A Survey of Numerical Methods for Trajectory Optimization

John T. Betts
Mathematics and Engineering Analysis,
Boeing Information and Support Services,
P.O. Box 3707, MS 7L-21,
Seattle, Washington 98124-2207

August 15, 1998

1 Introduction

It is not surprising that the development of numerical methods for trajectory optimization have closely paralleled the exploration of space and the development of the digital computer. Space exploration provided the *impetus* by presenting scientists and engineers with challenging technical problems. The digital computer provided the *tool* for solving these new problems. The goal of this paper is to review the state of the art in the field loosely referred to as "trajectory optimization."

Presenting a survey of a field as diverse as trajectory optimization is a daunting task. Perhaps the most difficult issue is restricting the *scope* of the survey to permit a meaningful discussion within a limited amount of space. In order to achieve this goal, a conscious decision has been made to focus on the two types of methods most widely used today, namely *direct* and *indirect*. We begin the discussion with a brief review of the underlying mathematics in both direct and indirect methods. We then discuss the complications that occur when path and boundary constraints are imposed on the problem description. Finally we describe unresolved issues that are the subject of ongoing research.

There are a number of recurrent themes appearing throughout the paper. First, the aforementioned *direct* versus *indirect* is introduced as a means of categorizing an approach. Unfortunately not every technique falls neatly into one category or another. We will attempt to describe the benefits and deficiencies in both approachs, and then suggest that the techniques may ultimately "merge" together. Second, we shall attempt to discriminate between *method* versus *implementation*. A numerical method is usually described by mathematical equations and/or algorithmic logic. Computational results are achieved by implementing the algorithm as software (e.g. FORTRAN code). A second level of implementation may involve translating a (preflight) scientific software implementation into an (onboard) hardware implementation. In general, method and implementation are not the same and we shall try to emphasize this fact. Third, we shall focus the discussion on *algorithms* instead of *physical models*. The definition of a trajectory problem necessarily entails a definition of the dynamic environment such as gravitational, propulsion, and aerodynamic forces. Thus it is common to use the same algorithm, with different physical models in order to solve different problems. Conversely different algorithms may be applied to the same physical models (with entirely different results). Finally, we shall attempt to focus on *general* rather than *special* purpose methods. A great deal of research has been directed toward solving specific problems. Carefully specialized techniques can either be very effective or very ad hoc. Unfortunately what works well for a launch vehicle guidance problem many be totally inappropriate for a low-thrust orbit transfer.

2 The Trajectory Optimization Problem

Let us begin the discussion by defining the problem in a fairly general way. A trajectory optimization or optimal control problem can be formulated as a collection of N *phases*. In general, the independent variable t for phase k is defined in the region $t_0^{(k)} \leq t \leq t_f^{(k)}$. For many applications the independent variable t is *time*, and the phases are sequential, that is $t_0^{(k+1)} = t_f^{(k)}$, however neither of these assumptions is required. Within phase k the dynamics of the system are described by a set of *dynamic* variables

$$\mathbf{z} = \begin{bmatrix} \mathbf{y}^{(k)}(t) \\ \mathbf{u}^{(k)}(t) \end{bmatrix} \tag{1}$$

made up of the $n_y^{(k)}$ *state variables* and the $n_u^{(k)}$ *control variables* respectively. In addition, the dynamics may incorporate the $n_p^{(k)}$ *parameters* $\mathbf{p}^{(k)}$ which are not dependent on t. For clarity we drop the phase-dependent notation from the remaining discussion in this section, however it is important to remember that many complex problem descriptions require different dynamics and/or constraints within each phase, and the formulation accomodates this requirement.

Typically the dynamics of the system are defined by a set of ordinary differential equations written in explicit form, which are referred to as the *state or system equations*

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] \tag{2}$$

where $\mathbf{y}$ is the n_y dimension state vector. Initial conditions at time t_0 are defined by

$$\boldsymbol{\psi}_{0\ell} \leq \boldsymbol{\psi}[\mathbf{y}(t_0), \mathbf{u}(t_0), \mathbf{p}, t_0] \leq \boldsymbol{\psi}_{0u}, \tag{3}$$

where $\boldsymbol{\psi}[\mathbf{y}(t_0), \mathbf{u}(t_0), \mathbf{p}, t_0] \equiv \boldsymbol{\psi}_0$ and terminal conditions at the final time t_f are defined by

$$\boldsymbol{\psi}_{f\ell} \leq \boldsymbol{\psi}[\mathbf{y}(t_f), \mathbf{u}(t_f), \mathbf{p}, t_f] \leq \boldsymbol{\psi}_{fu}, \tag{4}$$

where $\boldsymbol{\psi}[\mathbf{y}(t_f), \mathbf{u}(t_f), \mathbf{p}, t_f] \equiv \boldsymbol{\psi}_f$. In addition the solution must satisfy *algebraic path constraints* of the form

$$\mathbf{g}_\ell \leq \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] \leq \mathbf{g}_u, \tag{5}$$

where $\mathbf{g}$ is a vector of size n_g, as well as simple bounds on the state variables

$$\mathbf{y}_\ell \leq \mathbf{y}(t) \leq \mathbf{y}_u, \tag{6}$$

and control variables

$$\mathbf{u}_\ell \leq \mathbf{u}(t) \leq \mathbf{u}_u. \tag{7}$$

Note that an equality constraint can be imposed if the upper and lower bounds are equal, e.g. $(g_\ell)_k = (g_u)_k$ for some k.

Finally it may be convenient to evaluate expressions of the form

$$\int_{t_0}^{t_f} \mathbf{q}[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] dt \tag{8}$$

which involve the *quadrature* functions $\mathbf{q}$. Collectively we refer to those functions evaluated during the phase, namely,

$$\mathbf{F}(t) = \begin{bmatrix} \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] \\ \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] \\ \mathbf{q}[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] \end{bmatrix} \tag{9}$$

as the vector of *continuous functions*. Similarly functions evaluated at a specific point such as the boundary conditions $\boldsymbol{\psi}[\mathbf{y}(t_0), \mathbf{u}(t_0), t_0]$ and $\boldsymbol{\psi}[\mathbf{y}(t_f), \mathbf{u}(t_f), t_f]$ are referred to as *point functions*.

The basic optimal control problem is to determine the $n_u^{(k)}$-dimensional control vectors $\mathbf{u}^{(k)}(t)$ and parameters $\mathbf{p}^{(k)}$ to minimize the performance index

$$\begin{aligned} J = \phi \Big[& \mathbf{y}(t_0^{(1)}), t_0^{(1)}, \mathbf{y}(t_f^{(1)}), \mathbf{p}^{(1)}, t_f^{(1)}, \ldots, \\ & \mathbf{y}(t_0^{(N)}), t_0^{(N)}, \mathbf{y}(t_f^{(N)}), \mathbf{p}^{(N)}, t_f^{(N)} \Big]. \end{aligned} \tag{10}$$

Notice that the objective function may depend on quantities computed in each of the N phases.

This formulation raises a number of points which deserve further explanation. The concept of a phase, also referred to as an *arc* by some authors, partitions the time domain. In this formalism the differential equations cannot change within a phase, but may change from one phase to another. An obvious reason to introduce a phase is to accomodate changes in the dynamics, for example when simulating a multi-stage rocket. The boundary of a phase is often called an *event* or *junction point*. A boundary condition which uniquely defines the end of a phase is sometimes called an *event criterion* (and a well-posed problem can have only one criterion at each event). Normally, the simulation of a complicated trajectory may link phases together by forcing the states to be continuous, e.g. $\mathbf{y}(t_f^{(1)}) = \mathbf{y}(t_0^{(2)})$. However, for multi-path or branch trajectories this may not be the case.[1] The differential equations (2) have been written as an *explicit* system of first order equations (i.e. with the first derivative appearing explicitly on the left hand side) which is the standard convention for aerospace applications. While this simplifies the presentation it may not be either necessary or desirable, e.g. Newton's law $\mathbf{F} = m\mathbf{a}$ is not stated as an explicit first-order system! The objective function (10) has been written in terms of quantities evaluated at the ends of the phases and this is referred to as the *Mayer* form.[2] If the objective function only involves an integral (8) it is referred to as a problem of *Lagrange*, and when both terms are present it is called a problem of *Bolza*. It is well known that the Mayer form can be obtained from either the Lagrange or Bolza form by introducing an additional state variable, however, again this may have undesirable numerical consequences.

3 Nonlinear Programming

3.1 Newton's Method

Essentially all numerical methods for solving the trajectory optimization problem incorporate some type of iteration with a finite set of unknowns. In fact progress in optimal control solution methods closely parallels the progress made in the underlying nonlinear programming (NLP) methods. Space limitations preclude an in depth presentation of constrained optimization methods, however, it is important to review some of the fundamental concepts. For more complete information the reader is encouraged to refer to the books by Fletcher,[3] Gill, Murray, and Wright,[4] and Dennis and Schnabel.[5] The fundamental approach to most iterative schemes was suggested over 300 years ago by Newton. Suppose we are trying to solve the nonlinear algebraic equations $\mathbf{a}(\mathbf{x}) = \mathbf{0}$ for the root $\mathbf{x}^*$. Beginning with an estimate $\mathbf{x}$ we can construct a new estimate $\overline{\mathbf{x}}$ according to

$$\overline{\mathbf{x}} = \mathbf{x} + \alpha \mathbf{p} \tag{11}$$

where the *search direction* $\mathbf{p}$ is computed by solving the linear system

$$\mathbf{A}(x)\mathbf{p} = -\mathbf{a}(x). \tag{12}$$

The $n \times n$ matrix $\mathbf{A}$ is defined by

$$\mathbf{A} = \begin{bmatrix} \frac{\partial a_1}{\partial x_1} & \frac{\partial a_1}{\partial x_2} & \cdots & \frac{\partial a_1}{\partial x_n} \\ \frac{\partial a_2}{\partial x_1} & \frac{\partial a_2}{\partial x_2} & \cdots & \frac{\partial a_2}{\partial x_n} \\ \vdots & & \ddots & \\ \frac{\partial a_n}{\partial x_1} & \frac{\partial a_n}{\partial x_2} & \cdots & \frac{\partial a_n}{\partial x_n} \end{bmatrix}. \tag{13}$$

When the scalar *step length* α is equal to one the iteration scheme is equivalent to replacing the nonlinear equation by a linear approximation constructed about the point $\mathbf{x}$. We expect the method to converge provided the initial guess is "close" to the root $\mathbf{x}^*$. Of course this simple scheme is not without pitfalls. First, in order to compute the search direction the matrix $\mathbf{A}$ must be nonsingular (invertible) and for arbitrary nonlinear functions $\mathbf{a}(x)$ this may not be true. Second, when the initial guess is not "close" to the root, the iteration may diverge. One common way to stabilize the iteration is to reduce the length of the step by choosing α such that

$$\|\mathbf{a}(\overline{\mathbf{x}})\| \leq \|\mathbf{a}(\mathbf{x})\|. \tag{14}$$

The procedure for adjusting the steplength is called a *linesearch*, and the function used to measure progress (in this case $\|\mathbf{a}\|$) is called a *merit function.* In spite of the need for caution, Newton's method enjoys broad applicability, possibly because when it works the iterates exhibit *quadratic convergence.* Loosely speaking this property means that the number of significant figures in $\mathbf{x}$ (as an estimate for $\mathbf{x}^*$) doubles from one iteration to the next.

3.2 Unconstrained Optimization

Let us now consider an unconstrained optimization problem. Suppose that we must choose the n variables $\mathbf{x}$ to minimize the scalar objective function $F(\mathbf{x})$. Necessary conditions for $\mathbf{x}^*$ to be a stationary point are

$$\mathbf{g}(\mathbf{x}^*) = \nabla_x F = \begin{bmatrix} \frac{\partial F}{\partial x_1} \\ \frac{\partial F}{\partial x_2} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{bmatrix} = \mathbf{0}. \tag{15}$$

Now if Newton's method is used to find a point where the *gradient* (15) is zero, we must compute the search direction using

$$\mathbf{H}(x)\mathbf{p} = -\mathbf{g}(x). \tag{16}$$

where the *Hessian* matrix $\mathbf{H}$ is the symmetric matrix of second derivatives of the objective function. Just as before, there are pitfalls in using this method to construct an estimate of the solution. First, we note that the condition $\mathbf{g} = \mathbf{0}$ is necessary but not sufficient. Thus a point with zero gradient can either be a maximum or a minimum. At a minimum point the Hessian matrix is positive definite but this may not be true when $\mathbf{H}$ is evaluated at some point far from the solution. In fact it is quite possible that the direction $\mathbf{p}$ computed by solving (16) will point uphill rather than downhill. Second, there is some ambiguity in the choice of a merit function, if a linesearch is used to stabilize the method. Certainly we would hope to reduce the objective function, that is $F(\overline{\mathbf{x}}) \leq F(\mathbf{x})$, however this may not produce a decrease in the gradient

$$\|\mathbf{g}(\overline{\mathbf{x}})\| \leq \|\mathbf{g}(\mathbf{x})\|. \tag{17}$$

In fact what has been described are two issues that distiguish a *direct* and an *indirect* method for finding a minimum. For an indirect method a logical choice for the merit function is $\|\mathbf{g}(\mathbf{x})\|$. In constrast, for a direct method one probably would insist that the objective function is reduced at each iteration and to achieve this it may be necessary to modify the calculation of the search direction to ensure that it is downhill. A consequence of this is that the region of convergence for an indirect method may be considerably smaller than the region of convergence for a direct method. Stated differently an indirect method may require a better initial guess, than required by a direct method. Secondly in order to solve the equations $\mathbf{g} = \mathbf{0}$ it is necessary to compute the expressions $\mathbf{g}(\mathbf{x})$! Typically this implies that analytic expressions for the gradient are necessary when using an indirect method. In contrast finite difference approximations to the gradient are often used in a direct method.

