COOPT - A Software Package for Optimal Control of Large-Scale Differential-Algebraic Equation Systems ⋆

Radu Serban [1] and Linda R. Petzold [2]

Department of Mechanical and Environmental Engineering University of California, Santa Barbara, CA 93106

Abstract

This paper describes the functionality and implementation of COOPT. This software package implements a direct method with modified multiple shooting type techniques for solving optimal control problems of large-scale differential-algebraic equation (DAE) systems. The basic approach in COOPT is to divide the original time interval into multiple shooting intervals, with the DAEs solved numerically on the subintervals at each optimization iteration. Continuity constraints are imposed across the subintervals. The resulting optimization problem is solved by sparse sequential quadratic programming (SQP) methods. Partial derivative matrices needed for the optimization are generated by DAE sensitivity software. The sensitivity equations to be solved are generated via automatic differentiation.

COOPT has been successfully used in solving optimal control problems arising from a wide variety of applications, such as chemical vapor deposition of superconducting thin films, spacecraft trajectory design and contingency/recovery problems, and computation of cell traction forces in tissue engineering.

Key words: DAE, optimal control, ...

⋆ This research was supported in part by NSF KDI ATM-9873133, NSF-ARPA PC 239415, NSF CCR 98-96198, and DOE DE-FG03-98ER25354.

[1] Email: radu@engineering.ucsb.edu

[2] Email: petzold@engineering.ucsb.edu

1 Introduction

We consider the differential-algebraic equation (DAE) system

$$\begin{aligned} \mathbf{F}(t, \mathbf{x}, \mathbf{x}', \mathbf{p}, \mathbf{u}(t)) &= 0 \\ \mathbf{x}(t_1) &= \mathbf{x}_1 \end{aligned} \tag{1}$$

where the DAE is index one (see [5]) and the initial conditions have been chosen so that they are consistent (so that the constraints of the DAE are satisfied). The control parameters $\mathbf{p}$ and the vector-valued control function $\mathbf{u}(t)$ must be determined such that the scalar objective function

$$\int_{t_1}^{t_{\max}} \Psi(t, \mathbf{x}(t), \mathbf{p}, \mathbf{u}(t))\, dt$$

is minimized and some additional inequality constraints

$$G(t, \mathbf{x}(t), \mathbf{p}, \mathbf{u}(t)) \geq 0$$

are satisfied. The optimal control function $\mathbf{u}^*(t)$ is assumed to be continuous. In many applications, the DAE system is large-scale, that is the dimension N_x of $\mathbf{x}$ may be large. In some of our applications [11] $N_x \approx 1000$, but even larger problems can be considered. However, the dimensions of the parameters $\mathbf{p}$ and of the representation of the control function $\mathbf{u}(t)$ are assumed here to be much smaller. To represent $\mathbf{u}(t)$ in a low-dimensional vector space, we use piecewise polynomials on $[t_1, t_{\max}]$, their coefficients being determined by the optimization. For ease of presentation we can therefore assume that the vector $\mathbf{p}$ contains both the parameters and these coefficients (we let M denote the combined number of these values) and discard the control function $\mathbf{u}(t)$ in the remainder of this section. Also, we consider that the initial states are fixed and therefore discard the dependency of $\mathbf{x}_1$ on $\mathbf{r}$. Hence the problem is given by

$$\mathbf{F}(t, \mathbf{x}, \mathbf{x}', \mathbf{p}) = 0, \qquad \mathbf{x}(t_1) = \mathbf{x}_1, \tag{2a}$$

$$\int_{t_1}^{t_{\max}} \psi(t, \mathbf{x}(t), \mathbf{p})\, dt \qquad \text{is minimized}, \tag{2b}$$

$$\mathbf{g}(t, \mathbf{x}(t), \mathbf{p}) \geq 0. \tag{2c}$$

There are a number of well-known methods for direct discretization of this optimal control problem (2), for the case that the DAEs can be reduced to ordinary differential equations (ODEs) in standard form [3]. The *single shooting method* solves the ODEs (2a) over the interval $[t_1, t_{\max}]$, with the set of controls generated at each iteration by the optimization algorithm. However, it is well-known that single shooting can suffer from a lack of stability and robustness [1]. Moreover, for this method it is more difficult to maintain additional

constraints and to ensure that the iterates are physical or computable. The *finite-difference method* or *collocation method* discretizes the ODEs over the interval $[t_1, t_{\max}]$. The ODE solutions at each discrete time and the set of controls are generated at each iteration by the optimization algorithm. Although this method is more robust and stable than the single shooting method, it requires the solution of an optimization problem which for a large-scale ODE system is enormous, and it does not allow for the use of adaptive ODE or (in the case that the ODE system is the result of semi-discretization of PDEs) PDE software.

We thus consider the *multiple-shooting method* for the discretization of (2). In this method, the time interval $[t_1, t_{\max}]$ is divided into subintervals $[t_i, t_{i+1}]$ $(i = 1, \ldots, N_{ms})$, and the differential equations (2a) are solved over each subinterval, where additional intermediate variables $\mathbf{X}_i$ are introduced. On each subinterval we denote the solution at time t of (2a) with initial value $\mathbf{X}_i$ at t_i by $\mathbf{x}(t, t_i, \mathbf{X}_i, \mathbf{p})$.

Continuity between subintervals is achieved via the continuity constraints

$$\mathbf{C}_1^i(\mathbf{X}_{i+1}, \mathbf{X}_i, \mathbf{p}) \equiv \mathbf{X}_{i+1} - \mathbf{x}(t_{i+1}, t_i, \mathbf{X}_i, \mathbf{p}) = \mathbf{0}.$$

For the DAE solution to be defined on each multiple shooting subinterval, it must be provided with a set of initial values which are consistent (that is, the initial values must satisfy any algebraic constraints in the DAE). This is not generally the case with initial values provided by methods like sequential quadratic programming (SQP) because these methods are not feasible (in other words, intermediate solutions generated by the optimizer do not necessarily satisfy constraints in the optimization problem although the final solution does). To begin each interval with a consistent set of initial values, we first project the intermediate solution generated by SNOPT onto the constraints, and then solve the DAE system over the subinterval. In the case of index-1 problems with well-defined algebraic variables and constraints, this means that we perturb the intermediate initial values of the algebraic variables so that they satisfy the constraints at the beginning of each multiple shooting subinterval. The additional constraints (2c) are required to be satisfied at the boundaries of the shooting intervals

$$\mathbf{C}_2^i(\mathbf{X}_i, \mathbf{p}) \equiv \mathbf{g}(t_i, \mathbf{X}_i, \mathbf{p}) \geq \mathbf{0}.$$

Following common practice, we write

$$\Phi(t) = \int_{t_1}^{t} \psi(\tau, \mathbf{x}(\tau), \mathbf{p})\, d\tau, \tag{3}$$

which satisfies $\Phi'(t) = \psi(t, \mathbf{x}(t), \mathbf{p})$, $\Phi(t_1) = 0$. This introduces another equation and variable into the differential system (2a). The discretized optimal

control problem becomes

$$\min_{\mathbf{X}_2,\ldots,\mathbf{X}_{N_{ms}},\mathbf{p}} \Phi(t_{\max}) \tag{4}$$

subject to the constraints

$$\mathbf{C}_1^i(\mathbf{X}_{i+1}, \mathbf{X}_i, \mathbf{p}) = \mathbf{0}, \tag{5a}$$
$$\mathbf{C}_2^i(\mathbf{X}_i, \mathbf{p}) \geq \mathbf{0}. \tag{5b}$$

This problem can be solved by an optimization code. We use the solver SNOPT [7], which incorporates an SQP method. The SQP methods require a gradient and Jacobian matrix that are the derivatives of the objective function and constraints with respect to the optimization variables. We compute these derivatives via highly efficient DAE sensitivity software DASPK3.0 [8]. The sensitivity equations to be solved by DASPK3.0 are generated via the automatic differentiation software ADIFOR [4].

