Building and Solving Nonlinear Optimal Control and Estimation Problems

Jan Poland Alf J. Isaksson
ABB Corporate Research
<jan.poland@ch.abb.com> <alf.isaksson@se.abb.com>

Peter Aronsson
MathCore Engineering AB
< peter.aronsson@mathcore.com>

Abstract

We introduce a tool for obtaining optimal control and estimation problems from graphical models. Graphical models are constructed by combining blocks that can be implemented in Modelica or taken from a palette. The models can be used for predictive control, moving horizon estimation, and/or parameter estimation. We show that the solution time and robustness of the resulting nonlinear program strongly depends on the way the model was built and translated. These observations yield modeling guidelines for increasing robustness and efficiency of the optimization. In particular, we find out that eliminating as many variables as possible from the optimization problem may help a lot.

Keywords: Modelica; Optimization; Optimal Control; State Estimation; Receding Horizon; MPC; MHE

1 Introduction

Model-based methods have been important in many industrial applications for a long time, and their importance still increases today. One typical application field is simulation, where computer models are used to approximate physical processes to great accuracy. Today's models used for simulation are often very complex.

In contrast, computational limits are reached much earlier if a model is used for (online) *optimization*. Nevertheless, Model Predictive Control (MPC) [1] is nowadays a widely applied optimal control method which works by translating the model to an optimization problem, with the help of a performance measure ("cost function") defined in terms of the variables in the model. As opposed to other optimal control methods such as the linear quadratic regulator (LQR), this allows to accommodate constraints, which is very important in practice. In most applications, the optimization problem is formulated and solved for a fixed horizon in time, and the resulting first control move is applied to the plant. This procedure is applied repeatedly ("receding horizon control").

Most MPC applications work with models that are discrete in time or discretized. Depending on the type of the model (linear, nonlinear, involving continuous or discrete variables or both) and the cost functions, different types of optimization problems can arise: linear programs (if both the model and the cost function are linear), quadratic programs (linear model and quadratic cost function), mixed integer linear / quadratic programs (linear model with discrete variables), or (mixed integer) nonlinear programs (NLP) (resulting from a nonlinear model without or with discrete variables). We restrict our attention to the online optimization approach, where the full optimization problem has to be solved in each time step (as opposed to approaches where this is avoided, such as explicit MPC). Hence, the question if MPC can be used efficiently and robustly for an application can be answered in the affirmative if an appropriate solver for the optimization is available.

Consequently, most existing industrial applications use MPC based on *linear* models. Furthermore, since reliable and efficient mixed integer linear solvers have been available for some time now, also models with discrete variables become increasingly popular [2]. On the other hand, nonlinear models result in nonlinear programs (NLPs) which are much harder to solve in general. In particular, solving to global optimality is not possible unless the problem is small or additional structure is given (e.g. convexity holds). Consequently, for nonlinear MPC (NMPC), proving guarantees on the performance, stability, etc.

is often impossible. Still, there have been successful applications of NMPC, e.g. [3].

Recently, both computational hardware and nonlinear solvers have become more powerful, making nonlinear MPC applicable for more complex models in principle. In this paper, we will solve NMPC problems without further considering provable performance guarantees, stability, etc. All results in this work have been obtained with IpOpt [4]. We will see later on that it happens quite easily that models are translated to NLPs that are highly multimodal and very difficult to solve. Hence, it is important to construct models in a favorable way. Showing how to do so is one focus of this paper.

Design of the model is rightly considered to be the most difficult and involved task when constructing a model predictive controller. In industrial applications in particular, it is highly desirable that engineers with a moderate mathematical background are able to do so. For this aim, *graphical modeling environments* are especially appropriate: Models are constructed by using blocks and connecting them by lines. Each line represents a signal that leaves one block and enters another block[1].