3.3 Equality Constraints

Suppose that we must choose the n variables $\mathbf{x}$ to minimize the scalar objective function $F(\mathbf{x})$ and satisfy the m equality constraints

$$\mathbf{c}(\mathbf{x}) = \mathbf{0} \tag{18}$$

where $m \leq n$. We introduce the *Lagrangian*

$$L(\mathbf{x}, \boldsymbol{\lambda}) = F(\mathbf{x}) - \boldsymbol{\lambda}^\top \mathbf{c}(\mathbf{x}) \tag{19}$$

which is a scalar function of the n variables $\mathbf{x}$ and the m *Lagrange multipliers* $\boldsymbol{\lambda}$. Necessary conditions for the point $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ to be a constrained optimum require finding a stationary point of the Lagrangian which is defined by

$$\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{g}(\mathbf{x}) - \mathbf{G}^\top(\mathbf{x})\boldsymbol{\lambda} = \mathbf{0} \tag{20}$$

and

$$\nabla_\lambda L(\mathbf{x}, \boldsymbol{\lambda}) = -\mathbf{c}(\mathbf{x}) = \mathbf{0}. \tag{21}$$

By analogy with the development in the preceding sections, we can use Newton's method to find the $(n+m)$ variables $(\mathbf{x}, \boldsymbol{\lambda})$ that satisfy the conditions (20) and (21). Proceeding formally to construct the linear system equivalent to (12) one obtains

$$\begin{bmatrix} \mathbf{H}_L & \mathbf{G}^\top \\ \mathbf{G} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ -\overline{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} -\mathbf{g} \\ -\mathbf{c} \end{bmatrix}. \tag{22}$$

This system requires the Hessian of the Lagrangian

$$\mathbf{H}_L = \nabla_x^2 F - \sum_{i=1}^{m} \lambda_i \nabla_x^2 c_i. \tag{23}$$

The linear system (22) is referred to as the Kuhn-Tucker (KT) or Karush-Kuhn-Tucker (KKT) system. It is important to observe that an equivalent way of defining the search direction $\mathbf{p}$ is to minimize the quadratic

$$\frac{1}{2}\mathbf{p}^\top \mathbf{H}_L \mathbf{p} + \mathbf{g}^\top \mathbf{p} \tag{24}$$

subject to the linear constraints

$$\mathbf{G}\mathbf{p} = -\mathbf{c}. \tag{25}$$

This is referred to as a *quadratic programming* (QP) subproblem. Just as in the unconstrained and root solving applications discussed above, when Newton's method is applied to equality constrained problems it may be necessary to modify either the magnitude of the step via a linesearch or the direction itself using a "trust region" approach. However regardless of the type of stablization invoked at points far from the solution, near the answer all methods try to mimic the behavior of Newton's method.

3.4 Inequality Constraints

An important generalization of the previous problem occurs when inequality constraints are imposed. Suppose that we must choose the n variables $\mathbf{x}$ to minimize the scalar objective function $F(\mathbf{x})$ and satisfy the m inequality constraints

$$\mathbf{c}(\mathbf{x}) \geq \mathbf{0}. \tag{26}$$

In contrast to the equally constrained case, now m may be greater than n. However, at the optimal point $\mathbf{x}^*$, the constraints will fall into one of two classes. Constraints that are strictly satisfied, i.e. $c_i(\mathbf{x}^*) > 0$ are called *inactive*. The remaining *active* constraints are on their bounds, i.e. $c_i(\mathbf{x}^*) = 0$. If the *active set* of constraints is known, then one can simply ignore the remaining constraints and treat the problem using methods for an equality constrained problem. However algorithms to efficiently determine the active set of constraints are nontrivial since they require repeated solution of the KT system (22) as constraints are added and deleted. In spite of the complications, methods for nonlinear programming based on the solution of a series of quadratic programming subproblems are widely used. A popular implementation of the so-called successive or sequential quadratic programming (SQP) approach is found in the software NPSOL.[6,7]

In summary the *nonlinear programming* or NLP problem requires finding the n-vector $\mathbf{x}$ to minimize

$$F(\mathbf{x}) \tag{27}$$

subject to the m constraints

$$\mathbf{c}_L \leq \mathbf{c}(\mathbf{x}) \leq \mathbf{c}_U \tag{28}$$

and bounds

$$\mathbf{x}_L \leq \mathbf{x} \leq \mathbf{x}_U. \tag{29}$$

In this formulation equality constraints can be imposed by setting $c_L = c_U$.

3.5 Historical Perspective

Progress in the development of nonlinear programming algorithms has been closely tied to the advent of the digital computer. Since NLP software is a primary piece of the trajectory optimization toolkit, it has been a pacing item in the development of sophisticated trajectory optimization software. In the early 1960's most implementations were based on a simple Newton method (11), (12) with optimization done "parametrically" (i.e. by hand). The size of a typical application was $n = m \approx 10$. In the 1970's quasi-Newton approximations[3,5] became prevalent. One popular approach for dealing with constraints was to apply an unconstrained minimization algorithm to a modified form of the objective, e.g. minimize $J(\mathbf{x}, \rho) = F(\mathbf{x}) + \frac{1}{2}\rho\mathbf{c}(\mathbf{x})^{\top}\mathbf{c}(\mathbf{x})$, where ρ is "large." Although these techniques have generally been superceded for general optimization, curiously enough they are fundamental to the definition of the merit functions used to stabilize state of the art algorithms. A second popular approach for constrained problems referred to as the "reduced gradient" approach identifies a "basic" set of variables which are used to eliminate the active constraints, permitting choice of the "nonbasic" variables using an unconstrained technique. Careful implementations of this method[8–10] can be quite effective, especially when the constraints are nearly linear, and the number of inequalities is small. Most applications in the 1970's and early 1980's were of moderate size, i.e. $n = m < 100$. Current applications have incorporated advances in numerical linear algebra which exploit matrix sparsity, thereby permitting applications with $n, m \approx 100000$.[11–20]

4 Optimal Control

4.1 Dynamic Constraints

The optimal control problem may be interpreted as an extension of the nonlinear programming problem to an infinite number of variables. For fundamental background in the associated *calculus of variations* the reader should refer to Bliss.[21] First let us consider a simple problem with a single phase and no path constraints. Specifically suppose we must choose the control functions $\mathbf{u}(t)$ to minimize

$$J = \phi\left[\mathbf{y}(t_f), t_f\right] \tag{30}$$

subject to the state equations

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)] \tag{31}$$

and the boundary conditions

$$\boldsymbol{\psi}[\mathbf{y}(t_f), \mathbf{u}(t_f), t_f] = \mathbf{0}, \tag{32}$$

where the initial conditions $\mathbf{y}(t_0) = \mathbf{y}_0$ are given at the fixed initial time t_0 and the final time t_f is free. Note this is a very simplified version of the problem (2)–(10), and we have purposely chosen a problem with only equality constraints. However, in contrast to the previous discussion we now have a "continuous" equality constraint (31) as well as a "discrete" equality (32). In a manner analogous to the definition of the Lagrangian function (19) we form an augmented performance index

$$\hat{J} = \left[\phi + \boldsymbol{\nu}^\top \boldsymbol{\psi}\right]_{t_f} + \int_{t_0}^{t_f} \boldsymbol{\lambda}^\top(t) \left\{\mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)] - \dot{\mathbf{y}}\right\} dt. \tag{33}$$

Notice that in addition to the Lagrange multipliers $\boldsymbol{\nu}$ for the discrete constraints we also have multipliers $\boldsymbol{\lambda}(t)$ referred to as *adjoint or costate variables* for the continuous (differential equation) constraints. In the finite dimensional case, the necessary conditions for a constrained optimum (20) and (21) were obtained by setting the first derivatives of the Lagrangian to zero. The analogous operation is to set the *first variation* $\delta\hat{J} = 0$. It is convenient to define the *Hamiltonian*

$$H = \boldsymbol{\lambda}^\top(t)\mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)] \tag{34}$$

and the auxiliary function

$$\Phi = \phi + \boldsymbol{\nu}^\top \boldsymbol{\psi}. \tag{35}$$

The necessary conditions referred to as the *Euler-Lagrange equations* which result from setting the first variation to zero in addition to (31) and (32) are

$$\dot{\boldsymbol{\lambda}} = -\mathbf{H_y}^\top \tag{36}$$

called the *adjoint equations*,

$$\mathbf{0} = \mathbf{H_u}^\top \tag{37}$$

called the *control equations*, and

$$\boldsymbol{\lambda}(t_f) = \left.\boldsymbol{\Phi_y}^\top\right|_{t=t_f} \tag{38}$$

$$0 = \left.(\Phi_t + H)\right|_{t=t_f} \tag{39}$$

$$\mathbf{0} = \boldsymbol{\lambda}(t_0) \tag{40}$$

called the *transversality conditions.* The partial derivatives $\mathbf{H_y}$, $\mathbf{H_u}$ and $\boldsymbol{\Phi_y}$ are considered row vectors, i.e. $\mathbf{H_y} \doteq (\partial H/\partial y_1, \ldots, \partial H/\partial y_n)$ in these expressions. The control equations (37) are an application of the *Pontryagin Maximum Principle.*[22] A more general expression is

$$\mathbf{u} = \arg\min_{\mathbf{u}\in U} \mathbf{H} \tag{41}$$

where U defines the domain of feasible controls. Note that (41) is really a "minimum" principle in order to be consistent with the algebraic sign conventions used elsewhere. The maximum principle states that the control variable must be chosen to optimize the Hamiltonian (at every instant in time), subject to limitations on the control imposed by state and control path constraints. In essence the maximum principle is a constrained optimization problem in the variables $\mathbf{u}(t)$ at all values of t. The complete set of necessary conditions consists of a differential-algebraic (DAE) system (31),(36), and (37) with boundary conditions at both t_0 and t_f (38),(39), and (32). This is often referred to as a *two-point boundary value problem.* A more extensive presentation of this material can be found in Bryson and Ho.[23]

4.2 Algebraic Equality Constraints

Generalizing the problem in the previous section, let us assume that we impose algebraic path constraints of the form

$$\mathbf{0} = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t] \tag{42}$$

in addition to the other conditions (31), (32). Using notation similar to the preceding section let us define the matrix

$$\mathbf{g}_u = \begin{bmatrix} \frac{\partial g_1}{\partial u_1} & \frac{\partial g_1}{\partial u_2} & \cdots & \frac{\partial g_1}{\partial u_n} \\ \frac{\partial g_2}{\partial u_1} & \frac{\partial g_2}{\partial u_2} & \cdots & \frac{\partial g_2}{\partial u_n} \\ \vdots & & \ddots & \\ \frac{\partial g_n}{\partial u_1} & \frac{\partial g_n}{\partial u_2} & \cdots & \frac{\partial g_n}{\partial u_n} \end{bmatrix}. \tag{43}$$

Two possibilities exist. If the matrix $\mathbf{g}_u$ is full rank then the system of differential and algebraic equations (31), (42) is referred to as a DAE of *index 1*, and (42) is termed a *control variable equality constraint.* For this case the Hamiltonian (34) is replaced by

$$H = \boldsymbol{\lambda}^\top \mathbf{f} + \boldsymbol{\mu}^\top \mathbf{g} \tag{44}$$

which will result in modification to both the adjoint equations (36) and the control equations (37).

The second possibility is that the matrix $\mathbf{g}_u$ is rank deficient. In this case we can differentiate (42) with respect to t yielding

$$\begin{aligned} 0 &= \mathbf{g}_y \dot{\mathbf{y}} + \mathbf{g}_u \dot{\mathbf{u}} + \mathbf{g}_t && (45) \\ &= \mathbf{g}_y \mathbf{f}[\mathbf{y}, \mathbf{u}] + \mathbf{g}_u \dot{\mathbf{u}} + \mathbf{g}_t && (46) \\ &\doteq \mathbf{g}'[\mathbf{y}(t), \mathbf{u}(t), t], && (47) \end{aligned}$$

where the second step follows by substituting the definition (31) and changing the definition of $\mathbf{y}$ and $\mathbf{u}$. The result is a new path constraint function $\mathbf{g}'$ which is mathematically equivalent provided that the original constraint is imposed at some point on the path say $0 = \mathbf{g}[\mathbf{y}(t_0), \mathbf{u}(t_0), t_0]$. For this new path function again the matrix $\mathbf{g}'_u$ may be full rank or rank deficient. If the matrix is full rank, the original DAE system is said to have *index 2* and this is referred to as a state variable constraint of order 1. In the rank deficient case we may redefine the Hamiltonian using $\mathbf{g}'$ in place of $\mathbf{g}$. Of course if the matrix $\mathbf{g}'_u$ is rank deficient the process must be repeated. This is referred to as *index reduction* in the DAE literature.[24, 25] It is important to note that index reduction may be difficult to perform and imposition of a high-index path constraint may be prone to numerical error.

4.3 Singular Arcs

In the preceding section we addressed the DAE system

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{u}, t] \tag{48}$$

$$\mathbf{0} = \mathbf{g}[\mathbf{y}, \mathbf{u}, t] \tag{49}$$

which can appear when path constraints are imposed on the optimal control problem. However, even in the absence of path constraints the necessary conditions (31),(36),(37) lead to the DAE system

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{u}, t] \tag{50}$$

$$\dot{\boldsymbol{\lambda}} = -\mathbf{H_y}^\top \tag{51}$$

$$\mathbf{0} = \mathbf{H_u}^\top. \tag{52}$$

Viewed as a system of DAE's one expects the optimality condition $\mathbf{0} = \mathbf{H_u}^\top$ to define the control variable provided the matrix $\mathbf{H_{uu}}$ is nonsingular. On the other hand if $\mathbf{H_{uu}}$ is a singular matrix, the control $\mathbf{u}$ is not uniquely defined by the optimality condition. This situation is referred to as a *singular arc*, and the analysis of this problem involves techniques quite similar to those discussed above for path constraints. Furthermore singular arc problems are not just mathematical curiosities since $\mathbf{H_{uu}}$ is singular whenever $\mathbf{f}[\mathbf{y}, \mathbf{u}, t]$ is a linear function of $\mathbf{u}$. The famous sounding rocket problem proposed by Robert Goddard in 1919[26] contains a singular arc. Recent interest in periodic optimal flight,[27, 28] and the analysis of wind shear during landing[29] all involve formulations with singular arcs.

