Formulation of dynamic optimization problems using Modelica and their efficient solution

Rüdiger Franke
ABB Corporate Research
Wallstadter Str. 59
68526 Ladenburg, Germany
E-Mail: Ruediger.Franke@de.abb.com

Abstract

Dynamic optimization problems often arise in advanced model based control. For example in model based predictive control and in the estimation of process parameters or not measured process signals, the underlying problems can be treated with optimization.

A process model formulated in Modelica [10] can be used as a core part in the formulation of dynamic optimization problems. This allows an efficient engineering of advanced control applications as simulation models are reused for optimization.

The paper discusses, how different types of dynamic optimization problems can be formulated based on a nonlinear dynamic system model. Furthermore, the efficient numerical solution of dynamic optimization problems as large-scale nonlinear programming problems is outlined. The treatment of state constraints is emphasized in this context. Possibilities for obtaining model sensitivities as required by an optimization solver are discussed.

However, the class of models that can be used for optimization in this way is limited, compared to all models that can be formulated in Modelica and used for initial-value simulation. Specific requirements by optimization solvers are discussed together with features of the Modelica language supporting their consideration in model formulations.

The optimal startup of a power plant serves as a practical example.

1 Introduction

Dynamic optimization problems occur if parameters and control inputs of a dynamic system shall be influenced so that a cost criterion is minimized subject to constraints. They are playing an increasingly important role in control engineering and in process engineering. Typical applications involving dynamic optimization are e.g. nonlinear model predictive control (NMPC), data reconciliation, and integrated design and control of technical processes.

Higher requirements on the efficiency of industrial processes, together with the availability of new modeling and solution technologies, are causing a trend towards the treatment of dynamic optimization problems for rigorous physical models. Unfortunately a substantial effort is generally needed to formulate an optimization model fulfilling both: high model accuracy and high solution efficiency.

This paper discusses the use of Modelica to formulate dynamic system models for optimization. A substantial reduction of the effort for model building is achieved by reusing available simulation models for optimization and by exploiting features of Modelica for application specific model adaptation. The solution of dynamic optimization problems applying large-scale nonlinear programming is outlined and requirements of state-of-the-art optimization solvers on the model are discussed.

2 Dynamic optimization problems

2.1 Nonlinear Dynamic System Model

Modelica allows the object oriented modeling of dynamic systems by differential and algebraic equations. The object oriented Modelica model is typically translated to a mathematical system of differential and algebraic equations prior to its treatment with numerical solvers. Here it is assumed that the result of the model translation is a system of ordinary differential equations of the form

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(t),\mathbf{u}(t),\mathbf{z}(t),\mathbf{p},t], \qquad (1)$$
$$\mathbf{f} : \mathbb{R}^{n_x} \times \mathbb{R}^{m} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \mapsto \mathbb{R}^{n_x}$$
$$\mathbf{y}(t) = \mathbf{g}[\mathbf{x}(t),\mathbf{u}(t),\mathbf{z}(t),\mathbf{p},t], \qquad (2)$$
$$\mathbf{g} : \mathbb{R}^{n_x} \times \mathbb{R}^{m} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \mapsto \mathbb{R}^{n_y}$$

Model variables are internal continuous-time states $\mathbf{x} \in \mathbb{R}^{n_x}$, control inputs $\mathbf{u} \in \mathbb{R}^{m}$, disturbance inputs $\mathbf{z} \in \mathbb{R}^{n_z}$, constant parameters $\mathbf{p} \in \mathbb{R}^{n_p}$, and model outputs $\mathbf{y} \in \mathbb{R}^{n_y}$.

The model behavior is completely determined by the system equations $\mathbf{f}$ and the output equations $\mathbf{g}$, if initial states $\mathbf{x}_0 = \mathbf{x}(t_0)$, external inputs $\mathbf{u}(t),\mathbf{z}(t),t \in [t_0,t_f]$, and parameters $\mathbf{p}$ are given. The outputs $\mathbf{y}(t),t \in [t_0,t_f]$ can then be obtained by solving the system of differential equations using initial-value simulation.

However, often some of the required information is not explicitly known, but can be obtained by minimizing a cost function. In many of those cases, a feasible solution can be further specified by constraining model variables. Optimization is a universal tool for treating those inverse problems.