Blocks should be intuitively understandable functions, e.g. summation of signals, some (nonlinear) function of a signal, or an integrator. A model library or palette should be available which contains a sufficiently flexible collection of predefined blocks, while it must be possible to implement customized blocks. In the present tool, this is done in Modelica. In short, the main graphical modeling functionality of existing Modelica environments (e.g. Dymola, MathModelica) is desirable.

A corresponding graphical modeling environment for linear models with continuous and discrete variables has been realized in ABB's commercial control and optimization platform Expert Optimizer [8], [5]. Combination of blocks is based on matrix multiplication in principle, and the resulting optimization problems (linear programs, quadratic programs, mixed integer linear programs, or mixed integer quadratic programs) are internally represented as matrices. Blocks can be implemented in a description language for this kind of hybrid systems, HYSDEL [6].

We will see that for nonlinear optimization, the way how blocks are implemented and combined can really make a computational difference. Standard graphical Modelica environments (e.g. Dymola, MathModelica) treat a model as a DAE system, and each block that is added to the system typically adds to the number of variables in the system. For instance, if the squared difference of some process variable x to some other signal is of interest, one could attach a corresponding difference to the signal x and then a square function to the difference. Usually, all these quantities will become extra variables in the DAE system. For simulating the system, this will not introduce particular difficulties. For optimization however, it can be very important that these extra variables do not enter the system, but are eliminated. In our framework, still Modelica code written in MathModelica is used to realize user-defined blocks[2]. When connecting blocks however, the chain rule is the crucial instrument that we use to eliminate variables.

In the tool we present in this work, graphical models are formulated (stated) in Matlab/Simulink. They are then used to state objective, constraints, derivatives, Jacobian and Hessian of the associated optimization problem by recursively parsing the model for each evaluation.

So far, we have been talking only of optimal control problems. Typical MPC needs starting values for all states of the model in order to compute and optimize future trajectories. If not all states are observed, then the model can be conveniently used for estimating them, using the *moving horizon estimation* (MHE, [10]) approach. Here, for a fixed number of time steps in the past, an optimization problem is formulated and solved in order to find those values for the state variables that are most in accordance with the observations and the model dynamics. Technically, the MHE task translates to a similar optimization problem as MPC. MHE can be extended to estimating some of the model parameters by adding them as states to the model.

The optimization framework discussed in this paper is similar to ABB's Dynamic Optimizer described in [7], but has a different scope: It is located at a higher level of a plant's automation system and to be integrated into Expert Optimizer, and it focuses on highly customizable modeling by offering a direct access to the graphical modeling environment. As opposed to other modeling frameworks, it *directly* translates a graphical model into an optimization problem, where care is taken to perform the transla-

[1] Note that this is less expressive than standard Modelica, which is object-oriented, and where signals can represent causal structure. However, since our graphical modeling environment allows importing Modelica blocks from MathModelica (see below), we do not lose Modelica's full expressiveness.

[2] Actually, MathCore and ABB have developed together the ABB edition of MathModelica, which the optimization framework we discuss is based on.

tion in a way that robust and fast-to-solve optimization problems are obtained. One key for this is variable elimination, as we will see below.

2 Translating a Graphical Model

Consider, for an illustration, the well-known inverted pendulum model

$$\ddot{\theta} = \frac{g \sin(\theta)}{l} + u,$$

where θ is the angle of the pendulum (the upright position is at $\theta = 0$), g is the gravitational force, l is the length of the pendulum (in the experiments below, we used $g / l = 0.5$), and u is (a constant multiple of) the torque applied. A graphical model representing this is shown in the red framed part of Figure 1. Here, the constants are hidden in the "plus" block, and the evolution of θ and $\dot{\theta}$ is implemented by single input single output linear time-continuous dynamics

$$\begin{aligned} \dot{x} &= Ax + Bu, \\ y &= Cx + Du. \end{aligned} \quad \text{(eq. 1)}$$

Each of the blocks "theta" and "theta_dot" has one own state, and the parameters are set to $A = 0$, $B = 1$, $C = 1$, and $D = 0$, respectively.

