Rainer Schöpf  Peter Deuflhard

OCCAL
A mixed symbolic–numeric
Optimal Control CALculator

Preprint SC 91–13 (December 1991)
OCCAL
A mixed symbolic–numeric
Optimal Control CALculator

Abstract

The numerical solution of optimal control problems by indirect methods (such as multiple shooting or collocation) requires a considerable amount of analytic calculation to establish a numerically tractable system. These analytic calculations, though being rather tedious in realistic examples, are nowadays mostly still done by hand—and thus prone to calculation errors. The paper aims at automating this analytic processing to a reasonable extent by means of a modern symbolic manipulation language (here: REDUCE). In its present stage of development the package OCCAL (mnemotechnically for Optimal Control CALculator) permits an interactive use, covering tasks like automatic determination of control and, in case of a singular control, of its order. In simpler problems, the present version of OCCAL automatically produces the full subroutine input for a MULTiple shooting code (MULCON) with adaptive numerical CONTinuation. In more complicated problems where singular sub-arcs may occur or where the sequence of sub-arcs of the optimal trajectory is unclear OCCAL is a significant help in reducing analytic pre-processing. Examples illustrate the performance of OCCAL/MULCON.
## Contents

1 Introduction 3

2 A Short Introduction to Optimal Control Problems 4

3 Application of symbolic computation to optimal control problems 8

4 Examples 12

5 Conclusion 15

Appendix A An OCCAL sample session 17

Appendix B Files for the reentry problem 25
   B.1 Problem description 25
   B.2 REDUCE input file 26

Appendix C Input file format for the pre-processor 28
1 Introduction

Optimal Control problems appear in a wide variety of important scientific and technical applications. In spite of the rather high standards of numerical algorithms for such problems \([1, 2, 3, 4, 8]\) tackling such a problem is still a complicated task. The reason for that is that certain steps in the analytical preparation of the calculation, which are simple in principle, may be very elaborate and can lead to rather complex expressions. Implementation of these into numerical code by hand is tiresome and error-prone.

The purpose of our system OCCAL to be presented here is to overcome this tedious preparation phase. We intend to solve as many of the arising problems as possible.

To this end we employ a combined symbolic system that symbolically analyzes the given problem, using the Computer Algebra System REDUCE. In passing, this approach additionally permits to generate optimized numerical subprograms for use with existing numerical solvers for boundary value problems. These are employed in the second step to obtain numerical solutions.

Symbolic computation (as most computations) is often applied to tasks that are simple in principle, but tedious to carry out by hand. Consequently, often only some well defined steps in a calculation are done with the help of a Computer Algebra System. The results are then treated more or less by hand.

It should be stressed at this point that the symbolic processor in OCCAL is not only used for the tasks mentioned as “tedious, but simple in principle”. Nowadays, advanced symbolic techniques are available, e.g., for the solution of systems of nonlinear equations or for computations with inequalities. This can be used, for example, to determine the sign of a function of several variables over a given range.

The need for these advanced methods within OCCAL has not only led to their improvement and adaptation to the problems at hand, but also to their inclusion into the recently released new version of the Computer Algebra System REDUCE. Therefore these techniques are now available to a much greater community than before, and hopefully also for a much wider area of applications.

In Section 2 below the typical structure of optimal control problems is presented—mainly to fix notations and to elucidate the kind of analytical work involved in the pre-numerical stage. On this basis those tasks that can be automated are listed in detail in Section 3— together with the corresponding details of symbolic devices. Finally in Section 4 the performance of OCCAL in a few well-known examples is documented.

Of course, the full use of such a software system can only be estimated by actually trying it!
2 A Short Introduction to Optimal Control Problems

Consider a system that is described by a number of time-dependent state variables $y$. Let $y \in K \subseteq C^1$, where $K$ is a suitable class of continuously differentiable functions, and $y: [a, b] \rightarrow \mathbb{R}^n$. This system is controlled by certain control variables $u \in C^0$ piecewise, $u: [a, b] \rightarrow \mathbb{R}^k$, and governed by the following system of differential equations

$$y' = f(t, y, u),$$

the with separated boundary conditions $y(a) = y_a$ and

$$r(b, y(b)) = 0, \quad r: \mathbb{R}^{n+1} \rightarrow \mathbb{R}^p, \quad p \leq n.$$ (2)

The task is to minimize the integral

$$I[u] := \int_a^b f_0(y, u, t) dt$$ (3)

over $K$. Similar to the calculus of variations there is the so-called Hamiltonian

$$H(t, y, \lambda, u) := \sum_{i=1}^n \lambda_i f_i(t, y, u) + \lambda_0 f_0(t, y, u),$$ (4)

where $\lambda(t) = (\lambda_1(t), \ldots, \lambda_n(t))$ denotes the adjoint variables.