This basic multiple-shooting type of strategy can work very well for small-to-moderate size ODE systems, and has an additional advantage that it is inherently parallel. However, for large-scale ODE and DAE systems there is a problem because the computational complexity grows rapidly with the dimension of the ODE system. The difficulty lies in the computation of the derivatives of the continuity constraints (5a) with respect to the variables $\mathbf{X}_i$. The work to compute the derivative matrix $\partial\mathbf{x}(t)/\partial\mathbf{X}_i$ is of order $O(N_x^2)$, and for the problems under consideration N_x can be very large (for example, for an ODE system obtained from the semi-discretization of a PDE system, N_x is the product of the number of PDEs and the number of spatial grid points). In contrast, the computational work for the single shooting method is of order $O(N_x N_p)$ although the method is not as stable, robust or parallelizable.

We reduce the computational complexity of the multiple shooting method for this type of problem by modifying the method to make use of the structure of the continuity constraints to reduce the number of sensitivity solutions which are needed to compute the derivatives. To do this, we recast the continuity constraints in a form where only the matrix-vector products $(\partial\mathbf{x}(t)/\partial\mathbf{X}_i)\mathbf{w}_j$ are needed, rather than the entire matrix $\partial\mathbf{x}(t)/\partial\mathbf{X}_i$. The matrix-vector products are directional derivatives; each can be computed via a single sensitivity analysis. The number of vectors $\mathbf{w}_j$ such that the directional sensitivities are needed is small, of order $O(N_p)$. Thus the complexity of the modified multiple shooting computation is reduced to $O(N_x N_p)$, roughly the same as that of single shooting. Unfortunately, the reduction in computational complexity comes at a price: the stability of the modified multiple shooting algorithm suffers from the same limitations as single shooting. However, for many PDE systems, including the applications described here, this is not an issue, and the modified method is more robust for nonlinear problems. This is due to the

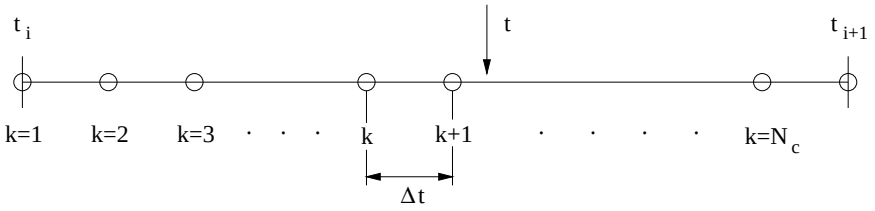

Fig. 1. Control subintervals within a shooting interval

fact that optimal control is usually considered for problems which are already simulated and are thus stable from the left. Further details on the algorithm can be found in [6].

In the context of the SQP method, the use of modified multiple shooting involves a transformation of the constraint Jacobian. The affected rows are those associated with the continuity constraints and any path constraints applied within the shooting intervals. Path constraints enforced at the shooting points (and other constraints involving only discretized states) are not transformed. The transformation is cast almost entirely at the user level and requires minimal changes to the optimization software, which is important because software in this area is constantly being modified and improved. Gill et al. ([6]) have shown that the modified quadratic subproblem yields a descent direction for the ℓ_1 penalty function. DAOPT is a modification to the SNOPT optimization code that uses a merit function based on an ℓ_1 penalty function.

2 Description

In this section we describe in more detail the basic structure and implementation of COOPT. We start by describing the control parameterization adopted in COOPT. In Section 2.2 we discuss the optimization constraints of the discretized optimal control problem (4) with emphasis on the constraints imposing state and control continuity at the multiple shooting interfaces. Section 2.3 describes the solution of the state and sensitivity equations, while Section 2.4 presents a simple method for computing the sensitivity of the solution of the optimal control problem with respect to perturbations in equality and/or inequality constraints.

2.1 Control Parameterization

On each multiple shooting interval, each control u_j, $j = 1, 2, ..., N_u$ is represented as a piecewise polynomial of order N_q. Each multiple shooting interval is subdivided into N_c control intervals (see Fig. 1). The control u_j is then parameterized by a minimum number of parameters that ensure continuity of the

control and of its derivative; i.e., $u_j \in \mathcal{C}^1$. Consider the multiple shooting interval $[i, i+1]$. The length of each control subinterval is then $\Delta t_i = (t_{i+1} - t_i)/N_c$. Let k be such that $k \cdot \Delta t_i \leq t - t_i < (k+1) \cdot \Delta t_i$. Consider the order N_q polynomial approximation of control u_j at t, using the nondimensionalized variable $t^* = (t - t_i - k \cdot \Delta t_i)/\Delta t_i \in [0, 1)$

$$\begin{aligned} u_j(t) &= U_{j,i}^{k+1,0} + U_{j,i}^{k+1,1} t^* + \sum_{q=2}^{N_q} U_{j,i}^{k+1,q} t^{*q} \\ \Delta t_i \cdot u_j'(t) &= U_{j,i}^{k+1,1} + \sum_{q=2}^{N_q} U_{j,i}^{k+1,q} q t^{*q-1} \qquad\qquad k = 0, 1, 2, ..., N_c \end{aligned} \tag{6}$$

Imposing the conditions $u_j \in \mathcal{C}^1$, the parameters $U_{j,i}^{k+1,0}$ and $U_{j,i}^{k+1,1}$ must satisfy the following recursive relations:

$$\begin{aligned} U_{j,i}^{k+1,0} &= U_{j,i}^{k,0} + U_{j,i}^{k,1} + \sum_{q=2}^{N_q} U_{j,i}^{k,q} \\ U_{j,i}^{k+1,1} &= U_{j,i}^{k,1} + \sum_{q=2}^{N_q} q U_{j,i}^{k,q} \qquad\qquad k = 1, 2, ..., N_c \end{aligned} \tag{7}$$

By induction, we can see that

$$\begin{aligned} U_{j,i}^{k+1,0} &= U_{j,i}^{1,0} + k \cdot U_{j,i}^{1,1} + \sum_{k1=1}^{k} \sum_{q=2}^{N_q} [q(k - k1) + 1] U_{j,i}^{k1,q} \\ U_{j,i}^{k+1,1} &= U_{j,i}^{1,1} + \sum_{k1=1}^{k} \sum_{q=2}^{N_q} q U_{j,i}^{k1,q} \qquad\qquad k = 0, 1, 2, ..., N_c \end{aligned} \tag{8}$$

The piece-wise polynomial approximation of u_j in the shooting interval $[i, i+1]$ can thus be represented by the following $2 + N_c(N_q - 1)$ parameters:

$$
\begin{array}{cccccc}
U_{j,i}^{1,0} & & & & & \\
U_{j,i}^{1,1} & & & & & \\
U_{j,i}^{1,2} & U_{j,i}^{2,2} & \cdots & U_{j,i}^{k,2} & U_{j,i}^{k+1,2} & \cdots & U_{j,i}^{N_c,2} \\
U_{j,i}^{1,3} & U_{j,i}^{2,3} & \cdots & U_{j,i}^{k,3} & U_{j,i}^{k+1,3} & \cdots & U_{j,i}^{N_c,3} \\
\vdots & \vdots & & \vdots & \vdots & & \vdots \\
U_{j,i}^{1,N_q} & U_{j,i}^{2,N_q} & \cdots & U_{j,i}^{k,N_q} & U_{j,i}^{k+1,N_q} & \cdots & U_{j,i}^{N_c,N_q}
\end{array}
$$

The total number of control parameters is then $N_{ms}N_u(2 + N_c(N_q - 1))$.

