A Toolchain for Solving Dynamic Optimization Problems Using Symbolic and Parallel Computing

Evgeny Lazutkin Siegbert Hopfgarten Abebe Geletu Pu Li

Group Simulation and Optimal Processes, Institute for Automation and Systems Engineering, Technische Universität Ilmenau, P.O. Box 10 05 65, 98684 Ilmenau, Germany.
{evgeny.lazutkin,siegbert.hopfgarten,abebe.geletu,pu.li}@tu-ilmenau.de

Abstract

Significant progresses in developing approaches to dynamic optimization have been made. However, its practical implementation poses a difficult task and its real-time application such as in nonlinear model predictive control (NMPC) remains challenging. A toolchain is developed in this work to relieve the implementation burden and, meanwhile, to speed up the computations for solving the dynamic optimization problem. To achieve these targets, symbolic computing is utilized for calculating the first and second order sensitivities on the one hand and parallel computing is used for separately accomplishing the computations for the individual time intervals on the other hand. Two optimal control problems are solved to demonstrate the efficiency of the developed toolchain which solves one of the problems with approximately 25,000 variables within a reasonable CPU time.
Keywords: nonlinear optimization, combined multiple shooting and collocation, symbolic manipulation, parallel computing, satellite problem, combined cycle power plant

1 Introduction

Over the last decades nonlinear model predictive control (NMPC) has been increasingly popular for the control of complex systems (Mayne, 2014). To carry out NMPC, the first step is to formulate a nonlinear optimal control problem. By using a discretization scheme over a prediction horizon, it is then transformed into a constrained nonlinear programming (NLP) problem. Finally, the realization of NMPC is made by repeatedly solving this problem online with an NLP solver which requires appropriate function values and gradients. Although many theoretical progresses on NMPC have been achieved, its implementation for real-life applications is certainly not trivial. Therefore, a toolchain is developed in this work based on open-source software tools to relieve the burdens in the implementation of NMPC.

A schematic description of implementing NMPC is shown in Fig. 1. Based on the current process state $x(k)$ obtained through the state observer or measurement, resp., the optimal control problem is solved in the optimizer in each sample time. The resulting optimal control strategy in the first interval $u(k)$ of the moving horizon is then realized through the local control system. Therefore, an essential limitation of applying NMPC is due to its long computation time taken to solve the NLP problem for each sample time, especially for the control of fast systems (Wang and Boyd, 2010). In general, the computation time should be much less than the sample time of the NMPC scheme (Schäfer et al., 2007). Although powerful methods are available, e.g. multiple-shooting (Houska et al., 2011; Kirches et al., 2012) and collocation on finite elements (Biegler et al., 2002; Zavala et al., 2008; Word et al., 2014) with simultaneous characteristics, control parametrization (Balsa-Canto et al., 2000; Barz et al., 2012) with sequential characteristics, and quasi-sequential technique, (Hong et al., 2006; Bartl et al., 2011), the computation speed is not swift enough for very fast systems such as mechanical, electrical and mechatronic systems. Therefore, it is highly desired to further enhance the computation efficiency for solving nonlinear dynamic optimization problems.

The combined multiple-shooting with collocation (CMSC) method (Tamimi and Li, 2010) and the modified multiple-shooting and collocation (MCMSC) method (Lazutkin et al., 2014) are proved to be highly efficient. The efficiency of this method is considerably improved in this work with the following targets:

- to reduce the computation time by using symbolic methods for calculating gradients, Jacobians, and Hessians,
- to further accelerate the computation by using parallel computing facilities, especially for real-time applications.

To achieve these aims, this work develops a toolchain as described in subsequent sections. In section 2 the problem will be formulated. Section 3 illustrates the interior-point solution method with symbolic computations of first- and second-order derivatives. The toolchain, its components, functionality, and some code examples are presented in section 4. The efficiency of the toolchain is demonstrated in section 5 by applying it to a satellite control and a large-scale dynamic optimization of a combined cycle power plant. Conclusions of the paper are given in section 6.

2 Problem description

The nonlinear optimal control problem (NOCP) reads

$$\begin{aligned}
&\min_{\boldsymbol{u}(t)} \left\{ J = M\left(\boldsymbol{x}(t_f), t_f\right) + \int_{t_0}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \right\} \\
&\text{s. t. } \boldsymbol{f}(\dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{u}(t), t) = \boldsymbol{0}, \quad t_0 \le t \le t_f, \\
&\quad \boldsymbol{x}(t_0) = x_0, \ \boldsymbol{x}(t_f) \text{ fixed or free}, \\
&\quad \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \le \boldsymbol{0}, \\
&\quad \boldsymbol{x}_{min} \le \boldsymbol{x}(t) \le \boldsymbol{x}_{max}, \\
&\quad \boldsymbol{u}_{min} \le \boldsymbol{u}(t) \le \boldsymbol{u}_{max},
\end{aligned} \tag{1}$$

with $t \in [t_0, t_f]$ - time, t_0, t_f - initial, final time, $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$ - state variable vector, $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$ - control variable vector, $\boldsymbol{x}_0$ - initial state vector, $\boldsymbol{x}_f$ - final state vector, $\boldsymbol{f} \in \mathbb{R}^{n_x+n_u} \to \mathbb{R}^{n_x}$ - implicit differential equation, J - performance index with Mayer and Lagrange term $M : \mathbb{R}^{n_x+1} \to \mathbb{R}$ and $L : \mathbb{R}^{n_x+n_u} \to \mathbb{R}$, resp., belonging to corresponding function spaces, $\boldsymbol{g}$ - additional equality and/or inequality constraints, $\boldsymbol{x}_{min}, \boldsymbol{x}_{max}, \boldsymbol{u}_{min}, \boldsymbol{u}_{max}$, - componentwise lower and upper bounds for states $\boldsymbol{x}(t)$ and controls $\boldsymbol{u}(t)$, resp.

Envisaging the application of the modified combined multiple shooting and collocation (MCMSC) method, a transformation of the infinite-dimensional NOCP (1) to a finite-dimensional nonlinear programming (NLP) problem is needed. Due to the multiple-shooting technique a division of the whole time horizon $[t_0, t_f]$ into N time intervals, so called shooting intervals has to be performed. The controls are assumed to be constant in each shooting interval and are parametrized, i.e. the control vector is composed as $\boldsymbol{V} = [\boldsymbol{v}_0 \ \boldsymbol{v}_1 \ \dots \ \boldsymbol{v}_{N-1}]^T$, $n_u = n_v$. The states are also discretized and parametrized at the shooting interval boundaries, i.e. the vector $\boldsymbol{X}^p = [\boldsymbol{x}_{p,0} \ \boldsymbol{x}_{p,1} \ \dots \ \boldsymbol{x}_{p,N}]$ is constructed and equality constraints for continuity reasons are taken into account. All other restrictions are correspondingly discretized.