2.2 Estimation Problem

An example for an inverse problem is the estimation of unknown parameters $\mathbf{p}$ and/or initial states $\mathbf{x}_0$ based on measured inputs and outputs. The estimation problem can be solved by minimizing a least squares criterion

$$\sum_{i=1}^{n_{\bar{y}}} \|\mathbf{y}(t_i) - \bar{\mathbf{y}}(t_i)\|^2 \to \min_{\mathbf{x}_0,\mathbf{p}} \qquad (3)$$

for the set of measurement data $\{\bar{\mathbf{y}}(t_i), t_i \in [t_0,t_f], i = 1,\ldots,n_{\bar{y}}\}$.

2.3 Design Parameter Optimization Problem

Some model parameters might be free or given within useful ranges, instead of with fixed values. Optimization can be used to determine values for those unknown parameters that minimize a criterion $F(\mathbf{p}) : \mathbb{R}^{n_p} \mapsto \mathbb{R}^1$

$$F(\mathbf{p}) \to \min_{\mathbf{p}} \qquad (4)$$

subject to parameter bounds $\mathbf{p}_{\min} \le \mathbf{p} \le \mathbf{p}_{\max}$ and required system outputs, e.g. $\mathbf{y}(t) \ge \mathbf{y}_{\min}(t), t \in [t_0,t_f]$.

2.4 Optimal Control Problem

The control inputs $\mathbf{u}(t),t \in [t_0,t_f]$ might be free to be chosen so that a criterion

$$F_0[t_f,\mathbf{x}(t_f)] + \int_{t_0}^{t_f} f_0[t,\mathbf{x}(t),\mathbf{u}(t)]\,dt \to \min_{\mathbf{x}_0,\mathbf{u}(t)}, \qquad (5)$$
$$F_0 : \mathbb{R} \times \mathbb{R}^{n_x} \mapsto \mathbb{R},$$
$$f_0 : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \mapsto \mathbb{R}.$$

is minimized subject to constraints on model inputs $\mathbf{u}_{\min}(t) \le \mathbf{u}(t) \le \mathbf{u}_{\max}(t)$ and outputs $\mathbf{y}_{\min}(t) \le \mathbf{y}(t) \le \mathbf{y}_{\max}(t),\ t \in [t_0,t_f]$.

2.5 Discrete-Time Optimal Control Problem

In order to use a digital computer to solve dynamic optimization problems, continuous-time functions have to be discretized. Here multistage control parameterization is applied to formulate dynamic optimization problems as discrete-time optimal control problems.

The time horizon $[t_0,t_f]$ is divided into K stages with $t_0 = t^0 < t^1 < \ldots < t^K = t_f$. The controls $\mathbf{u}(t)$ are described in each interval $[t^k,t^{k+1}], k = 0,\ldots,K-1$ as function of the discrete-time input variables $\mathbf{u}^k \in \mathbb{R}^m$. The unknown parameters $\mathbf{p}$ are converted to state variables with the state equation $\dot{\mathbf{p}} = \mathbf{0}$ and with unknown initial values $\mathbf{p}_0 = \mathbf{p}(t_0)$. They are described together with the continuous-time model states $\mathbf{x}(t)$ with the discrete-time state variables $\mathbf{x}^k \in \mathbb{R}^n, n = n_x + n_p$. The state equation (1) is solved for the stage k with the initial values $\mathbf{x}^k$ and the controls $\mathbf{u}^k$ using a numerical integration formula.

This results in the multistage optimization problem:

$$F^K(\mathbf{x}^K) + \sum_k f_0^k(\mathbf{x}^k,\mathbf{u}^k) \to \min_{\mathbf{u}^k,\mathbf{x}_0}, \qquad (6)$$
$$F^K : \mathbb{R}^n \mapsto \mathbb{R}^1,\ f_0^k : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^1$$

with respect to the discrete-time system equations

$$\mathbf{x}^{k+1} = \mathbf{f}^k(\mathbf{x}^k,\mathbf{u}^k), \qquad (7)$$
$$\mathbf{f}^k : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$$

and the additional constraints

$$\begin{aligned} \mathbf{c}^k_{\min} &\le \mathbf{c}^k(\mathbf{x}^k,\mathbf{u}^k) \le \mathbf{c}^k_{\max}, \\ \mathbf{c}^K_{\min} &\le \mathbf{c}^K(\mathbf{x}^K) \le \mathbf{c}^K_{\max}, \\ \mathbf{c}^k &: \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^{m_k}, \mathbf{c}^K : \mathbb{R}^n \mapsto \mathbb{R}^{m_K}. \end{aligned} \qquad (8)$$

Note that initial conditions of the system model are formulated as general constraints (8) as well. Discretization formulae, known parameter values, and predetermined disturbances are included into the discrete-time functions F^K, f_0^k, $\mathbf{f}^k$, $\mathbf{c}^k$, and $\mathbf{c}^K$. The discrete-time functions are assumed to be two times continuously differentiable with respect to their variables.