Such a parameterization of the control has two advantages over the approach in which an order N_q polynomial is used to represent the control on each control subinterval $[k, k+1]$, with $\mathcal{C}^1$ control continuity enforced by the optimization algorithm. First, a reduced number of parameters is required, $N_{ms}N_u(2 + N_c(N_q - 1))$ vs. $N_{ms}N_uN_c(N_q + 1)$, which means that fewer sensitivity equations must be solved. Using a separate polynomial of order N_q for each control subinterval introduces discontinuities in the sensitivities with respect to parameters $U_{j,i}^{k,0}$ and therefore, a consistent initial condition computation must be performed at the beginning of each control subinterval. With the current control parameterization the integration can be carried out over the entire multiple shooting interval without restarts at the beginning of each control subinterval.

2.2 Optimization Constraints

In a multiple shooting type method, continuity of states and controls at the multiple shooting points must be enforced. We impose $\mathcal{C}^0$ conditions on the states (including the additional state for the cost function) and $\mathcal{C}^1$ conditions on the controls. These conditions result in nonlinear state continuity constraints and linear control continuity constraints.

The complete set of constraints in the discretized optimal control problem (4) is then obtained by collecting the user-defined constraints, the state continuity constraints, the control continuity constraints, and some additional control

constraints (see Section 2.2.3).

2.2.1 State Continuity Constraints

The state equations (1) are solved on each multiple shooting interval $[t_i, t_{i+1}]$, $i = 1, 2, ..., N_{ms}$. We denote the solution at time t of (1) with initial value $\mathbf{X}_i$ at t_i by $\mathbf{x}(t, t_i, \mathbf{X}_i, \mathbf{p}, \mathbf{u})$. Continuity of states between subintervals is achieved via the nonlinear constraints

$$\begin{aligned} &\mathbf{C}_i \equiv \mathbf{X}_{i+1} - \mathbf{x}(t_{i+1}, t_i, \mathbf{X}_i, \mathbf{p}, \mathbf{u}) = \mathbf{0} \\ &\text{where } \mathbf{C}_i \in \mathbf{R}^{N_x+1}, \quad i = 1, 2, ..., N_{ms} \end{aligned} \tag{9}$$

We denote by

$$\mathbf{X} = \{\mathbf{X}_2, \mathbf{X}_3, ..., \mathbf{X}_{N_{ms}+1}\} \in \mathbf{R}^{N_{ms}(N_x+1)}$$

the vector of discretized states and by

$$\mathbf{U} = \{\mathbf{U}_1, \mathbf{U}_2, ..., \mathbf{U}_{N_{ms}}\} \in \mathbf{R}^{N_{ms}N_u(2+N_c(N_q-1))}$$

the vector of control parameters, with $\mathbf{U}_i$ representing the parameterization of all controls on the interval $[t_i, t_{i+1}]$. Note that the states $\mathbf{X}_1 \in \mathbf{R}^{N_x+1}$ at $t = t_1$ are excluded from the array $\mathbf{X}$. Next, we collect the constraints (9) into the array

$$\mathbf{C}(\mathbf{p}, \mathbf{X}, \mathbf{U}) \equiv \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ \vdots \\ \mathbf{C}_i \\ \vdots \\ \mathbf{C}_{N_{ms}} \end{bmatrix} \equiv \begin{bmatrix} \mathbf{X}_2 - \mathbf{x}(t_2, t_1, \mathbf{X}_1, \mathbf{p}, \mathbf{U}_1) \\ \mathbf{X}_3 - \mathbf{x}(t_3, t_2, \mathbf{X}_2, \mathbf{p}, \mathbf{U}_2) \\ \vdots \\ \mathbf{X}_{i+1} - \mathbf{x}(t_{i+1}, t_i, \mathbf{X}_i, \mathbf{p}, \mathbf{U}_i) \\ \vdots \\ \mathbf{X}_{N_{ms}+1} - \mathbf{x}(t_{N_{ms}+1}, t_{N_{ms}}, \mathbf{X}_{N_{ms}}, \mathbf{p}, \mathbf{U}_{N_{ms}}) \end{bmatrix} = \mathbf{0} \tag{10}$$

The Jacobian of these constraints with respect to the vector of optimization parameters $[\mathbf{p}, \mathbf{X}, \mathbf{U}]$ is

$$\mathbf{J} = [\mathbf{J}_\mathbf{p}, \mathbf{J}_\mathbf{X}, \mathbf{J}_\mathbf{U}] \tag{11}$$

where

$$\mathbf{J_p} = \begin{bmatrix} -\partial\mathbf{x}(t_2)/\partial\mathbf{p} \\ -\partial\mathbf{x}(t_3)/\partial\mathbf{p} \\ \vdots \\ -\partial\mathbf{x}(t_{i+1})/\partial\mathbf{p} \\ \vdots \\ -\partial\mathbf{x}(t_{N_{ms}+1})/\partial\mathbf{p} \end{bmatrix} \tag{12}$$

$$\mathbf{J_X} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \cdots & \mathbf{0} & \mathbf{0} \cdots & \mathbf{0} & \mathbf{0} \\ -\frac{\partial\mathbf{x}(t_3)}{\partial\mathbf{X}_2} & \mathbf{I} & \mathbf{0} \cdots & \mathbf{0} & \mathbf{0} \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\frac{\partial\mathbf{x}(t_4)}{\partial\mathbf{X}_3} & \mathbf{I} \cdots & \mathbf{0} & \mathbf{0} \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \cdots & -\frac{\partial\mathbf{x}(t_{i+1})}{\partial\mathbf{X}_i} & \mathbf{I} \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \cdots & \mathbf{0} & \mathbf{0} \cdots & -\frac{\partial\mathbf{x}(t_{N_{ms}+1})}{\partial\mathbf{X}_{N_{ms}}} & \mathbf{I} \end{bmatrix} \tag{13}$$

$$\mathbf{J_U} = \begin{bmatrix} -\frac{\partial\mathbf{x}(t_2)}{\partial\mathbf{U}_1} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & -\frac{\partial\mathbf{x}(t_3)}{\partial\mathbf{U}_2} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -\frac{\partial\mathbf{x}(t_{i+1})}{\partial\mathbf{U}_i} & \cdots & \mathbf{0} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & -\frac{\partial\mathbf{x}(t_{N_{ms}+1})}{\partial\mathbf{U}_{N_{ms}}} \end{bmatrix} \tag{14}$$

In the previous equations, we have used the simplified notation $\mathbf{x}(t_{i+1}) \equiv \mathbf{x}(t_{i+1}, t_i, \mathbf{X}_i, \mathbf{p}, \mathbf{U}_i)$.