The minimum principle of Pontrjagin states then that a necessary condition for $(y_0, u_0)$ being a solution of the optimal control problem is

$$H(t, y_0, \lambda, u_0) = \min_v H(t, y_0, \lambda, v)$$ (5)

with $v$ fulfilling the conditions. Calculating the first variation leads to the so-called canonical equations

$$y' = H_{\lambda_i} = f_i(t, y, u), \quad i = 1, \ldots, n,$$

$$\lambda_i' = -H_{y_i} = -\frac{\partial f_0}{\partial y_i}(t, y, u) - \sum_{j=1}^n \lambda_j \frac{\partial f_j}{\partial y_i}(t, y, u), \quad i = 1, \ldots, n.$$ (6)

We number the control variables $u_i$ in such a way that the first $m$ components (denoted by $u^{(n)}$) appear nonlinearly in $f$, while the others (denoted by $u^{(l)}$) appear only linearly, so that $f$ splits in the form

$$f(t, y, u) =: g(t, y, u^{(n)}) + h(t, y, u^{(n)})u^{(l)},$$

$$u^{(n)} \in \mathbb{R}^n, u^{(l)} \in \mathbb{R}^q, \quad m + q = k.$$ (7)

This splitting carries over to the Hamiltonian $H$ as well. Since $u^{(l)}$ appears only linearly in $f$ and $H$ (cf. (7)) minimization of $H$ can be done in two steps:
1. Determination of $u^{(n)}$ from 

\[ H_{u^{(n)}} = 0 \]

\[ H_{u^{(n)}u^{(n)}} \text{ positive (semi-)definite} \]  

(Here $H_{u^{(n)}}$ denotes the Fréchet derivatives.) One obtains an analytic expression $u^{(n)} = u^{(n)}(t, y, \lambda)$ which can be substituted into the original expressions (6). Thus $u^{(n)}$ essentially drops out of the set of variables.

2. Determination of $u^{(l)}$:

After substitution of $u^{(n)}$ the Hamiltonian can be written as

\[ H(t, y, \lambda, u^{(n)}(t, y, \lambda), u^{(l)}) = H_0(t, y, \lambda) + \sum_{i=1}^{q} S_i(t, y, \lambda) u_{m+i} \]  

Obviously we have

\[ S_i(t, y, \lambda) = H_{u^{(l)}}(t, y, \lambda, u^{(n)}(t, y, \lambda), u^{(l)}) \]

with $i = 1, \ldots, q$. The condition (5) is then equivalent to

\[ S_i(t, y_0, \lambda) \cdot u_0(t) = \min_u S_i(t, y, \lambda) \cdot v(t), \]

which is a linear optimization problem. It is therefore necessary that the control variables $u^{(l)}$ are subject to control constraints usually of the form

\[ \alpha_i \leq u_i \leq \beta_i, \quad i = m+1, \ldots, k \]

\[ \alpha_i, \beta_i \in \mathbb{R}. \]  

The further procedure depends on which one of the following two cases holds:

(a) $S_i \neq 0$, i.e. $S_i$ has only isolated zeros. This is the case of a bang-bang control. From the minimum principle we deduce

\[ u_{m+i} = \begin{cases} \alpha_{m+i} & \text{for } S_i > 0 \\ \beta_{m+i} & \text{for } S_i < 0 \end{cases} \]  

The $S_i$ are called switching functions. A change of sign can only occur in the switching points $\tau^i$:

\[ S_i(\tau^i, y(\tau^i), \lambda(\tau^i)) = 0. \]

(b) $S_i \equiv 0$ within a non-empty interval. This is the so-called singular case. Let us assume, for simplicity, that $q = 1$, i.e. there is only one
linear control $u$, and that there is only one such interval. Define the
following functions recursively by

$$S^{(0)}(t, y, \lambda) := S(t, y, \lambda)$$

and

$$S^{(k+1)}(t, y, \lambda) := \frac{dS^{(k)}(t, y, \lambda)}{dt}$$

$$= S_y^k \cdot \dot{y} + S_{\lambda}^k \cdot \dot{\lambda} + S_t^k$$

(16)

Define $m$ to be the smallest index such that $\partial S^{(m)}/\partial u \neq 0$. Then
theory [5, 6, 10] shows that $m = 2r$ is an even number. $r$ is called order
of the singular control.

Since $u^{(0)}$ appears only linearly in $H_y$ and $H_{\lambda}$ it follows that $S^{(m)}$ is of
the form

$$S^{(m)}(t, y, \lambda) = A(t, y, \lambda) + B(t, y, \lambda) \cdot u(t).$$

(17)

For $B(t, y, \lambda) \neq 0$ the singular control can be determined to be

$$u_0(t, y, \lambda) = -\frac{A(t, y, \lambda)}{B(t, y, \lambda)}$$

(18)

In certain cases $u_0$ does not depend on the adjoint variables $\lambda$; this is
called a feedback control. For an optimal trajectory $y(t)$ and an optimal
control $u_0(t)$ of order $k = 2r$ holds the generalized Legendre-Clebsch
condition [10]

$$0 \leq (\frac{d}{dt})^r \cdot (d^{2r}/d^2u) = (-1)^r \cdot B(t, y(t), \lambda(t)),$$

(19)

In the case of a feedback control a strict inequality holds.

Finally we can determine the boundary condition for the $\lambda_i$ as

$$\lambda_i|_b = \Phi_{y_i(t)} + \nu^T r_{y_i(t)},$$

(20)

where the $\nu$ are Lagrange multipliers.