4.4 Algebraic Inequality Constraints

The preceding sections have addressed the treatment of equality path constraints. Let us now consider inequality path constraints of the form

$$\mathbf{0} \leq \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{53}$$

Unlike an equality constraint which must be satisfied for all $t_0 \leq t \leq t_f$, inequality constraints may either be active $\mathbf{0} = \mathbf{g}$ or inactive $\mathbf{0} < \mathbf{g}$ at each instant in time. In essence the time domain is partitioned into constrained and unconstrained subarcs. During the unconstrained arcs the necessary conditions are given by (31),(36), and (37), whereas the conditions with modified Hamiltonian (44) are applicable in the constrained arcs. Thus the imposition of inequality constraints presents three major complications. First, the <u>number</u> of constrained subarcs present in the optimal solution are not known a priori. Second, the <u>location</u> of the *junction points* when the transition from constrained to unconstrained (and vice-versa) occurs is unknown. Finally, at the junction points it is possible that both the control variables $\mathbf{u}$ and the adjoint variables $\boldsymbol{\lambda}$ are discontinous. Additional *jump conditions* which are essentially boundary conditions imposed at the junction points must be satisfied. Thus what was a two point boundary value problem, may become a multi-point boundary value problem when inequalities are imposed. For a more complete discussion of this subject the reader is referred to the tutorial by Pesch[30] and the text by Bryson and Ho.[23]

4.5 NLP vs Optimal Control

To conclude the discussion let us reemphasize the relationship between optimal control and nonlinear programming problems with a simple example. Suppose we must choose the control functions $\mathbf{u}(t)$ to minimize

$$J = \phi\left[\mathbf{y}(t_f), t_f\right] \tag{54}$$

subject to the state equations

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)] \tag{55}$$

$\mathbf{y}(t_0) = \mathbf{y}_0$ are given at the fixed initial and final times t_0 and t_f. Let us define NLP variables

$$\mathbf{x} = (\mathbf{u}_0, \mathbf{y}_1, \mathbf{u}_1, \mathbf{y}_2, \mathbf{u}_2, \ldots, \mathbf{y}_M, \mathbf{u}_M) \tag{56}$$

as the values of the state and control evaluated at $t_0, t_1, \ldots, t_M$ where $t_k = t_{k-1} + h$ with $h = t_f/M$. Now

$$\dot{\mathbf{y}} \approx \frac{\mathbf{y}_k - \mathbf{y}_{k-1}}{h}. \tag{57}$$

Let us substitute this approximation into (55) thereby defining the NLP constraints

$$c_k(\mathbf{x}) = \mathbf{y}_k - \mathbf{y}_{k-1} - h\mathbf{f}(\mathbf{y}_{k-1}, \mathbf{u}_{k-1}) \tag{58}$$

for $k = 1, \ldots, M$, and NLP objective function

$$F(\mathbf{x}) = \phi(\mathbf{y}_M) \tag{59}$$

The problem defined by (56), (58), and (59) is a nonlinear program. From (19) the Lagrangian is

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\lambda}) = F(\mathbf{x}) - \boldsymbol{\lambda}^\top \mathbf{c}(\mathbf{x}) &= \phi(\mathbf{y}_M) \\ &- \sum_{k=1}^{M} \boldsymbol{\lambda}_k^\top \left[\mathbf{y}_k - \mathbf{y}_{k-1} - h\mathbf{f}(\mathbf{y}_{k-1}, \mathbf{u}_{k-1})\right] \end{aligned} \tag{60}$$

The necessary conditions for this problem follow directly from the definitions (20) and (21):

$$\frac{\partial L}{\partial \boldsymbol{\lambda}_k} = \mathbf{y}_k - \mathbf{y}_{k-1} - h\mathbf{f}(\mathbf{y}_{k-1}, \mathbf{u}_{k-1}) = 0 \tag{61}$$

$$\frac{\partial L}{\partial \mathbf{y}_k} = (\boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_k) + h\boldsymbol{\lambda}_{k+1}^\top \frac{\partial \mathbf{f}}{\partial \mathbf{y}_k} = 0 \tag{62}$$

$$\frac{\partial L}{\partial \mathbf{u}_k} = h\boldsymbol{\lambda}_{k+1}^\top \frac{\partial \mathbf{f}}{\partial \mathbf{u}_k} = 0 \tag{63}$$

$$\frac{\partial L}{\partial \mathbf{y}_M} = -\boldsymbol{\lambda}_M + \frac{\partial \phi}{\partial \mathbf{y}_M} = 0 \tag{64}$$

Now let us consider the limiting form of this problem as $M \to \infty$ and $h \to 0$. Clearly in the limit equation (61) becomes the state equation (31), equation (62) becomes the adjoint equation (36), equation (63) becomes the control equation (37), and equation (64) becomes the transversality condtion (38). Essentially what has been demonstrated is that the NLP necessary conditions (i.e. Kuhn-Tucker) approach the optimal control necessary conditions as the number of variables grows. The NLP Lagrange multipliers can be interpreted as discrete approximations to the optimal control adjoint variables. While this discussion is of theoretical importance it also suggests a number of ideas which are the basis of modern numerical methods. In particular if the analysis is extended to inequality constrained problems, it is apparent that the task of identifying the NLP active set is equivalent to defining constrained subarcs and junction points in the optimal control setting. Early results on this "transcription" process can be found in Canon, Cullum, and Polak,[31] Polak,[32] and Tabak and Kuo,[33] and more recently interest has focused on using alternate methods of discretization.[34–37]

5 Numerical Analysis

5.1 Initial Value Problems

The numerical solution of the initial value problem (IVP) for ordinary differential equations (ODE) is fundamental to most trajectory optimization methods. The problem can be stated as follows: compute the value of $\mathbf{y}(t_f)$ for some value of $t_0 < t_f$ which satisfies

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), t] \tag{65}$$

with the known initial value $\mathbf{y}(t_0) = \mathbf{y}_0$. Notice that unlike the state equations (2) the right hand side of these equations do not explicitly involve either the controls $\mathbf{u}(t)$ or the parameters $\mathbf{p}$. This distinction is extremely important in the context of trajectory optimization since this requires that the control is completely determined by specifying the state, i.e. it implies that we can write $\mathbf{u}(t) = \tilde{\mathbf{g}}[\mathbf{y}(t), \mathbf{p}, t]$. Numerical methods for solving the ODE IVP are relatively mature in comparison to the other fields in trajectory optimization.

Most schemes can be classified as *one step* or *multistep* methods. A popular family of one-step methods are the *Runge-Kutta* schemes,

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \sum_{j=1}^{k} \beta_j \mathbf{f}_{ij} \tag{66}$$

where

$$\mathbf{f}_{ij} = \mathbf{f}\left[\left(\mathbf{y}_i + h_i \sum_{\ell=1}^{k} \alpha_{j\ell} \mathbf{f}_{i\ell}\right), (t_i + h_i \rho_j)\right] \tag{67}$$

for $1 \le j \le k$ and k is referred to as the "stage". In these expressions $\{\rho_j, \beta_j, \alpha_{j\ell}\}$ are known constants with $0 \le \rho_1 \le \rho_2 \le \ldots, \le 1$. The schemes are called *explicit* if $\alpha_{j\ell} = 0$ for $l \ge j$, and *implicit* otherwise. A convenient way to define the coefficients is to use the so-called *Butcher diagram*

$$\begin{array}{c|ccc} \rho_1 & \alpha_{11} & \dots & \alpha_{1k} \\ \vdots & \vdots & & \vdots \\ \rho_k & \alpha_{k1} & \dots & \alpha_{kk} \\ \hline & \beta_1 & \dots & \beta_k \end{array}$$

Four common examples of k-stage Runge-Kutta schemes are:

Euler's Explicit, $k = 1$

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

Classical Runge-Kutta Explicit, $k = 4$

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Trapezoidal Implicit, $k = 2$

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}$$

Hermite-Simpson Implicit, $k = 3$

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 5/24 & 1/3 & -1/24 \\ 1 & 1/6 & 2/3 & 1/6 \\ \hline & 1/6 & 2/3 & 1/6 \end{array}$$

An obvious appeal of an explicit scheme is that the computation of each integration step can be performed without iteration, that is given the value $\mathbf{y}_i$ at the time t_i the value $\mathbf{y}_{i+1}$ at the new time t_{i+1} follows directly from available values of the right hand side functions $\mathbf{f}$. In contrast, for an implicit method the unknown value $\mathbf{y}_{i+1}$ appears nonlinearly, e.g. the trapezoidal method requires

$$0 = \mathbf{y}_{i+1} - \mathbf{y}_i - \frac{h_i}{2}\left[\mathbf{f}(\mathbf{y}_{i+1}, t_{i+1}) + \mathbf{f}(\mathbf{y}_i, t_i)\right] \doteq \boldsymbol{\zeta}_i \tag{68}$$

Consequently to compute $\mathbf{y}_{i+1}$ given the values t_{i+1}, $\mathbf{y}_i$, t_i, and $\mathbf{f}[\mathbf{y}_i, t_i]$ requires solving the nonlinear expression (68) to drive the *defect* $\boldsymbol{\zeta}_i$ to zero. The iterations required to solve this equation are called *corrector* iterations. An initial guess to begin the iteration is usually provided by the so-called *predictor* step. There is considerable latitude in the choice of predictor and corrector schemes. For some well-behaved differential equations a single predictor and corrector step are adequate. In contrast, it may be necessary to perform multiple corrector iterations, e.g. using Newton's method, especially when the differential equations are *stiff*. To illustrate this, suppose that instead of (65) the system dynamics are described by

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t] \\ \epsilon \dot{\mathbf{u}} &= \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t] \end{aligned} \tag{69}$$

where ϵ is a "small" parameter. Within a very small region $0 \le t \le t_\epsilon$ the solution displays a rapidly changing behavior, and thereafter the second equation can effectively be replaced by its limiting form

$$\mathbf{0} = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{70}$$

The *singular perturbation* problem (69) is in fact a stiff system of ODE's, and in the limit approaches a DAE system. Techniques designed specifically for solving singular perturbation formulations have been suggested for guidance applications.[38, 39]

The second class of integration schemes are termed *multistep* schemes and have the general form

$$\mathbf{y}_{i+k} = \sum_{j=0}^{k-1} \alpha_j \mathbf{y}_{i+j} + h \sum_{j=0}^{k} \beta_j \mathbf{f}_{i+j} \tag{71}$$

where α_j and β_j are known constants. If $\beta_k = 0$ then the method is explicit, otherwise it is implicit. The *Adams schemes* are members of the multistep class which are based on approximating the functions $\mathbf{f}(t)$ by interpolating polynomials. The Adams-Bashforth method is an explicit multistep method,[40] whereas the Adams-Moulton method is implicit.[41] Multistep methods must address three issues the we have not discussed for single step methods. First, as written the method requires information at $(k-1)$ previous points. Clearly, this implies some method must be used to start the process, and one common technique is to take one or more steps with a one step method (e.g. Euler). Second, as written, the multistep formula assumes the stepsize h is a fixed value. When the stepsize is allowed to vary, careful implementation is necessary to insure that the calculation of the coefficients is both efficient and well conditioned. Finally, similar remarks apply when the number of steps k (i.e. the order) of the method are changed.

Regardless of whether a one-step or multistep method is utilized, a successful implementation must address the accuracy of the solution. How well does the discrete solution $\mathbf{y}_i$ for $i = 0, 1, \ldots, M$ produced by the integration scheme agree with the "real" answer $\mathbf{y}(t)$? All well-implemented schemes have some mechanism for adjusting the integration stepsize and/or order to control the integration error. The reader is urged to consult the works of Dahlquist and Björk,[42] Stoer and Bulirsch,[43] Hindmarsh,[44] Shampine and Gordon[45] and Gear[46] for additional information. It is also worth noting that a great deal of discussion has been given to the distinction between explicit and implicit methods. Indeed it is often tempting to use an explicit method simply because it is more easily implemented (and understood?) However, the trajectory optimization problem is a boundary value problem (BVP) not an initial value problem, and "... for a boundary value problem ... any scheme becomes effectively, implicit. Thus, the distinction between explicit and implicit initial value schemes becomes less important in the BVP context." [24, p. 69]

Methods for solving initial value problems when dealing with a system of differential-algebraic equations have appeared more recently. For a semi-explicit DAE system such as (48)-(49) it is tempting to try to "eliminate" the algebraic (control) variables in order to utilize a more standard method for solving ODE's. Proceeding formally to "solve" (49) one can write

$$\mathbf{u}(t) = \mathbf{g}^{-1}[\mathbf{y}, t]. \tag{72}$$

When this value is substituted into (48) one obtains the nonlinear differential equation

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{g}^{-1}[\mathbf{y}, t], t] \tag{73}$$

which is amenable to solution using any of the ODE techniques described above. Another elimination technique referred to as *differential inclusion*[47] attempts to form an expression of the form

$$\mathbf{u}(t) = \mathcal{F}[\dot{\mathbf{y}}, \mathbf{y}, t]. \tag{74}$$

by solving a subset of the differential equations (48). Since the number of state and control variables are not necessarily equal, it is imperative to partition the differential equations in some stable manner in order to perform this "elimination." Unfortunately it is seldom possible to analytically construct a feedback control of the form (72) or (74). When analytic elimination is impossible, the only recourse is to introduce a nonlinear iterative technique (e.g. Newton's method) that must be executed at every integration step. This approach is not only very time consuming but can conflict with logic used to control integration error in the dynamic variables $\mathbf{y}$. If an implicit method is used for solving the ODE's, this "elimination" iteration must be performed within each corrector iteration, in other words it becomes an iteration within an iteration. In essence, methods that attempt to eliminate the control, in order to avoid the DAE problem are cumbersome, numerically unstable, and problem specific.