The computational complexity of the basic multiple shooting method results mainly from computation of the derivative matrix $\mathbf{J_X}$. For problems where N_x is large, the time to compute this Jacobian is prohibitive. To overcome this problem, we first note that, as Eq. (13) shows, the Jacobian $\mathbf{J_X}$ is nonsingular. The matrix $\mathbf{J_X^{-1}}$ can be used to transform the state continuity constraints (9) so that the number of sensitivities is reduced.

Multiplying the constraints $\mathbf{C}$ and the Jacobian $\mathbf{J}$ to the left by $\mathbf{J_X^{-1}}$, we obtain

modified constraints and a modified Jacobian

$$\hat{\mathbf{C}} = \mathbf{J}_{\mathbf{X}}^{-1}\mathbf{C}, \tag{15}$$

$$\hat{\mathbf{J}} = [\mathbf{J}_{\mathbf{X}}^{-1}\mathbf{J}_{\mathbf{p}}, \mathbf{J}_{\mathbf{X}}^{-1}\mathbf{J}_{\mathbf{X}}, \mathbf{J}_{\mathbf{X}}^{-1}\mathbf{J}_{\mathbf{U}}] \equiv [\hat{\mathbf{P}}, \mathbf{I}, \hat{\mathbf{U}}]. \tag{16}$$

The important feature of this modified Jacobian is that the sensitivities associated with the identity block need not be computed. The modified constraints (15) are the solution of the system $\mathbf{J}_{\mathbf{X}}\hat{\mathbf{C}} = \mathbf{C}$, while the remaining two blocks in the modified Jacobian are solutions of the matrix systems $\mathbf{J}_{\mathbf{X}}\hat{\mathbf{P}} = \mathbf{J}_{\mathbf{p}}$ and $\mathbf{J}_{\mathbf{X}}\hat{\mathbf{U}} = \mathbf{J}_{\mathbf{U}}$, respectively. The modified constraints $\hat{\mathbf{C}}$ and the matrices $\hat{\mathbf{P}}$ and $\hat{\mathbf{U}}$ can be computed iteratively, block by block, outside the optimization code. Indeed, to compute the matrix $\hat{\mathbf{C}}$, we partition it into N_{ms} vertical blocks. We find that

$$\hat{\mathbf{C}}_1 = \mathbf{C}_1, \qquad \hat{\mathbf{C}}_i = \mathbf{C}_i + \frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{C}}_{i-1}, \quad i = 2, 3, ..., N_{ms}. \tag{17}$$

With a similar partition for $\hat{\mathbf{P}}$ we get

$$\hat{\mathbf{P}}_1 = -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{p}}, \qquad \hat{\mathbf{P}}_i = -\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{p}} + \frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{P}}_{i-1}, \quad i = 2, 3, ..., N_{ms}. \tag{18}$$

With a block-lower triangular partition of the matrix $\hat{\mathbf{U}}$ we have that

$$\begin{aligned} \hat{\mathbf{U}}_{i,j} &= \frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{U}}_{i-1,j}, \quad j = 1, 2, ..., i-1 \\ \hat{\mathbf{U}}_{i,i} &= -\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{U}_i} \end{aligned} \tag{19}$$

for $i = 1, 2, ..., N_{ms}$. Note that the matrix-matrix product $\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{P}}_{i-1}$ can be computed directly via N_p sensitivity solves. This shows that the computation of $\hat{\mathbf{P}}$ requires $N_p(2 * N_{ms} - 1)$ sensitivities. Recalling that each of the matrices $\hat{\mathbf{U}}_{i,j}$ has $2 + N_c(N_q - 1)$ columns, a similar argument proves that computation of $\hat{\mathbf{U}}$ requires $\frac{1}{2}N_{ms}(N_{ms} + 1)[2 + N_c(N_q - 1)]$. Therefore, when $N_p \ll N_x$ and $N_u \ll N_x$, the total number of sensitivity solves is substantially less than the number of sensitivities required to compute the Jacobian of Eq. (11).

The only complication of this procedure is that $\hat{\mathbf{J}}$ is not the Jacobian of $\hat{\mathbf{C}}$. As a result, the augmented Lagrangian merit function used by SNOPT cannot be used in this situation. However, Gill et. al. ([6]) have shown that the modified problem yields a descent direction for a merit function based on the ℓ_1 penalty function. This modification of the optimization algorithm is implemented in DAOPT.

In Section 2.3 we present in more detail how the sensitivity equations are formed and solved.

2.2.2 Control Continuity Constraints

Continuity conditions on u_j and u_j' $(j = 1, ..., N_u)$ at t_i, $i = 2, ..., N_{ms}$ result in the following $2(N_{ms} - 1)N_u$ linear constraints on the control parameters:

$$\begin{aligned} C^1_{j,i} &\equiv U^{1,0}_{j,i} + N_c \cdot U^{1,1}_{j,i} + \sum_{k=1}^{N_c} \sum_{q=2}^{N_q} [q(N_c - k) + 1] U^{k,q}_{j,i} - U^{1,0}_{j,i+1} \\ C^2_{j,i} &\equiv U^{1,1}_{j,i} + \sum_{k=1}^{N_c} \sum_{q=2}^{N_q} q U^{k,q}_{j,i} - \frac{\Delta t_{i+1}}{\Delta t_i} U^{1,1}_{j,i+1}, \qquad i = 1, 2, ..., N_{ms}. \end{aligned} \tag{20}$$

The Jacobian entries of these constraints are

$$\begin{aligned} &\frac{\partial C^1_{j,i}}{\partial U^{1,0}_{j,i}} = 1; \quad \frac{\partial C^1_{j,i}}{\partial U d^1_{j,i}} = N_c; \quad \frac{\partial C^1_{j,i}}{\partial U^{k,q}_{j,i}} = q(N_c - k) + 1; \quad \frac{\partial C^1_{j,i}}{\partial U^{1,0}_{j,i+1}} = -1 \\ &\frac{\partial C^2_{j,i}}{\partial U^{1,0}_{j,i}} = 0; \quad \frac{\partial C^2_{j,i}}{\partial U^{1,1}_{j,i}} = 1; \quad \frac{\partial C^2_{j,i}}{\partial U^{k,q}_{j,i}} = q; \quad \frac{\partial C^2_{j,i}}{\partial U^{1,1}_{j,i+1}} = -\frac{\Delta t_{i+1}}{\Delta t_i} \\ &i = 1, 2, ..., N_{ms} \\ &j = 1, 2, ..., N_u \\ &q = 2, 3, ..., N_q \\ &k = 1, 2, ..., N_c. \end{aligned} \tag{21}$$

Note that both SNOPT and DAOPT are such that, if the linear constraints of the optimization problem are initially feasible, all subsequent iterates satisfy the linear constraints. Since the control continuity constraints (20) are linear in the control parameters $U^{k,q}_{j,i}$, it follows that $u_j \in \mathcal{C}^1$ everywhere in $[t_1, t_{max}]$.

2.2.3 Additional Control Constraints

Because of the control parameterization scheme that we employ, user-defined bounds on controls and first derivatives of controls can be directly applied only at the beginning of each multiple shooting interval (where they can be directly converted into bounds on the control parameters $U^{1,0}_{j,i}$ and $U^{1,1}_{j,i}$). To enforce the user-defined bounds at all control points, we add $2 \cdot N_c$ linear constraints for each control, on each shooting interval i. If bl_u and bu_u are the user-defined upper and lower bounds for control j and $bl_{u'}$ and $bu_{u'}$ are the upper and lower

bounds defined for the control derivative, then we impose

$$\begin{aligned} & bl_u \le U_{j,i}^{k,0} + U_{j,i}^{k,1} + \sum_{q=2}^{N_q} U_{j,i}^{k,q} \le bu_u \\ & \Delta t_i \cdot bl_{u'} \le U_{j,i}^{k,1} + \sum_{q=2}^{N_q} q U_{j,i}^{k,q} \le \Delta t_i \cdot bu_{u'} \qquad\qquad k = 1, 2, ..., N_c. \end{aligned} \tag{22}$$

Like the control continuity constraints of Section 2.2.2, these additional linear constraints are satisfied at all optimization iterates, thus ensuring physically meaningful values of the controls at all times.

