COOPT - Control and Optimization of Dynamic Systems Users' Guide

Radu Serban
Computational Science and Engineering Group
University of California, Santa Barbara

July 30, 2002

Report UCSB-ME-99-1

This research was partially supported by NSF/DARPA grant DMS-9615858 and NSF grant CCR-9896198 and by Department of Energy grant DE-FG03-98ER25354.

The latest version of this document is available at
`www.engineering.ucsb.edu/~cse`
`www.engineering.ucsb.edu/~radu/projects.html`

Contents

List of Figures

List of Tables

1 Preliminaries

1.1 Method Description

We consider the differential-algebraic equation (DAE) system

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

where the DAE is index one (see [5] or [1]) 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 objective function

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

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 our application the DAE system is large-scale. Thus, the dimension N_x of $\mathbf{x}$ is large. However, the dimension of the control parameters and of the representation of the control function $\mathbf{u}(t)$ is 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 we consider

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

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

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

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. 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 [2]. 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 $[t_1, t_{\max}]$ with the ODE solutions at each discrete time and the set of controls 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 $[t_{itx}, t_{itx+1}]$ $(itx = 1, \ldots, N_{tx})$, and the differential equations (2a) are solved over each subinterval, where additional intermediate variables $\mathbf{X}_{itx}$ are introduced. On each subinterval we denote the solution at time t of (2a) with initial value $\mathbf{X}_{itx}$ at t_{itx} by $\mathbf{x}(t, t_{itx}, \mathbf{X}_{itx}, \mathbf{p})$.

Continuity between subintervals is achieved via the continuity constraints

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

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 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 such as the problem considered in this paper, 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

$$\mathbf{C}_2^{itx}(\mathbf{X}_{itx}, \mathbf{p}) \equiv \mathbf{g}(t_{itx}, \mathbf{X}_{itx}, \mathbf{p}) \geq \mathbf{0}.$$

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

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

subject to the constraints

$$\mathbf{C}_1^{itx}(\mathbf{X}_{itx+1}, \mathbf{X}_{itx}, \mathbf{p}) = \mathbf{0}, \tag{5a}$$

$$\mathbf{C}_2^{itx}(\mathbf{X}_{itx}, \mathbf{p}) \geq \mathbf{0}. \tag{5b}$$

This problem can be solved by an optimization code. We use the solver SNOPT [7], which incorporates a sequential quadratic programming (SQP) method (see [9]). 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 differential-algebraic equation (DAE) sensitivity software DASPK3.0 [12]. The sensitivity equations to be solved by DASPK3.0 are generated via the automatic differentiation software ADIFOR [3]. Our basic algorithms and software for the optimal control of dynamical systems are described in detail in [11].

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 with respect to the variables $\mathbf{X}_{itx}$. The work to compute the derivative matrix $\partial \mathbf{x}(t)/\partial \mathbf{X}_{itx}$ is of order $\mathcal{O}(N_x^2)$, 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 $\mathcal{O}(N_x N_p)$ 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 $(\partial \mathbf{x}(t)/\partial \mathbf{X}_{itx})\mathbf{w}_j$ are needed, rather than the entire matrix $\partial \mathbf{x}(t)/\partial \mathbf{X}_{itx}$. 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 $\mathcal{O}(N_p)$. Thus the complexity of the modified multiple shooting computation is reduced to $\mathcal{O}(N_x N_p)$, 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 dissipative PDE systems including the application described here, this is not an issue, and the modified method is more robust for nonlinear problems.

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

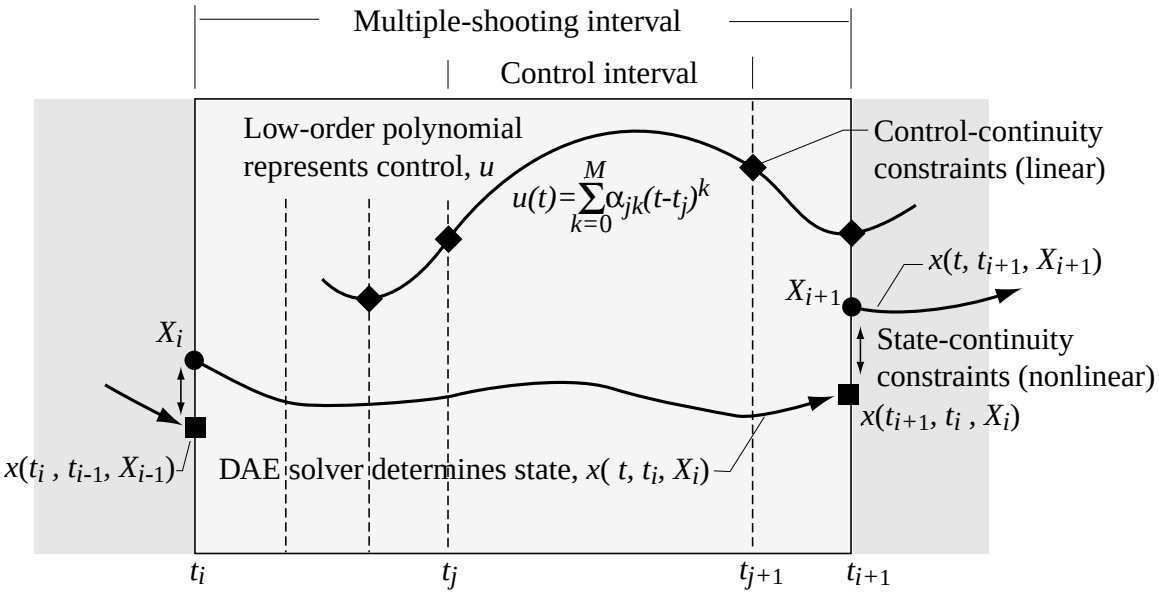

Figure 1: Description of the multiple shooting method

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.

The main features of the multiple shooting method are shown in Fig. 1. Details of the modified multiple shooting method are given in [6].

1.2 Installation

The file `coopt.tar` unpacks into a directory named `coopt`. This directory contains:

- `Makefile`: Top level makefile.
- `Main.f`: The main program. Sets integer and real work spaces and calls the driver subroutine.
- `src`: Contains the source files for COOPT.
- `templates`: Contains templates for user files.
- `daspk`: Contains the source for the DASPK3.0 library.
- `snopt`: Contains the source for the SNOPT library.
- `daopt`: Contains the source for the DAOPT library.
- `libs`: Contains the DASPK3.0, SNOPT, and DAOPT libraries.
- `example`: Contains user files for the heat example (Section 4)

To build the DASPK3.0 library use the following command in the directory `coopt/daspk`:

```
% make lib
```

This will compile the DASPK3.0 source files, create the library `libdaspk.a`, and move it to `coopt/libs`. To build the SNOPT library use the following command in the directory `coopt/snopt`:

```
% make lib
```

This will compile the SNOPT source files, create the library `libsnopt.a`, and move it to `coopt/libs`. To build the DAOPT library use the following command in the directory `coopt/daopt`:

```
% make lib
```

This will compile the DAOPT source files, create the library `libdaopt.a`, and move it to `coopt/libs`.

2 Setting Up a New Problem

In this section we discuss the main steps that you have to take in order to create and run a new problem with COOPT. For more information on SNOPT, DASPK3.0, and ADIFOR consult the following references:

- *User's Guide for SNOPT 5.3: A Fortran Package for Large-Scale Nonlinear Programming*, P.E. Gill, W. Murray, and A. Saunders
- *Design of New DASPK for Sensitivity Analysis*, S. Li and L.R. Petzold
- *ADIFOR 2.0 Users' Guide - Revision D*, C. Bischof, A. Carle, P. Hovland, P. Khademi, and A. Mauer

For details on code structure and implementation see Section 3.

2.1 User Subroutines

To generate a new problem, first create a subdirectory in `coopt` that will contain all problem specific files. You are free to choose any name for the user subdirectory, as long as it does not conflict with existing directories in `coopt`. In the following discussions, we assume that your files will be placed in the directory `coopt/myproblem`. Templates of all user files are provided in `coopt/templates`. The file names, as well as the subroutine names are fixed. To prevent possible conflicts with user-defined names, subroutine names in COOPT start with `cU` (user defined subroutine) and with `cN` (COOPT internal subroutines).

The user Fortran files and subroutines are listed in Table 1. In addition to these Fortran files, you must also create an ADIFOR script file and two ADIFOR composition files.

2.1.1 File `Initial.f`

The file `Initial.f` contains the following subroutines:

1. **Name** `cUnames`
   Purpose Specifies the SNOPT/DAOPT specification file, output directory, and problem name.
   Template

```
SUBROUTINE cUnames(SpecName,OutDir,ProbName)
IMPLICIT NONE
CHARACTER*256 SpecName, OutDir
CHARACTER*8 ProbName

RETURN
END
```

   Arguments
   `SpecName` - Name of the optimizer specification file
   `OutDir` - Name of the output directory
   `ProbName` - Problem name
   Note
   These names are used to generate output file names (see Section 2.3)

2. **Name** `cUspace`
   Purpose Sets integer and real user array dimensions.
   Template

```
SUBROUTINE cUspace(leniu,lenru)
IMPLICIT NONE
INTEGER leniu, lenru

RETURN
END
```

Arguments
`leniu` - Length of integer user array
`lenru` - Length of real user array

3. **Name** `cUinit`
Purpose Sets problem dimensions and flags and integer and real problem data.
Template

```
      SUBROUTINE cUinit(idim, iflag,
     &                  idata, rdata,
     &                  iuser, leniu, ruser, lenru)
      IMPLICIT NONE
      INTEGER idim(20),iflag(30), idata(15), leniu,iuser(leniu),lenru
      DOUBLE PRECISION rdata(50),ruser(lenru)

      RETURN
      END
```

Arguments
`idim` - Array of problem dimensions
`iflag` - Array of solution flags
`idata` - Array of integer problem data
`rdata` - Array of real problem data
`iuser` - Integer user array
`leniu` - Dimension of integer user array
`ruser` - Real user array
`lenru` - Dimension of real user array
Note
For a desription of the `idim`, `iflag`, and `idata` arrays, see Tables 2, 3, and 4.

4. **Name** `cUvars`
Purpose Specifies the algebraic variables and equations.
Template

```
      SUBROUTINE cUvars(Nxa, ivars, ieqs, iuser, leniu, ruser, lenru)
      IMPLICIT NONE
      INTEGER Nxa, ivars(Nxa), ieqs(Nxa), leniu, iuser(leniu),lenru
      DOUBLE PRECISION ruser(lenru)

      RETURN
      END
```

Arguments
`Nxa` - Number of algebraic variables
`ivars` - Indices of the algebraic variables
`ieqs` - Indices of the index-2 algebraic equations
`iuser` - Integer user array
`leniu` - Dimension of integer user array
`ruser` - Real user array
`lenru` - Dimension of real user array
Note
The information about algebraic variables and equations is required only for consistent initial condition computations at the beginning of the multiple shooting intervals. The array `ivars` needs to be set only if the model equations represent a DAE of index 1 or 2. The array `ieqs` needs to be set only if the model equations represent a DAE of index 2.

5. **Name** `cUtime`
Purpose Returns times at the multiple shooting points.
Template

```
      SUBROUTINE cUtime(Ntx, tx,
     &   iuser, leniu, ruser, lenru)
      IMPLICIT NONE
      INTEGER Ntx, leniu, iuser(leniu), lenru
      DOUBLE PRECISION tx(Ntx+1), ruser(lenru)

      RETURN
      END
```

Arguments
`Ntx` - Number of multiple shooting intervals
`tx` - Time at multiple shooting points
`iuser` - Integer user array
`leniu` - Dimension of integer user array
`ruser` - Real user array
`lenru` - Dimension of real user array

6. **Name** `cUguess`
Purpose Sets initial guesses for parameters, 'initial parameters', states, and controls.
Template

```
      SUBROUTINE cUguess(Np, Nr, Nx, Nu,
     &   guessP, guessR, guessX, guessU,
     &   iuser, leniu, ruser, lenru)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z), INTEGER (I-N)
      INTEGER Np, Nr, Nx, Nu, leniu, iuser(leniu), lenru
      DOUBLE PRECISION guessP(*), guessR(*), guessX(*), guessU(*)
      DOUBLE PRECISION ruser(lenru)

      RETURN
      END
```

`Arguments`
`Np` - Number of parameters
`Nr` - Number of 'initial' parameters
`Nx` - Number of model states
`Nu` - Dimension of control
`guessP` - Initial guess for parameters
`guessR` - Initial guess for 'initial' parameters
`guessX` - Initial guess for states
`guessU` - Initial guess for controls
`iuser` - Integer user array
`leniu` - Dimension of integer user array
`ruser` - Real user array
`lenru` - Dimension of real user array

7. **Name** `cUbounds`
Purpose Returns lower and upper bounds for parameters, 'initial' parameters, controls, control derivatives, initial controls, states, and final states.
Template

```
      SUBROUTINE cUbounds(Np, Nr, Nx, Nu,
     $   blP, buP, blR, buR,
     $   blXf, buXf, blX, buX,
     &   blU0, buU0, blUf, buUf, blU, buU,
     &   blU0dot, buU0dot, blUdot, buUdot,
     &   iuser, leniu, ruser, lenru)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z), INTEGER (I-N)
      INTEGER Np, Nr, Nx, Nu, leniu, lenru, iuser(leniu)
      DOUBLE PRECISION ruser(lenru)
      DOUBLE PRECISION blP(Np),    buP(Np)
      DOUBLE PRECISION blR(Nr),    buR(Nr)
      DOUBLE PRECISION blU0(Nu),   buU0(Nu)
      DOUBLE PRECISION blUf(Nu),   buUf(Nu)
      DOUBLE PRECISION blU(Nu),    buU(Nu)
      DOUBLE PRECISION blU0dot(Nu),buU0dot(Nu)
      DOUBLE PRECISION blUdot(Nu), buUdot(Nu)
      DOUBLE PRECISION blX(Nx),    buX(Nx)
      DOUBLE PRECISION blXf(Nx),   buXf(Nx)

      RETURN
      END
```

Arguments
`Np` - Number of parameters
`Nr` - Number of 'initial' parameters
`Nx` - Number of model states
`Nu` - Dimension of control
`blP` - Lower bounds for parameters
`buP` - Upper bounds for parameters
`blR` - Lower bounds for 'initial' parameters
`buR` - Upper bounds for 'initial' parameters
`blU0` - Lower bounds for controls at initial time
`buU0` - Upper bounds for controls at initial time
`blUf` - Lower bounds for controls at final time
`buUf` - Upper bounds for controls at final time
`blU` - Lower bounds for controls
`buU` - Upper bounds for controls
`blU0dot` - Lower bounds for control time derivatives at initial time
`buU0dot` - Upper bounds for control time derivatives at initial time
`blUdot` - Lower bounds for control time derivatives
`buUdot` - Upper bounds for control time derivatives
`blX` - Lower bounds for states
`buX` - Upper bounds for states
`blXf` - Lower bounds for final states
`buXf` - Upper bounds for final states
`iuser` - Integer user array
`leniu` - Dimension of integer user array
`ruser` - Real user array
`lenru` - Dimension of real user array
Note
A lower bound of -10^{20} and an upper bound of 10^{20} are automatically assigned to all variables for which bounds are not defined by the user.