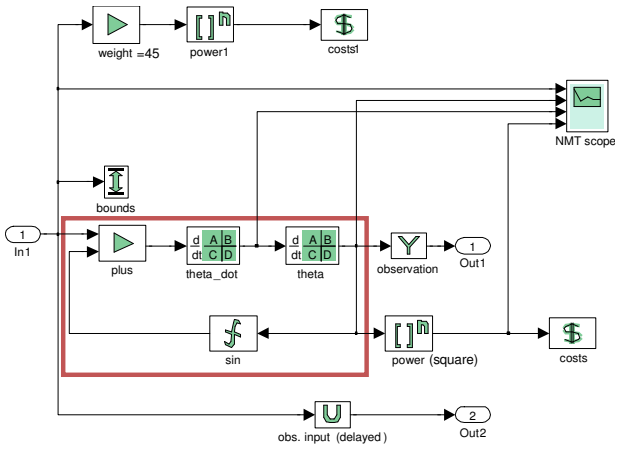


Figure 1: An inverted pendulum model

The implementation shown in Figure 1 is the Simulink model, which allows for a maximum variable elimination. A functionally equivalent model can be implemented graphically in MathModelica shown in Figure 2. Also, the following Modelica listing represents the same model:

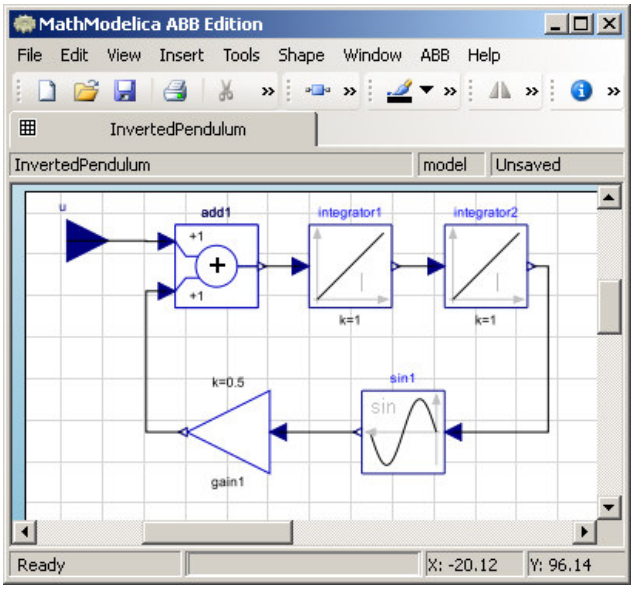


Figure 2: The inverted pendulum model graphically implemented in MathModelica

```
block InvertedPendulum
 parameter Real g;
 parameter Real l;
 Real theta;
 Real th_dot;
 output Real y;
 input Real u;
equation
 der(theta) = th_dot;
 der(th_dot)= g*sin(theta)/l+u;
 y=theta;
end InvertedPendulum;
```

2.1 The Optimal Control Problem

Translating a dynamic model into an optimal control problem including user-defined constraints is a well-established method: Let $x_t = (x_t^j)_{j=1}^n$ be the vector of variables in the model *realized* in a single time step *t*. A *realized* variable is a model variable that is present in the optimization problem, as opposed to a *virtual* variable, which is not present. Then an optimal control problem, discrete in time and for a horizon of $M > 0$, is stated as

$$J^{opt}(x) = \sum_{t=0}^{M} \text{cost}_t(x_t)$$

s.t.

$$x_t = \text{dynamics}(x_{t-1}) \text{ for all } 1 \le t \le M,$$

$$\text{equations}(x_t) = 0 \text{ for all } 0 \le t \le M,$$

$$\text{constraints}_t(x_t) \le 0 \text{ for all } 0 \le t \le M,$$

$$x_0^j = x_{start}^j \text{ for all } j \in \text{states}.$$

Here, the realized variables comprise at least the inputs to be optimized (and typically they comprise more variables). Costs and constraints may be time dependent, and x_{start} contains the starting values of the *states*. Note that this framework also permits the formulation of a static optimization problem, where $M = 0$ and *states* = ∅. Figure 1 shows how cost functions can be graphically stated.

