Defining and Solving Hybrid Optimal Control Problems with Higher Index DAEs

Radosław Pytlak[1] Damian Suski[2] Tomasz Tarnawski[3] Tomasz Zawadzki[4]

[1]Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland,
r.pytlak@mini.pw.edu.pl
[2]Institute of Automatic Control and Robotics, Warsaw University of Technology, Poland,
{d.suski,t.zawadzki}@mchtr.pw.edu.pl
[3]Department of Quantitative Methods and Information Technology, Kozminski University, Poland,
ttarnawski@kozminski.edu.pl
[4]Research and Academic Computer Network (NASK), Poland,
tomasz.zawadzki@nask.pl

Abstract

The paper deals with optimal control problems defined for hybrid systems described by higher index DAEs. We present a prototype solution that supports the whole process from defining such problem to solving it and presenting results. Problem's definition is done with Dynamic Optimization Modeling Language (DOML) which is based directly on Modelica. The proposed numerical procedure for solving the problems of interest has the following features: 1) it is based on the appropriately defined adjoint equations formulated for the discretized equations being the result of the numerical integration of system equations by an implicit Runge–Kutta method; 2) initialization for higher index DAEs is performed with the help of Pantelides' algorithm; 3) it does not require the system to be transformed to ODEs (through differentiation of some algebraic equations).

The paper presents numerical examples related to hybrid systems described by index three DAEs, showing the validity of the proposed approach. All software components needed to carry out the computations, i.e. the code editor, compiler, numerical libraries and GUI for presenting results are prepared as parts of a combined platform: Interactive Dynamic Optimization Server (IDOS).

Keywords: hybrid systems, optimal control problems, higher index DAEs

1 Introduction

The paper presents recent development of solver functionality implemented within DOML (Dynamic Optimization Modeling Language) environment and deployed as part of the IDOS (Interactive Dynamic Optimization Server, described in (Pytlak et al., 2014), see also (Pytlak et al., 2013)) infrastructure. It uses a numerical procedure based on control vector parameterization and RADAU5 together with event (discrete state transition) handling and is capable of solving optimal control problems for hybrid, high-index DAEs.

The DOML language (introduced in (Pytlak et al., 2014)) was devised an as extension of Modelica towards defining optimal control problems for systems described with Modelica language – quite analogically to Optimica (proposed earlier, in (Åkesson, 2007), see also (Åkesson, 2008)). In fact, the DOML compiler environment is heavily based on the open source Modelica (and Optimica) compiler environment JModelica.org (see e.g. (Åkesson et al., 2009)). During the efforts of adapting Optimica for the purpose of deploying it within IDOS environment a conclusion was reached to redesign some of its optimization-related constructs, as its original design brought in some troublesome limitations (details can be found e.g. in (Pytlak et al., 2013) and (Tarnawski and Pytlak, 2014)). To avoid confusion with Optimica, we then chose to refer to the language as DOML. Although the framework is (eventually) intended to be fully compatible with Modelica, the current development efforts are focused strictly on building prototype, proof-of-concept implementations of advanced optimization algorithms. Therefore, ensuring DOML's wide and flawless compatibility with Modelica syntax (and MSL models in particular) has to wait for its turn (still, being based on JModelica.org environment, DOML is in the position to enjoy a fair deal of compatibility inherited 'in the package'). Up to this point several different optimization algorithms and solver libraries have already been implemented (see Table 1 in (Pytlak et al., 2014)): e.g. solvers for optimal control problems with ODEs based on *a'priori* discretization of system equations (HQP package), solvers that use adjoint equations and do not require *a'priori* discretization of equations, solvers based on multiple shooting methods, solvers for control problems with integer valued controls which use BONMIN package, solvers that use integration procedures from SUNDIALS package and IPOPT as the optimization engine.

The new algorithm implementation presented here is designed to solve optimal control problems defined by hybrid systems, i.e. systems with mixed discrete-continuous

dynamics (van der Schaft and Schumacher, 2000), whose dynamics are described with high-index DAEs. It must be noted, that even separately each of these categories poses nontrivial difficulties for optimization and even for simulation alone. In particular:

- higher index DAE systems need the procedure for the automatic consistent initialization, the integration procedure for higher index DAEs and the optimization solver equipped with the procedure for evaluating gradients of functionals defining optimization problems

- hybrid systems require the procedure for the accurate location of discrete transition times, the integration procedure with an automatic handling of discrete dynamics an the optimization solver equipped with the procedure for evaluating gradients of optimization problem functionals, which is dedicated for a hybrid system dynamics.

The authors are not aware of any implementations (other then the one introduced in (Pytlak, 2011)) of a dynamic optimization algorithm capable of reliably solving high-index DAE problems without an application of the index reduction procedure. Similarly, the authors are not aware of an established and widely-used implementation of a dynamic optimization algorithm applicable to hybrid systems. In particular, the optimization algorithms implemented in environments featuring Optimica (JModelica.org, OpenModelica) cover the problems described by ODEs (or DAEs with the help of the index reduction) but not hybrid systems.