2.6 Large-Scale Nonlinear Programming Problem

Discrete-time optimal control problems can be treated as structured large-scale nonlinear optimization problems. This has the main advantage that recently developed methods for large-scale nonlinear optimization can be applied to their efficient solution [11, 4].

The discrete-time control and state variables for all stages k are collected to one large vector of optimization variables

$$\mathbf{v} = \begin{pmatrix} \mathbf{x}^0 \\ \mathbf{u}^0 \\ \mathbf{x}^1 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{x}^{K-1} \\ \mathbf{u}^{K-1} \\ \mathbf{x}^K \end{pmatrix}. \tag{9}$$

One specific feature of the optimization approach discussed here is that the discrete-time state variables at all stages are treated as optimization variables as well, even though they are determined by initial conditions and the control parameters. This leads to a significant increase of the size of the optimization problem. However, the consideration of states as constrained optimization variables generally improves robustness and efficiency of the solution. For instance trajectory constraints can be formulated directly on the discrete-time state variables. Furthermore the separation of the overall problem into multiple stages often leads to a reduction of the required number of nonlinear iterations. The computational overhead is relatively low if the number of state variables n_x is not too high, compared to the number of control variables n_u and and if the sparse multistage structure of the large-scale nonlinear optimization problem is exploited appropriately.

3 Solving nonlinear dynamic system models for optimization

Sequential Quadratic Programming (SQP) is generally considered as the most efficient numerical method available nowadays to solve nonlinear optimization problems [12]. This quasi Newton method treats nonlinear optimization problems by solving a sequence of local linear-quadratic approximations. The Lagrangian of the optimization problem is approximated quadratically, typically by applying a numerical update formula. Constraints are approximated linearly.

The differential equations (1) used to model a dynamic system together with the integration formulae determine the equality constraints (7) of the discrete-time optimal control problem. Accordingly the initial value problem

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(t), \mathbf{u}(\mathbf{u}^k, t), \mathbf{z}(t), \mathbf{p}(\mathbf{x}^k), t], \quad t \in [t^k, t^{k+1}], \quad \mathbf{x}(t^k) = \mathbf{I}^k(\mathbf{x}^k), \tag{10}$$

has to be solved for each stage $k = 0, \ldots, K-1$ in each nonlinear optimization iteration to evaluate the discrete-time system functions $\mathbf{f}^k(\mathbf{x}^k, \mathbf{u}^k), k = 0, \ldots, K-1$.

Furthermore the discrete-time sensitivities

$$\frac{d\mathbf{f}^k(\mathbf{x}^k, \mathbf{u}^k)}{d(\mathbf{x}^k, \mathbf{u}^k)} \tag{11}$$

are needed to obtain local linear approximations of the nonlinear system model. Often it turns out that the determination of these sensitivities is the most time consuming part when solving dynamic optimization problems.

A straightforward approach for obtaining the sensitivities is to numerically differentiate the system model together with the integration formula. This is normally done by performing multiple initial value simulations for perturbed control variables $\mathbf{u}^k$ and discrete-time states $\mathbf{x}^k$ (e.g. when using Matlab optimization routines together with a Simulink model). However, major drawbacks of this approach are low numerical efficiency and accuracy.

More robust and efficient results can be obtained when solving continuous-time sensitivity equations together with the differential model equations. In approach discussed here the continuous-time sensitivities are needed with respect to the optimization variables

$$\mathbf{q} = \begin{pmatrix} \mathbf{x}^k \\ \mathbf{u}^k \end{pmatrix}. \tag{12}$$

The required sensitivities are

$$\mathbf{s}_i(t) = \frac{d\mathbf{x}(t)}{dq_i}, \quad i = 1, \dots, n+m. \tag{13}$$

They are defined by the sensitivity equations

$$\dot{\mathbf{s}}_i(t) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}(t)} \mathbf{s}_i(t) + \frac{\partial \mathbf{f}}{\partial q_i}, \quad t \in [t^k, t^{k+1}] \tag{14}$$

with the initial conditions

$$\mathbf{s}_i(t^k) = \frac{d\mathbf{I}^k(\mathbf{x}^k)}{dq_i}. \tag{15}$$

See e.g. [9] for an extension of the famous DASSL integration algorithm with sensitivities.