8. **Name** `cUtol`
Purpose Specifies tolerances for state variables and order of magnitude information for states, param-

eters, and controls.
Template

```
      SUBROUTINE cUtol(Nx,Np,Nr,Nu,
     &   estX,estCF,estP,estR,estU,
     &   rtolX,rtolCF,atolX,atolCF)
      IMPLICIT NONE
      INTEGER Nx, Np, Nr, Nu
      DOUBLE PRECISION estX(Nx), estCF
      DOUBLE PRECISION estP(Np), estR(Nr), estU(Nu)
      DOUBLE PRECISION rtolX(Nx), rtolCF
      DOUBLE PRECISION atolX(Nx), atolCF


      RETURN
      END
```

Arguments
`Nx` - Number of model states
`Np` - Number of parameters
`Nr` - Number of 'initial' parameters
`Nu` - Number of controls
`estX` - Estimates of the order of magnitude of the model states
`estCF` - Estimate of the order of magnitude of the cost function
`estP` - Estimates of the order of magnitude of the parameters **p**
`estR` - Estimates of the order of magnitude of the 'initial' parameters **r**
`estU` - Estimates of the order of magnitude of the controls
`rtolX` - Array of relative error tolerances for the model state variables
`rtolCF` - Relative error tolerance for the additional variable associated with the cost function
`atolX` - Array of absolute error tolerances for the model state variables
`atolCF` - Absolute error tolerance for the additional variable associated with the cost function
Note
This subroutine is called only if $iflag(5) = 2$.
All entries in `estX, estP, estR, and estU`, as well as `estCF` must be positive non-zero real numbers.

2.1.2 File `Constraints.f`

The file `Constraints.f` contains the following subroutines:

1. **Name** `cUprxfLcon`
   Purpose Returns bounds (if $mode = 0$) or Jacobians (if $mode = 1$) of user-defined linear constraints on parameters **p**, **r**, and final states $\mathbf{x}(t_{N_{ts}+1})$.
   Template

```
      SUBROUTINE cUprxfLcon(mode, Np, Nr, Nx, Nflprxf,
     &   gLpr_p, gLpr_r, gLpr_xf,
     &   bl, bu,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Np, Nr, Nx, Nflprxf, iuser(*)
      DOUBLE PRECISION ruser(*),
     &   gLpr_p(Nflprxf, Np), gLpr_r(Nflprxf, Nr),
     &   gLpr_xf(Nflprxf,Nx)

      RETURN
      END
```

Arguments
`mode` - Job flag
`Np` - Number of parameters
`Nr` - Number of 'initial' parameters
`Nx` - Number of model states
`Nflprxf` - Number of linear constraints on $\mathbf{p}$, $\mathbf{r}$, and $\mathbf{x}(t_{N_{tx}+1})$
`gLpr_p` - Jacobian with respect to $\mathbf{p}$
`gLpr_r` - Jacobian with respect to $\mathbf{r}$
`gLpr_xf` - Jacobian with respect to $\mathbf{x}(t_{N_{tx}+1})$
`bl` - Array of lower bounds
`bu` - Array of upper bounds
`iuser` - Integer user array
`ruser` - Real user array

2. **Name** `cUxuLcon`
Purpose Returns bounds (if $mode = 0$) or Jacobians (if $mode = 1$) of user-defined linear constraints on states $\mathbf{x}$, controls $\mathbf{u}$, and parameters $\mathbf{p}$.
Template

```
      SUBROUTINE cUxuLcon(mode, Np, Nx, Nu, Nflxu,
     &   gLxu_p, gLxu_x, gLxu_u,
     &   bl, bu,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Np, Nx, Nu, Nflxu, iuser(*)
      DOUBLE PRECISION ruser(*),
     &   gLxu_p(Nflxu, Np), gLxu_x(Nflxu, Nx), gLxu_u(Nflxu, Nu)

      RETURN
      END
```

Arguments
`mode` - Job flag
`Np` - Number of parameters
`Nx` - Number of model states
`Nu` - Dimension of control
`Nflxu` - Number of linear constraints on $\mathbf{x}$, $\mathbf{u}$, and $\mathbf{p}$
`gLxu_p` - Jacobian with respect to $\mathbf{p}$
`gLxu_x` - Jacobian with respect to $\mathbf{x}$
`gLxu_u` - Jacobian with respect to $\mathbf{u}$
`bl` - Array of lower bounds
`bu` - Array of upper bounds
`iuser` - Integer user array
`ruser` - Real user array

3. **Name** `cUuLcon`
Purpose Returns bounds (if $mode = 0$) or Jacobians (if $mode = 1$) of user-defined linear constraints on controls $\mathbf{u}$ and parameters $\mathbf{p}$.
Template

```
      SUBROUTINE cUuLcon(mode, Np, Nu, Nflu,
     &   gLu_p, gLu_u,
     &   bl, bu,
     &   iuser, ruser)
      IMPLICIT NONE
```

```
      INTEGER Np, Nu, Nflu, iuser(*)
      DOUBLE PRECISION ruser(*),
     &   gLu_p(Nflu, Np), gLu_u(Nflu, Nu)

      RETURN
      END
```

Arguments
`mode` - Job flag
`Np` - Number of parameters
`Nu` - Dimension of control
`Nflu` - Number of linear constraints on $\mathbf{u}$, and $\mathbf{p}$
`gLu_p` - Jacobian with respect to $\mathbf{p}$
`gLu_u` - Jacobian with respect to $\mathbf{u}$
`bl` - Array of lower bounds
`bu` - Array of upper bounds
`iuser` - Integer user array
`ruser` - Real user array
Note These constraints are enforced both at multiple shooting grid points and at control parameterization grid points inside shooting intervals.

4. **Name** `cUprxfNLcon`
Purpose Returns value and Jacobians of user-defined nonlinear constraints on parameters $\mathbf{p}$, $\mathbf{r}$, and $\mathbf{x}(t_{N_{ts}+1})$.
Template

```
      SUBROUTINE cUprxfNLcon(Np, Nr, Nx, P, R, Xf, Nfnprxf,
     &   fNpr, gNpr_p, gNpr_r, gNpr_xf,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Np, Nr, Nx, Nfnprxf, iuser(*)
      DOUBLE PRECISION P(*), R(*), X(*),
     &   fNpr(Nfnprxf),
     &   gNpr_p(Nfnprxf, Np), gNpr_r(Nfnprxf, Nr),
     &   gNpr_xf(Nfnprxf, Nx),
     &   ruser(*)

      RETURN
      END
```

Arguments
`Np` - Number of parameters
`Nr` - Number of 'initial' parameters
`Nx` - Number of model states
`P` - Parameters
`R` - 'Initial' parameters
`Xf` - Final states
`Nfnpr` - Number of nonlinear constraints on $\mathbf{p}$, $\mathbf{r}$ and $\mathbf{x}(t_{N_{ts}+1})$
`fNpr` - Constraint value
`gNpr_p` - Jacobian with respect to $\mathbf{p}$
`gNpr_r` - Jacobian with respect to $\mathbf{r}$
`gNpr_xf` - Jacobian with respect to $\mathbf{x}(t_{N_{ts}+1})$
`iuser` - Integer user array
`ruser` - Real user array

5. **Name** `cUxuNLcon`
Purpose Returns value and Jacobians of user-defined nonlinear constraints on states **x**, controls **u**, and parameters **p**.
Template

```
      SUBROUTINE cUxuNLcon(Np, Nx, Nu, T, P, X, U, Nfnxu,
     &   fNxu, gNxu_p, gNxu_x, gNxu_u,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Np, Nx, Nu, Nfnxu, iuser(*)
      DOUBLE PRECISION T, P(*), X(*), U(*),
     &   fNxu(Nfnxu),
     &   gNxu_p(Nfnxu, Np), gNxu_x(Nfnxu, Nx),
     &   gNxu_u(Nfnxu, Nu),
     &   ruser(*)

      RETURN
      END
```

Arguments
`Np` - Number of parameters
`Nx` - Number of model states
`Nu` - Dimension of control
`T` - Current time
`P` - Parameters
`X` - Model states
`U` - Controls
`Nfnxu` - Number of nonlinear constraints on **x**, **u**, and **p**
`fNxu` - Constraint value
`gNxu_p` - Jacobian with respect to **p**
`gNxu_x` - Jacobian with respect to **x**
`gNxu_u` - Jacobian with respect to **u**
`iuser` - Integer user array
`ruser` - Real user array

6. **Name** `cUuNLcon`
Purpose Returns value and Jacobians of user-defined nonlinear constraints on controls **u** and parameters **p**.
Template

```
      SUBROUTINE cUuNLcon(Np, Nu, T, P, U, Nfnu,
     &   fNu, gNu_p, gNu_u,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Np, Nu, Nfnu, iuser(*)
      DOUBLE PRECISION T, P(*), U(*),
     &   fNu(Nfnu),
     &   gNu_p(Nfnu, Np), gNu_u(Nfnu, Nu),
     &   ruser(*)

      RETURN
      END
```

Arguments
`Np` - Number of parameters

`Nu` - Dimension of control
`T` - Current time
`P` - Parameters
`U` - Controls
`Nfnu` - Number of nonlinear constraints on **u** and **p**
`fNu` - Constraint value
`gNu_p` - Jacobian with respect to **p**
`gNu_u` - Jacobian with respect to **u**
`iuser` - Integer user array
`ruser` - Real user array
Note These constraints are enforced both at multiple shooting grid points and at control parameterization grid points inside shooting intervals.

2.1.3 File `Objective.f`

Name `cUprxfNLobj`
Purpose Evaluates the value and gradient of the nonlinear part in the objective function.
Template

```
      SUBROUTINE cUprxfNLobj(Np, Nr, Nx,
     &   P, R, Xf,
     &   fNpr, gNpr_p, gNpr_r, gNpr_xf,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Np, Nr, Nx, iuser(*)
      DOUBLE PRECISION P(*), R(*), Xf(*),
     &   fNpr, gNpr_p(Np), gNpr_r(Nr), gNpr_xf(Nx),
     &   ruser(*)

      RETURN
      END
```

Arguments
`Np` - Number of parameters
`Nr` - Number of 'initial' parameters
`Nx` - Number of model states
`P` - Parameters
`R` - 'Initial' parameters
`Xf` - Final states
`fNpr` - Objective value
`gNpr_p` - Jacobian with respect to **p**
`gNpr_r` - Jacobian with respect to **r**
`gNpr_xf` - Jacobian with respect to $\mathbf{x}(t_{N_{ts}+1})$
`iuser` - Integer user array
`ruser` - Real user array
Note - This subroutine is called only if $i_{nlcf} = 1$ (see Table 3). In this case, the final states must be part of the optimization variables (i.e.; $i_{xf} = 1$).

2.1.4 File `setX0.f`

Name `cUsetX0`
Purpose Returns inital values for the model states, for given values of the 'initial' parameters **r**.
Template

```
      SUBROUTINE cUsetX0(Nx,Nr,R,X0,iuser,ruser)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z), INTEGER (I-N)
```

```
      INTEGER Nr, Nx, iuser(*)
      DOUBLE PRECISION R(Nr), X0(Nx), ruser(*)

      RETURN
      END
```

Arguments
`Nx` - Number of model states
`Nr` - Number of 'initial' parameters
`R` - Array of 'initial' parameters
`X0` - Initial states
`iuser` - Integer user array
`ruser` - Real user array
Note
This subroutine is processed through ADIFOR to generate the file `s_setX0`.

2.1.5 File `Krylov.f`

The file `Krylov.f` contains the subroutines `cUpsol` and `cUpjac` that are used if the Kryolv iterative method is selected ($iflag(6)$). For a detailed description of these two subroutines, see the DASPK3.0 documentation. If a direct method is selected then these subroutines must be treated as dummy subroutines. In this case you can use the file `coopt/templates/Krylov.f`.

