D5.1.8 – Collocation-based NMPC algorithm in JModelica.org, based on the results from D5.1.1, D5.1.2 and D5.1.3

WP 5.1 Nonlinear Model Predictive Control
WP 5 Optimized system operation

MODRIO (11004)

Version 1.2

Date 11/11/2015

Authors

Toivo Henningsson — Modelon

Executive summary

This document describes how capabilities for Nonlinear Model Predictive Control (NMPC) have been implemented in the open source Modelica platform JModelica.org. A new tool chain for trajectory optimization in JModelica.org has been jointly developed between the present deliverable and D3.1.6, providing significant benefits in terms of flexibility and efficiency over the previous tool chain. NMPC functionality has then been implemented as one application on top of this tool chain. The developed functionality has also been tested and improved in a number of application scenarios, including optimization of thermal power plants in the demonstrators D8.1.4 and D8.1.9.

1. Introduction

Model Predictive Control (MPC) is an advanced process control methodology that has gained increasing traction over the last three decades. Its popularity in industry stems from its ability to handle multiple-input-multiple-output control problems in a straightforward manner, and to directly handle constraints on states and control signals. In MPC, control signals are determined by solving an optimization problem to optimize the evolution of the system over a time horizon of a given length into the future, with the current plant state as initial state. The control signal for the first sample is applied, and the optimization is then solved again for the next sample based on updated states. This setup naturally incorporates both handling of reference values (using cost terms) and constraints on the variables involved.

Linear MPC is by this time well understood and widely applied. Linear plant dynamics combined with convex cost functions yield convex optimization problems, for which there exist very robust solvers. But for some plants, linear dynamics is not a good enough approximation. Such cases necessitate the use of Nonlinear Model Predictive Control, NMPC. NMPC is much less developed than its linear counterpart. Since the optimization problem is no longer convex, convergence cannot always be guaranteed. In practice, it becomes much more important to find a good numerical formulation of the control problem at hand, and to have tools available to help debug convergence problems. Since NMPC is aimed to be applied online, robustness is very important.

This report describes the developments for NMPC that have been carried out within the open source platform JModelica.org for Modelica-based optimization, simulation and analysis of complex dynamic systems. We first describe the actual NMPC implementation, and then the underlying trajectory optimization tool chain. We also discuss additional application scenarios where the tool chain and NMPC functionality have been tested and improved.

2. NMPC

The NMPC support in JModelica.org was implemented in [7]. The main functionality is provided by the `MPC` class. The code is implemented in Python, which is the scripting language used with JModelica.org.

To perform NMPC, the user first creates a Modelica model to describe the plant. Optimization information in the form of constraints and cost function is added using Optimica [1], a language extension of Modelica that is supported by JModelica.org.

The user then loads the optimization problem, and creates an `MPC` instance based on the model and MPC parameters such as horizon length and sampling time. Once the `MPC` instance is created, it can be used to optimize given the current states, and the resulting control signals can be retrieved.

It is possible to implement an NMPC controller directly based on the trajectory optimization tool chain in JModelica.org, but the `MPC` class simplifies the procedure by taking care of a lot of the administration involved, and exploits the structure of repeatedly solving similar optimization problems to make the solution time significantly faster than what would be achieved by solving independent optimization problems for each sample.

The `MPC` class also provides a number of features that are specifically useful for Nonlinear Model Predictive Control, including

- Warm starting. This includes
  - Reusing the same discretization for optimization in each sample. This typically saves a significant part of the computation time. To realize this, the discretization of the original trajectory optimization problem is parameterized by a number of parameters such as initial states, which can be changed after discretization.
  - Using the last successful optimization to create an initial guess for the optimization in the next sample. This reduces computation time and can also increase robustness, since it is easier for a Nonlinear Program (NLP) solver to converge if it starts closer to the optimum.
    The NLP solver that is typically used, IPOPT [2], is a primal-dual method, which

means that it converges simultaneously the primal variables (the actual decision variables in the optimization problem), and the dual variables (Lagrange multipliers for the problem's constraints). To achieve a warm starting effect, initial guesses for both are based on the last successful optimization. The initial guess for the primal variables is also adjusted by shifting it one sample in time.