**Numerical solution by means of Multiple Shooting.**

After $u^{(n)}$ has been substituted back into the canonical equations (6) we can
proceed to a numerical solution of the resulting boundary value problem. If
there is a linear control $u^{(0)}$ we have to take the structure of the optimal control
problem into account, especially the number $m_*$ and position of the switching
points $\tau_j$ defined by the equations

$$S_i(\tau_j, y(\tau_j), \lambda(\tau_j)) = 0, \quad j = 1, \ldots, m_*$$

(21)
Since the position of these points is in general not known beforehand, we introduce them as additional parameters
\[ \tau_1 < \tau_2 < \ldots < \tau_{m_s}, \]
and transform every sub-interval into an interval of the same length, i.e.,
\[
[a, \tau_1] \rightarrow [0, 1]: \tilde{t} := \frac{t - a}{\tau_1 - a} \\
[\tau_1, \tau_2] \rightarrow [1, 2]: \tilde{t} := \frac{t - \tau_1}{\tau_2 - \tau_1} + 1 \\
\vdots
\]
\[
[\tau_{m_s}, b] \rightarrow [m_s, m_s + 1]: \tilde{t} := \frac{t - \tau_{m_s}}{b - \tau_{m_s}} + m_s
\]
\[ y(t) \rightarrow \bar{y}(\tilde{t}), \quad \lambda(t) \rightarrow \bar{\lambda}(\tilde{t}) \]

This leads to extra factors in the right hand side of the differential equations which then additionally depend on one parameter (for the border intervals) or on two parameters (for the intermediate intervals). The parameters \( \tau_1, \ldots, \tau_{m_s} \) are determined by the inner point conditions
\[ S_i(\tau_j, \bar{y}(\tau_j), \bar{\lambda}(\tau_j)) = 0, \quad j = 1, \ldots, m_s. \]

It may happen that the switching structure changes during the numerical iteration. In that case the whole procedure has to be restarted with the new structure taken into account.

Thus the problem has been reduced to a parameter dependent boundary value problem of the general form:
\[
y' = f(y; p) \quad \text{(24)}
\]
\[
r(y(a), \ldots, y(b); p) = 0. \quad \text{(25)}
\]

The associated multiple shooting Jacobian then has the general block structure
\[
\begin{bmatrix}
G_1 & -I & \cdots & P_1 \\
\vdots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots \\
\vdots & \ddots & G_{m-1} & -I & P_{m-1} \\
R_1 & R_{m-1} & R_m & P_m
\end{bmatrix}
\]  
\[ \text{(26)} \]
in terms of the sub-matrices

\[ G_j := W(t_{j+1}, t_j)_{y(t|x_j,p)} \]

\[ R_j := \frac{\partial r}{\partial y(t_j)}_{y(t|x_j,p)} \]

\[ P_j := P(t_{j+1}, t_j)_{y(t|x_j,p)} \]

\[ P_m := \frac{\partial r}{\partial p}_{y(t|x_j,p)}. \]

Here the matrices \( W(t_{j+1}, t_j) \) are the Wronskian matrices

\[ \dot{W}(t, t_j) = \frac{\partial y(t)}{\partial y(t_j)} \]

which are the solutions of the variational equation

\[ \frac{dW(t, t_j)}{dt} = f_y(y(t|x_j,p); p)W(t, t_j), \quad W(t_j, t_j) = I, \]

and the sensitivity matrices \( P_j = P(t_{j+1}, t_j) \) are the solutions of the generalized variational equation

\[ \frac{dP(t, t_j)}{dt} = f_y(y(t|x_j,p); p)P(t, t_j) + f_p(y(t|x_j,p); p) \]

with initial value

\[ P(t_j, t_j) = 0. \]

### 3 Application of symbolic computation to optimal control problems

Summarizing the calculation steps outlined in the previous section we have the following sub-tasks

- Calculation of Hamiltonian.
- Identification of nonlinear versus linear control.
- Determination of nonlinear control by solving the system (8).
- Substitution into differential equations (6).
- Determination of linear control.
• In the case of bang-bang control: determination of the switching points.

• In the case of singular control: determination of the order of the control and test of the generalized Legendre-Clebsch condition.

• Generation of boundary conditions for the adjoint variables.

• Determination of the variational equations.

• Code generation (with optional optimization) for the differential equations, the boundary conditions, and the variational equations.

Some of these steps can easily be executed by any Computer Algebra System. Examples of this type of tasks are the computation of the Hamiltonian (4), of the canonical equations (6), of the derivatives $H_{u(n)}$ and $H_{u(n)}(n)$, or the substitution of the solution $u^{(n)}$ into the Hamiltonian.

The OCCAL system, however, does much more than that. We head at accomplishing all the aforementioned tasks automatically, and have already succeeded to a great extent. Let us first give a short overview of the operation of our system.

The first thing for the user to do is to formulate his problem in a form that is suitable for processing with OCCAL. Since the symbolic program is written in REDUCE, input must be given in REDUCE's Algol-like syntax. Even though this syntax is not too difficult to learn and understand, it is an unnecessary burden to the user who is not necessarily familiar with this type of program.

Therefore we decided to allow the input to be in a very natural, user-friendly form, with automatic translation into REDUCE syntax. To this end we developed a pre-processor, using the utility perl [12]. We chose perl due to its wide-spread availability and excellent string manipulation facilities that allowed us to implement the program in about one day's time.

The basic idea is to provide an easy-to-understand input file format, with keywords to identify the different sections, and some freedom for the user to choose a formulation that is most suitable to his or her own way of thinking. Furthermore, we want to keep in mind the possibility of a future system where this input file is only an intermediate step in a much more sophisticated system with, e.g., a graphical user interface. A description of the input file format is given in appendix Appendix C.

This pre-processing step does already some simple consistency checks: constants, parameters and variables may be defined only once, differential equations may only be given for already defined dynamic variables, and so on. If there was no error, it outputs a complete definition of the problem in REDUCE syntax to a file, taking care of arranging the statements in the correct order. This file is then read by the REDUCE program and processed symbolically.

