Keywords Mixed-Integer Optimal Control, Differential-Algebraic Equations, Domain Specific Languages, Mathematical Modeling
Abstract We describe a set of extensions to the AMPL modeling language to conveniently model mixed–integer optimal control problems for ODE or DAE dynamic processes. These extensions are realized as AMPL user functions and suffixes. Hence, no intrusive changes to the AMPL language standard or implementation itself are required. We describe and provide TACO, a toolkit for optimal control in AMPL that reads AMPL *stub.nl* files and detects the optimal control problem's structure. This toolkit is designed to facilitate the coupling of existing optimal control software packages to AMPL. We discuss requirements, capabilities, and the current implementation. Using the example of the multiple shooting code for optimal control MUSCOD-II, a direct and simultaneous method for DAE-constrained optimal control, we access the problem information provided by the TACO toolkit in order to interface this solver with AMPL. Moreover, we show how the MS-MINTOC algorithm for mixed–integer optimal control can be used to efficiently solve mixed–integer optimal control problems modeled in AMPL. Three exemplary control problems are modeled using the proposed AMPL extensions to discuss how these affect the representation of optimal control problems. Solutions to these problems are obtained using MUSCOD-II and MS-MINTOC inside the AMPL environment. A collection of further AMPL control models is provided on the web site `mintoc.de`. Using the TACO toolkit to enable input of AMPL models, MUSCOD-II and MS-MINTOC are made available on the NEOS Server for Optimization.

TACO — A Toolkit for AMPL Control Optimization

Christian Kirches · **Sven Leyffer**

September 28, 2011

1 Introduction

This paper is concerned with a set extensions to the AMPL modeling language that aims at extending AMPL's applicability to mathematical optimization problems including dynamics. The mission is to describe and implement AMPL syntax elements for convenient modeling of ODE- and DAE-constrained optimal control problems.

We consider the class (1) of mixed–integer optimal control problems on the time horizon $[0,T]$. Our goal is to minimize an objective function comprising an integral Lagrange term L on $[0,T]$, a Mayer term E at the end point T of the time horizon, and a point least-squares term with N_{lsq} possibly nonlinear residuals l_i on a grid $\{t_i\}_{1\leq i\leq N_{\mathrm{lsq}}}$ of time points. All objective terms depend on the trajectory $(x(\cdot),z(\cdot)) : [0,T] \to \mathbb{R}^{n_{\mathrm{x}}} \times \mathbb{R}^{n_{\mathrm{z}}}$ of a dynamic process described in terms of a system of ordinary differential equations (ODEs, 1b) or differential–algebraic equations (DAEs, 1b, 1c). This system is affected by continuous controls $u(\cdot) : [0,T] \to \mathbb{R}^{n_{\mathrm{u}}}$ and integer–valued controls $w(\cdot) : [0,T] \to \Omega_{\mathrm{w}}$ from a finite discrete set $\Omega_{\mathrm{w}} := \{w^1,\ldots,w^{n_{\Omega_{\mathrm{w}}}}\} \subset \mathbb{R}^{n_w}$, $|\Omega_{\mathrm{w}}| = n_{\Omega_{\mathrm{w}}} < \infty$ (1h). Both control profiles are subject to optimization. Moreover, we allow the end time T (1g), the continuous model parameters $p \in \mathbb{R}^{n_{\mathrm{p}}}$, and the discrete–valued model parameters $\rho \in \Omega_\rho := \{\rho^1,\ldots,\rho^{n_{\Omega_\rho}}\} \subset \mathbb{R}^{n_\rho}$, $|\Omega_\rho| = n_\rho < \infty$ (1i) to be subject to optimization as well. Control, parameters, and process trajectory may be constrained by inequality path constraints $c(\cdot)$ (1d), equality constraints $r_{\mathrm{c}}^{\mathrm{eq}}(\cdot)$ and inequality constraints $r_{\mathrm{c}}^{\mathrm{in}}(\cdot)$ coupled in time (1e), and decoupled equality and inequality constraints $r_i^{\mathrm{eq}}(\cdot)$, $r_i^{\mathrm{in}}(\cdot)$, $1 \leq i \leq n_{\mathrm{r}}$ (1f), imposed on a grid $\{t_i\}_{1\leq i\leq N_{\mathrm{r}}}$

C. Kirches
Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Im Neuenheimer Feld 368, 69120 Heidelberg, GERMANY.
Ph.: +49 (6221) 54-8895; Fax: +49 (6221) 54-5444; E-mail: christian.kirches@iwr.uni-heidelberg.de

S. Leyffer
Mathematics & Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, U.S.A.

of time points on time horizon $[0,T]$. This constraint grid need not coincide with the point least-squares grid. Constraints (1d–1f) include initial- and boundary values, mixed state–control constraints, periodicity constraints, and simple bounds on states, controls, and parameters.

$$\underset{\substack{x(\cdot),u(\cdot),w(\cdot),\\ p,\rho,T}}{\text{minimize}} \quad \int_{t=0}^{T} L(t,x(t),z(t),u(t),w(t),p,\rho)\,\mathrm{d}t + E(T,x(T),z(T),p,\rho) \\ + \sum_{i=1}^{N_{\text{lsq}}} ||l_i(t_i,x(t_i),z(t_i),u(t_i),w(t_i),p,\rho)||_2^2, \quad t_i \in [0,T], \tag{1a}$$

$$\text{subject to} \quad \dot{x}(t) = f(t,x(t),z(t),u(t),w(t),p,\rho), \quad t \in [0,T], \tag{1b}$$

$$0 = g(t,x(t),z(t),u(t),w(t),p,\rho), \quad t \in [0,T], \tag{1c}$$

$$0 \le c(t,x(t),z(t),u(t),w(t),p,\rho), \quad t \in [0,T], \tag{1d}$$

$$0 = r_{\text{c}}^{\text{eq}}(x(0),z(0),x(T),z(T),p,\rho), \tag{1e}$$

$$0 \le r_{\text{c}}^{\text{in}}(x(0),z(0),x(T),z(T),p,\rho),$$

$$0 = r_i^{\text{eq}}(x(t_i),z(t_i),p,\rho), \quad t_i \in [0,T],\ 1 \le i \le N_{\text{r}}, \tag{1f}$$

$$0 \le r_i^{\text{in}}(x(t_i),z(t_i),p,\rho),$$

$$T_{\min} \le T \le T_{\max}, \tag{1g}$$

$$w(t) \in \Omega_{\text{w}}, \quad t \in [0,T], \tag{1h}$$

$$\rho \in \Omega_{\rho}. \tag{1i}$$

Direct and simultaneous methods for attacking the class (1) of optimization problems posed in function spaces are based on the first–discretize–then–optimize approach. A discretization scheme is chosen and applied to objective (1a), dynamics (1b, 1c), and path constraints (1d) of the optimization problem in order to obtain a finite-dimensional counterpart problem accessible to mathematical programming algorithms. This counterpart problem will usually be a high-dimensional and highly structured nonlinear problem (NLP) or mixed-integer nonlinear problem (MINLP). As such, it readily falls into the domain of applicability of many mathematical optimization software packages already interfaced with AMPL, such as e.g. filterSQP [19], IPOPT [45], or SNOPT [22] for nonlinear programming, and e.g. Bonmin [9] or FilMINT [1] for mixed-integer nonlinear programming.

More citations here?

The AMPL Modeling Language described by Fourer et al [21] was designed as a mathematical modeling language for linear programming problems, and has later been extended to integer and mixed–integer linear problems, to mixed–integer nonlinear problems, to complementarity problems [] , and to semidefinite problems [] . (citation) AMPL's syntax closely resembles the symbolic and algebraic notation used to present (citation) mathematical optimization problems, and allows for fully automatic processing of optimization problems by a computer system. The availability of a symbolic problem representation enabled by AMPL's approach to modeling mathematical optimization problems has a number of significant advantages. These include the possibility for automatic differentiation, extended error checking facilities, automated analysis of model parts for certain structural properties such as linearity or convexity, and the

(does someone do this? citation)

opportunity for automatically generating model code for lower-level languages [] . The long–standing popularity and success of AMPL is underlined by the wealth of mathematical optimization software packages available on the NEOS Server for Optimization [23] that provide interfaces to read and solve problems modeled using the AMPL language.

Benefits Hence it appears desirable to extend the AMPL modeling language by a facility that conveys the notion of ODE and DAE constraints (1b, 1c) to a nonlinear or mixed–integer nonlinear problem solver. If given a generic description of a selected discretization method, the solver would apply this discretization to obtain an (MI)NLP with discretized objective and constraints to work on. This approach would effectively remove the need for tedious explicit encoding of fixed discretization schemes in the AMPL model, as is e.g. done for collocation schemes in [17]. Adaptive choice and iterative refinement of the discretization, see e.g. [24], would become possible. At the same time, this idea would open up the possibility of using alternative and more involved discretization schemes that preclude themselves from straightforward encoding in AMPL. One example are direct multiple–shooting methods, see [8], that enable the use of state-of-the-art ODE and DAE solvers, see e.g. [4, 35], with increased opportunities for exploiting structure and adaptivity. Vice versa, the envisioned approach would open up a generic way of tackling challenging mixed-integer dynamic optimization problems with the most recent and up-to-date methods developed in the MINLP community.

Related Efforts The idea of a high-level modeling language for dynamic simulation and optimization problems is by all means not a new one. Driven by demands mostly arising out of the industrial and engineering communities, several efforts have lead to the creation of commercial development environments for simulation and partly also for optimization of dynamic processes, some examples being gPROMS [5], Tomlab's PROPT [39], Dymola [18], Modelica [31], and recently the open-source initiative JModelica/Optimica [3]. In contrast to the approach we envision, these development environments however mainly focus on the end-user's convenience of interaction with the development environment, while often being closely tied to one or a few selected discretization schemes and solvers only.

1.1 Contributions and Structure

In this paper we describe a set of extensions to the AMPL modeling language that aims at extending AMPL's applicability to mathematical optimization problems constrained by ODE and DAE dynamics, i.e. optimal control problems. All proposed extensions are realized as either *AMPL user functions* or *AMPL suffixes*. Both user functions and suffixes are integral parts of the existing AMPL language standard. Hence, no intrusive changes to the AMPL language standard or implementation itself are required to realize the proposed extensions.

We describe an open–source implementation of the proposed extensions that we refer to as TACO, the Toolkit for AMPL Control Optimization. TACO is designed

to facilitate the coupling of existing optimal control software packages, as it allows us to read AMPL *stub.nl* files and detect the optimal control problem's structure. We use TACO to interface the two codes MUSCOD-II and MS-MINTOC with AMPL. MUSCOD-II, the multiple shooting code for optimal control described by [27, 28] and [8, 15], is a direct and simultaneous method for DAE-constrained optimal control. MS-MINTOC extends this software package by multiple–shooting mixed–integer optimal control algorithms [40, 42].

The applicability of the proposed and implemented extensions is shown using three examples of ODE-constrained control problems. A larger collection of AMPL models for ODE and DAE-constrained mixed-integer optimal control is made available on the web site `mintoc.de` [42]. Using the TACO interface, the two solvers MUSCOD-II and MS-MINTOC are made available on the NEOS Server for Optimization [23].

