BLOM: Berkeley Library for Optimization Modeling

Sergey Vichik and Anthony Kelman

UC Berkeley
Department of Mechanical Engineering
Berkeley, CA

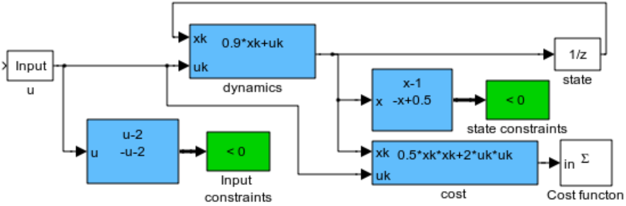
{sergv, kelman}@berkeley.edu

March, 2012

What is BLOM ?

- A language of modeling dynamical nonlinear systems for optimization problems, especially MPC
- Support for the following design phases:
  - Developing the model with an intuitive block diagram
  - Forward simulation and validation of the model
  - Automatic export of the optimization problem to a solver
- Developed to handle non trivial problems
  - C++ or Matlab code generation
  - Explicit evaluation of Jacobian and Hessian
  - Proven with problems of tens of thousands variables
- Eliminates manual problem coding, eases maintenance and assures that the same model used for optimization as for simulation

"Hello World" example



$$\min_{u_k, x_k} \sum_k 0.5x_k^2 + 2u_k^2$$

$$s.t. : -2 \leqslant u_k \leqslant 2 \,; 0.5 \leqslant x_k \leqslant 1 \,; x_{k+1} = 0.9x_k + u_k$$

- The Functional block holds expression of the form $\frac{f(x)}{g(x)}$
- The Constraint block marks variable as $\geqslant 0$ or $\leqslant 0$
- The continuous or discrete State block
- The Cost block accumulates cost variables over horizon
- The Input/External variable modifiers marks the control and the external variables

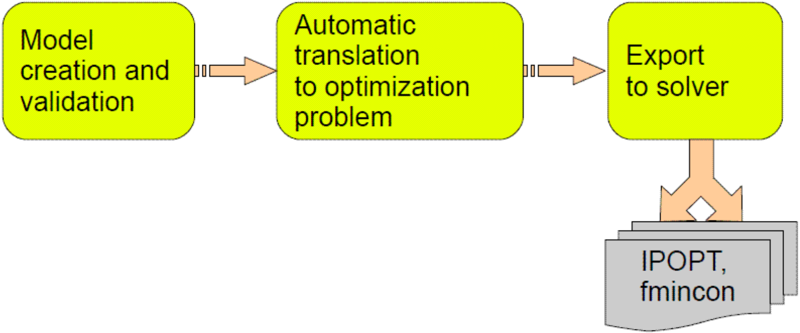
The functional block "Polyblock"

- Polynomial-like function described by two matrices
  - P defines exponents or special functions for each polynomial-like term
  - K specifies multiplier coefficients on those terms
- Has the form $f_i(x) = \sum_k K_{ik} \prod_j v(x_j, P_{kj})$, where $v(x, p) \in \{x^p, \exp(x), \log(x), \text{etc}\}$
- Example:

$$f(x) = 4x_1^3 + 0.2x_1^2 x_2^{0.7} - 0.8x_1 \exp(x_3) + 0.5 \log(x_2)$$

$$K = \begin{bmatrix} 4 & 0.2 & -0.8 & 0.5 \end{bmatrix}, \; P = \begin{bmatrix} 3 & 0 & 0 \\ 2 & 0.7 & 0 \\ 1 & 0 & \exp \\ 0 & \log & 0 \end{bmatrix}$$

BLOM work flow



- Create model using Simulink with BLOM library
- Run and compare the model to reference data
- Translate to optimization problem: **ExtractModel(steps,dt,'RK4');**
- Export the problem to a solver: e.g. **CreateIpoptCPP**

BLOM status and features

- Discrete and continuous models
- For continuous model, supports Euler, trapezoidal and RK4 discretization (easily expandable)
- Full vector support
- Model debugging features:
  - Color coded constraint violations
  - Polyblocks display the user defined function
  - User defined port labeling
- Export to IPOPT and fmincon solvers (more to come)
- Used in joint project with UTRC for large HVAC MPC problem (dynamical model with 430 states, typically $\sim$ 30k variables in solver)

Why BLOM is efficient

- Optimization algorithms require gradients, Jacobians, often Hessian
- Options for calculating gradients:
  1. Finite differences: suffers from curse of dimensionality, inaccuracy
  2. Automatic differentiation: used by AMPL and some other modeling languages, may require explicit code generation which grows with problem size
  3. Problem representation in structured form: BLOM approach to give closed-form Jacobian and Hessian, nonlinearities must fit polynomial-like structure or implemented special functions (general enough for most practical nonlinear problems)
- Fully sparse concise nonlinear problem formulation
  - Nonlinear functions represented implicitly: new variables introduced for function outputs, functional relationship enforced as equality constraint
  - Leads to more optimization variables, but very sparse Jacobian, Hessian
- Sparsity-preserving fixed-timestep discretizations of nonlinear continuous-time models

Why BLOM is fast

- Function evaluations with compiled problem-independent code
- C++ interface to compiled IPOPT solver results in standalone executable with low interfacing overhead
- High efficiency: with IPOPT the function (objective, Jacobian, Hessian) evaluation typically takes less than 10% of total solver time
- Solver time from milliseconds for small problems to minutes for sparse problems with tens of thousands of variables