2.3 Solution of State and Sensitivity Equations

We obtain values and Jacobians of the nonlinear state continuity constraints of Section 2.2.1 by solving the following state and sensitivity equations:

$$\mathbf{F}(t, \mathbf{x}, \mathbf{x}', \mathbf{p}, \mathbf{u}(t)) = 0 \tag{23}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\mathbf{s} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\mathbf{s}' = 0 \tag{24}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\mathbf{s} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\mathbf{s}' + \frac{\partial \mathbf{F}}{\partial \mathbf{p}} = 0 \tag{25}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\mathbf{s} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\mathbf{s}' + \frac{\partial \mathbf{F}}{\partial \mathbf{U}} = 0. \tag{26}$$

A central observation behind the modified multiple shooting approach is that (24) has no forcing term. As a consequence, if $\mathbf{s}_j$ is the j-th column of the solution of (24) with initial condition $\mathbf{s}_j(t_i) = \mathbf{e}_j$ then any linear combination of these columns is also a solution of (24), with initial condition given by the coefficients of the linear combination. For a given parameter α, we say that computing $\mathbf{x}_\alpha(t)$ represents *one sensitivity computation.* Thus, computation of the modified constraint $\hat{\mathbf{c}}$ requires only one sensitivity calculation. The modified Jacobian $\hat{\mathbf{P}}$ can be obtained with only $2 \cdot N_p$ sensitivity computations, while each block $\hat{\mathbf{U}}_{i1,i2}$ in the Jacobian $\hat{\mathbf{U}}$ requires only $2+N_c(N_q-1)$ sensitivity computations. When the dimension of the state vector $\mathbf{x}$ is large, this approach substantially reduces the number of sensitivity computations required by a conventional multiple shooting approach.

On the first multiple shooting interval ($i = 1$), we have to solve $N_x + 1$ state equations and $(N_x + 1)\{1 + N_p + N_u[2 + N_c(N_q - 1)]\}$ sensitivity equations.

Table 1
State and sensitivity equations

	Dimension	Initial	Eq.	Final	Result
			$i = 1$		
1	$N_x + 1$	$\mathbf{X}_1$	23	$\mathbf{x}(t_2)$	$\mathbf{C}_1 = \mathbf{X}_2 - \mathbf{x}(t_2)$
2	$N_x + 1$	$\mathbf{0}$	24	$\mathbf{0}$	never used
3	$N_p(N_x + 1)$	$\mathbf{0}$	25	$\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{p}}$	$\hat{\mathbf{P}}_1 = -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{p}}$
4	(†)	$\mathbf{0}$	26	$\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{U}_1}$	$\hat{\mathbf{U}}_{1,1} = -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{U}_1}$
			$i > 1$		
1	$N_x + 1$	$\mathbf{X}_i$	23	$\mathbf{x}(t_{i+1})$	$\mathbf{C}_i = \mathbf{X}_{i+1} - \mathbf{x}(t_{i+1})$
2	$N_x + 1$	$\hat{\mathbf{C}}_{i-1}$	24	$\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{C}}_{i-1}$	$\hat{\mathbf{C}}_i = \mathbf{C}_i + \frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{C}}_{i-1}$
3	$N_p(N_x + 1)$	$\mathbf{0}$	25	$\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{p}}$	
4	(†)	$\mathbf{0}$	26	$\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{U}_i}$	$\hat{\mathbf{U}}_{i,i} = -\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{U}_i}$
5	$N_p(N_x + 1)$	$-\hat{\mathbf{P}}_{i-1}$	24	$-\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{P}}_{i-1}$	$\hat{\mathbf{P}}_i = -\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{p}} + \frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{P}}_{i-1}$
6	(‡)	$-\hat{\mathbf{U}}_{i-1,1}$ $-\hat{\mathbf{U}}_{i-1,2}$ $\vdots$ $-\hat{\mathbf{U}}_{i-1,i-1}$	24	$-\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{U}}_{i-1,1}$ $-\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{U}}_{i-1,2}$ $\vdots$ $-\frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{U}}_{i-1,i-1}$	$\hat{\mathbf{U}}_{i,1} = \frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{U}}_{i-1,1}$ $\hat{\mathbf{U}}_{i,2} = \frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{U}}_{i-1,2}$ $\vdots$ $\hat{\mathbf{U}}_{i,i-1} = \frac{\partial \mathbf{x}(t_{i+1})}{\partial \mathbf{X}_i}\hat{\mathbf{U}}_{i-1,i-1}$

(†) $N_u(2 + N_c(N_q - 1))(N_x + 1)$
(‡) $(i-1)N_u(2 + N_c(N_q - 1))(N_x + 1)$

On each of the following multiple shooting intervals ($i > 1$) the number of equations to be solved is $(N_x + 1)\{2 + 2N_p + iN_u[2 + N_c(N_q - 1)]\}$. Collecting the states and all their sensitivities into a vector $\mathbf{v}$, we can split this vector into 6 parts. Table 1 lists the state and sensitivity equations that are solved on each multiple shooting interval. The additional variable corresponding to the cost function (3) is inserted in the vector of model states at a position specified by the user. This option is provided to preserve the possible band structure of the model equations' Jacobian. If the *banded Jacobian* option is not used, the additional state can be positioned anywhere between states 1 and N_x.

If no explicit bounds are defined for the model states at $t = t_{N_{ms}+1} \equiv t_{max}$ and if they are not part of the cost function or of the user constraints, it is more efficient not to include them among the optimization variables. This not only leads to an optimization problem with fewer variables, but also results in N_x

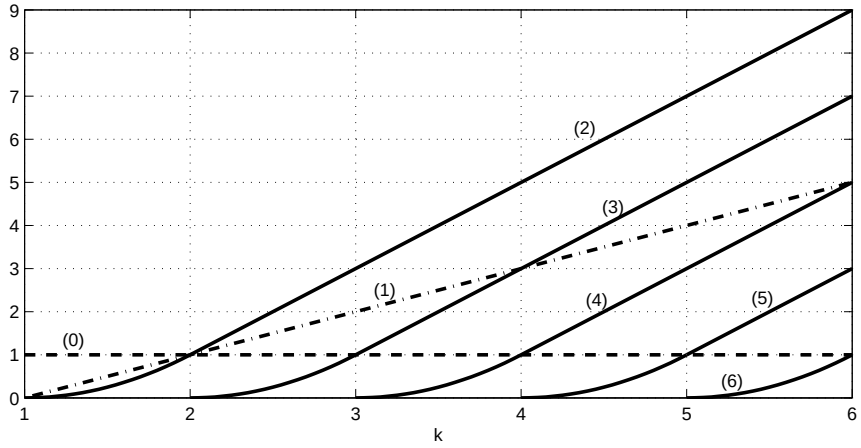

Fig. 2. Sensitivities of the control with respect to coefficients of the parameterization (for $N_c = 5$, $N_q = 2$). (0): $\partial u/\partial U^{1,0}$ (1): $\partial u/\partial U^{1,1}$ (2): $\partial u/\partial U^{1,2}$ (3): $\partial u/\partial U^{2,2}$ (4): $\partial u/\partial U^{3,2}$ (5): $\partial u/\partial U^{4,2}$ (6): $\partial u/\partial U^{5,2}$

fewer nonlinear state continuity constraints. However, even in this case, the additional variable corresponding to the cost function at t_{max} is still added to the optimization variables, to obtain a linear cost function in the discretized nonlinear programming problem (4).