The remainder of this paper is structured as follows. In §2 we propose several extensions to the AMPL modeling language and realize them through AMPL user functions and suffixes. Two exemplary AMPL models are discussed in §3, and a collection of further AMPL models available from the web site `mintoc.de` is described. Consistency and smoothness checks performed by TACO and possible errors that may be encountered in this phase are described in §4. Here, we also discuss the communication of optimal solutions of time-dependent variables to AMPL. In §5 we show how to use the information provided by TACO to realize an interface coupling the optimal control software package MUSCOD-II and the mixed-integer optimal control algorithm MS-MINTOC to AMPL. The use of AMPL's automatic derivatives within an ODE/DAE solver is discussed. In §6, limitations and possible future extensions of the considered problem class are addressed. Finally, §7 concludes this paper with a brief summary.

Three appendix sections hold technical information about the TACO implementation that may be of interest to developers of optimal control codes who wish to interface their codes with AMPL.

2 AMPL Extensions for Optimal Control

The modeling language AMPL has been designed for finite-dimensional optimization problems. When attempting to model optimal control problems for DAE dynamic processes in AMPL without encoding a discretization of the dynamics in the model itself, one is confronted with several challenges. First, variables representing trajectories on a time horizon $[0,T]$ require a notion of time dependency. Second, certain constraints model ODE or DAE dynamics, and this knowledge must be conveyed to the solvers. Finally, objectives and constraints containing trajectory–representing variables may either refer to the entire trajectory, or to an evaluation of that trajectory in a certain point in time.

In this section we describe extensions to the AMPL syntax that address these points and allow for a convenient formulation of optimal control problems from class (1). All extensions are realized either as user functions or via AMPL suffixes, and do not require modifications of the AMPL system itself. Hence, they are readily appli-

cable in any development or research environment that provides access to an AMPL installation. A first example of the proposed AMPL extensions is shown in Fig. 1 for a nonlinear toy problem found in [14].

$$\begin{aligned}
&\underset{x(\cdot),u(\cdot),p}{\text{minimize}} \quad \int_0^3 x(t)^2 + u(t)^2 \,\mathrm{d}t \\
&\text{subject to } \dot{x}(t) = (x(t)+p)x(t) - u(t), \\
&\qquad x(0) = -0.05, \\
&\qquad -1 \le x(t) \le 1, \\
&\qquad -1 \le u(t) \le 1.
\end{aligned}$$

```
var t;
var x >= -1, <= 1, := -0.05;
var u >= -1, <= 1, := 0 suffix type "u0";
param p := 1.0;

minimize Lagrange: integral (x^2 + u^2, 3);
subject to
ODE: diff (x, t) = (x + p) * x - u;

IC: eval (x, 0) = -0.05;
```

Fig. 1 Exemplary use of the proposed AMPL extensions `integral`, `diff`, and `eval` in the AMPL model (right) for an ODE-constrained optimal control problem (left). Initial guesses for p and for $x(t)$ and $u(t)$ on $[0,T]$ found in the AMPL model are not given in the mathematical problem formulation.

2.1 Modeling Optimal Control Problem Variables

We first address the modeling of optimal control problem variables. A major task in detecting the optimal control problem's structure is to automatically infer the role AMPL variables should play in problem (1). We introduce the AMPL suffix `type` required to help here, and describe the rules of inference.

Independent Time In the following we expect any optimal control problem to declare a variable representing independent time t in (1). For consistency, we assume this variable to always be named `t` in this paper. There is, however, no restriction to the naming or phyiscal interpretation of this variable in an actual AMPL model. The independent time variable is detected by its appearance in a call to the user-function `diff`. Use of the same variable in all calls to `diff` is enforced for well-posedness.

For the end point T of the time horizon $[0,T]$, occasionally named `T` in AMPL models in this paper, the user is free to either introduce a variable, e.g. if T is free and subject to optimization, or to use a numerical constant (as in Fig. 1) or a defined variable if T is fixed.

Differential State Variables are denoted by $x(\cdot)$ or $x_i(\cdot)$ in this paper. They are detected by their appearance in a call to the user-function `diff` (see Fig. 1) which defined the differential right hand side for this state. Only one such call may be made for each differential state variable.

Algebraic State Variables are denoted by $z(\cdot)$ or $z_i(\cdot)$ in this paper. As any DAE constraint may involve more than one algebraic state variable, there is, unlike in the ODE case, no one–one correspondence between DAE constraints and DAE variables. Hence, we propose to detect DAE variables by flagging them as such using an AMPL suffix named `type`, which is `let` to the symbolic value `"dae"`.

Continuous and Integer Control Variables Control variables are detected by flagging them as such using the proposed AMPL suffix `type`, which is `let` to the one of several symbolic values representing choices for the control discretization (see Fig. 1). The current implementation offers the piecewise constant (`type` assumes the value `"u0"`), piecewise linear (`"u1"`), piecewise linear continuous (`"u1c"`), piecewise cubic (`"u3"`), and piecewise cubic continuous (`"u3c"`) discretizations. Integer controls may simply be declared by making use of existing AMPL syntax elements such as the keywords `integer` and `binary`.

Continuous and Integer Parameters Any AMPL variable not inferred to be independent or final time, a differential or algebraic state, or a control according to the rules described above is considered a model parameter p_i or ρ_i that is constant in time but may be subject to optimization e.g. in parameter estimation problems. Again, integer parameters may be declared by making use of the existing AMPL syntax elements `integer` and `binary`.

2.2 Modeling Optimal Control Problem Constraints

In this section we address the various types of constrains found in problem class (1).

Dynamics: ODE Constraints We propose a user-function `diff(var,t)` that is used in equality constraints to denote the left–hand side of an ODE (1b). The first argument `var` denotes the differential state variable for which a right–hand side is to be defined. The second argument `t` is expected to denote the independent time variable. Currently, only explicit ODEs are supported, i.e. the `diff` expression in ODE constraints must appear isolated on one side of the constraint.

That point of interest here is the mere appearance of a call to `diff` in an AMPL constraint expression that allows to distinguish an ODE constraint from other constraint types. The actual implementation of `diff` is bare of any functionality and may simply return the value zero.

Dynamics: DAE Constraints An *equality* constraint that calls neither `eval` nor `diff` is understood as a DAE constraint (1c).

Most DAE solvers will expect the DAE system to be of index 1. This means that the number of DAE constraints (1c) must match the number n_z of algebraic states declared by suffixes, and the common Jacobian of all DAE constraints w.r.t. the algebraic states must be regular in a neighborhood of the DAE trajectory $t \in [0,T] \mapsto (x(t),z(t))$. The first requirement can be enforced. The burden of ensuring regularity is put on the modeler creating the AMPL model, but can be verified locally by a DAE solver during runtime.

Path Constraints An *inequality* constraint that calls neither `eval` nor `diff` is understood as an inequality path constraint (1d).

Point Constraints impose restrictions on the values of trajectories only in certain points $t_i \in [0,T]$. To this end, we propose a user function `eval(expr,time)` that denotes the evaluation of a state or control trajectory, more generally an analytic expression `expr` referring to such trajectories, at a given fixed point `time` on $[0,T]$. This allows to model point constraints (1e, 1f) in a straightforward way. Fig. 1 shows an initial-value constraint.

A Note on Evaluation Time Points For problems with fixed end-time T, time points `time` in calls to `eval` obviously are absolute values in the range $[0,T]$. Negative values and values exceeding T are forbidden.

For problems with free final time T, i.e. for problems using an AMPL variable `T` representing the final time T properly introduced and free to vary between its lower and upper bound, all time points necessarily need to be understood as *relative* times on the normalized time horizon $[0,1]$. For such problems it is neither possible nor desirable to specify absolute time points other than the boundaries $t = 0$ and $t = T$.

2.3 Modeling Optimal Control Problem Objectives

Problem class (1) uses an objective function consisting of a Lagrange–type integral term L and a Mayer–type end–point term E. In addition, it is of advantage to detect least–squares structure of the integral Lagrange–term to be exploited after discretization. We also support a point least–squares term in the objective, arising e.g. in parameter estimation problems.

Lagrange–Type and Integral Least–Squares–Type Objective Terms We propose a user function `integral(expr,T)` to denote an integral–type objective function. The first argument `expr` is the integrand, the second one `T` is expected to denote the final time variable or value T, see Fig. 1. We always assume integration with respect to the independent time t, starting in $t = 0$. If `expr` is a sum of squares, the Lagrange–type objective is treated as an integral least–squares one.

Mayer– and Point Least–Squares–Type Objective Terms The proposed user function for evaluating trajectory variables, `eval(expr,time)`, can be used to model both Mayer–type (`time` is T) and point least–squares–type objective terms (`expr` is a sum of squares, and `time` is an arbitrary point $t_i \in [0,T]$ or $t_i \in [0,1]$). Note that for well-posedness, the Mayer type objective function must not depend on control variables, a restriction that can be enforced.

2.4 A Universal Header File for Optimal Control Problems

For convenience, necessary AMPL declarations of the proposed user functions `diff`, `eval`, and `integral` as well as of the suffix `type` and its possible symbolic values are collected in a header file named `OptimalControl.mod` that may be included in the first line of an AMPL optimal control model. Its contents are shown in Fig. 2

```
suffix type symbolic IN;

option type_table '\
1  u0        piecewise constant control\
2  u1        piecewise linear control\
3  u1c       piecewise linear continuous control\
4  u3        piecewise cubic control\
5  u3c       piecewise cubic continuous control\
6  dae       DAE algebraic state variable\
7  Lagrange  Prevent least-squares detection in an objective\
';

function diff;
function eval;
function integral;
```

Fig. 2 Convenience declaration of user functions and suffixes in the header file `OptimalControl.mod`.

3 Examples of Optimal Control Models in AMPL

In this section we give two examples of using the proposed AMPL extensions to model dynamic optimization problems. The first problem is an ODE boundary value problem from the COPS library [17]. The second one is an ODE-constrained mixed-integer control problem due to [40].

3.1 An Example from the COPS Library of Optimal Control Problems

Analyze the flow of a fluid during injection into a long vertical channel, assuming that the flow is modeled by the boundary value problem

$$u^{(4)}(t) = R\left(u^{(1)}(t)u^{(2)}(t) - u(t)u^{(3)}(t)\right), \tag{2a}$$

$$u(0) = 0, \qquad u(1) = 1, \tag{2b}$$

$$u^{(1)}(0) = 0, \qquad u^{(1)}(1) = 0, \tag{2c}$$

where u is the potential function, $u^{(1)}$ is the tangential velocity of the fluid, and $R > 0$ is the Reynolds number. This problem due to [17] is a feasibility problem for the boundary constraints. Fig. 5 shows the optimal solution obtained with MUSCOD-II for $R = 10^4$.

```
include OptimalControl.mod

var t;                  # independent time
param tf;               # ODEs defined in [0,tf]
var u{1..4} := 0;       # differential states
param R >= 0;           # Reynolds number

subject to

d1: diff(u[1],t) = u[2];
d2: diff(u[2],t) = u[3];
d3: diff(u[3],t) = u[4];
d4: diff(u[4],t) = R*(u[2]*u[3] - u[1]*u[4]);

u1s: eval(u[1],0) = bc[1,1];
u2s: eval(u[2],0) = bc[2,1];
u1e: eval(u[1],tf) = bc[1,2];
u2e: eval(u[2],tf) = bc[2,2];
```

Fig. 3 The fluid flow in a channel problem of the COPS library, using the proposed AMPL extensions.

