Application of Sparse Nonlinear Programming to Trajectory Optimization

John T. Betts* and William P. Huffman*
Boeing Computer Services, Seattle, Washington 98124

The most effective numerical techniques for the solution of trajectory optimization and optimal control problems combine a nonlinear iteration procedure with some type of parametric approximation to the trajectory dynamics. Early methods attempted to parameterize the dynamics using a small number of variables because the iterative search procedures could not successfully solve larger problems. With the development of more robust nonlinear programming algorithms, it is now feasible and desirable to consider formulations of the trajectory optimization problem incorporating a large number of variables and constraints. The purpose of this paper is to address the manner in which a trajectory is parameterized and the design of the nonlinear programming algorithm to effectively deal with this formulation.

I. Introduction

THE optimal design of a trajectory is a problem that can be solved using a wide variety of seemingly different computational techniques. Most state-of-the-art trajectory optimization programs combine a nonlinear iteration technique with a numerical trajectory simulation. The iteration techniques are based on approximating the nonlinear constraint functions by linear models and the objective function by a quadratic model. When an iterative technique based on a quadratic-linear model is applied to a trajectory problem that is well approximated by this model, the resulting method can be quite robust and efficient. Many ascent vehicle trajectories can be parameterized in terms of a small number of variables and constraints that are well represented by the quadratic-linear model. Consequently, for these applications state-of-the-art nonlinear programming codes employing dense linear algebra are quite effective. Successful application of the technique to orbit transfer and launch vehicle trajectory design are well documented,[1-3] and production quality software utilizing the approach includes the POST program[4] and the GTS program.[5]

Ascent vehicle trajectories are characterized by dynamics in which the dominant force is thrust and yield relatively well-behaved trajectories. When the trajectory dynamics are more nonlinear, such as in aerodynamic re-entry problems, it is more difficult to construct a parameterization of the trajectory using a small number of variables and contraints that are well approximated by a quadratic-linear model. Consequently, standard shooting methods do not work well for problems in this class. However, it is possible to construct a formulation that is well approximated by a quadratic-linear model simply by increasing the number of variables and constraints. Methods based on multiple shooting and collocation fall in this class[6,7] and have been found to be quite effective for the solution of nonlinear boundary value problems. An approach that combines Hermite collocation with a dense successive quadratic programming (SQP) nonlinear optimization algorithm has been implemented by Hargraves and Paris[8] in the OTIS program. Computational experience indicates the method is robust, even on problems with very nonlinear dynamics.

Received May 22, 1990; revision received Aug. 31, 1990; accepted for publication Sept. 11, 1990. Copyright © 1990 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

*Applied Mathematician, Applied Mathematics and Statistics, P.O. Box 24346, MS 7L-21.

This paper describes how sparse matrix methods can be incorporated into a nonlinear programming algorithm for the solution of trajectory optimization problems. Alternate formulations of the trajectory problem that can more fully exploit sparsity are suggested. Computational experience on a collection of ascent and re-entry trajectory problems is presented to corroborate the theoretical results.

II. Nonlinear Programming Problem

The nonlinear programming problem can be stated as follows: find the N-vector x that minimizes the objective function

$$f(x) \tag{1}$$

subject to the m_e equality constraints

$$c_i(x) = 0 \tag{2}$$

for $i = 1, \ldots, m_e$ and the m_i inequality constraints

$$c_i(x) \geq 0 \tag{3}$$

for $i = (m_e + 1), \ldots, m$, where the total number of constraints is $m = m_e + m_i$. The number of constraints m can be less than, equal to, or greater than the number of variables N. For clarity in presentation, it has been assumed that the first m_e constraints are equalities, and the remaining are inequalities.

It is assumed that the functions are continuously differentiable to second order, although in practice these attributes must be inferred. In fact, it may be necessary for the optimization algorithm to detect discontinuities and respond appropriately. Furthermore, the functions may not be computable for all x. It is convenient to visualize the function generator as a black box that either 1) evaluates the problem functions f and c_i or 2) sets a flag (the function error flag) indicating that evaluation is impossible. Thus the functions are computable over some region Ω, which, in general, cannot be precisely defined.