The first general technique for solving DAE's was proposed by Gear[48] and utilizes a backward differentiation formula (BDF) in a linear multistep method. In contrast to the elimination methods in the previous paragraph, the algebraic variables $\mathbf{u}(t)$ are treated the same as the differential variables $\mathbf{y}(t)$. The method was originally proposed for the semi-explicit index one system described by (48)-(49) and soon extended to the fully implicit form

$$\mathbf{F}\left[\dot{\mathbf{z}}, \mathbf{z}, t\right] = \mathbf{0} \tag{75}$$

where $\mathbf{z} = (\mathbf{y}, \mathbf{u})$. The basic idea of the BDF approach is to replace the derivative $\dot{\mathbf{z}}$ by the derivative of the polynomial which interpolates the solution computed over the preceding k steps. The simplest example is the implicit Euler method which replaces (75) with

$$\mathbf{F}\left[\frac{\mathbf{z}_i - \mathbf{z}_{i-1}}{h_i}, \mathbf{z}_i, t_i\right] = \mathbf{0}. \tag{76}$$

The resulting nonlinear system in the unknowns $\mathbf{z}_i$ is usually solved by some form of Newton's method at each time step t_i. The widely used production code DASSL developed by Petzold[49, 50] essentially uses a variable stepsize, variable order implementation of the BDF formulas. The method is appropriate for index one DAE's with consistent initial conditions. Current research into the solution of DAE's with higher index (≥ 2) has renewed interest in one-step methods, specifically the implicit Runge-Kutta (IRK) schemes described for ODE's. A discussion of methods for solving DAE's is found in the book by Brenan, Campbell, and Petzold.[25]

5.2 Tabular Data

In practice the numerical solution of a trajectory optimization problem inevitably involves tabular data. Typically propulsion, aerodynamic, weight, and mass properties for a vehicle are specified using "tables." For example the thrust of a motor may be specified by a finite set of "table" values $\{T(M_k, h_k), M_k, h_k\}$ for $k = 1, \ldots, N$, in lieu of defining the functional form in terms of Mach number and altitude. In some cases this approach is necessary simply because there is not enough information to permit an analytic representation of the function based on the laws of physics. Often tabular data is obtained as the result of experimental tests. Finally, there may be historical precedence for specifying data in this format as a convenient way for communication between disciplines. Regardless of the reason for specifying a nonlinear function as a collection of tabular values, the numerical implementation of a trajectory optimization must deal with this format. The necessary conditions described in Section 3 assume continuity and differentiability for the objective and constraint functions. Similar restrictions are implied when stating the necessary conditions for the optimal control problem in Section 4. The numerical integration techniques given in the previous subsection make similar assumptions about continuity and differentiability for the right hand sides of the differential algebraic equations. Successful application of these techniques requires that tabular data be represented using a smooth differentiable function. Unfortunately, by far the single most widely used approach is linear interpolation. This is also by far the single most catastrophic impediment to an efficient solution of the trajectory optimization problem! A piecewise linear representation is not differentiable at the table points and thus is fundamentally inconsistent with the theory described in the preceding sections. Recognizing this difficulty is certainly not new and has been discussed by other authors[51, 52] in the simulation of space launch vehicles. Nevertheless, inappropriate data modeling techniques persist, presumably for historical reasons, in many real applications.

There are many alternatives for representing tabular data using a smooth functional form. For some applications an appropriate model is suggested by the physics, e.g. a "quadratic" drag polar. In lieu of a form derived from physical considerations, functional approximation based solely on the mathematical requirements can be incorporated. Function approximation using B-splines[53] can effectively produce the required continuity. Methods for constructing smooth approximations utilizing nonlinear programming techniques have also been developed.[54–57] Nonlinear rational function, or neural network approximations can also produce sufficient smoothness, although it is not clear that these are preferable to B-splines.

6 A Compendium of Methods

The basic elements involved in the specification of a numerical method for solving the trajectory optimization problem have been described in the preceding sections. There is a broad spectrum of possible ways to put the pieces together to form a complete algorithm however all techniques have one attribute in common. Since all of the algorithms involve application of Newton's method, a convenient way to organize the discussion is to describe the function evalution procedure for each method. Specifically we will describe the "function generator" for each algorithm. The inputs to the function generator are the *variables.* The outputs of the function generator are the *objective and constraints.* The basic concept is illustrated in Figure 1.

6.1 Direct Shooting

6.1.1 Algorithm

The variables for a direct shooting application are chosen as a subset of the initial conditions, the final conditions and the parameters. Thus for each phase let us define

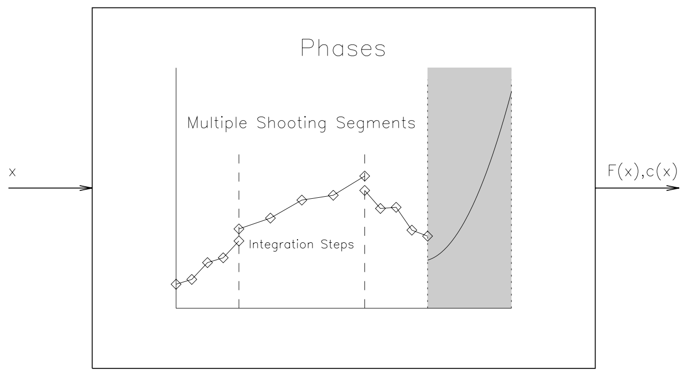
$$X^{(k)} = \{\mathbf{y}(t_0), \mathbf{p}, t_0, \mathbf{y}(t_f), t_f\}. \tag{77}$$

The total set of NLP variables is then

$$\mathbf{x} \subset \{X^{(1)}, X^{(2)}, \ldots, X^{(N)}\}. \tag{78}$$

Notice that any time varying quantities must be represented using the finite set of parameters $\mathbf{x}$, and consequently this implies that the control/time history must be defined by a finite set of parameters. For

Figure 1: Function Generator



example one might have an explicit representation such as

$$u = p_1 + p_2 t \tag{79}$$

or an implicit relationship such as

$$0 = p_1 u(t) + \sin[p_2 u(t)]. \tag{80}$$

When the control is defined explicitly as in (79) propagation of the trajectory from the beginning to the end of the phase can be accomplished using an ODE initial value method as described. On the other hand, if the control is defined implicitly, the phase propagation will require the use of a DAE initial value method as described in Section 5.1. Notice also that problems with path inequality constraints (5) must be treated as a sequence of constrained and unconstrained arcs. Thus phases must be introduced to account for these individual arcs, in addition to phases that are necessary to model known problem discontinuities such as jettison of a stage.

The NLP constraints and objective function are quantities that are evaluated at the boundaries of one or more of the phases. Thus we have

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} \boldsymbol{\psi}^{(1)}[\mathbf{y}(t_0), \mathbf{p}, t_0] \\ \boldsymbol{\psi}^{(1)}[\mathbf{y}(t_f), \mathbf{p}, t_f] \\ \vdots \\ \boldsymbol{\psi}^{(N)}[\mathbf{y}(t_0), \mathbf{p}, t_0] \\ \boldsymbol{\psi}^{(N)}[\mathbf{y}(t_f), \mathbf{p}, t_f] \end{bmatrix}. \tag{81}$$

In summary the function generator for the direct shooting method is of the following form:

```
Direct Shooting

Input: $\mathbf{x}$
do for (each phase) $k = 1, N$
    Initialize Phase $k$:
        $\mathbf{y}^{(k)}(t_0), \mathbf{p}^{(k)}, t_0^{(k)}$
    Constraint Evaluation:
        Compute $\boldsymbol{\psi}^{(k)}[\mathbf{y}(t_0), \mathbf{p}, t_0]$
    Initial Value Problem: Given $t_f$ compute
        $\mathbf{y}^{(k)}(t_f)$ i.e. solve (65), or (48)-(49)
    Constraint Evaluation:
        Compute $\boldsymbol{\psi}^{(k)}[\mathbf{y}(t_f), \mathbf{p}, t_f]$
end do
Terminate Trajectory
    Compute objective $F(\mathbf{x}), \mathbf{c}(\mathbf{x})$
Output: $F(\mathbf{x}), \mathbf{c}(\mathbf{x})$
```

6.1.2 Examples

The direct shooting method is one of the most widely used methods, and is especially effective for launch vehicle and orbit transfer applications. The POST (Program to Optimize Simulated Trajectories) program developed by Martin-Marietta[58] for simulating the trajectories of launch vehicles such as the Titan is a widely distributed implementation of the direct shooting method. Originally developed to support military space applications it is similar in functionality to the GTS (Generalized Trajectory Simulation) program[59] developed at The Aerospace Corporation. Most major aerospace firms either use POST or have equivalent capability for launch vehicle optimization and mission analysis. Early versions of POST utilized a reduced gradient optimization algorithm similar to the methods in Rosen[9] and Lasdon,[10] and more recent releases have incorporated an SQP method.[7] GTS utilizes a modified form of the reduced gradient algorithm[8] which incorporates quasi-Newton updates for constraint elimination and Hessian approximation. Programs such as POST and GTS have extensive libraries of application-specific "models." In particular the libraries permit definition of the vehicle dynamics (the right hand side functions $\mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t]$) in many coordinate systems, e.g. earth centered inertial (ECI), intrinsic, orbital, etc. It is also common to have 10-20 different models for computing the gravitational, propulsive, and aerodynamic forces. In most cases the user can also specify the type of numerical integration, and interpolation to be used as well as the trajectory input and output formats.

Direct shooting applications have been most successful in launch and orbit transfer problems primarily because this class of problem lends itself to parameterization with a relatively small number of NLP variables. For example, an orbit transfer problem with impulsive burns[60, 61] can be posed with 4 variables per burn namely the time of ignition and the velocity increment $(t_i, \boldsymbol{\Delta}\mathbf{V}_i)$. Typically the mission orbit can be defined using 3-5 nonlinear constraints inforced at the end of the trajectory. Thus a typical two-burn orbit transfer can be posed as an NLP with 8 variables and 4 or 5 constraints. When the vehicle thrust-to-weight ratio is high, there is little motivation to consider a more elaborate mathematical model of the thrust variation for two reasons. First, the performance benefit that can be achieved with a thrust variation (i.e. by introducing time varying control $\mathbf{u}(t)$) is negligible. Second, most real vehicles do not have the ability to implement a variable direction thrust even if it was computed. In fact many spacecraft incorporate spin stabilization, which implies a constant inertial attitude during the burns. Stated simply, the application neither permits nor warrants a mathematical model of higher fidelity, and direct shooting is very effective.

A similar situation exists when designing optimal launch vehicle trajectories. During the early portion of an ascent trajectory it is common to define the turning by a finite set of "pitch rates." This approach is used for most expendable launch vehicles (e.g Titan, Delta, Atlas/Centaur) and is often a part of the onboard mission data load (MDL). Consequently, the steering during the early portion of a launch vehicle trajectory is defined by a relatively small number of parameters and the resulting optimization problem is readily formulated using the direct shooting method. Steering during the second stage of the Space Shuttle ascent trajectory is defined by a *linear tangent steering* law. In this case the control can be defined by six parameters $\mathbf{p}_1, \mathbf{p}_2$

$$\mathbf{u}(t) = \mathbf{p}_1 + \mathbf{p}_2 t \tag{82}$$

which then determine the inertial yaw and pitch angles according to

$$\psi_I = \arctan\{u_2/u_1\} \tag{83}$$

$$\theta_I = \arcsin\{u_3/\|\mathbf{u}\|\} \tag{84}$$

This form of the control law is an exact solution of the optimal control problem when gravity is constant,[23, 62] and is implemented in the shuttle flight avionics. Again the optimal steering is approximated by a finite set of parameters, and the resulting trajectory optimization problem is amenable to direct shooting.

6.1.3 Issues

Most successful direct shooting applications have one salient feature in common, namely the ability to describe the problem in terms of a relatively small number of optimization variables. If the dynamic behavior of the control functions $\mathbf{u}(t)$ cannot be represented using a limited number of NLP variables the success of a direct shooting method can be degraded significantly. For example, it is tempting to approximate the controls using an expansion such as

$$\mathbf{u}(t) = \sum_{k=1}^{M} \mathbf{p}_k B_k(t) \tag{85}$$

where $M \gg 1$, and $B_k(t)$ are as a set of basis functions (e.g. B-spline). This approach impacts the direct shooting method in two ways. Both are related to the calculation of gradient information for the NLP iteration. The first issue is related to the sensitivity of the variables. Changes early in the trajectory (near t_0) propagate to the end of the trajectory. The net effect is that the constraints can behave very nonlinearly with respect to variables, thereby making the optimization problem difficult to solve. This is one of the major reasons for the multiple shooting techniques, which will be described below. The second issue is the computational cost of evaluating the gradient information. The most common approach to computing gradients is via finite difference approximations. A *forward difference* approximation to column j of the Jacobian matrix $\mathbf{G}$ in (20) is

$$\mathbf{G}_{\cdot j} = \frac{1}{\delta_j}\left[\mathbf{c}(\mathbf{x}+\boldsymbol{\delta}_j) - \mathbf{c}(\mathbf{x})\right] \tag{86}$$

where the vector $\boldsymbol{\delta}_j = \delta_j \mathbf{e}_j$ and $\mathbf{e}_j$ is a unit vector in direction j. A *central difference* approximation is

$$\mathbf{G}_{\cdot j} = \frac{1}{2\delta_j}\left[\mathbf{c}(\mathbf{x}+\boldsymbol{\delta}_j) - \mathbf{c}(\mathbf{x}-\boldsymbol{\delta}_j)\right]. \tag{87}$$

In order to calculate gradient information this way it is necessary to integrate the trajectory for each perturbation. Consequently at least n trajectories are required to evaluate a finite difference gradient, and this information may be required for each NLP iteration. The cost of computing the finite difference gradients is reduced somewhat in the GTS[59] program by using a "partial trajectory" mechanism. This approach recognizes that it is not necessary to integrate the trajectory from t_0 to t_f if the optimization variable is introduced later say at $t_s > t_0$. Instead the gradient information can be computed by integrating from the "return point" t_r to t_f, where $t_s \geq t_r > t_0$ since the portion of the trajectory from t_0 to t_r will not be altered by the perturbation. A less common alternative to finite difference gradients, is to integrate the so-called variational equations. In this technique, an additional differential equation is introduced for each NLP variable and this augmented system of differential equations must be solved along with the state equations. Unfortunately the variational equations must be derived for each application and consequently are used far less in general purpose trajectory software.