The remaining task is to provide the partial derivatives of the model equations (10) as required by the sensitivity equations (14). They can be obtained with the help of algorithmic, or automatic, differentiation of the model equations [6]. Alternatively the model equations can also be differentiated numerically. This leads to a comparable simpler implementation at the cost of less accuracy and robustness. Good experiences have been made with both: application of algorithmic differentiation using ADOL-C and numerical differentiation of a model implemented as Simulink S-function. It turns out that numeric differentiation of the model equations alone gives more robust results than differentiating the model together with the integration formula numerically. This is especially true for a variable step size integration algorithm that takes different steps in subsequent runs when differentiating model equations and integration formula together numerically.

As the simulation code is generated by a model translation tool from a Modelica specification, one would wish for the future that a Modelica translator like Dymola generates required sensitivity equations together with the model equations. This would considerably simplify the treatment of dynamic optimization problems.

4 Requirements on dynamic system models used for optimization

Especially the exploitation of model sensitivities and the treatment as multistage problem are important for an efficient solution of dynamic optimization problems. However, both techniques do also imply requirements on the optimized model.

The main advantage of the exploitation of sensitivities is that the superior performance of state-of-the-art nonlinear optimization algorithms can be utilized. This is especially important for problems with a high number of unknown parameters, e.g. to describe a complex control trajectory. However, the model must be smooth with respect to the optimization variables. This means that the values of model variables or their derivatives may not jump (e.g. caused by a state event or by discontinuous functions like absolute value, respectively). From the point of view of optimization, state events have to be formulated as integer variables. This leads to mixed integer nonlinear optimization problems that require a significantly higher solution effort than smooth nonlinear optimization problems. Fortunately in many cases discrete events can be circumvented, e.g. a diode can be modeled ideally utilizing a state event or approximately with a smooth non-linear function. Furthermore it might be sufficient to formulate an optimization problem for a restricted range of the validity of the overall model by introducing constraints on optimization variables. For instance a flow model expressing flow reversal with a state event might be restricted to only exhibit flow into one direction when used in a dynamic optimization.

It is important to note that the model must not be smooth with respect to time. This means that time events, or more generally speaking a sequence of events with fixed switching structure, can easily be incorporated into the dynamic optimization problem. In fact mixed integer nonlinear optimization solvers often exploit this feature and treat a problem with state events on two levels: integer variables are modified on an upper level, while for each set of fixed integer variables the resulting nonlinear optimization problem with fixed switching structure is solved on a lower level.

Besides the exploitation of sensitivities, the treatment as multistage problem offers following advantages:

- improved treatment of state trajectory constraints, because sampled values of the state variables are optimization variables,
- non-linearities do only occur within stages involving only discrete-time variables at specific discrete time points (often leading to a reduction of non-linear iterations),

- the time consuming sensitivity analysis can be performed in parallel for all stages because the initial states for each stage are optimization variables.

The price that has to be paid for these features is that not only sensitivities with respect to the free parameters are required, but also with respect to the initial states of each stage. That is why the number of unconstrained state variables should not be too high, compared to the number of optimized control inputs or model parameters, as otherwise the expensive calculation of sensitivities for these states does not pay off. Fortunately this practical requirement of low model complexity is not specific to dynamic optimization, but is generally known from control applications. If for instance the dynamic optimization shall be performed on-line starting at a transient initial state, the availability of measurement data for estimating the initial state often restricts the allowed model complexity too.

5 Modelica features supporting the formulation of optimization models

One mathematical model can hardly fulfill all requirements that are caused by different applications. That is why it is considered important that a modeling language supports a flexible model management allowing to build different mathematical models describing the same dynamic system depending on requirements by specific applications.

5.1 Separation of model interface and model implementation

A well known object-oriented technique is to separate interface definition and implementation. This technique is also well supported by the object oriented modeling language Modelica. An interface can be defined as partial model:

```
partial model ShellModel
  // interface definitions
end ShellModel;
```

Different implementations can be based on the same interface, e.g. an ideal model with exact switching behavior:

```
model IdealModel
  extends ShellModel;
  // implementation using
  // state events
end IdealModel;
```

and alternatively a smooth model:

```
model SmoothModel
  extends ShellModel;
  // alternative
  // implementation using
  // smooth non-linear function
end SmoothModel;
```

Further implementations can for instance provide models of different complexity, e.g. introducing different numbers of state variables.

