BLOM: The Berkeley Library for Optimization Modeling and Nonlinear Model Predictive Control

Anthony Kelman, Sergey Vichik, and Francesco Borrelli

Department of Mechanical Engineering
University of California, Berkeley, CA 94720-1740 USA
e-mail: {kelman, sergv, fborrelli}@berkeley.edu

Abstract: We describe a new library for model integration and optimization formulation of linear and nonlinear model predictive control.

1. INTRODUCTION

When modeling of a dynamical system is considered, a researcher or engineer may choose a tool from two main groups: simulation oriented tools and optimization oriented tools. Model formulation for the purposes of optimization-based control does not fit exclusively in either of these groups and imposes a set of requirements on modelers and the tools they use. Requirements not commonly met by modeling tools designed for conventional forward simulation include: representation of constraints and a cost function, distinguishing unknown input signals that can be freely chosen from signals with known input values, and efficient calculation of derivatives for gradient, Jacobian, and Hessian information used by optimization algorithms. The existing simulation modeling languages such as Simulink or Modelica (Fritzson and Engelson, 1998) use block diagram or object oriented approaches to describe systems. This approach is widely accepted due to its natural perception by system and control engineers and good scalability.

Many languages and tools exist for optimization modeling (Kallrath, 2004), but a common weakness of existing methods is that they are difficult for multiple engineers to use in collaborative model development for large-scale systems. Model integration and representing signal flow and connectivity between distinct, independently developed component or subsystem models cannot be done productively using unfamiliar optimization modeling tools. A very large model that is developed using a conventional simulation and technical computing environment like Matlab cannot be easily ported into an optimization formulation in languages like AMPL (Fourer et al., 2003), GAMS (Rosenthal, 2008), AIMMS (Bisschop, 2006), etc. Unlike in the simulation tools, the model formulation in the existing optimization languages is not natural for most engineers, therefore it is prone to mistakes and harder to use.

Recently the Optimica (Åkesson, 2008; Åkesson et al., 2010) extension for the Modelica language was introduced. This extension facilitates the conversion of a dynamic model into an optimization problem. However, this library lacks intuitive graphical model representation and is compatible only with Modelica models.

To remedy this situation we have created a new tool, the Berkeley Library for Optimization Modeling (BLOM). The primary intended use of this library is for nonlinear model predictive control (MPC) problems, but static optimization problems with no system dynamics or time horizon can also be represented, formulated and solved with BLOM.

2. BASIC FORMULATION AND BUILDING BLOCKS

BLOM is a collection of block diagram components for the Simulink graphical modeling environment, and Matlab functions to export system models to a variety of optimization solvers. The visual block diagram representation of a system model in Simulink intuitively captures the signal flow, connectivity, and hierarchy of a large-scale system. A BLOM model can also use the standard forward simulation features of Simulink for verification and comparison with optimization results. Thus, the same model can be initially ran in forward simulation mode, in order to compare it to reference data and verify constraint satisfaction, and later the same model can be automatically exported to an optimization solver. Our convention for optimization modeling is that every output signal of a BLOM block corresponds to an optimization variable. As in standard Simulink models, a signal (wire) can be scalar or vector-valued.

2.1 Polyblocks: sparse multivariate nonlinear functions

The primary building block used by BLOM is a representation of a nonlinear multivariable function, which we call a Polyblock. A Polyblock has n inputs, which can either be n separate scalar signals, a vector signal with n elements, or a concatenated set of multiple scalars and/or vectors (using the Simulink Mux block) with a total of n elements. A Polyblock has m outputs which can either be m separate scalar signals or a vector signal with m elements.

In the general case, denoting the input elements by u_j and the output elements by y_j, the input-output functional relationship of a Polyblock is defined by the following equality constraint.

$$0 = \sum_{k=1}^{r} K_{ik} \left(\prod_{j=1}^{n} v(u_j, P_{kj}) \right) \left(\prod_{j=n+1}^{n+m} v(y_{j-n}, P_{kj}) \right), \quad \forall\, i \in \{1, \ldots, m\} \tag{1}$$

The parameterized function v is defined as

$$v(x,p) = \begin{cases} x^p & \text{if } p \text{ is not an exception code} \\ \exp(x) & \text{if } p \text{ is the code for exp} \\ \log(x) & \text{if } p \text{ is the code for log} \\ \text{etc.} & \end{cases} \tag{2}$$

We reserve a list of exception codes for transcendental functions. This list can be extended to include any differentiable single-operand function.