Fig. 3 shows the AMPL mode of this problem using the proposed extensions. For comparison, the unchanged model as found in the COPS library is shown in Fig. 4. Clearly, removal of the discretization scheme leads to a shorter (14 versus 32 lines) and more readable model code. Making use of an `include` statement admittedly hides parts of the AMPL code here. Being able to do so actually is an advantage, though, and would not be possible at all with the original COPS library code of Fig. 4 as the discretization scheme cannot easily be isolated from the remainder of the model.

3.2 A Mixed–Integer ODE–Constrained Example

The following Lotka–Volterra–type fishing problem with binary restriction on the fishing activity is due to [40] and not part of the COPS library, which contains continuous problems only. The goal is to minimize the deviation x_2 of predator and prey amounts from desired target values $\bar{x}_0$, $\bar{x}_1$ over a horizon of $T = 12$ time units by repeated decisions of whether or not to fish off proportional amounts of both predators and prey at every instant in time. The problem reads

$$\begin{aligned}
\underset{x(\cdot),w(\cdot)}{\text{minimize}} \quad & x_2(T) && && \text{(3a)}\\
\text{subject to} \quad & \dot{x}_0(t) = x_0(t) - x_0(t)x_1(t) - p_0 w(t) x_0(t), && t \in [0,T], && \text{(3b)}\\
& \dot{x}_1(t) = -x_1(t) + x_0(t)x_1(t) - p_1 w(t) x_1(t), && t \in [0,T], && \text{(3c)}\\
& \dot{x}_2(t) = (x_0(t) - \bar{x}_0)^2 + (x_1(t) - \bar{x}_1)^2, && t \in [0,T], && \text{(3d)}\\
& x(0) = (0.5, 0, 7, 0), && && \text{(3e)}\\
& 0 \le x_0(t), \qquad 0 \le x_2(t), && t \in [0,T], && \text{(3f)}\\
& w(t) \in \{0,1\}, && t \in [0,T], && \text{(3g)}
\end{aligned}$$

```
param nc > 0, integer;              # number of collocation points
param nd > 0, integer;              # order of the differential equation
param nh > 0, integer;              # number of partition intervals

param rho {1..nc};                  # roots of k-th degree Legendre polynomial
param bc {1..2,1..2};               # boundary conditions
param tf;                           # ODEs defined in [0,tf]
param h := tf/nh;                   # uniform interval length
param t {i in 1..nh+1} := (i-1)*h;  # partition

param fact {j in 0..nc+nd} := if j = 0 then 1 else (prod{i in 1..j} i);

param R >= 0;                       # Reynolds number

# The collocation approximation u is defined by the parameters v and w.
# uc[i,j] is u evaluated at the collocation points.
# Duc[i,j,s] is the (s-1)-th derivative of u at the collocation points.

var v {i in 1..nh,j in 1..nd};
var w {1..nh,1..nc};

var uc {i in 1..nh, j in 1..nc, s in 1..nd} =
  v[i,s] + h*sum {k in 1..nc} w[i,k]*(rho[j]^k/fact[k]);

var Duc {i in 1..nh, j in 1..nc, s in 1..nd} =
  sum {k in s..nd} v[i,k]*((rho[j]*h)^(k-s)/fact[k-s]) + h^(nd-s+1)*
  sum {k in 1..nc} w[i,k]*(rho[j]^(k+nd-s)/fact[k+nd-s]);

minimize constant_objective: 1.0;

subject to bc_1: v[1,1] = bc[1,1];
subject to bc_2: v[1,2] = bc[2,1];
subject to bc_3:
  sum {k in 1..nd} v[nh,k]*(h^(k-1)/fact[k-1]) + h^nd*
  sum {k in 1..nc} w[nh,k]/fact[k+nd-1] = bc[1,2];
subject to bc_4:
  sum {k in 2..nd} v[nh,k]*(h^(k-2)/fact[k-2]) + h^(nd-1)*
  sum {k in 1..nc} w[nh,k]/fact[k+nd-2] = bc[2,2];

subject to continuity {i in 1..nh-1, s in 1..nd}:
  sum {k in s..nd} v[i,k]*(h^(k-s)/fact[k-s]) + h^(nd-s+1)*
  sum {k in 1..nc} w[i,k]/fact[k+nd-s] = v[i+1,s];

subject to collocation {i in 1..nh, j in 1..nc}:
  sum {k in 1..nc} w[i,k]*(rho[j]^(k-1)/fact[k-1]) =
  R*(Duc[i,j,2]*Duc[i,j,3] - Duc[i,j,1]*Duc[i,j,4]);
```

Fig. 4 The fluid flow in a channel problem as found in the COPS library.

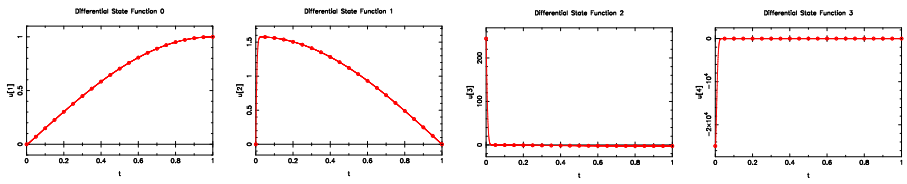

Fig. 5 Optimal solution of the flow in a channel problem for $R = 10^4$, computed with MUSCOD-II running inside the AMPL environment. `nshoot`=20.

with $p_0 := 0.4$, $p_1 := 0.2$, $\bar{x}_0 = 1$, $\bar{x}_1 = 1$. The AMPL model using the proposed extensions is given in Fig. 6. MS-MINTOC obtains the optimal solution shown in Fig. 7 by applying Partial Outer Convexification, and solving the convexified and relaxed optimal control problem. A rounding strategy with $\varepsilon-$optimality certificate is applied to find the switching structure, which is refined by switching time optimization afterwards. For details, we refer to [40, 42].

```
include OptimalControl.mod;

var t;
var xd0 := 0.5, >= 0, <= 20;
var xd1 := 0.7, >= 0, <= 20;
var dev := 0.0, >= 0, <= 20;

var u := 1, >= 0, <= 1 integer suffix type "u0";

param p0 := 0.4;
param p1 := 0.2;
param ref0 := 1.0;
param ref1 := 1.0;

# Minimize accumulated deviation from reference after 12 time units
minimize Mayer: eval(dev,12);

subject to

# ODE system
ODE_0: diff(xd0,t) = xd0 - xd0*xd1 - p0*u*xd0;      # prey
ODE_1: diff(xd1,t) = -xd1 + xd0*xd1 - p1*u*xd1;     # predator
ODE_2: diff(dev,t) = (xd0-ref0)^2 + (xd1-ref1)^2;   # deviation from reference

# initial value constraints
IVC_0: eval(xd0,0) = 0.5;
IVC_1: eval(xd1,0) = 0.7;
IVC_2: eval(dev,0) = 0.0;
```

Fig. 6 The predator-prey mixed-integer problem modeled in AMPL, using the proposed extensions.

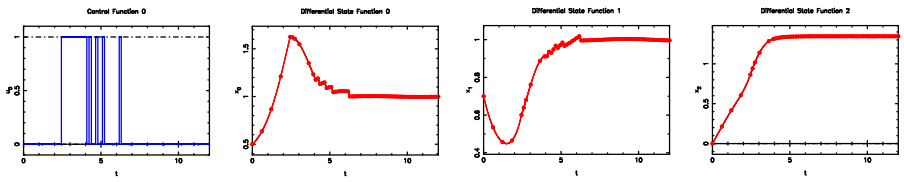

Fig. 7 Optimal integer control (blue) and optimal differential state trajectories (red) of the mixed-integer predator-prey problem, computed with MUSCOD-II and MS-MINTOC running inside the AMPL environment.

3.3 AMPL Models of Optimal Control Problems in the `mintoc.de` Collection

An online library of mixed-integer optimal control problems has been described in [41] and is available from the web site `http://mintoc.de`. This site holds a wiki-based publicly accessible library, and is actively encouraging the optimal control community to contribute interesting problems modeled in various languages. So far, models in C, C++, JModelica, and AMPL (using encoded collocation schemes) can be found together with best known optimal solutions and optimal control and state profiles.

We contributed to this web site 18 optimal control and parameter estimation problems modeled using the proposed AMPL extensions, as shown in Tab. 1. These include ODE-constrained problems and a DAE-constrained control problem as well as continuous problems and a mixed-integer control problem. Among them are 10 dynamic control and parameter estimation problems found in the COPS library [17].

Name	n_x	n_z	$n_u + n_w$	n_{pf}	Type[1]	References
batchdist	11[2]	–	1	–	OCP	[16]
brac	3	–	1	–	OCP	[7]
catmix	2	–	1	–	OCP	[17], [43]
cstr	4	–	2	2	OCP	[14]
fluidflow	4	–	–	–	BVP	[17]
gasoil	2	–	–	3	PE	[17], [44]
goddart	3	–	–	1	OCP	[17]
hangchain	3	–	1	–	OCP	[17], [12]
lotka	3	–	1	–	MIOCP	[40]
marine	8	–	–	15	PE	[17], [38]
methanol	3	–	–	5	PE	[17], [20], [30]
nmpc1	1	–	1	–	OCP	[14]
particle	4	–	1	1	OCP	[17], [11]
pinene	5	–	–	5	PE	[17], [10]
reentry	3	–	1	1	OCP	[36, 26, 37]
robotarm	6	–	3	1	OCP	[17], [33]
semibatch	5	1	4	–	OCP	[25, 32]
tocar1	2	–	1	1	OCP	[13, 29]

Table 1 List of AMPL optimal control problems modeled using the proposed AMPL extensions and contributed to the mixed-integer optimal control problem library at `http://mintoc.de`.
(1) OCP: optimal control, BVP: boundary value, PE: parameter estimation, MIOCP: mixed-integer OCP.
(2) batchdist is a 1-dimensional PDE and n_x can be increased by choosing a finer spatial discretization.

4 The TACO Toolkit for AMPL Control Optimization

This section describes the design and usage of the TACO toolkit for AMPL control optimization. Its purpose is to infer the role AMPL variables, objectives, and constraints play in problems of the class (1), to verify that the AMPL model conforms to this problem class, and finally to build a database of its findings for later use by optimal control problem solvers.

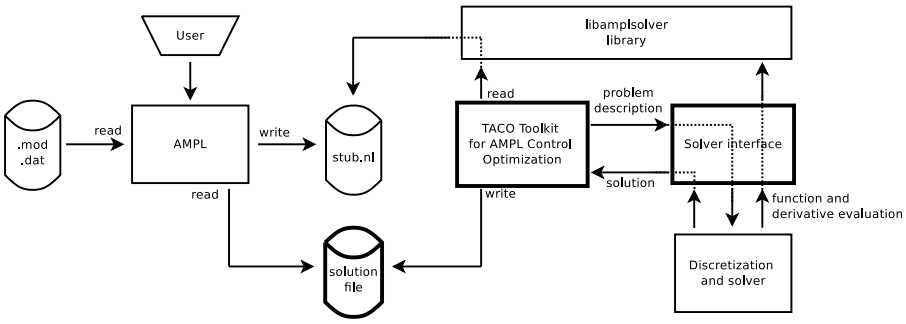

Fig. 8 Data flow between AMPL, the proposed TACO toolkit, and an optimal control problem solver.

Details about data structures and function calls found in the C implementation of TACO that may be of interest to developers of optimal control codes, wishing to interface their code with AMPL, can be found in the appendix to this paper.