This optimization problem is solved, in the present work, always with IpOpt [4]. We provide all derivative information up to the Hessian, computed analytically, to the solver.

Translation of Modelica components is done by MathModelica ABB Edition.

If we are dealing with a time continuous system, the system must be discretized. Here and in the following we assume that discretization is performed according to implicit Euler, i.e. a differential equation

$\dot{x} = f(x, u, ...)$ is translated to

$\mathrm{x}_t = x_{t-1} + \Delta t \cdot f(x_t, u_t, ...).$

A crucial question to be posed here is which respectively how many variables should be realized as part of x_t. It is clear that at least the inputs to be optimized and the states need to be realized (unless we are able to and desire to recursively solve the evolution equations for the states). However, we can include more or less of the variables defined by the equations into our optimization problem[3]. This decision has a significant impact on the computational time required to solve the resulting optimization problem, as well as the robustness of the solving. We show two numerical examples at this point and draw some conclusions.

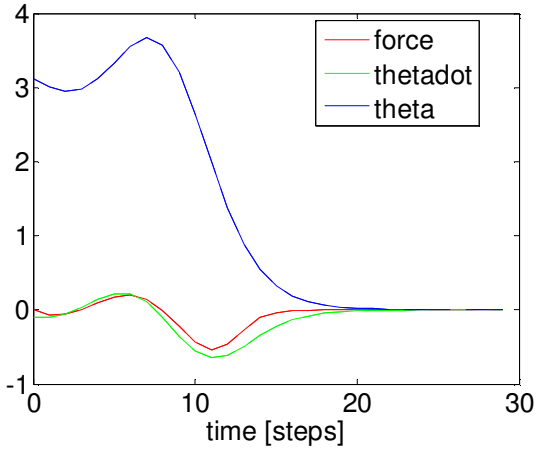


Figure 3: Swing-up trajectory for the inverted pendulum

[3] Not including, i.e. eliminating a variable that appears in a cost function or a constraint implies that we will need to use the chain rule to compute its derivatives.

For the first example, we consider the inverted pendulum model. The weights and bounds on the input are tuned in a way that the optimum is a swing-up, as shown in Figure 3. In the following table, we evaluated the probability that IpOpt finds this optimal solution, starting from randomly initialized values (uniformly in [0, 10]) for all realized variables. We further show the time IpOpt requires on average, as well as the average quality of the successful solutions: the number of time steps after which the predicted trajectory converges to the target. The averages of 500 runs each are shown.

Realized variables	Success rate[%]	Quality [steps]	Time [s]
States and input only (3 per step)	41.2	22.7	0.27
States, input + input and output of θ (5 per step)	42.4	23.0	0.37
As above plus input and output of $\dot{\theta}$ (6 per step)	46	22.6	0.42
As above plus output of the "sin" block (7 per step)	37.4	22.7	0.55

The second numerical example we show is a static nonlinear optimization problem defined by the model in Figure 4, which represents the equation

$$y = u_2^2(u_1 - u_2)^3\left(2(u_1 - 8)^2 + 1 + 8.649 \cdot 10^{-9}\exp(2u_1) - 4.65 \cdot 10^{-5}\exp(u_1)\right) + 5(u_1 - 8)^2$$

s.t.

