New Equation-based Method for Parameter and State Estimation

Luis Corona Mesa-Moles[1] Erik Henningsson[2] Daniel Bouskela[1]
Audrey Jardin[1] Hans Olsson[2]

[1]R&D, Electricité de France, France, {luis.corona-mesa-moles,daniel.bouskela, audrey.jardin}@edf.fr
[2]Dassault Systèmes, Sweden, {erik.henningsson,hans.olsson}@3ds.com

Abstract

To get reliable simulation results from a Modelica model it is important to parametrize and initialize the model using the best estimate of the state of the system. Commonly, this state estimation is done by inverse calculation on a square system of equations requiring as many known values as states to be computed. In practice this constraint is an important limitation and, in addition, this method does not provide any information on the uncertainties or confidence level associated to the estimated state.

Taking advantage of the mathematical formulation of Modelica equations, this paper presents a new method to cope with the difficulties associated to the inverse calculation method. This approach adapts and extends the framework of data assimilation to provide a fully-integrated Modelica tool, which efficiently can handle every type of state estimation problem for static models. This method has been successfully tested with simple and complex Modelica models. Finally, the Modelica implementation of this technique allows to easily extend it to further applications.

Keywords: Modelica, parameter estimation, state estimation, model, data assimilation

1 Introduction

The safe and economically efficient operation of power plants and energy systems at large such as power grids or district heating and cooling networks rely on data that are used to assess the current state of the system and make predictions on the future states of the system at different timescales ranging from seconds to years. Finding the best estimation of the system state is important to diagnose functioning problems such as energy or efficiency losses and make the right decisions for predictive maintenance such as the best time to change a pump before it breaks.

The best estimation depends on the quality and the number of measurements. Unfortunately, raw data is always subject to uncertainties, and in general the number of states to be estimated far exceeds the number of available measurements. When dealing with a behavioral model (or digital twin) of the system, the term "states" refers to the initial values of the dynamic states and the values of the parameters, which are quantities that are not constrained by the model's equations. Poor state estimation limits the predictive power of the model.

Modelica models must be initialized by giving values to all states. This is normally done by inverse calculation on a square system of equations that requires to provide as many known values as states to be computed. Because the number of states most often far exceeds the number of known variables, the user must make a choice between the states to be computed and the states to be manually initialized from assumptions. Therefore, there is no guarantee on the accuracy of the estimation, and therefore no guarantee on the accuracy of the predictions. Static or dynamic checks on the model do not provide any information on the consistency of the initial conditions as initial conditions are not constrained by the model's equations (however, unphysical initial conditions often, but not always, lead to numerical divergence at simulation time). An answer to that problem is data assimilation.

Data assimilation is a set of techniques that combines statistical data on the measurements with a priori knowledge from the expert (the so-called background) and knowledge embedded in the model (the model's equations) to provide the best estimate of the system initial state in the form of a mean value and confidence range for each estimated state. The quality and the accuracy of the estimated states will depend on the quality of the input data considered. Data assimilation uses all available information, that is, the more data that is provided by the user (mainly background and statistical data) the more accurate the result of the data assimilation will be. In case that this detailed information is not known by the user, uninformative prior or weakly informative prior (such as guess values) can be provided.

Data assimilation was initially developed for weather forecasting and proved to be essential for the accuracy of weather prediction, which is particularly difficult, the atmosphere being a chaotic system. It is therefore expected that this technique delivers good results on a larger timescale for power plants which are stable systems.

A previous work (Corona Mesa-Moles L. Argaud J.P., Jardin A., Benssy A., Dong Y, 2019) has shown that data assimilation can be used with a Modelica model to estimate the state of the secondary loop of a nuclear power plant. In this experiment, the data

assimilation algorithms were coded in Python[1] and the Modelica model used as a black box to provide numerical values to the iterations of the Python code. This method has some limitations such as lengthy calculations and the impossibility of estimating the values of boundary conditions when several operating points are considered. One reason for the time-consuming calculations is the need to compute numerical Jacobian matrices by multiple calls to the full model of the plant.

From Dymola's perspective the calibration option has already been considered. This option is primarily designed for the case where there are more measurements than states and parameters to estimate, and normally that there are time-serie(s) of measurements (Dassault Systèmes, 2021, Section 2). The implementation, which is found in the Dymola Design library, solves the non-linear least squares problem, using Levenberg–Marquardt, which uses Gauss–Newton internally (Fletcher, 1987, Sections 5.2 and 6.1).

This paper presents a solution that is better integrated with Modelica: it takes advantage of the knowledge of the mathematical form of the Modelica equations to compute analytical Jacobians and allows the coding of data assimilation algorithms directly in Modelica. It also paves the way to extracting, from the original Modelica model, the equations that are strictly necessary to perform the state estimation, thus reducing the size of the computations. These equations correspond to the observation operator that binds the measured values (the inputs of the calculation) to the states (the outputs of the calculations): after all, no equation is needed if the measured or observed variables are the same as the estimated ones (the observation operator is then the identity operator).

This paper is structured as follows. Section 2 presents the new equation-based method for parameter and state estimation. Section 3 describes how this method is implemented in Modelica and finally Section 4 presents the application of this new method on a complex Modelica model developed with the open source library ThermoSysPro.

2 The New Equation-based Method

2.1 Optimization problem

Data assimilation intends to combine different sources of information in order to estimate at best the true state of a system. The combination of different sources of information such as a physical model and observations can be understood as an optimization problem. There are different mathematical approaches to handle data assimilation problems, among which control theory (variational methods) and estimation theory (Kalman filter or optimal interpolation) can be cited. Variational methods define explicitly the optimization problem considering a cost function in which terms associated to the model and to the observations appear. Compared to other methods, the variational approach enables to handle easily non-linear models and combine different types of observations while keeping a very efficient computation time and accurate solution for the optimization problem.

The most well-known variational method in data assimilation is the so-called 3D-Var approach (see Asch M. et al., 2016 for more details). This classical approach can be extended to take into consideration the estimation of boundary conditions as well, and this is one of the main novelties presented in this article.