The paper is organized as follows. In section 2 we outline the features of DOML and in particular the language constructs used to express the problems of interest. They are then used in section 3 to define two simple exemplary problems used to test and illustrate capabilities of the implemented solver. Trajectories obtained by solving these problems are also presented. The following two sections are devoted to formal description and analysis of the algorithm. First, in section 4, a formal definition of hybrid optimal control problem is laid out (and used to formally re-define the example problems). Then, based on that, section 5 discusses the most important mathematical and implementational details of the proposed optimization procedure. The paper is wrapped up with short concluding remarks.

2 Hybrid optimal control problem definition in DOML

The provisions of DOML that differentiate it against Optimica were already described in earlier works (e.g.: (Pytlak et al., 2014), (Tarnawski and Pytlak, 2014) or (Pytlak et al., 2013)). To avoid excessive repetition, here they are only listed very briefly:

- new keywords, **`minimize`** and **`maximize`** were introduced making it possible to have meaningfully named optimized parameters with the direction of optimization chosen by the user. It also became possible to specify multicriteria optimization problems.

- **`annotation`**`(solver)` was used to allow the user to specify the algorithm to be used to solve the given problem, along with its runtime settings. A preliminary procedure of choosing the most appropriate solver, based on elements detected in the problems's definition, was also implemented.

- a mechanism for labeling equations and constraints was introduced, by means of which the adjoint variables (for equations) and Lagrange multipliers (for constraints) could be referred to. Implementation of some solvers required that functionality.

- decision variable's `InitialGuess` attribute was redefined with *continuous* variability, so that it became possible for initial guess to be defined as a signal changing over time. This was developed particularly with 'chaining of solvers' in mind: an approximate solver could be run first, then the solution obtained could be used to warm-start another solver – a more exact one, yet also more fussy with respect to the starting point.

For the most part, these earlier provisions were well geared for formally defining the optimal control problems of interest. In the case of hybrid systems, however, we found it necessary to devise specific syntactic constructs to fully express the system's dynamics. The resulting 'canonical form' description of a hybrid system is structured around:

- singular **`enumeration`** type variable whose values range over possible discrete states of the system;

- compound **`if elseif`** statement containing equations defining the dynamics specific for each state;

- embedded **`when`** clauses specifying *transition guards*—conditions for exiting the current state and the new state reached with the transition;

- optional `reinit` operator used to define any potential discontinuous behavior upon a change–of–state event.

In result, the description of a hybrid system may take the form along the lines drawn on Listing 1. Please note, that the presented code is meant as a mere example and does not necessarily reflect sensible dynamics of any actual system.

In a general case the state-transition conditions (guards) between a pair of states do not have to be the same for both directions. One good example, when this is certainly not the case, is hysteresis (in the example shown: between

DOI 10.3384/ecp17132265

Listing 1. Example of DOML code defining a hybrid system

```
optimization Hybrid_ex(startTime = 0.0,
                       finalTime = 1.0)
  minimize Real objective = x3(finalTime);
   ... // decl's of vars (x) and inputs (u)
  type Q = enumeration(A, B, C);
  discrete Q q(start = Q.A);  // the state
equation
    ... // equations common to all states
  if q == Q.A then
    der(x1) = -x1 + x2 + 2*u;
    when x1 < -1 then      //transition A->B
      q = Q.B;
    end when;
  elseif q == Q.B then
    der(x1) = -2*x1 +x2 + u;
    when x1 > 1 then       //transition B->A
      q = Q.A;
    end when;
  else  // q == Q.C
    der(x1) = -x2 + u;
    when x2 < 0 then      //bounce-back C->C
      q = Q.C;
      reinit(x3, -x3);
    elsewhen x2 > 10 then //transition C->A
      q = Q.A;
    end when;
  end if;
end Hybrid_ex;
```

states A and B). In addition, with the proposed construct, it is possible to define transition from a state onto itself (in the example: happens in C) which is applicable for instance in the case of a ball bouncing off a wall, back into the realm of the same dynamics, but with it's velocity (abruptly) altered (hence the reinit placed in the example).

3 Examples

Below, we present the results of applying the presented algorithm to solving two optimal control problems based on hybrid systems. They have been previously discussed in (Pytlak and Suski, 2017), but here we show results for different versions of these problems. Please note, that the purpose of these examples is purely illustrative within the discussion of capabilities of the implemented solver and they do not necessarily represent sensible problems of real engineering importance or application. Also, as noted earlier, the algorithm's implementation is at the stage of proof-of-concept and has only been tested on cases with limited complexity, where the whole problem is defined in one stand-alone input file. Defining optimization problems through models constructed with MSL components was not tested (however, the compiler is build on top of the JModelica.org compiler, which itself provides a substantial MSL support).