Modelica supports the redeclaration of submodels. Exploiting this features, a system model defined for one application, say a real-time simulation, can be adapted to fulfill the requirements of an other application, say a dynamic optimization.

5.2 Model containing multiple implementations

Alternatively to defining different models for different formulations, one model can also provide multiple implementations. One possibility is to use the Modelica built-in operator **`analysisType()`**:

```
model UniversalModel
  // interface definitions
equation
  if analysisType() == "dynamic"
    // implementation using
    // state events
  else if analysisType() == "linear"
    // implementation using
    // smooth non-linear function
  end;
end UniversalModel;
```

The model translation tool picks out the appropriate implementation depending on the analysis type. Analysis type linear means that the continuous part of the model shall be transformed in a linear system. This implies that the model should be formulated in an appropriate way allowing linearization at given operating points.

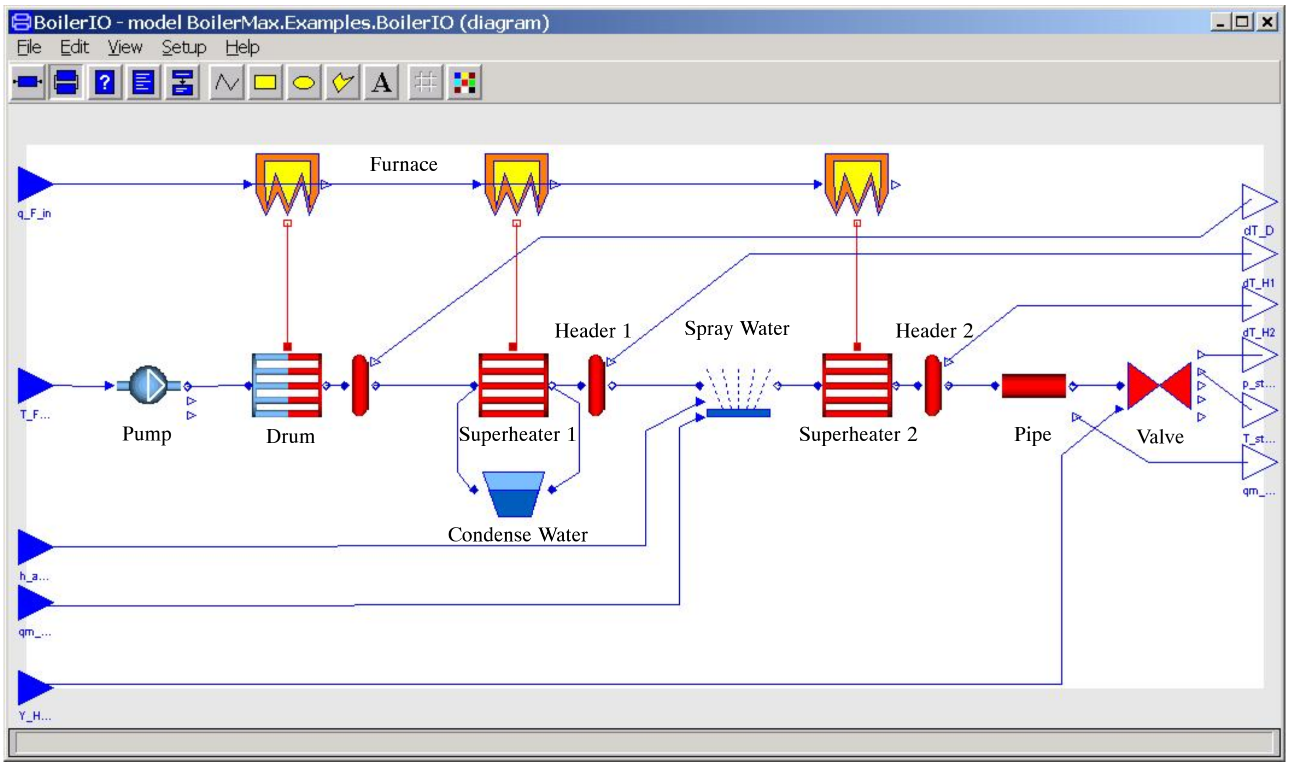

Figure 1: Flowsheet of a boiler model describing the generation of superheated steam.