4.1 Implementations of the Proposed User Functions

For the three proposed user functions `diff`, `eval`, and `integral`, two sets of implementations need to be provided. The first set is used during parsing of AMPL model files and writing the *stub.nl* file. It serves to indicate the names of the user functions and the number and type of their respective arguments. AMPL expects these implementations to be provided in a shared object (.so) or dynamic link library (.dll) named `amplfunc.dll` on all systems. Hence, these implementations are readily provided along with the optimal control frontend.

The second set of implementations is used during evaluation of AMPL objectives and constraints by the optimal control problem solver. Here, `diff` shall return the value zero as ODE constraints are explicit. `eval` shall access the solver's current iterate to return the expression's current value at the specified point in time. `integral` shall return the integrand's value in the same way. Consequentially, the second set of user function implementations is solver dependent and must be carried out separately for each solver to be interfaced with the optimal control frontend. We provide implementations for MUSCOD-II and MS-MINTOC.

4.2 Consistency Checks

In this section we address automated checks for consistency and smoothness of an AMPL optimal control problem formulation, performed by the TACO toolkit before passing the model on to a solver.

Checks for Sufficient Smoothness Most numerical algorithms applied in optimal control as well as in mixed-integer nonlinear programming assume all functions to be sufficiently smooth. In particular, this means that the objective terms L, E, r_i, $1 \leq i \leq n_{\mathrm{lsq}}$,

and the constraints c, r_c, r_s, r_e have to be twice continuously differentiable with respect to the continuous variables $t, x(t), z(t), u(t)$, and p. This guarantees a continuous Hessian of the Lagrangian of the discretized optimization problems, see e.g. [34]. Depending on the numerical methods applied to solve the DAE system, a higher order of continuous differentiability may be required for the ODE dynamics f (1b), though this is not enforced. The algebraic constraint function g (1c) may need to be invertible w.r.t. $z(t)$ in the neighborhood of trajectories $(t, x(t), z(t))$, $t \in [0, T]$, a property that can be checked by a DAE solver only during the runtime.

Consistency Checks for AMPL Extensions For each objective and constraint the optimal control frontend accesses the directed acyclic graph (DAG) representing the AMPL expression of the objective or constraint's body. This allows to detect which variables are part of the expression, to learn about calls to one of the three proposed user functions, and to check for sufficient smoothness of the expression. The following structural properties of the optimal control problem are currently verified automatically by the optimal control frontend during examination of the DAGs:

Dynamic constraints
- There is exactly one call to `diff` present per differential state variable.
- All ODE constraints are explicit, i.e. `diff` appears isolated on either the left-hand or right-hand side.
- The time variable in calls to `diff` is the same for all such calls.
- The state variables in calls to `diff` have not already been declared to be control or algebraic variables by means of the suffix `type`.
- All ODE and DAE constraints are equality constraints.
- The number of DAE constraints matches the number of algebraic variables.
- Calls to `eval` do not appear in ODE or DAE constraints.

Objective functions
- Calls to `diff` do not appear in objectives.
- All objective functions conform to one of the four types Lagrange, Mayer, integral- or point-least-squares.
- Expressions evaluated in calls to `integral` are time-dependent and not constants.
- The end-time in calls to `integral` agrees with the end-time variable or value T.
- The Mayer-type objective must not depend on control variables.

Path- and point constraints
- Calls to `integral` do not appear in constraints.
- Expressions evaluated in calls to `eval` are time-dependent and not constants.
- Evaluation time points in calls to `eval` are on $[0, T]$ if T is fixed, and on $[0, 1]$ if T is free.

Miscellaneous
- Absence of logical, relational, or other non-smooth operators in the bodies of functions that are assumed to be twice continuously differentiable.
- The `integer` and `binary` attributes are applied to control and parameter variables only.

– Calls to `diff`, `eval`, or `integral` are not nested into one another.

Violations are detected before the optimal control problem solver itself is invoked. They terminate the solution process and are reported back to the user in a human-readable way, quoting the name of the offending variable, objective, or constraint.

4.3 User Diagnosis of the Detected Optimal Control Problem Structure

In addition to automated consistency checks, TACO provides the AMPL solver option `verbose` that may be invoked (e.g. `option muscod_solver "verbose=1";`) to obtain diagnostic output of the types, names, and further properties of all variables, constraints, and objectives that are part of the optimal control problem, see Fig. 9. This has proven helpful for narrowing down the source of modeling errors.

```
#   name     svar     lbnd     init     ubnd    scale
t   t           2        -        -        -        -
tf  tf          6       12       12       12        1
Differential States
#   name     svar     lbnd     init     ubnd    scale fix
0   xd0         3        0      0.5       20        1 IVC_0      4
1   xd1         4        0      0.7       20        1 IVC_1      5
2   dev         1        0        0       25        1 IVC_2      6
Controls
#   name     svar     lbnd     init     ubnd    scale discr int
0   u           5        0        1        1        1    u0 yes
ODE Constraints
#   name     scon   factor    scale
0   ODE_0       1        1        1
1   ODE_1       2        1        1
2   ODE_2       3        1        1
Objectives
#   name     sobj     type  dpnd    scale parts
0   -           1    Mayer .x...        1     -
```

Fig. 9 Exemplary diagnostic output for the predator-prey example of §3.2 invoked by the solver option `verbose`. The output allows to learn about the automatically detected associations of the different types of optimal control problem variables, constraints and objectives in problem (1) with their AMPL counterparts.

4.4 Output of Time–Dependent Solution Variables

Following the solution of an optimal control problem using our extensions to AMPL, we must also provide a way to return this solution to the user. Currently, every AMPL variable can assume a single scalar value only, possibly augmented by scalar valued suffixes. Hence, communication of the values of discretized trajectories is not possible in a straightforward way. For the AMPL optimal control frontend, we opted in favor of writing a separate plain-text solution file holding newline-separated values with the following layout:

- Optimal objective function value
- Remaining infeasibility (exact meaning depends on solver)
- Optimal final time T
- Number N of trajectory output points
- Output grid times (N values)
- Optimal values of free model parameters p (n_p values)
- Optimal differential state trajectories ($n_\mathrm{x} \times N$ values, row-wise)
- Optimal algebraic state trajectories ($n_\mathrm{z} \times N$ values, row-wise)
- Optimal control trajectories ($n_\mathrm{u} \times N$ values, row-wise)

Future solvers may expand this file format. This data format is designed to be read back into the AMPL environment easily using existing AMPL language elements such as `data` and `read`. Example AMPL code that accomplishes this is given in Fig. 10. This functionality is of vital importance as it allows to solve multiple related optimal control problems in a loop, each one based on the solution of the previous ones.

```
model;
param obj;
param infeas;
param tf;
param ndis integer;
var t_sol{1..ndis}
var p_sol{1..np};
var x_sol{1..nx, 1..ndis};
var z_sol{1..nz, 1..ndis};
var u_sol{1..nu, 1..ndis};

data;
read obj < filename.sol;
read inf < filename.sol;
read tf < filename.sol;
read ndis < filename.sol;
read{j in 1..ndis} (t_sol[j]) < filename.sol;
read{j in 1..np} (p_sol[j]) < filename.sol;
read{i in 1..nx, j in 1..ndis} (x_sol[i,j]) < filename.sol;
read{i in 1..nz, j in 1..ndis} (z_sol[i,j]) < filename.sol;
read{i in 1..nu, j in 1..ndis} (u_sol[i,j]) < filename.sol;
```

Fig. 10 A universal AMPL script for reading a discretized optimal control problem's solution from a file `filename.sol` back to AMPL. Variables `nx`, `nz`, `nu`, and `np` denote dimensions of the differential and algebraic state vectors and the control vector, and are assumed to be known.

5 Using TACO to Interface the Software Packages MUSCOD-II and MS-MINTOC

Here we describe the use of TACO to interface the multiple shooting code for optimal control MUSCOD-II [27, 28] with AMPL. MUSCOD-II is based on a multiple shooting discretization in time [8], and implements a direct and all–at–once approach to solving DAE-constrained optimal control problems. Moreover, MUSCOD-II has been extended by the MS-MINTOC algorithm for DAE–constrained mixed–integer

optimal control, see [40, 42]. For more details on these solvers, we refer to the User's Manual [15].

5.1 The Direct Multiple Shooting Method for Optimal Control

This section briefly sketches the direct multiple shooting method, first described by [8] and extended in a series of subsequent works (see e.g. [28, 40]). With the optimal control software package MUSCOD-II [15], an efficient implementation of this method is available.

The purpose of this method is to transform the infinite-dimensional problem (1) into a finite-dimensional (mixed-integer) nonlinear program by discretization of the control functions on a time grid $t_0 < t_1 < \ldots < t_{N_{\text{shoot}}} = T$. For this, let $b_{ij} : [0,T] \to \mathbb{R}^{n_u}$, $1 \le j \le n_{\text{q}_i}$ be a set of sufficiently often continuously differentiable base function of the control discretization for the shooting interval $[t_i, t_{i+1}] \subset [0,T]$. Further, let $q_i \in \mathbb{R}^{n_{\text{q}_i}}$ be the corresponding set of control parameters, and define

$$\hat{u}_i(t, q_i^{\text{u}}) := \sum_{j=1}^{n_{\text{q}_i^{\text{u}}}} q_{ij}^{\text{u}}\, b_{ij}(t) \qquad t \in [t_i, t_{i+1}],\ 0 \le i < N_{\text{shoot}}. \tag{4}$$

The control space is hence reduced to functions that can be written as in (4), depending on finitely many parameters q_i. For integer controls $w(\cdot)$ we usually choose a piecewise constant discretization, i.e. we have $n_{\text{q}_i^{\text{w}}} = 1$ and $b_{i1}(t) = q_{i1}^{\text{w}}$, leading to $\hat{w}_i(t, q_i^{\text{w}}) = q_{i1}^{\text{w}}$. For continuous controls $u(\cdot)$, more elaborate discretizations based on e.g. linear or cubic base functions $b_{ij}(\cdot)$ are easily found.

The right-hand side functions f, g, and the constraint functions c, r_{c}^{eq}, r_{c}^{in}, r_i^{eq}, and r_i^{in} are assumed to be adapted accordingly. Multiple shooting variables s_i are introduced on the time grid to parameterize the differential states. The node values serve as initial values for an ODE or DAE solver computing the state trajectories independently on the shooting intervals.

$$\dot{x}_i(t) = f(t, x_i(t), \hat{u}_i(t, q_i), \hat{w}_i(t, q_i), p, \rho), \qquad t \in [t_i, t_{i+1}],\ 0 \le i < N_{\text{shoot}}, \tag{5a}$$

$$0 = g(t, x_i(t), z_i(t), \hat{u}_i(t, q_i), \hat{w}_i(t, q_i), p, \rho), \tag{5b}$$

$$x_i(t_i) = s_i, \qquad 0 \le i \le N_{\text{shoot}}. \tag{5c}$$

One advantage of the multiple shooting approach is the ability to use state-of-the-art adaptive integrator methods. In MUSCOD-II, the solution of the arising ODE and DAE initial-value problems is carried out using the code DAESOL [6]. Obviously, we obtain from the above IVPs N_{shoot} trajectories, which in general will not combine to a single continuous trajectory, see Fig. 11. Thus, continuity across shooting intervals needs to be ensured by additional matching conditions entering the NLP as equality constraints,

$$s_{i+1} = x_i(t_{i+1};\ t_i, s_i, z_i, q_i^{\text{u}}, q_i^{\text{w}}, p, \rho) \qquad 0 \le i < N_{\text{shoot}}. \tag{6}$$

Here we denote by $x_i(t_{i+1};\ t_i, s_i, z_i, q_i^{\text{u}}, q_i^{\text{w}}, p, \rho)$ the solution of the IVP on shooting interval i, evaluated in t_{i+1}, and depending on the initial values s_i, z_i, control parameters q_i^{u}, q_i^{w}, and model parameters p, ρ. Finally, path constraints $c(\cdot)$ are discretized

on an appropriately chosen grid. To ease the notation, we assume in the following that all constraint grids match the shooting grid.