$5 \le u_1 \le 9.8707$

$1 \le u_2 \le 4.$

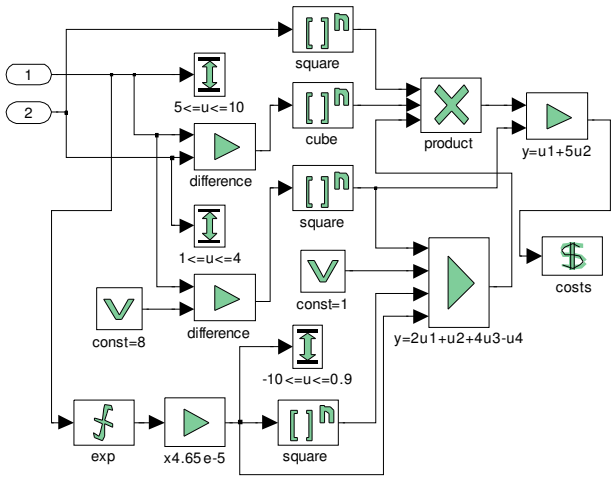


Figure 4: A static nonlinear optimization problem

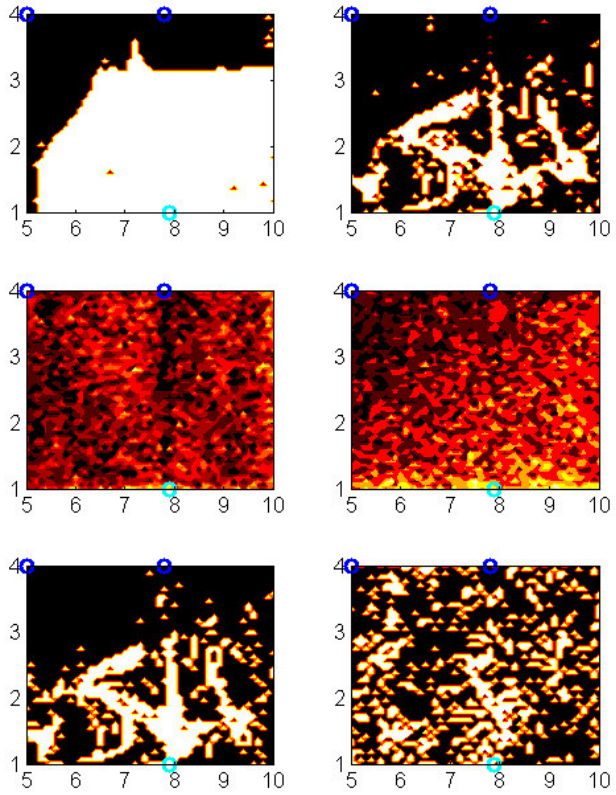


Figure 5: Static nonlinear optimization problem: Probability of converging to the global optimum for six different scenarios, shown in dependence of the 2-dimensional starting points. Hot colors indicate high probabilities, the large area in the top left plot is probability one. The globally optimal solution is shown as a cyan circle, there are two local optimal shown as blue circles. Top left: only the inputs are realized (1), top right: the inputs and the output of the product are realized (2), middle left: the inputs and in- and outputs of the product are realized (3), middle right: all signals are realized (4), bottom: scenarios (2a) and (3a) with consistent initialization.

The first input variable is plotted over [5, 10], the second input variable over [1, 4]. We test four different scenarios: (1) realizing only the input variables (i.e. 2 variables), (2) realizing in addition the output of the product (i.e. 3 variables), (3) realizing in addition the inputs of the product (i.e. 6 variables), and (4) realizing all signals in Figure 4. The resulting computation time and probability of finding the global optimum are summarized in the table below. Figure 5 shows the probabilities of convergence as a function of the starting point for the two inputs (all other realized variables have been initialized with Gaussian distributed values). We observe that for starting points around the global optimum, we reliably converge in scenario (1), while this is no longer true in scenario (2) and seems almost completely arbitrary in scenarios (3) and (4). If we modify scenario (2) and (3) such that the realized variables are initialized with the values corresponding to the inputs instead of Gaussian, we get scenarios (2a) and (3a), and the computation times and probabilities can be seen below.