5.3 Attributes of predefined types

The predefined Modelica type **`Real`** defines several attributes that are important for the formulation of optimization models. These are:

`nominal` The nominal attribute should be used to scale optimization variables (control inputs, unknown parameters, model states).

`stateSelect` This attribute is useful to guide the model translator to select specific states that will become optimization variables.

Furthermore the attributes **`min`** and **`max`** could be utilized to formulate bounds on model variables and constraints. However, it should be noted that Modelica is not intended to be an optimization modeling language. The primary intention of the attributes **`min`** and **`max`** is to restrict the range a model is valid for, not to define constraints like operational bounds.

Generally Modelica should not be seen as a modeling language to define a whole optimization problem, including optimization criterion and constraints. Instead Modelica is considered a powerful language to define the dynamic system model in a dynamic optimization problem.

6 Example

The optimal startup of a boiler for the generation of superheated steam in a coal fired power plant is discussed as example. The optimal control problem is to obtain a new operating point as fast and efficient as possible considering constraints on the thermal stress on thick walled parts, see [8]. Main new challenges, compared to approaches known so far, e.g. [7], are to formulate a nonlinear dynamic process model that is capable to accurately predict the behaviour over a wide range of operation, including cold start, to be open for flexible adaptation of the model to specific power plants, and to solve the optimal control problem considering constraints on multiple thermal stresses that may become active in different situations.

The process model is formulated in the object-oriented modeling language Modelica. This allows the flexible composition of a process model from sub-models for typical components. Figure 1 shows an example flowsheet. Submodels are a feedwater pump, an evaporator, two superheaters, a long pipe, and a high pressure bypass valve. Further submodels cover the furnace. The phenomenon of condense water is modeled in a separate submodel that is attached to the first superheater. A spray water inlet is placed between the two superheaters. Thick walled parts are outlet headers of the superheaters and the boiler drum. The model components are based on the ThermoFluid model library [15]. The ThermoFluid library implements, besides others, the IAPWS Industrial Formulation IF 97 standard for the thermodynamic properties of water and steam, enabling accurate and efficient models. The reuse of this model library is considered crucial for an effortable model development concentrating on application specific phenomena.

The implementation of water and steam properties in the ThermoFluid library is accomplished by partial derivatives allowing the flexible selection of state variables, see also the model development in [1]. In the example discussed here, mainly temperatures are selected as state variables, besides pressures and mass flow rates. This simplifies the treatment of constraints on thermal stresses.

Controlled inputs are the fuel flow rate, the amount of spray water, and the position of the outlet valve. Model outputs are pressure, temperature and mass flow rate of generated steam as well as three observed thermal stresses.

The Dymola tool is applied to generate a mathematical system of differential and algebraic equations as required for an efficient numerical solution. After collecting all submodels from the used model libraries, the overall differential-algebraic equation system (DAE) contains 636 variables and equations. This DAE is converted to a system of ordinary differential equations (ODE) with 11 dynamic state variables and is compiled to a Simulink S-function.

Note that the dynamic optimization method discussed here requires the mathematical model in the same form as simulation solvers do. This means that no optimization specific extensions are required to the Dymola model translator. The S-function is directly used to treat dynamic optimization problems, in our case the estimation of model parameters and the optimal boiler startup. Sensitivities are obtained by numerical differentiation of the model.

The optimal boiler startup problem is formulated for 60 time intervals. The control trajectories are parameterized piecewise linear. The resulting large-scale nonlinear optimization problem has 1034 optimization variables, 854 equality constraints, and 1212 inequality constraints. Its solution with the HQP solver takes about 3 minutes on a PC with Pentium III 850 MHz processor.

Figure 2 shows optimization results. The optimization solver has to obtain three trajectories for the controlled inputs so that the optimization criterion is minimized subject to the constraints on thermal stresses and the required new operating point. It can be seen that first the constraints on thermal stress of superheater 2 (dT_{SH2}) and drum (dT_D) are active. Later on, when the condense water has been evaporated, the thermal stress of superheater 1 (dT_{SH1}) is becoming active between 750 s and 1900 s. Generally the constraints are limiting the amount of fuel ($q_{m,F}$) that can be fed into the boiler. Starting from 1500 s, spray water ($q_{m,AW}$) is utilized to reduce the thermal stress on superheater 2. The thermal stress of the drum is becoming active again. The high pressure bypass valve (Y_{HPB}) is primarily used to control the steam flow rate ($q_{m,Steam}$), but it influences other process variables like steam pressure (p_{Steam}) and steam temperatures (T_{Steam}) as well. The required new operating point is reached after about 2500 s.