Define the basic set of constraints by

$$\mathcal{Z}^* = \{i \mid c_i(x^*) = 0, \qquad i \in [1, \ldots, m]\} \tag{4}$$

The set consists of all equality constraints and all inequality constraints that are zero at the solution point x^* and hence is denoted by the symbol $\mathcal{Z}^*$. For consistency, denote the set of constraints that are strictly positive at the solution by $\mathcal{P}$. An estimate of the basic set $\mathcal{Z}$ is referred to as a basis or active set. The Kuhn-Tucker necessary conditions for a local minimum require that

$$g(x^*) + [G(x^*)]^T\lambda^* = 0 \tag{5}$$

where $\nabla_x f(x) = g(x)$ is the N-dimensional gradient vector, $G(x^*)$ is the $m \times N$ Jacobian matrix, and λ^* is the m vector of Lagrange multipliers. Furthermore,

$$c_i(x^*) = 0 \tag{6}$$

for $i = 1, \ldots, m_e$ and

$$\lambda_i^* \, c_i(x^*) = 0 \tag{7}$$

with

$$\lambda_i^* \leq 0 \tag{8}$$

for $i = (m_e + 1), \ldots, m$. These optimality conditions are necessary for a local solution.

The most common way to treat inequality constraints is to solve a series of equality-constrained subproblems constructed by making an estimate of the active set of constraints. If the active set is correct, then the solution of the equality-constrained problem is also the solution of the inequality constrained problem. For clarity in presentation, we shall restrict the remaining discussion to problems involving only the m_z active constraints. It should be noted that for many trajectory problems the number of equality constraints is large relative to the number of variables, and the number of degrees of freedom $n = N - m_z$ is small. A discussion of the impact of inequalities on algorithm philosophy is included.

The necessary conditions for the equality-constrained problem can be viewed as a system of nonlinear equations in the unknowns (x, λ), namely,

$$\nabla_x L(x, \lambda) = g(x) + G^T(x)\lambda = 0 \tag{9}$$

$$\nabla_\lambda L(x, \lambda) = c(x) = 0 \tag{10}$$

where the Lagrangian

$$L(x, \lambda) = f(x) + \lambda^T c(x) \tag{11}$$

Suppose we want to construct a new estimate of the unknowns $(\bar{x}, \bar{\lambda})$ from information evaluated at the current point (x, λ). Direct application of Newton's method to the set of nonlinear equations, Eqs. (9) and (10), yields the linear system

$$\begin{bmatrix} H & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} s \\ -\bar{\lambda} \end{bmatrix} = \begin{bmatrix} g \\ c \end{bmatrix} \tag{12}$$

Solution of this system of linear equations defines the new variables according to the formula

$$\bar{x} = x - \rho s \tag{13}$$

and the vector s is referred to as the search direction. The scalar ρ determines the step length and is typically set to one. The symmetric $N \times N$ Hessian matrix of the Lagrangian is defined by

$$H = \nabla_x^2 f + \sum_{i=1}^{m_z} \lambda_i \nabla_x^2 c_i \tag{14}$$

It can be demonstrated that an equivalent way of defining the search direction s is to minimize the quadratic

$$f(\bar{x}) = f(x) - g^T s + \tfrac{1}{2} s^T H s \tag{15}$$

subject to the linear constraints

$$Gs = c \tag{16}$$

hence explaining the quadratic-linear aspect of the model.

Most state-of-the-art nonlinear programming algorithms construct iterates that can be be derived from the solution to the Kuhn-Tucker (KT) system, Eq. (12). There are two primary computational costs associated with this system, namely, 1) constructing the elements G and H and 2) solving the system. When most of the elements in the Jacobian G and Hessian H are nonzero, the problem is said to be dense. In contrast, if most of the elements in these matrices are zero, the problem is considered to be sparse. For trajectory optimization problems, it is common to have fewer than 1% of the elements be nonzero. Sparsity in the problem affects the design of the optimization algorithm in a number of significant ways.