The matrices $K \in \mathbb{R}^{m \times r}$ and $P \in \mathbb{R}^{r \times (n+m)}$ contain, respectively, the coefficient and exponent data of a multivariable polynomial-like function. The number of monomial terms in this function is given by r. Typically the matrix P will be sparse, and K may be sparse as well. Linearly dependent rows of K would represent redundant constraints, so we assume K is full rank. Let $P = [\, \widehat{P} \;\; \widetilde{P} \,]$, where $\widetilde{P} \in \mathbb{R}^{r \times m}$. We assume each column of $\widetilde{P}$ contains at least one nonzero entry, otherwise some elements of the output vector y could take on any value. If any nonzero entry in $\widetilde{P}$ is not equal to 1, or if there are multiple nonzero entries in any row of $\widetilde{P}$, then the output y is defined implicitly and there may not be a unique solution. In this case the output of the polyblock is undefined in forward simulation, but the equality constraint (1) is valid in an optimization problem.

Note that this polyblock format is general enough to represent rational functions as well. If the desired input-output relationship is $y = f(u)/g(u)$ where $f(u)$ and $g(u)$ are polynomial-like functions and $g(u) \neq 0$, this can be captured by encoding the constraint $0 = f(u) - y \cdot g(u)$ in the P and K matrices of a polyblock.

We choose to formulate the optimization problem by introducing variables and equality constraints for the output signals of every nonlinear function. This increases the size of the optimization problem compared to a substitution approach, but has the benefit of preserving system model sparsity structure in the optimization formulation. A substitution approach would lead to an optimization problem with fewer variables and constraints, however the constraints and cost function would be denser and the calculation of gradient, Jacobian, and Hessian information would be more difficult due to nested nonlinearities.

2.2 Constraints

Constraints are represented in BLOM by Simulink blocks with one input signal. An option setting determines whether a constraint block is a nonnegativity constraint ($u \geq 0$), a nonpositivity constraint ($u \leq 0$), or an equality constraint ($u = 0$). If the input signal is vector-valued, then the constraint is applied elementwise.

Constraints may be violated during forward simulation. In this event we can notify the user with a Simulink assertion, warning messages, visual indication such as changing block colors, and/or an optional boolean output indicating whether a constraint is satisfied or violated.

2.3 Cost function

The cost function for optimization is also represented in BLOM by a Simulink block with one input signal. This block does not play any role in forward simulation mode, but the input signal is treated as a cost function in an optimization problem. If the input signal is a vector, the current version sums the input elements. In the future, an option setting will be available to determine which norm to take over the input elements (1, 2, or ∞).

For models with system dynamics and time horizons longer than 1 step, the cost function is a discrete accumulator of this signal. This functionality can be further expanded to continuous integrator, a terminal cost (final value of the input signal), the peak value of the input signal over the time horizon (∞ norm), or another norm (1, 2) over time of the input signal.

If there are multiple cost function blocks within the same BLOM model, the optimization cost function is treated as the sum of the values over all cost function blocks.

2.4 Control and external inputs

There are two classes of input signals to a BLOM model: unknown signals that the optimization algorithm is free to choose in order to minimize the cost function subject to constraints, and time-varying signals with a known trajectory of values over a future horizon. We will refer to the former as control inputs, and the latter as external inputs. In a MPC problem, external inputs correspond to predicted future model parameters or disturbances.

Both classes of inputs are parameterized as uniformly-sampled time series. Time variation for each input signal within one time step is important for discretization accuracy. It can be either piecewise linear and continuous (first order hold (FOH)), or piecewise constant with discontinuities at the sample times (zero order hold (ZOH)). Current implementation supports only FOH, and ZOH support will be added in the future.

We assume all input signals have the same sample rate, with the exception of an optional simple implementation of move blocking on control inputs (Cagienard et al., 2007). In order to reduce the number of optimization variables, the user can specify that certain control inputs should have constant values over some integer number of time steps.

In forward simulation, control input and external input blocks take input signals from non-BLOM sources in Simulink, such as Constant or From Workspace blocks. Optimization formulation only requires the dimensions of control and external input signals. The values used for external input trajectories over the optimization horizon are communicated between the Matlab workspace and the optimization solver, and do not need to be the same values as used in the Simulink block diagram.

