2010 IEEE International Symposium on Computer-Aided Control System Design
Part of 2010 IEEE Multi-Conference on Systems and Control
Yokohama, Japan, September 8-10, 2010

Solving Complex Optimal Control Problems at No Cost with $\mathcal{PSOPT}$

Victor M. Becerra

***Abstract*— This paper introduces $\mathcal{PSOPT}$, an open source optimal control solver written in C++. $\mathcal{PSOPT}$ uses pseudospectral and local discretizations, sparse nonlinear programming, automatic differentiation, and it incorporates automatic scaling and mesh refinement facilities. The software is able to solve complex optimal control problems including multiple phases, delayed differential equations, nonlinear path constraints, interior point constraints, integral constraints, and free initial and/or final times. The software does not require any non-free platform to run, not even the operating system, as it is able to run under Linux. Additionally, the software generates plots as well as LaTeX code so that its results can easily be included in publications. An illustrative example is provided.**

I. INTRODUCTION

Computational optimal control is a field that has been developing over several decades. Many researchers and engineers have implemented their own code to test their theoretical developments, or to solve specific problems. Over the years, a number of general purpose software tools have become available to solve optimal control problems. They differ in their licensing cost and terms, the type of computational technique that they implement, the nonlinear programming code employed, and on the facilities available to solve complex problems.

Indirect methods solve optimal control problems by finding a numerical solution to the two point boundary value problem that arises from the application of the optimality conditions associated with the problem. On the other hand, direct methods for computational optimal control involve the discretization of the differential equations of the problem. Different techniques have been proposed to discretize the problem's differential equations. The use of well known approximations for the numerical solution of initial value problems in ordinary differential equations (e.g. trapezoidal, Runge-Kutta and Hermite-Simpson methods) for the solution of optimal control problems has been implemented in various software packages and has been tested in many applications over several years [1]. These approximations have local support, this is the discretization points are located so that they support the local behaviour of the state functions. Other approximations involve the use of orthogonal polynomials (such as Legendre or Chebyshev polynomials) to collocate the differential equations at carefully selected nodes that yield highly accurate approximations for smooth state functions [2]. Methods of this type are known in the literature as *psudospectral* [3] or *orthogonal collocation on finite elements* [4]. There are formulations where the orthogonal polynomials extend over the whole time interval of the problem, such that these methods have global support, while in other formulations the problem's time interval is divided into several segments, such that they have local support. Pseudospectral techniques were originally developed for the solution of partial differential equations [5], but their use in optimal control is also extending, and many successful applications have been reported [6], [7].

A well known proprietary tool for solving large scale optimal control problems is SOCS [8], which is a Fortran based package that implements direct transcription methods based on various local approximations. DIRCOL [9] is another Fortran based tool implementing a direct collocation method, while PROPT [10] and DIDO [11] are examples of proprietary packages that run under MATLAB and implement pseudospectral methods. A MATLAB based open source tool that uses the Gauss pseudospectral method is GPOPS [12].

While industrial and academic users benefit from the use of proprietary closed-source optimal control software, these packages do not allow users to see in detail how the underlying algorithms are implemented, and they cannot be customized or changed by their users, which limits their value for the purposes of education or academic research.

$\mathcal{PSOPT}$ is an open source optimal control package written in C++ that uses direct collocation methods, including Legendre and Chebyshev pseudospectral discretizations, as well as local transcriptions such as trapezoidal or Hermite-Simpson. The combination of local ODE transcriptions, and global or semi-global pseudospectral methods in a single package is a unique feature of the software. The use of C++ allows users to link $\mathcal{PSOPT}$ from a stand alone applications written in C++ or other programming languages. Moreover, its zero cost, open source nature, licensing terms, and independence from proprietary software platforms may be attractive to users.

II. PROBLEM DOMAIN

$\mathcal{PSOPT}$ is able to deal with problems with the following characteristics:

- Single or multiphase problems
- Continuous time nonlinear dynamics
- General endpoint constraints
- Nonlinear path constraints (equalities or inequalities) on states and/or control variables
- Integral constraints
- Interior point constraints
- Bounds on controls and state variables
- General cost function with Lagrange and Mayer terms.

Victor M. Becerra is with the School of Systems Engineering, University of Reading, Reading RG6 6AY, United Kingdom v.m.becerra@ieee.org

978-1-4244-5355-9/10/$26.00 ©2010 IEEE

- Free or fixed initial and final conditions
- Linear or nonlinear linkages between phases
- Fixed or free initial time
- Fixed or free final time
- Optimisation of static parameters
- Differential equations with delayed variables.