The objective of the method described in this article is to give the best estimate of the tuners x of a given model H considering a certain number of observations $y_{\mathrm{obs},k}$ (which can be measurements or design assumptions for example) obtained for a set of different boundary conditions p_k. In practice, considering the uncertainties related to the model P_{b}, to the boundary conditions $P_{\mathrm{b},k}$ and to the observations $P_{\mathrm{R},k}$ (all given in the form of covariance matrices) it is possible to compute the best estimate of the tuners x and of the boundary conditions p_k on the basis of an initial value x_{b} and $p_{\mathrm{b},k}$ that correspond to the *a priori* knowledge of the state (the so-called background). The difference between the tuners and the boundary conditions is that boundary conditions vary from one operating point to another while tuners remain the same for all the operating points considered.

The cost function described in the previous paragraph, and corresponding to an extended version of the 3D-Var approach, is defined as follows:

$$\min_{x,\,p} J(x,p) \ \textbf{with}\ J(x,p) = n_{\mathrm{op}} \cdot \|x - x_{\mathrm{b}}\|^2_{P_{\mathrm{b}}{}^{-1}} + \sum_{k=1}^{n_{\mathrm{op}}} \left(\|p_k - p_{\mathrm{b},k}\|^2_{P_{\mathrm{b},k}{}^{-1}} + \|y_k - y_{\mathrm{obs},k}\|^2_{P_{\mathrm{R},k}{}^{-1}} \right) \quad (1)$$

Where:

- n_{op} are the total number of operating points;
- x are the tuners to be estimated;
- x_{b} are the tuners' background (*a priori* knowledge of the tuners to be estimated);
- p_k are the boundary conditions for operating point k;
- $p_{\mathrm{b},k}$ are the boundary conditions' background (*a priori* knowledge of the boundary conditions to be estimated);

[1] The Python codes were mainly based on the LGPL free distributed tool ADAO (Salome, 2018)

DOI 10.3384/ecp21181129

- $y_k = H(x, p_k)$ are the values computed by the model for operating point k and H is the Modelica model;
- $y_{\text{obs},k}$ are the observations (corresponding to the values computed by H);
- ${P_\text{b}}^{-1}$ is the tuner background error covariance matrix;
- ${P_{\text{b},k}}^{-1}$ is the boundary condition background error covariance matrix for operating point k
- ${P_{\text{R},k}}^{-1}$ is the observation error covariance matrix for operating point k.

It can be noted that weighted Euclidean norms have been used in the definition of cost function J, i.e. for a vector v, $\|v\|^2_{A^{-1}} = v^T A^{-1} v$, where A^{-1} is a positive definite matrix.

Solving this optimization problem gives the best estimate of tuner x (which is denoted by x_a) and of boundary conditions p_k (which is denoted by $p_{\text{a},k}$), these optimal solutions are referred to as the *analysis*.

The action of the model is captured by H, relating the observed variables to the tuners and boundary conditions ($y_k = H(x, p_k)$). However, for general Modelica models y_k is not explicitly given as a function of x and p_k. Instead, y_k is implicitly given by

$$G(x, p_k, y_k, z_k) = 0 \tag{2}$$

where G is the mathematical representation of the entire static Modelica model and z_k are all of the computed variables except y_k. Thus, in general we have $n_z \gg n_y$. As discussed in Section 1, G and z_k may, for certain applications, be substantially reduced in size by extracting from G only those equations that are necessary to link x and p_k to y_k. In this article we consider models with large algebraic loops comprising most of the model, where the connections between the tuners and observed variables are traced through these loops. Thus, a substantial reduction of G is not possible, see Section 3. Nonetheless, we consider such an extraction algorithm an interesting topic for future work.

When generating simulation code, a Modelica tool will perform a causalization, this means that the implicit equation $G(x, p_k, y_k, z_k,) = 0$ is transformed into the explicit equation $y_k = H(x, p_k)$, where z_k are treated as internal variables inside H. Thus, we may continue to use the notation and formulae as presented earlier in this section.

2.2 Computation of uncertainties

The data assimilation approach gives as well the possibility to compute the uncertainties associated to the parameters and boundary conditions that are estimated. The uncertainty associated to the analysis is given in the form of an error covariance matrix, the so-called analysis-error covariance matrix: $P_\text{a}$associated to x_a and $P_{\text{a,k}}$ associated to $p_{\text{a},k}$. They play the same role with respect to the analysis as P_b and ${P_{\text{b},k}}^{-1}$ with respect to the background.

In the case of the initial approach formulation of the data assimilation problem, it can be shown that, assuming that the model operator H can be approximated by a linear operator $\boldsymbol{H}$ in a neighborhood of the analysis, the analysis-error covariance matrix P_a is equal to the Hessian $\mathcal{H}$ of the cost function at x_a (see Tarantola A., 2005 and Bousserez N. et al., 2015). In such case, the Hessian of the cost function is given by:

$$\mathcal{H} = {P_\text{b}}^{-1} + \boldsymbol{H}^T {P_\text{R}}^{-1} \boldsymbol{H} = {P_\text{a}}^{-1} \tag{3}$$

For the final formulation of the data assimilation problem presented in Section 2.1, the Hessian of the cost function with respect to x and all the p_k can be computed by differentiating the final form of the cost function twice. The first differentiation of the cost function gives its gradient:

$$\nabla_x J = 2 \cdot n_\text{op} \cdot {P_\text{b}}^{-1}(x - x_\text{b}) + 2 \cdot \sum_{k=1}^{n_\text{op}} \left({\nabla_x y_k}^T \cdot {P_{\text{R},k}}^{-1} \cdot \left(y_k - y_{\text{obs},k}\right)\right) \tag{4}$$

$$\nabla_{p_k} J = 2 \cdot n_\text{op} \cdot {P_{\text{b},k}}^{-1} \cdot \left(p_k - p_{\text{b},k}\right) + 2 \cdot {\nabla_{p_k} y_k}^T \cdot {P_{\text{R},k}}^{-1} \cdot \left(y_k - y_{\text{obs},k}\right), \forall\, k \tag{5}$$