This leads to the NLP notation

$$\begin{aligned}
&\min_{\boldsymbol{X}^p, \boldsymbol{V}} \left\{ M\left(\boldsymbol{x}_{p,N}\right) + \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} L(\boldsymbol{x}(t), \boldsymbol{v}_i)dt \right\} \\
&\text{s. t. } \ \boldsymbol{x}_{p,i+1} = \boldsymbol{x}\left(t_{i+1}; \boldsymbol{x}_{p,i}, \boldsymbol{v}_i\right), \ i = 0, \dots, N-1, \\
&\boldsymbol{x}_{p,0} = \boldsymbol{x}_0, \\
&\bar{\boldsymbol{g}}(\boldsymbol{X}^p, \boldsymbol{V}) \le \boldsymbol{0}, \\
&\bar{\boldsymbol{x}}_{min} \le \boldsymbol{X}^p \le \bar{\boldsymbol{x}}_{max} \\
&\bar{\boldsymbol{u}}_{min} \le \boldsymbol{V} \le \bar{\boldsymbol{u}}_{max} \ .
\end{aligned} \tag{2}$$

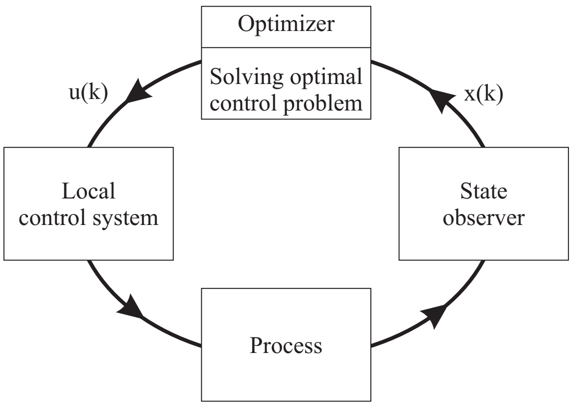

Figure 1. Nonlinear model predictive control (NMPC) scheme

Due to the combined character of the approach to be applied the model equations are not directly integrated in the NLP formulation (2) but solved to obtain the state variables $\boldsymbol{x}_{p,i+1}$ by a collocation scheme. For detailed description of the NOCP and its transformation to an NLP refer to (Tamimi and Li, 2010, 2009; Lazutkin et al., 2014).

3 Solution method

3.1 Solution of the resulting NLP problem

For simplicity reasons the NLP problem (2) is rewritten in the compact form

$$\begin{aligned}
&\min_{\boldsymbol{\omega}} \{J(\boldsymbol{\omega})\} \\
&\text{s. t. } \ \boldsymbol{E}(\boldsymbol{\omega}) = \boldsymbol{0}, \\
&\qquad \boldsymbol{S}(\boldsymbol{\omega}) \le \boldsymbol{0},
\end{aligned} \tag{3}$$

where $\boldsymbol{\omega}$ contains all optimization variables, $\boldsymbol{E}$ all equality, and $\boldsymbol{S}$ all inequality constraints.

The NLP (4) can be solved using an interior-point optimization solver (e. g. Ipopt (Wächter and Biegler, 2006)). Hence, the barrier function formulation of the NLP reads

$$\begin{aligned}
&\min_{\omega, z} \left\{ J(\boldsymbol{\omega}) - \mu \sum_{j=1}^{n_S} \ln(z_j) \right\} \\
&\text{s. t. } \quad \boldsymbol{E}(\boldsymbol{\omega}) = \boldsymbol{0}, \\
&\qquad \boldsymbol{S}(\boldsymbol{\omega}) + \boldsymbol{z} = \boldsymbol{0},
\end{aligned} \tag{4}$$

with a slack variable $\boldsymbol{z}$ and the corresponding Lagrange function

$$\begin{aligned}
\mathscr{L}(\boldsymbol{\omega}, \boldsymbol{z}, \boldsymbol{\lambda}) \ = \ & J(\boldsymbol{\omega}) - \mu \sum_{j=1}^{n_S} \ln(z_j) + (\boldsymbol{\lambda}^E)^T \boldsymbol{E}(\boldsymbol{\omega}) \\
& + (\boldsymbol{\lambda}^S)^T (\boldsymbol{S}(\boldsymbol{\omega}) + \boldsymbol{z})
\end{aligned} \tag{5}$$

for a fixed value of the barrier parameter μ. The vector $\boldsymbol{\lambda}^T = [(\boldsymbol{\lambda}^E)^T, (\boldsymbol{\lambda}^S)^T] \in \mathbb{R}^{n_E+n_S}$ contains multipliers associated with the $n_E + n_S$ equality constraints. The iteration scheme of the interior-point algorithm is

$$\begin{aligned} \boldsymbol{\omega}_{l+1} &= \boldsymbol{\omega}_l + \alpha_l \cdot \Delta\boldsymbol{\omega}_l,\ z_{l+1} = \boldsymbol{\omega}_l + \alpha_l \cdot \Delta z_l, \\ \boldsymbol{\lambda}_{l+1} &= \boldsymbol{\lambda}_l + \alpha_l \cdot \Delta\boldsymbol{\lambda}_l,\ l = 0,1,\ldots \end{aligned} \tag{6}$$

The different search directions $\Delta\boldsymbol{\omega}_l$, Δz_l, $\Delta\boldsymbol{\lambda}_l^E$, $\Delta\boldsymbol{\lambda}_l^S$ are obtained applying the Newton method to the KKT optimality conditions of problem (4). Let $\boldsymbol{Z} = \operatorname{diag}(z)$ and $\boldsymbol{e}^T = (1,\ldots,1) \in \mathbb{R}^{n_S}$. Thus, the following system needs to be solved at each iteration step

$$\boldsymbol{K} \cdot \begin{bmatrix} \Delta\boldsymbol{\omega}_l \\ \Delta z_l \\ \Delta\boldsymbol{\lambda}_l^E \\ \Delta\boldsymbol{\lambda}_l^S \end{bmatrix} = - \begin{bmatrix} \nabla_{\boldsymbol{\omega}}\mathscr{L}(\boldsymbol{\omega}_l, z_l, \boldsymbol{\lambda}_l) \\ -\mu \boldsymbol{e}^T \boldsymbol{Z}_l^{-1} + \boldsymbol{\lambda}_l^S \\ \boldsymbol{E}(\boldsymbol{\omega}_l) \\ \boldsymbol{S}(\boldsymbol{\omega}_l) + z_l \end{bmatrix}, \text{ where} \tag{7}$$

$$\boldsymbol{K} = \begin{bmatrix} \nabla_{\boldsymbol{\omega\omega}}\mathscr{L}(\boldsymbol{\omega}_l, z_l, \boldsymbol{\lambda}_l) & \boldsymbol{0} & \nabla\boldsymbol{E}(\boldsymbol{\omega}_l) & \nabla\boldsymbol{S}(\boldsymbol{\omega}_l) \\ \boldsymbol{0} & -\mu\boldsymbol{Z}_l^{-2} & \boldsymbol{0} & \boldsymbol{I}_{n_S} \\ \nabla\boldsymbol{E}(\boldsymbol{\omega}_l)^T & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \nabla\boldsymbol{S}(\boldsymbol{\omega}_l)^T & I_{n_S} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}$$