Although we consider additional control subintervals inside each multiple shooting interval, the control parameterization employed (see Section 2.1) assures $\mathcal{C}^1$ continuity of the controls inside each shooting interval. Thus the integration can be carried out without restarts. This can be seen best in Fig. 2 which shows that sensitivities of the control u with respect to the coefficients of the parameterization are also $\mathcal{C}^1$ continuous. Moreover, the optimization algorithm implemented in SNOPT/DAOPT satisfies the linear constraints at all iterates. The controls and their derivatives are therefore within the prescribed bounds at any control subdivision.

DASPK3.0 ([8]) is a software package for solving DAE initial value problems. It uses variable-order variable-step backward differentiation formulas. The linear systems that arise at each time step are solved with either dense or banded direct linear system methods, or with preconditioned Krylov iterative methods. In addition to solving the DAE (23), DASPK3.0 implements several highly efficient algorithms for performing sensitivity analysis. During integration on a given shooting interval i, DASPK3.0 requires computation of the residual of the state and sensitivity equations (23)-(26), as well as computation of the Jacobian with respect to the states and their derivatives. The subroutine `cNsys` calls the user-defined routines `cUsysM` and `cUsysC` to evaluate the residual of the state equations and the right side of the additional equation (3), respectively, at a given time t, for given values of the parameters $\mathbf{p}$, states $\mathbf{x}$, state

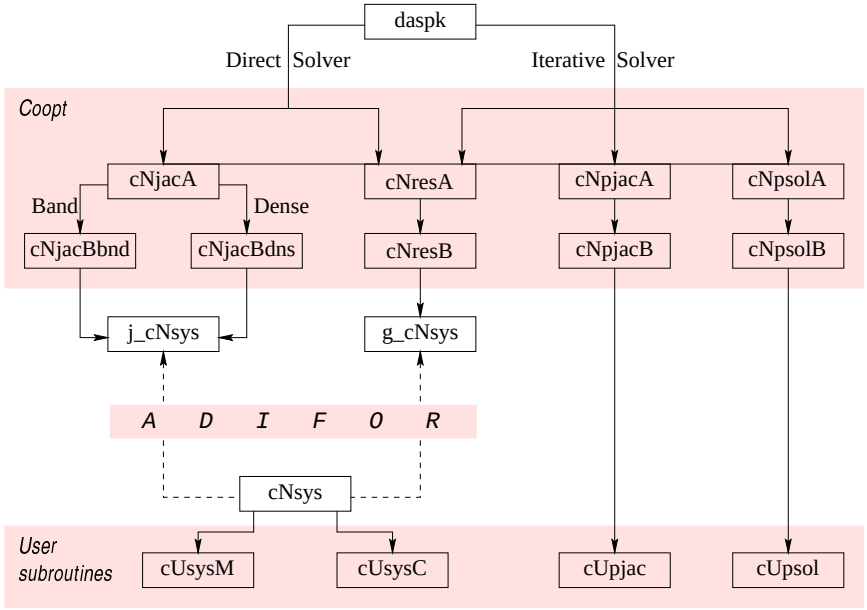

Fig. 3. Subroutines called by DASPK3.0

derivatives $\mathbf{x}'$, and controls $\mathbf{u}$; i.e.,

$$\mathbf{F} = \mathbf{F}(t, \mathbf{x}, \mathbf{x}', \mathbf{p}, \mathbf{u}). \tag{27}$$

The subroutine `cNsys` is preprocessed through ADIFOR to generate the following two subroutines (see Fig. 3): (a) `g_cNsys` to evaluate

$$\bar{\mathbf{F}} = \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\bar{\mathbf{x}} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\bar{\mathbf{x}}' + \frac{\partial \mathbf{F}}{\partial \mathbf{p}}\bar{\mathbf{p}} + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}\bar{\mathbf{u}} \tag{28}$$

for given values of the seed vectors $\bar{\mathbf{x}}$, $\bar{\mathbf{x}}'$, $\bar{\mathbf{p}}$, and $\bar{\mathbf{u}}$; and (b) `j_cNsys` to evaluate

$$\bar{\mathbf{J}} = \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\bar{\mathbf{X}} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\bar{\mathbf{X}}' \tag{29}$$

for given values of the seed matrices $\bar{\mathbf{X}}$, $\bar{\mathbf{X}}'$.

2.3.1 Residual Computation

The equations that must be solved on a given shooting interval i can be partitioned into six parts, corresponding to the similar partition of the vector $\mathbf{v}$. We evaluate each part of the residual by repeatedly calling `g_cNsys` with convenient seed vectors. As an example, consider the fourth part of the residual (see Table 1). The subroutine `g_cNsys` must be called $N_u(2 + N_c(N_q - 1))$ times to evaluate the residual (26) for each control parameter in the interval i. Comparing (26) and (28) it follows that $\bar{\mathbf{p}} = \mathbf{0}$. The seed vector $\bar{\mathbf{x}}$ is set to that part of the vector $\mathbf{v}$ that contains the sensitivities with respect to the current control parameter. The seed vector $\bar{\mathbf{x}}'$ is set to the corresponding part

in $\mathbf{v}'$. For each control $j = 1, \cdots, N_u$, the initialization of the seed vector $\bar{\mathbf{u}}$ follows from

$$u_j(t) = \left[U_{j,i}^{1,0} + k \cdot U_{j,i}^{1,1} + \sum_{k1=1}^{k} \sum_{q=2}^{N_q} [q(k-k1)+1] U_{j,i}^{k1,q} \right] + \left[U_{j,i}^{1,1} + \sum_{k1=1}^{k} \sum_{q=2}^{N_q} q U_{j,i}^{k1,q} \right] t^* + \sum_{q=2}^{N_q} U_{j,i}^{k+1,q} t^{*q},$$

where we have used (6) and (8).

Therefore, to compute the residual of the sensitivity equations with respect to the control parameter $U_{j,i}^{1,0}$ the only nonzero component of $\bar{\mathbf{u}}$ is $\bar{u}_j = 1$. To compute the residual of the sensitivity equations with respect to $U_{j,i}^{1,1}$, we set $\bar{u}_j = k + t^*$. To compute the residual of the sensitivity equations with respect to $U_{j,i}^{k1,q}$, $k1 = 1, ..., k$, $q = 2, ..., N_q$, we set $\bar{u}_j = q(k-k1) + 1 + qt^*$. Finally, to compute the residual of the sensitivity equations with respect to $U_{j,i}^{k+1,q}$, $q = 2, ..., N_q$, we set $\bar{u}_j = t^{*q}$.