From these expressions and in a neighborhood of the analysis (assuming that the model derivatives can be approximated by a linear model (the so-called linear tangent model), i.e. $\nabla_x y_k$ and $\nabla_{p_k} y_k$ can be considered as constant matrices) the second derivatives of the cost function can be easily computed as follows:

$$\mathcal{H}_x = 2\left(n_\text{op} \cdot {P_\text{b}}^{-1} + \sum_{k=1}^{n_\text{op}} \left({\nabla_x y_k}^T \cdot {P_{\text{R},k}}^{-1} \cdot \nabla_x y_k \right) \right) \tag{6}$$

$$\mathcal{H}_{p,k} = 2\left(n_\text{op} \cdot {P_{\text{b},k}}^{-1} + {\nabla_{p_k} y_k}^T \cdot {P_{\text{R},k}}^{-1} \cdot \nabla_{p_k} y_k \right), \forall\, k \tag{7}$$

Considering the results presented in (Tarantola A., 2005 and Bousserez N. et al., 2015), the analysis-error covariance matrices for x and all the p_k can straightforwardly be derived from the previous expressions in order to compute the analysis uncertainties: $P_\text{a} = {\mathcal{H}_x}^{-1}$ and $P_{\text{a},k} = {\mathcal{H}_{p,k}}^{-1}$.

2.3 Solving method 1: Gradient descent

The gradient descent method is a well-known method to solve optimization problems. It is an iterative process based on the computation of the cost function J. It can be implemented in many different ways, and it can be used in the context of data assimilation to solve the optimization problem mentioned above.

When boundary conditions are considered in the optimization problem, this method can be described by the following algorithm:

- Initialization : $x_0 = x_\text{b}$ and $p_{0,k} = p_{\text{b},k}, \forall\, k$
- While $\|\nabla_x J\|$ or $\|\nabla_{p_k} J\|, \forall\, k > \varepsilon$ or $n \leq n_\text{max}$, do:
  - Compute $\nabla_x J$ and $\nabla_{p_k} J, \forall\, k$
  - Descent and update of x and p_k

- $x_{n+1} = x_n - \text{step} \cdot \nabla_x J$
- $p_{n+1,k} = p_{n,k} - \text{step} \cdot \nabla_{p_k} J, \forall\, k$
- $n = n + 1$

The explicit expressions of the gradients can be found in Section 2.2.

2.4 Solving method 2: Stationary point

The optimization problem presented in Section 2.1 can be interpreted as a minimization of a cost function J under the constraints that the model equations must be satisfied. In this case, a necessary condition to find the minimum of the cost function is that its gradient is equal to zero.

This solution can be derived from the following equations:

- $\nabla_x J = 0$
- $\nabla_{p_k} J = 0, \forall\, k$

where the model equations $y_k = H(x, p_k)$ are used in the computations of the above objective function gradients. The stationary point equations are typically solved using a Newton-type solver for optimization (Fletcher, 1987, Chapter 3). However, we do not need to take further steps here as writing the optimization problem all in Modelica allows for stating equations, not just algorithms. For details see Section 3.1.2.

2.5 Solving method 3: BFGS

As a third alternative, we consider the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. It is similar to gradient descent, but modifies the search direction by a gradually improved approximation of the Hessian of J, costing no additional function or gradient evaluations. In our tests, we have used a golden-section line search to find the minimum of J in the descent direction. For details, we refer to (Fletcher, 1987, Sections 2.6 and 3.2).

3 Implementation

As discussed in Section 2.1, the data assimilation procedure involves a model H, observation data $y_{\text{obs},k}$, background data x_b, $p_{\text{b},k}$, and the uncertainties for the observation and background data P_R, P_b, $P_{\text{b},k}$.These components are used to construct an optimization problem, here the 3D-Var formulation (1).

The above model, considered for data assimilation, is here referred to as *original model*. The observation, background, and uncertainty data will be collectively referred to as *user input*.

We identify the following requirements for a user-friendly implementation.

1. The original model should be easy to use both for data assimilation and normal simulation;
2. The optimization model should be separated from the original model and use general notation as in Section 2.1;
3. The Modelica extensions should be kept to a minimum allowing for easy modification and maintenance of the original and optimization models.

We use the expression *optimization model* rather than optimization algorithm since, defining the optimization problem in Modelica, allows us to use equations-based concepts, not just algorithms. The requirements suggest a separation between the original model, the optimization model, and the user input. Pending the eventual combination of them all in symbolic and numeric treatment.

In Sections 3.1 and 3.2 we describe the prototype implementation included in Dymola 2022 and 3DEXPERIENCE 2022x. All described features are enabled by:

```
Hidden.Assimilation.Enable = true.
```

The flag and other names described below may change in future versions.

As a recurring example we consider the simple model that computes the mass flow rate of a fluid through a pipe. In Modelica, this example is described as follows:

```
model TSP_Pipe
  parameter Real K = 10;
  parameter Real delta_P = 2e5;
  parameter Real rho = 998.8404;
  Real Q;
  Real q;
  Real G[2];
equation
  G[1] = delta_P - K*rho*q*abs(q);
  G[2] = rho*q - Q;
  G = {0, 0};
end TSP_Pipe;
```

This model is equivalent to the one presented in (El Hefni B. and Bouskela D., 2017) and developed with ThermoSysPro (El Hefni B. and Bouskela D., 2019).

3.1 Definition of the data assimilation optimization problem in Modelica

3.1.1 Extended original model

To specify which variables in a Modelica model are to be considered for data assimilation, a new variable attribute is used. To this end, the Modelica built-in type `Real` has been extended with the attribute `assimilation`, which is a record of type

```
record Assimilation
  StringType iotype;
  RealType data[:];
  RealType uncertainty[:];
end Assimilation;
```

where `StringType` and `RealType` are the Modelica primitives for strings and floating-point numbers, respectively (MLSv35 (Modelica Association 2021), Section 4.8). The string `iotype` designates a `Real` variable as a tuner x (`"Tuner"`), boundary condition p_k (`"BC"`), or observed variable y_k `"Observed"`. In the elements `data` and `uncertainty`, background or

DOI 10.3384/ecp21181129

observed data and their respective uncertainties can be specified. For tuners these two elements are scalars, for boundary conditions and observed variables they are vectors of size n_{op}, the number of operating points. Note that, the elements `data` and `uncertainty` can also be modified later, see the upcoming section.