According to (7) the gradient $\nabla J(\omega_l)$, Jacobians $\nabla\boldsymbol{E}(\boldsymbol{\omega}_l)$, $\nabla\boldsymbol{S}(\boldsymbol{\omega}_l)$, and Hessian matrices $\nabla^2 J(\boldsymbol{\omega}_l)$, $\nabla^2\boldsymbol{E}_i(\boldsymbol{\omega}_l)$, $i = 1,\ldots,n_E$, $\nabla^2\boldsymbol{S}_j(\boldsymbol{\omega}_l)$, $j = 1,\ldots,n_S$ are required and made available either analytically or approximately to the optimization solver. In the subsequent discussions the expression

$$\begin{aligned} \boldsymbol{H} &= \nabla_{\boldsymbol{\omega\omega}}\mathscr{L}(\boldsymbol{\omega}_l, z_l, \boldsymbol{\lambda}_l) \\ &= \nabla^2 J(\boldsymbol{\omega}) + \sum_{i=1}^{n_E} \boldsymbol{\lambda}_i^E \nabla^2\boldsymbol{E}_i(\boldsymbol{\omega}) + \sum_{j=1}^{n_S} \boldsymbol{\lambda}_j^S \nabla^2\boldsymbol{S}_j(\boldsymbol{\omega}) \end{aligned} \tag{8}$$

will be referred as the analytic Hessian (AH).

3.2 First- and second-order sensitivities

A collocation method is used in each shooting interval within the framework of the MCMSC approach. The states are approximated inside each shooting interval by a linear combination of the Lagrange polynomials (9) using a shifted Legendre collocation scheme,

$$\widehat{x}(t) = \sum_{j=1}^{n_c} \left(\prod_{k=1,k\neq j}^{n_c} \frac{t - t_k}{t_j - t_k} \right) \cdot x_j^c, \tag{9}$$

with $\widehat{x}(t)$ - a polynomial approximation of a single state variable, x_j^c - the unknown collocation-coefficient at the j-th collocation point, $\{t_1,\ldots,t_{n_c}\}$ - collocation points in the shooting interval $[t_q, t_{q+1}]$. A shifted scheme means, that the last collocation point t_{n_c} is shifted to the right interval border – a necessity for continuity reasons – and the other ones are also shifted accordingly.

The derivative of the collocation polynomial reads

$$\begin{aligned} \frac{d\widehat{x}(t)}{dt} &= \sum_{j=1}^{n_c} \left(\frac{dl_j(t)}{dt} \right) x_j^c \\ \text{with } l_j(t) &= \prod_{k=1,k\neq j}^{n_c} \frac{t - t_k}{t_j - t_k} \end{aligned} \tag{10}$$

The parametrized states can be put into a vector $\boldsymbol{X}^{p,q}$ and the controls into a vector $\boldsymbol{V}^q$, where $q = 1,2,\ldots,N$. The components of the vectors $\boldsymbol{X}^{p,q}$ and $\boldsymbol{V}^q$ are included in the decision variables $\boldsymbol{X}^p$ and $\boldsymbol{V}$, respectively, in the NLP formulation. Furthermore, the vector $\boldsymbol{X}^{c,q}$ represents all collocation coefficients in the shooting interval q. Hence, the discretized nonlinear differential equation system results in the nonlinear algebraic equation system

$$\begin{aligned} \boldsymbol{G}^q &= \dot{\boldsymbol{W}} \cdot \boldsymbol{X}^{c,q} + \dot{\boldsymbol{W}}_0 \cdot \boldsymbol{X}^{p,q} \\ &\quad - \frac{(t_f - t_0)}{N} \cdot \boldsymbol{F}(\boldsymbol{X}^{c,q}, \boldsymbol{V}^q) = \boldsymbol{0} \end{aligned} \tag{11}$$

with $q = 1,2,\ldots,N$, q - index of shooting interval, N - number of shooting intervals, $\boldsymbol{F}$ - discretized $\boldsymbol{f}$, $\dot{\boldsymbol{W}}$ and $\dot{\boldsymbol{W}}_0$ - derivative matrices of the Lagrange polynomials.

At each iteration step of the optimization procedure, for given values $\boldsymbol{X}^{p,q}$ and $\boldsymbol{V}^q$, (11) is solved by a Newton method. The results will be $\boldsymbol{X}^{c,q}$ as well as the first- and second-order sensitivities These results are utilized in the SQP solver for calculation of the functions $\boldsymbol{E}$, $\boldsymbol{S}$, the Jacobians $\nabla\boldsymbol{E}$, $\nabla\boldsymbol{S}$, and the Hessian $\boldsymbol{H}$.

Neglecting the shooting interval index q and writing (11) in a compressed form delivers

$$\boldsymbol{G}(\boldsymbol{X}^c(\boldsymbol{X}^p, \boldsymbol{V}), \boldsymbol{X}^p, \boldsymbol{V}) = \boldsymbol{0}. \tag{12}$$

To obtain the first-order sensitivities, Eq. (12) has implicitely to be differentiated and provides

$$\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{X}^c} \frac{\partial \boldsymbol{X}^c}{\partial \left[\boldsymbol{X}^{pT} \boldsymbol{V}^T\right]^T} - \frac{\partial \boldsymbol{G}}{\partial \left[\boldsymbol{X}^{pT} \boldsymbol{V}^T\right]^T} = \boldsymbol{0}. \tag{13}$$

Eq. (13) represents a linear equation system. Typically, $\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{X}^c}$ and $\frac{\partial \boldsymbol{G}}{\partial [\boldsymbol{X}^{pT} \boldsymbol{V}^T]^T}$ have sparsity structures that can be exploited in determination of $\frac{\partial \boldsymbol{X}^c}{\partial [\boldsymbol{X}^{pT} \boldsymbol{V}^T]^T}$.