2.3.2 Jacobian Computation

If a direct linear system solution method is selected, DASPK3.0 also requires Jacobians of (23) with respect to $\mathbf{x}$ and $\mathbf{x}'$. Depending on the *banded Jacobian* user option, the ADIFOR-generated subroutine `j_cNsys` is called with different arguments to obtain either a dense or a banded Jacobian. If the Jacobian is not banded, subroutine `j_cNsys` is called with the following seed matrices:

$$\bar{\mathbf{X}} = \mathbf{I}_{N_x+1}, \qquad \bar{\mathbf{X}}' = \frac{1}{h} \cdot \mathbf{I}_{N_x+1}, \tag{30}$$

while if the Jacobian is banded, `j_cNsys` is called with the following seed matrices:

$$\bar{\mathbf{X}} = \begin{bmatrix} \mathbf{I}_w \\ \mathbf{0} \end{bmatrix}, \qquad \bar{\mathbf{X}}' = \frac{1}{h} \cdot \begin{bmatrix} \mathbf{I}_w \\ \mathbf{0} \end{bmatrix}, \tag{31}$$

where w is the Jacobian bandwidth and h is the current integration step size.

2.3.3 Integration Tolerances

The accuracy of the states and sensitivities computed by DASPK3.0 is specified by the error tolerances *rtol* and *atol*. There are three methods of specifying these values in COOPT.

The simplest use is to take both *rtol* and *atol* to be scalars. These user defined values will then be used for all variables, both state and sensitivity.

A second option is to solve the state and sensitivity equations with different tolerances. The values $rtol_x$ and $atol_x$ are used for all state variables and $rtol_s$ and $atol_s$ are used for all sensitivity variables.

As a third option, the user can provide a subroutine which specifies *rtol* and *atol* values for each of the state variables and for the cost function variable. In addition, order of magnitude information for the states, parameters, and controls must be provided. Error tolerances for the sensitivity variables are computed as follows:

- For all sensitivity variables, the relative tolerance *rtol* is set to the *rtol* tolerance of the corresponding state variable.
- For sensitivities with respect to parameters $\mathbf{p}$ the absolute tolerance *atol* is computed by dividing the *atol* tolerance of the corresponding state variable by the estimate provided for that parameter.
- For variables that represent linear combinations of sensitivities with respect to initial conditions (see Table 1), the absolute error tolerance *atol* is computed as a weighted sum of corresponding *atol* tolerances for the state variables divided by estimates of the states. The weights are the coefficients of the linear combinations. Let $[a_1, a_2, ..., a_{N_x}]$ be the absolute tolerances for the state variables $\mathbf{x} \in R^{N_x}$ and consider the array $\mathbf{s} = \frac{\partial \mathbf{x}}{\partial \mathbf{x_0}}\xi$, where $\xi \in R^{N_x}$. Then the absolute tolerances used in computing $\mathbf{s}$ is set to $[a_1, a_2, ..., a_{N_x}] \sum_{i=1}^{N_x} |\xi_i/\bar{x}_i|$, where $\bar{x}_i$ is an estimate for x_i.

2.4 Sensitivity of the Optimal Solution

In many optimal control problems, obtaining an optimal solution is not the only goal. The influence of problem parameters on the optimal solution (the so called sensitivity of the optimal solution) is also of interest. Sensitivity information provides a first-order approximation of the behavior of the optimal solution when parameters are not at their optimal values or when constraints are slightly violated. In COOPT, we make use of the Sensitivity Theorem (see [2]) for nonlinear programming problems with equality and/or inequality constraints:

Theorem 2.1. *Let f, h, and g be twice continuously differentiable and consider the family of problems*

$$\begin{aligned} &\textit{minimize } f(x) \\ &\textit{subject to } h(x) = u, \quad g(x) \leq v, \end{aligned} \tag{32}$$

parameterized by the vectors $u \in R^m$ and $v \in R^r$. Assume that for $(u, v) =$

$(0,0)$ this problem has a local minimum x^*, which is regular and which together with its associated Lagrange multiplier vectors λ^* and μ^*, satisfies the second order sufficiency conditions. Then there exists an open sphere S centered at $(u,v) = (0,0)$ such that for every $(u,v) \in S$ there is an $x(u,v) \in R^n$ and $\lambda(u,v) \in R^m$, $\mu(u,v) \in R^r$, which are a local minimum and associated Lagrange multiplier vectors of problem (32). Furthermore, $x(\cdot)$, $\lambda(\cdot)$, and $\mu(\cdot)$ are continuously differentiable in S and we have $x(0,0) = x^*, \lambda(0,0) = \lambda^*, \mu(0,0) = \mu^*$. In addition, for all $(u,v) \in S$, there holds

$$\nabla_u p(u,v) = -\lambda(u,v),$$
$$\nabla_v p(u,v) = -\mu(u,v),$$

where $p(u,v)$ is the optimal cost parameterized by (u,v),

$$p(u,v) = f(x(u,v)).$$

The desired sensitivities are thus directly obtained from the Lagrange multipliers computed during the solution of the optimization problem. Although the above method gives only sensitivities of the optimal solution with respect to constraint perturbation, other sensitivities can also be obtained with minor changes in the problem set-up. Consider the case where sensitivity of the optimal solution with respect to some problem parameter is desired. The solution is to add that parameter among the optimization variables and add an equality constraint to keep the parameter to its nominal value. At the optimum solution, the Lagrange multiplier associated with this additional constraint yields the desired sensitivity.

3 A Simple Example

To run a problem in COOPT the user must supply subroutines that

- specify flags for the optimizer and the integrator;
- specify the initial guess and the bounds on the parameters **p** and the controls **u**;
- evaluate the additional constraints (5b) and their Jacobians;
- evaluate the residual of the model equations (2a);
- evaluate the integrand of the additional differential equation (3).

To illustrate the use of COOPT consider the 1-D heat equation

$$\frac{\partial x}{\partial t} = \frac{\partial^2 x}{\partial y^2} \tag{33}$$

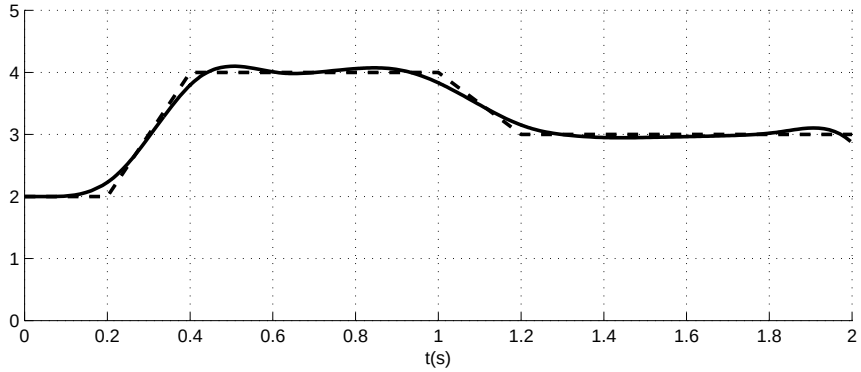

Fig. 4. Optimization results for the heat problem. Solid line $x_6^*(t)$. Dashed line $\tau(t)$

defined on the region $t \geq 0$, $0 \leq y \leq 1$ with boundary conditions at $x(t,0)$ and $x(t,1)$, and initial conditions at $x(0,0)$.