III. FEATURES OF THE SOFTWARE

The implementation has the following features:

- Choice between pseudospectral (Legendre, Chebyshev) or local discretizations (Trapezoidal, Hermite-Simpson).
- Automatic scaling of variables and constraints.
- Automatic first and second derivatives using the ADOL-C library.
- Numerical differentiation by using sparse finite differences.
- Automatic mesh refinement options for global or local discretizations.
- Automatic identification of the Jacobian and Hessian sparsity.
- DAE formulation, so that differential and algebraic constraints can be implemented in the same C++ function.
- Interfaces to generate surface plots. polar plots, as well as parametric 3D curves in GNUplot.
- Automatic generation of LaTeXcode to produce a table that summarizes the results of the manual or automatic mesh refinement process, which may be useful to include in reports or papers.
- Automatic definition of multi-segment problems.

$\mathcal{PSOPT}$ has interfaces to the following NLP solvers:

- IPOPT: an open source C++ implementation of an interior point method for large scale problems [13]. Note that IPOPT requires the user to employ a sparse linear solver from a number of options, some of which can be acquired at no cost by academic or industrial users.
- SNOPT: a well known and widely used proprietary large scale NLP solver [14]. Note that the use of SNOPT is optional.

IV. PROBLEM FORMULATION

$\mathcal{PSOPT}$ solves problems with one or more phases. Some optimal control problems can be conveniently formulated as having N_p phases. Phases may be inherent to the problem (for example, a spacecraft drops a section and enters a new phase). Phases may also be introduced by the analysist to allow for peculiarities in the solution of the problem such as discontinuities in the control variables. Note that phases need not be sequential.

The family of problems that can be solved by $\mathcal{PSOPT}$ can be formulated as follows. Find the control trajectories, $u^{(i)}(t), t \in [t_0^{(i)}, t_f^{(i)}]$, state trajectories $x^{(i)}(t), t \in [t_0^{(i)}, t_f^{(i)}]$, static parameters $p^{(i)}$, and times $t_0^{(i)}, t_f^{(i)}$, $i = 1, \ldots, N_p$, to minimize the following performance index:

$$J = \sum_{i=1}^{N_p} \left[\varphi^{(i)}[x^{(i)}(t_f^{(i)}), p^{(i)}, t_f^{(i)}] + \int_{t_0^{(i)}}^{t_f^{(i)}} L^{(i)}[x^{(i)}(t), u^{(i)}(t), p^{(i)}, t] dt \right]$$

subject to the differential constraints:

$$\dot{x}^{(i)}(t) = f^{(i)}[x^{(i)}(t), u^{(i)}(t), p^{(i)}, t], \ t \in [t_0^{(i)}, t_f^{(i)}],$$

the path constraints

$$h_L^{(i)} \leq h^{(i)}[x^{(i)}(t), u^{(i)}(t), p^{(i)}, t] \leq h_U^{(i)}, \ t \in [t_0^{(i)}, t_f^{(i)}],$$

the event constraints:

$$e_L^{(i)} \leq e^{(i)}[x^{(i)}(t_0^{(i)}), u^{(i)}(t_0^{(i)}), x^{(i)}(t_f^{(i)}), u^{(i)}(t_f^{(i)}), p^{(i)}, t_0^{(i)}, t_f^{(i)}] \leq e_U^{(i)},$$

the phase linkage constraints:

$$\begin{aligned} \Psi_l \leq \Psi[&x^{(1)}(t_0^{(1)}), u^{(1)}(t_0^{(1)}), \\ &x^{(1)}(t_f^{(1)}), u^{(1)}(t_f^{(1)}), p^{(1)}, t_0^{(1)}, t_f^{(1)}, \\ &x^{(2)}(t_0^{(2)}), u^{(2)}(t_0^{(2)}), \\ &x^{(2)}(t_f^{(2)}), u^{(2)}(t_f^{(2)}), p^{(2)}, t_0^{(2)}, t_f^{(2)} \\ &\vdots \\ &x^{(N_p)}(t_0^{(N_p)}), u^{(N_p)}(t_0^{(N_p)}), \\ &x^{(N_p)}(t_f^{(N_p)}), u^{(N_p)}(t_f^{(N_p)})), p^{(N_p)}, t_0^{(N_p)}, t_f^{(N_p)}] \leq \Psi_u \end{aligned}$$

the bound constraints:

$$u_L^{(i)} \leq u^{(i)}(t) \leq u_U^{(i)}, \ t \in [t_0^{(i)}, t_f^{(i)}],$$

$$x_L^{(i)} \leq x^{(i)}(t) \leq x_U^{(i)}, \ t \in [t_0^{(i)}, t_f^{(i)}],$$

$$p_L^{(i)} \leq p^{(i)} \leq p_U^{(i)},$$

$$\underline{t}_0^{(i)} \leq t_0^{(i)} \leq \bar{t}_0^{(i)},$$

$$\underline{t}_f^{(i)} \leq t_f^{(i)} \leq \bar{t}_f^{(i)},$$

and the following constraints:

$$t_f^{(i)} - t_0^{(i)} \geq 0,$$

Function	Description
`endpoint_cost`	Computes the end point cost for each phase
`integrand_cost`	Computes the integrand of the cost function for each phase
`dae`	Computes the right hand side of the differential equations and the function of path constraints for each phase
`events`	Computes the event constraint function for each phase
`linkages`	Computes the linkages constraint function

TABLE I

DESCRIPTION OF C++ FUNCTIONS REQUIRED TO SETUP A PROBLEM UNDER $\mathcal{PSOPT}$

where $i = 1, \ldots, N_p$, and

$$
\begin{aligned}
&u^{(i)} : [t_0^{(i)}, t_f^{(i)}] \rightarrow \mathcal{R}^{n_u^{(i)}} \\
&x^{(i)} : [t_0^{(i)}, t_f^{(i)}] \rightarrow \mathcal{R}^{n_x^{(i)}} \\
&p^{(i)} \in \mathcal{R}^{n_p^{(i)}} \\
&\varphi^{(i)} : \mathcal{R}^{n_x^{(i)}} \times \mathcal{R}^{n_x^{(i)}} \times \mathcal{R}^{n_p^{(i)}} \times \mathcal{R} \times \mathcal{R} \rightarrow \mathcal{R} \\
&L^{(i)} : \mathcal{R}^{n_x^{(i)}} \times \mathcal{R}^{n_u^{(i)}} \times \mathcal{R}^{n_p^{(i)}} \times [t_0^{(i)}, t_f^{(i)}] \rightarrow \mathcal{R} \\
&f^{(i)} : \mathcal{R}^{n_x^{(i)}} \times \mathcal{R}^{n_u^{(i)}} \times \mathcal{R}^{n_p^{(i)}} \times [t_0^{(i)}, t_f^{(i)}] \rightarrow \mathcal{R}^{n_x^{(i)}} \\
&h^{(i)} : \mathcal{R}^{n_x^{(i)}} \times \mathcal{R}^{n_u^{(i)}} \times \mathcal{R}^{n_p^{(i)}} \times [t_0^{(i)}, t_f^{(i)}] \rightarrow \mathcal{R}^{n_h^{(i)}} \\
&e^{(i)} : \mathcal{R}^{n_x^{(i)}} \times \mathcal{R}^{n_u^{(i)}} \times \mathcal{R}^{n_x^{(i)}} \times \mathcal{R}^{n_u^{(i)}} \times \Re^{n_p^{(i)}} \times \mathcal{R} \times \mathcal{R} \\
&\qquad \rightarrow \mathcal{R}^{n_e^{(i)}} \\
&\Psi : U_\Psi \rightarrow \mathcal{R}^{n_\psi}
\end{aligned}
\tag{1}
$$

where U_ψ is the domain of function Ψ.

V. USER INTERFACE

Defining an optimal control problem involves specifying all the necessary values and functions that are needed to solve the problem. With $\mathcal{PSOPT}$, this is done by implementing C++ functions (e.g. the cost function), and assigning values to data structures to define various options. Once $\mathcal{PSOPT}$ has obtained a solution, the relevant variables can be obtained by interrogating a data structure. Table I describes the interface functions required to set up a problem.

VI. DISCRETIZATION METHODS

$\mathcal{PSOPT}$ allows the user to choose between the following methods to discretize the differential equations associated with the problem.