Below, we define the exemplary optimal control problems solely be means of DOML code. In the following section, after introducing the necessary formalism, we restate them in a mathematically formal way.

Example 1. It is an optimal control problem of a non-standard pendulum described by DAEs with index three (van der Schaft and Schumacher, 2000), in Cartesian coordinates x_1 and x_2. On the vertical axis there is a fixed pin interfering with pendulum's string and effectively halving its length when $x_1 \geq 0$. For $x_1 \leq 0$ the pendulum swings with its original length. There is no jump in differential state variables during state transition. The control variable u represents force applied horizontally to the pendulum's end (constrained above by 0.3, in either direction). The objective of the control is to cause the system to reach such final state in which the pendulum's position $x_1(t_f)$ is as close as possible to the neutral point, while its velocity component $v_2(t_f)$ is equal to 0.06. The DOML definition of the problem is presented in Listing 2.

After 10 iterations optimality conditions were satisfied with accuracy 10^{-6} while equality constraint $v_2(t_f) = 0.06$ with accuracy 10^{-7}. The obtained optimal trajectories are given in Figure 2 while the optimal control in Figure 1 – as it turns out, it exhibits a relatively complex switching structure.

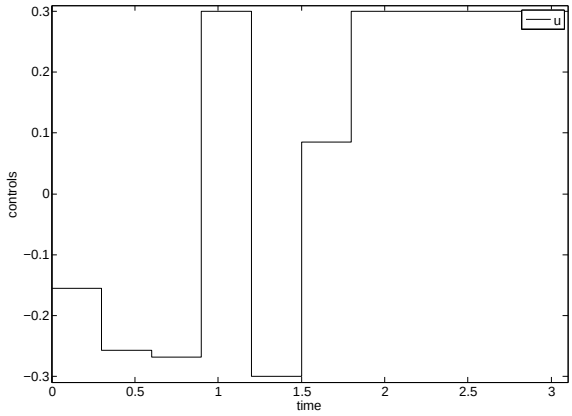

Figure 1. The pendulum example—optimal control.

Example 2. The second example is the popular bouncing ball problem discussed in several papers e.g. (van der Schaft and Schumacher, 2000). The ball bounces off the fixed surface level at $x_1 = 0$ with the opposite velocity (assuming no energy loss, i.e. the coefficient of restitution is 1). The system has only one discrete state. The control variable u is a force applied vertically to the ball (with the constraint that it cannot exceed the value of 2.5, either upwards or downwards). The objective is to end up (at $t_f = 1$) with the ball being at the height of 0.5 and having minimum velocity (in terms of its absolute value). The DOML script of the problem is presented in Listing 3.

Applying the SQP code resulted, after 23 iterations, in an approximate optimal solution given in Figure 3. The

```
package Pendulum

  optimization Pendulum_opt(startTime = 0,
      finalTime = 3.0)

    minimize Real obj = x1(finalTime)^2;

    type states =
              enumeration(short, normal);
    discrete states State;
    parameter Real p = 0.5;
    ... // decl's of vars (x1, x2, ... L)
    input Real u1(min=-0.3, max=0.3,
        initialGuess=0.0);

  initial equation
    if x1 >= 0.0 then
      State = states.normal;
    else
      State = states.short;
    end if;

  equation
    der(x1) = v1;
    der(x2) = v2;
    der(v1) = w1;
    der(v2) = w2;

    if State == states.normal then
      0 = w1+x1*L-u1;
      0 = w2+1.0+x2*L;
      0 = x1*x1+x2*x2-1.0;
      when x1 < 0.0 then
        State = states.short;
      end when;

    else
      0 = w1+x1*L/p-u1;
      0 = w2+1.0+(x2+1.0-p)*L/p;
      0 = x1*x1+(x2+1.0-p)*(x2+1.0-p)-p*p;
      when x1 > 0.0 then
        State = states.normal;
      end when;

    end if;

  constraint
     c1: v2(finalTime) = 0.06;

  end Pendulum_opt;

end Pendulum;
```

Listing 2. The DOML file of the pendulum problem.

```
package Bouncing
  optimization Bouncing_opt (startTime =
      0.0, finalTime = 1.0)

    minimize Real obj = x2(finalTime)^2;

    type states = enumeration(normal);
    discrete states State;

    parameter Real c = 5.0;
    Real x1(start = 1.0);
    Real x2(start = 0.0);

    input Real u(min=-2.5,max=2.5,
        initialGuess=0.001);

  initial equation
    State = states.normal;

  equation
    der(x1) = x2;
    der(x2) = -c + u;

    if State == states.normal then
      when x1 < 0.0 then
        State = states.normal;
          reinit(x2, -pre(x2));
      end when;
    else
      State = states.normal;
    end if;

  constraint
    c1: x1(finalTime) = 0.5;

  end Bouncing_opt;

end Bouncing;
```

Listing 3. The DOML file of the bouncing ball problem.