2.5 Discrete and continuous states

BLOM models can support discrete and continuous states. Discrete states are equivalent to the $1/z$ Unit Delay block in Simulink, and continuous states are equivalent to the $1/s$ Integrator block in Simulink. If a BLOM

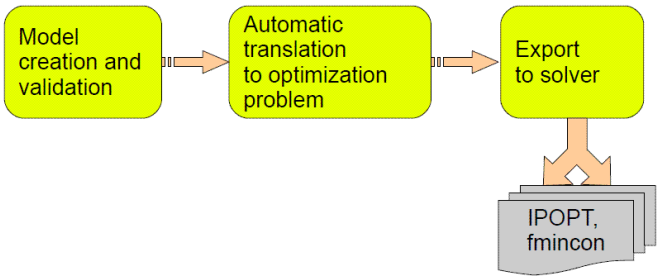

Fig. 1. Work flow with BLOM. First a model is created and validated using the BLOM library. Then, it is converted to an optimization problem and exported to one of the supported solvers.

model contains multiple discrete state blocks, they are all assumed to have the same sample rate. In the future, for hybrid discrete/continuous-time models, the inter-sample behavior of a discrete state block will be set to either zero order hold or first order hold. In current version this behavior depends on selected discretization method.

Continuous-time models are converted into finite dimensional optimization problems using a fixed-timestep discretization method. If there are both discrete and continuous states in the same model, the discretization step length for the continuous states must be equal to the sample time of the discrete states. The discretization method and the timestep length are specified by the user. The discretization method can be any general Runge-Kutta method or linear multistep method. Methods can be either explicit or implicit, and of arbitrary order. Runge-Kutta methods require the introduction of additional variables for intermediate function calculations. Multistep methods, which will be supported in the future, require multiple steps of state and input history for initialization.

3. AUTOMATED GENERATION OF AN EFFICIENT OPTIMIZATION PROBLEM

As shown in Figure 1, after it is created and validated, the BLOM model can be converted to an optimization problem and exported to a solver. This section details this process.

3.1 Model validation

If the Polyblocks in a BLOM model are of explicit form (see section 2.1), the model can be executed in forward simulation mode. In this mode, the model can be verified by comparing it to reference data and checking constraints feasibility. Although ensuring feasibility is important, it may be intractable in forward simulation mode, especially for non-trivial models. Therefore, this step is not mandatory.

3.2 Gradient, Jacobian and Hessian evaluation

The BLOM formulation facilitates calculation of closed-form gradients, Jacobians and Hessians for solvers that make use of derivative information, such as IPOPT (Wachter and Biegler, 2006). In the optimization formulation, there is no need to distinguish between input and output variables (u and y in (1)). Therefore, (1) for a single row i of K can be restated as

$$f_i(\mathbf{x}) = \sum_{k=1}^{r} K_{ik} \left(\prod_{j=1}^{n+m} v(x_j, P_{kj}) \right), \tag{3}$$

where $f_i(\mathbf{x})$ is the constraint function of vector $\mathbf{x}$. The derivative of a $f_i(\mathbf{x})$ with respect to a variable x_d is

$$\frac{\partial f_i(\mathbf{x})}{\partial x_d} = \sum_{k=1}^{r} K_{ik} \frac{\partial v(x_d, P_{kd})}{\partial x_d} \left(\prod_{j \in \mathcal{J}} v(x_j, P_{kj}) \right), \tag{4}$$

where $\mathcal{J}$ is the set of variable indexes excluding d, $\mathcal{J} = \{1, \ldots, d-1, d+1, \ldots, n+m\}$. The second derivative can be computed in a similar way.

Only the non zero elements of the Jacobian and Hessian matrices are exported to a solver, because sparsity structure is immediately available from the K and P Polyblock matrices. Therefore, BLOM preserves sparsity structure which results in efficient performance for very large problems.

3.3 Code generation

BLOM library can currently generate efficient C and Matlab source code for use by the solvers. The optimization problem can be exported to a solver using two methods

(1) Explicit code generation for each element of the cost, gradient, Jacobian and Hessian.
(2) Efficient storage of the Polyblock functional representation to be used by a generic BLOM adapter for a solver.

