TECHNISCHE UNIVERSITÄT ILMENAU

ITEA 2
INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

D5.2.1 – Prototype for optimisation toolchain in JModelica.org

WP 5.2 Optimisation Toolchain
WP 5 Optimized system operation

MODRIO (11004)

Version 2.0 (M39)

Date 17/11/2015

Authors

Evgeny Lazutkin	TU Ilmenau, Germany
Siegbert Hopfgarten	TU Ilmenau, Germany

Executive summary

This is the final report (delivery M39), which describes the task of the deliverable and summarizes obtained results. The purpose of the research is to solve dynamic optimization problems within a nonlinear model predictive control (NMPC) optimization framework. There is a dependency to work package 5.1 (Nonlinear model predictive control). For the purpose of an efficient implementation a chain of open-source tools is combined and further developed.

The objective of the deliverable concerns the implementation of the method for discretization of the dynamic optimization problem by combined multiple shooting with collocation (CMSC), as well as pure collocation, parameter identification, adaptive discretization (free length of time intervals) online realization on an industrial power plant, and parallelization of the algorithm. The tool-chain should be tested with strong focus on application in power generation. The benchmark optimization problem is developed by Siemens AG, Erlangen, Germany.

The following results have been obtained:

- The CMSC method accepts Modelica/Optimica model formulations.
- The CMSC method has been extended to work with symbolic computations, automatic differentiation for first as well as second order sensitivities, which are firstly obtained within this project.
- Different types of optimization problems can be solved, i.e., optimal control problems (e.g. minimum-energy, minimum-time problems), and parameter estimation problems.
- The toolchain has been parallelized using the standard Python “Multiprocessing” module.
- The model equations of an optimization problem under consideration can be either explicit or implicit differential-algebraic equations (DAEs) and can be solved without user intervention meaning that the toolchain is in a general form.
- The algorithm has been tested on several optimization problems including the benchmark problem developed by Siemens AG, Erlangen, Germany;
- The adaptive discretization has been implemented, where the length of each interval is concerned to be an optimization variable including error estimation at the non-collocation points using multiple level simulations.
- Control-variable correlation analysis is introduced to decide either approximated or analytical second-order sensitivities should be used in optimization.

The proposed CMSC algorithm uses Ipopt as nonlinear programming (NLP) problem software to find an approximate optimal solution of the dynamic optimization problem and CasADi to provide automatic and symbolic differentiation, a user-friendly interface, and a simplified optimization process. The detailed manual and web-based toolchain will be available in a near future, to allow users to easily solve their dynamic optimization tasks.

Contents

1. Combined multiple shooting with collocation method

1.1. Dynamic optimization problem

This deliverable considers dynamic nonlinear optimization problems of the form

$$\min_{u(t)} \left\{ J = M\left(x(t_f)\right) + \int_{t_0}^{t_f} L(x(t), z(t), u(t), t)dt \right\}$$

$$\begin{aligned} \text{subject to:} \quad & \dot{x}(t) = f(x(t), z(t), u(t), t) = 0, x(t_0) = x_0 \\ & g(x(t), z(t), u(t), t) \leq 0 \\ & x_{min} \leq x(t) \leq x_{max} \\ & z_{min} \leq z(t) \leq z_{max} \\ & u_{min} \leq u(t) \leq u_{max} \end{aligned} \tag{1}$$

where $x(t) \in \Re^{n_x}$ and $z(t) \in \Re^{n_z}$ represents the vectors of state and algebraic variables, $u(t) \in \Re^{n_u}$ is the control vector. The time horizon is defined as $t \in [t_0, t_f]$, with initial time t_0 and terminal time t_f. The initial state $x(t_0)$ is usually known, while the terminal state $x(t_f)$ can be either fixed or free. The functions $f: \Re^{n_x} \times \Re^{n_z} \times \Re^{n_u} \times [t_0, t_f] \to \Re^{n_x}$ as well as $g: \Re^{n_x} \times \Re^{n_z} \times \Re^{n_u} \times [t_0, t_f] \to \Re^{n_z}$ in model equations, and the objective function J, consisting of a Mayer term $M: \Re^{n_x} \to \Re$ and a Lagrange term $L: \Re^{n_x} \times \Re^{n_z} \times \Re^{n_u} \times [t_0, t_f] \to \Re$, are assumed to be twice continuously differentiable. Due to the physical limits, controls $u(t)$, states $x(t)$, and algebraic variables $z(t)$ are usually bounded as indicated by the box constraints above.

In JModelica.org the optimization problem of the form (1) can already be formulated, but the solution can so far be obtained using pure collocation framework developed by Modelon AB, Sweden [1]. Therefore the main objective is to adapt the CMSC method to accept Modelica and Optimica language.

1.2. Transformation to NLP problem

In this deliverable, the CMSC method is used to transform a dynamic optimization problem (1) into a finite-dimensional NLP problem. The time horizon $[t_0, t_f]$ is divided into N intervals and in each interval the state and algebraic variables are treated using the multiple-shooting method. Subsequently, on each time interval, state and control variables are parameterized as initial values and piecewise constants, respectively, i.e., $\boldsymbol{X^p} = [X_{p,0}, X_{p,1}, \dots, X_{p,N}]$ and $\boldsymbol{V} = [V_0, V_1, \dots, V_{N-1,}]$. Equality constraints $X_{p,i+1} = X^{c,d}(t_{i+1}; X_{p,i}, V_i)$ are introduced to ensure continuity of the states between interval i and $i+1$. Since the decision variables are $\boldsymbol{X^p} \in \Re^{n_x \cdot (N+1)}$ and $\boldsymbol{V} \in \Re^{n_u \cdot N}$, problem (1) is now transformed to the following NLP formulation

$$\min_{\boldsymbol{X^p}, \boldsymbol{V}} \left\{ M(X_{p,N}) + \sum_{i=0}^{N-1} L_i(X_{p,i}, V_i) \right\}$$

$$\begin{aligned} \text{subject to:} \quad & X_{p,i+1} = X^{c,d}(t_{i+1}; X_{p,i}, V_i),\ i = 0, \dots, N-1 \\ & X_{p,0} = X_0 \\ & G(X^c, V_i) \leq 0,\ i = 0, \dots, N-1 \\ & x_{min} \leq \boldsymbol{X^p} \leq x_{max} \\ & u_{min} \leq \boldsymbol{V} \leq u_{max} \end{aligned} \tag{2}$$

Note that model equations are not directly involved in the NLP formulation, since, in each time interval, the discretized model equations are solved in a Newton step to obtain state trajectories and first- as well as second-order sensitivities described in the following subsections. More details about the

CMSC method can be found in Tamimi and Li [2].