A. Legendre and Chebyshev pseudospectral methods

In the Legendre pseudospectral approximation of the optimal control problem, the state for each phase $x(\tau) \in \mathcal{R}^{n_x}$, where $\tau \in [-1, 1]$ is a normalised independent variable, is approximated by the N-order Lagrange polynomial $x^N(\tau)$ based on interpolation at the Legendre-Gauss-Lobatto (LGL) quadrature nodes (see [2]), so that:

$$x(\tau) \approx x^N(\tau) = \sum_{k=0}^{N} x(\tau_k)\mathcal{L}_k(\tau) \tag{2}$$

where $\mathcal{L}_k$ are Lagrange basis polynomials. Note that $x^N(\tau_k) = x(\tau_k)$ and $u^N(\tau_k) = u(\tau_k)$. The derivative of the state vector is approximated as follows:

$$\dot{x}(\tau_k) \approx \dot{x}^N(\tau_k) = \sum_{i=0}^{N} D_{ki} x^N(\tau_i),\ i = 0, \ldots N \tag{3}$$

where D is the $(N+1) \times (N+1)$ the differentiation matrix given by ([2]):

$$D_{ki} = \begin{cases} -\frac{L_N(\tau_k)}{L_N(\tau_i)} \frac{1}{\tau_k - \tau_i} & \text{if } k \neq i \\ N(N+1)/4 & \text{if } k = i = 0 \\ -N(N+1)/4 & \text{if } k = i = N \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

A set of $n_x(N+1)$ differential defect constraints is constructed on the basis of (3).

The Chebyshev approximation is similar, but it employs the Chebyshev-Gauss-Lobatto (CGL) quadrature nodes, and a different differentiation matrix, see [2].

B. Central differences method

This method computes the differential defect constraints using a $(N+1) \times (N+1)$ differentiation matrix given by:

$$
\begin{aligned}
&D_{0,0} = -1/h_0 \\
&D_{0,1} = 1/h_0 \\
&D_{i-1,i} = 1/(h_i + h_{i-1}), \qquad i = 2, \ldots N \\
&D_{i-1,i-2} = -1/(h_i + h_{i-1}) \quad i = 2, \ldots N \\
&D_{N,N-1} = -1/h_{N-1} \\
&D_{N,N} = 1/h_{N-1}
\end{aligned}
$$

where $h_k = \tau_{k+1} - \tau_k$. The method uses forward differences at τ_0, backward differences at τ_N, and central differences at τ_k, $k = 1, \ldots, N-1$. Notice that this discretization has less accuracy at both ends of the interval. This is compensated by the use of pseudospectral grids, which concentrate more grid points at both ends of the interval.

C. Trapezoidal method

With the trapezoidal method ([1]), the defect constraints are computed as follows:

$$\zeta(\tau_k) = x(\tau_{k+1}) - x(\tau_k) - \frac{h_k}{2}(f_k + f_{k+1}), \tag{5}$$

where $\zeta(\tau_k) \in \Re^{n_x}$ is the vector of differential defect constraints at node τ_k, $k = 0, \ldots, N-1$, $h_k = \tau_{k+1} - \tau_k$, $f_k = f[(\tau_k), u(\tau_k), p, \tau_k]$, $f_{k+1} = f[x(\tau_{k+1}), u(\tau_{k+1}), p, \tau_{k+1}]$. This gives rise to $n_x N$ differential defect constraints.

D. Hermite-Simpson method

With the Hermite-Simpson method ([1]), the defect constraints are computed as follows:

$$\zeta(\tau_k) = x(\tau_{k+1}) - x(\tau_k) - \frac{h_k}{6}(f_k + 4\bar{f}_{k+1} + f_{k+1}), \tag{6}$$

where

$$\bar{f}_{k+1} = f[\bar{x}_{k+1}, \bar{u}_{k+1}, p, \tau_k + \frac{h_k}{2}]$$

$$\bar{x}_{k+1} = \frac{1}{2}(x(\tau_k) + x(\tau_{k+1})) + \frac{h_k}{8}(f_k - f_{k+1})$$

where $\zeta(\tau_k) \in \Re^{n_x}$ is the vector of differential defect constraints at node τ_k, $k = 0, \ldots, N-1$, $h_k = \tau_{k+1} - \tau_k$, $f_k = f[(\tau_k), u(\tau_k), p, \tau_k]$, $f_{k+1} = f[x(\tau_{k+1}), u(\tau_{k+1}), p, \tau_{k+1}]$, and $\bar{u}_{k+1} = \bar{u}(\tau_{k+1})$ is a vector of midpoint controls (which are also decision variables). This gives rise to $n_x N$ differential defect constraints.

VII. DIFFERENTIATION OF USER SPECIFIED FUNCTIONS

Users are encouraged to use, whenever possible, the automatic differentiation facilities provided by the ADOL-C library. The use of automatic derivatives is the default behaviour. $\mathcal{PSOPT}$ uses the ADOL-C drivers for sparsity determination, Jacobian, Hessian, and gradient evaluation. Automatic derivatives are more accurate than numerical derivatives as they are free of truncation errors. Moreover, $\mathcal{PSOPT}$ works faster when using automatic derivatives.