2.1.6 File `sysM.f`

Name `cUsysM`
Purpose Returns the residual of the state equations (1) at a given time, for given values of the problem parameters, controls, derivative of controls, states, and derivative of states.
Template

```
      SUBROUTINE  cUsysM(Nx, Np, Nu,
     &   t, X, Xdot, P, U, Ud, F, cj,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Nx, Np, Nu
      DOUBLE PRECISION t, X(Nx), Xdot(Nx)
      DOUBLE PRECISION P(Np), U(Nu), Ud(Nu)
      DOUBLE PRECISION F(Nx), cj
      INTEGER iuser(*)
      DOUBLE PRECISION ruser(*)

      RETURN
      END
```

Arguments
`Nx` - Number of model states
`Np` - Number of parameters
`Nu` - Dimension of control
`t` - Current time
`X` - Model states
`Xdot` - Model state derivatives
`P` - Array of parameters
`U` - Value of control at current time
`Ud` - Value of first control derivative at current time
`F` - Array of residuals of the model equations
`cj` - Reciprocal of integration time step

iuser - Integer user array
ruser - Real user array
Note
This subroutine is processed through ADIFOR (together with the file sysC.f) to generate the files g_sysM and j_sysM.

2.1.7 File sysC.f

Name cUsysC
Purpose Returns the right side of the additional equation (3) corresponding to the cost function, for given values of the problem parameters, controls, derivative of controls, states, and derivative of states.
Template

```
      SUBROUTINE  cUsysC(Nx, Np, Nu,
     &   t, X, Xdot, P, U, Ud, Fc,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Nx, Np, Nu
      DOUBLE PRECISION t, X(Nx), Xdot(Nx)
      DOUBLE PRECISION P(Np), U(Nu), Ud(Nu)
      DOUBLE PRECISION Fc
      INTEGER iuser(*)
      DOUBLE PRECISION ruser(*)

      RETURN
      END
```

Arguments
Nx - Number of model states
Np - Number of parameters
Nu - Dimension of control
t - Current time
X - Model states
Xdot - Model state derivatives
P - Array of parameters
U - Value of control at current time
Ud - Value of first control derivative at current time
Fc - Right side of Eq. 3
iuser - Integer user array
ruser - Real user array
Note
This subroutine is processed through ADIFOR (together with the file sysM.f) to generate the files g_sysC and j_sysC.

2.1.8 File Output.f

The file Output.f contains the following subroutines:

1. **Name** cUoutState
   Purpose Allows the user to output initial and final simulation results in different files and using a different format than the default provided by COOPT.
   Template

```
      SUBROUTINE cUoutState(whichRun,whichTime,t,Nx,v)
      IMPLICIT NONE
      INTEGER whichRun, whichTime,Nx
```

```
      DOUBLE PRECISION t,v(Nx+1)

      RETURN
      END
```

Arguments
`whichRun` - Specifies if this is initial run (-1) or final run (1)
`whichTime` - Specifies the integration time. A value of -1 means that this is the first call to `cUoutState` with $t = t_1$, while a value of 1 means that this call is at $t = t_{max}$. A value of 0 indicates a call at an intermediate time.
`t` - Current time
`Nx` - Number of model states
`v` - Integration variables (contains the model states in the first Nx positions and the additional quadrature variable in the last position.

2. **Name** `cUoutSensi`
Purpose Allows the user to output results of a sensitivity solve in a different file and using a different format than the default provided by COOPT.
Template

```
      SUBROUTINE cUoutSensi(whichRun,whichTime,t,Nx,Np,Nr,Nuu1,
     &                      v,v_p,v_r,v_uu)
      IMPLICIT NONE
      INTEGER whichRun, whichTime,Nx,Np,Nr,Nuu1
      DOUBLE PRECISION t
      DOUBLE PRECISION v(Nx+1)
      DOUBLE PRECISION v_p(Nx+1,Np), v_r(Nx+1,Nr), v_uu(Nx+1,Nuu1)

      RETURN
      END
```

Arguments
`whichRun` - Specifies if this is initial run (-1) or final run (1)
`whichTime` - Specifies the integration time. A value of -1 means that this is the first call to `cUoutState` with $t = t_1$, while a value of 1 means that this call is at $t = t_{max}$. A value of 0 indicates a call at an intermediate time.
`t` - Current time
`Nx` - Number of model states
`Np` - Number of parameters
`Nr` - Number of 'initial' parameters
`Nuu1` - Number of control parameters used to parameterize the control in the current shooting interval
`v` - Integration variables (contains the model states in the first Nx positions and the additional quadrature variable in the last position.
`v_p` - sensitivities with respect to parameters **p**
`v_r` - sensitivities with respect to parameters **r**
`v_uu` - sensitivities with respect to control parameters **UU**
Note The first Nx rows of `v_p`, `v_r`, and `v_uu` contain sensitivities of the model states, while the last row contains sensitivities of the additional variable.

2.1.9 ADIFOR Script File (`AdiScript`)

In order to generate the integration Jacobian and the sensitivity equation residuals, we need partial derivative matrices of the model equations. In COOPT, these matrices are obtained using the automatic differentiation tool ADIFOR. The script file `AdiScript` is provided in `coopt/templates`. You can copy it into your example directory (`coopt/myproblem`) and modify it to suit your problem. Normally, you should not change any

of the ADIFOR preprocessor options in this file. All problem-dependent information is contained in the composition files. For more information on ADIFOR, see the ADIFOR 2.0 Users' Guide. For more details on how the ADIFOR generated subroutines are called from COOPT see Section 3.

The synopsis of `AdiScript` is

```
AdiScript [-h|-H] [-a|-A] Nx [-g|-G] [-s|-S] [-j|-J] Nx
```

with the following options supported:

`-h`	Print command synopsis
`-g`	Generate ADIFOR files for residual computation
`-s`	Generate ADIFOR files for computing $\mathbf{X}_0$
`-j Nx`	Generate ADIFOR files for Jacobian computation
`-a Nx`	Generate all ADIFOR files

Note The argument to the option `j` or `a` must agree with the *banded Jacobian* option selected through $info(32)$ in `cUinit` (file `Initial.f`). If you have set $info(32) = 0$, then call `AdiScript` with an argument of at least $Nx + 1$; if $info(32) = 1$, then call `AdiScript` with an argument of at least $ML + MU + 1$, where ML is the lower band and MU is the upper band of the Jacobian.

2.1.10 ADIFOR Composition Files (`setX0.cmp` and `system.cmp`)

The ADIFOR composition files specify all files that need to be processed at the same time through ADIFOR. You must include all files containing subroutines that are called from the top subroutines (`sys` and `setX0`). If your top subroutines do not call any other routine, you can simply use the composition files provided in `coopt/templates`.

2.2 Compiling and Running the Problem

At this point you should have a directory `cooopt/myproblem` which contains the following Fortran files: `Initial.f`, `Constraints.f`, `ShowResults.f`, `setX0.f`, `sysM.f`, and `sysC.f`, the ADIFOR script file `AdiScript`, and the ADIFOR composition files `setX0.cmp` and `system.cmp`.

The next step is to invoke ADIFOR to automatically generate derivative computation subroutines. Before running the ADIFOR script check that

- The composition files list all the Fortran files containing subroutines that are called from the top subroutines (`setX0` and `sys`)
- The variable names in the script file are the same as the arguments in the top subroutines
- The ADIFOR preprocessor options listed in Table 5 are unchanged

Running the ADIFOR script file will create a set of Fortran files in the ADIFOR output directory (the default directory is `coopt/myproblem/AD_output`) which you must copy into your directory.

Next, modify the makefile `coopt/myproblem/Makefile` to list all user files, and create the executable by typing

```
% make coopt USER=myproblem
```

in the `coopt` directory. The executable file `coopt` will be created in `coopt/myproblem`.

2.3 Output Files

A problem definition file, specifying problem dimensions and user selected flags is generated as `{OutDir}/{Prob}.def`. The SNOPT/DAOPT output file is generated as `{OutDir}/{Prob}.out`. The following two files contain the optimal values of the parameters $\mathbf{p}^*$ and $\mathbf{r}^*$ and the time history of the optimal control $\mathbf{u}^*$:

- `{OutDir}/{Prob}.param` - Optimal parameters
  This file contains three sections specifying the optimal parameters $\mathbf{p}$, the optimal 'initial' parameters $\mathbf{r}$, and the optimal control parameters.

- {OutDir}/{Prob}.con - Optimal control
  Each line of this file contains $N_u + 1$ values for the time and controls.

If the $iflag(1) = 1$ then initial and final simulations (that is simulations with the initial guess of parameters and controls and with the optimal parameters and controls, respectively) are performed. In this case the user has the option of requiring automatic output (by setting $iflag() = 0$ in which case the following two result files are generated:

- {OutDir}/{Prob}.init - Simulation results with initial control
  Each line of this file contains $N_{tx} + 2$ values, corresponding to the integration time, model states, and additional cost function variable.

- {OutDir}/{Prob}.final - Simulation results with optimal control
  Each line of this file contains $N_{tx} + 2$ values, corresponding to the integration time, model states, and additional cost function variable.

Additionaly, if $iflag(1) = 1$ and $iflag() = 1$ then the user has control on the names and format of the data files containing initial and final simulation results. These two simulations are run using DASPK3.0's *intermediate output* mode with control being given to the user subroutine `cUoutState` after each successful integration step.

2.4 Minimum Time Problems

In this section we consider the case in which the simulation length is one of the optimization parameters.

In order to solve such a problem in COOPT, you have to specify which optimization parameter $\mathbf{p}$ is the final simulation time. This is done by setting the flag i_{tf} in the user subroutine `cUinit` (see Table 3). To specify the dependency of your model on the final time (i.e. $p_{i_{tf}}$) you must scale the time by this parameter in both `sysM` and `sysC`. That is, you have to rewrite your model equations in terms of the scaled time $\tau = t/p_{i_{tf}}$.

However, if you already have the files `sysM` and `sysC` written in terms of the 'real' time t, rescaling the time in the model equations can be done with minimal changes. First, note that the integration will be performed over $\tau = t/\rho \in [0, 1]$, where $\rho = p_{i_{tf}}$. As a consequence, all time derivatives scale as

$$\frac{d}{dt} = \frac{d}{d\tau}\frac{d\tau}{dt} = \frac{1}{\rho}\frac{d}{d\tau} \tag{6}$$

Therefore, assuming that $i_{tf} = 1$, the modified `sysM` subroutine will be

```
      SUBROUTINE  sysM(Nx, Np, Nu,
     &   TAU, X, Xdot, P, U, Ud, F, cj,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Nx, Np, Nu
      DOUBLE PRECISION TAU, X(Nx), Xdot(Nx)
      DOUBLE PRECISION P(Np), U(Nu), Ud(Nu)
      DOUBLE PRECISION F(Nx), cj
      INTEGER iuser(*)
      DOUBLE PRECISION ruser(*)
      DOUBLE PRECISION TIME
      INTEGER i

      TIME = P(1)*TAU
      do i = 1,Nx
         Xdot(i) = Xdot(i)/P(1)
      enddo
      do i = 1,Nu
         Ud(i) = Ud(i)/P(1)
```

```
      enddo
C
C     User inserts here evaluation of F
C     ......
C
      do i = 1,Nx
         Xdot(i) = Xdot(i)*P(1)
      enddo
      do i = 1,Nu
         Ud(i) = Ud(i)*P(1)
      enddo

      RETURN
      END
```

Note that the time derivatives must be reconverted to derivatives with respect to τ, as the subroutine `sysM` is not supposed to alter $\mathbf{x}'$ or $\mathbf{u}'$.

A similar treatment must be applied to the subroutine `sysC`. However, since this routine returns only the right side of Eq. 3, the final result must be multiplied by ρ. The modified subroutine `sysC` becomes

```
      SUBROUTINE  cUsysC(Nx, Np, Nu,
     &   TAU, X, Xdot, P, U, Ud, Fc,
     &   iuser, ruser)
      IMPLICIT NONE
      INTEGER Nx, Np, Nu
      DOUBLE PRECISION TAU, X(Nx), Xdot(Nx)
      DOUBLE PRECISION P(Np), U(Nu), Ud(Nu)
      DOUBLE PRECISION Fc
      INTEGER iuser(*)
      DOUBLE PRECISION ruser(*)
      DOUBLE PRECISION TIME
      INTEGER i

      TIME = P(1)*TAU
      do i = 1,Nx
         Xdot(i) = Xdot(i)/P(1)
      enddo
      do i = 1,Nu
         Ud(i) = Ud(i)/P(1)
      enddo
C
C     User inserts here evaluation of Fc
C     ....
C
      Fc = Fc*P(1)

      do i = 1,Nx
         Xdot(i) = Xdot(i)*P(1)
      enddo
      do i = 1,Nu
         Ud(i) = Ud(i)*P(1)
      enddo

      RETURN
      END
```

Table 1: User files and subroutines

File	Subroutine	Description
`Initial.f`	`cUnames`	Set the specification, output directory, and problem names
	`cUspace`	Specify integer and real user space requests
	`cUinit`	Specify problem dimensions and flags for the optimization and the integration
	`cUvars`	Specify the algebraic variables
	`cUguess`	Provides initial guess for problem parameters, states at multiple shooting points, and controls
	`cUbounds`	Specify bounds on the optimization variables
	`cUtol`	Specifiy integration tolerances
`Constraints.f`	`cUprxfLcon`	Set the Jacobian of the linear constraints involving parameters $\mathbf{p}$, $\mathbf{r}$ and final states $\mathbf{X}_f$
	`cUxuLcon`	Set the Jacobian of the linear constraints involving parameters $\mathbf{p}$, states $\mathbf{X}$, and controls $\mathbf{u}$
	`cUuLcon`	Set the Jacobian of the linear constraints involving parameters $\mathbf{p}$ and controls $\mathbf{u}$
	`cUprxfNLcon`	Set the value and Jacobian of the nonlinear constraints involving parameters $\mathbf{p}$, $\mathbf{r}$ and final states $\mathbf{X}_f$
	`cUxuNLcon`	Set the value and Jacobian of the nonlinear constraints involving parameters $\mathbf{p}$, states $\mathbf{X}$, and controls $\mathbf{u}$
	`cUuNLcon`	Set the value and Jacobian of the nonlinear constraints involving parameters $\mathbf{p}$ and controls $\mathbf{u}$
`Objective.f`	`cUprxfNLobj`	Evaluates the value and gradient of the nonlinear part of the objective function
`Krylov.f`	`cUpsol`[a]	
	`cUpjac`[a]	
`setX0.f`	`setX0`[b]	Set initial states $\mathbf{X}_0$, for given parameters $\mathbf{r}$
`sysM.f`	`sysM`[c]	Evaluates the residual of the model equations
`sysC.f`	`sysC`[c]	Evaluates the right side of the cost function equation

[a] These subroutines are called only if the Krylov iterative solver option is selected.

