temperature through the heating capacity of the component material. The heating energy is in turn exchanged with the cooling system *Tc* subject to a thermal resistance. The model has four parameters, namely: loss rate *U*, heating capacity *C*, thermal resistance *Rth* and cooling temperature *Tc*. In order to emulate the effect of a fault in the real system, an artificial error/fault was produced by changing the loss rate. In this example, measured mechanical power from a real mechanical system, is connected to the model via *Pin*. Readings from a temperature sensor in the real system are used as a reference signal in order to measure the validity of the estimated value.

To test the implementation of the global particle filter estimator, we converted the Modelica model in (Figure 8) to an FMI 2.0 Co-Simulation FMU using SimulationX. Table 1 show the quantities that were exported as tunable parameters in the FMU.

Table 1. Parameters of the thermal model

Parameter Name	*Value*
Thermal Capacitor (C)	$0.5\ J/K$
Loss Rate (U)	0.73
Thermal Resistance (R_{th})	$28.011\ K/W$
Cooling Temerature (T_c)	36^0

Subsequently, the particle filter was applied to tune the parameters, thus quantifying the degree of fault in the system.

Figure 9 shows that without tuning of the loss rate parameter U, there is a significant difference between measured data and the output of the model.

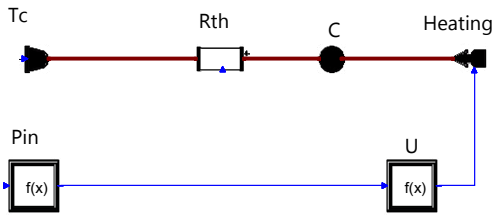


Figure 8. Thermal System. Pin is the input power, U, loss rate, C, Rth and Tc are the heating capacity, heating resistance and cooling temperature respectively.

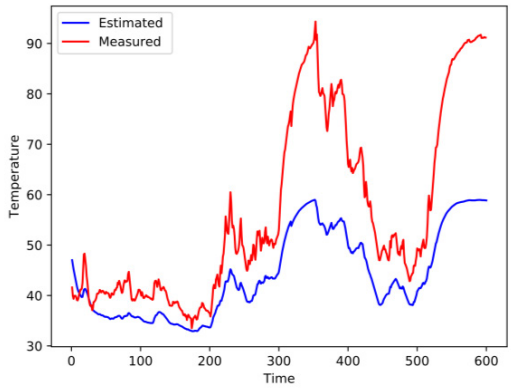


Figure 9. Estimated temperature of thermal model without tuning via particle filter (blue) compared to reference thermal signal (red)

When applying parameter tuning, the particle filter was seen to correct the estimated temperature to the measured temperature (Figure 10) by changing the value of the loss rate from 0.70% to 0.93% (Figure 11).

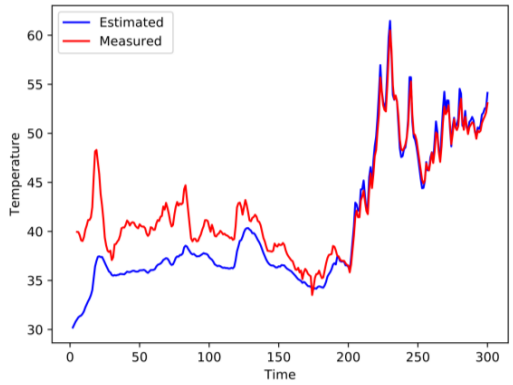


Figure 10. Estimated temperature of thermal model with tuning via particle filter (blue) compared to reference thermal signal (red)

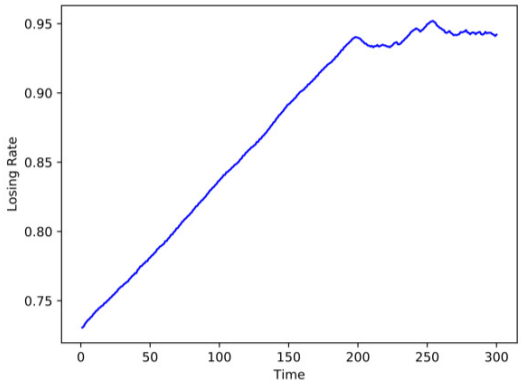


Figure 11. Tuned thermal loss factor, representing degree of fault in thermal system

4 Conclusions

In this paper we considered a number of different parameter estimation algorithms and how they could be implemented using Modelica and FMI. We examined the applicability of two different types of estimation methodology (local and global). In each case, we detailed the level of knowledge of the system necessary to implement the estimation method. We investigated the tradeoffs between the modelling effort and granularity of the estimation obtained.