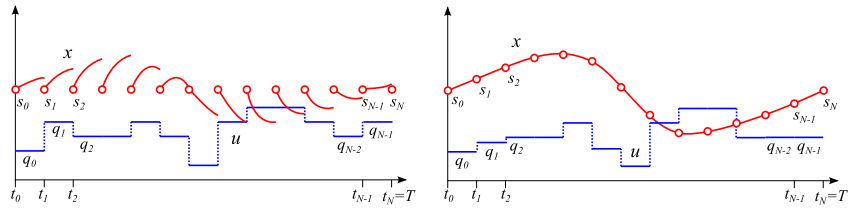

Fig. 11 A multiple shooting discretization in time. A constant initialization of state nodes s_i, $0 \le i \le N$ is shown on the left, and a continuous solution after convergence on the right. Continuity is required in the optimal solution only, ensured by matching conditions.

Structured Nonlinear Programming From this discretization and parameterization results a highly structured (MI)NLP of the form

$$\min_{\xi_C,\xi_I} \quad \sum_{i=0}^{N_{\text{shoot}}} \Phi_i\big(s_i, z_i, \hat{u}_i(t_i, q_i^{\text{u}}), \hat{w}_i(t_i, q_i^{\text{w}}), p, \rho\big) \tag{7a}$$

$$\text{s.t.} \quad 0 = x_i(t_{i+1}; s_i, z_i, q_i^{\text{u}}, q_i^{\text{w}}, p, \rho) - s_{i+1}, \qquad 0 \le i < N_{\text{shoot}}, \tag{7b}$$

$$0 = g\big(t_i, s_i, z_i, \hat{u}_i(t_i, q_i^{\text{u}}), \hat{w}_i(t_i, q_i^{\text{w}}), p, \rho\big), \qquad 0 \le i \le N_{\text{shoot}}, \tag{7c}$$

$$0 \le c\big(t_i, s_i, z_i, \hat{u}_i(t_i, q_i^{\text{u}}), \hat{w}_i(t_i, q_i^{\text{w}}), p, \rho\big), \qquad 0 \le i \le N_{\text{shoot}}, \tag{7d}$$

$$0 = r_{\text{c}}^{\text{eq}}\big(s_0, s_{N_{\text{shoot}}}, p, \rho\big), \tag{7e}$$

$$0 \le r_{\text{c}}^{\text{in}}\big(s_0, s_{N_{\text{shoot}}}, p, \rho\big),$$

$$0 = r_i^{\text{eq}}\big(s_i, z_i, \hat{u}_i(t_i, q_i^{\text{u}}), \hat{w}_i(t_i, q_i^{\text{w}}), p, \rho\big), \qquad 0 \le i \le N_{\text{shoot}}, \tag{7f}$$

$$0 \le r_i^{\text{in}}\big(s_i, z_i, \hat{u}_i(t_i, q_i^{\text{u}}), \hat{w}_i(t_i, q_i^{\text{w}}), p, \rho\big), \qquad 0 \le i \le N_{\text{shoot}},$$

$$q_i^{\text{w}} \in \Omega_{\text{w}}, \quad \rho \in \Omega_\rho, \tag{7g}$$

where the vectors ξ_C, ξ_I shall contain all unknowns of the problem

$$\xi_C = \big(s_0, \ldots, s_{N_{\text{shoot}}}, q_0^{\text{u}}, \ldots, q_{N_{\text{shoot}}-1}^{\text{u}}, p\big), \quad \xi_I = \big(q_0^{\text{w}}, \ldots, q_{N_{\text{shoot}}-1}^{\text{w}}, \rho\big). \tag{8}$$

For the ease of notation in (7d, 7f) we write

$$\hat{u}_{N_{\text{shoot}}}(t_{N_{\text{shoot}}}, q_{N_{\text{shoot}}}) := \hat{u}_{N_{\text{shoot}}-1}(t_{N_{\text{shoot}}}, q_{N_{\text{shoot}}-1}).$$

and analogously for $\hat{w}$. We may then continue to solve this large-scale structured (MI)NLP using one of the solvers mentioned in the introduction. To be efficient, this usually requires extensive exploitation of the arising NLP structures. Possible approaches include e.g. SQP methods with block-wise high-rank updates of Hessian approximations, partial reduction using the algebraic constraints to eliminate the z_i from the problem, and condensing algorithms for a reduction of the size of the arising quadratic subproblems. For details, we refer to e.g. [8, 27, 28].

5.2 Sensitivity Computation and Automatic Derivatives

From the structured NLP (7) is becomes clear that AMPL objectives and most AMPL constraints can be evaluated as usual, and this also holds true for gradients or Jacobians. The exception to this are ODE and DAE constraint functions as they do not directly enter (7). Evaluation of these functions occurs during solution and sensitivity computation for the multiple shooting IVPs, carried out by an ODE/DAE solver called to evaluate the residual or a derivative of the matching constraint (7b).

We exemplarily consider the ODE case and look in more detail at the computation of a directional derivative $X \cdot d$ of differential state trajectory with respect to the problem's unknowns,

$$X = \frac{\mathrm{d}x(t_{i+1};t_i,s_i,z_i,q_i,p,\rho)}{\mathrm{d}(s_i,q_i,p)} \in \mathbb{R}^{n_\mathrm{x}\times(n_\mathrm{x}+n_\mathrm{q}+n_\mathrm{p})} \tag{9}$$

being the Jacobian of the solution of the IVP on $[t_i,t_{i+1}]$ with respect to the unknowns (s_i,q_i,p_i) of the problem in the shooting node at t_i. For a direction $d = (d_s,d_{q,p}) \in \mathbb{R}^{n_\mathrm{x}+n_\mathrm{q}+n_\mathrm{p}}$ this may e.g. be done by solving the system (10) of so-called variational differential equations,

$$\dot{x}_\mathrm{d}(t) = f_x(t)\cdot x_\mathrm{d}(t) + f_{q,p}(t)\cdot d_{q,p}, \quad t\in[t_i,t_{i+1}], \tag{10a}$$

$$x_\mathrm{d}(t_i) = d_s, \tag{10b}$$

simultaneously and using the same scheme (e.g. choice of method, step sizes, orders, etc.) with the IVP itself. Basic calculus then shows $x_\mathrm{d}(t_{i+1}) = X\cdot d$. In (10), time-dependent Jacobians of the ODE constraint are denoted by

$$f_x(t) = \frac{\partial f}{\partial x}(t,x(t),\hat{u}(t),w(t),p,\rho), \quad f_{q,p}(t) = \frac{\partial f}{\partial(q,p)}(t,x(t),\hat{u}(t),w(t),p,\rho)$$

Much in the same spirit, directional derivatives of $z(t)$ and derivatives into directions containing the shooting node algebraic unknowns z_i can be computed for partial reduction of DAE systems, see [27].

For direct multiple shooting NLPs, typically a significant amount of total the runtime, easily in excess of 80%, is spent on computing sensitivities of IVP solutions. Hence, it is vitally important to exploit the availability of automatic sparse derivatives of the ODE and DAE constraints in AMPL when computing f_x and $f_{q,p}$. Here, our approach of modeling dynamic constraints in AMPL has the additional advantage of not requiring a separate automatic differentiation tool, while at the same time being faster and more precise than finite difference approximations. In the MUSCOD-II interface to AMPL, we provide both dense and sparse Jacobians as well as directional derivatives formed from sparse matrix-vector products to the DAE solver DAESOL.

5.3 MUSCOD-II Specific Solver Options and AMPL Suffixes

This section addresses the specification of shooting discretizations, of interpolated initial guesses, of scaling factors, and of slope constraints in AMPL models to be solved by MUSCOD-II.

Specifying a Discretization MUSCOD-II applies a multiple shooting discretization in time to control trajectories, differential- and algebraic state trajectories, path constraints, and Lagrange- and integral least-squares-type objectives, cf. [27]. The number of multiple shooting intervals can be chosen using the solver option `nshoot`. For point least-squares objectives as well as for point constraints, the MUSCOD-II backend ensures the introduction of shooting nodes in the respective time points. Hence, the solver option `nshoot` specifies a minimum number of shooting intervals only. Moreover, adaptive refinement of the shooting discretization may be triggered by several mixed–integer strategies of the MS-MINTOC algorithms, cf. [40]. The final number of shooting intervals used for computation of the optimal solution is available from the solution file written by the frontend, see §4.4.

Initial Guesses, Scaling, and Slope Constraints For MUSCOD-II we provide several additional suffixes that allow to model more elaborate initializations, to apply scale factors to variables, objective, and constraints, and to impose constraints on the slope of linear or cubic spline controls.

`interp_to` Specifies a second initializer for a differential state at the end of the time horizon. Let α be the initial guess and β be the value of suffix `interp_to`. The initialization of the state trajectory shall be a linear interpolation between the first and the second initializer,

$$x(t) = \frac{T-t}{T}\alpha + \frac{t}{T}\beta, \quad t \in [0,T]. \tag{11}$$

Since AMPL does not provide a means of telling whether a suffix is uninitialized, and since interpreting $\beta = 0$ as uninitialized is not a viable option, the dedicated solver option `s_spec` has to be set in order to enable the `interp_to`. In this case, suffix `interp_to` has to be set for *all* differential state variables.

`scale` Specifies scale factors for variables, constraint, and objectives. The default value 0 means to infer the scale factor from the initial guess. If none is provided, a default scale factor of 1 is assumed.

`slope_max` and `slope_min` Specify upper and lower bounds on the slopes of piecewise linear or cubic controls. Since AMPL does not allow for non-zero default values of suffices, this suffix *must* be set for *all* controls that are not of piecewise constant type.

Solver Options Solver options recognized by the MUSCOD-II backend comprise the following keywords:

`nshoot` The minimum number of multiple shooting intervals to use. As already described above, constraint grids, point least-squares objective grids, and also MS-MINTOC strategies may increase this.

`itmax` The maximum number of SQP iterations allowed.

`atol` The acceptable KKT tolerance for termination of the SQP solver.

`levmar` Levenberg-Marquardt regularization for the Hessian of the Lagrangian.

`bflag` The MS-MINTOC strategy code, see [40].

`stiff` Set to use the BDF-type solver DAESOL [6] which can cope with stiff DAE systems. Unset to use a faster Runge-Kutta-Fehlberg–type method for non-stiff ODEs.

`deriv` Set to `sparse` to use sparse Jacobians of the ODE/DAE provided by AMPL. Set to `dense` to use dense Jacobians of the ODE/DAE provided by AMPL. Set to `finitediff` to approximate derivatives by finite differences, ignoring AMPL derivatives.

`solve` Selects the globalization method for the SQP solver. Available methods include `solve_fullstep` (no globalization), `solve_slse` (a line search method), and `solve_tbox` (a box-shaped trust region method).

`hess` Selects the Hessian approximation method to use. Available options include `hess_const` (constant Hessian), `hess_update` (a BFGS Hessian approximation), `hess_limitedmemoryupdate` (a limited memory BFGS Hessian approximation), `hess_gaussnewton` (Gauß–Newton approximation of the Hessian), and `hess_finitediff` (finite difference exact Hessian).