DOI 10.3384/ecp17132265

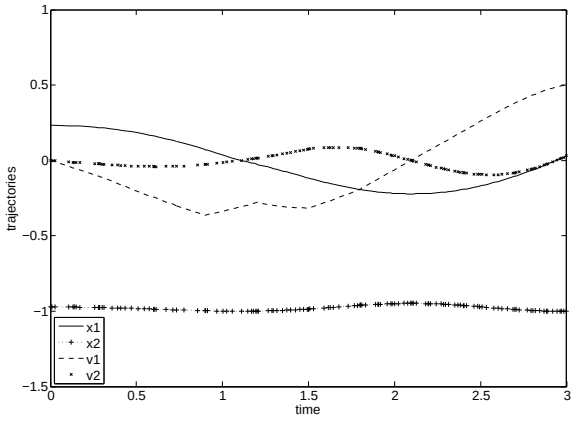

Figure 2. The pendulum example—optimal trajectories.

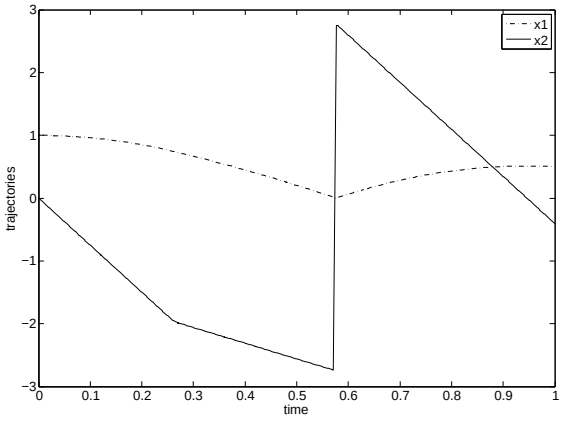

Figure 4. The bouncing ball example—optimal trajectories.

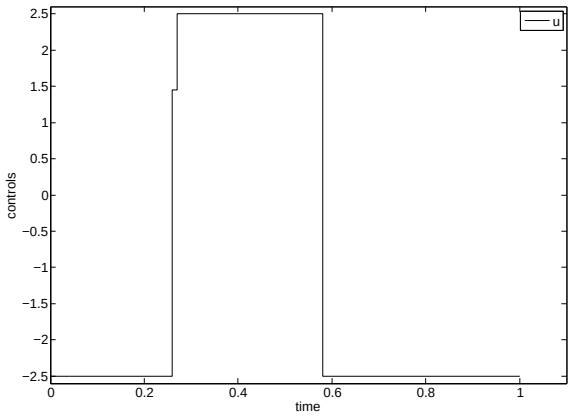

Figure 3. The bouncing ball example—optimal control.

optimality conditions and equality constraint $x_1(t_f) = 0.5$ were satisfied with accuracy 10^{-6}. Some of the final trajectories are illustrated in Figure 4. At final time the ball is at the required height, and has a relatively small velocity, $|x_2(t_f)| \approx 0.3$.

4 Formal discussion of hybrid systems

Hybrid systems are systems with mixed discrete-continuous dynamics (van der Schaft and Schumacher, 2000). In this work we use the formal definition of a hybrid system based on the one given in (Suski and Pytlak, 2016), which is similar to many other definitions given in the literature e.g. (Lygeros et al., 1999), (van der Schaft and Schumacher, 2000), (Shaikh, 2004). We restrict our analysis to systems with autonomous transitions.

A hybrid system $\mathscr{H}$ is a tuple

$$\mathscr{H} = (\mathscr{Q}, \mathscr{U}, \mathscr{I}, \mathscr{F}, \mathscr{T}, \mathscr{G}, \mathscr{J}) \tag{1}$$

where

- $\mathscr{Q}$ is a finite set of discrete states. Its elements are denoted by q.
- $\mathscr{U}$ is a set of admissible controls. The elements of $\mathscr{U}$ are measurable functions $u : I \to U$, where I can be any closed interval of $\mathbb{R}$ and U is a fixed subset of $\mathbb{R}^m$.
- $\mathscr{I}$ is a function which assigns to every discrete state q a set

$$\mathscr{I}(q) = \left\{x \in \mathbb{R}^n : \psi_q(x) \leq 0\right\},\ \psi_q : \mathbb{R}^n \to \mathbb{R}^{n_{\psi_q}} \tag{2}$$