Such an optimization can be used as core routine of a nonlinear model based controller (NMPC). In this way startup cost savings of about 10% can be reached, compared to a traditional control strategy.

7 Conclusions

The general principle of Modelica of separating the model specification from the numerical solution method allows the reuse of simulation models for optimization. Furthermore, the object-oriented features of the Modelica language and the availability of model libraries greatly simplify the development of rigorous physical models for complex dynamic systems.

Nonlinear dynamic optimization problems can be treated efficiently as discrete-time optimal control problems and solved numerically by applying large-scale nonlinear optimization methods, see also [3]. This is especially true for problems with state constraints. The HQP dynamic optimization solver has been integrated with the Dymola modeling and simulation software using Matlab and Simulink as integration platform [13, 14, 2, 5].

The optimal startup of a power plant is discussed as example. The system model is formulated based on the ThermoFluid model library [15]. The reuse of model libraries is considered crucial for an effortable model development concentrating on specific phenomena of an application.

The example demonstrates the main strengths of model based predictive control: the treatment of multi-input multi-output problems and the consideration of state constraints. For reasons of efficiency, it is important to carefully select state variables during the modeling process. The treatment of state variables as optimization variables simplifies the consideration of state trajectory constraints and allows a more robust and efficient solution of the dynamic optimization problem, even though the problem size increases.

For the future it appears desirable that a model translation tool generates required sensitivity equations in addition to the model differential equations. Model libraries might provide alternative sub-models for specific phenomena, e.g. description of sudden changes with discrete events or with an approximate non-linear function. These sub-models could then be exchanged with each other depending on the intended application and requirements by the solution method.

References

[1] K.J. Åström and R.D. Bell. Drum-boiler dynamics. *Automatica*, 36:363–378, 2000.

[2] Dynasim AB. Dymola: Dynamic Modeling Laboratory. http://www.dynasim.se.

[3] R. Franke. *Integrated dynamic modeling and optimization of systems with seasonal heat storage*, volume 394 of *Fortschritt-Berichte VDI, Reihe 6 (in German)*. VDI-Verlag, Düsseldorf, 1998.

[4] R. Franke and E. Arnold. Applying new numerical algorithms to the solution of discrete-time optimal control problems. In K. Warwick and M. Kárný, editors, *Computer-Intensive Methods in Control and Signal Processing: The Curse of Dimensionality*, pages 105–118. Birkhäuser Verlag, Basel, 1997.

[5] R. Franke, E. Arnold, and H. Linke. HQP: a solver for nonlinearly constrained large-scale optimization. http://hqp.sourceforge.net.

[6] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, volume 19 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 1992.

[7] P. Kallappa, Michael S. Holmes, and Asok Ray. Life-extending control of fossil fuel power plants. *Automatica*, 33(6):1101–1118, 1997.

[8] Klaus Krüger, Manfred Rode, and Rüdiger Franke. Optimal control for fast boiler start-up based on a nonlinear model and considering the thermal stress on thick-walled components. In *Proceedings of the IEEE Conference on Control Applications*. Mexico City, September 2001.

[9] T. Maly and L.R. Petzold. Numerical methods and software for sensitivity analysis of differential-algebraic systems. *Applied Numerical Mathematics*, 20:57–79, 1996.

[10] Modelica Association. Modelica: Modeling of Complex Physical Systems. http://www.modelica.org.

[11] Walter Murray. Sequential quadratic programming methods for large-scale problems. *Computational Optimization and Applications*, 7(1):127–142, 1997.

[12] P. Spellucci. *Numerische Verfahren der nichtlinearen Optimierung*. Birkhäuser Verlag, Basel, 1993.

[13] The MathWorks, Inc. MATLAB: the language of technical computing. http://www.mathworks.com.

[14] The MathWorks, Inc. Simulink: for model-based and system level design. http://www.mathworks.com.

[15] Hubertus Tummescheit, Jonas Eborn, and Falko Jens Wagner. Development of a Modelica base library for modeling of thermo-hydraulic systems. In *Proceedings of the 1st Modelica Workshop 2000*. Lund, Sweden, 2000.