Consider again the pipe model `TSP_Pipe` introduced in Section 3. As to not modify the original model we may extend from it and specify the assimilation attributes as modifiers.

```
model TSP_Pipe_Assimilation
  extends TSP_Pipe(
    K(assimilation(
      iotype="Tuner",
      data=950.0,
      uncertainty=100000000)),
    delta_P(assimilation(
      iotype="Tuner",
      data=2.0e5,
      uncertainty=100000000)),
    rho(assimilation(
      iotype="BC",
      data={998.8, 995.0},
      uncertainty={1, 1})),
    Q(assimilation(
      iotype="Observed",
      data={447.0, 450.0},
      uncertainty={1, 1})));
end TSP_Pipe_Assimilation;
```

Two operating points are considered with relatively high confidence in the observations and boundary conditions. There are more parameters to estimate than observations.

3.1.2 Interface for the optimization model

With the original model extended to include the relevant user input, it remains to formulate the optimization problem and equations to find an optimum. To allow for general optimization models, Dymola automatically generates a library `AssimilationPrototype`, containing the wrapper model `AssimilationModel`. The interface consists of

```
input Real x[nx] "Tuners";
input Real p[nop, np] "BCs";
output Real y[nop, ny] "Observed";
output Real dydx[nop, ny, nx] "Jacobians";
output Real dydp[nop, ny, np] "Jacobians";
```

in combination with parameters for all of the measurement data, background data, and uncertainties. The dependency $y_k = H(x, p_k)$ imposed by the original model (Section 2.1) is handled by the wrapper, cf. Section 3.2.

To exemplify the usage of this interface we construct a simple and equation-based optimization model for the 3D-Var optimization problem (1), with gradients as derived in Section 2.2.

```
partial model DA_3DVar
  AssimilationPrototype.AssimilationModel
    mod(x=x, p=p);
  Real x[mod.nx];
  Real p[mod.nop, mod.np];

  Real dJdx[mod.nx] = 2*mod.nop*
    (x - mod.x_bg)*mod.P_b_inv +
    sum(2*(mod.y[k,:] - mod.y_obs[k,:])*
      mod.P_R_inv[k,:,:]*mod.dydx[k,:,:]
      for k in 1:mod.nop);

  Real dJdp[mod.nop, mod.np] =
    {2*(p[k,:] - mod.p_bg[k,:])*
    mod.P_bk_inv[k,:,:] +
    2*(mod.y[k,:] - mod.y_obs[k,:])*
    mod.P_R_inv[k,:,:]*mod.dydp[k,:,:]
    for k in 1:mod.nop};
end DA_3DVar;
```

The `AssimilationModel` component `mod` is used to access the dimensions of the data assimilation problem, the parameters containing all user input, and the co-variance matrices. The inputs x and p_k are bound to the local unknowns with the same name. The observed variable y_k together with the Jacobians $\nabla_x y_k$ and $\nabla_{p_k} y_k$ are computed in the `AssimilationModel`. Using the Jacobians, the objective gradients $\nabla_x J$ and $\nabla_{p_k} J$ can also be computed, cf. Equations (4, 5). The model is partial as there are $n_x + n_{\text{op}} \cdot n_p$ more variables than equations; an optimality condition has to be enforced.

```
model StationaryPoint
  extends DA_3DVar;
equation
  dJdx = zeros(mod.nx);
  dJdp = zeros(mod.nop, mod.np);
  annotation(experiment(StopTime=0));
end StationaryPoint;
```

Here, we reap the full benefit of equation-based modelling as we simply set the derivatives to zero to find a stationary point, cf. Section 2.4. Dymola will do the rest and employ its Newton-type nonlinear system solver as a Newton method for optimization; there is no need to write an optimization algorithm.

Finally, a special top-level annotation is provided to point to the original model to assimilate.

```
model StationaryPoint_TSP_Pipe_Assimilation
  extends StationaryPoint;
  annotation(assimilationModel =
    "TSP_Pipe_Assimilation");
end StationaryPoint_TSP_Pipe_Assimilation;
```

When this model is translated, Dymola also translates the original model `TSP_Pipe_Assimilation` and then populates the wrapper model `AssimilationModel` with the specifics of the original model. The parameters containing the user input have default values giving by the `assimilation` attributes in the original model. These parameters can also be modified from the optimization model, indeed even after translation in Dymola's Variable Browser.

Translating `StationaryPoint_TSP_Pipe_Assimilation` we get a single nonlinear system to be solved for the two tuners and the two operating points of the single

boundary condition. When performing the static simulation, the system is solved using 19 iterations and one Jacobian computation. Note that the Jacobian of `simulation.nonlinear[1]` is the Hessian of J. The results are as follows: `x[1] = K = 991.2` and `x[2] = delta_P = 199999.8`. For the boundary condition `rho`, we get for the respective operating points `p[1,1] = 998.4` and `p[2,1] = 995.4`. Note how the low uncertainty for the boundary conditions results in values close to their background data.

Other optimization models may use the time as an iteration variable. That is, let all variables be discrete, in particular the iterates x and p_k. Then, the `sample` operator can be used to perform one iteration at each sample. For example, a simple Gradient Descent algorithm can be implemented as follows.

```
when sample(1,1) then
  x = pre(x_new);
  p = pre(p_new);
end when;
x_new = x - dJdx;
p_new = p - dJdp;
```

The iterations may be stopped at convergence using the `terminate` operator. For more complex optimization models the `AssimilationModel` wrapper also offers auxiliary functions `ComputeOutput` and `ComputeGradient` to be used in algorithms. For example BFGS can use the former to update y_k and eventually J when performing line-search.

3.1.3 Robustness of the optimization model

When performing data assimilation under the 3D-Var formulation (1), an optimization algorithm can use the available background and observed data to improve robustness of the algorithm. For example, the solution is expected to be close to the background data, so this data can be used for starting guesses, nominals, and other scaling. The variables in the `AssimilationModel` wrapper are automatically assigned these attributes.