For dense problems, it is common to construct the Jacobian and Hessian matrices using recursive updates. Update formulas typically produce algorithms in which the number of iterations is proportional to the number degrees of freedom n. Unfortunately, when n is large, this superlinear convergence rate may be prohibitive. In contrast, when the exact Jacobian and Hessian are available, iterations based on Eq. (12) will converge quadratically, i.e., the rate of convergence is not dictated by the problem size! Calculation of the Jacobian and Hessian using sparse finite differencing will be discussed later.

For dense problems, the computational cost of solving a linear system is proportional to N^3. In contrast, the solution of a sparse linear system can be achieved in $\mathcal{O}(c^2 N)$ operations where the number of nonzeros is $\tau = cN$. However, to exploit this sparsity, it is critical that operations performed by the optimization algorithm do not create fill, that is, introduce nonzero elements into a previously sparse structure. In particular, note that the inverse of a sparse matrix is not necessarily sparse. Furthermore, many projection operations commonly perfomed in dense optimization algorithms also destroy sparsity. For example, whereas the Hessian matrix H is sparse, the projected Hessian $P^T H P$ is not necessarily sparse.

In summary, it appears that the use of popular recursive update formulas (Broyden, BFGS, Symmetric Rank One, etc.) is not promising for sparse problems. Furthermore, it is clear that linear algebra operations commonly performed for dense problems are no longer appropriate for sparse applications. With this in mind, the following section describes an algorithm for sparse nonlinear programming.

III. Sparse Nonlinear Programming Algorithm

A. Sparse Linear Algebra

Development of efficient, robust software for the solution of sparse linear systems is a field of active research. The nonlinear programming algorithm to be described employs a state-of-the-art linear algebra package. The package solves $Ax = b$ for x where A is an $n \times n$ real symmetric indefinite sparse matrix. Since A is symmetric, it can be factored using a square-root-free Cholesky factorization $A = LDL^T$, where L is a unit lower-triangular matrix and D is a diagonal matrix. Since A is not necessarily positive definite, pivoting to preserve stability is required. The software requires storage for the nonzero elements in the lower triangular portion of the matrix and a work array. A complete description of the multifrontal algorithm is found in Refs. 9 and 10.

B. Initial Feasible Point

The algorithm to be presented is a feasible point method. The first step in an algorithm of this type is to locate a feasible point, i.e., find any point satisfying the constraints. Although other strategies are possible, this approach is motivated by the fact that the number of equality constraints is large for the proposed applications and points that satisfy the constraints represent real, though suboptimal, trajectories.

The approach employed is to take a series of steps of the form given by Eq. (13) with the search direction computed to solve the underdetermined linear system given by Eq. (16). However, since the solution is not unique, we impose the additional requirement that the search direction have minimum norm, i.e., $\| s \|$. But the minimum norm solution can be obtained from Eq. (12) by setting $H = I$ and $g = 0$ to give the linear system

$$\begin{bmatrix} I & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} s \\ -\bar{\nu} \end{bmatrix} = \begin{bmatrix} 0 \\ c \end{bmatrix} \tag{17}$$

This symmetric indefinite system can be solved efficiently by the multifrontal algorithm described previously. Since the solution of this system is based on a linear model of the constraint functions, it may be necessary to adjust the step length ρ in Eq. (13) to produce a reduction in the constraint error. Specifically, a quadratic line search is used to adjust ρ so that $\| c(\bar{x}) \| \leq \| c(x) \|$. In summary, a feasible point is located by taking a series of steps, where each individual step is constructed from the linear constraint model, with the step length adjusted to produce a reduction in the constraint error.

C. Equality-Constrained Procedure

After a feasible point has been located, the optimization of the objective function is undertaken. This section describes the method for solving an equality-constrained optimization problem. The iteration begins at the point x, which satisfies the constraints, i.e. $c(x) = 0$, and proceeds as follows:

1) Terminate if the Kuhn-Tucker conditions are satisfied; otherwise proceed to step 2.

2) Construct the optimization search direction by solving the KT system, Eq. (12), and initialize $t = u = 0$, $\rho = 1$, and $\tilde{\rho} = 0$.

3) Compute the predicted point from

$$\bar{x} = x - \rho s + \rho^2 t + \rho^3 u \tag{18}$$

and evaluate the constraints $\bar{c} = c(\bar{x})$ at the predicted point, then a) if $\| \bar{c} \| \leq \epsilon$, set $\hat{x} \Leftarrow \bar{x}$ and go to step 7 or else b) proceed to step 4.