It should be stressed that this output is well-readable: the experienced user may want to intervene at that point instead of changing the input file.
In the second step, the symbolic program tackles the tasks outlined above. First the Hamiltonian (4) is calculated. This object is analyzed and the nonlinear and linear controls $u^{(n)}$ and $u^{(0)}$ identified.

The determination of the nonlinear control is done in three steps, of which only the first one is straightforward in general:

- **Calculation of $H_{u^{(n)}}$.** This involves only a series of symbolic differentiations, a standard problem for any Computer Algebra System.

- **Solution of the system of equations.** Here advanced symbolic techniques are used: algebraic equations are solved using the methods of Groebner bases, transcendental equations may be reduced to algebraic ones by a suitable change of variables. For example, the transcendental equation
  \[ A \sin u + B \cos u = C \]  
  (32)
  may be converted to a system of algebraic equations by substituting
  \[
  x = \sin u \\
  y = \cos u
  \]
  and adding the relation $x^2 + y^2 - 1 = 0$.

- **Determination of the sign of $H_{u^{(n)}}$.** This is done using new symbolic techniques developed at ZIB. Two different methods are available: the simpler one of these tries to deduce the sign of the expression at hand by breaking it down into its parts and determining the sign of those. Though this seems to be a rather simple algorithm, it is nevertheless very powerful and can be applied to a wide range of expressions. If this is not sufficient, the user will be queried for the missing information.

Finally, the solution is substituted into the Hamiltonian and the differential equations. This uses again very much the standard facilities of the REDUCE system for differentiation, substitution and extraction of parts of expressions.

If there is no linear control $u^{(0)}$, all that remains to do is to solve the system of differential equations. To this end, OCCAL generates code for use with a numerical code for solving boundary value problems. The code generation step uses the GENTRAN translator for its work; this package has interfaces for FORTRAN, RATFOR, PASCAL, and C programming languages.

In our examples we use exclusively C; our numerical BVP solver is MULCON [7] which was translated to C semi-automatically, first using a FORTRAN-to-C translator and then beautifying the automatically produced code by hand to make it better readable.

Several functions are generated in the target language:

- **Calculation of boundary values $r$.**
Figure 1: An overview of the OCCAL system

- Calculation of differential equations $f$.
- Calculation of $f_y$ to solve the variational equation (for Wronskian approximations $G_j, P_j$).
- Calculation of the matrices of boundary condition derivatives $R_1 = r_{y(a)}$ and $R_m = r_{y(b)}$.

These are exactly the functions that are needed to compute the Jacobian (26).

The symbolic calculation of the matrix entries is of some importance for the whole computation, since this avoids the (external) numerical differentiation of $f$ and $r$, which might well lead to problems in the numerical calculation. Furthermore, one may reorder the variables to take advantage of the possible block structure of $R_1$ and $R_m$.

Without further manipulation, the resulting code is generated directly from the expressions obtained in the previous phase, i.e. in a fully expanded form. This is, of course, non-optimal, except for really simple cases. Therefore several different optimization strategies can be applied.

We developed a very simple one that separates the calculation of algebraic and transcendental functions like $\sqrt{x}$ or $e^x$, (so that their values are computed only once) and uses Horner's scheme to evaluate polynomials. In the case of
sparse polynomials in many variables, however, using Horner's scheme is not straightforward, since it is not a priori clear how the variables should be ordered.

A more elaborate optimization scheme was developed and implemented by Van Hulzen et al. [9]. It attempts to minimize the number of arithmetic operations. The optimizer performs heuristic searches on arithmetic expressions, detecting and extracting common subexpressions, and replacing them by temporary variable names.

The choice of optimization strategy depends mainly on the problem at hand. Complete optimization takes a considerable amount of computing time, but this may be payed off by the generation of more efficient and better readable numerical code. For every new problem, it is therefore necessary to exploit all possibilities to arrive at an optimal solution.

If, on the other hand, we have a problem with linear control \( u^0 \), the next step is to try to solve the remaining linear optimization problem. The present OCCAL system is not yet capable to distinguish between the singular and the bang-bang case. This is being worked on.

Figure 1 depicts the overall structure of the system.

4 Examples

In this section we present a few simple and well-known examples to illustrate the performance of our system.

Example 1: Turbo Generator [13].

This example is simple enough to do all necessary calculations by hand, and therefore well-suited to check the performance of OCCAL.

We will not discuss the technical background of the turbo generator as a machine. Rather we concentrate on the mathematical model derived from these technical considerations.

The dynamic model of the turbo generator consists of the following system of ordinary differential equations:

\[
\begin{align*}
\dot{x}_1 &= x_2x_4 \\
\dot{x}_2 &= \frac{1}{M}(u_1 - s_4x_1x_4 - s_5x_1x_3 - \kappa dx_2) \\
\dot{x}_3 &= u_2 - Ax_3 + cx_4 \\
\dot{x}_4 &= -x_1x_2
\end{align*}
\]

(33)

Here \( x_i \) are the dynamic variables, \( u_i \) the control variables, \( s_4, s_5, \kappa, A, M, \) and \( c \) are constants. The stationary state of the machine is characterized by \( x_i = x_i^* \) and \( u_i = u_i^* \), with certain constants \( x_i^*, u_i^* \).
The functional to be minimized describes the deviation from a stationary state

\[ J_1(x, u) = \int_0^T \left( \alpha((x_1 - x_1^*)^2 + (x_4 - x_4^*)^2) + \alpha_2 x_2^2 + \alpha_3(x_3 - x_3^*)^2 + \beta_1(u_1 - u_1^*)^2 + \beta_2(u_2 - u_2^*)^2 \right) dt \]  

with certain weight factors \( \alpha, \alpha_2, \alpha_3, \beta_1, \) and \( \beta_2. \) Boundary conditions are

\[ x(0) = (x_1^0, x_2^0, x_3^0, x_4^0) =: x_0^0 \]  
\[ x(T) \text{ free} \]  

The control variables \( u^{(n)} = (u_1, u_2) \) appear quadratically in the Hamiltonian, which leads to a system of two linear equations with a unique solution. After substituting the solution into the differential equations we end up with a boundary value problem with four dynamical variables. Due to its simplicity, no optimization of the code is necessary.