There may be cases, however, where it is preferrable or necessary to use numerical derivatives. If the user chooses to employ numerical derivatives, then $\mathcal{PSOPT}$ will calculate the derivatives required by the nonlinear programming algorithm by using sparse finite differences, such that groups of variables are perturbed simultaneously, see [15].

VIII. EVALUATING THE DISCRETIZATION ERROR

$\mathcal{PSOPT}$ evaluates the discretization error using a method adopted from [1]. Define the error in the differential equation as a function of time:

$$\epsilon(t) = \dot{\tilde{x}}(t) - f[\tilde{x}(t), \tilde{u}(t), p, t]$$

where $\tilde{x}$ is an interpolated value of the state vector given the grid point values of the state vector, $\tilde{x}$ is an estimate of the derivative of the state vector given the state vector interpolant, and $\tilde{u}$ is an interpolated value of the control vector given the grid points values of the control vector. The type of interpolation used depends on the collocation method employed. For Legendre and Chebyshev methods, the interpolation done by the Lagrange interpolant. For Trapezoidal and Hermite-Simpson methods, cubic spline interpolation is used. The absolute local error corresponding to state i on a particular interval $t \in [t_k, t_{k+1}]$, is defined as follows:

$$\eta_{i,k} = \int_{t_k}^{t_{k+1}} |\epsilon_i(t)| \mathrm{d}t$$

where the integral is computed using the composite Simpson method. The default number of integration steps for each interval is 10, but this can be changed by the user. The relative local error is defined as:

$$\epsilon_k = \max_i \frac{\eta_{i,k}}{w_i + 1}$$

where

$$w_i = \max_{k=1}^{N} \left[|\tilde{x}_{i,k}|, |\dot{\tilde{x}}_{i,k}| \right]$$

IX. MESH REFINEMENT

A. Manual mesh refinement

Manual mesh refinement, which is the default option, is performed by interpolating a previous solution based on n_1 nodes, into a new mesh based on n_2 nodes, where $n_2 > n_1$, and using the interpolated solution as an initial guess for a new optimization. The variables which are interpolated include the controls, states and Lagrange multipliers associated with the differential defect constraints, which are related to the co-states.

B. Automatic mesh refinement with pseudospectral grids

In this case, $\mathcal{PSOPT}$ will compute the maximum discretization error $\epsilon^{(i,m)}$ for every phase i at every mesh refinement iteration m, as described in Section VIII.

The method is based on a nonlinear least squares fit of the maximum discretization error for each phase with respect to the mesh size:

$$\hat{y}^i = \varphi_1 \theta_1 + \theta_2$$

where $\hat{y}^i$ is an estimate of $\log(\epsilon^{(i)})$, $\varphi_1 = \log(N)$, θ_1 and θ_2 are parameters which are estimated based on the mesh refinement history. This is equivalent to modelling the dependency of $\epsilon^{(i)}$ with respect to the number of nodes N_i as follows:

$$\epsilon^{(i)} = C \frac{1}{N_i^m}$$

where $m = -\theta_1$, $C = \exp(\theta_2)$. This dependency relates to the upper bound on the $\mathcal{L}_2$ norm of the interpolation error given by [2].

C. Automatic mesh refinement with local collocation

In this case, $\mathcal{PSOPT}$ will compute the discretization error $\epsilon^{(i,m)}$ for every phase i at every mesh refinement iteration m, as described in Section VIII. The goal of the local mesh refinement procedure is to reduce as much as possible the maximum discretization error within each phase, by using a specified number of new points to subdivide the discretization intervals according to their individual discretization error. The local mesh refinement method implemented in $\mathcal{PSOPT}$ is based on the algorithm described by Betts [1]. By default the local mesh refinement algorithm starts with trapezoidal discretization, which is then switched to Hermite-Simpson after two iterations.

X. EXAMPLE

The following example illustrates some of the capabilities of the software. Readers are referred to the user's manual [16] which contains over 30 different examples that illustrate more comprehensively the capabilities of the software.