4) Solve the underdetermined system

$$Gd = \bar{c} \tag{19}$$

for the direction d and initialize $\mu = 1$.

5) Compute the corrected point

$$\hat{x} = \bar{x} - \mu d \tag{20}$$

6) Evaluate the constraints $\hat{c}$ at the corrected point $\hat{x}$, then a) if $\| \hat{c} \| \leq \epsilon$ and $\tilde{\rho} = 0$ compute

$$t = (\hat{x} - x + \rho s)/\rho^2 \tag{21}$$

save the corrected point (set $\tilde{x} \Leftarrow \hat{x}$, and $\tilde{\rho} \Leftarrow \rho$), and then go to step 7, or else b) if $\| \hat{c} \| \leq \epsilon$ and $\tilde{\rho} \neq 0$, compute the elements of t_i and u_i for $i = 1, \ldots, N$ from the system

$$\begin{bmatrix} \rho^2 & \rho^3 \\ \tilde{\rho}^2 & \tilde{\rho}^3 \end{bmatrix} \begin{bmatrix} t_i \\ u_i \end{bmatrix} = \begin{bmatrix} \hat{x}_i - x_i + \rho s_i \\ \tilde{x}_i - x_i + \tilde{\rho} s_i \end{bmatrix} \tag{22}$$

save the corrected point (set $\tilde{x} \Leftarrow \hat{x}$, and $\tilde{\rho} \Leftarrow \rho$), and then go to step 7, or else c) if $\| \hat{c} \| \leq \| \bar{c} \|$, update the corrected point (set $\bar{x} \Leftarrow \hat{x}$, and $\bar{c} \Leftarrow \hat{c}$), and return to step 4, or else d) reduce the step length μ to achieve constraint reduction and return to step 5.

7) Evaluate the Lagrangian $\hat{L} = L(\hat{x}, \lambda)$ on the constraint surface and a) if the Lagrangian $\hat{L}$ is sufficiently less than L, then $\hat{x}$ is an improved feasible point—update all quantities and return to step 1, or else b) change the step length ρ to minimize L and return to step 3.

The steps outlined describe the fundamental elements of the optimization process; however, a number of points deserve additional clarification. First, note the the algorithm consists of an outer loop (steps 2–7) to minimize the Lagrangian function and an inner loop (steps 3–6) to eliminate the error in the active constraints. The outer loop can be viewed as a univariate (line) search in the direction s, with the step length ρ adjusted to minimize the Lagrangian restricted to the constraint surface. The inner loop can be viewed as a nonlinear root-solving process designed to eliminate the error in the active constraints for the specified value of ρ. The inner constraint elimination process must be initiated with an estimate of the variables, and this estimate is given by the expression Eq. (18). Notice that the first prediction is based on a linear model for the constraints since $t = u = 0$ in step 2. However, after the constraint error has been eliminated, the value of t is updated by Eq. (21) and the second prediction is based on a quadratic model of the contraint. After the second corrected point is obtained, subsequent predictions utilize the cubic model defined by solving Eq. (22). Adjusting the value of the step length ρ as required in step 7b is accomplished using a univariate search procedure described in Ref. 11, which constructs a quadratic model of the Lagrangian.

Because the constraint elmination process requires the solution of the underdetermined system Eq. (19), there is some ambiguity in the algorithm. This ambiguity is eliminated by choosing the minimum norm direction that can be obtained by solving the augmented system

$$\begin{bmatrix} I & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} d \\ -\bar{\nu} \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{c} \end{bmatrix} \tag{23}$$

Notice that the Jacobian G is evaluated at the reference point x and not re-evaluated during the inner loop iteration, even though the right-hand side $\bar{c}$ does change. Because the Jacobian is not re-evaluated, the inner loop will have a linear convergence rate. Nevertheless, this approach has been found attractive because 1) the coefficient matrix can be factored only once per outer optimization iteration, thereby significantly reducing the linear algebra expense, and 2) the corrections defined by d are orthogonal to the constraint tangent space at the reference point x and hence tend to produce a well-conditioned constraint iteration process.