Usage in AMPL Fig. 12 shows two lines of exemplary AMPL code that demonstrate how to load the MUSCOD-II solver and pass algorithmic options from an AMPL model file.

```
option solver muscod;
option muscod_options "nshoot=20 atol=1e-8 itmax=200 hess=hess_gaussnewton";
option muscod_auxfiles "rc";
```

Fig. 12 Exemplary AMPL code to load the MUSCOD-II solver and pass solver options and row and column names.

6 Limitations and Possible Further Extensions

The extensions to the AMPL modeling language presented so far suffice to treat the class (1) of mixed-integer DAE-constrained optimal control problems. However, a number of possible extensions of this problem class are discussed next: Modeling of fully implicit ODE and DAE systems of the form

$$\begin{aligned} 0 &= f(t,x(t),\dot{x}(t),z(t),u(t),w(t),p,\rho) \quad t \in [0,T], \\ 0 &= g(t,x(t),z(t),u(t),w(t),p,\rho), \end{aligned}$$

or alternatively of a semi-implicit form which is preferred by many DAE solver implementations,

$$\begin{aligned} A(t,x(t),z(t),u(t),w(t),p,\rho)\dot{x}(t) &= f(t,x(t),z(t),u(t),w(t),p,\rho) \quad t \in [0,T], \\ 0 &= g(t,x(t),z(t),u(t),w(t),p,\rho), \end{aligned}$$

might be realized using a more sophisticated `diff` user function. As the current implementation of the AMPL solver library does not provide sufficient information to

associate $\dot{x}(t)$ arguments with $x(t)$ arguments, this would require manipulation of AMPL expression DAGs for all ODE constraints.

For convenience, the functionality of `diff()` could also be extended to allow higher-order ODEs to be formulated by passing the differential's order as a third argument, e.g. write `diff(x,t,2)` for $\ddot{x}(t)$. This would further reduce the number of lines required for the presented COPS fluid flow problem of §3.1.

For some DAE problems, in order to promote sparsity, automatic introduction of defined variables as additional algebraic states might be preferred over the in-place evaluation of defined variables that is currently carried out.

The use of non-smooth operators such as max, min, $|\cdot|$, or conditional statements could be allowed inside ODE constraints. This would open the possibility for modeling hybrid and implicitly switched systems in a quick and convenient way, given an ODE and DAE capable of computing derivatives of switching ODEs' solutions. The use of certain logical and non-smooth operators could also be allowed in path and point constraints, leading e.g. to optimal control problems with complementarity constraints [] or vanishing constraints [2]. citations

Certain optimal control problems of practical relevance require a multi–stage setup. Here, the number of differential and/or algebraic states and the number of controls may change at a certain, possibly implicitly determined point in time. Modeling multi–stage optimal control problems currently appears difficult using the extensions described in this report. Here AMPL's syntax provides insufficient contextual information about the stage a certain variable or constraint should be assigned to.

7 Summary and Outlook

We described an extension to the AMPL modeling language that extends the applicability of the AMPL modeling language beyond the domain of MINLPs by allowing to conveniently model mixed-integer DAE-constrained optimal control problems in AMPL. Contrary to prior approaches at modeling such problems in AMPL by explicitly encoding a discretization scheme for the dynamic parts of the model, our approach separates model equations and discretization scheme.

We showed that the proposed extensions do not require intrusive changes to the AMPL language standard or implementation itself, as they consist of a set of three AMPL user functions and an AMPL suffix. The TACO toolkit for AMPL control optimization was presented and serves as an interface between AMPL *stub.nl* files and an optimal control code. TACO is open–source and designed to facilitate the coupling of existing optimal control software packages to AMPL.

To demonstrate the applicability of TACO, we used this new toolkit to implement an AMPL interface for the optimal control software packages MUSCOD-II and its mixed–integer optimal control extension MS-MINTOC.

The modeling and solution of two exemplary control problems in AMPL using the proposed extensions showed the benefits of the proposed approach, namely shorter model code, improved readability, flexibility in the choice of a discretization scheme, and the possibility to adaptively modify and refine such a scheme.

In the future, it would be desirable to have implementations of a number of alternative schemes for evaluation of ODE and DAE constraints available, e.g. various collocation schemes. This would enable the immediate use of NLP and MINLP solvers.

Acknowledgements The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement n^{o} FP7-ICT-2009-4 248940. The first author acknowledges a travel grant by Heidelberg Graduate Academy, funded by the German Excellence Initiative. We thank Hans Georg Bock, Johannes P. Schlöder, and Sebastian Sager for permission to use the optimal control software package MUSCOD-II and the mixed–integer optimal control algorithm MS-MINTOC.

References

1. Abhishek K, Leyffer S, Linderoth T (2010) FilMINT: An Outer Approximation–Based Solver for Mixed–Integer Nonlinear Programs. INFORMS Journal on Computing 22(4):555–567
2. Achtziger W, Kanzow C (2008) Mathematical programs with vanishing constraints: optimality conditions and constraint qualifications. Mathematical Programming Series A 114:69–99
3. Åkesson J, Årzén K, Gräfvert M, Bergdahl T, Tummescheit H (2008) Modeling and optimization with Optimica and JModelica.org — Languages and tools for solving large-scale dynamic optimization problems. Computers & Chemical Engineering 34(11):1737–1749
4. Albersmeyer J (2010) Adjoint based algorithms and numerical methods for sensitivity generation and optimization of large scale dynamic systems. PhD thesis, Ruprecht–Karls–Universität Heidelberg
5. Barton P, Pantelides C (1993) gPROMS — A Combined Discrete/Continuous Modelling Environment for Chemical Processing Systems. Simulation Series 25:25–34
6. Bauer I, Bock H, Körkel S, Schlöder J (1999) Numerical Methods for Initial Value Problems and Derivative Generation for DAE Models with Application to Optimum Experimental Design of Chemical Processes. In: Scientific Computing in Chemical Engineering II, Springer, pp 282–289
7. Betts J, Eldersveld S, Huffman W (1993) Sparse nonlinear programming test problems (Release 1.0). Tech. Rep. BCSTECH-93-074, Boeing Computer Services
8. Bock H, Plitt K (1984) A Multiple Shooting algorithm for direct solution of optimal control problems. In: Proceedings of the 9th IFAC World Congress, Pergamon Press, Budapest, pp 242–247
9. Bonami P, Biegler L, Conn A, Cornuéjols G, Grossmann I, Laird C, Lee J, Lodi A, Margot F, Sawaya N, Wächter A (2005) An Algorithmic Framework For Convex Mixed Integer Nonlinear Programs. Research Report RC23771, IBM T.J. Watson Research Center
10. Box G, Hunter W, MacGregor J, Erjavec J (1973) Some problems associated with the analysis of multiresponse data. Technometrics 15:33—-51

11. Bryson A, Ho Y (1975) Applied Optimal Control: Optimization, Estimation, and Control. John Wiley & Sons
12. Cesari L (1983) Optimization — Theory and Applications. Springer Verlag
13. Cuthrell J, Biegler L (1987) On the optimization of differential-algebraic process systems. AIChE 33:1257–1270
14. Diehl M (2001) Real-Time Optimization for Large Scale Nonlinear Processes. PhD thesis, Ruprecht–Karls–Universität Heidelberg
15. Diehl M, Leineweber D, Schäfer A (2001) MUSCOD-II Users' Manual. IWR Preprint 2001-25, Interdisciplinary Center for Scientific Computing (IWR), Ruprecht–Karls–Universität Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany
16. Diehl M, Bock H, Kostina E (2006) An approximation technique for robust nonlinear optimization. Mathematical Programming 107:213–230
17. Dolan E, Moré J, Munson T (2004) Benchmarking Optimization Software with COPS 3.0. Tech. Rep. ANL/MCS-TM-273, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, U.S.A.
18. Elmqvist H, Brück D (2001) Dymola — Dynamic Modeling Language. User's manual, Dynasim AB
19. Fletcher R, Leyffer S (1999) User manual for filterSQP. Tech. rep., University of Dundee
20. Floudas C, Pardalos P, Adjiman C, Esposito W, Gumus Z, Harding S, Klepeis J, Meyer C, Schweiger C (1999) Handbook of Test Problems for Local and Global Optimization. Kluwer Academic Publishers
21. Fourer R, Gay D, Kernighan B (1990) A modeling language for mathematical programming. Management Science 36:519–554
22. Gill P, Murray W, Saunders M (2002) SNOPT: An SQP Algorithm for large–scale constrained optimization. SIAM Journal on Optimization 12:979–1006
23. Gropp W, Moré J (1997) Optimization Environments and the NEOS Server. In: Buhmann M, Iserles A (eds) Approximation Theory and Optimization, Cambridge University Press, pp 167–182
24. Kameswaran S, Biegler L (2008) Advantages of nonlinear-programming-based methodologies for inequality path-constrained optimal control problems - a numerical study. SIAM Journal on Scientific Computing 30:957–981
25. Kühl P, Milewska A, Diehl M, Molga E, Bock H (2005) NMPC for runaway-safe fed-batch reactors. In: Proc. Int. Workshop on Assessment and Future Directions of NMPC, pp 467–474
26. Leineweber D (1995) Analyse und Restrukturierung eines Verfahrens zur direkten Lösung von Optimal-Steuerungsproblemen. Diploma thesis, Ruprecht–Karls–Universität Heidelberg
27. Leineweber D (1999) Efficient reduced SQP methods for the optimization of chemical processes described by large sparse DAE models, Fortschritt-Berichte VDI Reihe 3, Verfahrenstechnik, vol 613. VDI Verlag, Düsseldorf
28. Leineweber D, Bauer I, Schäfer A, Bock H, Schlöder J (2003) An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization (Parts I and II). Computers and Chemical Engineering 27:157–174

29. Logsdon J, Biegler L (1992) Decomposition strategies for large-scale dynamic optimization problems. Chemical Engineering Science 47(4):851–864
30. Maria G (1989) An adaptive strategy for solving kinetic model concomitant estimation-reduction problems. Can J Chem Eng 67:825
31. Mattsson S, Elmqvist H, Broenink J (1997) Modelica: An International effort to design the next generation modelling language. Journal A, Benelux Quarterly Journal on Automatic Control 38(3):16–19, special issue on Computer Aided Control System Design, CACSD, 1998.
32. Milewska A (2006) Modelling of batch and semibatch chemical reactors – safety aspects. PhD thesis, Warsaw University of Technology
33. Moessner-Beigel M (1995) Optimale Steuerung für Industrieroboter unter Berücksichtigung der getriebebedingten Elastizität. Diploma thesis, Ruprecht–Karls–Universität Heidelberg
34. Nocedal J, Wright S (2006) Numerical Optimization, 2nd edn. Springer Verlag, Berlin Heidelberg New York
35. Petzold L, Li S, Cao Y, Serban R (2006) Sensitivity analysis of differential-algebraic equations and partial differential equations. Computers and Chemical Engineering 30:1553–1559
36. Plitt K (1981) Ein superlinear konvergentes Mehrzielverfahren zur direkten Berechnung beschränkter optimaler Steuerungen. Diploma thesis, Rheinische Friedrich–Wilhelms–Universität Bonn
37. Potschka A, Bock H, Schlöder J (2009) A minima tracking variant of semi-infinite programming for the treatment of path constraints within direct solution of optimal control problems. Optimization Methods and Software 24(2):237–252
38. Rothschild B, Sharov A, Kearsley A, Bondarenko A (1997) Estimating growth and mortality in stage-structured populations. Journal of Plankton Research 19:1913—1928
39. Rutquist P, Edvall M (2010) PROPT — Matlab Optimal Control Software. User's manual, TOMLAB Optimization
40. Sager S (2005) Numerical methods for mixed–integer optimal control problems. Der andere Verlag, Tönning, Lübeck, Marburg
41. Sager S (2011) A benchmark library of mixed-integer optimal control problems. In: Proceedings MINLP09, (accepted)
42. Sager S, Bock H, Diehl M (2011) The Integer Approximation Error in Mixed-Integer Optimal Control. Mathematical Programming A DOI 10.1007/s10107-010-0405-3
43. von Stryk O (1999) User's guide for DIRCOL (Version 2.1): A direct collocation method for the numerical solution of optimal control problems. Tech. rep., Technische Universität Müchen, Germany
44. Tjoa IB, Biegler L (1991) Simultaneous solution and optimization strategies for parameter estimation of differential-algebraic equations systems. Ind Eng Chem Res 30:376—-385
45. Wächter A, Biegler L (2006) On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. Mathematical Programming 106(1):25–57

Appendix

The following appendix sections contain supplementary material intended to guide software developers interested in using the presented TACO toolkit to interface their optimal control codes with AMPL. Section A explains the most important data structures that hold information about the mapping from the optimal control point of view to the AMPL one. Section B lists functions available to optimal control codes for evaluating AMPL functions. Section C explains error codes emitted by the TACO toolkit, and mentions possible remedies.