Scenario	Success rate [%]	Time [s]
(1)	62.0	0.004
(2)	22.1	0.004
(3)	10.9	0.021
(4)	20.2	0.01
(2a)	22.1	0.005
(3a)	27.4	0.022

2.2 Consequences and Recommendations for realizing or eliminating variables

The two examples clearly show that the time required for solving the optimization problem increases with the number of realized variables. Although this has been observed for IpOpt only here, it is realistic to say that the statement can be generalized to any solver. In particular, more variables require more equality constraints to be satisfied and hence typically increase the number of iterations needed by the solver. In the second of the above examples, scenarios (3) and (4) need about 20 times more CPU times than (1) and (2). Looking a bit closer into what happens when IpOpt optimizes scenarios (3) and (4), what can be observed is the following: In the first step of the optimization, the equality constraints are non-satisfied to a high degree, causing the actual solution to move very far away, where in particular the input variables leave the box. Then, it takes quite long for the solution to get back into the region which is feasible for the input variables. The dependence on the exact starting points is very sensitive (chaotic), which causes the rough behavior of the middle two graphs in Figure 5. This happens to a much lesser degree in scenario (2), and not at all in scenario (1). Even when we use a consistent initialization, i.e. one that satisfies the constraints, we see in the lower two graphs that we do not improve the roughness much.

In the inverted pendulum example, we see that similar things happen (not as strongly as in the other example). However, introducing additional variables may to a certain extent even help the optimizer to find the global optimum.

In general, we expect that the optimization problem will be not only more efficiently, but also more robustly solved, the *less* variables are realized. In a MPC situation, typically, we start already close to the optimal solution, since the starting solution is obtained by the optimum from one step before. Hence, the roughness observed for the scenarios (2), (3) and (4) of the second example is a very undesirable property here: it increases the probability that even in the absence of major disturbances, we will not be able to track the optimal trajectory.

We summarize: with IpOpt as underlying solver, our results indicate that for both robustness and solution time it is favorable in general to realize as few variables as possible. For the robustness, there are exceptions where realizing *some* variables may be good. The solution time is always expected to increase with more variables, this should also hold for other solvers.

2.3 The Estimation Problem

The optimal control problem needs in particular starting values for all states. If they are not directly observed, they can be estimated with a model-based observer that uses the same model as MPC. To this aim, the following cost function ("moving horizon estimator") can be minimized [10]:

$$J^{est}(x) = \left\|\varepsilon^{initial}\right\|_2^2 + \sum_{t=-N+2}^{0} \left\|\varepsilon_t^{state}\right\|_2^2 + \sum_{t=-N+1}^{0} \left\|\varepsilon_t^{obs}\right\|_2^2$$

s.t.

$$x_t = \text{dynamics}(x_{t-1}) + \sigma_t^{state}\varepsilon_t^{state} \quad for\ all\ -N+2 \le t \le 0,$$

$$\text{equations}(x_t) = 0 \quad for\ all\ -N+1 \le t \le 0,$$

$$\text{constraints}_t(x_t) \le 0 \quad for\ all\ -N+1 \le t \le 0,$$

$$y_t^{obs} = y(x_t) + \sigma_t^{obs}\varepsilon_t^{obs} \quad for\ all\ -N+1 \le t \le 0,$$

$$x_{initial}^j = x_{-N+1}^j + \sigma_{initial}^j \varepsilon_{initial}^j \quad for\ all\ \ j \in states.$$

Here, we minimize the quadratic state noise ε_t^{state}, measurement noise ε_t^{obs}, and initial state noise $\varepsilon_{initial}$, with weighting factors that are standard deviations (i.e. scales of the noise) σ_t^{state}, σ_t^{obs}, and $\sigma_{initial}$, respectively. For the standard deviations, the valid range is [0, ∞]. The quantity y_t^{obs} contains the observations available at time t, while $x_{initial}^j$ is the target for the initial state (if desired). The constraints may but need to be the same as in the definition of J^{opt}, in general it makes sense to consider a subset here.