  such that as long as a hybrid systems is in a discrete state q the continuous state trajectory x stays in $\mathscr{I}(q)$. We therefore say that $\mathscr{I}(q)$ is an *invariant set* for a discrete state q.
- $\mathscr{F}$ is a function which assigns to every discrete state q a function $F_q : R^n \times \mathscr{I}(q) \times U \to \mathbb{R}^n$ such that in a discrete state q the continuous state evolves according to a differential-algebraic equation

$$F_q(\dot{x}, x, u) = 0. \tag{3}$$

- $\mathscr{T}$ is a subset of $\mathscr{Q} \times \mathscr{Q}$, which collects all pairs of discrete states (q, q') such that the transition from a state q to a state q' is possible.
- $\mathscr{G}$ assigns to each pair $(q, q') \in \mathscr{T}$ a subset of $\mathscr{I}(q)$ boundary such that when a continuous state trajectory is about to leave $\mathscr{I}(q)$ through its boundary $\partial\mathscr{I}(q)$ at a point $x_t \in \mathscr{G}(q, q') \subset \partial\mathscr{I}(q)$ a discrete state changes from q to q'. We call such an event a *transition* and $\mathscr{G}$ plays a role of a *transition guard*.
- $\mathscr{J}$ assigns to each pair $(q, q') \in \mathscr{T}$ a function $\chi_{qq'} : \mathscr{G}(q, q') \to \mathscr{I}(q')$ such that when a discrete state

changes from q to q' at a transition time instant t_t then a continuous state undergoes a jump according to

$$x(t_t^+) = \chi_{qq'}(x(t_t^-)). \tag{4}$$

Having the definition of hybrid systems we can formally state the optimal control problem with hybrid system as:

$$\min_u \phi(x(t_f)) \tag{5}$$

subject to the constraints:

$$\begin{aligned} h_i^1(x(t_f)) &= 0,\ i \in E && (6)\\ h_j^2(x(t_f)) &\le 0,\ j \in I && (7)\\ u(t) &\in \Omega \text{ a.e. } t \in T. && (8) \end{aligned}$$

where the continuous dynamics of a hybrid systems is described by a system of higher index differential–algebraic equations (DAEs)

$$F_q(\dot{x}, x, u, t) = 0 \text{ a.e. } t \in T = [0, t_f] \tag{9}$$

which depends on the actual discrete state q.

We assume that $u(t) \in \mathscr{R}^m$, $x(t) \in \mathscr{R}^n$, I, E are finite sets of indices and Ω is a convex bounded subset of $\mathscr{R}^m$. A more general problem with parameters as decision variables, with an unspecified time horizon and with the state constraints could also be considered but for the simplicity of presentation we analyze the problem stated above.

We also assume that a solution to system (9) exists and is unique for any $u \in \mathscr{U}$ where

$$\mathscr{U} = \{u \in \mathscr{L}_1^m[T] \mid u(t) \in \Omega \text{ a.e. on } T\}, \tag{10}$$

and any consistent initial condition $x(0)$.

Example Problems restated

Having defined the above one can transcribe the earlier examples in a mathematically formal way. The hybrid optimal control problem from the first example (pendulum) may be stated as follows:

$$\min_{u \in \mathscr{U}} \left[x_1(t_f)^2\right] \tag{11}$$

subject to the constraints

$$v_2(t_f) - 0.06 = 0 \tag{12}$$

and

$$\begin{aligned} \dot{x}_1 &= v_1 && (13)\\ \dot{x}_2 &= v_2 && (14)\\ \dot{v}_1 &= w_1 && (15)\\ \dot{v}_2 &= w_2 && (16)\\ 0 &= w_1 + x_1 L/p - u && (17)\\ 0 &= w_2 + 1 + (x_2 + 1 - p)L/p && (18)\\ 0 &= x_1^2 + (x_2 + 1 - p)^2 - p^2 && (19) \end{aligned}$$

with

$$\mathscr{U} = \left\{u \in \mathscr{L}_1^1[0, t_f] \mid -0.3 \le u(t) \le 0.3 \text{ a.e. on } [0, t_f]\right\}$$

and $t_f = 3.0$. The parameter p satisfies $p = 1$ for $x_1 \le 0$ and $p = 0.5$ for $x_1 \ge 0$. The hybrid system has therefore two discrete states q^1 and q^2 with invariant sets: $\mathscr{I}(q^1) = \{(x_1, x_2, v_1, v_2, w_1, w_2, L) \in \mathscr{R}^7 : x_1 \le 0\}$, $\mathscr{I}(q^2) = \{(x_1, x_2, v_1, v_2, w_1, w_2, L) \in \mathscr{R}^7 : x_1 \ge 0\}$.