A corresponding OCCAL sample session, the input file for OCCAL itself, the REDUCE input file generated from this by the pre-processor, and the C source file are shown in Appendix A. It can be seen that the intermediate REDUCE input file is well-readable, so that the experienced user may want to edit this file directly. This example takes about 25 seconds of real time on a typical (Sun 4) workstation.

The file of generated C code shows several interesting points:

- The constants defined in the CONSTANTS section of the input file are rendered as C pre-processor macros.
- There are three C functions defined: for the boundary values (R1), for the functions \( f_i \) (FCN1), and for the functions \( f_y \) that are needed in the solution of the variational equation (DFCN1).
- The functions for the calculation of the matrices \( R_{ij} \) and \( R_{y_1} \) have been skipped in this example.
- The code optimization process introduces several intermediate variables to reduce the number of operations in the program. These are easily identified by their name which consists always of the letter G (short for generated) followed by a number. It is possible to choose other names as well.

With a rather simple main program to drive the calculation it is easy to reproduce the numerical results of [13].


This is a well-known test problem of somewhat higher degree of complexity. Here, the expressions arising from the calculation turn out to be rather lengthy.

13
The re-entry of an Apollo vehicle into the Earth’s atmosphere is governed by the following system of differential equations:

\[
\begin{align*}
\dot{v} &= V(v, \gamma, \xi, u) = -\frac{Spv^2}{2m}C_W(u) - \frac{g \sin \gamma}{(1 + \xi)^2} \\
\dot{\gamma} &= \Gamma(v, \gamma, \xi, u) = \frac{Spv}{2m}C_A(u) + \frac{v \cos \gamma}{R(1 + \xi)} - \frac{g \cos \gamma}{v(1 + \xi)^2} \\
\dot{\xi} &= \Xi(v, \gamma, \xi, u) = \frac{v \sin \gamma}{R} \\
\dot{\zeta} &= \Xi(v, \gamma, \xi, u) = \frac{v \cos \gamma}{1 + x_1}
\end{align*}
\]

where \( u \) is the control variable; and state variables are: \( v \) (tangential velocity), \( \gamma \) (flight path angle), \( \xi = h/R \) (relative height over ground), \( \zeta \) (velocity over ground). Moreover, the expressions \( p = \rho_0 \exp(-\beta R \xi) \), \( C_W(u) \) (aerodynamical resistance), \( C_A(u) \) (drag or lift coefficient), \( S \) (front area), and \( m \) (mass of the vehicle) appear in the above equations.

The boundary values at the begin of the reentry into the atmosphere and at the time of landing are

\[
\begin{align*}
v(0) &= 0.36, & \gamma(0) &= -8.1^\circ \cdot \frac{\pi}{180^\circ}, & \xi(0) &= \frac{4}{R} \\
v(T) &= 0.27, & \gamma(T) &= 0, & \xi(T) &= \frac{2.5}{R}
\end{align*}
\]

The final time \( T \) is free; \( \zeta \) is not considered for the optimization. The functional to be minimized is the heating of the capsule, approximated by

\[
J = \int_0^T 10v^3 \sqrt{\rho} dt.
\]

The problem has only one control variable, but equation (8) turns out to be transcendental, of the type (32) discussed earlier. It is possible, by a change of variables, to turn it into a system of algebraic equations which can be solved by computing the Groebner base of the polynomials. This is done automatically by the REDUCE equation solver. As a result, we get two solutions, only one of which leads to a positive definite determinant. This can be determined automatically if the additional information of the variable \( v > 0 \) is given to the system. Otherwise the user is prompted for the sign of \( v \).

The further calculation then proceeds, in principle, just as in the previous example. However, the expressions to be turned into numerical code are quite large, so that we apply optimization techniques during the code generation process. Of the two levels described above, the SCOPE package by van Hulzen [9] is able to reduce the size of the C source code from 35 kBytes (808 lines) to 12 kBytes (378 short lines). The simpler package which only extracts simple common subexpression reduces the size to 27 kBytes (570 lines). Using the latter, the calculation takes about 50 seconds of real time on a Sun 4 workstation.
5 Conclusion

As documented above, OCCAL has reached a first stage of development where it can be successfully used to help in the solution of optimal control problems. The tasks already accomplished are:

- Calculation of Hamiltonian.
- Determination of nonlinear and linear control.
- Code generation with optional optimization.

Next steps of development will include the following features that we are planning to incorporate in the future:

- Symbolic-numeric determination of nonlinear control: It is rather obvious that not for all systems of equations an analytic solution can be found. In these cases the algebraic equations have to be coupled to the dynamical system (1) and the whole must be solved numerically. The numeric treatment of algebro-differential equations is being worked on.

- Determination of inequalities: The determination of the sign of the determinant (8) is only a first simple step in the treatment of inequalities. There are a number of other places where inequalities arise, e.g., when inequalities involving the state variables appear. This is especially important in order to find the intervals where the switching functions vanish identically.