[b] This subroutine is processed through ADIFOR to obtain `s$setX0`. Only the ADIFOR generated routine is called in COOPT.

[c] This subroutines are processed through ADIFOR (together with the subroutine `sys`) to obtain `g$sys` and `j$sys`. Only the ADIFOR generated routines are called in COOPT.

Table 2: The `idim` array

Entry	Name	Description
1	N_p	Number of parameters
2	N_r	Number of 'initial' parameters
3	N_{xd}	Number of differential model states
4	N_{xa}	Number of algebraic model states
5	N_u	Number of controls
6	N_q	Order of control parameterization polynomial
7	N_{tx}	Number of multiple shooting intervals
8	N_{tu1}	Number of control parameterization intervals per shooting interval
9	N_{fnpr}	Number of nonlinear constraints on $\mathbf{p}$, $\mathbf{r}$ and $\mathbf{X}_f$
10	N_{fnu}	Number of nonlinear constraints on $\mathbf{p}$ and $\mathbf{u}$
11	N_{fnxu}	Number of nonlinear constraints on $\mathbf{p}$, $\mathbf{X}$, and $\mathbf{u}$
12		not used
13		not used
14		not used
15	N_{flpr}	Number of linear constraints on $\mathbf{p}$, $\mathbf{r}$, and $\mathbf{X}_f$
16	N_{flu}	Number of linear constraints on $\mathbf{p}$ and $\mathbf{u}$
17	N_{flxu}	Number of linear constraints on $\mathbf{p}$, $\mathbf{X}$, and $\mathbf{u}$
18		not used
19		not used
20		not used

Table 3: The `iflag` array

Entry	Name	Description
1	i_{run}	Perform initial and/or final simulations? 0 No 1 Both. Use default output 2 Only initial simulation and then stop. Use default output 3 Only sensitivity computation and then stop. Use default output -1 Both. Use user output -2 Only initial simulation and then stop. Use user output -3 Only sensitivity computation and then stop. Use user output
2	i_{opt}	Which optimizer? 0: SNOPT; 1: DAOPT
3	i_{Xf}	Exclude final states from the optimization? 0: No; 1: Yes
4	i_{Tf}	Include final time in the optimization? 0: No; 1: Yes
5	i_{NLcf}	Does the cost function contain a nonlinear part depending on the final state and final time? 0: No; 1: Yes
6	i_{bndU}	Are there additional bounds on the control? 0 No 1 Yes, the control must be within bounds everywhere (works only for $N_q = 2$) 2 Yes, the control must be monotonic on subintervals (works only for $N_q = 2$)
7	i_{guessU}	How is the initial guess for controls provided? 0 Constant value over the entire time interval 1 Control parameters provided 2 Tabulated data
8	i_{initX}	How is the initial guess for states at shooting interfaces obtained? 0 Same set of values at all shooting interfaces 1 Obtained from an initial simulation with initial guess for parameters and controls
9-15		not used
16	i_{X0}	Should a consistent DAE initialization algorithm be applied at the begining of each shooting interval? See DASPK3.0 documentation
17	i_{step}	Is the initial interagtion step size provided? 0: No; 1: Yes and set $rdata(5)$
18	i_{tol}	How are integration tolerances provided? 0 Same relative and absolute tolerances for both states and sensitivities. Set $rdata(1) = rtol$ and $rdata(2) = atol$ 1 Different tolerances for states and sensitivities Set $rdata(1) = rtol_states$, $rdata(2) = atol_states$, $rdata(3) = rtol_sensi$, $rdata(4) = atol_sensi$ 2 Compute absolute tolerances for sensitivity variables from estimates of state magnitudes (provided through the subroutine `cUtol`). Set $rdata(1) = rtol_states$, $rdata(2) = atol_states$.
19	i_{ls}	What linear solver is used by DASPK3.0? 0: Direct; 1: Iterative
20	i_{bandJ}	Is the integration Jacobian banded? (used only if $i_{ls} = 0$) 0 No 1 Yes. Set $idata(6) = b_{upper}$ and $idata(7) = b_{lower}$.
21	i_{Kr1}	Does the iterative solver use default settings? (used only if $i_{ls} = 1$) 0 Yes 1 No. Set $idata(8) = MAXL$, $idata(9) = KMP$, $idata(10) = NRMAX$, and $rdata(6) = EPLI$ (see DASPK3.0 documentation).
22	i_{Kr2}	Does the iterative solver use a Jacobian? (used only if $i_{ls} = 1$) 0: No; 1: Yes
23-24		not used
25	i_{stagg}	What method is used in sensitivity computations? (see DASPK3.0 documentation)
26	i_{err}	Are the sensitivity variables included in the error test? 0: Yes; 1: No (see DASPK3.0 documentation)
27-30		not used

Table 4: The `idata` array

Entry	Name	Description
1	i_{Udata}	Number of points in the initial control data (if $i_{guessU} = 2$)
2-4		not used
5	i_{dae}	DAE index of the model equations
6	b_{upper}	Upper band width of the Jacobian (used only if $i_{bandJ} = 1$)
7	b_{lower}	Lower band width of the Jacobian (used only if $i_{bandJ} = 1$)
8	$MAXL$	MAXL parameter (used only if $i_{Kr1} = 1$)
9	KMP	KMP parameter (used only if $i_{Kr1} = 1$)
10	$NRMAX$	NRMAX parameter (used only if $i_{Kr1} = 1$)
12	l_{wp}	Length of real work array for `cUpsol` and `cUpjac` (used only if $i_{ls} = 1$)
13	l_{iwp}	Length of integer work array for `cUpsol` and `cUpjac` (used only if $i_{ls} = 1$)

Table 5: Fixed ADIFOR preprocessor options

Section	Option	Value
1	AD_TOP	cNsysG
	AD_EXCEPTION_FLAVOR	performance
	AD_SCALAR_GRADIENTS	true
	AD_PREFIX	g
	AD_PMAX	1
2	AD_TOP	cNsysJ
	AD_EXCEPTION_FLAVOR	performance
	AD_PREFIX	j
3	AD_TOP	cUsetX0
	AD_EXCEPTION_FLAVOR	performance
	AD_SUPRESS_LDG	true
	AD_PREFIX	s
	AD_PMAX	1

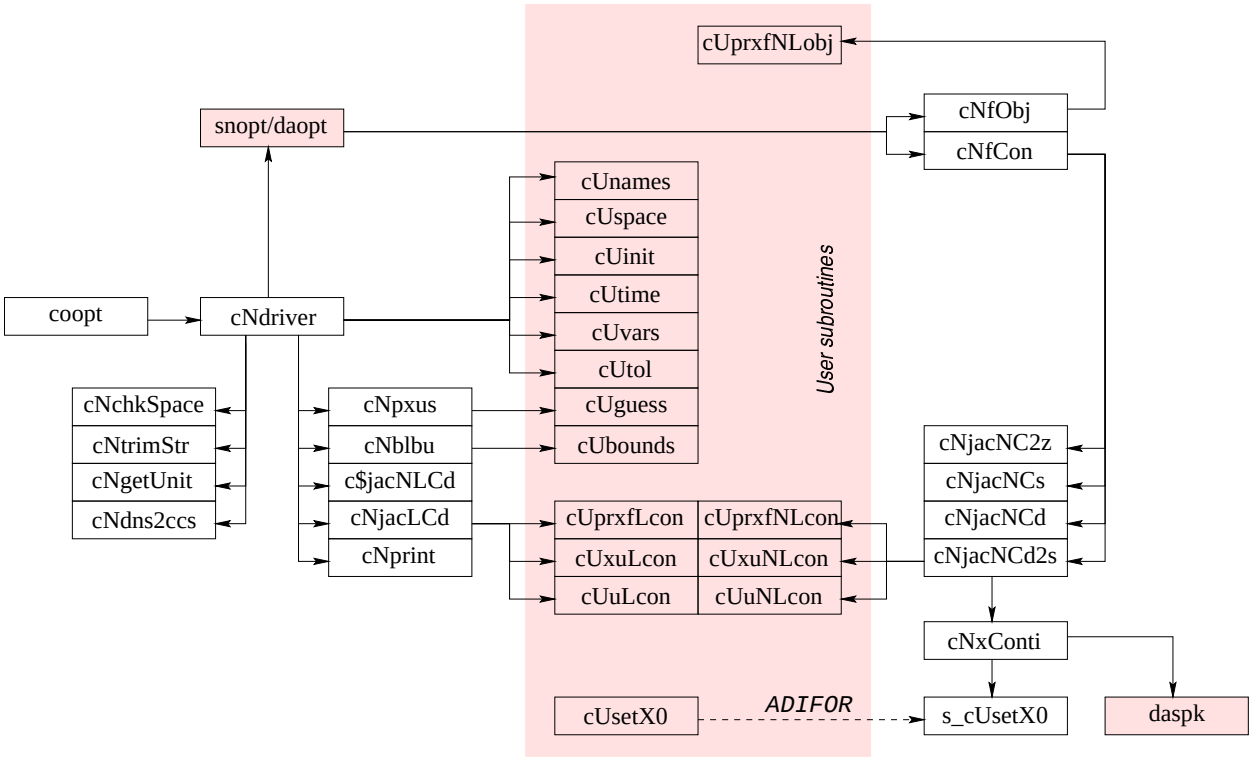

Figure 2: Structure of COOPT

3 Implementation

The structure and components of COOPT are presented in Fig 2. All notation used in the following sections are those used in the code. All symbols and their meanings are listed in Table 6.

3.1 Control Parameterization

On each multiple shooting interval, each control u_{iu}, $iu = 1, 2, ..., N_u$ is represented as a piecewise polynomial of order N_q. Each multiple shooting interval is subdivided into N_{tu1} control intervals (see Fig. 3). The control u_{iu} is then parameterized by a minimum number of parameters that ensure continuity of the control and of its derivative; i.e., $u_{iu} \in \mathcal{C}^1$. Consider the multiple shooting interval $[itx, itx+1]$. The length of each control subinterval is then $\Delta t_{itx} = (t_{itx+1} - t_{itx})/N_{tu1}$. Let t be such that $it \cdot \Delta t_{itx} \leq t - t_{itx} < (it+1) \cdot \Delta t_{itx}$. Consider the order N_q polynomial approximation of control u_{iu} at t, using the nondimensionalized variable $t^* = (t - t_{itx} - it \cdot \Delta t_{itx})/\Delta t_{itx} \in [0, 1)$

$$
\begin{aligned}
u_{iu}(t) &= U_{iu,itx}^{it+1,0} + U_{iu,itx}^{it+1,1} t^* + \sum_{j=1}^{N_q-1} U_{iu,itx}^{it+1,j+1} t^{*j+1} \\
\Delta t_{itx} \cdot u'_{iu}(t) &= U_{iu,itx}^{it+1,1} + \sum_{j=1}^{N_q-1} U_{iu,itx}^{it+1,j+1} (j+1) t^{*j} \qquad\qquad it = 0, 1, 2, ..., N_{tu1}
\end{aligned}
\tag{7}
$$

Imposing the conditions $u_{iu} \in \mathcal{C}^1$, the parameters $U_{iu,itx}^{it+1,0}$ and $U_{iu,itx}^{it+1,1}$ must satisfy the following recursive relations:

$$\begin{aligned} U_{iu,itx}^{it+1,0} &= U_{iu,itx}^{it,0} + U_{iu,itx}^{it,1} + \sum_{j=1}^{N_q-1} U_{iu,itx}^{it,j+1} \\ U_{iu,itx}^{it+1,1} &= U_{iu,itx}^{it,1} + \sum_{j=1}^{N_q-1} (j+1) U_{iu,itx}^{it,j+1} \qquad\qquad it = 1, 2, ..., N_{tu1} \end{aligned} \tag{8}$$

By induction, we can see that

$$\begin{aligned} U_{iu,itx}^{it+1,0} &= U_{iu,itx}^{1,0} + it \cdot U_{iu,itx}^{1,1} + \sum_{k=1}^{it} \sum_{j=1}^{N_q-1} [(j+1)(it-k)+1] U_{iu,itx}^{k,j+1} \\ U_{iu,itx}^{it+1,1} &= U_{iu,itx}^{1,1} + \sum_{k=1}^{it} \sum_{j=1}^{N_q-1} (j+1) U_{iu,itx}^{k,j+1} \qquad\qquad it = 0, 1, 2, ..., N_{tu1} \end{aligned} \tag{9}$$

The piece-wise polynomial approximation of u_{iu} in the shooting interval $[itx, itx+1]$ can thus be represented by the following $2 + N_{tu1}(N_q - 1)$ parameters:

$$\begin{array}{cccccccc} U_{iu,itx}^{1,0} & & & & & & & \\ U_{iu,itx}^{1,1} & & & & & & & \\ U_{iu,itx}^{1,2} & U_{iu,itx}^{2,2} & \cdots & U_{iu,itx}^{it,2} & U_{iu,itx}^{it+1,2} & \cdots & U_{iu,itx}^{Ntu1,2} & \\ U_{iu,itx}^{1,3} & U_{iu,itx}^{2,3} & \cdots & U_{iu,itx}^{it,3} & U_{iu,itx}^{it+1,3} & \cdots & U_{iu,itx}^{Ntu1,3} & \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \\ U_{iu,itx}^{1,N_q} & U_{iu,itx}^{2,N_q} & \cdots & U_{iu,itx}^{it,N_q} & U_{iu,itx}^{it+1,N_q} & \cdots & U_{iu,itx}^{N_{tu1},N_q} & \end{array}$$

The total number of control parameters is then

$$N_{uu} = N_{tx} N_u (2 + N_{tu1}(N_q - 1)) \tag{10}$$

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 $[it, it+1]$, with $\mathcal{C}^1$ control continuity enforced by the optimization algorithm. First, a reduced number of parameters is required, $N_{tx} N_u (2 + N_{tu1}(N_q - 1))$ vs. $N_{tx} N_u N_{tu1}(N_q + 1)$ which means that fewer sensitivity equations must be solved. Using an order N_q polynomial on each control subinterval introduces discontinuities in the sensitivities with respect to parameters $U_{iu,itx}^{it,0}$ 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 on the entire multiple shooting interval without restarts at the beginning of each control subinterval.