In turn, the optimal control problem given as the second example (bouncing ball) can be formally defined as: The optimal control problem transcribed formally is as follows

$$\min_{u \in \mathscr{U}} \left[x_2(t_f)^2\right] \tag{20}$$

subject to the constraints

$$x_1(t_f) - 0.5 = 0 \tag{21}$$

and

$$\begin{aligned} \dot{x}_1 &= x_2 && (22)\\ \dot{x}_2 &= -5 + u && (23) \end{aligned}$$

with

$$\mathscr{U} = \left\{u \in \mathscr{L}_1^1[0, t_f] \mid -2.5 \le u(t) \le 2.5 \text{ a.e. on } [0, t_f]\right\}.$$

and $t_f = 1.0$. The jump function is given by

$$\mathscr{J}(x(t_t^-)) = \begin{bmatrix} x_1(t_t^+) \\ x_2(t_t^+) \end{bmatrix} = \begin{bmatrix} x_1(t_t^-) \\ -x_2(t_t^-) \end{bmatrix}. \tag{24}$$

The system has only one discrete state q with invariant set defined by $\mathscr{I}(q) = \{x \in \mathscr{R}^2 : x_1 \ge 0\}$.

5 Numerical procedure

The utilized numerical procedure for solving hybrid optimal control problems include the mechanisms for

- numerical integration of system equations (3) between transitions
- proper handling of transitions—precise location of transition times and application of state jumps
- the calculation of adjoint variables for the evaluations of gradients of functionals defining the problem.

The numerical procedure for solving optimal control problems described by higher index DAEs was introduced in (Pytlak, 2011). For the completeness of a presentation the essential ingredients of the procedure are also discussed here.

Suppose that we want to integrate numerically the set of differential–algebraic equations

$$F(\dot{x}, x, u) = 0 \tag{25}$$

DOI 10.3384/ecp17132265

by an implicit Runge–Kutta method. If we denote by $x(k)$ the value of x calculated at the kth integration step (at a time t_k), then the next value $x(k+1)$ is evaluated by solving the set of nonlinear equations

$$F(\dot{x}_i(k+1), x(k) + h(k)\sum_{j=1}^{s} a_{ij}\dot{x}_j(k+1), u(k)) = 0, \quad (26)$$

$$x(k+1) - x(k) - h(k)\sum_{i=1}^{s} b_i\dot{x}_i(k+1) = 0, \quad (27)$$

where coefficients a_{ij}, b_i depend on a Runge–Kutta method.

In order to define state equations of the discrete time system (26)–(27) we introduce the state vector $X(k)$:

$$X(k) = \begin{bmatrix} \dot{x}_1(k) \\ \vdots \\ \dot{x}_s(k) \\ x(k) \end{bmatrix}, \quad (28)$$

then equations (26)–(27) become

$$\tilde{F}(X(k+1), X(k), u(k)) = 0. \quad (29)$$

System (29) is fully implicit and, under some nonsingularity assumption, can be expressed as explicit. If the Jacobian of $\tilde{F}$ with respect to $X(k+1)$, denoted by $\tilde{F}_{X^+}$, exists and is nonsingular for all $k = 0, \ldots, N-1$, then from the Implicit Function Theorem there exists unique function φ such that

$$X(k+1) = \varphi(X(k), u(k)) \quad (30)$$

and

$$\tilde{F}(\varphi(X(k), u(k)), X(k), u(k)) = 0 \quad (31)$$

for $k = 0, \ldots, N-1$.

Under differentiability assumptions imposed on $\tilde{F}$ the function φ is differentiable with respect to $X(k)$ and $u(k)$ and we have

$$\varphi_X(k) = -\left[\tilde{F}_{X^+}(k)\right]^{-1}\tilde{F}_X(k) \quad (32)$$

$$\varphi_u(k) = -\left[\tilde{F}_{X^+}(k)\right]^{-1}\tilde{F}_u(k). \quad (33)$$

where $\tilde{F}_{X^+}(k)$, $\tilde{F}_X(k)$, $\tilde{F}_u(k)$ are evaluated at a point $(X(k+1), X(k), u(k))$ and $\varphi_X(k)$, $\varphi_u(k)$ are evaluated at $(X(k), u(k))$.

If we consider the function

$$\hat{F}^0(u) = \phi(X^u(N)) \quad (34)$$

then its gradient can be calculated by referring to adjoint equations for the functional (34) and the system (30).

The adjoint equations for the functional (34) and the system (30) are considered, for example, in (Pytlak, 1999):

$$p(N) = \phi_X(X^u(N))^T \quad (35)$$

$$p(k) = \varphi_X(k)^T p(k+1), \quad (36)$$

for $k = 0, \ldots, N-1$. The adjoint variables p are the means for the gradient evaluation according to the formula

$$\hat{F}^0_{u(k)}(u) = \varphi_u(k)^T p(k+1). \quad (37)$$

Using (32)–(33), the adjoint equations (36) and the formula (37) can be expressed without the knowledge of φ:

$$p(k) = -\tilde{F}_X(k)^T\left[\tilde{F}_{X^+}(k)\right]^{-T} p(k+1) \quad (38)$$

$$\hat{F}^0_{u(k)}(u) = -\tilde{F}_u(k)^T\left[\tilde{F}_{X^+}(k)\right]^{-T} p(k+1). \quad (39)$$

Eventually we have the viable formula for the gradient of $\hat{F}^0(u)$ provided that matrices $\tilde{F}_{X^+}(k)$ are nonsingular.