The first approach is implemented to interface with Matlab Optimization Toolbox solvers, such as fmincon. Although providing the best execution times, the explicit approach becomes intractable for very large problems. When the number of non zero elements increases to tens or hundreds of thousands, the explicitly generated code cannot be compiled and executed due to its size. Therefore, for large scale problems, the second approach is used. When implemented as a C++ adapter for IPOPT, this approach is about two times slower than the explicit approach. In large problems, Jacobian and Hessian evaluation takes less then 5% of the total solver time, so selecting the second approach does not have a significant impact on total solver time.

3.4 Solvers and environments

BLOM supports Matlab solvers (fmincon, linprog and quadprog) and the compiled IPOPT solver. We plan to add support for the Matlab Mex interface to IPOPT, AMPL, and other optimization environments and solvers.

4. EXAMPLES

4.1 Simple MPC

Consider the discrete dynamic system $x_{k+1} = 0.9x_k + u_k$ with bounded input $|u| \leq 2$ and state constraints $0.5 \leq x \leq 1$. We want to create a constrained finite time optimal

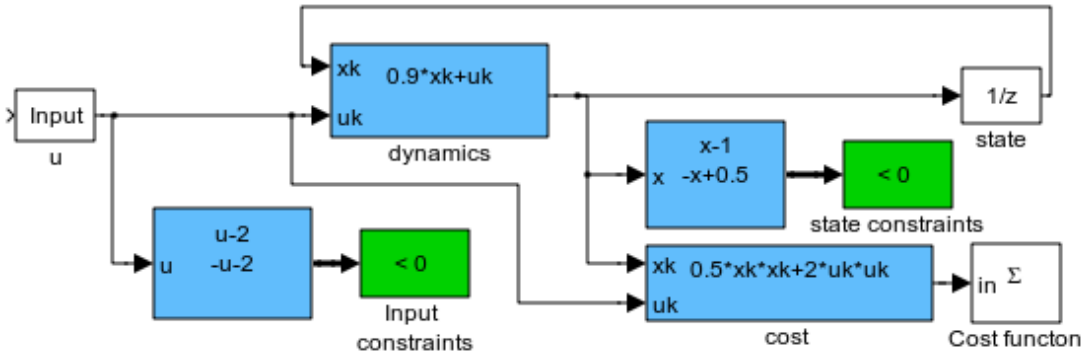

Fig. 2. Simple example of dynamic system with state and input constraints and a quadratic cost function.

Table 1. BLOM with IPOPT performance on a large HVAC problem for various prediction horizon lengths

Prediction horizon length	5	10	15	20	25	30
Number of variables in solver	7147	12176	18553	24846	31223	37516
Number of constraints	7329	15151	22974	30796	38619	46441
Non-zeros in Jacobian and Hessian	24057	52208	80442	108593	136827	164978
Export time to IPOPT [sec]	11	18	30	57	96	1114 *
Number of solver iterations	46	230	89	82	65	68
Total solution time [sec]	2.5	8.6	26.2	48	42	97
Time spent in BLOM callbacks	11%	11%	9%	6.8%	8%	5%

* The jump in export time is due to a memory allocation bottleneck. This is an implementation issue that will be resolved in future versions.

control (CFTOC) problem with cost $J = \sum_{k=0}^{N} 0.5x_{k+1}^2 + 2u_k^2$.

Figure 2 shows a BLOM model that is used to create the CFTOC problem. The system dynamics are implemented in the "dynamics" Polyblock (blue), the cost function is calculated by the "cost" Polyblock. Two other polyblocks are used to calculate vector constraint functions. Two constraint blocks (green) are enforcing "less than zero" inequality condition. This system has a single input u that is marked by an input block to let BLOM know that it is a free optimization variable. In order to convert this model to an optimization problem, the user specifies the prediction horizon (N) and the following CFTOC problem is created

$$\begin{aligned} &\min_{u_k, x_k} \sum_{k=0}^{N} 0.5x_{k+1}^2 + 2u_k^2 \\ &s.t. : -2 \leq u_k \leq 2,\ 0.5 \leq x_k \leq 1, \\ &\quad x_{k+1} = 0.9x_k + u_k,\ x_0 = x(0) \\ &\quad k = 0, \ldots, N. \end{aligned} \tag{5}$$

4.2 Large scale HVAC example

BLOM is used for nonlinear MPC design of a large HVAC sytem. The system consists of 42 thermal zones. The system dynamics are modeled with 430 state variables that represent thermal masses of elements in a building. In addition the model includes an air handling unit (AHU) model, fan model and 41 variable air volume (VAV) box models with one reheating coil each. The thermodynamic model is bilinear and additional nonlinear terms exist in the model. This system has 85 control variables that need to be determined at each time step.