1.3. Computation of first-order sensitivities

In the CMSC method for each time interval a collocation method is employed. The state and algebraic variables are approximated inside each time interval by a linear combination of the Lagrange polynomials using the Radau scheme with N_c collocation points. After this, the state and algebraic variables X^c at all collocation points have implicit dependency on the decision variables X^p and V. Therefore, the discretized dynamic model equations for each time interval can be written in the following compact form

$$[G(X^c(X^p,V),X^p,V)]_{N_{xz}\times 1} = \mathbf{0} \tag{3}$$

where $N_{xz} = (n_x + n_z)\cdot N_c$ and $X^c \in \Re^{N_{xz}}$. For the sake of brevity, the index for time intervals is dropped here. To obtain the first-order sensitivities for individual time intervals, two differentiation operators $\frac{\partial}{\partial X^p}$ and $\frac{\partial}{\partial V}$ are applied to (3), which yields

$$\left[\frac{\partial G}{\partial X^c}\right]_{N_{xz}\times N_{xz}} \cdot \left[\frac{\partial X^c}{\partial X^p}\right]_{N_{xz}\times n_x} = -\left[\frac{\partial G}{\partial X^p}\right]_{N_{xz}\times n_x} \tag{4}$$

$$\left[\frac{\partial G}{\partial X^c}\right]_{N_{xz}\times N_{xz}} \cdot \left[\frac{\partial X^c}{\partial V}\right]_{N_{xz}\times n_u} = -\left[\frac{\partial G}{\partial V}\right]_{N_{xz}\times n_u} \tag{5}$$

The construction of the linear equation systems above for the first-order sensitivities can be done straightforward using automatic differentiation. Due to the CMSC method, equations (4) and (5) have a sparse structure and thus can be solved for $\frac{\partial X^c}{\partial X^p}$ and $\frac{\partial X^c}{\partial V}$ by a linear sparse algebra algorithm. Details can be found in [3] and [4].

1.4. Computation of second-order sensitivities

We derive the formulas for computing second-order sensitivities for the CMSC method. It should be noted that, in general, there are two types of second-order sensitivities: derivatives of functions to all variables (both states and controls) in the context of simultaneous approaches and derivatives of state variables to controls in the context of (quasi-)sequential approaches. The CMSC method is a kind of sequential approach and thus we need to compute the second-order derivatives of states with respect to controls. In comparison to the previous studies [5, 6], where sensitivities have to be transferred from one time interval to the next interval (so-called global sensitivities) in the context of single-shooting, we employ the advantage of the CMSC method that the computation of the second-order sensitivities is independent in different time intervals.

In each time interval the first-order sensitivities $\frac{\partial X^c}{\partial X^p}$ and $\frac{\partial X^c}{\partial V}$ calculated from equations (4) and (5) have an implicit dependency on the decision variables of this interval only. To clearly explain the derivation, the first-order sensitivity (matrix) equation (4) is reformulated as a vector of linear equations. This is made by multiplying the matrix $\frac{\partial G}{\partial X^c}$ with the vector $\left[\frac{\partial X^c}{\partial X^p}\right]^k$ and the result is added with the vector $\left[\frac{\partial G}{\partial X^p}\right]^k$, where $k = 1,\dots,n_x$ is a column number. This procedure is also done for (5), where $k = 1,\dots,n_u$. As a result, equations (4) and (5) can be re-written in the following compact form

$$\left[\Phi\left(X^c(X^p,V),X^p,V,\frac{\partial X^c}{\partial X^p}(X^p,V)\right)\right]_{N_{XZ}^X\times 1} = \mathbf{0} \tag{6}$$

$$\left[\Psi\left(X^c(X^p,V),X^p,V,\frac{\partial X^c}{\partial V}(X^p,V)\right)\right]_{N_{XZ}^U\times 1} = \mathbf{0} \tag{7}$$

where $N_{XZ}^X = N_{xz}\cdot n_x$ and $N_{XZ}^U = N_{xz}\cdot n_u$. Applying both differentiation operators $\frac{\partial}{\partial X^p}$ and $\frac{\partial}{\partial V}$ to (6) and (7), the equations for the second-order sensitivities can be expressed as follows

$$\left[\frac{\partial \Phi}{\partial\left(\frac{\partial X^c}{\partial X^p}\right)}\right]_{N_{XZ}^X \times N_{XZ}^X} \cdot \left[\frac{\partial^2 X^c}{\partial X^{p,2}}\right]_{N_{XZ}^X \times n_x} = -\left[\frac{\partial \Phi}{\partial X^c}\right]_{N_{XZ}^X \times N_{xz}} \cdot \frac{\partial X^c}{\partial X^p} - \left[\frac{\partial \Phi}{\partial X^p}\right]_{N_{XZ}^X \times n_x} \quad (8)$$

$$\left[\frac{\partial \Phi}{\partial\left(\frac{\partial X^c}{\partial X^p}\right)}\right]_{N_{XZ}^X \times N_{XZ}^X} \cdot \left[\frac{\partial^2 X^c}{\partial X^p \partial V}\right]_{N_{XZ}^X \times n_u} = -\left[\frac{\partial \Phi}{\partial X^c}\right]_{N_{XZ}^X \times N_{xz}} \cdot \frac{\partial X^c}{\partial V} - \left[\frac{\partial \Phi}{\partial V}\right]_{N_{XZ}^X \times n_u} \quad (9)$$

$$\left[\frac{\partial \Psi}{\partial\left(\frac{\partial X^c}{\partial V}\right)}\right]_{N_{XZ}^U \times N_{XZ}^U} \cdot \left[\frac{\partial^2 X^c}{\partial V \partial X^p}\right]_{N_{XZ}^U \times n_x} = -\left[\frac{\partial \Psi}{\partial X^c}\right]_{N_{XZ}^U \times N_{xz}} \cdot \frac{\partial X^c}{\partial X^p} - \left[\frac{\partial \Psi}{\partial X^p}\right]_{N_{XZ}^U \times n_x} \quad (10)$$

$$\left[\frac{\partial \Psi}{\partial\left(\frac{\partial X^c}{\partial V}\right)}\right]_{N_{XZ}^U \times N_{XZ}^U} \cdot \left[\frac{\partial^2 X^c}{\partial V^2}\right]_{N_{XZ}^U \times n_u} = -\left[\frac{\partial \Psi}{\partial X^c}\right]_{N_{XZ}^U \times N_{xz}} \cdot \frac{\partial X^c}{\partial V} - \left[\frac{\partial \Psi}{\partial V}\right]_{N_{XZ}^U \times n_u} \quad (11)$$

It can be seen that the matrix dimensions grow rapidly as the number of state and control variables increases, as indicated in (8) – (11). In these equations, the partial derivative matrices can be generated by applying automatic differentiation. Note that (10) is dropped, since sensitivities $\frac{\partial^2 X^c}{\partial X^p \partial V}$ and $\frac{\partial^2 X^c}{\partial V \partial X^p}$ are exactly the same. Then the second-order derivatives can be calculated by solving the above linear matrix equations by a sparse linear solver and the results provide an analytical Hessian (AH) for the CMSC method.