3.2 Continuity 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.

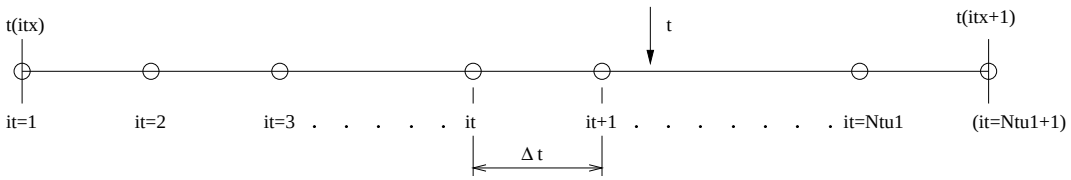

Figure 3: Control subintervals within a shooting interval

3.2.1 State Continuity Constraints

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

$$\begin{aligned} &\mathbf{C}_{itx} \equiv \mathbf{X}_{itx+1} - \mathbf{x}(t_{itx+1}, t_{itx}, \mathbf{X}_{itx}, \mathbf{p}, \mathbf{u}) = \mathbf{0} \\ &\text{where } \mathbf{C}_{itx} \in \mathbf{R}^{N_x+1}, \quad itx = 1, 2, ..., N_{tx} \end{aligned} \tag{11}$$

We denote by

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

the vector of discretized states and by

$$\mathbf{U} = \{\mathbf{U}_1, \mathbf{U}_2, ..., \mathbf{U}_{N_{tx}}\} \in \mathbf{R}^{N_{tx} N_u (2+N_{tu1}(N_q-1))}$$

the vector of control parameters, with $\mathbf{U}_{itx}$ representing the parameterization of all controls on the interval $[t_{itx}, t_{itx+1}]$. 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 of Eq. 11 into the array

$$\mathbf{C}(\mathbf{p}, \mathbf{r}, \mathbf{X}, \mathbf{U}) \equiv \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ \vdots \\ \mathbf{C}_{itx} \\ \vdots \\ \mathbf{C}_{N_{tx}} \end{bmatrix} \equiv \begin{bmatrix} \mathbf{X}_2 - \mathbf{x}(t_2, t_1, \mathbf{X}_1, \mathbf{p}, \mathbf{r}, \mathbf{U}_1) \\ \mathbf{X}_3 - \mathbf{x}(t_3, t_2, \mathbf{X}_2, \mathbf{p}, \mathbf{U}_2) \\ \vdots \\ \mathbf{X}_{itx+1} - \mathbf{x}(t_{itx+1}, t_{itx}, \mathbf{X}_{itx}, \mathbf{p}, \mathbf{U}_{itx}) \\ \vdots \\ \mathbf{X}_{N_{tx}+1} - \mathbf{x}(t_{N_{tx}+1}, t_{N_{tx}}, \mathbf{X}_{N_{tx}}, \mathbf{p}, \mathbf{U}_{N_{tx}}) \end{bmatrix} = \mathbf{0} \tag{12}$$

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

$$\mathbf{J} = [\mathbf{J}_\mathbf{p}, \mathbf{J}_\mathbf{r}, \mathbf{J}_\mathbf{X}, \mathbf{J}_\mathbf{U}] \tag{13}$$

where

$$\mathbf{J}_\mathbf{p} = \begin{bmatrix} -\partial\mathbf{x}(t_2)/\partial\mathbf{p} \\ -\partial\mathbf{x}(t_3)/\partial\mathbf{p} \\ \vdots \\ -\partial\mathbf{x}(t_{itx+1})/\partial\mathbf{p} \\ \vdots \\ -\partial\mathbf{x}(t_{N_{tx}+1})/\partial\mathbf{p} \end{bmatrix} \tag{14}$$

$$\mathbf{J}_\mathbf{r} = \begin{bmatrix} -\partial\mathbf{x}(t_2)/\partial\mathbf{r} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \tag{15}$$

$$\mathbf{J_X} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ -\frac{\partial \mathbf{x}(t_3)}{\partial \mathbf{X}_2} & \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\frac{\partial \mathbf{x}(t_4)}{\partial \mathbf{X}_3} & \mathbf{I} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} & \mathbf{I} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & -\frac{\partial \mathbf{x}(t_{N_{tx}+1})}{\partial \mathbf{X}_{N_{tx}}} & \mathbf{I} \end{bmatrix} \tag{16}$$

$$\mathbf{J_U} = \begin{bmatrix} -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{U}_1} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & -\frac{\partial \mathbf{x}(t_3)}{\partial \mathbf{U}_2} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{U}_{itx}} & \cdots & \mathbf{0} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & -\frac{\partial \mathbf{x}(t_{N_{tx}+1})}{\partial \mathbf{U}_{N_{tx}}} \end{bmatrix} \tag{17}$$

In the previous equations, we have made the simplifying notation $\mathbf{x}(t_{itx+1}) \equiv \mathbf{x}(t_{itx+1}, t_{itx}, \mathbf{X}_{itx}, \mathbf{p}, \mathbf{U}_{itx})$. As Eq. 16 shows, the Jacobian $\mathbf{J_X}$ is nonsingular. Multiplying the constraints $\mathbf{C}$ and the Jacobian $\mathbf{J}$ to the left by $\mathbf{J_X^{-1}}$, we obtain the modified constraints and modified Jacobian as

$$\hat{\mathbf{C}} = \mathbf{J_X^{-1}} \mathbf{C}, \tag{18}$$

$$\hat{\mathbf{J}} = [\mathbf{J_X^{-1}} \mathbf{J_p}, \mathbf{J_X^{-1}} \mathbf{J_r}, \mathbf{J_X^{-1}} \mathbf{J_X}, \mathbf{J_X^{-1}} \mathbf{J_U}] \equiv [\hat{\mathbf{P}}, \hat{\mathbf{R}}, \mathbf{I}, \hat{\mathbf{U}}]. \tag{19}$$

To compute the matrix $\hat{\mathbf{C}}$, we partition it into N_{tx} vertical blocks. We find that

$$\begin{aligned} \hat{\mathbf{C}}_1 &= \mathbf{C}_1 \\ \hat{\mathbf{C}}_{itx} &= \mathbf{C}_{itx} + \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{C}}_{itx-1}, \qquad itx = 2, 3, ..., N_{tx}. \end{aligned} \tag{20}$$

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

$$\begin{aligned} \hat{\mathbf{P}}_1 &= -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{p}} \\ \hat{\mathbf{P}}_{itx} &= -\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{p}} + \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{P}}_{itx-1}, \qquad itx = 2, 3, ..., N_{tx} \end{aligned} \tag{21}$$

and

$$\begin{aligned} \hat{\mathbf{R}}_1 &= -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{r}} \\ \hat{\mathbf{R}}_{itx} &= \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{R}}_{itx-1}, \qquad itx = 2, 3, ..., N_{tx}. \end{aligned} \tag{22}$$

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

$$\begin{aligned}
\hat{\mathbf{U}}_{1,1} &= -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{U}_1} \\
\hat{\mathbf{U}}_{itx,1} &= \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{U}}_{itx-1,1} \\
\hat{\mathbf{U}}_{itx,2} &= \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{U}}_{itx-1,2} \\
&\vdots \\
\hat{\mathbf{U}}_{itx,itx-1} &= \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{U}}_{itx-1,itx-1} \\
\hat{\mathbf{U}}_{itx,itx} &= -\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{U}_{itx}}
\end{aligned} \tag{23}$$

for $itx = 2, 3, ..., N_{tx}$. In Section 3.5 we present in more detail how the sensitivity equations are formed and solved.

3.2.2 Control Continuity Constraints

Continuity conditions on u_{iu} and u'_{iu} at t_{itx}, $itx = 2, ..., N_{tx}$ result in the following $N_{fu} = 2(N_{tx} - 1)N_u$ linear constraints on the control parameters:

$$\begin{aligned}
C^1_{iu,itx} &\equiv U^{1,0}_{iu,itx} + N_{tu1} \cdot U^{1,1}_{iu,itx} + \sum_{k=1}^{N_{tu1}} \sum_{j=1}^{N_q - 1} [(j+1)(N_{tu1} - k) + 1] U^{k,j+1}_{iu,itx} - U^{1,0}_{iu,itx+1} \\
C^2_{iu,itx} &\equiv U^{1,1}_{iu,itx} + \sum_{k=1}^{N_{tu1}} \sum_{j=1}^{N_q - 1} (j+1) U^{k,j+1}_{iu,itx} - \frac{\Delta t_{itx+1}}{\Delta t_{itx}} U^{1,1}_{iu,itx+1}, \qquad itx = 1, 2, ..., N_{tx}.
\end{aligned} \tag{24}$$

The Jacobian entries of these constraints are

$$\begin{aligned}
&\frac{\partial C^1_{iu,itx}}{\partial U^{1,0}_{iu,itx}} = 1; \quad \frac{\partial C^1_{iu,itx}}{\partial U d^1_{iu,itx}} = N_{tu1}; \quad \frac{\partial C^1_{iu,itx}}{\partial U^{k,j+1}_{iu,itx}} = (j+1)(N_{tu1} - k) + 1; \quad \frac{\partial C^1_{iu,itx}}{\partial U^{1,0}_{iu,itx+1}} = -1 \\
&\frac{\partial C^2_{iu,itx}}{\partial U^{1,0}_{iu,itx}} = 0; \quad \frac{\partial C^2_{iu,itx}}{\partial U^{1,1}_{iu,itx}} = 1; \quad \frac{\partial C^2_{iu,itx}}{\partial U^{k,j+1}_{iu,itx}} = j + 1; \quad \frac{\partial C^2_{iu,itx}}{\partial U^{1,1}_{iu,itx+1}} = -\frac{\Delta t_{itx+1}}{\Delta t_{itx}} \\
&itx = 1, 2, ..., N_{tx} \\
&iu = 1, 2, ..., N_u \\
&j = 1, 2, ..., N_q - 1 \\
&k = 1, 2, ..., N_{tu1}.
\end{aligned} \tag{25}$$

The control continuity constraints are arranged in the following order:

for each multiple shooting interval itx
  for each control iu
    $C^1_{iu,itx}$
    $C^2_{iu,itx}$
  end
end

3.3 Additional Control Constraints

3.3.1 Bounds at control intervals

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.

Therefore, on each shooting interval itx, we add $2 \cdot N_{tu1}$ linear constraints for each control, to enforce the user-defined bounds at all control points. If bl_u and bu_u are the user-defined upper and lower bounds for control iu and $bl_{u'}$ and $bu_{u'}$ are the upper and lower bounds defined for the control derivative, then we impose

$$\begin{aligned} & bl_u \leq U_{iu,itx}^{it,0} + U_{iu,itx}^{it,1} + \sum_{j=1}^{N_q-1} U_{iu,itx}^{it,j+1} \leq bu_u \\ & \Delta t_{itx} \cdot bl_{u'} \leq U_{iu,itx}^{it,1} + \sum_{j=1}^{N_q-1} (j+1) U_{iu,itx}^{it,j+1} \leq \Delta t_{itx} \cdot bu_{u'} \qquad\qquad it = 1, 2, ..., N_{tu1}. \end{aligned} \tag{26}$$

If the simulation length is an optimization parameter (see Section 2.4), then the bounds on u' are multiplied by $p_{i_{tf}}$ to take into account the scaling of the time derivatives (Eq. 6).