It is just a matter of notational convenience to include the noise variables ("slack variables") ε_t^{state}, ε_t^{obs}, and $\varepsilon_{initial}$ explicitly in the formulation of the optimization problem. It not necessary to realize them, in fact, it may be more efficient not to do so. In the following simulations, we do not realize the noise variables.

Figure 6 shows an example trajectory of the estimation (and then optimization) with the inverted pendulum model from Figure 1. There are two observations defined, namely the input force and θ. The speed $\dot{\theta}$ is not observed, but is reliably estimated.

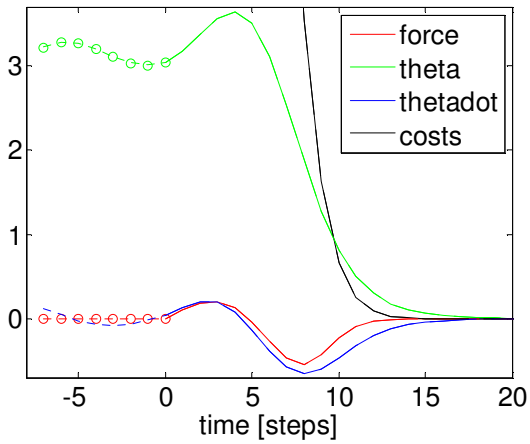


Figure 6: Estimation and optimization trajectory for the inverted pendulum model. Observations are marked by circles, dotted lines are estimations.

2.4 Reducing Interactions in Case of Poor Observability

The toolbox presented in this work facilitates combining models of a large plant from its parts, and optimizing the plant as a whole. In practice however, the whole plant may not be sufficiently observable to allow a reliable state estimation based on the complete model. Namely, if there are relatively poor measurements available in the parts of the plant, the estimation will use the model in order to obtain the numerically most likely estimator for the joint state of the plant. If the model does not very well match the plant, this procedure can easily result in estimates that are poor, or that drastically change from one time step to the next. Of course, this is highly undesirable, as it prevents the model-based controller from stabilizing the plant.

Here, we describe two techniques for cutting the model in several parts that do not interact in the estimation. Hence, the measurements available within one part of the plant are used only within that part, and do not interact with different parts.

2.4.1 Cutting with dynamics

If the parts to be separated are connected by one or more blocks that evolve dynamically, e.g. a delay or

a filter, then it is possible to not estimate the states of these blocks, but rather provide their values by an external simulation. In the estimation problem, the dynamics of these blocks are removed from the cost function / constraints.

2.4.2 Cutting without dynamics

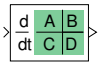
If two parts to be separated are connected without dynamics, then the following method can be used: The connection is broken, the estimation is executed with open connection, and some constant value is used as input for the destination of the broken connection. After the estimation, the states in the part connected to the source of the broken connection have attained their values. Now, also the value the broken signal should have is known. Hence, we need to use this value as an input for the destination of the broken connection and rerun the estimation. In general, if the model is divided into n parts, we need to run the estimation for n times. Note that this method may not converge if the dependence of the parts is circular, hence it should be used only for tree structures (and we should run the estimation n times, where n is then length of the longest branch of the tree). Also, a signal to be cut must have a clear direction, providing its value from the source to the destination. It must not be used as an implicit input, which enters the dynamics or equations at the source and is defined by some constraint at the destination.

2.5 Parameter Estimation

If observability permits, parameters can be coded as states and estimated by the state estimation. One standard procedure to do so is coding a parameter as a state which evolves as a constant and admits state noise.

3 A Building Block Library

Here we describe a basic set of blocks that is sufficient to model a broad variety of plants.



3.1.1 Single input single output linear time-continuous dynamics, as described in (eq. 1).

3.1.2 Single input single output linear time-discrete dynamics:

$$x_t = Ax_{t-1} + Bu_{t-1},$$
$$y_t = Cx_t + Du_t.$$

3.1.3 Constant

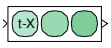
3.1.4 Adaptive constant / bias term, see 2.5.