- Inclusion of state constraints: these may be very important as they may determine the structure of the linear control (i.e., singular vs. bang-bang). As far as inequalities are concerned, this is related to the previous point, and has probably to be treated in a common framework.

- Treatment of singular sub-arcs.

Acknowledgments. The authors wish to thank M. Wulkow for several helpful discussions.

References


Appendix A  An OCCAL sample session

As an example we treat the example of the turbo generator. We assume that the following input file has been prepared.

COMMENT
   Turbo generator  Input file

NAME
   turbo

TITLE
   TURBOGENERATOR

CONSTANTS
   x10 := 0.60295
   x20 := 0.0
   x30 := 1.87243
   x40 := 0.79778
   alpha := 2.5
   alpha2 := 1.0
   alpha3 := 0.1
   beta1 := 1.0
   beta2 := 1.0
   M := 0.04225
   x1s := 0.60295
   x2s := 0.00
   x3s := 1.87253
   x4s := 0.79778
   s4 := 0.0
   s5 := 0.0
   c := 0.0
   A := 0.17
   u1s := 0.80
   u2s := 0.73962
   kappad := 0.02535

FUNCTIONAL
   alpha * ((x1 - x1s)**2 + (x4 - x4s)**2) +
   alpha2 * x2**2 +
   alpha3 * (x3-x3s)**2 +
   beta1 * (u1 - u1s)**2 + beta2 * (u2 - u2s)**2;

DYNAMIC_VARIABLES
   x1
CONTROL_VARIABLES
u1
u2

BOUNDARIES
0;
ts;

TIME_VARIABLE
t

DIFFERENTIAL_EQUATIONS
x1 " x2 * x4;
x2 " 1/M *
           (u1 - s4 * x1 * x4 - s5 * x1 * x3 - kappa * x2);
x3 " u2 - A * x3 + c * x4;
x4 " - x1 * x2;

BOUNDARY_CONDITIONS
x1(0) - x10;
x2(0) - x20;
x3(0) - x30;
x4(0) - x40;
lamb_x1(ts);
lamb_x2(ts);
lamb_x3(ts);
lamb_x4(ts);

It is further assumed that the name of this file ends in .description, e.g., turbo.description. Type in:

occal turbo

OCCAL displays

Parsing input file... done.
Starting symbolic processor...
REDUCE 3.4, 15-Jul-91 ...
and then a protocol of the following REDUCE statements that were generated by the pre-processor.

```
SET!_NAME ("turbo");
SET!_TITLE ({" TURBO GENERATOR "});

DEFINE!_CONSTANT (A, 0.17);  
DEFINE!_CONSTANT (x30, 1.87243);  
DEFINE!_CONSTANT (s5, 0.0);  
DEFINE!_CONSTANT (beta2, 1.0);  
DEFINE!_CONSTANT (c, 0.0);  
DEFINE!_CONSTANT (alpha3, 0.1);  
DEFINE!_CONSTANT (betal, 1.0);  
DEFINE!_CONSTANT (x3s, 1.87253);  
DEFINE!_CONSTANT (M, 0.04225);  
DEFINE!_CONSTANT (u2s, 0.73962);  
DEFINE!_CONSTANT (x20, 0.0);  
DEFINE!_CONSTANT (alpha2, 1.0);  
DEFINE!_CONSTANT (x4s, 0.79778);

DEFINE!_TIME!_VARIABLE (t);
DEFINE!_DYNAMIC!_VARIABLES ({x4, x3, x2, x1});
DEFINE!_CONTROL!_VARIABLES ({u1, u2});
DEFINE!_BOUNDARIES (0, ts);
DEFINE!_BC (x1(0) - x10);
DEFINE!_BC (x2(0) - x20);
DEFINE!_BC (x3(0) - x30);
DEFINE!_BC (x4(0) - x40);
```
DEFINE!_BC (lamb_x1(ts));
DEFINE!_BC (lamb_x2(ts));
DEFINE!_BC (lamb_x3(ts));
DEFINE!_BC (lamb_x4(ts));

DEFINE!_FUNCTIONAL (alpha * ((x1 - x1s)**2 + (x4 - x4s)**2) +
alpha2 * x2**2 +
alpha3 * (x3-x3s)**2 +
beta1 * (u1 - u1s)**2 + beta2 * (u2 - u2s)**2);

DEFINE!_DIFFERENTIAL!_EQUATION (x4, - x1 * x2);
DEFINE!_DIFFERENTIAL!_EQUATION (x3, u2 - A * x3 + c * x4);
DEFINE!_DIFFERENTIAL!_EQUATION (x2, 1/M *
    (u1 - s4 * x1 + x4 - s5 * x1 + x3 - kappa * x2));

DEFINE!_DIFFERENTIAL!_EQUATION (x1, x2 * x4);
SET!_RIGHT!_BD!_FREE (0);
SET!_TARGET!_LANGUAGE (C);
end;

After a while it displays
=================================
That's all, folks!
=================================

and then
Quitting
OCCAL finished.

There is now a file with name turbo1.c. We show its contents for illustration.