We demonstrate how these methods can be applied to use external stimuli, in this case thermal sensor data, to obtain an estimate of the health of a real system.

Acknowledgements

Funding for this work, through the Innovate UK WindTwin project, is gratefully acknowledged by the authors.

References

S. Agarwal, K. Mierle, et al. Ceres solver. http:// ceres-solver.org, 2012.

C. Andersson, J. Åkesson, C. Führer. PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface, volume LUTFNA-5008-2016 of Technical Report in Mathematical Sciences. Centre for Mathematical Sciences, Lund University, 2016.

K. Arendt , M. Jradi, M. Wetter, C. Veje. ModestPy: An Open-Source Python Tool for Parameter Estimation in Functional Mock-up Units. In Proceedings of the American Modelica Conference, 2018.

K.J. Astrom, B. Wittenmark. Adaptive Control (2nd. ed.). Addison-Wesley Longman Publishing Co., Inc., USA. 1994.

T. Bogodorova, L. Vanfretti, V. S. Perić and K. Turitsyn, "Identifying Uncertainty Distributions and Confidence Regions of Power Plant Parameters," in IEEE Access, vol. 5, pp. 19213-19224, 2017.

J. Bonilla, J. A. Carballo, L. Roca, M. Berengue. Development of an open source multi-platform software tool for parameter estimation studies in FMI models. Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, 2017.

M. Bonvini, M. Wetter, M.D. Sohn. An FMI-based framework for state and parameter estimation. In: Proceedings of the 10th International Modelica Conference, Lund, Sweden, pp. 647–656. 2014.

S. A. Bortoff, C. R. Laughman. An Extended Luenberger Observer for HVAC Application using FMI. Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019.

J. Brembeck, A. Pfeiffer, M. Fleps-Dezasse, M. Otter, K. Wernersson, H. Elmqvist. Nonlinear State Estimation with an Extended FMI 2.0 Co-Simulation Interface. In Proceedings of the 10th International Modelica Conference. Lund, Sweden, pages 53–62, 2014.

J. Brembeck. A Physical Model-Based Observer Framework for Nonlinear Constrained State Estimation Applied to Battery State Estimation. Sensors (Basel). 2019.

M. G. Cocho, O. Salgado, J. Croes, B. Pluymers, W. Desmet. Model-based virtual sensors by means of Modelica and FMI, Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, pp 337-344, May 15-17, 2017.

N. J. Gordon, D. J. Salmond and A. F. M. Smith, Novel approach to nonlinear/non-Gaussian Bayesian state estimation, IEE Proceedings F - Radar and Signal Processing, vol. 140, no. 2, pp. 107-113, April 1993.doi: 10.1049/ip-f-2.1993.0015

S.J. Julier, J.K. Uhlmann. Unscented filtering and nonlinear estimation. Proceedings of the IEEE. 92 (3): 401–422, 2004. doi:10.1109/jproc.2003.823141.

R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME—Journal of Basic Engineering. 82(D): pp 35-45, 1960.

J.S. Liu, R. Chen. Sequential Monte Carlo methods for dynamic systems. Journal of the American Statistical Association. 93 (443): pp1032–1044, 1998.

P. D. Moral, A. Doucet, A. Jasra. On Adaptive Resampling Procedures for Sequential Monte Carlo Methods. Bernoulli. 18 (1): 252–278, 2012. doi:10.3150/10-bej335.

Modelica Association. Modelica – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 3.3. Modelica Association, May 2012. URL: https://www.modelica.org/documents/ModelicaSpec33.pdf

L. Vanfretti, M. Baudette, A. Amazouz, T. Bogodorova, T. Rabuzin, J. Lavenius, and F. J. Goméz-López. RaPId: A modular and extensible toolbox for parameter estimation of Modelica and FMI compliant models. SoftwareX, Volume 5, Pages 144-149, 2016.

J. I. Videla, B. Lie. “Using Modelica/Matlab for parameter estimation in a bioethanol fermentation model”. In: Proceedings of the 6th International Modelica Conference. Bielefeld, Germany, Mar. 3–4, pp. 287–299, 2008.

G. S. Watson. Linear Least Squares Regression. Ann. Math. Statist. 38 no. 6, pp 1679—1699, 1967.