However, there are two major obstacles to use an AH, especially for solving large-scale problems. First, the generation of an AH is complicated in general. Nevertheless, it is quite straightforward by using our approach described above. Second, the computation time for each NLP iteration of an AH may be expensive, i.e. the computation time for each NLP iteration by using AH will be higher than by using a Broyden-Fletcher-Goldfarb-Shanno (BFGS) approximation. Hence, the improvement of the computation time for solving an NLP problem can be achieved only if the difference between the number of NLP iterations by using an AH and a BFGS formula is sufficiently large.

1.5. Adaptive discretization and error estimation

The first approaches to error handling have been published in the 1970s by de Boor [7]. The aim was to find an optimal distribution of collocation points, which can approximate state trajectories more efficiently. His ideas were further analyzed by Russell and Christiansen [8] and compared with the approaches of Dodson [9], Pereyra and Sewell [10], and White [11]. In these works a suitable time partition for a boundary value problem was analyzed, which minimizes the error between the actual solution and the numerical approximation by polynomials. Based on a very rough partition an algorithm refines the time grid until a tolerance specified by the user is reached. Furthermore it is shown that the calculation of the error and the problem to find a suitable partition can be considered independently. The subsequent work [12] finally includes a derivation of an approximation error over the entire partition to be equally distributed. In the 1980s first publications were available, where the state error estimation plays a role within an optimization algorithm. In the work of Renfro [13] an optimization problem was formulated to find the minimum number of discrete intervals which satisfies the Euler-Lagrange condition. However, in this work only an equidistant partition was considered by pointing to a similar approach in the case of variable interval lengths. Then Vasantharajan and Biegler have used in [14] the explicit calculation of the approximation error to solve optimization problems of parameter estimation as precisely as possible. Two different strategies for implementation were presented. In the first variant the uniform distribution of the error is analyzed [14]. In the other variant, a much more intuitive approach has been investigated. The transformation to NLP problem was based on the direct calculation, where the error estimation was integrated in a NLP formulation by introducing inequality constraints. This formulation could also be extended by adding new discretization intervals. Finally, it was noted that the use of the approaches can occur in optimal control problems stability and accuracy problems and must be investigated further. One way to solve such problems was presented

by Tanartkit [15]. This approach uses a bi-level algorithm. To increase the efficiency and robustness of the previous algorithms, the optimization problem is divided into two interconnected minor problems. While the partitioning of the time axis is considered in the outer problem, in the inner problem the optimal control is calculated for constant time intervals. However, this means a relatively high expense, because every problem for achieving a solution and a termination criterion for the entire algorithm must be formulated. In addition, a method was presented to find the number of discrete intervals needed to ensure sufficiently accurate approximations. Another way to ensure the accuracy of the approximation of the states below a tolerance limit was introduced by Bartl et al. [17]. In this approach, the states are calculated by a simulation layer for given control variables and the length of each interval. To keep the error under given tolerance, original intervals are further subdivided until the desired accuracy is reached. This work was an extension of a quasi-sequential approach, and serves as a basis for the error estimation using CMSC method.

The error estimation can be performed using the following formula

$$e_{i,nonc}(t) = \left| c_{nonc}(\tau_{nonc}) \cdot R^{j}_{i,nonc}(t_{nonc}) \right|, i = 1, \dots, N; j = 1, \dots, n_x \tag{12}$$

The residual $R^{j}_{i,nonc}(t_{nonc})$ can be calculated through the model equations but the states must be computed at non-collocation points in the normalized time interval $\tau = [0,1]$, which are uniformly distributed. The parameter $c_{nonc}(\tau_{nonc})$ depends only on the position of non-collocation points and is computed by

$$c_{nonc}(\tau_{nonc}) = \frac{1}{\prod_{i=1}^{N_c}(\tau_{nonc} - \tau_i)} \cdot \int_0^{\tau_{nonc}} \prod_{i=1}^{N_c} (s - \tau_i) ds \tag{13}$$

Now the implemented error estimation used for the states will be explained. First, the global error is calculated over the whole interval using (12). If at any point the error limit e_{max} is violated, the sub-simulation will be provided. Firstly, we have to find in which sub-interval the error is too high and between which collocation points this occurs. This is done using non-collocation points defined by user. However, the resulting new end point differs from the original endpoint, i.e. starting from this new point, the next segment must be recalculated. This happens until the interval bound of the original interval is reached. Figure 1 is designed to support this declaration visually. Section I-III represents an originally given discretization, where the control variable is kept constant. Section IV already belongs to the next interval, which can be treated independently, thanks to the nature of multiple-shooting method. The solid black line represents the calculated state trajectory. The blue circles represent the original collocation points used by the optimization. In the Figure 1 (upper part) the sub-simulations are already carried out in sub-intervals II and III, because the error $e(t)$ was above the given tolerance limit. Thereby, the green triangles are collocation points, which must be inserted from the sub-simulation. The simulation of the new state trajectory (green dotted line) has always to be performed until the end of the original interval. As a result, new points must be computed. Apparently, after the first sub-simulation, the error trajectory is still above the given limit. Therefore the second sub-interval is divided in two new sub-intervals, i.e. II-I and II-II. The calculated endpoint of the sub-interval II-II serves as an initial point for the sub-interval III. The subdivision of intervals is continued until the error is small enough, i.e. below a given limit e_{max} or until a preselected number of sub-divisions exceeds maximum possible sub-simulations.