A. Missile terminal burn maneouvre

This example illustrates the design of a missile trajectory to strike a specified target from given initial conditions in minimum time [17]. Figure 1 shows the variables associated with the dynamic model of the missile employed in this example, where γ is the flight path angle, α is the angle of

TABLE II
PARAMETERS VALUES OF THE MISSILE MODEL

Parameter	Value	Units
m	1005	kg
g	9.81	m/s^2
S_{ref}	0.3376	m^2
A_1	-1.9431	
A_2	-0.1499	
A_3	0.2359	
B_1	21.9	
B_2	0	
C_1	3.312×10^{-9}	kg/m^5
C_2	-1.142×10^{-4}	kg/m^4
C_3	1.224	kg/m^3

attack, V is the missile speed, x is the longitudinal position, h is the altitude, D is the axial aerodynamic force, L is the normal aerodynamic force, and T is the thrust.

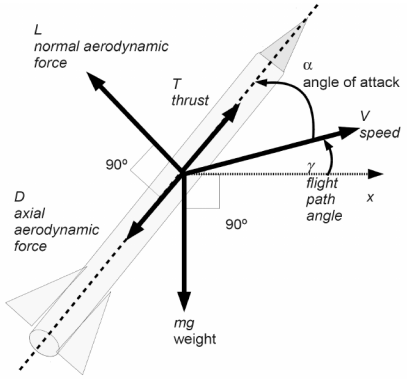

Fig. 1. Ilustration of the variables associated with the missile model

The equations of motion of the missile are given by:

$$\dot{\gamma} = \frac{T-D}{mg}\sin\alpha + \frac{L}{mV}\cos\alpha - \frac{g\cos\gamma}{V}$$
$$\dot{V} = \frac{T-D}{m}\cos\alpha - \frac{L}{m}\sin\alpha - g\cos\gamma$$
$$\dot{x} = V\cos\gamma$$
$$\dot{h} = V\sin\gamma$$

where $D = \frac{1}{2}C_d\rho V^2 S_{\text{ref}}$, $C_d = A_1\alpha^2 + A_2\alpha + A_3$, $L = \frac{1}{2}C_l\rho V^2 S_{\text{ref}}$, $C_l = B_1\alpha + B_2$, $\rho = C_1 h^2 + C_2 h + C_3$.

All the model parameters are given in Table II. The initial conditions for the state variables are: $\gamma(0) = 0$, $V(0) = 272\text{m/s}$, $x(0) = 0\text{m}$, $h(0) = 30\text{m}$. The terminal conditions on the states are: $\gamma(t_f) = -\pi/2$, $V(t_f) = 310\text{m/s}$, $x(t_f) = 10000\text{m}$, $h(t_f) = 0\text{m}$. The problem constraints are given by: $200 \le V \le 310$, $1000 \le T \le 6000$, $-0.3 \le \alpha \le 0.3$, $-4 \le \frac{L}{mg} \le 4$, $h \ge 30$ (for $x \le 7500\text{m}$), $h \ge 0$ (for $x > 7500\text{m}$).

Note that the path constraints on the altitude are non-smooth. Given that non-smoothness causes problems with nonlinear programming, the constraints on the altitude were approximated by a single smooth constraint:

$$\mathcal{H}_\epsilon(x - 7500))h(t) + [1 - \mathcal{H}_\epsilon(x - 7500)][h(t) - 30] \ge 0$$

where $\mathcal{H}_\epsilon(z)$ is a smooth version of the Heaviside function, which is computed as follows:

$$\mathcal{H}_\epsilon(z) = 0.5(1 + \tanh(z/\epsilon))$$

where $\epsilon > 0$ is a small number.