Using (13), analytic expressions are derived in order to calculate the second-order sensitivities. Hence, this equation is re-written here in the following compact form

$$\boldsymbol{\Phi}(\boldsymbol{X}^c(\boldsymbol{X}^p, \boldsymbol{V}), \boldsymbol{X}^p, \boldsymbol{V}, \frac{\partial \boldsymbol{X}^c}{\partial \left[\boldsymbol{X}^{pT} \boldsymbol{V}^T\right]^T}(\boldsymbol{X}^p, \boldsymbol{V})) = \boldsymbol{0}. \tag{14}$$

The derivative $\frac{\partial \boldsymbol{X}^c}{\partial [\boldsymbol{X}^{pT} \boldsymbol{V}^T]^T}$ indicates the dependencies of the first-order sensitivities on the decision variables $\boldsymbol{X}^p$ and $\boldsymbol{V}$. Applying the differentiation operator $\frac{\partial}{\partial [\boldsymbol{X}^{pT} \boldsymbol{V}^T]^T}$ to (14) the equations for second-order sensitivities will be available in matrix form and can be computed using LU decomposition.

4 Toolchain

As mentioned above in order to offer a comfortable and user-friendly way of modeling and optimizing physical systems, to unburden the user from automatable tasks like gradients and sensitivities calculations, to accelerate the implementation time as well as the computation time for solving the optimization problem, the implementation of the MCMSC method within an open-source toolchain is proposed. This toolchain consists of amongst others physically-based object-oriented modeling, using the modeling language Modelica with the extension Optimica, and the large scale nonlinear optimization solver Ipopt.

4.1 Components

Compared with block-oriented modeling, the object-oriented modeling approach provides a more comfortable and flexible alternative for physical systems. As a result this work uses the modeling language Modelica (Fritzson, 2014). Not only for simulation purposes but also for the formulation of different optimization tasks the platform JModelica.org is utilized. Besides the standard Modelica features JModelica.org contains the extension Optimica (Åkesson, 2008) including the possibility to formulate optimal control problems and other optimization tasks (Åkesson et al., 2010).

One essential tool utilized in many respects is the automatic differentiation tool CasADi (Andersson et al., 2011, 2012a,b). Hence, it is applied for the calculation of first- and second-order derivatives, symbolic manipulations of the objective and the constraints.

The standard multi-processing Python module (Hellmann, 2011) is a next module within the toolchain to perform parallel computations.

After adopting the optimization model to the MCMSC framework, Ipopt (Wächter and Biegler, 2006) is responsible for the solution of large scale nonlinear optimization problems. The interoperation of the toolchain within the optimizer (see Fig. 1) is illustrated in Fig. 2.

4.2 Functionality

After establishing an optimization model and implementing it by means of Modelica and the extension Optimica, the JModelica.org compiler transforms the model into a symbolic one. From the transformed model, i.e., model equations, variables, etc. are accessible by the Python scripting language.

The proposed MCMSC approach belongs to the category of quasi-sequential methods, i.e. the interior-point optimizer Ipopt solves in every iteration the state equations (11) and calculates the sensitivities (13) for given parametrized states and controls of each shooting interval. If the advantageous feature of an analytical Hessian symbolically calculated by CasADi is used, also in every iteration only the numerical values of the variables mattering have to be updated. It is also possible that numerical Hessians approximated through a BFGS formula which usually incurs more computational effort.

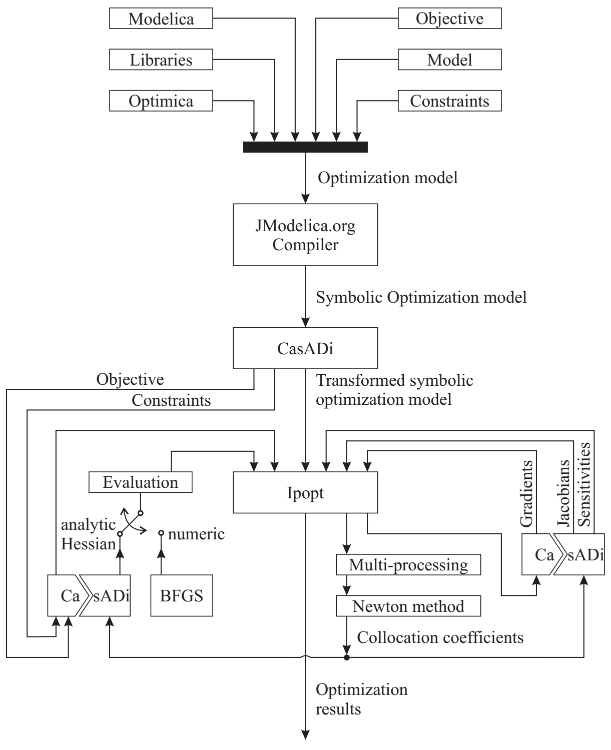

Figure 2. Parallelized MCMSC toolchain

The MCMSC framework consists of three main parts, i.e. the optimizer, the calculation of state trajectories by means of a Newton method, and the sensitivity computations. Ideally, both the Newton method and the sensitivity calculation are recommended to be executed in parallel in each shooting interval. Depending on the computer architecture (multi-core, etc.) the user can define the number of processes. On the one hand, one gains computing time improvements via parallelization. On the other hand, the communication effort increases, the more parallel threads occur. There is a maximum speed up depending on the size of the optimization problem and the computer architecture.

Concerning the first-order sensitivities computations from (13), an LU decomposition and the direct solver for sparse matrices CSparse (Davis, 2006) are applied and interfaced to CasADi. The second-order derivatives, if used, are transformed to a linear equation system and also solved by CasADi. This symbolic tool is furthermore responsible for the generation of the Jacobians, the gradients of the objective function, and the symbolic manipulations of the objective functions and the constraints.

The entire approach of the parallelized MCMSC method is realized in the Python scripting language using standard multi-processing module without any additional software packages.

Table 1. Classes and functions

(C)lass/(F)unction name	Functionality
`loading(object)`	auxiliary functions to prepare the optimization problem for Ipopt
`initialization`	loads options and uses JModelica.org to extract the optimization problem from Modelica/Optimica.
`extract`	prepares the optimization problem
`define_collocation`	constructs Lagrange polynomials and derivative matrices
`construct_vars`	creates vectors of discretization
`discretize`	implicit DAE discretization
`create_solvers`	creates for each interval the Newton solver and solvers for first-order derivatives
`interval_simulation`	simulates the DAE system for given parameters
`constr_vec`	constraints vector for Ipopt
`jacobian_of_constraint_vector`	Jacobian for Ipopt
`create_cost`	objective for Ipopt
`prep_ipopt`	required information for Ipopt, e.g. number of non-zeros in Jacobians, boundaries
`prep_sys_for_multiproc`	divide the system according to the number of cores
`optimize`	solve the optimization problem by Ipopt
`exact_hessian`	construct exact Hessian and corresponding linear solvers
`scaling`	scale the problem

4.3 Source code examples

Different classes and functions shown in Tab. 1 are realized dedicated to certain purposes. In particular, there are several important issues in the MCMSC toolchain. A brief description is given below with some source code fragments. Using JModelica.org the formulated optimization problem can be easily transferred for further manipulations using the function `transfer_optimization_problem`, which has two attributes: `name` is an optimization problem file and `file_path` is a path to this file. Let `OP` be the symbolic representation of the dynamic optimization problem, which includes all required information. To see the list of functions and methods for the `OP` variable, users have to refer to the JModelica.org source code files.