Table 1 presents performance of the BLOM library with IPOPT solver on this problem. We present the execution time of problem preparation and solution for various problem sizes. The table shows that even for very large problems with more than 20000 variables and constraints, the library achieves good performance and solves the CFTOC problem quickly.

5. CONCLUSION

The BLOM library enables development and solution of nonlinear large scale models and optimization problems. It does so by exploiting sparse structure of the problem and creates an efficient problem formulation for optimization solvers. Model description in Simulink enables for easy collaboration between multiple developers, a familiar and intuitive modeling environment, and integration with existing tools for Simuilink based development.

The library may be useful for any developer or researcher, who needs to develop and solve non-trivial optimization models, especially dynamical models for MPC implementation.

The current library does not support, and has no immediate plans to add support for: integer variables (mixed integer problems), partial differential equations and semi-definite constraints.

5.1 Future development

Although the BLOM library is already used in real projects, it may be further improved and expanded. Here is a partial list of capabilities that can be added with reasonable development effort:

(1) Support for general Simulink blocks - almost any algebraic Simulink block can be converted to a Polyblock, and, therefore, be integrated in the optimization model. Currently only Polyblocks and Mux/Demux are supported. Adding support for any block allows for easier model development and creates more readable and standardized models. If this support is added, even model that were developed originally for other purposes may be converted to optimization problems.
(2) Support for multiple subsystems. Currently, the models must reside in a single subsystem.
(3) Export to more optimization solvers.
(4) Model integrity checking tools may be developed that alert on modeling errors, e.g. variable without proper modifier (input/external).
(5) Discretization enhancements and fixes: full support for ZOH, more discretization methods, multistep methods.
(6) More exception codes to support additional transcendental functions.
(7) More cost function time-wise options: continuous integrator, terminal cost (final value of the input signal), peak value of the input signal over the time horizon (∞ norm), or another norm (1, 2) over time of the input signal.
(8) More cost function signal-wise options: norm to take over the input elements (1, 2, or ∞) or largest element.
(9) General library rearrangement is required due to lessons learned from real world usage and better understanding of performance bottlenecks and usage scenarios. This rearrangement will simplify feature additions and cooperative development.
(10) Development of regression tests to facilitate further development.
(11) Support of advanced MPC techniques, such as stochastic MPC, requires special modules to convert a model to an optimization problem with chance constraints, etc.

REFERENCES

Åkesson, J. (2008). *Optimica An Extension of Modelica Supporting Dynamic Optimization*, 57–66. Modelica Association. URL `https://www.modelica.org/events/modelica2008/Proceedings/sessions/session1b3.pdf`.

Åkesson, J., Årzén, K.E., Gäfvert, M., Bergdahl, T., and Tummescheit, H. (2010). Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problem. *Computers and Chemical Engineering*, 34(11), 1737–1749.

Bisschop, J. (2006). *AIMMS Optimization Modeling*. LULU PR.

Cagienard, R., Grieder, P., Kerrigan, E., and Morari, M. (2007). Move blocking strategies in receding horizon control. *Journal of Process Control*, 17(6), 563 – 570.

Fourer, R., Gay, D., and Kernighan, B. (2003). *AMPL: a modeling language for mathematical programming*. Thomson/Brooks/Cole.

Fritzson, P. and Engelson, V. (1998). Modelica a unified object-oriented language for system modeling and simulation. In E. Jul (ed.), *ECOOP98 Object-Oriented Programming*, volume 1445 of *Lecture Notes in Computer Science*, 67–90. Springer Berlin / Heidelberg. URL `http://dx.doi.org/10.1007/BFb0054087`. 10.1007/BFb0054087.

Kallrath, J. (2004). *Modeling Languages in Mathematical Optimization*. Applied Optimization. Kluwer Academic Publishers.

Rosenthal, R. (2008). *GAMS: A User's Guide*. GAMS Development Corporation.

Wachter, A. and Biegler, L.T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106, 25–57. URL `http://dx.doi.org/10.1007/s10107-004-0559-y`. 10.1007/s10107-004-0559-y.