The continuity of the state trajectories between intervals will be guaranteed by optimizer. At this point it becomes clear that this is not necessarily an improvement in terms of convergence and convergence speed in this process. As will be seen later, this kind of error estimation may negatively influence the optimizer and will lead to more iterations to converge to an optimal solution. Using the CMSC framework with an error estimation the interval lengths are considered as additional optimization variables.

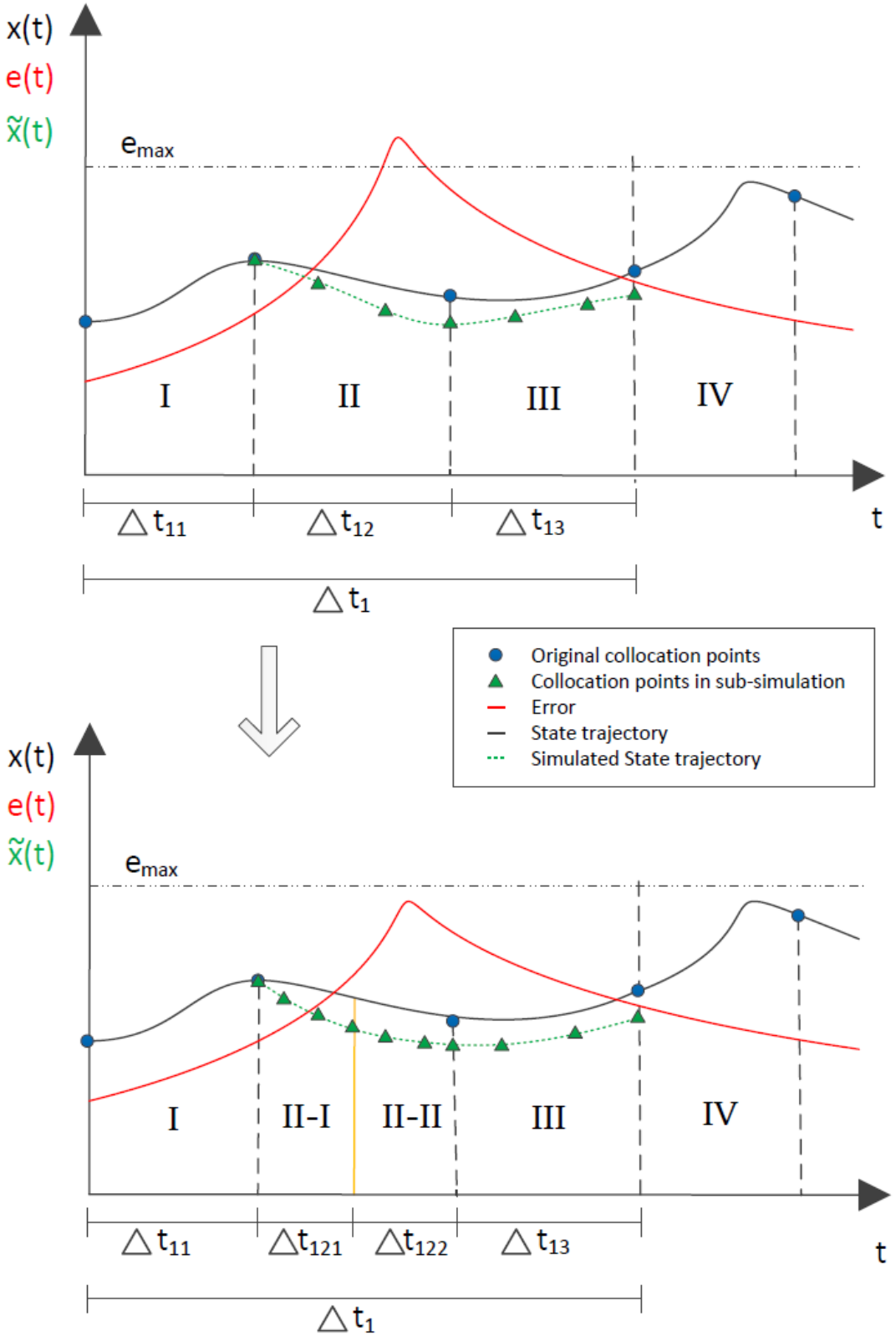

Figure 1: Principle of sub-simulations

1.6. The CMSC framework

The proposed solution approach is implemented based on Modelica and Optimica models.

Modelica is an object-oriented modelling environment commonly used for complex industrial systems and Optimica extension deals with optimization problems.

In this work, a toolchain for general usage is developed in such a way that the user needs only to define and to input the dynamic optimization problem. The transformation from the dynamic optimization problem to NLP formulation is made completely automatic. Then, by means of JModelica.org the Modelica and Optimica models are transferred to a symbolic representation and made accessible in Python. CasADi [18] is used for performing the discretization of the DAEs, automatic differentiation, and calculation of analytical first- as well as second-order derivatives. In addition, CasADi is also employed for generating Jacobian and Hessian as well as for solving the discretized DAEs. Finally, Ipopt [19] is used for solving the NLP problem, where options of either AH or BFGS can be chosen by the user.

Utilizing a personal computer with a limited number of processors, a master processor is defined for the NLP solver and the remaining processors are defined as worker processors each of which executes the computations for an equal number of time intervals. As soon as the NLP solver calls the multiprocessing function, the information will be distributed to the individual worker processors.

Computations for solving model equations (3) using a Newton method and for calculating the first-order sensitivities (see (4), (5)) as well as second-order sensitivities (see (8), (9), (11)) are performed independently for individual time intervals. Parallel computing is implemented for performing these computations for each time interval using the standard multiprocessing Python module.

It should be noted that the time taken for the communications between the master processor and the worker processors may be significant, if too many worker processors are used for solving a small-scale problem. In such a case, the maximum speed-up factor may be achieved when a fewer number of worker processors are used.

The framework is divided into four main parts: load algorithm options, load model, prepare NLP problem, and optimization procedure. The formulated optimization model is transferred using `transfer_optimization_problem` function, which is available from JModelica.org. If needed, the simulation procedure with CMSC method can be called firstly to obtain a proper initial guess for the optimizer. Since the multiple-shooting discretization strategy allows parallel computations, the user should specify how many parallel processes should be involved in the optimization process. Currently, parallelization is based on a memory shared-memory CPUs computers and not involving clustering or graphics processing units (GPUs). After preprocessing (model compilation, discretization, scaling, etc.) the optimizer calls required callback functions.

The CMSC framework has been successfully tested on the stand-alone computer equipped with Intel I7 4.4 GHz, 6 cores, 16GB RAM with Ubuntu 14.04.1 Server x64 operational system.