As as illustration of the code that is used to implement a model, the `dae()` function for this problem is given below. Table III lists and describes the parameters used by the interface functions. Note the use of the `adouble` type for dependent and indenpendent variables, as opposed to the standard `double` type used for constants and variables that only depend on constants. The `adouble` type is defined by the ADOL-C library, and it enables the automatic differentiation of the code.

```
void dae(adouble* derivatives, adouble* path, adouble* states,
         adouble* controls, adouble* parameters, adouble& time,
         adouble* xad, int iphase)
{

    adouble T  = controls[ CINDEX(1) ]; // Thrust
    adouble alpha       = controls[ CINDEX(2) ]; // Angle of attack
    adouble gamma  = states[ CINDEX(1) ]; // flight path angle
    adouble V  = states[ CINDEX(2) ]; // speed
    adouble x = states[ CINDEX(3) ]; // horizontal position
    adouble h  = states[ CINDEX(4) ]; // altitude
    double  m = 1005.0;  // kg
    double  g   = 9.81;    // m/s^2
    double  Sref   = 0.3376;  // m^2
    double  A1 = -1.9431;
    double  A2  = -0.1499;
    double  A3  = 0.2359;
    double  B1  = 21.9;
    double  B2 = 0.0;
    double  C1   = 3.312e-9;
    double  C2          = -1.142e-4;
    double  C3   = 1.224;
    adouble sina  = sin(alpha);
    adouble cosa  = cos(alpha);
    adouble sing = sin(gamma);
    adouble cosg        = cos(gamma);
    adouble Cd, D, L, Cl, rho;
    rho = C1*h*h + C2*h + C3;  // Air density
    Cd = A1*alpha*alpha + A2*alpha + A3;
    Cl = B1*alpha + B2;
    D = 0.5*Cd*rho*V*V*Sref; // Axial aerodynamic force
    L = 0.5*Cl*rho*V*V*Sref; // Normal aerodynamic force

    derivatives[ CINDEX(1) ]  = (T-D)/(m*V)*sina + L/(m*V)*cosa - g*cosg/V;
    derivatives[ CINDEX(2) ]  = (T-D)/m*cosa - L/m*sina - g*sing;
    derivatives[ CINDEX(3) ]  = V*cosg;
    derivatives[ CINDEX(4) ]  = V*sing;

    path[ CINDEX(1) ]         = L/(m*g);
    adouble H = smooth_heaviside( 7500.0-x, 10.0 );
    path[ CINDEX(2) ]         =  H*(h-30.0)  +  (1.0-H)*h;
}
```

The code to implement the objective function associated with the problem is given below.

```
adouble endpoint_cost(adouble* initial_states, adouble* final_states,
                      adouble* parameters,adouble& t0, adouble& tf,
                      adouble* xad, int iphase)
{
    return tf;
}
```

The problem was solved using trapezoidal discretization, followed by Hermite-Simpson discretization, with local automatic mesh refinement, starting with 20 nodes. The final solution, which is found after six mesh refinement iterations, has 82 nodes. Figure 2 shows the resulting missile altitude as a function of the longitudinal position. Figure 3 shows the angle of attack as a function of time. The resulting minimum time to the target was 40.91193 seconds. Table IV summarises the mesh refinement process.

Parameter	Description
controls	Array of intantaneous controls
derivatives	Array of intantaneous state derivatives
final_states	Array of final states within a phase
initial_states	Array of initial states within a phase
iphase	Phase index (starting from 1)
parameters	Array of static parameters within a phase
states	Array of intantaneous states within a phase
time	Instant of time within a phase
t0	Initial phase time
tf	final phase time
xad	array of scaled decision variables

TABLE III

DESCRIPTION OF PARAMETERS USED BY THE $\mathcal{PSOPT}$ INTERFACE FUNCTIONS

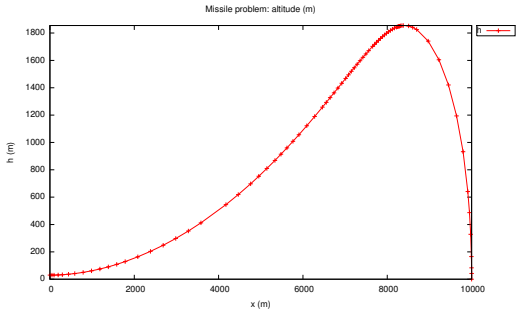

Fig. 2. Missile altitude and a function of the longitudinal position

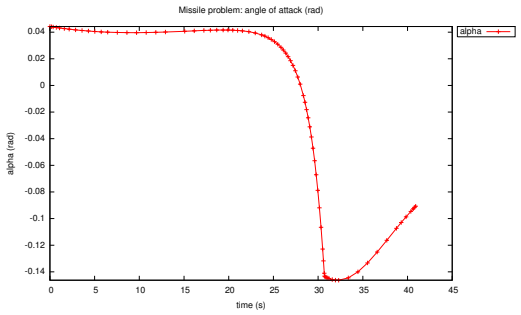

Fig. 3. Missile angle of attack as a function of time

TABLE IV

MESH REFINEMENT STATISTICS: MISSILE PROBLEM

Iter	DM	M	NV	NC	OE	CE	JE	HE	RHS	ϵ_{max}	CPU_a
1	TRP	20	122	129	278	277	17	15	10803	5.922e-03	2.500e-01
2	TRP	28	170	177	33	34	17	15	1870	1.636e-03	2.800e-01
3	H-S	39	312	319	27	28	14	12	3220	4.876e-04	8.700e-01
4	H-S	54	432	439	51	52	26	24	8320	6.517e-05	1.920e+00
5	H-S	75	600	607	23	24	12	10	5352	1.458e-05	1.320e+00
6	H-S	82	656	663	29	30	15	13	7320	9.082e-06	1.880e+00
CPU_b	-	-	-	-	-	-	-	-	-	-	4.060e+00
-	-	-	-	-	441	445	101	89	36885	-	1.058e+01