An alternate approach for dealing with the underdetermined system Eq. (19) is to partition the problem in a manner similar to that used for a generalized reduced gradient (GRG) algorithm. While the standard GRG philosophy can exploit sparsity in the Jacobian, in general the reduced Hessian matrix will be dense. Also, the partition itself can be poorly behaved since it is not locally orthogonal to the constraint surface. This approach was tested in a preliminary version of the algorithm and found to be less effective.

To evaluate the Hessian matrix Eq. (14), an estimate of the Lagrange multipliers is needed. The values $\bar{\nu}$ obtained by solving Eq. (17) are used for the first iteration, and thereafter the values $\bar{\lambda}$ from Eq. (12) are used.

D. Algorithm Strategy

Inequality-constrained problems can be solved as a series of equality-constrained problems using an active set strategy. The active set strategy utilized in the current implementation is described in Ref. 11. Computational experience with this approach suggests it is effective for a limited number of inequality constraints. However, as the number of inequalities becomes large, identification of the active set becomes combinatorial in nature, and the current approach may become inefficient. Efficient identification of the active set may be possible with either a sparse sequential quadratic programming or barrier function method.

The algorithm described is a feasible region method, since successive iterates maintain constraint feasibility. This philosophy is motivated by a number of considerations. When the constraints are satisfied, the variables describe a valid trajectory for a fixed discretization history. Mesh refinement to improve the accuracy of the discretization is meaningful when performed about a valid trajectory and may not be elsewhere. Secondly, vehicle characteristics (e.g., aerodynamic and propulsion data) are usually only valid in regions about real trajectories. Finally, in practice many problems are poorly posed, and this situation is readily detected when attempting to locate a feasible point. As yet it is not clear how to best combine the desirable properties of a feasible region method with the active set definition mechanism of a sequential quadratic programming approach.

In the neighborhood of a solution, it has been observed that the predicted values of the constraints are accurate enough to satisfy the convergence tolerances without any inner iterations (steps 3–6). When this occurs, the algorithm becomes a standard Newton method. Although no proof is given, experimental evidence tends to confirm that the rate of convergence is second order. It should be noted that the constraint satisfaction process greatly enhances the robustness and stability of the iterations at points where a standard Newton method would fail.

IV. Sparse Differences

The preceding section described a method for nonlinear optimization assuming the availability of gradient, Jacobian, and Hessian information. A method for efficiently obtaining this information when the relevent matrices are sparse is described in this section. Let us define the matrix of first derivatives of the vector $q = (c^T, f)^T$ by

$$D = \begin{bmatrix} G \\ g^T \end{bmatrix} \tag{24}$$

An efficient method for computing the sparse matrix D was suggested by Curtis et al.[12] Essentially the approach requires partitioning the column indices of D into subsets Γ^k. Each subset of the columns D has the property that there is at most one nonzero element in each row. Then define the perturbation direction vector by

$$\Delta^k = \sum_{j \in \Gamma^k} \delta_j e_j \tag{25}$$

where δ_j is the perturbation size for variable j, and e_j is a unit vector in direction j. Then for each nonzero row i of a column $j \in \Gamma^k$, the central difference approximation is

$$D_{ij} \approx \frac{1}{2\delta_j}[q_i(x+\Delta^k) - q_i(x-\Delta^k)] \tag{26}$$

Denote the total number of index sets Γ^k needed to span the columns of D by γ. Then a central difference estimate requires evaluations at 2γ perturbed points. For a sparse matrix, the number of index sets can be significantly less than the number of variables, i.e., $\gamma \ll N$. Consequently, the cost of evaluating a Jacobian using sparse differences can be substantially less than standard finite-difference methods.