2. Case studies

2.1. Efficient start-up of a combined cycle power plant

The model has been developed by Siemens AG, Erlangen, Germany. The purpose of this model is to find an optimal control strategy to obtain a minimum-time start-up procedure of a combined cycle power plant (CCPP). The main limiting factor is the thermal stress and the change rate of the turbine load. Taking these aspects into account, the optimal control problem can be formulated considering the system dynamics described by implicit DAEs and the initial conditions for differential and algebraic states.

It should be noted, that such thermo-dynamical systems are very complicated to solve, because the states have a different order of magnitude. To avoid such difficulties, scaling of the problem has been applied by means of nominal values provided in the model.

For the following application of the online NMPC, the problem is solved online using 30 discrete intervals. The optimal start up is given in Figure 2. Applying six parallel processes the solution of the offline optimization problem is obtained within approximately 6 seconds for CMSC method and 9 seconds by pure collocation method.

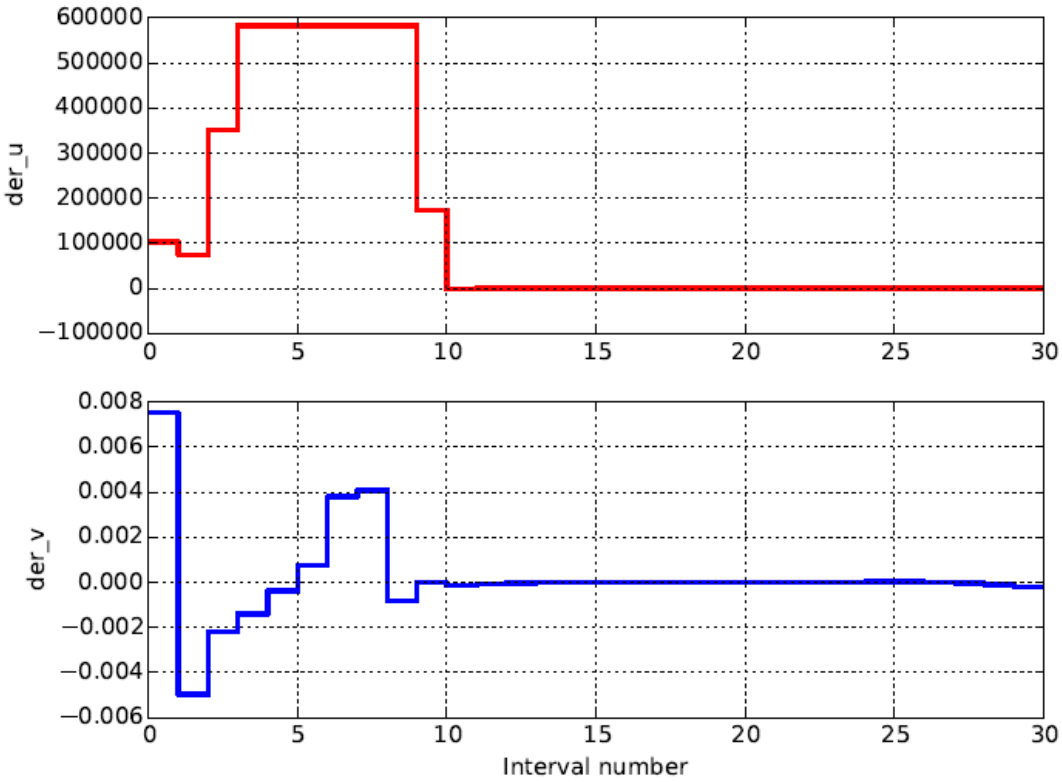

Figure 2: Optimal control profiles for the start-up of the CCPP

2.2. Continuous-stirred tank reactor

This problem was introduced by Luus [20] and further analyzed by Balsa-Canto et al. [21]. The purpose of this optimization problem is to determine four control variables of the CSTR in order to maximize the economic benefit. The system dynamics is represented by several simultaneous chemical reactions. The dynamic optimization problem is formulated as given in [21]. This example has been chosen to demonstrate the effect of the analytical Hessian. We transform the NOCP into NLP problem using 60 shooting intervals. In Table 1, the solutions of this problem using BFGS and AH are given. It can be seen that particularly for this problem using AH is efficient as BFGS, i.e., the number of NLP iterations is significantly reduced (last column in Table 1). As a result, the computation time is also reduced. Moreover, it is interesting to note from Table 1 that the improvement of computation time by parallel computing (using 6 processors) is insignificant because the communication takes a large portion of the total computation time. In addition, the control-variable correlation analysis has been introduced for dynamic optimization in order to decide either BFGS formula or AH should be used, including the optimal control of the distillation column [22].

Table 1: Solution of the continuous-stirred tank reactor

Method	1 CPU	6 CPU	Objective	Iteration
BFGS	112.407 s	93.119 s	21.82414	1687
AH	2.651 s	1.795 s	21.82414	22

2.3. Dielectrophoresis example

The explanations and details of this example were taken from [23]. The field of application of dielectrophoresis is in the sorting and detection of e.g. nanoparticles and viruses. The control variable here is an inhomogeneous electric field to manipulate with charged particles. The physical process

can be described as follows. We have a chamber, where a liquid with low Reynolds number flows through. There are electrodes at the bottom of the chamber which are arranged in parallel lengthwise. By applying a voltage, particles can now be moved within the liquid. The position of the particle is thereby limited to the size of electrodes. We assume that the particles have neutral buoyancy, which means that the buoyancy of the liquid and the gravity compensate each other.The optimization problem is formulated as follows

$$\min_{u(t),t_f} \{J = t_f\}$$

$$\text{subject to:} \quad \dot{x}(t) = y(t) \cdot u(t) + \alpha \cdot u^2(t) = 0, x(t_0) = 1, x(t_f) = 2$$

$$\dot{y}(t) = -C \cdot y(t) + u(t),\ y(t_0) = 0$$

$$\alpha = -\frac{3}{4}, C = 1$$

$$-1 \le u(t) \le 1 \qquad (14)$$

The aim is thus again minimizing the time for release of the particle from $x(t_0)$ to $x(t_f)$. The state x describes the position of the particle in the sense of the liquid height, while y describes the exponentially decaying part of the induced dipole moment. The constants α and C are the parameters of the dielectric conductivity of the particle and the liquid. The solution of this problem is given in Figure 3A and Figure 3B. The problem is solved with and without error estimation in 42 and 52 iterations, respectively. The final time obtained by optimizer without simulation equals to 7.81655545, while the final time obtained by applying error estimation with given error limit $e_{max} = 10^{-5}$ equals to 7.81530275, using 8 non-collocation points. As expected, the solution using the error estimation is better, i.e., using the error estimation the better objective function value is found.