The proposed toolchain requires a lot of functions from CasADi. For simplification, functionality of this software will be made completely available in the toolchain by importing all CasADi classes.

The first essential aspect is to get information about the declared variables (differential and algebraic states, controls, parameters) in the optimization problem formulation. The following has been implemented by the developer:

```
# Differential states
DIFF = OP.getVariables( ...
         OP.DIFFERENTIATED)
# Algebraic states
ALG = OP.getVariables( ...
        OP.REAL_ALGEBRAIC)
# Derivatives
DER = OP.getVariables(OP.DERIVATIVE)
# Controls
INPUT = OP.getVariables(OP.REAL_INPUT)
# Parameters
P_I = OP.getVariables( ...
        OP.REAL_PARAMETER_INDEPENDENT)
P_D = OP.getVariables( ...
        OP.REAL_PARAMETER_DEPENDENT)
```

In order to extract model dynamics, the `OP` variable has specific function to get DAEs, which returns a residual between left and right hand sides of the equations.

```
DAE = OP.getDaeResidual()
```

For further manipulations with the extracted DAE, the symbolic function using CasADi has to be established. To achieve this goal, all variables should be aggregated into an input vector.

```
MX_DAE = MXFunction(LIST_DER + ...
  LIST_DIFF + LIST_ALG + ...
  LIST_INPUT + P_I + P_D, [DAE])
MX_DAE.init()
```

Since JModelica.org works with the MX data type and the proposed toolchain accepts currently the SX data type, the MXFunction can be converted to SXFunction:

```
SX_DAE = SXFunction(MX_DAE)
SX_DAE.init()
```

For further information about data types in CasADi refer to the manual.

The function SX_DAE should be called with two arguments, the SX symbolic expression (converted from MX) and the scaled DAE system (with respect to the interval length).

```
SX_DAE_NUM = SX_DAE.call( ...
  SX_INPUTS_DAE)[0]
SX_DAE_FUNCTION = SXFunction( ...
  [vertcat(SX_INPUTS_DAE)],  ...
  [LENGTH*SX_DAE_NUM])
SX_DAE_FUNCTION.init()
```

This function SX_DAE_FUNCTION is involved in the discretization procedure, since it declares the variable order and accepts symbolical evaluation.

For the discretization of the model equations, additional variables should be introduced,

```
# Piecewise control
CTRL = SX.sym("c",N_INT*N_C)
# Parameterized states
P_S_P = SX.sym("ps",(N_INT+1)*(N_D))
# Collocated differential
# and algebraic states
S_P = SX.sym("s", ...
        N_INT*((N_D + N_A)*NCP))
```

where N_INT is the number of shooting intervals defined by the user, N_D, N_A, and N_C are the numbers of differential, algebraic, and control variables.

For each shooting interval, certain variables are chosen from vectors CTRL, P_S_P, and S_P. The SX_DAE_FUNCTION is called with these variables and the evaluation results are placed into RES variable. This procedure should be called for each interval.

```
RES = SX_DAE_FUNCTION.call( ...
  [vertcat([der, diff, alg, ctrl, ...
          p_i_v, p_d_v])])[0]
```

As mentioned before, the state trajectories and sensitivities have to be calculated for each interval. For this purpose, the toolchain uses Newton and LU solvers from CasADi.

```
# Newton solver
# interval_dae - discretized DAE
# system for one interval
# variables - states at collocation
# points
# parameters - parametrized states
# and controls
F = SXFunction( ...
  [vertcat([variables]), ...
  vertcat([parameters])], ...
[vertcat([interval_dae])])
solver = ImplicitFunction("newton",F)
solver.setOption("linear_solver", ...
                  "csparse")
solver.setOption("abstol",1e-12)
solver.init()
# LU solver
# NCP - number of collocation points
SYSTEM_INDEX = (N_D + N_A)*NCP
# Full Jacobian
partial_jacobian=interval_dae.jac()
dGdX_sym = partial_jacobian[ ...
           range(SYSTEM_INDEX), ...
          range(SYSTEM_INDEX)]
LHS = MX.sym('LHS', ...
           dGdX_sym.sparsity())
dGdXp_dU_sym = partial_jacobian[ ...
  range(SYSTEM_INDEX), ...
  SYSTEM_INDEX:SYSTEM_INDEX+N_D+N_C]
RHS = MX.sym('RHS', ...
           dGdXp_dU_sym.sparsity())
LUSolver = solve(LHS,RHS,"csparse")
FRHS = MXFunction( ...
         [LHS,RHS],[LUSolver])
FRHS.init()
```

The optimization constraints vector and the corresponding Jacobian matrix, objective function and its gradient are also constructed by means of CasADi using symbolic manipulation. For the sake of brevity, these details are not discussed here.

Corresponding to the number of the user-defined processes in the case of a multi-core CPU, shooting intervals can be equally distributed between them.

After problem initialization, the Ipopt instance is created to solve the optimization problem:

```
import pyipopt
nlp = pyipopt.create(args)
x_opt = nlp.solve(initialGuess)
```

5 Examples

Showing the efficiency of the proposed approach a small-scale and a large-scale problem are presented. All computations are performed on a stand-alone personal computer with Intel® I7 4.4 GHz, 6 cores, 16 GB RAM with Ubuntu 14.04.1 Server x64 operational system.

5.1 Satellite control problem

This nonlinear optimal control problem is devoted to the calculation of the optimal control and the concluding optimal torques according to a Bolza functional that bring the satellite to rest after an initial tumbling motion. The problem is listed e.g. in (Rudquist and Edvall, 2009).