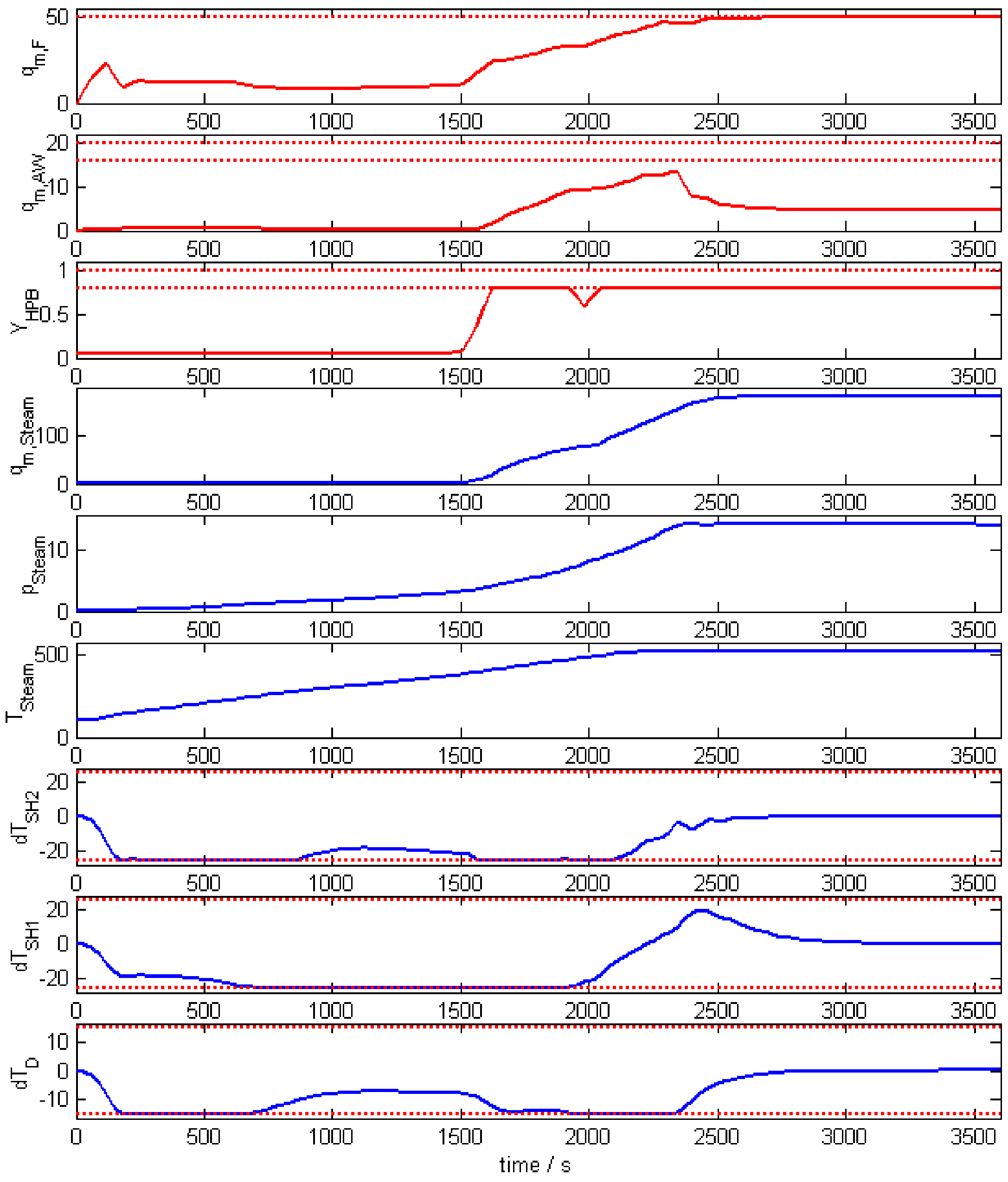

Figure 2: Results for the optimal boiler startup problem. The upper plots show the controlled inputs fuel flow rate $q_{m,F}/\%$, flow rate of spray water $q_{m,AW}/(kg/s)$, and position of high pressure bypass valve Y_{HPB}. Below process variables characterizing the generated steam are plotted: $q_{m,Steam}$/kg/s, p_{Steam}/MPa, $T_{m,Steam}/$℃. Furthermore three thermal stresses dT/K are shown.