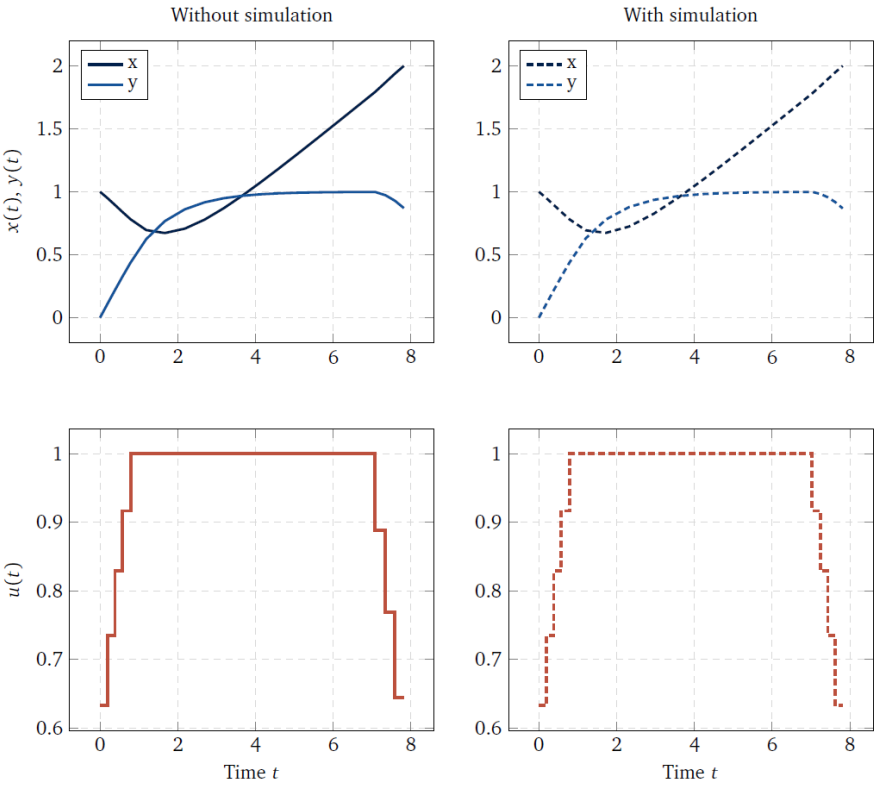

Figure 3A: State and control trajectories of the dielectrophoresis problem

From Figure 3A it is hard to see the difference between the state and control trajectories, except that control trajectories are different near the final time. To investigate the impact of the error estimation, interval lengths can be plotted (Figure 3B). It can be seen, that the length of the intervals seeks larger values, where the control profile is reached its upper limit.

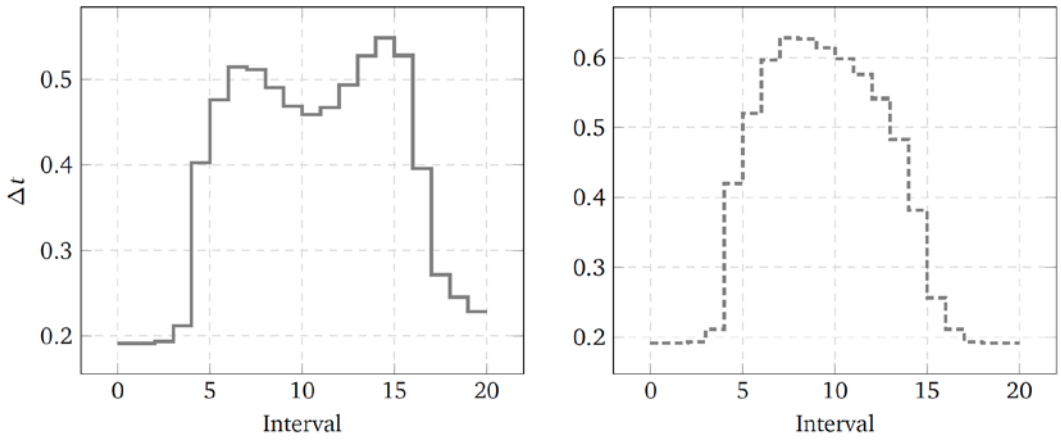

Figure 3B: Interval lengths of the dielectrophoresis problem without simulation (left) and with simulation (right)

To further analyze the problem, the convergence curves in terms of violation of the continuity condition and objective function value can be compared with and without error estimation. This situation is illustrated in Figure 4, where $max\left\|X^{c}_{i-1,N_c} - X_{p,i}\right\|$ means the maximum deviation at the interval boundaries over the entire optimization horizon, where $i = 1, \ldots, N$. It can be seen that the continuity conditions without simulating reaches the optimizer tolerance after about 20 iterations, while the optimal solution is finally find after 48 iterations. However, activating the error estimation the solution is obtained only after 80 iterations.