For special (but widely used) forms of higher index DAEs the matrices $\tilde{F}_{X^+}(k)$ are **nonsingular** provided that $h(k)$ are sufficiently small. For example in (Hairer et al., 1989) it is shown that the statement holds for DAEs in the Hessenberg form up to an index three and the RADAU IIA integration scheme. Therefore, for many higher index DAEs we have a valid technique for computing gradients of $\hat{F}^0(u)$ (and other functions involved in an optimal control problem).

For the numerical integration, our software utilizes the RADAU5 code, which implements the RADAU IIA scheme with number of stages $s = 3$. The coefficients of a method and its good numerical properties are described in (Hairer et al., 1989). The numerical integration of DAEs will succeed if initial states are consistent. The initialization problem is solved in two steps. In the first step, the system of equations, required for the consistent initialization, is formed. This step is carried out with the help of Pantelides' graph based procedure (Pantelides, 1988) together with symbolic differentiation implemented within JModelica.org compiler. In the second step the system of equations is solved with the help of `IPOPT` solver (Wachter and Biegler, 2006). The consistent initialization procedure is called at the initial time and at points at which control functions exhibit jumps.

RADAU5 code requires the preliminary analysis of higher index DAEs: the user has to identify variables which have so–called index 1, index 2 and index 3 property. These indices can be established during the consistent initialization process—in our approach that information is passed from the procedure which implements the Pantelides' algorithm into the integration procedure. As a result the user of our package does not have to specify indices of equations variables in the DOML script.

In hybrid systems, one needs a procedure, which locates the transition times. To complete this task, our procedure uses subroutines `drchek.f` and `droots.f` (Hiebert and Shampine, 1980). The subroutine `drchek.f` does the preliminary check on the presence of a transition point. Next, the subroutine `droots.f` is called to locate the root with the specified accuracy. The root finding problem is solved with the help of the interpolation procedure which uses Hermite polynomials.

Between transitions, the adjoint equations are solved with the help of a formula (36), the same as for non–hybrid systems. However, the situation at a transition time is different. Let us denote the system equations before transition as

$$X(k+1) = \varphi^1(X(k), u(k), h(k)) \tag{40}$$

and the system equations after transition as

$$X(k+1) = \varphi^2(X(k), u(k), h(k)). \tag{41}$$

In the above equations the dependence of system equations on $h(k)$ is explicitly stated. The reason for that is the following. The step sizes nearby a transition step k_t are not taken freely, but they are taken to satisfy the transition condition

$$\psi(X^-(k_t)) = 0 \tag{42}$$

where $X^-(k_t)$ denotes the state vector before jump. The value of a state after jump is determined by the equation

$$X^+(k_t) = \chi(X^-(k_t)). \tag{43}$$

Now to calculate the adjoint variables at a transition the following linear system of equations needs to be solved for variables $p(k_t)$ and π

$$p(k_t) + \pi\left(\psi_X(k_t)\right)^T = \left(\chi_X(k_t)\right)^T \left(\varphi_X^2(k_t)\right)^T p(k_t+1) \tag{44}$$

$$p(k_t)^T \varphi_h^1(k_t-1) = p(k_t+1)^T \varphi_h^2(k_t). \tag{45}$$

The partial derivative $\varphi_h(k)$ is calculated with the formula

$$\varphi_h(k) = -\left[\tilde{F}_{X^+}(k)\right]^{-1} \tilde{F}_h(k) \tag{46}$$

analogical to formulas (32)-(33). The formulae (44)–(45) are derived in (Pytlak and Suski, 2017).

To solve the optimal control problem, we replace control functions by their piecewise constant approximations and follow the optimization procedure outlined below.

Optimization Procedure

1. Choose initial control u_0 and set the iterations number: $k = 0$.
2. For the control u_k, integrate system equations by calling consistent initialization procedure when necessary. Calculate the cost function and evaluate all constraint functions. Evaluate the adjoint equations for each function defining the optimal control problem and on that basis calculate the gradients of all functions involved in optimal control problem.
3. Perform the optimization step. If optimality conditions are satisfied with the assumed accuracy then STOP. Otherwise evaluate u_{k+1}, increase k by one and go to Step 2).

In our code the main optimization loop is handled by the SQP code described in (Pytlak, 1999). The optimization procedure requires solving QP subproblems at each iteration. The interior point method described in (Gertz and Wright, 2003) is used for that purpose.