#include <math.h>
#define X4S 0.79778
#define ALPHA2 1
#define X20 0
#define U2S 0.73962
#define M 0.04225
#define X3S 1.87253
#define BETA1 1
#define ALPHA3 0.1
#define X10 0.60295
#define S4 0
#define U1S 0.8
#define X2S 0
#define X1S 0.60295
#define KAPPAD 0.02535
#define X40 0.79778
#define ALPHA 2.5
#define C 0
#define BETA2 1
#define S5 0
#define X30 1.87243
#define A 0.17

R1(YA,YB,TAU,W)
double YA[8],YB[8],TAU[1],W[8];
{
    W[7]=YA[0]-X10;
    W[6]=YA[1]-X20;
    W[3]=YB[4];
    W[2]=YB[5];
    W[1]=YB[6];
    W[0]=YB[7];
}

FCN1(TT,X,TAU,DX)
double X[8],DX[8],TT[1],TAU[1];
{
    double ANS1,LAMB_X4,LAMB_X3,LAMB_X2,LAMB_X1,X4,X3,X2,X1;
    LAMB_X4=X[7];
    LAMB_X3=X[6];
    LAMB_X2=X[5];
    LAMB_X1=X[4];
    X4=X[3];
    X3=X[2];
X2=X[1];
X1=X[0];
ANS1=M*M;

DX[0]=X2*X4;
DX[1]=-(M*X1*X3*S5*BETA1)-(M*X1*X4*S4*BETA1)-(M*X2*
  KAPPAD*BETA1)+M*U1S*BETA1-(0.5*LAMB_X2))/(ANS1*BETA1);
DX[2]=-(A*X3*BETA2)+C*X4*BETA2+BETA2*U2S-(0.5*LAMB_X3))
  /BETA2;
DX[3]=-X1*X2;
DX[4]=-(2.0*M*X1*ALPHA)+M*X2*LAMB_X4+2.0*M*ALPHA*X1S+X3
  *S5*LAMB_X2+X4+S4*LAMB_X2)/M;
DX[5]=(M*X1*LAMB_X4-(2.0*M*X2*ALPHA2)-(M*X4*LAMB_X1)+
  KAPPAD*LAMB_X2)/M;
DX[6]=(A*M*LAMB_X3-(2.0*M*X3*ALPHA3)+2.0*M*ALPHA3*X3S+X1
  *S5*LAMB_X2)/M;
DX[7]=-(C*M*LAMB_X3)-(M*X2*LAMB_X1)-(2.0*M*X4*ALPHA)+
  2.0*M*ALPHA*X4S+X1S4*LAMB_X2)/M;

DFCN1(TT,X,TAU,DFDX,DFDTAU)
double X[8],DFDX[64],TT[1],DFDTAU[8],TAU[1];
{

double ANS1,LAMB_X4,LAMB_X3,LAMB_X2,LAMB_X1,X4,X3,X2,X1;
LAMB_X4=X[7];
LAMB_X3=X[6];
LAMB_X2=X[5];
LAMB_X1=X[4];
X4=X[3];
X3=X[2];
X2=X[1];
X1=X[0];
ANS1=M*M;
DFDTAU[0]=0.0;
DFDTAU[1]=0.0;
DFDTAU[2]=0.0;
DFDTAU[3]=0.0;
DFDTAU[4]=0.0;
DFDTAU[5]=0.0;
DFDTAU[6]=0.0;
DFDTAU[7]=0.0;
DFDX[0]=0.0;
DFDX[1]=-(X3*S5)-(X4*S4))/M;
DFDX[2]=0.0;
DFDX[3]=-X2;
DFDX[4]=-(2.0*ALPHA);
DFDX[5]=LAMB_X4;
DFDX[6]=(S5*LAMB_X2)/M;
DFDX[7]=(S4*LAMB_X2)/M;
DFDX[8]=X4;
DFDX[9]=-KAPPAD/M;
DFDX[10]=0.0;
DFDX[12]=LAMB_X4;
DFDX[13]=-(2.0*ALPHA2);
DFDX[14]=0.0;
DFDX[15]=-LAMB_X1;
DFDX[16]=0.0;
DFDX[17]=-(X1*S5)/M;
DFDX[18]=-A;
DFDX[19]=0.0;
DFDX[20]=(S5*LAMB_X2)/M;
DFDX[21]=0.0;
DFDX[22]=-(2.0*ALPHA3);
DFDX[23]=0.0;
DFDX[24]=X2;
DFDX[25]=-(X1*S4)/M;
DFDX[26]=C;
DFDX[27]=0.0;
DFDX[28]=(S4*LAMB_X2)/M;
DFDX[29]=-LAMB_X1;
DFDX[30]=0.0;
DFDX[31]=-(2.0*ALPHA);
DFDX[32]=0.0;
DFDX[33]=0.0;
DFDX[34]=0.0;
DFDX[35]=0.0;
DFDX[36]=0.0;
DFDX[37]=-X4;
DFDX[38]=0.0;
DFDX[39]=-X2;
DFDX[40]=0.0;
DFDX[41]=-0.5/(ANSl*BETAl);
DFDX[42]=0.0;
DFDX[43]=0.0;
DFDX[44]=(X3*S5+X4*S4)/M;
DFDX[45]=KAPPAD/M;
DFDX[46]=(X1*S5)/M;
DFDX[47]=(X1*S4)/M;
DFDX[48]=0.0;
DFDX[49]=0.0;
DFDX[50] = -0.5/BETA2;
DFDX[51] = 0.0;
DFDX[52] = 0.0;
DFDX[53] = 0.0;
DFDX[54] = A;
DFDX[55] = -C;
DFDX[56] = 0.0;
DFDX[57] = 0.0;
DFDX[58] = 0.0;
DFDX[59] = 0.0;
DFDX[60] = X2;
DFDX[61] = X1;
DFDX[62] = 0.0;
DFDX[63] = 0.0;
Appendix B  Files for the reentry problem