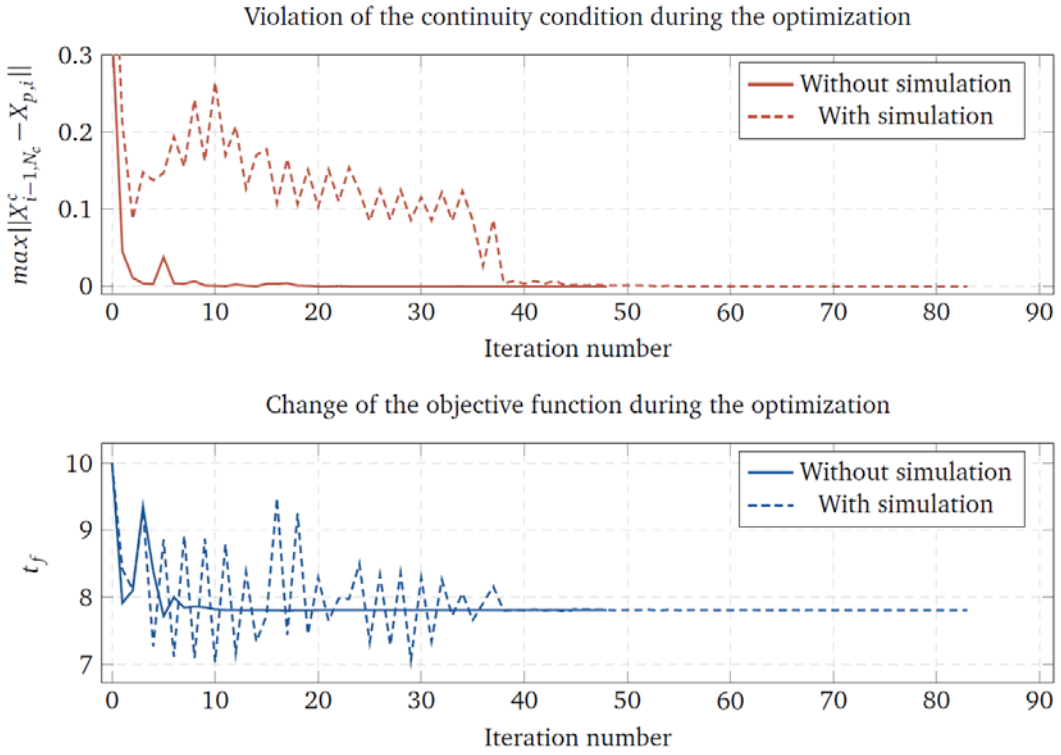

Figure 4: Continuity condition and objective function trajectory during the optimization

With activated simulation, the improvement of the state approximation in certain intervals breaks down the continuity condition. However, the continuity conditions are recalculated and the violation of these conditions is greater in magnitude. As a result, the convergence rate is slow, as indicated in Figure 4. This negative impact of the error estimation should be further analyzed in more detail to give the possibility of online applications.

2.4. Parameter estimation of the Quad-Tank problem

This model is taken from JModelica.org repository. Here a quadruple tank system is considered. The problem formulation can be found in appendix A.1.4. The CMSC framework solves this problem in 5 iterations within 0.289 seconds and the estimated values of the parameters can be found in Table 2. In comparison, the pure collocation method can solve this problem also in 5 iterations but the computation time is 1.556 seconds.

Table 2. Estimated parameters of the Quad-Tank problem

Parameter	CMSC method	Collocation method
a1	0.02656026 cm²	0.02659711 cm²
a2	0.02701864 cm²	0.02705287 cm²
a3	0.03005579 cm²	0.03006576 cm²
a4	0.0293089 cm²	0.02933232 cm²

The results in Table 2 conclude that CMSC method can efficiently solve parameter estimation problems.

3. Conclusions

This report describes obtained results of the deliverable D5.2.1. The prototype version of the new optimization toolchain for JModelica.org has been developed and tested using different types of optimization problems. The computational burden of the sequential computation in this toolchain has been avoided by applying parallel processing to speed up the whole optimization process. The adaptive discretization scheme has been implemented and tested. This work poses a great challenge and should be further improved. Using a new interface the optimization problem can be transferred directly from JModelica.org compiler, without creating FMUX, or XML files. Also for developing a CMSC toolchain, the new JModelica.org interface is more transparent. The CMSC method is extended to provide analytical second-order sensitivities. The CMSC framework has been successfully tested on the large-scale problem, i.e. a combined cycle power plant start-up, which is developed by Siemens AG, Erlangen, Germany. The resulted NLP problem can be solved within several seconds and applying parallel computing the computation time is significantly reduced. Additional contribution is a control-variable correlation analysis for efficient solution of dynamic optimization problems.

The pure collocation framework, which is already available in JModelica.org is already well documented and the user manual is available online from the official web page of JModelica.org.

4. References

[1] Magnusson, F.; Åkesson, J. Collocation Methods for Optimization in a Modelica Environment, Proceedings of the 9th International Modelica Conference, Munich, Germany, September 2012.

[2] Tamimi, J.; Li, P. A Combined approach to nonlinear model predictive control of fast systems. J. Process Control, 20(2010)9, pp. 1092-1102.

[3] Tamimi, J. Development of the Efficient Algorithms for Model Predictive Control of Fast Systems. PhD Thesis, Technische Universität Ilmenau, VDI Verlag, 2011.

[4] Lazutkin, E.; Geletu A.; Hopfgarten, S.; Li, P. Modified Multiple Shooting combined with Collocation method in JModelica.org with Symbolic Calculations, 10th International Modelica Conference, DOI 10.3384/ECP14096999 Lund, Sweden, 2014, pp. 999-1006.

[5] Vassiliadis, V. S.; Balsa-Canto, E.; Banga, J. R. Second-order sensitivities of general dynamic systems with application to optimal control problems, Chem. Eng. Sci., 1999, 54(17), 3851-3860.

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

[7] de Boor, C. Conference on the numerical solution of differential equations, Lecture Notes in Mathematics, 1974.

[8] Russel, R.D.; Christiansen, J. Adaptive mesh selection strategies for solving boundary value problems, SIAM, 15(1), 59-80, 1978.

[9] Dodson, D.S. Optimal order approximation by polynomial spline functions, PhD Thesis, Purdue University, 1972.

[10] Pereyra, V.; Sewell, E.G. Mesh selection for discrete solution of boundary value problems in ordinary differential equations, Numer. Math., 1975, 23, 19-39.

[11] White Jr., A.B. On selection of equidistributing meshes for two-point boundary-value problems, 1976.

[12] Russell, R.D.; Christiansen, J. Error analysis for spline collocation methods with application to knot selection, Mathematics of Computation, 32(142), 415-419, 1978.

[13] Renfro, J.G.; Morshedi, A.M.; Asbjornsen, O.A. Simultaneous optimization and solution of systems described by differential/algebraic equations. Comp. chem. Eng., 11(5), 503-5017, 1987.

[14] Vasantharajan, S.; Biegler, L.T. Simultaneous strategies for optimization of differential-algebraic systems with enforcement of error criteria. Comp. chem. Eng., 10, 1083-1100, 1990.

[15] Tanartkit, P.; Biegler, L.T. A nested, simultaneous approach for dynamic optimization problems- II: the outer problem. Comp. Chem. Eng, 22(12), 1365–1388, 1997.

[16] Bartl, M., Pu, L., Biegler, L. T.: Improvement of State Profile Accuracy in Nonlinear Dynamic Optimization with the Quasi-Sequential Approach, AIChE Journal, 57(2011), pp. 21852197.

[17] Hong, W., Wang, S., Li, P., Wozny, G., Biegler, L. T.: A quasi-sequential approach to large-scale dynamic optimization problems. AIChE Journal, 52(2006)1, pp. 255-268.