To additionally improve the robustness of the `StationaryPoint` optimization model, homotopy can be used. Again, the 3D-Var formulation lends itself to a natural choice: When the uncertainties for the observed variables y_k tends to infinity, the 3D-Var formulation breaks down to the simple assignments $x = x_\mathrm{b}$ and $p_k = p_{\mathrm{b},k}$. Similarly, the higher the uncertainties are for y_k the easier the system `simulation.nonlinear[1]` is to solve as the influence of the model is limited. Based on this we propose the following homotopy variant of `StationaryPoint`.

```
model StationaryPoint_Homotopy
  extends StationaryPoint(mod(
      y_obs_w = homotopy(mod.y_obs_w_const,
           1.0e4 * tuner_bg_w_max *
           ones(mod.nop, mod.ny))));
  constant Real tuner_bg_w_max =
    max(max(mod.x_bg_w_const),
        max(max(mod.p_bg_w_const)));
end StationaryPoint_Homotopy;
```

For the simple model, the uncertainties `y_obs_w` for y_k are set to 10^4 times the largest uncertainty of the tuners and background variables. During the homotopy process `y_obs_w` is successively decreased to attain its actual values. Here, the `_const` variables, also provided by the wrapper, are constant versions of the corresponding user input parameters.

3.2 Dymola implementation, back-end

The full optimization model consists of three parts:

- The transformed original model;
- The computation of the Jacobians $\nabla_x y_k$ and $\nabla_{p_k} y_k$;
- The optimization model.

In the following subsections we build up the optimization problem, item by item.

3.2.1 Transformation of the original model

The goal of data assimilation is to determine values for parameters (including initial values for states) in the original model. Thus, only Modelica parameters can be designated as assimilation tuners (`iotype = "Tuner"`) or boundary conditions (`iotype = "BC"`). To ease the presentation, we also enforce that all observed variables (`iotype = "Observed"`) must be computed variables (non-parameters) in the original model. That is, we disregard cases where a parameter in the original model has both measurement data and background data. The generalization to these cases merely means including such parameters both in the x (or p_k) vector and in the y_k vector.

To prepare the original model for data assimilation, Dymola automatically removes the bindings on the parameters to be assimilated. These parameters are afterwards transformed to inputs and the observed variables are transformed to outputs, matching the interface of the `AssimilationModel` wrapper described in Section 3.1.2. As the original model is assumed to be a normal simulation model, we may assume that it is determined (square). These transformations keep this determinacy.

3.2.2 Computation of the Jacobians

In the previous section, we saw how the original model readily can be transformed to fit into the structure of the `AssimilationModel` wrapper. Assuming known values for the inputs, we get a square transformed original model, here considered in its implicit form $G = 0$, cf. Section 3.1.1. After achieving a static simulation result for each operating point, it is straight-forward to formulate the adjoint equations for the Jacobians $\nabla_x y_k$ and $\nabla_{p_k} y_k$ around these solutions

$$0 = \nabla_{x_i} G + \nabla_{y_k} G \cdot \nabla_{x_i} y_k + \nabla_{z_k} G \cdot \nabla_{x_i} z_k \qquad (8)$$

DOI 10.3384/ecp21181129

$$0 = \nabla_{p_{k,i}} G + \nabla_{y_k} G \cdot \nabla_{p_{k,i}} y_k + \nabla_{z_k} G \cdot \nabla_{p_{k,i}} z_k \quad (9)$$

for $k = 1, \ldots, n_{\text{op}}$ and $i = 1, \ldots, n_{\text{x}}$ (or n_{p} respectively). These are $n_x \cdot n_{\text{op}} + n_p \cdot n_{\text{op}}$ linear systems of equations of size $n_G = n_y + n_z$. If we compute the partial derivatives of G analytically once and for all, then it is quick work to set up the Equations (8, 9) at each pair (i, k). However, note that G is the entire static model (or at least a major part of it, cf. Section 2.1). In consequence z_k and y_k are all (or most) of the computed variables. Thus, the size of each of the Equations (8, 9) quickly becomes large. Additionally, the number of introduced variables $n_{\text{op}} \cdot (n_y + n_z) \cdot (n_x + n_p)$ becomes unfeasibly large even for medium-sized models. For these reasons we chose not to use the adjoint equations for the Jacobian computations, notwithstanding recognition of several additional optimization that could have been done to Equations (8, 9).

Instead, we consider the built-in chain rule in Dymola that is used when computing analytic Jacobians for dynamic simulation and input-output-Jacobians for FMUs. Indeed, the transformed original model has already been endowed with inputs (x and p_k) and outputs (y_k) preparing it for the application of the chain rule.

The chain rule in Dymola internalizes all of the auxiliary variables z_k (compare $y_k = H(x, p_k)$) and caches all of the partial derivatives. As most equations in a typical Modelica model are simple, the partial derivatives in turn are also mostly simple. There are important exceptions to this rule. Most notably, we have assumed that the model has a large algebraic loop, cf. Section 2.1. Thus, for most partial derivatives of y_k with respect to most x or p_k, the chain rule has to run through the algebraic loop. To this end, the inverse of the system Jacobian for the loop is needed. Computing it is costly but can be done quite efficiently in Dymola by the help of tearing and caching.

Dymola's chain rule gives us even more for free: The dependencies for the partial derivatives are traced between each y_k and x or p_k (caching any new information). This means that only the equations that are actually needed for the partial derivative computations are extracted and used.

3.2.3 Efficient treatment of several operating points

The analytical expressions for the partial derivatives $\nabla_{p_k} y_k$ and $\nabla_{p_l} y_l$ for $k, \ell = 1, \ldots, n_{\text{op}}$ are identical. Therefore, the chain rule does not need to be applied to each operating point. This fact is reflected in the transformed original model, where each boundary conditions p_k and observed variable y_k only corresponds to a single input or output, respectively.

Instead, we handle the different operating points in the `AssimilationModel` wrapper. Namely, in the array `original[nop]`, where one copy of the original model is instantiated per operating point. The wrapper then takes care to broadcast the values of the input matrix `p[nop,np]` to the correct instances of the original model, cf. Section 3.1.2. Similarly, the wrapper collects the observed variables and Jacobians from the instances into the wrapper outputs.