A TACO Data Structures Exposed to Optimal Control Problem Solvers

This sections lists TACO data structures exposed to developers of codes for solving optimal control problems. We discuss several snippets taken from the header file `ocp_frontend.h`, which should be consulted for additional details.

A.1 Data Structures Mapping from AMPL to Optimal Control

The optimal control frontend provides a collection of fields that hold the optimal control problem interpretation of every AMPL variable passed to the solver. They are laid out as follows:

```
enum vartype_t {
    vartype_AMPL_defined = -2,  // AMPL "defined" variable
    vartype_unknown = -1,       // type not yet known
    vartype_t       =  0,       // independent variable ("time")
    vartype_x       =  1,       // differential state of an ODE/DAE
    vartype_z       =  2,       // algebraic state of a DAE
    vartype_u       =  3,       // control function to be discretized
    vartype_p       =  4,       // global model parameter
    vartype_tend    =  5        // end time variable
};

enum vartype_t *vartypes;       // OCP types assigned to AMPL variables
int            *varindex;       // OCP vector indices assigned to AMPL variables
```

The field `vartypes` gives the OCP variable type of an AMPL variable, i.e., whether the variable is t, T, a component of vector p, or a component of one of the vector trajectories x, z, or u and w. For the case of it being a vector component, the field `varindex` holds the index into the OCP variable or trajectory vector. For AMPL constraints and objectives, no mapping information from AMPL to the OCP perspective is provided.

A.2 Data Structures Mapping from Optimal Control to AMPL

The AMPL optimal control frontend provides a collection of structures holding the AMPL perspective for every component of an optimal control problem according to problem class (1). Starting with information about problem dimensions, the following variables are provided and should be self-explanatory.

```
int nx;      // number of differential states
int nz;      // number of algebraic states
int nu;      // number of control functions
int np;      // number of model parameters
int npc;     // number of inequality path constraints
int ncc_eq;  // number of coupled equality constraints
int ncc_in;  // number of coupled inequality constraints
```

Information about both the independent time variable and the end-time variable is held in a structured variable named `endtime` of the following layout.

```
struct ocp_time_t {
    int         index;  // AMPL index of the final time variable, or -1
    const char *name;   // AMPL name of the final time variable
    int         fixed;  // flag indicating whether tf is fixed, index may be -1 then
    real        init;   // initial or fixed end time, even if idx_tf=-1
    real        scale;  // scale factor for tf
    real        lbnd;   // lower bound for tf
    real        ubnd;   // upper bound for tf

    int         idx_t;  // AMPL index of the free time variable t
    const char *name_t; // AMPL name of the free time variable
};

struct ocp_time_t endtime; // time horizon information
```

For a fixed-endtime scenario, `endtime.index` is -1, the field `endtime.init` holds the fixed end time. For a variable end-time scenario, `endtime.index` is non-negative and the field `endtime.init` holds the initial guess for the free end time if available, and is set to 1.0 otherwise.

The following structured variable `xstates` holds information about differential state trajectory variables, and associated right hand side functions.

```
struct ocp_xstate_t {
    int         index;      // AMPL index of differential state trajectory variable
    const char *name;       // AMPL name of the differential state trajectory variable
    int         fixed;      // AMPL index of initial value constraint, -1 if none
    real        init[2];    // initializers at t=0 and t=tf
    real        scale;      // scale factor
    real        lbnd;       // lower bound
    real        ubnd;       // upper bound

    int         ffcn_index; // AMPL index of ODE constraint
    const char *ffcn_name;  // AMPL name of ODE constraint
    real        ffcn_scale; // scale factor
    real        rhs_factor; // constant factor in front of diff()
};

struct ocp_xstate_t *xstates; // differential state trajectories information
```

Here, the field `fixed` holds the AMPL index of the initial value constraint for this ODE state. It is -1 if the ODE state's initial value is free. The initializer `init` provides two values for linear interpolation (see suffix `.interp_to`).

The field `ffcn_index` holds the AMPL constraint index of the right hand side function associated with a differential state. Even though we currently support explicit ODEs only, AMPL-internal rearrangement of constraint expressions may cause a negative sign on the `diff()` call. Hence the field `rhs_factor` is introduced to compensate for AMPL-internal representations of the form $-\dot{x}(t) = f(t,x(t),\ldots)$.

Similar to differential states, the structured variable `zstates` holds information about algebraic state trajectories.

```
struct ocp_zstate_t {
    int         index;  // AMPL index of algebraic state trajectory variable
    const char *name;   // AMPL name of algebraic state trajectory variable
    real        init;   // constant initial guess for algebraic state trajectory
    real        scale;  // scale factor
    real        lbnd;   // lower bound
    real        ubnd;   // upper bound

    // we keep gfcn() information here as well, but keep in mind that the relation-
    // ship between z[] and gfcn() is fully implicit, i.e. no 1-1 correspondence!

    int         gfcn_index;  // AMPL index of DAE constraint
    const char *gfcn_name;   // AMPL name of DAE constraint
    real        gfcn_scale;  // scale factor for DAE constraint
};

struct ocp_zstate_t *zstates; // algebraic state trajectories information
```

It is important to keep in mind that DAE constraints are not associated with algebraic state trajectory variables by a one–one mapping. We merely keep both in the same array for simplicity, as their numbers must match.

Information about integer and continuous control trajectories is kept in a structured variable named `controls` with the following layout.

```
struct ocp_control_t {
    int         index;       // AMPL indices of control trajectory variable
    const char *name;        // AMPL name of control trajectory variable
    int         type;        // control discretization type
    int         integer;     // flag indicating integer controls
    real        init;        // initial guess for all control parameters on the horizon
    real        scale;       // scale factor for control
    real        lbnd;        // lower bound for control
    real        ubnd;        // upper bound for control

    // slope information for linear or cubic elements
    real        slope_init;  // initial guess for control slope
    real        slope_scale; // scale factor for control slope
    real        slope_lbnd;  // lower bound for control slope
    real        slope_ubnd;  // upper bound for control slope
};

struct ocp_control_t *controls; // control trajectories information
```

Herein, the field `type` denotes the (solver-dependent) discretization type to be applied to this control trajectory The field `integer` is set to 1 if the control is a binary or integer control, and to 0 if it is a continuous control. For piecewise linear and piecewise cubic discretization types, additional information about slope limits and initial guesses is provided.

Finally, model parameters information is provided in a structured variable named `params`, with layout as follows. This concludes AMPL variables information.

```
struct ocp_param_t {
    int         index;   // AMPL indices of free model parameters
    const char *name;    // AMPL name of free model parameters
    real        init;    // initial value for free parameter
    real        scale;   // scale factors for free parameter
    real        lbnd;    // lower bound for free parameter
    real        ubnd;    // upper bound for free parameter
    int         integer; // 1 if integer, 0 if real
};

struct ocp_param_t *params; // free parameters information
```

Information about Mayer-type, Lagrange-type, and integral least-squares-type objective functions (1a) is found in structured variables named `mayer`, `lagrange`, and `clsq`. Their layout is presented below.

```
struct ocp_objective_t {
    int         index;    // AMPL index of objective function
    const char *name;     // AMPL name of objective function
    int         dpnd;     // dependency flags from enum vartype_t
    int         maximize; // 1 for maximization, 0 for minimization
    real        scale;    // scale factor

    // additional information for least-squares type objectives
    int         nparts;   // number of residuals
    expr      **parts;    // residual AMPL expressions
};

struct ocp_objective_t mayer;    // Mayer type objective function information
struct ocp_objective_t lagrange; // Lagrange type objective function information
struct ocp_objective_t clsq;     // Integral least-squares type objective information
```

Herein, `dpnd` is a bit field with bit k ($k \geq 0$) set if and only if the objective function's AMPL expression depends on an AMPL variable with `vartypes` entry set to value k (see `enum vartype_t`). This allows for quick dependency checks that may save run time e.g. in derivative approximation. The field `maximize` is set to 1 if the objective function is to be maximized, and to 0 if it is to be minimized. Note that least-squares functions are recognized only if they are to be minimized. The fields `nparts` and `parts` hold information about the AMPL expression DAGs associated with the individual least-squares residual expressions of a least-squares objective.

Path constraints (1d) and coupled constraints (1e) information is held in structured variables named `pathcon`, `cpcon_eq`, and `cpcon_in` with the following layout.

```
struct ocp_constraint_t {
    int         index; // AMPL index of constraint
    const char *name;  // AMPL name of constraint
    int         side;  // side of inequality constraint (0=lower, 1=upper)
    int         dpnd;  // dependency flags from enum vartype_t
    real        scale; // scale factor
};

struct ocp_constraint_t  *pathcon; // inequality path constraints
struct ocp_constraint_t  *cpcon_eq; // coupled equality constraints
struct ocp_constraint_t  *cpcon_in; // coupled inequality constraints
```

Path constraints (1d) always are inequality constraints. For coupled constraints (1e), equality and inequality constraints are stored in separate arrays. For two-sided inequality constraints $l \leq c(x) \leq u$, the field `side` indicates which side of the constraint should be evaluated.