Another issue which must be addressed is the accuracy of the gradient information. Forward difference estimates are of order δ, whereas central difference estimates are $\mathcal{O}(\delta^2)$. Of course the more accurate central difference estimates are twice as expensive as forward difference gradients. Typically numerical implementations use forward difference estimates until nearly converged and then switch to the more accurate derivatives for convergence. While techniques for selecting the finite difference perturbation size might seem to be critical to accurate gradient evalution a number of effective methods are available to deal with this matter.[4] A more crucial matter is the interaction between the gradient computations and the underlying numerical interpolation and integration algorithms. We have already discussed how linear interpolation of tabular data can introduce gradient errors. However, it should be emphasized that sophisticated predictor corrector variable step variable order numerical integration algorithms also introduce "noise" into the gradients. Although these techniques enhance the efficiency of the integration, they degrade the efficiency of the optimization. In fact a simple fixed-step, fixed-order integrator may yield better overall efficiency in the trajectory optimization because the gradient information is more accurate. Two integration methods which are suitable for use inside the trajectory function generator are described in Brenan,[63] and Vu.[64, 65] Another issue arises in the context of a trajectory optimization application when the final time t_f is defined implicitly by a boundary or event condition, not explicitly. In this case we are not asking to integrate from t_0 to t_f but rather from t_0 <u>until</u> $\psi[\mathbf{y}(t_f), t_f] = 0$. Most numerical integration schemes *interpolate* the solution to locate the final point. On the other hand if the final point is found by *iteration* (e.g. using a root finding method), the net effect is to introduce noise into the external Jacobian evaluations. A better alternative is to simply add an extra variable and constraint to the overall NLP problem and avoid the use of an "internal" iteration. In fact inaccuracies in the gradient can be introduced by

1. internal iterations (e.g. solving Kepler's equation, event detection)

2. interpolation of tabular data

3. discontinuous functions (e.g. "ABS", "MAX", "IF" tests)

and a carefully implemented algorithm must avoid these difficulties.[66]

6.2 Indirect Shooting

6.2.1 Algorithm

Let us begin with a description of indirect shooting for the simplest type of optimal control problem with no path constraints and a single phase. The variables are chosen as a subset of the boundary values for the optimal control necessary conditions. For this case the NLP variables are

$$\mathbf{x} = \{\boldsymbol{\lambda}(t_0), t_f\}. \tag{88}$$

and the NLP constraints are

$$\mathbf{c}(\mathbf{x}) = \left. \begin{bmatrix} \boldsymbol{\psi}[\mathbf{y}(t), \mathbf{p}, t] \\ \boldsymbol{\lambda}(t) - \boldsymbol{\Phi_y}^{\top} \\ (\Phi_t + H) \end{bmatrix} \right|_{t=t_f} . \tag{89}$$

A major difference between direct and indirect shooting occurs in the definition of the control functions $\mathbf{u}(t)$. For indirect shooting the control is defined at each point in time by the maximum principle (41) or (37). Thus in some sense the values $\boldsymbol{\lambda}(t_0)$ become the "parameters" which define the optimal control function instead of $\mathbf{p}$. When the maximum principle is simple enough to permit an explicit definition of the control, propagation of the trajectory from the beginning to the end of the phase can be accomplished using an ODE initial value method. On the other hand if the control is defined implicitly the phase propagation will require the use of a DAE initial value method as described in Section 5.1. Notice also that problems with path inequality constraints (5) must be treated as a sequence of constrained and unconstrained arcs. Thus phases must be introduced to account for these individual arcs just as with direct shooting. When additional phases are introduced in general it will be necessary to augment the set of variables to include the unknown adjoint and multipliers at each of the phase boundaries. Furthermore additional constraints are added to reflect the additional necessary conditions.

In summary the function generator for the indirect shooting method is of the following form:

Indirect Shooting

```
Input: $\mathbf{x}$
do for (each phase) $k = 1, N$
    Initialize Phase $k$:
        $\mathbf{y}^{(k)}(t_0), \boldsymbol{\lambda}^{(k)}(t_0), \mathbf{p}^{(k)}, t_0^{(k)}$
    Initial Value Problem: Given $t_f$ compute
        $\mathbf{y}^{(k)}(t_f), \boldsymbol{\lambda}^{(k)}(t_f)$
        i.e. solve (31), (36), and (37)
    Constraint Evaluation:
        Evaluate (89)
end do
Terminate Trajectory
Output: $\mathbf{c}(\mathbf{x})$
```

6.2.2 Examples

Although the indirect shooting method would seem to be quite straight forward it suffers from a number of difficulties which will be described in the next section. Primarily because of the computational limitations successful applications of indirect shooting have focused on special cases. Because the method is very sensitive to the initial guess, it is most successful when the underlying dynamics are rather benign. The method has been utilized for launch vehicle trajectory design in the program DUKSUP[67] and for low-thrust orbit analysis.[68]

6.2.3 Issues

The sensitivity of the indirect shooting method has been recognized for some time. Computational experience with the technique in the late 1960's is summarized by Bryson and Ho [23, p 214]

> The main difficulty with these methods is *getting started*; i.e., finding a first estimate of the unspecified conditions at one end that produces a solution reasonably close to the specified conditions at the other end. The reason for this peculiar difficulty is the extremal solutions are often *very sensitive* to small changes in the unspecified boundary conditions. ... Since the system equations and the Euler-Lagrange equations are coupled together, it is not unusual for the numerical integration, with poorly guessed initial conditions, to produce "wild" trajectories in the state space. These trajectories may be so wild that values of $x(t)$ and/or $\lambda(t)$ exceed the numerical range of the computer!

A number of techniques have been proposed for dealing with this sensitivity. One rather obvious approach is to begin the iteration process with a "good" initial guess. Referred to as *imbedding, continuation or homotopy* methods, the basic idea is to solve a sequence of problems, and use the solution of one problem as the initial guess for a slightly modified problem. Thus suppose it is necessary to solve $\mathbf{a}(\mathbf{x}) = \mathbf{0}$ and we can *imbed* this problem into a family of related problems

$$\overline{\mathbf{a}}(\mathbf{x}, \tau) = \mathbf{0} \tag{90}$$

where the parameter $0 \le \tau \le 1$. Assume the problem $\overline{\mathbf{a}}(\mathbf{x}, 0) = \mathbf{0}$ is "easy" to solve, and when $\tau = 1$ the real solution is obtained, i.e. $\overline{\mathbf{a}}(\mathbf{x}, 1) = \mathbf{a}(\mathbf{x}) = \mathbf{0}$. Typically it is desirable to choose the parameter τ such the the solution $\mathbf{x}(\tau)$ varies smoothly along the *homotopy path*. For example it may be "easy" to solve an orbit transfer using two-body dynamics. A more accurate solution involving oblate earth perturbations, could then be obtained by "turning on" the gravitational perturbations with an imbedding technique. Clearly the homotopy method can be used with any trajectory optimizing algorithm, but it has been especially useful for indirect methods.

Another technique which has been used to reduce the solution sensitivity is referred to as the *sweep* method. Essentially the idea is to integrate the state equations forward, i.e. from t_0 to t_f and then integrate the adjoint equations backward, i.e. from t_f to t_0. The goal is to exploit the fact that the state equations may be integrated stably in the forward direction and the adjoint equations may be stable in the reverse direction. This approach requires that the state, control, and adjoint time histories by saved using some type of interpolation method. Numerical processing is further complicated by interaction between the interpolation scheme and the integration error control especially when dealing with discontinuities that occur at phase boundaries.

Perhaps the biggest issue that must be addressed when using an indirect method is the derivation of the necessary conditions themselves. For realistic trajectory simulations the differential equations (2), path constraints (5), and boundary conditions (4) may all be complicated mathematical expressions. In order to impose the optimality conditions (36), (37), (38), and (39) it is necessary to *analytically* differentiate the expressions for $\mathbf{f}$, $\mathbf{g}$, and $\boldsymbol{\psi}$. For a production software tool such as POST or GTS, which permits problem formulation using alternate coordinate systems, propulsion, gravitational, and aerodynamic models this can be a daunting task! As a consequence the optimality conditions are usually *not* derived for all possible combinations and models. The impact is that current implementations of indirect methods suffer from a lack of flexibility. Let us emphasize this is a limitation in current, but not necessarily future implementations. In particular a number of authors have explored the utility of automatic differentiation tools to eliminate this impediment. For example the ADIFOR software,[69] the OCCAL software,[70] and the approach taken by Mehlhorn and Sachs[71] represent promising attempts to automate this process.

6.3 Multiple Shooting

6.3.1 Algorithm

Both the direct and indirect shooting methods suffer from a common difficulty. The essential shortcoming of these methods is that small changes introduced early in the trajectory can propagate into very nonlinear changes at the end of the trajectory. While this effect can be catastrophic for an indirect method, it also represents a substantial limitation on the utility of a direct formulation. The basic notion of *multiple shooting* (in contrast to *simple shooting*) was originally introduced[72, 73] for solving two-point boundary value problems and we begin the discussion for this case. In its simplest form the problem can be stated as follows: compute the unknown initial values $\mathbf{v}(t_0) = \mathbf{v}_0$ such that the boundary condition

$$\mathbf{0} = \boldsymbol{\phi}[\mathbf{v}(t_f), t_f] \tag{91}$$

holds for some value of $t_0 < t_f$ which satisfies

$$\dot{\mathbf{v}} = \mathbf{f}[\mathbf{v}(t), t]. \tag{92}$$

The fundamental idea of multiple shooting is to break the trajectory into shorter pieces or segments. Thus we break the time domain into smaller intervals of the form

$$t_0 < t_1 < \ldots < t_M = t_f. \tag{93}$$

Let us denote $\mathbf{v}_j$ for $j = 0, \ldots, (M-1)$, as the initial value for the dynamic variable at the beginning of each segment. For segment j we can integrate the differential equations (92) from t_j to the end of the segment at t_{j+1}. Denote the result of this integration by $\overline{\mathbf{v}}_j$. Collecting the results for all segments let us define a set of NLP variables

$$\mathbf{x} = \{\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{M-1}\}. \tag{94}$$

Now we also must ensure that the segments join at the boundaries, consequently we impose the constraints

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} \mathbf{v}_1 - \overline{\mathbf{v}}_0 \\ \mathbf{v}_2 - \overline{\mathbf{v}}_1 \\ \vdots \\ \boldsymbol{\phi}[\mathbf{v}_M, t_f] \end{bmatrix} = \mathbf{0}. \tag{95}$$

One obvious result of the multiple shooting approach is an increase in the size of the problem that the Newton iteration must solve since additional variables and constraints are introduced for each shooting segment. In particular the number of NLP variables and constraints for a multiple shooting application is $n = n_v M$ where n_v is the number of dynamic variables $\mathbf{v}$ and M is the number of segments. Fortunately the Jacobian matrix $\mathbf{A}$ which appears in the calculation of the Newton search direction (12) is *sparse*. In particular only Mn_v^2 elements in $\mathbf{A}$ are nonzero. This sparsity is a direct consequence of the multiple shooting formulation since variables early in the trajectory do not change constraints later in the trajectory. In fact Jacobian sparsity is the mathematical consequence of "uncoupling" between the multiple shooting segments. For the simple case described, the Jacobian matrix is banded with $n_v \times n_v$ blocks along the diagonal, and very efficient methods for solving the linear system (12) can be utilized. It is important to note that the multiple shooting segments are introduced strictly for numerical reasons. The original optimal control problem may also have phases as described previously. Thus, in general, each phase will be subdivided into multiple shooting segments as illustrated in Figure 1. Furthermore within each phase the set of differential-algebraic equations and corresponding boundary conditions may be different depending on whether the arc is constrained or unconstrained, etc.

The multiple shooting concept can be incorporated into either a direct or indirect method. The distinction between the two occurs in the definition of the dynamic variables $\mathbf{v}$, the dynamic system (92), and the boundary conditions (91). For a *direct multiple shooting* method, we can identify the dynamic variables $\mathbf{v}$ with the state and control $(\mathbf{y}, \mathbf{u})$. By analogy the dynamics are given by the original state equations (2) and path constraints (5). In lieu of the simple boundary condtions (91), we directly impose (3) and (4). For an *indirect multiple shooting* algorithm the dynamic variables $\mathbf{v}$ must include the state, control, and adjoint variables $(\mathbf{y}, \mathbf{u}, \boldsymbol{\lambda})$. The dynamics are given by the original state equations (2) and the appropriate necessary conditions (36) and (37). In this instance the boundary conditions (91) are replaced with the transversality conditions (38), (39) along with (3) and (4). It also may be necessary to augment the set of NLP iteration variables $\mathbf{x}$ and constraints $\mathbf{c}(\mathbf{x})$ to account for the additional conditions that occur when entering and leaving path inequalities. As a final distinction for an indirect method the number of NLP variables $\mathbf{x}$ and constraints $\mathbf{c}(\mathbf{x})$ are equal, i.e. $m = n$ since the optimality conditions uniquely define the values of the NLP variables. For a direct method n and m may differ, and the objective function $F(\mathbf{x})$ must be used to define the optimal values of the NLP variables.