These additional control constraints are arranged in the following order:

```
for each multiple shooting interval itx
  for each control iu
    for each subinterval it
      Control iu is in bounds
    end
    for each subinterval it
      Derivative of control iu is in bounds
    end
  end
end
```

3.3.2 Bounds inside the control intervals

For piece wise quadratic control parameterization we provide the option of having the control bounded within the whole interval($iboundU$=1) or having the control monotonic within the interval($iboundU$=2). For a quadratic polynomial of the form $U = U_0 + at + bt^2$ for t ϵ [0,1] and a lower bound of m and upper bound of M we assume

$$m < U_0 < M$$

and impose the constraints:

$$\begin{aligned} U_0 + a + b < M \\ U_0 + a + b > m \end{aligned} \tag{27}$$

In the first case($iboundU$=1), the area of feasibility for the parameters a and b on the a-b plane is as shown in Figure 4. We consider only the shaded area to keep the additional constraints linear. Therefore, we impose the following additional constraints:

$$\begin{aligned} a + U_0 < M \\ a + U_0 > m \end{aligned} \tag{28}$$

In the second case($iboundU$=2), the region in the a-b plane which ensures that the control is monotonic within the control interval is as shown in Figure 5. The shaded area guarantess that $\frac{\partial U}{\partial t}$ doesn't change sign in [0,1]. Therefore, we get the additional constraint:

$$a(a + 2b) > 0 \tag{29}$$

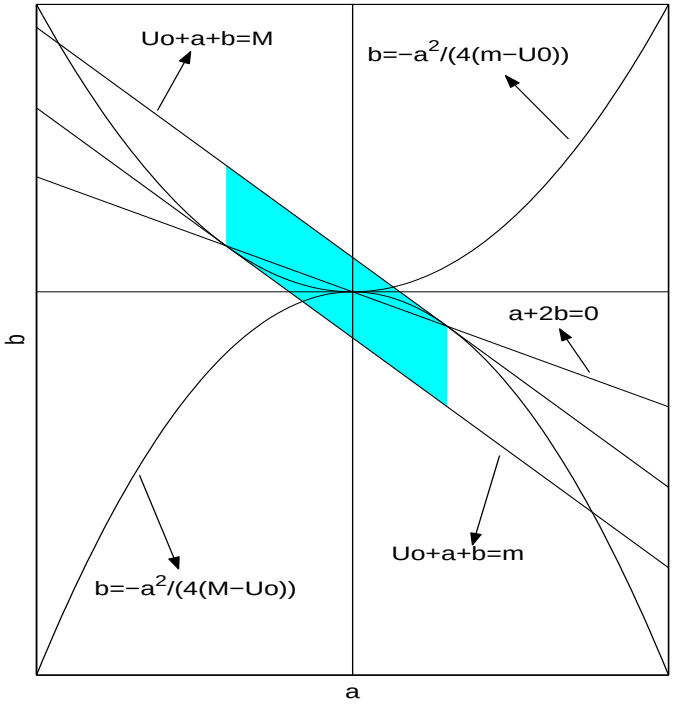

Figure 4: Feasibility region for control parameters

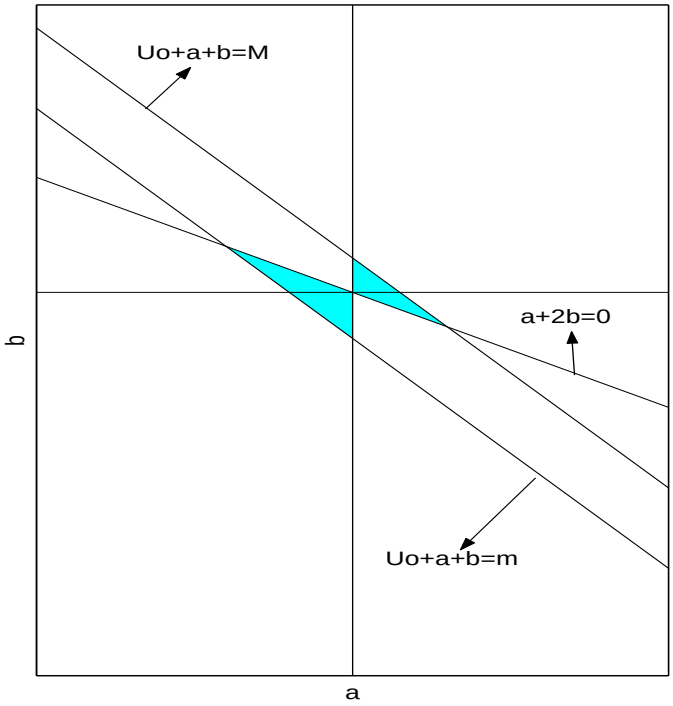

Figure 5: Region which ensures monotonicity of control within control interval

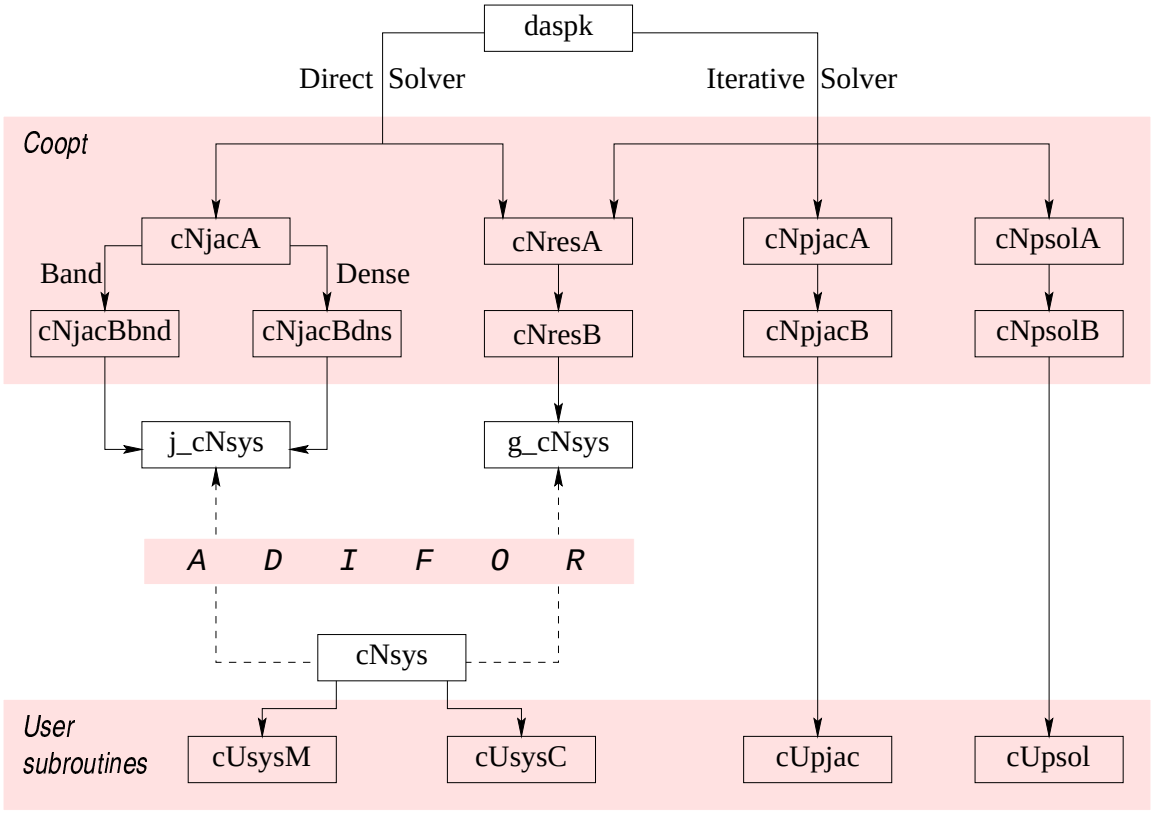

Figure 6: Subroutines called by DASPK3.0

3.4 Optimization Constraints

The complete set of constraints in the discretize optimal control problem (4) is then obtain by collecting the user-defined constraints (specified through the file `Constraints.f`), the state continuity constraints (Section 3.2.1), the control continuity constraints (Section 3.2.2), and the additional control constraints (Section 3.3). The number and order of the optimization constraints is presented in Table 7.

3.5 Solution of State and Sensitivity Equations

We obtain values and Jacobians of the nonlinear state continuity constraints of Section 3.2.1 by solving the following state and sensitivity equations:

$$\mathbf{F}(t, \mathbf{x}, \mathbf{x}', \mathbf{p}, \mathbf{u}(t), \mathbf{u}'(t)) = 0 \tag{30}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\mathbf{s} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\mathbf{s}' = 0 \tag{31}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\mathbf{s} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\mathbf{s}' + \frac{\partial \mathbf{F}}{\partial \mathbf{p}} = 0 \tag{32}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\mathbf{s} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\mathbf{s}' + \frac{\partial \mathbf{F}}{\partial \mathbf{U}} = 0. \tag{33}$$

The central observation behind the modified multiple shooting approach is that Eq. 31 has no forcing term. As a consequence, if $\mathbf{s}_j$ is the j-th column of the solution of (31) with initial condition $\mathbf{s}_j(t_i) = \mathbf{e}_j$ then any linear combination of these columns is also a solution of (31), 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*. Then, computation of the modified constraint $\hat{\mathbf{c}}$ requires only one sensitivity calculation. The

modified Jacobians $\hat{\mathbf{P}}$ and $\hat{\mathbf{R}}$ can be obtained with only $2 \cdot N_p$ and N_r sensitivity computations, respectively. Also, each block $\hat{\mathbf{U}}_{itx1,itx2}$ requires only $2 + N_{tu1}(N_q - 1)$ sensitivity computations. When the dimension of the state vector $\mathbf{x}$ is large, this approach will substantially reduce the number of sensitivity computations required by a conventional multiple shooting approach.

On the first multiple shooting interval ($itx = 1$), we have to solve $N_x + 1$ state equations and $(N_x + 1)\{1 + N_p + N_r + N_u[2 + N_{tu1}(N_q - 1)]\}$ sensitivity equations. On each of the following multiple shooting intervals ($itx > 1$) the number of equations to be solved is $(N_x + 1)\{1 + 1 + N_p + N_r + N_u[2 + N_{tu1}(N_q - 1)] + N_p + (itx - 1)N_u[2 + N_{tu1}(N_q - 1)]\}$. Collecting the states and all their sensitivities into a vector $\mathbf{v}$, we can split this vector into 7 parts. The columns in Table 8 have the following meanings:

1. Part in vector $\mathbf{v}$
2. Number of elements/equations
3. Initial content
4. Equation used
5. Content after integration
6. Relation in which the result is used

Notes

1. The additional variable corresponding to the cost function (Eq. 3) is inserted in the vector of model states at the position specified by the user through the flag $i_{cf} = idata(23)$. This option is provided to preserve the possible band structure of the model equations Jacobian. If the *banded Jacobian* option is not used, i_{cf} can have any value between 1 and $N_x + 1$.
2. If no explicit bounds are defined for the model states at $t = t_{N_{tx}+1} \equiv t_{max}$ 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 (specify $idata(22) = 1$). This not only leads to an optimization problem with fewer variables, but also results in N_x 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).
3. Although we consider additional control subintervals inside each multiple shooting interval, the control parameterization employed (see Section 3.1) assures $\mathcal{C}^1$ continuity of the controls inside each shooting interval and thus integration can be carried out without restarts. This can be seen best in Fig. 7 which shows that sensitivities of the control u with respect to the coefficients of the parameterization are also $\mathcal{C}^1$ continous. 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. However, this doesn't mean that these bounds are respected at all times. For problems in which the optimal control tends to be very close to some of the bounds, you will have to experiment with different combinations of N_{tx} and N_{tu1}, as well as different settings of the bounds $bl_{u'}$ and $bu_{u'}$ to keep the controls at all time within the feasible region.