In Ref. 7 it has been demonstrated that for many trajectory applications further simplification is possible. In particular, the computed functions q are often of the form

$$q = \tilde{q} + Ax \tag{27}$$

where the derivatives of $\tilde{q}$ are given by C, A is a constant matrix, and $D = C + A$. Because the sparsity pattern of C is even simpler than that of D, it is attractive to construct C by sparse differencing. After constructing index sets based on the sparsity pattern of C, the central difference estimate is

$$C_{ij} \approx \frac{1}{2\delta_j}[q_i(x+\Delta^k) - q_i(x-\Delta^k) - 2(A\Delta^k)_i] \tag{28}$$

Row i of $A\Delta^k$ is denoted by $(A\Delta^k)_i$ in these estimates. Thus, when the constant portion A is specified, finite differencing can be used to construct C, and D follows directly from the definition $D = C + A$.

It was demonstrated in Ref. 7 that trajectory optimization problems solved using a multiple shooting approach exhibit a natural structure that results in the blocks of C being uncoupled. The sparsity structure produced by a collocation method that will be discussed later is quite similar. In either case, constructing index sets to span the columns is achieved by choosing one column from each block of C. Clearly the block with the maximum number of columns defines the number of index sets needed. Thus the number of index sets is just equal to the maximum number of variables introduced in any block. It is especially important to note that the number of index sets is defined by the block size but does not grow with the number of blocks! Thus the number of perturbations does not grow when more phases are introduced in a multiple shooting method or when more grid points are introduced in a collocation method. Although the scheme is quite efficient, it may not be the optimal partitioning. Coleman and Moré[13] show that the problem of defining the minimum number of index sets that span the space can be formulated as a graph-coloring problem. Application of their algorithms might afford further computational efficiencies.

Equation (28) provides a way to construct the first derivative information, but the optimization algorithm presented in the previous section also requires the second derivative information in H. One approach is to treat the Hessian as the first derivative of the gradient vector and then construct index sets for this matrix. This approach has been implemented by Powell and Toint.[14] However, for the current application it is undesirable to construct separate index sets for both the first and second derivative operations. Instead, we have chosen to use the same index sets for both. When this approach is taken, function evaluations needed for the first derivatives are also used to construct second derivatives.

Denote a specific function in the vector q as q_α. From the defining property of an index set, note that for any pair of index sets Γ^i and Γ^j, there exists at most one $k \in \Gamma^i$ and one $\ell \in \Gamma^j$, such that the partial derivatives of q_α with respect to variables k and ℓ are nonzero. Thus the diagonal Hessian elements of q_α are given by

$$(\nabla_x^2 q_\alpha)_{kk} \approx \frac{1}{\delta_k^2}[q_\alpha(x+\Delta^i) + q_\alpha(x-\Delta^i) - 2q_\alpha(x)] \tag{29}$$

and the off-diagonal ($k \neq \ell$) terms are given by

$$(\nabla_x^2 q_\alpha)_{k\ell} \approx \frac{1}{\delta_k \delta_\ell}[q_\alpha(x+\Delta^i+\Delta^j) + q_\alpha(x) - q_\alpha(x+\Delta^i) - q_\alpha(x+\Delta^j)] \tag{30}$$

where the perturbation vectors are given by Eq. (25). Thus, all first and second derivative information can be computed using $\gamma(\gamma+3)/2$ perturbations. Specifically, it is necessary to evaluate the quantities

$$q(x+\Delta^i)$$

for $i = 1, \ldots, \gamma$,

$$q(x-\Delta^i)$$

for $i = 1, \ldots, \gamma$, and

$$q(x+\Delta^i+\Delta^j)$$

for $i = 1, \ldots, \gamma$, $j = 1, \ldots, \gamma$, with $j > i$. In fact, since a function evaluation computes all elements of the vector q, all second derivative matrices $\{\nabla_x^2 c_1, \ldots, \nabla_x^2 c_{m_z}, \nabla_x^2 f\}$ can be formed simultaneously. Thus the Hessian matrix can be formed directly from Eq. (14) without storing the intermediate matrices.

Since the same index sets are used for both the first and second derivative computations, the perturbation sizes δ_j must necessarily be the same. Currently the perturbation size is set to a small number, although it could be chosen to balance the truncation and roundoff error in the Jacobian matrix at the first point in the optimization iteration using a technique suggested by Hallman.[15] The perturbation size is suboptimal in three ways. First, the same value of δ_j is used for all dependent functions. Second, the value of δ_j chosen to best approximate the Jacobian is used without regard to the accuracy of the Hessian. Third, the perturbation size defined at the initial iteration is used for all subsequent iterations. Fortunately, Jacobian inaccuracies can be kept acceptably small because central difference estimates are used, and Hessian inaccuracies can be tolerated because they only affect the rate of convergence. Adjustment of the perturbation size from iteration to iteration can be incorporated if required.