[18] Andersson, J.; Åkesson, J.; Diehl, M. CasADi: A symbolic package for automatic differentiation and optimal control, Lect. Notes Comput. Sci. Eng., 2012, 87, 297-307.

[19] Wächter, A.; Biegler, L. T. On the implementation of a primal-dual interior point filter-line search algorithm for large-scale nonlinear programming, Math. Prog., 2006, 106(1), 25-57.

[20] Luus, R. Application of dynamic programming to high-dimensional non-linear optimal control problems, Inter. J. of Contr., 1990, 52(1), 239-250.

[21] Balsa-Canto, E.; Banga, J. R.; Alonso, A. A.; Vassiliadis, V. S. Dynamic optimization of chemical and biochemical processes using restricted second-order information, Comput. Chem. Eng., 2001, 25(4-6), 539-546.

[22] Lazutkin, E; Geletu, A; Hopfgarten, S; Li, P. An analytical Hessian and parallel computing approach for efficient dynamic optimization based on control-variable correlation analysis, Industrial & Engineering Chemistry Research, 2015, DOI: 10.1021/acs.iecr.5b02369.

[23] Chang, D.E.; Petit, N.; Rouchon, P. Time-optimal control of a particle in a dielectrophoretic system. IEEE Transactions on Automatic Control, (51), 2006.

A. Appendix

A.1. Used abbreviations and symbols

A.1.1. List of variables

$x(t)$	Vector of differential states
$x(t_f)$	Vector of terminal values of differential states
$x(t_0), x_0$	Vector of initial values of differential states
$\dot{x}(t)$	First-order derivative of differential states
$z(t)$	Vector of algebraic states
$u(t)$	Vector of control variables
t	Time
t_0	Start time
t_f	Final time
V	Vector of parameterized controls
X^p	Vector of parameterized differential states
X^c	Vector of collocated differential and algebraic states
$X^{c,d}$	Vector of collocated differential states
x_{min}, x_{max}	Lower and upper bound for differential states
z_{min}, z_{max}	Lower and upper bound for algebraic variables
u_{min}, u_{max}	Lower and upper bound for control variables
N_c	Number of collocation points
N	Number of intervals
N_{XZ}^X	Number of equations per shooting interval for second-order sensitivities with respect to parameterized states
N_{XZ}^U	Number of equations per shooting interval for second-order sensitivities with respect to parameterized control variables
N_{xz}	Number of equations per shooting interval
n_x	Number of the differential states
n_z	Number of the algebraic states
n_u	Number of the control variables
$\Re^{n_x}$	Function space of the state vector
$\Re^{n_z}$	Function space of the algebraic vector
$\Re^{n_u}$	Function space of the control vector
$e_{i,nonc}$	Estimated error
c_{nonc}	Help constant
τ_{nonc}	Time at non-collocation point
$R_{i,nonc}^j$	Residual in i-th interval for j-th state

A.1.2. List of functions

f	Differential-algebraic equation system
g	Path constraints
G	Discretized path constraints
J	Objective function
M	Mayer term
L	Lagrange term
Φ	Function for second-order sensitivities with respect to differential states
Ψ	Function for second-order sensitivities with respect to control variables

A.1.3. List of abbreviations

NMPC	Nonlinear model predictive control
CMSC	Combined multiple shooting with collocation method
DAEs	Differential-algebraic equations
NLP	Nonlinear programming
AH	Analytical Hessian
BFGS	Broyden-Fletcher-Goldfarb-Shanno method
CCPP	Combined cycle power plant

A.1.4. Quad-Tank problem formulation

```
package QuadTankPack
model QuadTank

parameter Modelica.SIunits.Area A1=4.9e-4,A2=4.9e-4,A3=4.9e-4,A4=4.9e-4;
parameter Modelica.SIunits.Area a1=0.3e-5,a2=0.3e-5,a3=0.3e-5,a4=0.3e-5;
parameter Modelica.SIunits.Acceleration g = 9.81;
parameter Real k1_nmp=0.56e-6,k2_nmp=0.56e-6;
parameter Real g1_nmp=0.30,g2_nmp=0.30;

Modelica.SIunits.Length x1(start=0.0627155, fixed=true);
Modelica.SIunits.Length x2(start=0.06043781, fixed=true);
Modelica.SIunits.Length x3(start=0.02394927, fixed=true);
Modelica.SIunits.Length x4(start=0.02324908, fixed=true);

connector InputVoltage = input Modelica.SIunits.Voltage;
InputVoltage u1;
InputVoltage u2;

equation

der(x1)=-a1/A1*sqrt(2*g*x1) + a3/A1*sqrt(2*g*x3) + g1_nmp*k1_nmp/A1*u1;
der(x2)=-a2/A2*sqrt(2*g*x2) + a4/A2*sqrt(2*g*x4) + g2_nmp*k2_nmp/A2*u2;
der(x3)=-a3/A3*sqrt(2*g*x3) + (1-g2_nmp)*k2_nmp/A3*u2;
der(x4)=-a4/A4*sqrt(2*g*x4) + (1-g1_nmp)*k1_nmp/A4*u1;

end QuadTank;

optimization QuadTank_ParEst2 (objective=sum((y1_meas[i] - x1(t_meas[i]))^2
+ (y2_meas[i] - x2(t_meas[i]))^2 + (y3_meas[i] - x3(t_meas[i]))^2 +
```

```
(y4_meas[i] - x4(t_meas[i]))^2 for i in 1:N_meas),startTime=0,finalTime=60)

extends QuadTank(
a1(free=true,initialGuess=0.3e-5,nominal=0.3e-5,min=0,max=0.1e-4),
a2(free=true,initialGuess=0.3e-5,nominal=0.3e-5,min=0,max=0.1e-4),
a3(free=true,initialGuess=0.3e-5,nominal=0.3e-5,min=0,max=0.1e-4),
a4(free=true,initialGuess=0.3e-5,nominal=0.3e-5,min=0,max=0.1e-4));

parameter Integer N_meas = 61;
parameter Real t_meas[N_meas] = 0:60.0/(N_meas-1):60;
parameter Real y1_meas[N_meas] = ones(N_meas);
parameter Real y2_meas[N_meas] = ones(N_meas);
parameter Real y3_meas[N_meas] = ones(N_meas);
parameter Real y4_meas[N_meas] = ones(N_meas);

end QuadTank_ParEst2;

end QuadTankPack;
```