3.1.5 Nonlinear function, to be chosen from {exp, log, sin, cos, tan, sinh, cosh, tanh, asin, acos, atan}

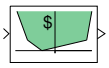
3.1.6 Power, i.e. $y = u^n$

3.1.7 Weighted sum (linear combination), i.e. $y = a_1u_1 + \ldots + a_nu_n$. Also implements gains.

3.1.8 Product: $y = u_1 \times \ldots \times u_n$



3.1.9 n-step delay



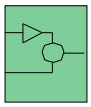
3.1.10 Generalized absolute function. The function $y = \text{abs}(u)$ can be implemented using an auxiliary variable z to be minimized, and including $z \geq u$ and $z \geq -u$ to the constraint set. This block implements a general convex piecewise linear function with the same principle.

3.1.11 Upper and lower bounds

3.1.12 Marks a signal as a cost function

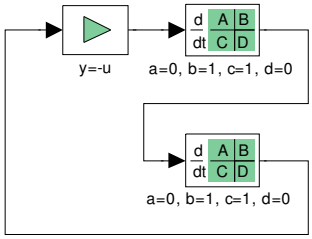
3.1.13 Marks a signal as an observation

3.1.14 Cut for estimation without dynamics as described in 2.4.2.



3.1.15 Users can implement own Modelica blocks.

Note that these basic blocks are sufficient to express a wide range of coupled ODE systems. For example, the following model encodes a harmonic oscillator.



4 Applications, Discussion and Conclusions

We have presented several simulations with toy examples in this paper. The modeling framework presented here will become part of ABB's advanced process control and optimization platform Expert Optimizer, which is described e.g. in [8]. This article and [7], [9] report some results from application of similar modeling techniques to industrial plants. Our modeling framework has been based on the results of ongoing collaboration of ABB and MathCore, [11] describes an early application of related methods.

We have presented a graphical modeling framework and a procedure to directly translate graphical models into optimization problems. We have collected some evidence that, for formulating the optimization problems, it is favorable for speed and robustness to realize as few variables as possible.

References

[1] J. Maciejowski: Predictive Control with Constraints. Pearson Education Limited, Prentice-Hall, Essex, United Kingdom (2002).

[2] A. Bemporad and M. Morari: Control of systems integrating logic, dynamics, and constraints, Automatica 35(3) (1999), 407-427.

[3] R. Franke, M. Rode, K. Krüger: On-line Optimization of Drum Boiler Startup. Proceedings of the 3rd International Modelica Conference, Linköping, November 3-4, 2003.

[4] A. Wächter and L. T. Biegler: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25-57, 2006.

[5] E. Gallestey, D. Castagnoli, and A. Stothert: Method of generating optimal control problems for industrial processes, European Patent EP1607809 (2006).

[6] F.D. Torrisi and A. Bemporad: HYSDEL - A tool for generating computational hybrid models for analysis and synthesis problems. IEEE Transactions on Control Systems Technology 12(2) (2004), 235-249.

[7] R. Franke, B. S. Babji, M. Antoine, A. Isaksson: Model-based online applications in the ABB Dynamic Optimization framework. Proceedings of the Modelica Conference, 2008, pp. 279-285.

[8] K. Stadler, E. Gallestey, J. Poland, G. Cairns: Optimal Trade-Offs. ABB Review, 2/2009, pp. 17-22.

[9] E. Dahlquist, F. Wallin, H. Ekwall: Dynamic simulators for process control and optimization as well as for operator training in pulp and paper industry, SIMS - 43rd Conference on Simulation and Modeling, Oulu, 2002

[10] C.V. Rao: Moving Horizon Strategies for the Constrained Monitoring and Control of Nonlinear Discrete-Time Systems, Ph.D. Thesis, University of Wisconsin, 2000.

[11] J. Pettersson, L. Ledung and X. Zhang: Decision support for pulp mill operations based on large-scale on-line optimization, Preprints of Control Systems 2006, Tampere 6-8 June, 2006.