The function generator for the multiple shooting method is of the following form:

Multiple Shooting

```
Input: $\mathbf{x}$
do for (each phase) $k = 1, N$
    Initialize Phase $k$:
    do for (each segment) $j = 0, M - 1$
        Initialize Segment $j + 1$:
            $\mathbf{v}_j, t_j$
        Initial Value Problem: Given $t_{j+1}$ compute
            $\overline{\mathbf{v}}_j$ i.e. solve DAE system
        Constraint Evaluation:
            save $\mathbf{v}_{j+1} - \overline{\mathbf{v}}_j$ in (95)
    end do
        save $\boldsymbol{\phi}[\mathbf{v}_M, t_f]$ in (95)
end do
Terminate Trajectory
Output: $\mathbf{c}(\mathbf{x})$ (and $F(\mathbf{x})$)
```

6.3.2 Examples

Perhaps the single most important benefit derived from a multiple shooting formulation (either direct or indirect) is enhanced robustness. The BNDSCO implementation,[74] of indirect multiple shooting is widely used in Germany, to solve very difficult applications. An optimal interplanetary orbit transfer involving planetary perturbations has been computed by Callies.[75] Pesch and his coworkers have utilized the approach for the study of landing in the presence of windshear,[29] shuttle reentry,[76] and a number of other aerospace applications.[30] System identification problems have been addressed by Bock and Plitt.[77]

An interesting benefit of the multiple shooting algorithm is the ability to exploit a parallel processor. The method is sometimes called *parallel shooting*, because the simulation of each segment and/or phase can be implemented on an individual processor. This technique was explored for a direct multiple shooting method[78] and remains an intriguing prospect for multiple shooting methods in general.

6.3.3 Issues

The multiple shooting technique greatly enhances the robustness of either direct or indirect methods. However, the number of NLP iteration variables and constraints increase markedly over simple shooting implementations. Consequently it is imperative to exploit matrix sparsity in order to efficiently solve the NLP Newton equations. For indirect shooting the matrix **A** has a simple block banded structure and efficient linear algebra methods are rather straight forward. For direct shooting, sparsity appears both in the Jacobian **G** and the Hessian **H** and the relevant sparse linear system is the KT system (22). This system can be solved efficiently using the multifrontal method for symmetric indefinite matrices.[13] In general all of the other issues associated with simple direct and indirect shooting still apply. Perhaps the most perplexing difficulty with shooting methods is the need to define constrained and unconstrained subarcs *a priori*, when solving problems with path inequalities.

6.4 Indirect Transcription

6.4.1 Algorithm

Historically *transcription* or *collocation* methods were developed for solving two-point boundary value problems such as those encountered when solving the necessary conditions with an indirect formulation. Let us again consider the simple boundary value problem (91), (92), and as with multiple shooting subdivide the interval as in (93). A fundamental part of the multiple shooting method was to solve the ODE's using an initial value method. Let us consider taking a *single* step with an explicit method such as Euler's. Following the multiple shooting methodology we then must impose constraints of the form

$$\mathbf{0} = \mathbf{v}_{j+1} - \overline{\mathbf{v}}_j \qquad (96)$$

$$= \quad \mathbf{v}_{j+1} - (\mathbf{v}_j + h_j \mathbf{f}_j) \tag{97}$$

where $t_{j+1} = h_j + t_j$ for all of the segments $j = 0, \ldots, (M-1)$. Of course there is no reason to restrict the approximation to an Euler method. In fact if we incorporate an implicit scheme such as the trapezoidal method (68), satisfaction of the *defect* constraint (97) is exactly the same as the corrector iteration step described for all implicit integrators. The only difference is that all of the corrector iterations are done at once in the boundary value context, whereas the corrector iterations are done one at a time (step by step) when the process is part of an initial value integration method. One of the most popular and effective choices for the defect constraint in collocation methods is the Hermite-Simpson method:[79]

$$\mathbf{0} = \mathbf{v}_{j+1} - \mathbf{v}_j - \frac{h_{j+1}}{6}\left[\mathbf{f}_{j+1} + 4\overline{\mathbf{f}}_{j+1} + \mathbf{f}_j\right] \doteq \boldsymbol{\zeta}_j \tag{98}$$

which is the "Simpson" defect with "Hermite" interpolant

$$\overline{\mathbf{v}}_{j+1} = \frac{1}{2}\left[\mathbf{v}_j + \mathbf{v}_{j+1}\right] + \frac{h_{j+1}}{8}\left[\mathbf{f}_j - \mathbf{f}_{j+1}\right] \tag{99}$$

for the variables at the interval midpoint. Schemes of this type are referred to as collocation methods[80] because the solution is a piecewise continuous polynomial which collocates (i.e. satisfies) the ODE's at the so-called collocation points in the subinterval $t_j \le t \le t_{j+1}$. For obvious reasons the points t_j are also called *grid points*, *mesh points*, or *nodes*.

The function generator for the indirect transcription method is of the following form:

```
Indirect Transcription

Input: $\mathbf{x}$
do for (each phase) $k = 1, N$
    Initialize Phase $k$:
    do for (each grid point) $j = 0, M-1$
        Constraint Evaluation: save
            discretization defect e.g. (98) or (68)
    end do
        save $\boldsymbol{\phi}[\mathbf{v}_M, t_f]$ in (95)
end do
Terminate Trajectory
Output: $\mathbf{c}(\mathbf{x})$
```

6.4.2 Examples

Collocation methods have been used for solving boundary value problems occurring in many fields for nearly forty years, and the reader is urged to consult Dickmanns,[81] Russell,[80] and Ascher.[24] Dickmanns[81] implemented the Hermite-Simpson method in the CHAP3 software and reported successful solution of a number of challenging applications including shuttle reentry problems with convective heating and maximum cross-range capability. The COLSYS package developed by Ascher, Christiansen, and Russell[82] has also been widely used for boundary value problems.

6.4.3 Issues

Collocation methods can be extremely effective for solving multipoint boundary value problems such as those encountered when optimizing a trajectory. As with all indirect methods however, the techniques cannot be applied without computing the adjoint equations. Furthermore, when path inequality constraints are present, it is imperative to predetermine the sequence of constrained and unconstrained subarcs in order to formulate the correct BVP.

6.5 Direct Transcription

6.5.1 Algorithm

There are two major reasons that direct transcription methods[83] are actively being investigated. First, like all direct methods, they can be applied without explicitly deriving the necessary conditions (i.e. adjoint, transversality, maximum principle). This feature makes the method appealing for complicated applications and promises versatility and robustness. Second, in contrast to all other techniques described, direct transcription methods do not require an a priori specification of the arc sequence for problems with path inequalities.

Just as before, we break the time domain into smaller intervals of the form

$$t_0 < t_1 < \ldots < t_M = t_f. \tag{100}$$

The NLP variables then become the values of the state and control at the grid points, namely,

$$\mathbf{x} = \{\mathbf{y}_0, \mathbf{u}_0, \mathbf{y}_1, \mathbf{u}_1, \ldots, \mathbf{y}_M, \mathbf{u}_M\}\,. \tag{101}$$

The set of NLP variables may be augmented to include the parameters $\mathbf{p}$, the times t_0 and t_f, and for some discretizations the values of the state and control at collocation points between the grid points. The key notion of the collocation methods is to replace the original set of ODE's (2) by a set of defect constraints $\boldsymbol{\zeta}_i = \mathbf{0}$ which are imposed on each interval in the discretization. Thus when combined with the original path constraints the complete set of NLP constraints is

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} \boldsymbol{\psi}_0 \\ \mathbf{g}[\mathbf{y}_0, \mathbf{u}_0, \mathbf{p}, t_0] \\ \boldsymbol{\zeta}_0 \\ \mathbf{g}[\mathbf{y}_1, \mathbf{u}_1, \mathbf{p}, t_1] \\ \boldsymbol{\zeta}_1 \\ \vdots \\ \mathbf{g}[\mathbf{y}_{M-1}, \mathbf{u}_{M-1}, \mathbf{p}, t_{M-1}] \\ \boldsymbol{\zeta}_{M-1} \\ \mathbf{g}[\mathbf{y}_M, \mathbf{u}_M, \mathbf{p}, t_M] \\ \boldsymbol{\psi}_f \end{bmatrix}. \tag{102}$$

The resulting formulation is a *transcription* of the original trajectory optimization problem defined by (2)–(7) into an NLP problem as given by (27)–(29).

Collecting results yields a function generator for the direct transcription method of the following form:

Direct Transcription

```
Input: $\mathbf{x}$
do for (each phase) $k = 1, N$
    Initialize Phase $k$: save $\boldsymbol{\psi}_0$
    do for (each grid point) $j = 0, M-1$
        Constraint Evaluation: save
            $\mathbf{g}[\mathbf{y}_j, \mathbf{u}_j, \mathbf{p}, t_j]$ and $\boldsymbol{\zeta}_j$
    end do
    Terminate Phase
        save $\mathbf{g}[\mathbf{y}_M, \mathbf{u}_M, \mathbf{p}, t_M]$ and $\boldsymbol{\psi}_f$
end do
Terminate Trajectory
Output: $\mathbf{c}(\mathbf{x})$ and $F(\mathbf{x})$
```

The nonlinear programming problem which results from this formulation is large. It is clear that the number of NLP variables $n \approx (n_y + n_u)MN$, with a similar number of NLP constraints. Thus a typical trajectory with 7 states, 2 controls, 100 grid points per phase, and 5 phases, produces an NLP with $n = 4500$. Fortunately the pertinent matrices for this NLP problem, namely the Jacobian $\mathbf{G}$ and the Hessian $\mathbf{H}$ are also *sparse*. So for an application of this type it would not be unusual for these matrices to have fewer than 1% of the elements be nonzero. Consequently exploiting sparsity to reduce both storage and computation time is a critical aspect of a successful implementation when using a direct transcription method.

6.5.2 Examples

The OTIS (Optimal Trajectories by Implicit Simulation) program originally proposed by Hargraves and Paris[84] implements the basic collocation method, in addition to a more standard direct shooting approach. The original implementation has been widely distributed to NASA, Air Force, academic and commercial institutions throughout the U.S. Early versions of the tool utilized the Hermite-Simpson defect,[79] and the NPSOL nonlinear programming software.[7] More recent versions of the OTIS software have incorporated the SOCS (Sparse Optimal Control Software).[85–89] The OTIS/SOCS library incorporates a number of features not available in earlier versions. In particular a sparse nonlinear programming algorithm,[11, 12, 14, 15] permits the solution of problems with $n \approx m \approx 100,000$ on an engineering workstation. The sparse NLP implements a sparse sequential quadratic programming (SQP) method based on a Schur-complement algorithm suggested by Gill, et.al.[6, 16] Jacobian and Hessian matrices are computed efficiently using sparse finite differencing as proposed by Curtis, Powell, and Reid.[18, 19] Automatic refinement of the mesh to achieve specified accuracy in the discretization is available.[90] Like other widely used production tools for trajectory design, OTIS provides an extensive library of models to permit ascent, reentry, and orbital simulations. The SOCS software has also been used independent of the OTIS software for low-thrust orbit transfers,[91] low-thrust interplanetary transfers,[92] commercial aircraft mission analysis,[93] and applications in chemical process control and robotics. The direct transcription method has also been implemented by Enright and Conway.[34, 94] The ALTOS (Advanced Launch Trajectory Optimization Software) program[95] developed in Germany for the European

Space Agency, incorporates many of the same features.[96] A sparse reduced gradient method has been investigated for collocation problems by Brenan and Hallman,[97] and Steinbach has investigated an interior point SQP approach.[98]

6.5.3 Issues

When the direct transcription method is implemented using a sparse nonlinear programming algorithm the overall approach does resolve many of the difficulties encountered in trajectory optimization. Some limitations can be attributed to the underlying sparse NLP algorithms. For example one of the principle attractions of a direct method is that adjoint equations do not need to be computed. On the other hand the underlying NLP must in fact use derivative information that is in some sense equivalent. The sparse NLP used in SOCS computes Jacobian and Hessian information by sparse finite differencing. This technique will be efficient as long as the number of perturbations is reasonably small. The number of perturbations are determined by the DAE right hand side matrices $\mathbf{f}_y$, $\mathbf{f}_u$, $\mathbf{g}_y$, and $\mathbf{g}_u$. If these matrices are dense then the number of perturbations γ needed by the sparse differencing technique is approximately equal to the number of state and control variables, i.e. $\gamma \approx n_y + n_u$. Hessian information can be computed using $\gamma^2/2$ perturbations. Thus as long as γ is "small" this technique is reasonable. Conversely the sparse differencing approach is too expensive when the size of the DAE system becomes large. On the other hand, when the right hand side matrices $\mathbf{f}_y$, $\mathbf{f}_u$, $\mathbf{g}_y$, and $\mathbf{g}_u$ are *sparse*, then it may be that $\gamma \ll n_y + n_u$. Stated simply, sparse finite differencing must either exploit right hand side sparsity, or is limited by the number of state and control variables. Reduced gradient algorithms[97] and reduced SQP methods[20] are limited in a different way. Algorithms of this type construct the reduced Hessian, which is a dense $n_d \times n_d$ matrix where the number of degrees of freedom is approximately equal to the number of controls times the number of grid points, i.e. $n_d \approx n_u M$. Since the reduced Hessian is dense and linear algebra operations are $\mathcal{O}(n_d^3)$ this implies an upper limit with $n_u M \approx 500$. Consequently NLP methods which form the reduced Hessian are limited by the number of control variables and/or the number of mesh points.