The optimal states are shown in Figs. 3 and 4, where the state x_8 represents the integrand of the Lagrange term. The optimal controls are contained in Fig. 5. The time horizon corresponds to 100 seconds and is divided

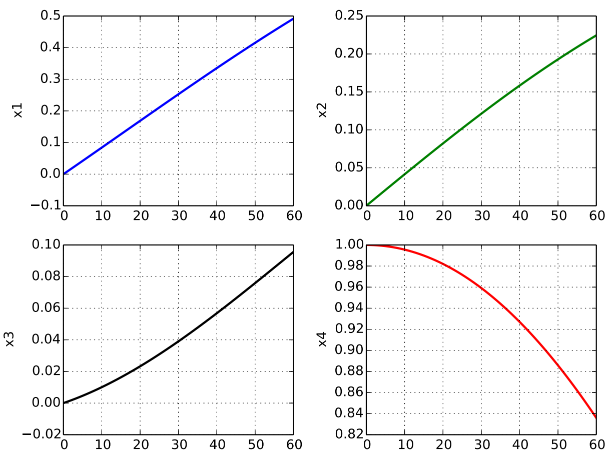

Figure 3. Optimal states $x_1(t)$ to $x_4(t)$

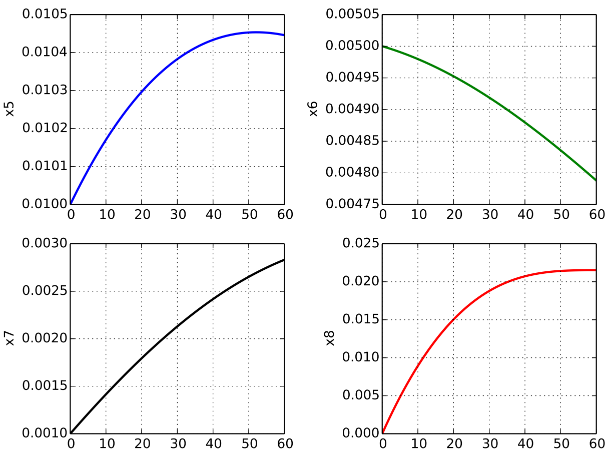

Figure 4. Optimal states $x_5(t)$ to $x_8(t)$

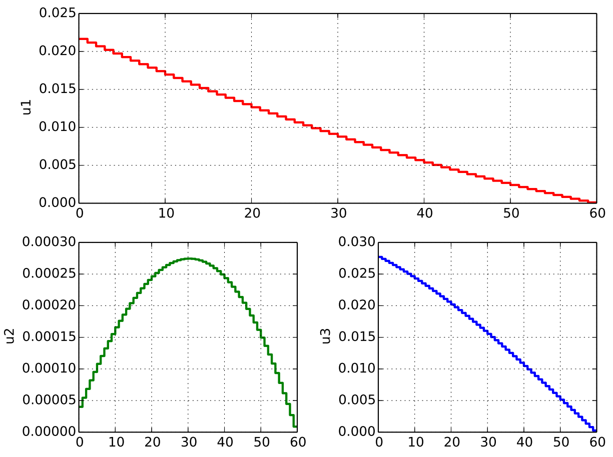

Figure 5. Optimal controls $u_1(t)$ to $u_3(t)$

Table 2. Satellite control problem dimensions

Number of ...	Value
intervals	60
non-zeros in Jacobians of eqs.	5,768
non-zeros in Jacobians of ineqs.	0
non-zeros in Hessian of Lagrangian	3,960
equality constraints	480
inequality constraints	0
variables	668
variables incl. Newton variables	2,108

into 60 intervals. Figs. 3 - 5 show the correct results. The optimal objective value is in every scenario $J^* = 0.4639$.

Data being constant through all scenarios are listed in Tab. 2. In case of the utilization of the analytical Hessians the number of non-zero elements in the Hessian of the Lagrangian equals to 3,960.

Tab. 3 shows the speed-up in different scenarios. In this small-scale problem the effect of parallelization is not substantial, but the number of iterations can be reduced and thus the computation time is less in most cases.

In a first case, comparing the speed-up by *parallelization within the same computation scheme for the Hessian (column s_1)*, i. e. considering the first and the second row in the column s_1 in BFGS, AH, and AHC case, resp., the speed-up factors s_1 reach from 1.08 (AHC), 1.33 (BFGS) to 1.56 (AH).

In the second case, comparisons are dedicated to the *non-parallelized versions (column s_2)* contrasting the Hessian calculation methods to each other. Unsurprisingly, no speed-up is achieved (AH/$n_c = 1/s_2 = 0.71$) due to the fact that the symbolic calculations take time to es-

Table 3. Speed-up by parallelization and utilization of analytic Hessian

Hess.	n_c	It.	t_Σ [s]	s_1	s_2	s_3
BFGS	1	8	0.515	1.00	1.00	N/A
	6	8	0.355	1.33	N/A	1.00
AH	1	5	0.723	1.00	0.71	N/A
	6	5	0.462	1.56	N/A	0.77
AHC	1	5	0.346	1.00	1.49	N/A
	6	5	0.320	1.08	N/A	1.11

Hess.: BFGS - approximated Hessian, AH - analytic Hessian (updated in every iteration of the optimizer), AHC - analytic Hessian (calculated once in iteration 0); n_c - no. of CPU cores ($n_c = 1$: no parallelization), It. - no. of iterations, t_Σ - total CPU time (mean value of 100 runs), s_i, $i = 1,2,3$ - speed-up factors, N/A - not applicable tablish the analytic Hessian. But, calculating the Hessian only once in the start iteration and leaving it constant through the iteration process also delivers the correct result with the risk that search direction could be non-descent, and the speed-up amounts to 1.49 (AHC/$n_c = 1/s_2 = 1.49$).

The third case considers the *parallelized versions (column s_3)*, also contrasting the Hessian calculation methods to each other. The computation with the BFGS approximation constitutes the reference. As to be expected in this small-scale example, one gets also no speed-up in the parallelized versions (AH/n_c=4/$s_3 = 0.77$) from BFGS approximation to analytic Hessian calculation. However, confronting BFGS with AHC, the speed-up accounts to 1.11 (AHC/$n_c = 4/s_3 = 1.11$).

5.2 Combined cycle power plant start-up control problem

Another example, a combined cycle power plant (CCPP) was chosen for several reasons. Firstly, this is an example of interest in the liberalized energy market, because classical power plants have to be adopted to the operation of electrical energy supply networks with renewable energies. Thus, they are more often set into operation or shutdown than in the past. Secondly, it is a high-dimensional problem compared with other academic examples. Thirdly, the example is used for the verification of the achieved results. In (Casella and Pretolani, 2006) the system was introduced. The plant is composed of a gas turbine unit, heat recovery steam generator, a steam turbine, and a condenser. The start-up time is limited due to the following facts: a maximum load change rate of the gas turbine, the thermal stress in the thick components (e.g. steam turbine shafts), and limited control variables.