V. Trajectory Optimization

A method for solving a sparse nonlinear programming problem using finite-difference gradient and Hessian information has been presented. The purpose of this section is to illustrate how the formulation of a trajectory optimization or optimal control problem yields a nonlinear programming problem. Particular attention will be given to the impact of the formulation on matrix sparsity. Although we shall concentrate on a collocation approach to the problem, it has been illustrated in Ref. 7 that all trajectory problems can exploit this structure.

Let us find the n_u-dimensional control vector $\boldsymbol{u}(t)$ to minimize the performance index

$$\phi[y(t_f),\ t_f] \tag{31}$$

evaluated at the final time t_f. The dynamics of the system are defined by the state equations

$$\dot{y} = \boldsymbol{h}[y(t),\ \boldsymbol{u}(t),t] \tag{32}$$

where the n_e dimension state vector y may have some specified initial conditions at time t_0; i.e., given

$$\bar{y}(t_0) \tag{33}$$

as well as some specified values at the final time

$$\bar{y}(t_f) \tag{34}$$

The notation $\bar{y}$ is used to indicate that not necessarily all of the initial and/or terminal values are fixed. More complex boundary conditions can be incorporated in the methodology but are omitted to simplify the present discussion.

A. Hermite Interpolation

The dynamic variables y are approximated using piecewise cubic polynomials. We begin by dividing the time interval into n_s segments

$$t_0 < t_1 < t_2 \ldots < t_f = t_{n_s}$$

Let us define the constants

$$\tau_{j+1} = \frac{t_{j+1} - t_j}{t_f - t_0} \tag{35}$$

for $j = 0,1, \ldots, (n_s - 1)$. Thus the constants τ_{j+1} just define the fraction of the total time interval allocated to each segment. Within each segment the function is approximated by a cubic polynomial. The four coefficients in the cubic polynomial are determined such that y and $\dot{y}$ at the mesh points match. The value of the cubic approximation at the midpoint of a segment is then

$$\hat{y}_j = \tfrac{1}{2}(y_j + y_{j+1}) + \tfrac{1}{8}\tau_{j+1}(t_f - t_0)(\dot{y}_j - \dot{y}_{j+1}) \tag{36}$$

for $j = 0,1, \ldots, (n_s - 1)$. We have used the notation $y_j \equiv y(t_j)$ to indicate the value at a mesh point. The expression for $\hat{y}_j$ applies to each component of the vector y. The derivative of the cubic approximation at the midpoint of the interval is

$$\dot{\hat{y}}_j = \frac{-3(y_j - y_{j+1})}{2\tau_{j+1}(t_f - t_0)} - \tfrac{1}{4}(\dot{y}_j + \dot{y}_{j+1}) \tag{37}$$

Since $\dot{y} = \boldsymbol{h}(y,\boldsymbol{u},t)$ by construction, and $\boldsymbol{h}$ is given by the right-hand side of Eq. (32), the differential equations are automatically satisfied at the mesh points. In general, they will not be satisfied at the midpoints of the segments, so let us define the defect vector

$$\zeta = \dot{\hat{y}} - \boldsymbol{h}[\hat{y},\boldsymbol{u}_m,t_m] \tag{38}$$

where t_m is the time at the midpoint of the segment, and $\boldsymbol{u}_m$ is the control vector at the midpoint. There is an n_e dimension defect vector for each of the n_s segments.

Let us now define the optimization iteration variables

$$\boldsymbol{x} = [\bar{y}_0,\ \boldsymbol{u}_0,\ \boldsymbol{u}_{1/2},\ y_1,\ \boldsymbol{u}_1,\ \boldsymbol{u}_{3/2},\ \ldots,\ \bar{y}_f,\ \boldsymbol{u}_f,\ t_f]^T \tag{39}$$

The notation $\bar{y}$ is used to denote the free state variables at the endpoints, and the controls at the segment midpoints are denoted by $\boldsymbol{u}_{1/2}, \boldsymbol{u}_{3/2}, \ldots$. Note that $\boldsymbol{x}$ has dimension

$$N = (n_e + 2n_u)(n_s + 1) - n_u - n_f + 1 \tag{40}$$

where n_f is the number of fixed end points.