During integration on a given shooting interval itx, DASPK3.0 requires computation of the residual of the state and sensitivity equations (30)-(33), as well as computation of the Jacobian with respect to states and their derivatives. The subroutine `sys` calls the user-defined routines `sysM` and `sysC` to evaluate the residual of the state equations at a given time t, for given values of the parameters $\mathbf{p}$, states $\mathbf{x}$, state derivatives $\mathbf{x}'$, controls $\mathbf{u}$, and control derivatives $\mathbf{u}'$; i.e.,

$$\mathbf{F} = \mathbf{F}(t, \mathbf{x}, \mathbf{x}', \mathbf{p}, \mathbf{u}, \mathbf{u}') \tag{34}$$

The subroutine `sys` is preprocessed through ADIFOR to generate the following two subroutines:

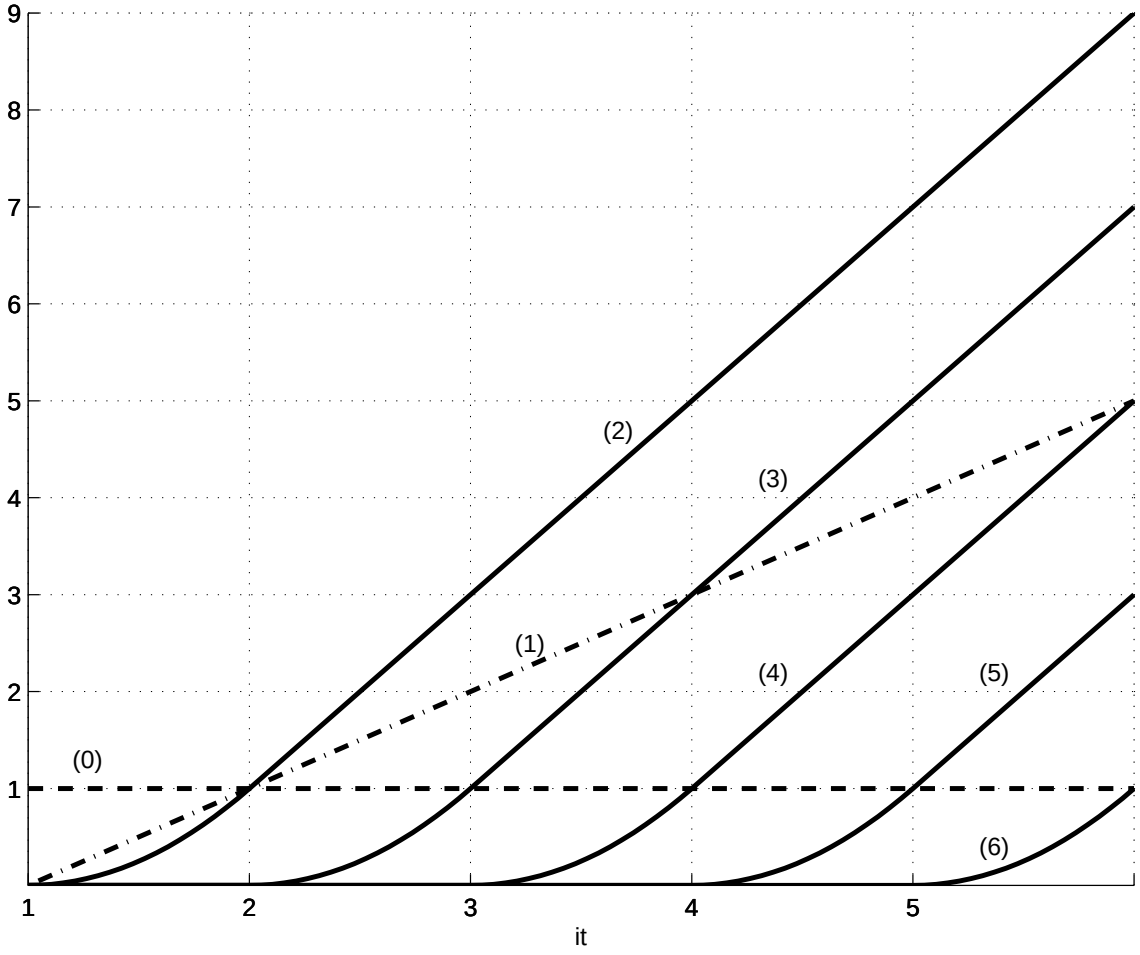

Figure 7: Sensitivities of the control with respect to coefficients of the parameterization (for $N_{tu1} = 5$, $N_q = 2$). (0): $\partial u/\partial U^{1,0}$ (1): $\partial u/\partial U^{1,1}$ (2): $\partial u/\partial U^{1,2}$ (3): $\partial u/\partial U^{2,2}$ (4): $\partial u/\partial U^{3,2}$ (5): $\partial u/\partial U^{4,2}$ (6): $\partial u/\partial U^{5,2}$

1. `g$sys` to evaluate

$$\bar{\mathbf{F}} = \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\bar{\mathbf{x}} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\bar{\mathbf{x}}' + \frac{\partial \mathbf{F}}{\partial \mathbf{p}}\bar{\mathbf{p}} + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}\bar{\mathbf{u}} + \frac{\partial \mathbf{F}}{\partial \mathbf{u}'}\bar{\mathbf{u}}' \tag{35}$$

for given values of the seed vectors $\bar{\mathbf{x}}$, $\bar{\mathbf{x}}'$, $\bar{\mathbf{p}}$, $\bar{\mathbf{u}}$, and $\bar{\mathbf{u}}'$.

2. `j$sys` to evaluate

$$\bar{\mathbf{J}} = \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\bar{\mathbf{X}} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}'}\bar{\mathbf{X}}' \tag{36}$$

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

3.5.1 Residual Computation

The equations that must be solved on a given shooting interval itx can be partitioned into seven parts, corresponding to the similar partition of the vector $\mathbf{v}$. We evaluate each part of the residual by repeatedly calling `g$sys` with convenient seed vectors. As an example, consider the fifth part of the residual. The subroutine `g$sys` must be called $N_u(2 + N_{tu1}(N_q - 1))$ times to evaluate the residual of Eq. 33 for each control parameter in the interval itx. Comparing Eqs. 33 and 35 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 $iu = 1, \cdots, N_u$, the initialization of the seed vectors $\bar{\mathbf{u}}$ and $\bar{\mathbf{u}}'$ follows from

$$u_{iu}(t) = \left[U^{1,0}_{iu,itx} + it \cdot U^{1,1}_{iu,itx} + \sum_{k=1}^{it} \sum_{j=1}^{N_q-1} [(j+1)(it-k)+1] U^{k,j+1}_{iu,itx} \right] + \left[U^{1,1}_{iu,itx} + \sum_{k=1}^{it} \sum_{j=1}^{N_q-1} (j+1) U^{k,j+1}_{iu,itx} \right] t^* + \sum_{j=1}^{N_q-1} U^{it+1,j+1}_{iu,itx} t^{*j+1}$$

and

$$\Delta t_{itx} \cdot u'(t) = \left[U^{1,1}_{iu,itx} + \sum_{k=1}^{it} \sum_{j=1}^{N_q-1} (j+1) U^{k,j+1}_{iu,itx} \right] + \sum_{j=1}^{N_q-1} U^{it+1,j+1}_{iu,itx} (j+1) t^{*j}$$

where we have used Eqs. 7 and 9.

Therefore, to compute the residual of the sensitivity equations with respect to the control parameter $U^{1,0}_{iu,itx}$ the only nonzero component of $\bar{\mathbf{u}}$ is $\bar{u}_{iu} = 1$, while $\bar{\mathbf{u}}' = \mathbf{0}$.

To compute the residual of the sensitivity equations with respect to $U^{1,1}_{iu,itx}$, we set $\bar{u}_{iu} = it + t^*$ and $\bar{u}'_{iu} = 1/\Delta t_{itx}$.

To compute the residual of the sensitivity equations with respect to $U^{k,j+1}_{iu,itx}$, $k = 1, ..., it$, $j = 1, ..., N_q - 1$, we set $\bar{u}_{iu} = (j+1)(it-k) + 1 + (j+1)t^*$ and $\bar{u}'_{iu} = (j+1)/\Delta t_{itx}$.

Finally, to compute the residual of the sensitivity equations with respect to $U^{it+1,j+1}_{iu,itx}$, $j = 1, ..., N_q - 1$, we set $\bar{u}_{iu} = t^{*j+1}$ and $\bar{u}'_{iu} = (j+1)t^{*j}/\Delta t_{itx}$.

The residual computation is implemented in the subroutine `cNresB` (file `src/Dynamics_subs.f`) and is transparent to the user.

3.5.2 Jacobian Computation

If a direct linear system solution method is selected, the DASPK3.0 software also requires Jacobians of Eq. 30 with respect to $\mathbf{x}$ and $\mathbf{x}'$. Note that these Jacobians are the same for the state and sensitivity equations. They are computed by the ADIFOR-generated subroutine, `j$sys` (obtained starting from the user-defined subroutine `sys`). Depending on the *banded Jacobian* user option selected through $idata(32)$, the subroutine `j$sys` is called with different arguments to obtain either a dense or a banded Jacobian.

If $idata(32) = 0$, subroutine `cNjacBdns` calls `j$sys` with the following seed matrices:

$$\begin{aligned} \bar{\mathbf{X}} &= \mathbf{I}_{N_x+1} \\ \bar{\mathbf{X}}' &= cj \cdot \mathbf{I}_{N_x+1} \end{aligned} \tag{37}$$

If $idata(32) = 1$, subroutine `cNjacBbnd` calls `j$sys` with the following seed matrices:

$$\bar{\mathbf{X}} = \begin{bmatrix} \mathbf{I}_w \\ \mathbf{0} \end{bmatrix}$$
$$\bar{\mathbf{X}}' = cj \cdot \begin{bmatrix} \mathbf{I}_w \\ \mathbf{0} \end{bmatrix} \tag{38}$$

where $w = MU + ML + 1$ is the total Jacobian bandwidth.

3.5.3 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. This is done by setting $iflag(5) = 0$ and specifying $rdata(1) = rtol$ and $rdata(2) = atol$. These values will then be used for all variables, both state and sensitivity.
- If $iflag(5) = 1$, then the state and sensitivity equations will be solved with different tolerances. $rdata(1) = rtol_{states}$ and $rdata(2) = atol_{states}$ will be used for all state variables and $rdata(3) = rtol_{sensitivity}$ and $rdata(4) = atol_{sensitivity}$ will be used for all sensitivity variables.
- If $iflag(5) = 2$ then the user must specify, through the subroutine `cUtol` (see Section 2.1.1), *rtol* and *atol* values for each of the state variables and for the additional 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 sensitivies with respect to parameters $\mathbf{p}$ or $\mathbf{r}$, 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 8), 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 $\mathbf{s} = \frac{\partial x}{\partial x_0}\xi$, 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.

Table 6: Notation

Symbol	Description
N_x	Dimension of the state vector
N_p	Number of parameters in the model
N_r	Number of 'initial' parameters
N_u	Number of control functions
N_q	Order of the control approximation polynomial
N_{tx}	Number of multiple shooting intervals
N_{tu1}	Number of control subintervals per shooting interval
N_{uu}	Total number of control parameters
itx	Current shooting interval
it	Current control subinterval
iu	Current control
$\mathbf{p}$	Vector of model parameters
$\mathbf{r}$	Vector of 'initial' parameters (used to parameterize the states at $t = t_1$)
$\mathbf{x}$	Solution of state equations
$\mathbf{X}_{itx}$	Vector of states at the begining of the shooting interval itx
$\mathbf{X}$	Vector of discretized states
$\mathbf{u}$	Vector of controls
$\mathbf{U}$	Vector of control parameters
$\mathbf{U}_{itx}$	Vector of control parameters on the interval itx
$U^{it,0}_{iu,itx}$	Value of control iu on interval itx at subdivision it
$U^{it,1}_{iu,itx}$	Value of derivative of control iu on interval itx at subdivision it
$U^{it,j+1}_{iu,itx}$	Coefficients of the polynomial approximation of control iu on interval itx
$\mathbf{C}_{itx}$	Vector of state continuity constraints at the end of the shooting interval itx
$\mathbf{C}$	Vector of state continuity constraints
$\hat{\mathbf{C}}_{itx}$	Vector of modified state continuity constraints at the end of the shooting interval itx
$\hat{\mathbf{C}}$	Vector of modified state continuity constraints
$\mathbf{J}$	Jacobian of the state continuity constraints with respect to $\mathbf{p}$, $\mathbf{r}$, $\mathbf{X}$, and $\mathbf{U}$
$\mathbf{J_P}$	Jacobian of the state continuity constraints with respect to the parameters $\mathbf{p}$
$\mathbf{J_r}$	Jacobian of the state continuity constraints with respect to the 'initial' parameters $\mathbf{r}$
$\mathbf{J_X}$	Jacobian of the state continuity constraints with respect to the discretized states $\mathbf{X}$
$\mathbf{J_U}$	Jacobian of the state continuity constraints with respect to the control parameters $\mathbf{U}$
$\hat{\mathbf{J}}$	Modified $\mathbf{J}$
$\hat{\mathbf{P}}$	Modified $\mathbf{J_p}$
$\hat{\mathbf{R}}$	Modified $\mathbf{J_r}$
$\hat{\mathbf{U}}$	Modified $\mathbf{J_U}$
$C^1_{iu,itx}$	Continuity constraint on control iu at itx
$C^2_{iu,itx}$	Continuity constraint on derivative of control iu at itx

Table 7: Optimization Constraints

		Dimension	Description	User
Nonlinear constraints	1	N_{fnprxf}	Nonlinear constraints on $\mathbf{p}$, $\mathbf{r}$ and $\mathbf{X}_f$	yes
	2	$N_{fnxu}(N_{tx} - i_{xfFree})$	Nonlinear constraints on $\mathbf{p}$, $\mathbf{x}$, and $\mathbf{u}$	yes
	3	$N_{fnu}(N_{tx} * N_{tu1} + 1 - i_{xfFree})$	Nonlinear constraints on $\mathbf{p}$ and $\mathbf{u}$	yes
	4	N_{fx}	Nonlinear state continuity constraints	no
Linear constraints	5	N_{flprxf}	Linear constraints on $\mathbf{p}$, $\mathbf{r}$ and $\mathbf{X}_f$	yes
	6	$N_{flxu}(N_{tx} - i_{xfFree})$	Linear constraints on $\mathbf{p}$, $\mathbf{x}$, and $\mathbf{u}$	yes
	7	$N_{flu}(N_{tx} * N_{tu1} + 1 - i_{xfFree})$	Linear constraints on $\mathbf{p}$ and $\mathbf{u}$	yes
	8	N_{fu}	Linear control continuity constraints	no
	9	N_{fu1}	Linear additional constraints on controls	no