Several authors used the object-oriented implemented model for the optimal control of the start-up process. In (Casella et al., 2011a) a simplified model is used. The contribution (Casella et al., 2011b) reports on a solution using JModelica.org, CppAD for automatic differentiation, and Ipopt for the NLP solution. The integration of CasADi and JModelica.org is described in (Andersson et al., 2011), where the CCPP system is also used as a benchmark system, but the solution is achieved by a direct collocation approach. An approach with OpenModelica and an optimization language specification, CasADi as the automatic differentiation tool, and different optimization methods including direct collocation and direct multiple shooting, is shown in (Shitahun et al., 2013). A parallel multiple shooting and a collocation optimization, performed with OpenModelica, is explained in (Bachmann et al., 2012). That paper discusses multiple shooting, multiple collocation, and total collocation methods using up to 8 cores of a multi-core CPU with OpenMP support. In (Ruge et al., 2014) the authors outline a toolchain including modeling with OpenModelica,

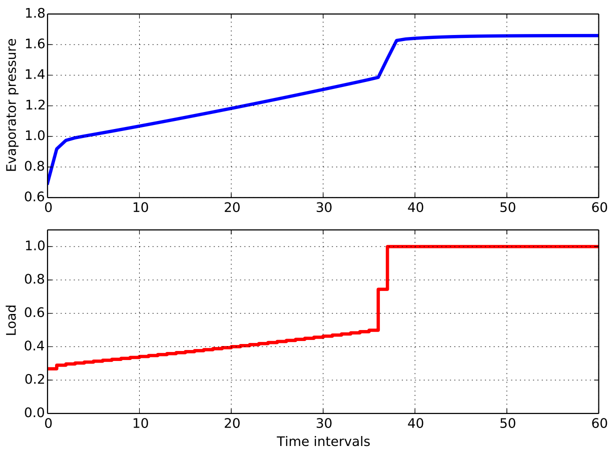

Figure 6. Optimal pressure and load

Table 4. CCPP problem dimensions

Number of ...	Value
intervals	60
non-zeros in Jacobians of eqs.	7,210
non-zeros in Jacobians of ineqs.	104,940
non-zeros in Hessian of Lagrangian	3,960
equality constraints	610
inequality constraints	9,540
variables	670
variables incl. Newton variables	24,790

but using automatic differentiation by ADOL-C.

The approach and toolchain discussed in our paper uses a modified combined multiple shooting and collocation (MCMSC) method, CasADi for automatic differentiation, JModelica.org for modeling and formulation of the optimization problem by means of Optimica, and Ipopt as NLP solver. Thus, a direct comparison to the contributions mentioned above is not possible due to different models, approaches, time horizons, tools, and computers used. Therefore, the only direct comparison between MCMSC and collocation method on finite elements using JModelica.org is given in Tab. 5.

Exemplarily, one of the essential optimal states (evaporator pressure) and the control (normalized load) are shown in Fig. 6 above and below, resp., over 60 time intervals corresponding to 4,000 seconds operation time, indicating the right behavior.

Data being constant through all scenarios are listed in Tab. 4. Using the analytical Hessians the number of non-zero elements in the Hessian of the Lagrangian equals to 3,960. Tab. 5 shows the acceleration of computation time in most cases.

Concerning this large-scale problem, the gain achieved by *parallelization within the same computation method for the Hessian (column s_1)* is better than in the small-scale problem above. Comparing the first and the second row in the column s_1 in each case (BFGS, AH,

Table 5. Speed-up by parallelization and utilization of analytic Hessian – comparison with collocation approach

MCMSC approach							
Hess.	n_c	It.	t_O [s]	t_{SN} [s]	s_1	s_2	s_3
BFGS	1	47	5.578	8.887	1.00	1.00	N/A
	6	47	5.585	4.509	1.97	N/A	1.00
AH	1	35	2.032	27.612	1.00	0.32	N/A
	6	35	2.038	12.530	2.20	N/A	0.36
AHC	1	34	1.044	7.112	1.00	1.25	N/A
	6	34	1.066	5.114	1.39	N/A	0.88

Collocation approach				
Hess.	n_c	It.	t_C [s]	s_{CMP}
BFGS	1	54	11.446	1.13
AH	1	60	7.892	0.54
AHC	1	77	6.299	1.02

Hess.: BFGS - approximated Hessian, AH - analytic Hessian (updated in every iteration), AHC - analytic Hessian (calculated once in iteration 0); n_c - no. of CPU cores, It. - no. of iterations, t_O - CPU time for optimization by Ipopt (not parallelizable), t_{SN} - CPU time for sensitivity calculation and Newton solver, t_C - CPU time for pure collocation on finite elements; (all CPU times are averages of 100 runs), s_i, $i = 1,2,3$ - speed-up factors, s_{CMP} - speed-up factor between collocation (CM) and parallelized MCMSC method, N/A - not applicable

and AHC, resp.), the speed-up factors s_1 reach from 1.39 (AHC), 1.97 (BFGS) to 2.20 (AH) referred to t_{SN}.

In the second case, the *non-parallelized versions (column s_2)* are under consideration referring the Hessian calculation methods to each other. Here, a speed-up is only achieved in the AHC case (AHC/$n_c = 1/s_2 = 1.25$).

The third comparison evaluates the *parallelized versions (column s_3)*, again contrasting the Hessian calculation methods to each other. Here, no speed-ups are achieved. Nevertheless, considering the optimization time t_O in the AH and AHC case compared with the BFGS case, it is significantly reduced by factors of 2.74 and 5.24, resp., because of conducive matrix structures.

To have at least one comparison on the same computer of the MCMSC method with collocation method (CM) on finite elements used in JModelica.org the lower part in Tab. 5 was added. In the parallelized version of the MCMSC method both the BFGS (BFGS/$n_c = 6/t_O + t_{SN} = 10.094$) and the AHC scenario (AHC/$n_c = 6/t_O + t_{SN} = 6.180$) are faster than the non-parallelized version of the CM.

The investigations and presented results show that the presented parallelized MCMSC approach is a powerful solution technique solving optimal control problems within the proposed toolchain. The number of iterations are reduced compared with both the non-parallelized cases and the collocation approach, but the effort in one iteration is typically higher if the analytic Hessian is used. The approach can most advantageously be applied to large-scale optimization problems.

6 Summary and Conclusions

An optimal control problem needs to be solved online in NMPC. This poses a challenge in the implementation of the numerical algorithms and the enhancement of the computation efficiency for the dynamic optimization approach. In this work, a toolchain for solving nonlinear dynamic optimization problems is developed based on the combined multiple shooting and collocation method. The toolchain is implemented in open-source software and both the first and the second-order sensitivities are automatically computed. As a result, the user needs only to provide the defined optimal control problem for implementing NMPC. In addition, parallel computing is realized for performing the computations in the individual time intervals, thus leading to a reasonable reduction of the computation time. The results of two case studies show the capability of the toolchain for efficiently solving small to large-scale dynamic optimization problems.

In future, it is planned to offer a web-based optimization service for solving nonlinear dynamic optimization problems using the proposed approach.