6 Conclusions

The paper presents the most recent advancement in the DOML-IDOS environment, where a number of optimization algorithms can be used to solve optimal control problems specified in a language directly derived from Modelica (and Optimica). The latest numerical procedure, described here, implements an advanced algorithm for solving hybrid optimal control problems with higher index DAEs. The procedure is based on the `RADAU5` program which is the implementation of an implicit Runge–Kutta method. The procedure uses variable stepsizes in order to locate precisely switching points. The procedure is based on the adjoint equations associated with the discretized equations being the result of system equations integration and does not require the transformation of higher index DAEs to ODEs by performing differentiations of some system equations.

So far, the development focus was placed strictly on implementing proof-of-concept, prototype solutions based on most recent dynamic optimization algorithms while ensuring wide Modelica compatibility, environment robustness, more graceful error handling, etc. have been put on hold. The authors are well aware of the need to devote more attention to those lagging issues and indeed intend to do so.

References

J. Åkesson. *Tools and Languages for Optimization of Large-Scale Systems*. PhD thesis, Department of Automatic Control, Lund University, Lund, Sweden, 2007.

J. Åkesson. Optimica–an extension of Modelica supporting dynamic optimization. In *Proceedings of the 6th Modelica Conference*, Bielefeld, Germany, March 2008. Modelica Association.

J. Åkesson, M. Gäfvert, and H. Tummescheit. JModelica—an open source platform for optimization of Modelica models. In *Proceedings of MATHMOD 2009 - 6th Vienna International Conference on Mathematical Modelling*, Vienna, Austria, February 2009. TU Wien.

E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, (29):58–81, 2003.

E. Hairer, Ch. Lubich, and M. Roche. The numerical solution of differential-algebraic equations by Runge–Kutta methods. *Lecture Notes in Mathematics*, 1409:56–225, 1989.

K. L. Hiebert and L. F. Shampine. Implicitly defined output points for solutions of ode-s. Technical report, United States Department of Energy, Sandia Laboratories, 1980.

J. Lygeros, K. H. Johansson, S. Sastry, and M. Egerstedt. On the existence of executions of hybrid automata. In *Proceedings of the 38th IEEE CDC*, pages 2249–2254, Phoenix, AZ; USA, December 1999. IEEE.

C. C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, 1988. doi:10.1137/0909014.

R. Pytlak. *Numerical Methods for Optimal Control Problems with State Constraints*. Lecture Notes in Mathematics 1707. Springer Berlin Heidelberg, 1999. ISBN 9783540662143.

R. Pytlak. Numerical procedure for optimal control of higher index DAEs. *Discrete Contin. Dyn. Syst.*, 29(2):647–670, 2011. ISSN 1078-0947; 0133-0189; 1553-5231/e. doi:10.3934/dcds.2011.29.647.

R. Pytlak and D. Suski. On solving hybrid optimal control problems with higher index DAEs. *Optimization Methods and Software*, 2017. doi:http://dx.doi.org/10.1080/10556788.2017.1288730.

R. Pytlak, J. Błaszczyk, A. Karbowski, K. Krawczyk, and T. Tarnawski. Solvers chaining in the IDOS server for dynamic optimization. In *Proceedings of 52nd IEEE CDC*, pages 7119–7124, Florence; Italy, 2013. IEEE. ISBN 978-1-4673-5714-2. URL `http://dblp.uni-trier.de/db/conf/cdc/cdc2013.html#PytlakBKKT13`.

R. Pytlak, T. Tarnawski, B. Fajdek, and M. Stachura. Interactive Dynamic Optimization Server – connecting one modelling language with many solvers. *Optimization Methods and Software*, 29(5):1118–1138, 2014. doi:10.1080/10556788.2013.799159.

M. S. Shaikh. *Optimal Control of Hybrid Systems: Theory and Algorithms*. PhD thesis, McGill University, Montreal, Que., Canada, 2004. URL `http://digitool.library.mcgill.ca/R/?func=dbin-jump-full&object_id=85095&local_base=GEN01-MCG02`. AAINR06340.

D. Suski and R. Pytlak. The weak maximum principle for hybrid systems. In *Proceedings of the of the 24th IEEE MED*, pages 338–343, Athens; Greece, 2016. IEEE.

T. Tarnawski and R. Pytlak. DOML - a compiler environment for dynamic optimization supporting multiple solvers. In *Proceedings of the 10th International Modelica Conference*, pages 1095–1104, Lund; Sweden, 2014. Linköping University Electronic Press.

A. J. van der Schaft and J. M. Schumacher. *An Introduction to Hybrid Dynamical Systems*. Lecture Notes in Control and Information Sciences. Springer, 2000. ISBN 9781852332334.

A. Wachter and L. T. Biegler. On the implementation of an interior point line search flter algorithm for large scale nonlinear programming. *Mathematical Programming*, (106):25–57, 2006.