We conclude that the Jacobian computations proposed here only extend the original model with $n_y \cdot (n_x + n_p)$ additional scalar variables, constituting $\nabla_x y_k$ and $\nabla_{p_k} y_k$. Noting that typically $n_z \gg n_y$, this is a crucial improvement over the straight-forward application of the adjoint equations (8, 9).

3.2.4 Causality of the data assimilation optimization model

Up to this point, the causality of the original model has been preserved; the transformed model is evaluated first, after which the Jacobians are computed. Optimization models using time and sample to iterate does not change this causality. With the proper choices `x` and `p` as iterates, their old values `pre(x)` and `pre(p)` are inputs at each iteration and can be sent to the transformed original model for computation of observed variables and Jacobians, eventually leading to an update of the iterates.

However, recall that our aim is to allow for the full power of equation-based modelling when writing optimization models for data assimilation. For example, the stationary point models presented in Sections 3.1.2 and 3.1.3 take advantage of this feature. The equations

```
dJdx = zeros(mod.nx);
dJdp = zeros(mod.nop, mod.np);
```

involve all computed variables of the optimization problem. In particular, the unknowns are the tuners and boundary conditions, the computed variables of the original model, the Jacobians and the gradients. The equations are those of the original model, their derivatives with respect to the tuners and boundary conditions, and the stationary point equations. We return to the `TSP_Pipe` example for an illustration

$$0 = G_1 = \Delta P - \rho K q |q|,$$
$$0 = G_2 = \rho q - Q,$$

with $x = (K, \Delta P)$ as tuners and $y = Q = Q(K, \Delta P)$ as observed variable, leaving $z = q = q(K, \Delta P)$ as an auxiliary variable (and disregarding boundary conditions). The stationary point optimization model for `TSP_Pipe` has the following incidence.

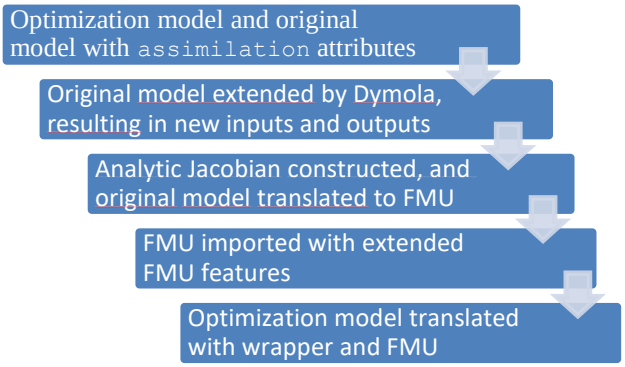
	K	ΔP	Q	q	$\frac{\partial Q}{\partial K}$	$\frac{\partial q}{\partial K}$	$\frac{\partial Q}{\partial \Delta P}$	$\frac{\partial q}{\partial \Delta P}$
G_1	*	*		*				
G_2			*	*				
$\mathrm{d}G_1/\mathrm{d}K$	*			*		*		
$\mathrm{d}G_2/\mathrm{d}K$					*	*		
$\mathrm{d}G_1/\mathrm{d}\Delta P$	*			*				*
$\mathrm{d}G_2/\mathrm{d}\Delta P$							*	*
$\mathrm{d}J/\mathrm{d}K$	*		*		*			
$\mathrm{d}J/\mathrm{d}\Delta P$		*	*				*	

As the example suggests, a partitioning can in general not be made to solve the stationary point optimization model in sequence. Rather, it must be considered at once as a (nonlinear) system of equation. The tuners and boundary conditions constitute a natural choice for iteration (tearing) variables as, when they are eliminated, the original model, the Jacobians, and the stationary point equations can be computed in sequence, where the latter are residual equations.

Commonly, and here by assumption (Section 2.1), the original model contains algebraic loops. Either these loops can be handled in the same nonlinear system as the optimization system, or they can be nested inside the optimization system. In the former case, the tuner and boundary condition iteration variables are mixed with the iteration variables for the loops in the original model. In the latter case, the loops of the original model are considered as blocks inside the optimization loop, and are solved in full each outer iteration. Compare DAE mode versus ODE mode for dynamic simulation (Braun, Casella and Bachmann, 2017; Henningsson, Olsson and Vanfretti, 2019). To keep the system of equations as small as possible we here choose the latter alternative. Additionally, it is more in line with the default choice of ODE mode for dynamic simulations in Dymola. A deeper investigation of these two alternatives may be an interesting topic for future work.

3.2.5 Synthesis

We summarize with a flowchart of the steps taken by Dymola when translating the full data assimilation problem to generate simulation code. To wrap the original model with Jacobian computations we chose to use (an extension of) FMI 2.0 for Model Exchange, which comes ready with an input-output interface and analytic Jacobian computations in Dymola.



In the last step, the causality of the optimization model is established and an (outer) nonlinear system is constructed if so needed. All the steps are done automatically by Dymola and the user does not need to be concerned about casualization questions. That is, the assimilation prototype uses the equation-based paradigm, as any other Modelica model.

In conclusion, the analytic Jacobian only need to be constructed once, which is done when translating the extended original model. This Jacobian can then be cheaply evaluated several times throughout the optimization procedure. In contrast, consider the traditional gradient-based approaches, e.g., those discussed in the Introduction, there a numeric Jacobian has to be constructed for each operating point during each optimization iteration.

4 Experimentation results on a more complex example

A complex ThermoSysPro model of the secondary loop of a pressurized water reactor is retained for this experimentation. It is the same model as the one presented in (Corona Mesa-Moles L. Argaud J.P., Jardin A., Benssy A., Dong Y, 2019). This model, presented in Figure 1, has over 12000 equations and it is used to compute the nominal operation point of the secondary loop. It is mainly composed of the following subsystems:

- A turbogenerator set made of high-pressure and low-pressure turbines and one generator;
- Two sets of moisture separator reheaters;
- One condenser;
- One feedwater tank and gas stripper system;
- Two turbine-driven feedwater pumps;
- Low and high pressure feedwater headers.