The second principle attraction involves the treatment of path inequality constraints. Defining an a priori distribution of constrained subarcs is not necessary since the underlying NLP effectively determines the subarcs using an active set strategy. However, this approach is really only an approximation to the true solution. First the number of grid points effectively defines the "resolution" of the constrained subarcs. Secondly, when entering or leaving a path constraint the control time history may require jump discontinuities at the junction points. If the control approximation does not permit discontinuities, then the result will be suboptimal. One can of course introduce a phase boundary at the junction point which will improve the control approximation. However, this technique has not been automated.

A third more fundamental difficulty occurs when the underlying DAE has index greater than one, as with singular arcs and/or state variable inequalities. In this case either the NLP subproblem is singular and/or attempts to refine the mesh for improved accuracy will fail. Essentially these difficulties can be attributed to the fact that none of the discretization schemes described are appropriate for high index DAE's. One approach is to impose additional nonlinear path conditions along the singular arc.[99] Unfortunately this approach requires analytic elimination of the adjoint variables, and a priori knowledge of the constrained subarcs. A more promising approach is to incorporate a four-point collocation scheme during the singular arc as proposed by Logsdon and Biegler.[100]

7 Other Methods

The vast majority of successful trajectory optimization applications incorporate some variant of the methods described above. For the sake of completeness, we include a brief discussion of two other approaches which have been considered, but which are generally not computationally competitive.

7.1 Dynamic Programming

In the late 1950's Richard Bellman introduced a generalization of the classical *Hamilton-Jacobi* theory.[101] The key notion of these so-called *extremal field* methods, is described by a system of first-order nonlinear partial differential equations, known as the *Hamilton-Jacobi-Bellman* or HJB equation. Essentially these PDE's describe the optimal control functions $\mathbf{u}^*[\mathbf{x}, t]$ as well as the optimal objective for all possible initial conditions $[\mathbf{x}(t_0), t_0]$. Hamilton-Jacobi-Bellman theory has played a major role in the development of necessary and sufficient conditions, and has provided a unified theoretical basis for the field of optimal control. Despite its theoretical importance, the utility of dynamic programming as the basis for a viable numerical method is summarized by Bryson and Ho: [23, p 136]

> The great drawback of dynamic programming is, as Bellman himself calls it, the "curse of dimensionality." Even recording the solution to a moderately complicated problem involves an enormous amount of storage. If we want only one optimal path from a known initial point, it is wasteful and tedious, to find a whole field of extremals ...

7.2 Genetic Algorithms

All of the trajectory optimization methods described above have well defined termination criteria. As a consequence it is possible to decide whether a candidate solution say $\hat{\mathbf{x}}$ is in fact an "answer" by evaluating the necessary conditions, e.g. (20)-(21) or (36)-(40). The ability to define "convergence" is a fundamental property of calculus-based methods. In constrast, when the variables are discrete, calculus-based methods do not apply. In general for problems with discrete variables the only way to decide if a candidate solution $\hat{\mathbf{x}}$ is in fact an "answer" is by comparison with all other possible candidates. Unfortunately this is a combinatorial problem which is computationally prohibitive for all but the smallest applications. In order to avoid direct comparison of all possible solutions, it is necessary to introduce randomness at some point in the optimization process, and abandon a definitive convergence criterion. The basic notion of genetic algorithms (GA), simulated annealing (SA), tabu search, and evolutionary or Monte Carlo methods is to randomly select values for the unknown problem variables. After a finite number of random samples the "best" value is considered the answer. For some applications, notably those with discrete variables, algorithms of this type are the only practical alternative. However, trajectory optimization problems do not fall in this class! Trajectory applications are not characterized by discrete variables and there simply is no reason to use a method which incurs the penalty associated with this assumption. Nevertheless, methods of this type have attracted the interest of many analysts, presumably because they are incredibly simple to apply without a detailed understanding of the system being optimized. Unfortunately, because they do not exploit gradient information they are not computationally competitive with the methods in Section 6.

8 Conclusions

There are many techniques for numerically solving trajectory optimization problems and it is sometimes helpful to classify the techniques as either *indirect* or *direct.* Indirect methods are characterized by explicitly solving the optimality conditions stated in terms of the adjoint differential equations, the maximum principle, and associated boundary (transversality) conditions. Using the calculus of variations, the optimal control necessary conditions can be derived by setting the first variation of the *Hamiltonian* function to zero. The indirect approach usually requires the solution of a nonlinear multi-point boundary value problem. By analogy, an indirect method for optimizing a function of n variables would require analytically computing the gradient and then locating a set of variables using a root-finding algorithm such that the gradient is zero. There are three primary drawbacks to this approach in practice. First it is necessary to derive analytic expressions for the necessary conditions, and for complicated nonlinear dynamics this can become quite daunting. Second, the region of convergence for a root-finding algorithm may be surprisingly small, especially when it is necessary to guess values for the adjoint variables which may not have an obvious physical interpretation. Third, for problems with path inequalities it is necessary to guess the sequence of constrained and unconstrained subarcs before iteration can begin.

In contrast, a direct method does not require an analytic expression for the necessary conditions, and typically does not require initial guesses for the adjoint variables. Instead, the dynamic (state and control) variables are adjusted to directly optimize the objective function. All direct methods introduce some parametric representation for the control variables. For simple shooting the control functions are defined by a relatively small number of NLP variables. For multiple shooting and transcription methods, the number of NLP variables used to describe the controls increase ultimately including values at each mesh point in the interval.

Throughout the paper we have emphasized the similarity between methods. All of the methods utilize a Newton-based iteration to adjust a finite set of variables. The methods can be distinguished by identifying the set of iteration variables and constraints. The optimal control necessary conditions can be interpreted as limiting forms of the NLP Kuhn-Tucker necessary conditions. At the present time perhaps the most widely used methods are direct shooting, indirect multiple shooting, and direct transcription. Each method has advantages and disadvantages and we have attempted to highlight them. Future research and development will undoubtedly focus on removing deficiencies in these techniques. Progress in the analysis of high-index differential-algebraic equations, automatic differentiation, and sparse nonlinear programming will certainly lead to refinements in existing software and methods. In fact one may expect many of the best features of seemingly disparate techniques to merge, forming still more powerful methods.

References

[1] W. P. Hallman, "Mission Timeline for a Shuttle-IUS Flight Using a Nonstandard Shuttle Park Orbit", Technical Operating Report TOR-0083-(3493-14), The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, Oct 1982.

[2] S. J. Citron, *Elements of Optimal Control*, Holt, Rinehart and Winston, New York, 1969.

[3] R. Fletcher, *Practical Methods of Optimization, Vol. 2, Constrained Optimization*, John Wiley and Sons,, New York, New York, 1985.

[4] P. E. Gill, Walter Murray, and Margaret H. Wright, *Practical Optimization*, Academic Press, Inc., London, 1981.

[5] J. E. Dennis, Jr. and Robert B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 07632, 1983.

[6] P. E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright, "Some Theoretical Properties of an Augmented Lagrangian Merit Function", Tech. Report Report SOL 86-6, Department of Operations Research, Stanford University, April 1986.

[7] P. E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright, "User's Guide for NPSOL (version 4.0): a Fortran Package for Nonlinear Programming", Tech. Report Report SOL 86-2, Department of Operations Research, Stanford University, Jan 1986.

[8] J. T. Betts and Wayne P. Hallman, "NLP2 Optimization Algorithm Documentation", Technical Operating Report TOR-0089(4464-06)-1, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, Jan 1997. Reissue A.

[9] J. B. Rosen and J. Kreuser, "A Gradient Projection Algorithm for Nonlinear Constraints", in Numerical Methods for Non-Linear Optimization, F. A. Lootsma, ed., Academic Press, London and New York, 1972, pp. 297–300.

[10] L. S. Lasdon and A. D. Waren, "Generalized Reduced Gradient Software for Linearly and Nonlinearly Constrained Optimization", in Design and Implementation of Optimization Software, H. J. Greenberg, ed., Sijthoff and Noordhoff, Netherlands, 1978, pp. 335–362.

[11] J. T. Betts and Paul D. Frank, "A Sparse Nonlinear Optimization Algorithm", Journal of Optimization Theory and Applications, 82 (1994), pp. 519–541.

[12] J. T. Betts, Michael J. Carter, and William P. Huffman, "Software For Nonlinear Optimization", Mathematics and Engineering Analysis Library Report MEA-LR-83 R1, Boeing Information and Support Services, The Boeing Company, PO Box 3707, Seattle, WA 98124-2207, Jun 1997.

[13] C. C. Ashcraft, Roger G. Grimes, and John G. Lewis, "Accurate Symmetric Indefinite Linear Equation Solvers". Submitted for publication, SIAM Journal of Matix Analysis, 1996.

[14] J. T. Betts, Samuel K. Eldersveld, and William P. Huffman, "A Performance Comparison of Nonlinear programming Algorithms for Large Sparse Problems", in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, AIAA-93-3751-CP, Monterey, CA, Aug 1993, pp. 443–455.

[15] J. T. Betts and William P. Huffman, "Sparse Nonlinear programming Test Problems (release 1.0)", BCS Technology Technical Report BCSTECH-93-047, Boeing Computer Services, The Boeing Company, PO Box 3707, Seattle, WA 98124-2207, 1993.

[16] P. E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright, "A Schur-complement Method for Sparse Quadratic Programming", Tech. Report Report SOL 87-12, Department of Operations Research, Stanford University, Oct 1987.

[17] J. T. Betts, "Sparse Jacobian Updates in the Collocation Method for Optimal Control Problems", AIAA Journal of Guidance, Control, and Dynamics, 13 (1990), pp. 409–415.

[18] T. F. Coleman and Jorge J. Moré, "Estimation of Sparse Jacobian Matrices and Graph Coloring Problems", SIAM Journal of Numerical Analysis, 20 (1983), pp. 187–209.

[19] A. R. Curtis, M. Powell, and J. Reid, "On the Estimation of Sparse Jacobian Matrices", Journal of the Institute of Mathematics and Applications, 13 (1974), pp. 117–120.

[20] P. E. Gill, Walter Murray, and Michael A. Saunders, "SNOPT: An SQP Algorithm for Large-scale Constrained Optimization", Tech. Report SOL 96-0, Department of Operations Research, Stanford University, 1996.

[21] G. A. Bliss, *Lectures on the Calculus of Variations*, University of Chicago Press, Chicago, IL, 1946.

[22] L. S. Pontryagin, *The Mathematical Theory of Optimal Processes*, Wiley-Interscience, New York, NY, 1962.

[23] A. E. Bryson, Jr. and Yu Chi Ho, *Applied Optimal Control*, John Wiley & Sons, New York, NY, 1975.

[24] U. M. Ascher, Robert M. M. Mattheij, and Robert D. Russell, *Numerical Solution of Boundary Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[25] K. E. Brenan, Stephen L. Campbell, and Linda R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, North-Holland, New York, NY, 1989.

[26] H. S. Tsien and R. C. Evans, "Optimum Thrust Programming for a Sounding Rocket", Journal of the American Rocket Society, 21 (1951), pp. 99–107.

[27] J. L. Speyer, "Periodic Optimal Flight", AIAA Journal of Guidance, Control, and Dynamics, 19 (1996), pp. 745–755.

[28] G. Sachs and Klaus Lesch, "Periodic Maximum Range Cruise with Singular Control", AIAA Journal of Guidance, Control, and Dynamics, 16 (1993), pp. 790–793.

[29] P. Berkmann and Hans Josef Pesch, "Abort Landing in Windshear: Optimal Control Problem with Third-Order State Constraint and Varied Switching Structure", Journal of Optimization Theory and Applications, 85 (1995), pp. 21–57.

[30] H. Josef Pesch, "A Practical Guide to the Solution of Real-life Optimal Control Problems", Control and Cybernetics, 23 (1994), pp. 7–60.

[31] M. D. Canon, C. D. Cullum, and E. Polak, *Theory of Optimal Control and Mathematical Programming*, McGraw-Hill, New York, 1970.

[32] E. Polak, *Computational Methods in Optimization*, Academic Press, New York, 1971.

[33] D. Tabak and B. C. Kuo, *Optimal Control by Mathematical Programming*, Prentice-Hall, New Jersey, 1971.

[34] P. J. Enright and Bruce A. Conway, "Discrete Approximations to Optimal Trajectories Using Direct Transcription and Nonlinear Programming", AIAA Journal of Guidance, Control, and Dynamics, 15 (1992), pp. 994–1002.

[35] A. L. Herman and Bruce A. Conway, “Direct Optimization using Collocation Based on High-Order Gauss-Lobatto Quadrature Rules”, AIAA Journal of Guidance, Control, and Dynamics, 19 (1996), pp. 592–599.

[36] O. von Stryk and Roland Bulirsch, “Direct and Indirect Methods for Trajectory Optimization”, Annals of Operations Research, 37 (1992), pp. 357–373.

[37] O. von Stryk, “Numerical Solution of Optimal control Problems by Direct Collocation”, in Optimal Control, R. Bulirsch, Angelo Miele, Josef Stoer, and Klaus H. Well, eds., vol. 111 of International Series of Numerical Mathematics, Basel, 1993, Birkhäuser Verlag, pp. 129–143.

[38] A. J. Calise, “Singular Perturbation Techniques for On-line Optimal Flight-path Control”, AIAA Journal of Guidance and Control, 3 (1981), pp. 398–405.

[39] A. J. Calise and Daniel D. Moerder, “Singular Perturbation Techniques for Real Time Aircraft Trajectory Optimization and Control”, Tech. Report CR-3597, NASA, Aug 1982.