B.1 Problem description

NAME
Reentry

TITLE
Stoer-Bulirsch, Kap. 7

CONSTANTS
R = 209
beta = 4.26
rho0 := 2.704e-3
g := 0.32172e-3

DYNAMIC_VARIABLES
v
gamma
xi

CONTROL_VARIABLES
u

ABBREVIATIONS
for all h let rho(h) = rho0 * exp(- beta * h);
c_W(u) := 1.174 - 0.9 * cos(u);
c_A(u) := 0.6 * sin(u);
S := 53200 * m;
h := xi * R;

TIME_VARIABLE
tt

DIFFERENTIAL_EQUATIONS
DOT(v) := -S*rho(h)*v~2*c_W(u)/(2*m) - g * sin(gamma) / (1 + xi)^2;
DOT(gamma) := S*rho(h)*v*c_A(u)/(2*m) + v*cos(gamma) / (R * (1 + xi)) - g*cos(gamma)/(v * (1 + xi)^2);
xiv = v * sin (gamma) / R;

FUNCTIONAL
10*v^3*sqrt(rho(h));
END_POINT_IS_FREE

BOUNDARIES
0;
TIME_T;

BOUNDARY_CONDITIONS
v(0) = 0.36;
gamma(0) = 8.1*Pi/180;
xi(0) = 4/R;

v(TIME_T) = 0.27;

gamma(TIME_T);

xi(TIME_T) = 2.5/R;

RESTRICTIONS
v > 0;

B.2 REDUCE input file

SET!_NAME ("Reentry");

SET!_TITLE ("Stoer-Bulirsch, Kap. 7 ");

for all h let rho(h) = rho0 * exp(- beta * h);

\[ c_W (u) := 1.174 - 0.9 \cos(u); \]

\[ c_A (u) := 0.6 \sin(u); \]

\[ S := 53200 \times m; \]

\[ h := xi \times R; \]

DEFINE!_CONSTANT (R, 209);

DEFINE!_CONSTANT (beta, 4.26);

DEFINE!_CONSTANT (rho0, 2.704e-3);

DEFINE!_CONSTANT (g, 0.32172e-3);

DEFINE!_TIME!_VARIABLE (tt);

DEFINE!_DYNAMIC!_VARIABLES (\{gamma, v, xi\});

DEFINE!_CONTROL!_VARIABLES (\{u\});
ASSUME (v > 0);

DEFINE!_BOUNDARIES (0, TIME_T);
DEFINE!_BC (v(0) - 0.36);
DEFINE!_BC (gamma(0) + 8.1*pi/180);
DEFINE!_BC (xi(0) - 4/R);
DEFINE!_BC (v(TIME_T) - 0.27);
DEFINE!_BC (gamma(TIME_T));
DEFINE!_BC (xi(TIME_T) - 2.5/R);
DEFINE!_FUNCTIONAL (10*v^3*sqrt(rho(h)));

DEFINE!_DIFFERENTIAL!_EQUATION (gamma, S*rho(h)*v*c_A(u)/(2*m) 
  + v*cos(gamma) / (R * (1 + xi)) 
  - g*cos(gamma)/(v * (1 + xi)^2));

DEFINE!_DIFFERENTIAL!_EQUATION (v, -S*rho(h)*v^2*c_W(u)/(2*m) 
  - g * sin(gamma) / (1 + xi)^2);

DEFINE!_DIFFERENTIAL!_EQUATION (xi, v * sin (gamma) / R);

SET!_RIGHT!_BD!_FREE (1);
SET!_TARGET!_LANGUAGE (C);

end;
Appendix C  Input file format
for the pre-processor

The input file consists of a number of sections, each identified by a keyword on a line of its own, and ended by an empty line. Possible keywords are:

COMMENT  Used only to explain what is going on. Everything up to the next empty line is ignored.

NAME  A short name to identify the problem, used for file names, etc.

TITLE  Description of the problem, can consist of several lines of text.

CONSTANTS  Constants of the problem, i.e., identifiers whose values do not change. Can only be set at compile time. A value is mandatory.

PARAMETERS  Parameters whose values may be set at run time. A value is optional.

DYNAMIC_VARIABLES  The dynamic variables, called $y_i$ in a previous section, each on a line of its own.

CONTROL_VARIABLES  The control variables, called $u_i$ in a previous section, each on a line of its own.

TIME_VARIABLE  The name of the independent variable.

BOUNDARIES  Two expressions in REDUCE syntax that denote the boundaries of the interval. Must be delimited by a semicolon.

FUNCTIONAL  The functional to be minimized.

DIFFERENTIAL_EQUATIONS  The dynamical system.

END_POINT_IS_FREE  Specifies that the right boundary is free and to be calculated.

BOUNDARY_CONDITIONS  The boundary conditions at the points specified in the BOUNDARIES section.

ABBREVIATIONS  Arbitrary REDUCE expressions, each delimited by a semicolon.

RESTRICTIONS  Additional conditions, such as a variable being bounded from below.

TARGET_LANGUAGE  The programming language in which the numerical programs are generated. Possible are: C, FORTRAN, RATFOR, and PASCAL.

TEMPVAR_PREFIX  The prefix used for constructing variable names during code generation. Default is ans.

The order of these sections is largely arbitrary, except for one restriction: the dynamic variables need to be defined before the corresponding differential equations.