DOI 10.3384/ecp21181129

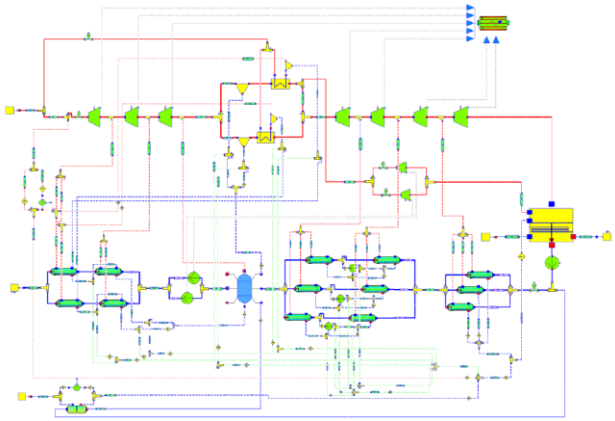


Figure 1. Model of the secondary loop of a 1300MW pressurized water reactor

In order to assess the correct implementation of the data assimilation method in Modelica, a twin experiment is considered. In such type of experiments, the observed variables used to perform data assimilation come from the simulation itself for a given state (the so-called reference state and obtained with a given value of tuners and boundary conditions) and the goal is to evaluate how close to this state the optimal state estimated through data assimilation is. The BFGS implementation is the method used for this experimentation.

In this twin experimentation, a state defined by three tuners and two boundary conditions for two different operating points is considered. This information, including the observed variables related to this state are given in Table 1.

It can be noted that in this experimentation the number of observed variables (3) is not be enough to correctly calibrate both tuners (3) and boundary conditions (2) using the usual method using square system. Data assimilation offers a general approach to deal efficiently with this type of calibration scenarios.

Table 1. Generation of the reference state

Quantity		Value	
		Op. Point 1	Op. Point 2
Tuners[2] x	1	2373.13	-
	2	2373.13	-
	3	405.762	-
Boundary conditions[3] p	1	3802.63	3820
	2	51.42	48
Observed variables[4] y	1	22753.38	223814.23
	2	22753.38	223814.23
	3	382447.47	383190.88

4.1 Scenarios considered

Two scenarios are considered to assess the correct implementation of the data assimilation technique in Modelica. Since one of the main novelties of this implementation is to take into consideration boundary conditions (BCs) in the state estimation problem, the two scenarios differ only on the *a priori* uncertainties associated to these boundary conditions.

The data assimilation inputs for Scenario 1 are given in Table 2. For observed variables and as a twin experimentation is performed, the values provided in Table 1 are kept with a low uncertainty associated (10^{-1} for each of them).

Table 2. Data assimilation inputs for Scenario 1

Quantity			Background	Uncertainty[5]
Tuners x	1		2450	10^{7}
	2		2320	10^{7}
	3		500	10^{7}
BCs p	1	Op. Point 1	3802.63	10^{-2}
		Op. Point 2	3820	10^{-2}
	2	Op. Point 1	51.42	10^{-4}
		Op. Point 2	48	10^{-4}

For Scenario 1, the uncertainties related to the tuners are much larger than the ones considered for the boundary conditions, the goal is that data assimilation will mainly adjust the values of the tuners. Therefore, it is expected for the adjusted value of the boundary conditions to be very close to the background values.

Scenario 2 is identical to Scenario 1 excepted for the uncertainty related to boundary conditions. In this scenario, an uncertainty of 10^{1} is considered for all boundary conditions. Compared to Scenario 1, it is expected to obtain different values for the optimal state and in particular for the boundary conditions.

4.2 Results

The results obtained for Scenario 1 are given in Table 3. In Table 3 the optimal values for tuners and boundary conditions within their associated uncertainties are detailed.

[2] Tuners considered in this example correspond to heat transfer coefficients of different heat exchangers.

[3] Boundary conditions considered in this example are the total thermal power extracted from the steam generators and the mass flow rate of the cooling water.

[4] Observed variables considered in this example correspond to enthalpies taken at different locations of the secondary loop.

[5] Diagonal values of the background error covariance matrices (P_{b} and $P_{\mathrm{b},k}, \forall\, k$)

Table 3. Optimal state estimation for Scenarios 1 and 2

Quantity			Analysis Scenario 1	Analysis Scenario 2
Tuners x	1		2373.1304	2386.46
	2		2373.1306	2386.46
	3		405.76154	417.57
BCs p	1	Op. Point 1	3802.6245	3828.46
		Op. Point 2	3819.9993	3846.12
	2	Op. Point 1	51.425	54.10
		Op. Point 2	47.99998	50.29

For Scenario 1, as expected due to their low background uncertainty, the values of the analysis for boundary conditions are very similar to the ones given as background (see Table 2). This is not the case for tuners for which the background uncertainty was much larger. As a consequence, the associated analysis values differ from the background ones and as expected the optimal values of the tuners correspond to the ones used to generate the reference state (see Table 2). This result shows that the data assimilation procedure is correctly implemented.

For Scenario 2, the analysis values for both tuners and boundary conditions are different from the initial background value. The higher uncertainty given to the boundary conditions is responsible for this result. For the tuners it is not surprising to find a different value from the one used to generate the reference state: this is an adjustment made as a consequence of the new analysis values for boundary conditions.

The computed uncertainties[6] for the analysis values of tuners and boundary conditions are low and they are very similar for both scenarios. For tuners 1 and 2 the uncertainty is around 10^{-4}, for tuner 3 the corresponding uncertainty is even lower, around 10^{-6}. For boundary conditions, the uncertainty is around 10^{-5} for the first one and around 10^{-7} for the second one.

With respect to the observed variables, the results are as well satisfactory. For both scenarios the observed variables corresponding to the optimal state (optimal values for tuners and boundary conditions) are extremely close to the observed variables computed from the reference state. In order to evaluate the quality of this adjustment, one can examine the evolution of the cost function (as defined in Section 2.1): from 6.60×10^8 to 0.006 for Scenario 1 and from 6.60×10^8 to 136.9 for Scenario 2. These results confirm the good implementation of the data assimilation procedure in Modelica.