Information about decoupled point constraints (1f) and point least-squares objectives (1a) is stored in a structured variable named `grid`. For each objective or constraint evaluation time, a grid node is introduced and holds information about the associated objective or constraint. Grid nodes are guaranteed to be unique, i.e. no two nodes share the same time point, and are sorted in ascending order. The grid is guaranteed to contain at least two nodes: The first grid node will always be at time 0, and the last grid node will always be at time t_f.

```
struct ocp_grid_node_t {
    int                     n_eq;     // number of equality point constraints
```

```
    struct ocp_constraint_t *con_eq;   // equality point constraint information
    int                      n_in;     // number of inequality point constraints
    struct ocp_constraint_t *con_in;   // inequality point constraint information
    int                      ncc_eq;   // number of equality coupled constraints
    int                     *ccidx_eq; // indices of equality coupled constraints
    int                      ncc_in;   // number of inequality coupled constraints
    int                     *ccidx_in; // indices of inequality coupled constraints
    struct ocp_objective_t   lsq;      // node least-squares objective information
};

struct ocp_grid_t {
    int                     n_nodes;  // number of grid nodes
    real                   *times;    // it's more practical to have the times here
    struct ocp_grid_node_t *nodes;    // information about what is on a grid node
};

struct ocp_grid_t grid;      // constraint and node-least-squares grid information
```

For equality and inequality point constraints (1f) on a grid node, the fields `n_eq` and `n_in` hold the dimensions and the fields `con_eq` and `con_in` the constraint information, respectively. For coupled constraints (1e), we do not store pointers to constraint information structures, but rather indices into the global lists `cpcon_eq` and `cpcon_in`. For solvers requiring linear separability of coupled constraints, this layout eases the setup of the coupled constraints' block structure. Finally, the field `lsq` holds information about the point least-squares objective contribution in a node; again index -1 indicates that no point least-squares objective is present.

B TACO Functions Exposed to Optimal Control Problem Solvers

This sections lists TACO functions exposed to developers of codes for solving optimal control problems.

Reading of the AMPL Model For reading and verifying the AMPL mode as well as creation of management of the database the optimal control frontend provides the following functions:

`allocate_mappings` allocates memory for the database to be created, prior to reading the *stub.nl* file provided by AMPL.

`read_mappings` calls the AMPL solver library to read the *stub.nl* file. Afterwards, the DAGs of all AMPL objectives and constraints are examined for appearance of variables, calls to user functions, and nonsmooth operators. The role of AMPL variables, constraints, and objectives in (1) is determined, and the database is filled with appropriate information.

`free_mappings` frees memory allocated for the database after the optimal control problem has been solved.

A developer wishing to interface his optimal control problem solver with AMPL is provided with a number of functions that infer the optimal control problem's structure from a *stub.nl* file. The following listing shows a framework that could serve as a starting point for development of a solver interface.

```
// allocate memory for the "fg" reader of AMPL stub files
ASL_alloc (ASL_read_fg);
```

```
asl->i.want_xpi0_    = 1; // indicate that we want the initial guesses
asl->p.need_funcadd_ = 1; // indicate that we need to add user functions

// get AMPL stub file name and allocate memory for AMPL data structures
nl_name = getstub (&argv, &Oinfo);
nl = jac0dim (nl_name, (fint) strlen (nl_name));
if (nl == NULL)
    error_out (error_readfile, nl_name);

// declare AMPL OCP frontend suffixes
declare_suffixes ();

// read AMPL stub file
if (fg_read(nl, ASL_return_read_err|ASL_findgroups) != 0)
    error_out (error_readfile, nl_name);

Oinfo.option_echo &= ~ASL_OI_echo;      // don't echo options
Oinfo.option_echo |= ASL_OI_badvalue;   // but report bad option values

// read solver options
if (getopts(&xargv[3], &Oinfo))
    error_out (error_solverOptions);

allocate_mappings ();       // allocate memory of AMPL OCP frontend data
read_mappings ();           // infer OCP structure from AMPL stub file

// call optimal control problem solver here
// use frontend information to set up problem and evaluate problem functions

free_mappings ();           // free memory of AMPL OCP frontend data
```

Evaluating Problem Functions Once the database has been successfully filled with the optimal control problem representation of the AMPL model, an appropriate optimal control problem solver can be called. This solver will have to evaluate problem functions. For conveniently doing so, the optimal control frontend provides a set of functions that wrap around the `conival` and `objival` functions provided by AMPL's solver library:

`evaluate_ode_rhs(i,y)` evaluates the ODE right hand side $\dot{x}_i(t) = f_i(\cdot)$ for a differential state $x_i(t)$, $0 \leq i < n_{\mathrm{x}}$. The vector `y` here is the plain vector of current AMPL variable values.

`evaluate_dae_rhs(i,y)` evaluates the residual of DAE constraint $g_i(\cdot)$, $0 \leq i < n_{\mathrm{z}}$.

`evaluate_scaled_mayer(y)` evaluates and scales the value of the Mayer objective term, if a Mayer term exists.

`evaluate_scaled_lagrange(y)` evaluates and scales the value of the Lagrange objective term's integrand, if a Lagrange objective exists.

`evaluate_scaled_clsq(y,res)` evaluates and scales the value of the Lagrange objective term's integrand, if the integrand has least-squares structure and if a Lagrange objective exists. Upon return the vector `res` holds the least-squares residuals, i.e. the individual values of each squared summand.

`evaluate_scaled_node_lsq(n,y,res)` evaluates one summand of a point least-squares objective term, if one exists. The integer `n` specified the number $0 < n < N_{\mathrm{lsq}}$ of the measurement point to be evaluated.

`evaluate_path_con(i,y)` evaluates and scales the residual of an inequality path constraint.

`evaluate_coupled_eq_con(i,y)` evaluates and scales the residual of a coupled equality constraint.
`evaluate_coupled_ineq_con(i,y)` evaluates and scales the residual of a coupled inequality constraint.
`evaluate_point_eq_con(n,i,y)` evaluates and scales the residual of an equality point constraint.
`evaluate_point_ineq_con(n,i,y)` evaluates and scales the residual of an inequality point constraint.

Writing Solutions from a Solver For writing a solution file of the proposed format, the optimal control frontend provides three function calls to be used by a backend interfacing an optimal control problem solver with AMPL:

`alloc_solution` allocates a solution structure given a discretization grid length.
`write_solution` writes a solution file according to the proposed format and fills it with the solution data found in the solution structure. The solver backend should have queried the solver itself for this solution and copied it over to this solution structure.
`free_solution` free a previously allocated solution structure after it has been written to a solution file.

C TACO Toolkit Error Messages

This section lists all possible error messages emitted by the TACO toolkit during analysis and verification of the optimal control problem structure. If option *solvername*`_auxfiles` is set to contain the characters "`r`" and "`c`", error messages will show the name of the offending variable, objective, or constraint.

1. to 5. are reserved for internal errors.
6. "`AMPL error reading stub.nl file` *filename*". AMPL could not read the *stub.nl* file.
7. "`AMPL error getting solver options`". AMPL could not read the solver options. Make sure your solver options string contains known keywords and valid values only.
8. "`I/O error writing solution file` *filename*". AMPL could not write the solution file. Make sure that the current working directory is writeable and that the volume is not full.
9. "`AMPL network variables are not supported`". The AMPL model contains network variables. Network modelling is currently not supported.
10. "`AMPL piecewise linear terms are not supported`". The AMPL model contains piecewise linear terms. Nonsmooth models are currently not supported.
11. "`diff() must be called exactly once for each differential state variable`".
    You called `diff()` more than once for the same variable. Every differential state variable is associated with exactly one right hand side function, tough. Most likely, this is a typo in your model.

12. "`Logical operator` *operator* `not permitted`". The shown constraint or objective make use of a logical operator. Nonsmooth models are currently not supported.
13. "`Relational operator` *operator* `not permitted`". The shown constraint or objective make use of a relational operator. Nonsmooth models are currently not supported.
14. "`Nonsmooth operator` *operator* `not permitted`". The shown constraint or objective make use of a nonsmooth operator. Nonsmooth models are currently not supported.
15. "`ODE constraints must be equality constraints`". You called `diff()` in an inequality constraint. All ODE constraints must be equality constraints, though.
16. "`Cannot determine type of objective functions`". The shown objective cannot be determined to be of Mayer, Lagrange, integral least-squares, or point least-squares type. Please refer to Section 2.3 to learn about the structure of supported objective functions.
17. "`Mayer objective function must not depend on controls`". The indicated objective is of Mayer type, and hence must not depend on control trajectory variables.
18. "`The number of DAE constraints does not match the apparent number of algebraic variables`". The number of DAE constraints found in your model does not match the number of algebraic variables you have declared by letting suffix `.type` to value `"dae"`. Make sure you've set the suffix of all algebraic state variables, and make sure all your DAE constraints are equality constraints.
19. "`Invalid value for suffix .type`". A variable, constraint, or objective has suffix `.type` set to an unrecognized value. This should not happen unless you're using an incompatible `OptimalControl.mod` file.
20. "`Missing independent time variable or end time variable`". TACO could not determine which AMPL variables represent independent time or end time. Make sure that the independent time variable appears as second argument to all `diff()` calls, and that the end-time variable appears as second argument to `integral()` in Lagrange-type objectives, or to `eval()` in Mayer-type objectives or end-point constraints. If your AMPL model uses a time horizon of fixed length, make sure the end-time value shows up consistently in these places.
21. "`Nesting a call to function` *name* `inside another is not allowed`". You nested two calls to functions `diff()`, `eval()`, or `integral()`. This should never be necessary.
22. "`As of now, diff() must be explicit. Try to write your ODE as diff(var, t) = ...;`". The AMPL optimal control frontend currently supported explicit ODEs only, i.e. the permitted formulation is $\dot{x}(t) = f(t,x(t),\ldots)$ instead of the more general $0 = f(t,x(t),\dot{x}(t),\ldots)$. Try to rewrite your model to use explicit ODEs only.
23. "`First argument to eval() must not be a constant`". The shown constraint or objective calls `eval()` with two constant arguments. This is not sensible. Either remove the calls to `eval()`, or make sure the expression to be eval-

uated references independent time or at least one state or control trajectory variable.

24. "`Second argument to eval() must be >= 0.0, or the final time variable`". The shown constraint or objective calls `eval()` with a time point that a) evaluates to a negative value, or b) is not a simple constant and not the end-time variable.
25. "`First argument to integral() must not be a constant`". An objective calls `integral()` with two constant arguments. This is not sensible. Either remove the call to `integral()`, or make sure the expression to be integrated references independent time or at least one state or control trajectory variable.
26. "`Second argument to integral() must be the final time variable`". The second argument of a call to `integral()` must be the final time variable, or the final time itself.
27. "`Second argument to diff() must be the independent time variable`". The second argument of a call to `diff()` must be the independent time variable.
28. "`Function diff() expects differential state variable and independent time variable as arguments`". The first argument to `diff` must be a differential state trajectory variable, and the second argument of a call to `diff()` must be the independent time variable.
29. "`Calls to integral() not allowed in constraints`". The function `integral()` was called inside a constraint expression. Calls to `integral()` are only allowed in a Lagrange-type or integral least-squares-type objective.
30. "`Function call to` *name* `must enclose entire expression`". Calls to `eval()` and `integral()` must enclose the entire expression to be evaluated or integrated.
31. "`Only controls or parameters can be binary/integer variables`". AMPL's `binary` or `integer` restriction may be applied to control trajectory variables and model parameter variables only. Time as well as differential and algebraic state trajectory variables must be continuous.
32. "`Invalid grid node position, must be within [0,1] (tf free) or [0,tf] (tf fixed)`". A call to `eval()` used an invalid evaluation time. Allowed times are constants 0 to 1, being relative times on the time horizon $[0, t_f]$ if t_f is free, absolute times 0 to t_f if t_f is fixed, or the end-time variable t_f itself.
33. "`Duplicate objective of` *name*`-type`". At most one objective of Mayer-, Lagrange-, and least-squares type is allowed per problem.
34. "`Calls to eval() not allowed in ODE constraints`". The function `eval()` was called inside an ODE constraint expression. This is not allowed.
35. "`Calls to diff() not allowed in objective functions`". The function `diff()` was called inside an objective function expression. This is not allowed.