Key: Iter=iteration number, DM= discretization method, M=number of nodes, NV=number of variables, NC=number of constraints, OE=objective evaluations, CE = constraint evaluations, JE = Jacobian evaluations, HE = Hessian evaluations, RHS = ODE right hand side evaluations, ϵ_{max} = maximum relative ODE error, CPU_a = CPU time in seconds spent by NLP algorithm, CPU_b = additional CPU time in seconds spent by PSOPT

XI. AVAILABILITY AND COMPATIBILITY OF THE SOFTWARE

$\mathcal{PSOPT}$ can be downloaded at no cost from http://www.psopt.org. The software is distributed under the terms of the GNU Lesser General Public License LGPL. $\mathcal{PSOPT}$ is compatible with recent releases of the Ubuntu Linux operating system. Moreover, the software has been ported to be compiled by Microsoft Visual Studio 2008 under Windows XP, Windows Vista or Windows 7.

XII. CONCLUSIONS

This paper has introduced $\mathcal{PSOPT}$, an open source software package intended for the solution of complex optimal control problems. The scope and features of the software have been described, and some of its salient feautures have been discussed. An application example illustrates some of the capabilities of the software.

REFERENCES

[1] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM, 2001.

[2] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang, *Spectral Methods: Fundamentals in Single Domains*. Berlin: Springer-Verlag, 2006.

[3] G. Elnagar, M. A. Kazemi, and M. Razzaghi, "The Pseudospectral Legendre Method for Discretizing Optimal Control Problems," *IEEE Transactions on Automatic Control*, vol. 40, pp. 1793–1796, 1995.

[4] L. T. Biegler, "An overview of simultaneous strategies for dynamic optimization," *Chemical Engineering and Processing: Process Intensification*, vol. 46, pp. 1043–1053, 2007.

[5] C. Canuto, M. Hussaini, A. Q. A., and T. Zang, *Spectral Methods in Fluid Dynamics*. Springer-Verlag, 1988.

[6] I. Ross and F. Fahroo, "Pseudospectral Knotting Methods for Solving Nonsmooth Optimal Control Problems," *Journal of Guidance Control and Dynamics*, vol. 27, pp. 397–405, 2004.

[7] N. Bedrossian, S. Bhatt, W. Kang, and I. Ross, "Zero Propellant Maneuver Guidance," *IEEE Control Systems Magazine*, vol. 29, pp. 53–73, 2009.

[8] J. T. Betts and Huffman, "Sparse Optimal Control Software - SOCS," Boeing Information and Support Services, Seattle, Washington, Tech. Rep. MEA-LR-085, 1997.

[9] O. V. Stryk, *User's guide for DIRCOL (Version 2.1): A direct collocation method for the numerical solution of optimal control problems*. Technical Report, Technische Universitat Munchen, 1999.

[10] P. E. Rutquist and M. M. Edvall, *PROPT Matlab Optimal Control Software*. Västerås, Sweden: TOMLAB Optimization, 2009.

[11] I.M. Ross, "A beginner's guide to DIDO (ver. 7.3): A MATLAB application package for solving optimal control problems," Elissar LLC, Monterey, CA, Tech. Rep. TR-711, 2007.

[12] A. Rao, D. Benson, C. Darby, M. A. Patterson, C. Francolin, I. Sanders, and G. Huntington, "Algorithm 902: GPOPS, a MATLAB software for solving multiple-phase optimal control problems using the Gauss pseudospectral method," *ACM Transactions on Mathematical Software (TOMS)*, vol. 37, 2010.

[13] A. Wächter and L. T. Biegler, "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006.

[14] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for Large-Scale Constrained Optimization," *SIAM Review*, vol. 47, no. 1, 2001.

[15] A. R. Curtis, M. J. D. Powell, and J. K. Reid, "On the Estimation of Sparse Jacobian Matrices," *Journal of the Institute of Mathematics and Applications*, vol. 13, pp. 117–120, 1974.

[16] V. Becerra, *PSOPT Optimal Control Solver User Manual - Release 2. Available: http://psopt.googlecode.com/files/PSOPT_Manual_R2.pdf*, 2010.

[17] S. Subchan and R. Zbikowski, *Computational optimal control: tools and practice*. Wiley, 2009.