We discretize the spatial derivative via the method of lines (MOL) using finite differences and convert the PDE into an index-1 DAE. Taking a uniform spatial grid $y_j = (j+1)\Delta y$, $1 \leq j \leq N$, and using centered differences, we obtain the DAE in the variables $x_j(t) = x(t, y_j)$

$$\begin{aligned} & x_1 - x(t,0) = 0 \\ & x_j' - \frac{x_{j-1} - 2x_j + x_{j+1}}{(\Delta y)^2} = 0, \qquad j = 2, ..., N-1 \\ & x_N - x(t,1) = 0 \end{aligned} \tag{34}$$

Imposing $x(t,0) = x(t,1) = u$ as the control, we can formulate an optimal control problem to find u such that, for some k, $2 \leq k \leq N-1$, the temperature x_k follows a predefined path $\tau(t)$.

Consider $N = 11$, $k = 6$, and $\tau(t)$ defined as in Fig. 4. A user defined subroutine (`cUinit`) is used to specify problem specific dimensions and flags. The significant values for this problem are

$N_{xd} = 9$	The number of differential variables is $N - 2$
$N_{xa} = 2$	There are 2 algebraic variables
$N_u = 1$	There is one control function
$N_q = 3$	Cubic approximation of the control
$N_{ms} = 10$	There are 10 shooting intervals
$N_c = 1$	There is 1 control interval per shooting interval

Additional flags specify that states at t_{max} are not part of the optimization variables and that the additional differential equation (3) is inserted in the

last position in the DAE system.

Indices of the two algebraic variables are set through the user subroutine `cUvars` as 1 and 11. Integration tolerances are defined in the subroutine `cUinit` as $rtol_x = 10^{-6}$, $atol_x = 10^{-5}$, $rtol_s = 10^{-4}$, and $atol_s = 10^{-3}$.

As initial guess (user subroutine `cUguess`) we set $x_j = 2.0$, $j = 1, ..., N$ at all shooting intervals and $u = 2.0$ everywhere. We impose zero lower bounds for both states and controls and we fix the control at $t = 0$ by specifying $blU0(1) = buU0(1) = 2.0$ (user subroutine `cUbounds`).

The state equations are specified through the user subroutine `sysM`:

```
dx = 1.0d0/(Nx-1)
dx2 = dx*dx
f(1) = X(1) - U(1)
do ix = 2,Nx-1
   f(ix) = Xdot(ix) - (X(ix-1)-2.0d0*X(ix)+X(ix+1))/dx2
enddo
f(Nx) = X(Nx) - U(1)
```

The right side of the additional equation (3) is specified in the user subroutine `sysC`:

```
tt = 2.0d0
ww = 1.0d0
if(t.GT.0.2 .AND. t.LT.0.4) then
   tt = 2.0d0+2.0d0*(t-0.2)/0.2
endif
if(t.GE.0.4 .AND. t.LE.1.0) then
   tt = 4.0d0
endif
if(t.GT.1.0 .AND. t.LT.1.2) then
   tt = 4.0d0-1.0d0*(t-1.0)/0.2
endif
if(t.GE.1.2) then
   tt = 3.0d0
endif
fc = ww*(X(6)-tt)*(X(6)-tt)
```

Optimization settings are passed to the optimizer through an SNOPT specification file. For this particular problem we have used a feasibility tolerance of 10^{-5} and an optimality tolerance of $5 \cdot 10^{-3}$. With the above settings, COOPT converged to the optimal solution in 12 major iterations. The cost function corresponding to the initial guess was 3.928 and was reduced to $9.600 \cdot 10^{-3}$. Figure 4 shows the temperature x_6^* obtained with the optimal control as com-

pared to the desired function $\tau(t)$.

4 Conclusions

In this paper we have presented the most important features of COOPT. The software package includes more options than the ones described here, such as options to include initial conditions among the optimization variables, objective functions dependent on the final state, and the possibility to optimize over the final time. A complete description of all the options supported by COOPT is given in [12].

COOPT has been used successfully in a variety of applications with very different problem specifications and requirements. In a first application ([11]), the algorithm was demonstrated for a stagnation-flow type reactor configuration. The equations describing chemically reacting stagnation flows were written in a transient compressible similarity form. After discretizing the spatial derivatives on a finite-volume mesh, the system becomes a set of DAEs. Dynamic optimization was then used to control the film stoichiometry during imposed transients in the chemical vapor deposition of multicomponent superconducting thin films.

In [13], COOPT has been applied to the problem of trajectory design and orbit insertion in the circular restricted three-body problem. The algorithm was employed to design a continous trajectory from the Earth-Moon system to the periodic halo orbit around the libration point L_1 of the Sun-Earth system. Optimal values for maneuver times and magnitudes were computed to correct for possible errors in injection velocity, while minimizing fuel consumption.

As a third application of COOPT we cite the work of Xu *et.al.* ([14]) where COOPT was used to identify time dependent parameters in models of confined compression of collagen for tissue engineering. By defining the objective function to be the sum of the squared error between the model prediction and experimental result, COOPT was used to regress time varying parameters in the model.

References

[1] U. M. Ascher, R. M. M. Mattheij, and R. D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Society for Industrial and Applied Mathematics (SIAM) Publications, Philadelphia, PA, 1995. ISBN 0-89871-354-4.

[2] D.R. Bertsekas, *Nonlinear Programming*, Athena Scientific, Belmont, Ma, 1995

[3] J.T. Betts, *Survey of Numerical Methods for Trajectory Optimization*, Journal of Guidance Control and Dynamics, Vol. 21(2), pp. 193-207, 1998.

[4] C. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland, *ADIFOR—generating derivative codes from Fortran programs*, Scientific Programming, 1 (1992), pp. 11–29.

[5] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, SIAM Publications, Philadelphia, second ed., 1995. ISBN 0-89871-353-6.

[6] P. E. Gill, L. O. Jay, M. W. Leonard, L. R. Petzold and V. Sharma, *An SQP Method for the Optimal Control of Large-Scale Dynamical Systems*, J. Comp. Appl. Math, to appear.

[7] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization*, Numerical Analysis Report 97-2, Department of Mathematics, University of California, San Diego, La Jolla, CA, 1997.

[8] S. Li and L.R. Petzold, *Design of New DASPK for Sensitivity Analysis*, UCSB Department of Computer Science Technical Report, 1999.

[9] T. Maly and L. R. Petzold, *Numerical methods and software for sensitivity analysis of differential-algebraic systems*, Applied Numerical Mathematics 20 (1996), pp. 57–79.

[10] L.R. Petzold, J. B. Rosen, P. E. Gill, L. O. Jay and K. Park, *Numerical Optimal Control of Parabolic PDEs using DASOPT*, Large Scale Optimization with Applications, Part II: Optimal Design and Control, Eds. L. Biegler, T. Coleman, A. Conn and F. Santosa, IMA Volumes in Mathematics and its Applications, Vol. 93, (1997), pp. 271-300.

[11] L. Raja, R. Kee, R. Serban, and L.R. Petzold, *Dynamic Optimization of Chemically Reacting Stagnation Flows*, 1998 Electrochemical Society Conference, Boston, Ma.

[12] R. Serban, *COOPT - Control and Optimization of Dynamic Systems - Users' Guide*, UCSB, Department of Mechanical and Environmental Engineering, Report UCSB-ME-99-1,1999.

[13] R. Serban, W.S. Koon, M.W. Lo, J.E. Marsden, L.R. Petzold, S.D. Ross, and R.S. Wilson, *Halo Orbit Mission Correction Maneuvers Using Optimal Control*, submitted to Automatica, 1999.

[14] J. Xu, J.J. Heys, V.H. Barocas, and T.W. Randolph, *Permeability and Diffusion in Vitreous Humor: Implications for Drug Delivery*, submitted to Pharmaceutical Research, 1999.