- Support for specifying bounds and penalties on changes in the control signal, which are popular and useful in Model Predictive Control. Control signals are usually taken as piecewise constant over each sample, so it is not possible to formulate these as bounds and costs on the control signals' derivatives.
- Softening of variable bounds. An issue in Model Predictive Control is how to avoid that the optimization problem becomes infeasible. Even if it is impossible to meet all demands that a solution should ideally satisfy, a best effort solution is most often much preferred to none at all. Infeasibility stems from conflicting constraints. By replacing variable bounds by penalties for exceeding the bounds, this problem is alleviated, while still producing the same solution as with strict bounds as long as the penalty is above a certain threshold.
- Fallback to the last successful optimization. This feature exploits the fact that each NMPC optimization results in a control signal trajectory over the whole horizon. If the optimization fails in a given step, the control signal for the appropriate time point is used from the last successful optimization.

3. New Optimization Tool Chain

The work on the new trajectory optimization tool chain that forms the basis of the NMPC implementation as well as the developments in D3.1.6 was initiated in [6]. It has since been continuously improved and tested; some of the work has been performed in [4], [5], [7], and [8].

Trajectory optimization refers to optimization problems that are based on dynamic models. The unknowns are trajectories for the models' variables over a given time horizon and model parameters (the initial/final times may be free parameters), while the cost function can contain terms that are integrated over the optimization horizon as well as terms that depend on variables at specific time points, often the initial and final time. In addition to the constrains provided by the model's dynamics, additional equality and inequality constraints can be added, that can hold for all times or based on variable values at specific time points.

The new trajectory optimization tool chain (as well as the old one) in JModelica.org is based on the Modelica language extension Optimica [1]. This is a small extension that allows specifying optimization related information such as cost functions, constraints, initial guesses and free parameters.

To solve a trajectory optimization problem, it is first discretized using a method called collocation. The time horizon is divided into a number of elements, and the trajectories of model variables are approximated as piecewise polynomial, with one polynomial piece per element. A number of collocation points are chosen on each element, where the equations of the original continuous time model are required to hold. Additional constraints are added, e.g. to relate the values of states to the corresponding state derivatives. The result of the discretization is a Nonlinear Program (NLP), which is passed to a nonlinear solver such as IPOPT [2]. The results are then interpreted in the form of solution trajectories for the original continuous time system.

To solve an NLP using e.g. IPOPT, the NLP solver must be able to evaluate the NLP's constraints and cost function, but also their first and second derivatives. To provide these derivatives, and to be able to carry out the discretization as a symbolic manipulation of the original continuous time model, the new tool chain uses the optimization framework CasADi [3]. CasADi provides a framework for building symbolic representations of expressions and functions, which can be combined, differentiated an arbitrary number of times, and evaluated.

The new tool chain imports flat Modelica and Optimica models and exposes them in a format familiar to Modelica users. Models contain variables with attributes, equations, functions, etc. All expressions and functions are represented using CasADi (including binding expressions for attributes and parameters) and can be extracted by the user to perform further symbolic processing and numeric evaluation.

The new collocation code discretizes the optimization problem by manipulating these CasADi expressions. CasADi then provides for efficient evaluation of the resulting expressions and the required derivatives, and keeps track of and exploits the sparsity of the NLP, which is required to handle any trajectory optimization problem of even modest size with collocation.

The symbolic representation of the optimization problem is preserved all the way to the invocation of the NLP solver, which provides for a great deal of flexibility when extending the optimization tool chain, and for the advanced user. It is e.g. possible to load an optimization problem and manipulate it in various ways before applying collocation, to load a model and add optimization information in a script, or to load a model and then write your own CasADi-based solver. The latter is done e.g. in D5.2.1, which implements its own discretization scheme on top of the tool chain's CasADi based model representation.