In addition to the numerical results, it is interesting to point out some results with respect to the computation time. With the current non-optimized implementation of the BFGS algorithm and on a standard machine, Dymola requires 15.1 seconds per iteration in average. This time should be compared to the 26.6 seconds per iteration on average for the ADAO implementation of data assimilation. The time reduction per iteration obtained with this new equation based approach is therefore already considerable. Practically all the time required for the simulation is spent on model evaluations; reducing the time required for this task appears therefore as a good solution to reduce the time required to perform the data assimilation procedure (for example extracting only the model equations that are necessary to complete the calculations). In addition, the stopping criteria retained for the optimization algorithm may have an important impact on the total number of iterations (and therefore on the total time) required to solve the optimization problem.

5 Conclusion and perspectives

5.1 Conclusion

Based on the mathematical form of the Modelica equations, this paper presents a new method for parameter and state estimation of Modelica models. This method considers the problem of state estimation as an optimization problem and it has been adapted from the data assimilation framework.

Traditionally, this task is performed in Modelica by inverse calculation on a square system of equations which requires that the user provides as many known values as states to be computed. In practice this is an important limitation to state estimation since it is very rare to have the same number of known values as states to be estimated. The new method presented in this paper enables to efficiently handle non-square problems which are the most frequent ones.

Integrating this approach directly in a Modelica tool allows to use the analytic expressions of the Jacobians that are necessary to solve the optimization problem. The time necessary for computation can therefore be reduced compared to other methods in which numerical Jacobians have to be computed. In addition, with this approach it is possible to compute the uncertainties of the final estimated state (making it possible to specify the uncertainties related to tuners and/or boundary conditions for example). This information is an important tool for the user to evaluate the adequacy of the estimated state.

The prototype implementation in Dymola 2022 and 3DEXPERIENCE 2022x of this new method has been tested successfully with different simple and complex

[6] Diagonal values of the analysis error covariance matrices (P_a and $P_{a,k}, \forall\, k$)

DOI 10.3384/ecp21181129

Modelica models such as the model of a secondary loop of a pressurized water reactor.

5.2 Future work

As we saw in Section 3.2.2, when computing the Jacobians, we only use those equations in the original model that are actually needed. However, for the function evaluations, i.e., the computations of y_k, we still consider the full model. The data assimilation procedure therefore needs to assume that the model simulates successfully around the solution of the optimization problem. This is a reasonable assumption when the solution is close to the background data, where it, in turn, is reasonable to expect that the model is well behaved. On the other hand, data assimilation techniques may also help in the initialization of failing models. For example, by starting with a small amount of assimilation variables, constituting only a part of the model, and then successively adding more variables. Such a procedure requires that only the relevant equations are extracted also for function evaluations. Additionally, the model topology must allow it, i.e., the model must not entirely be made up by an algebraic loop. We consider this an interesting topic for future work, as the challenging problem of robust initialization is one of the major obstacles in contemporary Modelica applications.

To ease the presentation, we have only discussed optimization with no lower or upper bounds on the tuners and boundary conditions. However, the presented optimization models can easily be extended to take into account such constraints. To better support this the `assimilation` attribute may be extended in future Dymola releases to allow specification of bounds directly in the original model.

Finally, we have limited our presentation and implementation to focus on the data assimilation problem. However, as the objective function and optimization models are written all in Modelica the techniques can be extended to more general optimization problems.

Acknowledgements

This work was funded and supported by BPI France through the French FUI ModeliScale research project.

References

Asch M., M. Bocquet, M. Nodet (2016), *Data Assimilation - Methods, Algorithms and Applications*, SIAM.

Bousserez, N., D.K. Henze, A. Perkins, K.W. Bowman, M. Lee, J. Liu, F. Deng and D.B.A. Jones (2015). "Improved analysis-error covariance matrix for high-dimensional variational inversions: application to source estimation using a 3D atmospheric transport model". In: *Q.J.R. Meteorol. Soc.*, 141: 1906-1921. https://doi.org/10.1002/qj.2495

Braun W., F. Casella and F. Bachmann (2017). "Solving large-scale Modelica models: New approaches and experimental results using OpenModelica". In *Proceedings of the 12th International Modelica Conference*, pages 557–563. Linköping University Electronic Press,.

Corona Mesa-Moles L. J.P. Argaud, A. Jardin, A. Benssy, Y. Dong (2019). "Robust Calibration of Complex ThermosysPro Models using Data Assimilation Techniques: Application on the Secondary Loop of a Pressurized Water Reactor". In: *Proceedings of the 13th International Modelica Conference*, pages 553-560. Linköping University Electronic Press.

Dassault Systèmes (2021). *Dymola 2022: Dymola, Dynamic Modeling Laboratory, User Manual 2A: Model Development Tools.* Dassault Systèmes AB, Lund, Sweden.

El Hefni B. and D. Bouskela (2017). "Modeling and simulation of a complex ThermoSysPro model with OpenModelica – Dynamic Modeling of a combined power plant". In: *Proceedings of the 12th International Modelica Conference*, May 15-17, Prague, Czech Republic.

El Hefni, B. and D. Bouskela (2019). *Modeling and Simulation of ThermalPower Plants with ThermoSysPro - A Theoretical Introduction and a Practical Guide*. Springer.

Fletcher, R. (1987). *Practical methods of optimization*. 2nd edn. New York: John Wiley & Sons.

Henningsson E., H. Olsson and L. Vanfretti (2019). "DAE Solvers for Large-Scale Hybrid Models". In: *Proceedings of the 13th International Modelica Conference*, pages 491–502. Linköping University Electronic Press.

Modelica Association (2021-02). Modelica – A Unified ObjectOriented Language for Systems Modeling. Language Specification Version 3.5. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.5/MLS.html.

SALOME The Open Source Integration Platform for Numerical Simulation, accessed 2021-08-19, http://www.salome-platform.org/

Tarantola A. (2005). *Inverse Problem Theory and Methods for Model Parameter Estimation.* SIAM: Philadelphia, PA.