Table 8: State and sensitivity equations

	Dimension	Initial	Eq.	Final	Result
			$itx = 1$		
1	$N_x + 1$	$\mathbf{X}_1$	30	$\mathbf{x}(t_2)$	$\mathbf{C}_1 = \mathbf{X}_2 - \mathbf{x}(t_2)$
2	$N_x + 1$	$\mathbf{0}$	31	$\mathbf{0}$	never used
3	$N_p(N_x + 1)$	$\mathbf{0}$	32	$\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{p}}$	$\hat{\mathbf{P}}_1 = -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{p}}$
4	$N_r(N_x + 1)$	$\mathbf{x_r}(t_1)$	31	$\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{r}}$	$\hat{\mathbf{R}}_1 = -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{r}}$
5	(†)	$\mathbf{0}$	33	$\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{U}_1}$	$\hat{\mathbf{U}}_{1,1} = -\frac{\partial \mathbf{x}(t_2)}{\partial \mathbf{U}_1}$
			$itx > 1$		
1	$N_x + 1$	$\mathbf{X}_{itx}$	30	$\mathbf{x}(t_{itx+1})$	$\mathbf{C}_{itx} = \mathbf{X}_{itx+1} - \mathbf{x}(t_{itx+1})$
2	$N_x + 1$	$\hat{\mathbf{C}}_{itx-1}$	31	$\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{C}}_{itx-1}$	$\hat{\mathbf{C}}_{itx} = \mathbf{C}_{itx} + \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{C}}_{itx-1}$
3	$N_p(N_x + 1)$	$\mathbf{0}$	32	$\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{p}}$	
4	$N_r(N_x + 1)$	$-\hat{\mathbf{R}}_{itx-1}$	31	$-\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{R}}_{itx-1}$	$\hat{\mathbf{R}}_{itx} = \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{R}}_{itx-1}$
5	(†)	$\mathbf{0}$	33	$\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{U}_{itx}}$	$\hat{\mathbf{U}}_{itx,itx} = -\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{U}_{itx}}$
6	$N_p(N_x + 1)$	$-\hat{\mathbf{P}}_{itx-1}$	31	$-\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{P}}_{itx-1}$	$\hat{\mathbf{P}}_{itx} = -\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{p}} + \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{P}}_{itx-1}$
7	(‡)	$-\hat{\mathbf{U}}_{itx-1,1}$ $-\hat{\mathbf{U}}_{itx-1,2}$ $\vdots$ $-\hat{\mathbf{U}}_{itx-1,itx-1}$	31	$-\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{U}}_{itx-1,1}$ $-\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{U}}_{itx-1,2}$ $\vdots$ $-\frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{U}}_{itx-1,itx-1}$	$\hat{\mathbf{U}}_{itx,1} = \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{U}}_{itx-1,1}$ $\hat{\mathbf{U}}_{itx,2} = \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{U}}_{itx-1,2}$ $\vdots$ $\hat{\mathbf{U}}_{itx,itx-1} = \frac{\partial \mathbf{x}(t_{itx+1})}{\partial \mathbf{X}_{itx}} \hat{\mathbf{U}}_{itx-1,itx-1}$

(†) $N_u(2 + N_{tu1}(N_q - 1))(N_x + 1)$

(‡) $(itx - 1)N_u(2 + N_{tu1}(N_q - 1))(N_x + 1)$

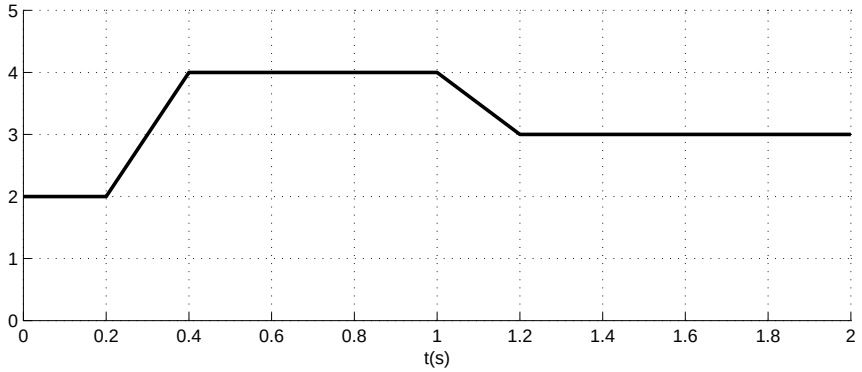

Figure 8: Function $\tau(t)$

4 1-D Heat Problem

To illustrate the use of COOPT consider the heat equation

$$\frac{\partial x}{\partial t} = \frac{\partial^2 x}{\partial y^2} \tag{39}$$

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 use the method of lines (MOL) by discretizing the spatial derivative 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 $x_j(t) = x(t, y_j)$

$$\begin{aligned} & x_1 - x(t,0) = 0 \\ & x_j' - \frac{x_{j-1} - 2x_j + x_{j+1}}{(\Delta y)^2} = 0, \qquad j = 2, ..., N-1 \\ & x_N - x(t,1) = 0 \end{aligned} \tag{40}$$

Imposing $x(t,0) = x(t,1) = u$ as the control, we can formulate an optimal control problem as 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. 8.

The user files for this problem can be found in the directory `coopt/example`. The significant values in the $idim$, $iflag$, and $idata$ arrays (subroutine `cUinit`) are

Entry	Name	Value	Comment
		idim	
3	N_{xd}	9	The number of differential variables is $N-2$
4	N_{xa}	2	There are 2 algebraic variables
5	N_u	1	There is one control function
6	N_q	3	Cubic approximation of the control
7	N_{tx}	10	There are 10 shooting intervals
8	N_{tu1}	1	There is 1 control interval per shooting interval
		iflag	
3	i_{xf}	1	The final states are not optimization variables
8	i_{InitX}	1	Initial condition computation
18	i_{tol}	1	Use different tolerances for state and sensitivity equations
26	i_{err}	1	The sensitivity variables are not included in the error test
		idata	
1	i_{cf}	12	The additional variable is the last one $(N+1)$
5	i_{dae}	1	The DAE is index 1

Indices of the two algebraic variables are set through the array *ivars* in the subroutine `cUvars` as $ivars(1) = 1$ and $ivars(2) = 11$.

The *rdata* array (defined in subroutine `cUinit`) contains

Entry	Name	Value	Comment
1	rtoleq	10^{-6}	*rtol* for state equations
2	atoleq	10^{-5}	*atol* for state equations
3	rtolsen	10^{-4}	*rtol* for sensitivity equations
4	atolsen	10^{-3}	*atol* for sensitivity equations

As initial guess (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 $blU0(1) = buU0(1) = 2.0$ (subroutine `cUbounds`).

The state equations are specified in the file `sysM.f`:

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

The right side of the additional equation (3) is specified in the file `sysC.f`:

```
      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
c
      fc = ww*(X(6)-tt)*(X(6)-tt)
```

Optimization parameters are passed to SNOPT through the specification file `example.spc`:

```
Begin example

Derivative level            3
Feasibility Tolerance       1.0d-4
Function Precision          1.0d-4
Linesearch Tolerance        0.9
Major iterations            200
Major Print level           1
Minor print level           0
```

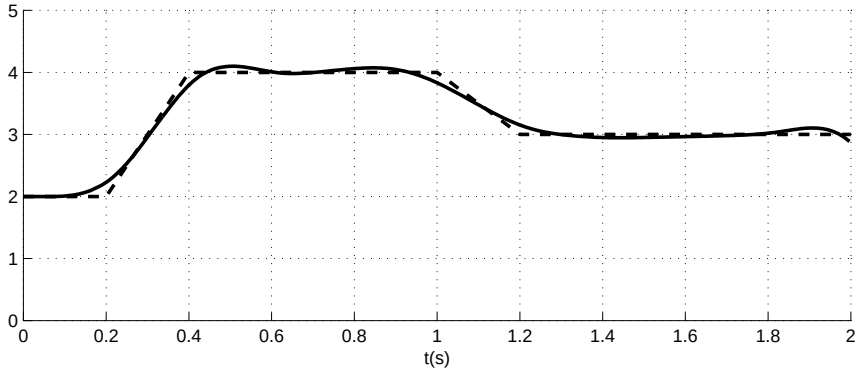

Figure 9: Optimization results for the heat problem. Solid line $x_6^*(t)$. Dashed line $\tau(t)$

```
Minor iterations            1000
Minimize
Optimality Tolerance        5.0d-3
Print frequency             1
Row Tolerance               1.0d-5
Scale option                0
Summary frequency           1
Verify level               -1
Solution                    yes

End example
```

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 9 shows the temperature x_6^* obtained with the optimal control as compared to the targeted function $\tau(t)$.

4.1 Results with quadratic control parameterization

In this section, we illustrate the results obtained using a quadratic control parameterization and with various bounding schemes.

Figure 10 shows for different values of *iboundU*, a plot of control and $x_6(t)$ against time when there are no bounds on the control (ie $bu_u = \infty$ and $bl_u = -\infty$). When *iboundU*=0, the control is bounded only at the control points and when *iboundU*=1, control is bounded inside the control intervals. But, since there are no bounds, there are no additional control constraints in both cases and the same solution is obtained. When *iboundU*=2, the control is monotonic within the control intervals and therefore a different solution is obtained as seen in Figure 10.

Figure 11 shows for different values of *iboundU*, a plot of control and $x_6(t)$ against time when $bu_u = 3.5$ and $bl_u = -\infty$. This time, we get different solutions for *iboundU*=0 and *iboundU*=1 because of the bounds.

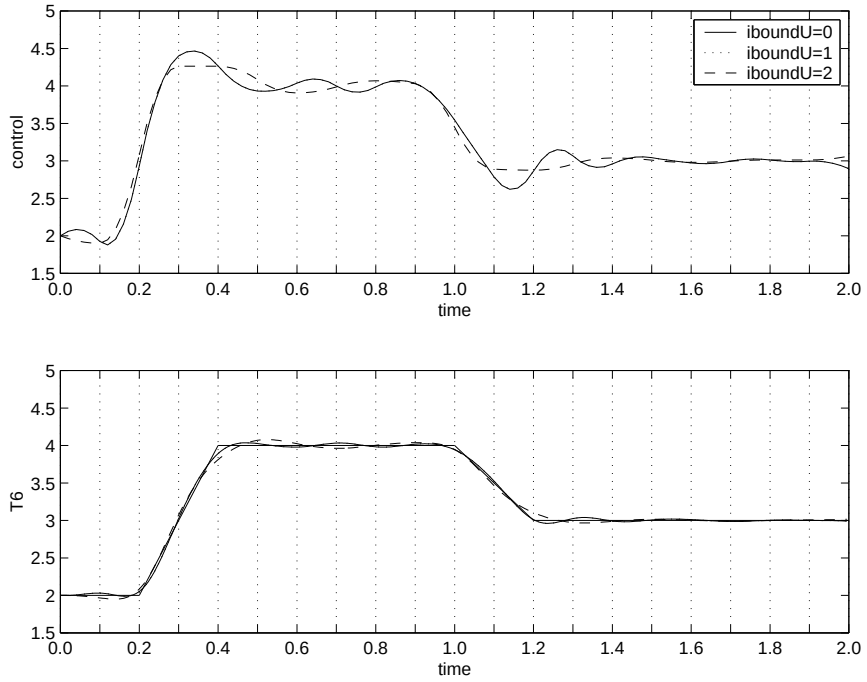

Figure 10: Plot of control and $x_6(t)$ when there are no bounds

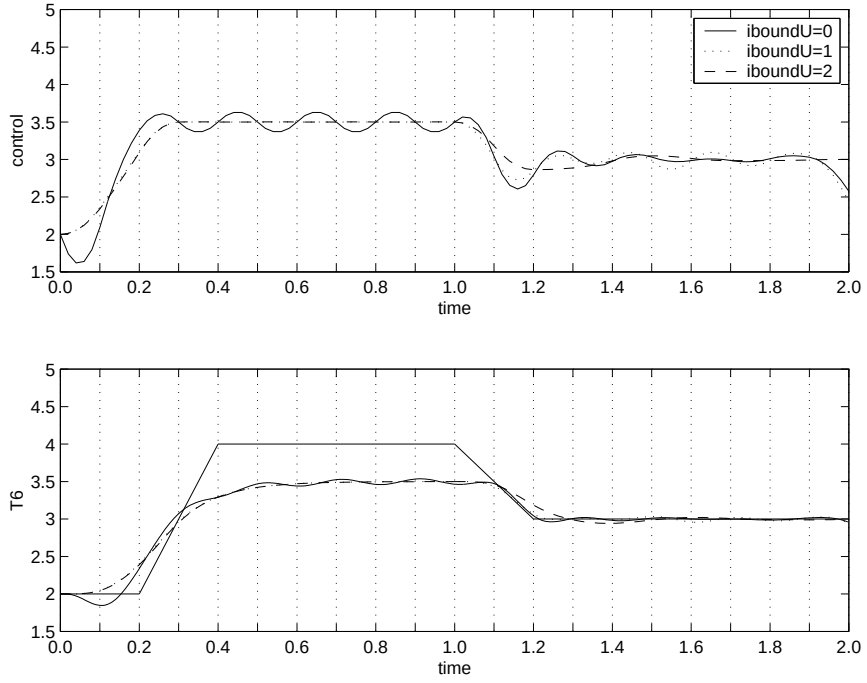

Figure 11: Plot of control and $x_6(t)$ when $bu_u = 3.5$ and $bl_u = -\infty$

References

[1] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, 1998.

[2] 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.

[3] 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.

[4] C. Bischof, A. Carle, P. Hovland, P. Khademi, and A. Mauer, *ADIFOR 2.0 Users' Guide*, MCS Division, Argonne National Laboratory, Technical Memorandum No. 192, June 1998.

[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*, submitted, 1998.

[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] P. E. Gill, W. Murray, and M. A. Saunders, *User's Guide for SNOPT 5.3: A Fortran Package for Large-Scale Nonlinear Programming*, Department of Mathematics, University of California, San Diego, La Jolla, CA, 1998.

[9] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London and New York, 1981. ISBN 0-12-283952-8.

[10] 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.

[11] 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.

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