[40] F. Bashforth and J. C. Adams, *Theories of Capillary Action*, Cambridge University Press, New York, NY, 1883.

[41] F. R. Moulton, *New Methods in Exterior Ballistics*, University of Chicago, Chicago, IL, 1926.

[42] G. Dahlquist and Å. Björk, *Numerical Methods*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1974.

[43] J. Stoer and Roland Bulirsch, *Introduction to Numerical Analysis*, Springer, New York, Berlin, Heidleberg, 1980.

[44] A. C. Hindmarsh, “ODEPACK, A Systematized Collection of ODE Solvers”, in Scientific Computing, e. a. R. S. Stepleman, ed., North Holland, Amsterdam, 1983, pp. 55–64.

[45] L. F. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman and Co., San Francisco, 1975.

[46] C. William Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[47] H. Seywald, “Trajectory Optimization Based on Differential Inclusion”, AIAA Journal of Guidance, Control, and Dynamics, 17 (1994), pp. 480–487.

[48] C. William Gear, “The Simultaneous Numerical Solution of Differential-Algebraic equations”, IEEE Transactions on Circuit Theory, CT-18 (1971), pp. 89–95.

[49] L. R. Petzold, “Differential/Algebraic Equations are not ODEs”, SIAM Journal of Scientific and Statistical Computing, 3 (1982), pp. 367–384.

[50] L. R. Petzold, “A Description of DASSL:, A Differential/Algebraic System Solver”, in Scientific Computing, e. a. R. S. Stepleman, ed., North Holland, Amsterdam, 1983, pp. 65–68.

[51] W. P. Hallman, “Smooth Curve Fits for the Shuttle Solid Rocket Booster Data”, Interoffice Correpondence IOC No. A85-5752.5-05, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, Mar 1985.

[52] R. A. Luke, “Computational Efficiency Considerations for High-fidelity Launch Vehicle Trajectory Optimization”, in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, AIAA-89-3446-CP, Aug 1989, pp. 181–192.

[53] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[54] D. R. Ferguson and Richard A. Mastro, “Modeling and Analysis of Aerodynamic Data II. Practical Experience”, in *Proceedings of the AIAA-AHS-ASEE Aircraft Design, Systems and Operations Conference*, AIAA-89-2076, Seattle, WA, Jul–Aug 1989.

[55] D. R. Ferguson and Richard A. Mastro, “Modeling and Analysis of Aerodynamic Data II”, in *Proceedings of the 27th Aerospace Sciences Meeting*, AIAA-89-0476, Reno, NV, Jun 1989.

[56] D. R. Ferguson, “Construction of Curves and Surfaces using Numerical Optimization Techniques”, CAD, 18 (1986), pp. 15–21.

[57] J. T. Betts, “The Application of Sparse Least Squares in Aerospace Design problems”, in Optimal Design and Control, Proceedings of the Workshop on Optimal Design and Control, J. Borggaard, John Burkardt, Max Gunzburger, and Janet Peterson, eds., vol. 19 of Progress in Systems and Control Theory, Birkhäuser, Basel, Apr 1994, pp. 81–96.

[58] G. L. Brauer, D. E. Cornick, and R. Stevenson, “Capabilites and Applications of the Program to Optimize Simulated Trajectories (POST)”, Tech. Report CR-2770, NASA, Feb 1977.

[59] D. S. Meder and Jerry L. Searcy, “Generalized Trajectory Simulation (GTS), Volumes I-V”, Technical Report SAMSO-TR-75-255, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, Jan 1975.

[60] J. T. Betts, “Optimal Three-burn Orbit Transfer”, AIAA Journal, 15 (1977), pp. 861–864.

[61] R. G. Brusch, “Constrained Impulsive Trajectory Optimization for Orbit-to-Orbit Transfer”, AIAA Journal of Guidance, Control, and Dynamics, 2 (1979), pp. 204–212.

[62] T. P. Bauer, John T. Betts, Wayne P. Hallman, William P. Huffman, and K. P. Zondervan, “Solving the Optimal Control Problem Using a Nonlinear Programming Technique Part 2: Optimal Shuttle Ascent Trajectories”, in *Proceedings of the AIAA/AAS Astrodynamics Conference*, AIAA-84-2038, Seattle, WA, Aug 1984.

[63] K. E. Brenan, “Engineering Methods Report: The design and Development of a Consistent Integrator/interpolator For Optimization Problems”, Aerospace Technical Memorandum ATM No. 88(9975)-52, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, 1988.

[64] C. William Gear and Thu Van Vu, “Smooth Numerical Solutions of Ordinary Differential Equations”, in *Proceedings of the Workshop on Numerical Treatment of Inverse Problems for Differential and Integral Equations*, Heidelberg, 1982.

[65] T. Van Vu, “Numerical Methods for Smooth Solutions of Ordinary Differential Equations”, Tech. Report No. UIUCDCS-R-83-1130, Dept. of Computer Science, University of Illinois, Urbana, IL, May 1983.

[66] J. T. Betts, “Frontiers in Engineering Optimization”, Journal of Mechanisms, Transmissions, and Automation in Design, 105 (1983), pp. 151–154.

[67] L. R. Balkanyi and Omer F. Spurlock, “DUKSUP - A High Thrust Trajectory Optimization Code”, in *AIAA/AHS/ASEE Aerospace Design Conference*, AIAA 93-1127, Irvine, CA, Feb 1993.

[68] T. Edelbaum, L. Sackett, and H. Malchow, “Optimal Low Thrust Geocentric Transfer”, in *AIAA 10th Electric Propulsion Conference*, AIAA 73-1074, Lake Tahoe, NV, Oct–Nov 1973.

[69] C. Bischof, Alan Carle, George Corliss, Andreas Griewank, and P. Hovland, “ADIFOR: Generating derivative codes from Fortran programs”, Scientific Programming, 1 (1992), pp. 11–29.

[70] R. Schöpf and P. Deulfhard, “OCCAL: A mixed symbolic-numeric Optimal Control CALculator”, in Computational Optimal Control, R. Bulirsch and Dieter Kraft, eds., vol. 115 of International Series of Numerical Mathematics, Birkhäuser Verlag, Basel, 1994, pp. 269–278.

[71] R. Mehlhorn and Gottfried Sachs, “A New Tool for Efficient Optimization by Automatic Differentiation and Program Transparency”, Optimization Methods and Software, 4 (1994), pp. 225–242.

[72] Roland Bulirsch, “Die Mehrzielmethode zur numerischen Lösung von nichtlinearen Randwertproblemen und Aufgaben der optimalen Steuerung”, Report of the Carl-Cranz Gesellschaft, Carl-Cranz Gesellschaft, Oberpfaffenhofen, Germany, 1971.

[73] H. B. Keller, *Numerical Methods for Two-Point Boundary Value Problems*, Waltham: Blaisdell, London, 1968.

[74] H. J. Oberle and Werner Grimm, "BNDSCO – A Program for the Numerical Solution of Optimal control Problems, User Guide", Internal Report DLR IB/515-89/22, Institut für Dynamik der Flugsysteme, Deutsche Forschungsanstalt für Luft und Raumfahrt DLR, Oberpfaffenhofen, Germany, 1989.

[75] R. Callies, "Optimal Design of a Mission to Neptune", in Optimal Control, R. Bulirsch, Angelo Miele, Josef Stoer, and Klaus H. Well, eds., vol. 111 of International Series of Numerical Mathematics, Basel, 1993, Birkhäuser Verlag, pp. 341–349.

[76] H. Kreim, Bernd Kugelmann, Hans Josef Pesch, and Michael H. Breitner, "Minimizing the Maximum Heating of a Re-entering Space Shuttle: An Optimal Control Problem with Multiple Control Constraints", Optimal Control Applications and Methods, 17 (1996), pp. 45–69.

[77] H. Georg Bock and K. J. Plitt, "A Multiple Shooting Algorithm for Direct Solution of Optimal control Problems", in *Proceedings of the 9th IFAC World Congress*, Budapest, Hungary, 1984, Pergamon Press, pp. 242–247.

[78] J. T. Betts and William P. Huffman, "Trajectory Optimization on a Parallel Processor", AIAA Journal of Guidance, Control, and Dynamics, 14 (1991), pp. 431–439.

[79] E. D. Dickmanns and Klaus H. Well, "Approximate Solution of Optimal Control Problems Using Third-Order Hermite Polynomial Functions", in *Proceedings of the 6th Technical Conference on Optimization Techniques*, vol. IFIP-TC7, New York, 1975, Springer-Verlag.

[80] R. D. Russell and Lawerence F. Shampine, "A Collocation Method for Boundary Value Problems", Numerische Mathematik, 19 (1972), pp. 13–36.

[81] E. D. Dickmanns, "Efficient Convergence and Mesh Refinement Strategies for Solving General Ordinary Two-point Boundary Value Problems by Collocated Hermite Approximation". 2nd IFAC Workshop on Optimisation, Oberpfaffenhofen, Sep 1980.

[82] U. M. Ascher, J. Christiansen, and Robert D. Russell, “COLSYS–a Collocation Code for Boundary-Value Problems”, in Codes for Boundary Value Problems in Ordinary Differential Equations, B. Childs, M. Scott, J. W. Daniel, E. Denman, and P. Nelson, eds., Springer Verlag, Berlin, 1979. Lecture Notes in Computer Science 76.

[83] D. G. Hull, “Conversion of Optimal Control Problems into Parameter Optimization Problems”, AIAA Journal of Guidance, Control, and Dynamics, 20 (1997), pp. 57–60.

[84] C. R. Hargraves and Stephen W. Paris, “Direct Trajectory Optimization Using Nonlinear Programming and Collocation”, AIAA Journal of Guidance, Control, and Dynamics, 10 (1987), p. 338.

[85] J. T. Betts and William P. Huffman, “Application of Sparse Nonlinear Programming to Trajectory Optimization”, AIAA Journal of Guidance, Control, and Dynamics, 15 (1992), pp. 198–206.

[86] J. T. Betts and William P. Huffman, “Path Constrained Trajectory Optimization Using Sparse Sequential Quadratic programming”, AIAA Journal of Guidance, Control, and Dynamics, 16 (1993), pp. 59–68.

[87] J. T. Betts, “Issues in the Direct Transcription of Optimal Control Problems to Sparse Nonlinear Programs”, in Computational Optimal Control, R. Bulirsch and Dieter Kraft, eds., vol. 115 of International Series of Numerical Mathematics, Birkhäuser Verlag, Basel, 1994, pp. 3–18.

[88] J. T. Betts, “Trajectory Optimization Using Sparse Sequential Quadratic Programming”, in Optimal Control, R. Bulirsch, Angelo Miele, Josef Stoer, and Klaus H. Well, eds., vol. 111 of International Series of Numerical Mathematics, Basel, 1993, Birkhäuser Verlag, pp. 115–128.

[89] J. T. Betts and William P. Huffman, “Sparse Optimal Control Software SOCS”, Mathematics and Engineering Analysis Technical Document MEA-LR-085, Boeing Information and Support Services, The Boeing Company, PO Box 3707, Seattle, WA 98124-2207, Jul 1997.

[90] J. T. Betts and William P. Huffman, “Mesh Refinement in Direct Transcription Methods for Optimal Control”, Optimal Control Applications and Methods, 19 (1998), pp. 1–21.

[91] J. T. Betts, “Using Sparse Nonlinear Programming to Compute Low Thrust Orbit Transfers”, The Journal of the Astronautical Sciences, 41 (1993), pp. 349–371.

[92] J. T. Betts, “Optimal Interplanetary Orbit Transfers by Direct transcription”, The Journal of the Astronautical Sciences, 42 (1994), pp. 247–268.

[93] J. T. Betts and Evin J. Cramer, “Application of Direct Transcription to Commercial Aircraft Trajectory Optimization”, AIAA Journal of Guidance, Control, and Dynamics, 18 (1995), pp. 151–159.

[94] P. J. Enright and Bruce A. Conway, “Optimal Finite-thrust Spacecraft Trajectories Using Collocation and Nonlinear Programming”, AIAA Journal of Guidance, Control, and Dynamics, 14 (1991), pp. 981–985.

[95] A. J. Roenneke, Christian Jänsch, and A. Markl, “Advanced Launch and Reentry Trajectory Optimization Software documentation (ALTOS)”, tech. report, European Space Science and Technology Center, Noordwijk, The Netherlands, May 1995. Software User Manual, Draft 1.

[96] D. Kraft, “On Converting Optimal Control Problems Into Nonlinear Programming Problems”, in *Computational Mathematical Programming*, vol. F15 of NATO ASI Series, Springer-Verlag, 1985.

[97] K. E. Brenan and Wayne P. Hallman, “A Generalized Reduced Gradient Algorithm for Large-scale Trajectory Optimization Problems”, in Optimal Design and Control, Proceedings of the Workshop on Optimal Design and Control, J. Borggaard, John Burkardt, Max Gunzburger, and Janet Peterson, eds., vol. 19 of Progress in Systems and Control Theory, Birkhäuser, Basel, Apr 1994, pp. 117–132.

[98] Marc C. Steinbach, “A Structured Interior Point SQP Method for Nonlinear Optimal Control Problems”, in Computational Optimal Control, R. Bulirsch and Dieter Kraft, eds., vol. 115 of International Series of Numerical Mathematics, Birkhäuser Verlag, Basel, 1994, pp. 213–222.

[99] J. R. Downey and Bruce A. Conway, “The Solution of Singular Optimal Control Problems using Direct Collocation and Nonlinear Programming”, in *AIAA/AAS Astrodynamics Specialist Conference*, AAS 91-443, Durango, CO, Aug 1991.

[100] J. S. Logsdon and Lorenz T. Biegler, “Accurate Solution of Differential-algebraic Optimization Problems”, Industrial Engineering Chemical Research, 28 (1989), pp. 1628–1639.

[101] R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, N.J., 1957.