4. Other Developments

In addition to the main development of trajectory optimization and NMPC functionality, the two have been tested in a number of different applications, some of which have also contributed back to the development of the core tool chain and NMPC code.

In [4], the tool chain was used to generate C code from the NLP, which was then used with a dedicated solver for large scale parallel dynamic optimization. In the process, the collocation algorithm in JModelica.org was improved to allow to generate a more compact and efficient representation of the resulting NLP, benefitting both the case when C code is generated from the NLP, and when it is evaluated directly in its symbolic form.

In [5], a prototype was implemented to export Modelica and Optimica models from the JModelica.org compiler in XML format, and import them into the new trajectory optimization tool chain, according to the XML format developed in D5.1.3. The experience made it possible to identify some improvements to the XML format, which were fed back to the specification.

In [8], the `MPC` class was successfully applied to for real time control of two processes at a lab scale, and a number of issues were identified and solved related to using the code in a real time environment with connection to a physical process.

In [9], the new optimization tool chain was applied for startup optimization of one of Vattenfall's thermal power plants, with stress constraints to maximize the lifetime of the components even though frequent startups might be necessary. In [10], the `MPC` class was applied to startup of a thermal power plant model from Vattenfall, achieving a substantial improvement in startup time compared to the reference solution.

For further applications of the optimization tool chain, see D3.1.6.

5. Conclusion

Functionality for Nonlinear Model Predictive Control (NMPC) has been implemented in the open source platform JModelica.org for Modelica-based optimization, simulation and analysis of complex dynamic systems. The development has been divided into a general tool chain for dynamic optimization, and NMPC functionality built on top. This means that the optimization tool chain can be reused for other optimization applications, which has been done e.g. in D3.1.6 for Moving Horizon Estimation (MHE) and grey box identification. The MHE functionality implemented in D3.1.6 can be directly coupled to the NMPC functionality developed in the present deliverable. Various additional application concepts have been explored based on the core NMPC and trajectory optimization functionality, demonstrating its usefulness and flexibility, and feeding back requirements and improvements to the core.

6. References

[1] Åkesson, Johan. "Optimica—An Extension of Modelica Supporting Dynamic Optimization",

Proc. 6th International Modelica Conference 2008. Modelica Association. March 2008. Available at http://www.control.lth.se/Publication/ake08mod08.html

[2] 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), pp. 25-57, 2006. Available at http://www.optimization-online.org/DB_HTML/2004/03/836.html

[3] Andersson, Joel. "A General-Purpose Software Framework for Dynamic Optimization", Arenberg Doctoral School, KU Leuven, October 2013. Available at https://lirias.kuleuven.be/bitstream/123456789/418048/1/thesis_final2.pdf

[4] Rodriguez, Jose, "Large-scale dynamic optimization using code generation and parallel computing", Master's Thesis, KTH, School of Engineering Sciences (SCI), Mathematics (Dept.), Numerical Analysis, NA. August 2014. Available at http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-150443

[5] Landin, Niklas, "XML export and import of Modelica models", Master's Thesis, Chalmers Univeristy, Department of Computer Science and Engineering. June 2014.

[6] Lennernäs, Björn, "A CasADi Based Toolchain For JModelica.org", Master's Thesis, Lund Univeristy, Department of Automatic Control. June 2013.

[7] Axelsson, Magdalena, "Nonlinear Model Predictive Control in JModelica.org", Master's Thesis, Lund Univeristy, Department of Automatic Control. March 2015.

[8] Ekström, Sebastian, "Real Time Model Predictive Control in JModelica.org", Master's Thesis, Lund Univeristy, Department of Automatic Control. June 2015.

[9] Runvik, Håkan, "Modelling and start-up optimization of a coal-fired power plant", Master's Thesis, Lund Univeristy, Department of Automatic Control. June 2014.

[10] Thelander Andrén, Marcus and Wedding, Christoffer. "Development of a Solution for Start-up Optimization of a Thermal Power Plant", Master's Thesis, Lund Univeristy, Department of Automatic Control. June 2015.