7 Acknowledgments

The authors are grateful for the support of the ITEA2/ EUREKA Cluster programme by the European Commission (project no. 11004, Model Driven Physical Systems Operation (MODRIO)) and for the financing by German Ministry of Education and Research (BMBF, Förderkennzeichen: 01IS12022H).

References

J. Åkesson. Optimica – an extension of Modelica supporting dynamic optimization. In *Proc. 6th Int. Modelica Conf.*, pages 57–66. Modelica Association, March 3-4 2008.

J. Åkesson, K.-E. Årzén, M. Gåfvert, T. Bergdahl, and H. Tummescheit. Modeling and optimization with Optimica and JModelica.org-languages and tools for solving large-scale dynamic optimization problems. *Comput. Chem. Eng.*, 34(11):1737–1749, 2010.

J. Andersson, J. Åkesson, F. Casella, and M. Diehl. Integration of CasADi and JModelica.org. In *Proc. 8th Int. Modelica Conf.*, pages 218–231, 2011. doi:10.3384/ecp11063.

J. Andersson, J. Åkesson, and M. Diehl. Dynamic optimization with CasADi. In *51st IEEE Conference on Decision and Control*, pages 681–686, 10-13 December 2012a. doi:10.1109/CDC.2012.6426534.

J. Andersson, J. Åkesson, and M. Diehl. CasADi: A symbolic package for automatic differentiation and optimal control. *Lect. Notes Comput. Sci. Eng.*, 87:297–307, 2012b.

B. Bachmann, L. Ochel, V. Ruge, M. Gebremedhin, P. Fritzson, V. Nezhadali, L. Eriksson, and M. Sivertsson. Parallel multiple-shooting and collocation optimization with OpenModelica. In *Proc. 9th Int. Modelica Conf.*, pages 659–668, 2012.

E. Balsa-Canto, J. R. Banga, A. A. Alonso, and V. S. Vassiliadis. Efficient optimal control of bioprocesses using second-order information. *Ind. Eng. Chem. Res.*, 39(11): 4287–4295, 2000. doi:10.1021/ie990658p.

M. Bartl, P. Li, and L. T. Biegler. Improvement of state profile accuracy in nonlinear dynamic optimization with the quasi-sequential approach. *AIChE J.*, 57(8):2185–2197, 2011. doi:10.1002/aic.12437.

T. Barz, R. Klaus, L. Zhu, G. Wozny, and H. Arellano-Garcia. Generation of discrete first- and second-order sensitivities for single shooting. *AIChE J.*, 58(10):3110–3122, 2012.

L. T. Biegler, A. M. Cervantes, and A. Wächter. Advances in simultaneous strategies for dynamic process optimization. *Chem. Eng. Sci.*, 57(4):575–593, 2002.

F. Casella and F. Pretolani. Fast Start-up of a Combined-Cycle Power Plant: a Simulation Study with Modelica. In *Proc. 5th Modelica Conf.*, pages 3–10, 2006.

F. Casella, F. Donida, and J. Åkesson. Object-Oriented Modeling and Optimal Control: A Case Study in Power Plant Start-Up. In *Prepr. 18th IFAC World Congress. Milano. Italy*, pages 9545–9554, 2011a.

F. Casella, M. Farina, F. Righetti, R. Scattolini, D. Faille, F. Davelaar, A. Tica, H. Gueguen, and D. Dumur. An optimization procedure of the start-up of Combined Cycle Power Plants. In *Prepr. 18th IFAC World Congress. Milano. Italy*, pages 7043–7048, 2011b.

T. A. Davis. *Direct methods for sparse linear systems*. SIAM, 2006.

P. Fritzson. *Principles of object-oriented modeling and simulation with Modelica 3.3: A cyber-physical approach*. Wiley-IEEE Press, 2014.

D. Hellmann. *The Python standard library by example*. Addison-Wesley Professional, 1st edition edition, 2011.

W. R. Hong, S. Q. Wang, P. Li, G. Wozny, and L. T. Biegler. A quasi-sequential approach to large-scale dynamic optimization problems. *AIChE J.*, 52(1):255–268, 2006. doi:10.1002/aic.10625.

B. Houska, H. J. Ferreau, and M. Diehl. An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, 47(10):2279–2285, 2011.

C. Kirches, L. Wirsching, H. G. Bock, and J. P. Schlöder. Efficient direct multiple shooting for nonlinear model predictive control on long horizons. *J. Process Contr.*, 22(3):540–551, 2012.

E. Lazutkin, A. Geletu, S. Hopfgarten, and P. Li. Modified multiple shooting combined with collocation method in JModelica.org with symbolic calculations. In *Proc. 10th Int. Modelica Conf.*, pages 999–1006, 2014. doi:10.3384/ECP14096999.

D. Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.

P. E. Rudquist and M. M. Edvall. PROPT - Matlab Optimal Control Software. User's Guide. TOMLAB Optimization, 2009.

V. Ruge, W. Braun, B. Bachmann, A. Walther, and K. Kulshreshtha. Efficient Implementation of Collocation Methods for Optimization using OpenModelica and ADOL-C. In *Proc. 10th Int. Modelica Conf.*, pages 1017–1025, 2014. doi:10.3384/ECP140961017.

A. Schäfer, P. Kühl, M. Diehl, J. Schlöder, and H. G. Bock. Fast reduced multiple-shooting method for nonlinear model predictive control. *Chem. Eng. Process.*, 46(11):1200–1214, 2007.

A. Shitahun, V. Ruge, M. Gebremedhin, B. Bachmann, L. Eriksson, J. Andersson, M. Diehl, and P. Fritzson. Model-Based Dynamic Optimization with OpenModelica and CasADi. In *7th IFAC Symp. on Advances in Automotive Control*, volume 1, pages 446–451, 2013. doi:10.3182/20130904-4-JP-2042.00166.

J. Tamimi and P. Li. Nonlinear model predictive control using multiple shooting combined with collocation on finite elements. In *7th IFAC Int. Symp. on Advanced Control of Chemical Processes*, pages 703–708, 2009. doi:10.3182/20090712-4-TR-2008.00114.

J. Tamimi and P. Li. A combined approach to nonlinear model predictive control of fast systems. *J. Process Contr.*, 20(9): 1092–1102, 2010.

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

Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE T. Contr. Syst. T.*, 18(2):267–278, 2010.

D. P. Word, J. Kang, J. Åkesson, and C. D. Laird. Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Comput. Optim. Appl.*, 59(3):667–688, 2014. doi:10.1007/s10589-014-9651-2.

V. M. Zavala, C. D. Laird, and L. T. Biegler. Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. *Chem. Eng. Sci.*, 63(4834-4845):19, 2008.