The iteration variables must be chosen such that the constraint vector

$$\boldsymbol{c}(\boldsymbol{x}) = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \vdots \\ \zeta_f \end{bmatrix} = 0 \tag{41}$$

The total constraint vector is made of the defect vectors from each of the n_s segments. When the constraints $\boldsymbol{c}(\boldsymbol{x})$ are satisfied, the differential equation [Eq. (32)], is satisfied at the mesh points and the interval midpoints. Clearly the accuracy of the solution is dictated by the number and location of the mesh points.

B. Trapezoidal Integration

An alternate discretization scheme can be derived based on the trapezoidal rule for integration. The approach can be constructed by taking an Euler step from each end of the segment to the midpoint and defining the difference in the predicted values as the defect for the interval. Specifically, we obtain

$$\zeta = y_{j+1} - \tfrac{1}{2}\kappa\dot{y}_{j+1} - [y_j + \tfrac{1}{2}\kappa\dot{y}_j] \equiv \theta - \vartheta \tag{42}$$

where $\kappa = \tau_{j+1}(t_f - t_0)$ and as before $\dot{y} = \boldsymbol{h}(y,\boldsymbol{u},t)$. Notice that the control variable at the midpoint of the segment is not needed in this formula, and it can be written in terms of quantities θ and ϑ evaluated at the right and left endpoints, respectively. In this case, the optimization iteration variables are

$$\boldsymbol{x} = [\bar{y}_0,\ \boldsymbol{u}_0,\ y_1,\ \boldsymbol{u}_1,\ \ldots,\ \bar{y}_f,\ \boldsymbol{u}_f,\ t_f]^T \tag{43}$$

which has dimension

$$N = (n_e + n_u)(n_s + 1) - n_f + 1 \tag{44}$$

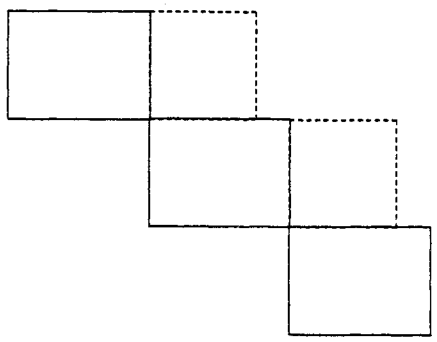

Fig. 1 Jacobian sparsity pattern.

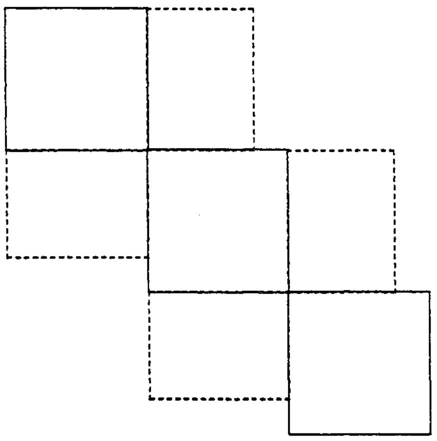

Fig. 2 Hessian sparsity pattern.

C. Sparsity Pattern

Many other discretization schemes are possible for converting the optimal control problem into a nonlinear programming problem. An exhaustive comparison of different schemes will not be presented. Instead let us illustrate how the discretization scheme interacts with the nonlinear programming algorithm. Of particular interest are differences in the sparsity pattern between the two schemes. Figure 1 illustrates the sparsity pattern for the Jacobian matrix. For both the trapezoidal and Hermite discretization schemes, $\boldsymbol{G}$ has rectangular matrices along the diagonal and above the diagonal as shown. In both schemes, the blocks have n_e rows, and for the trapezoidal scheme, all the blocks have $n_e + n_u$ columns. For the Hermite scheme, the diagonal blocks have $n_e + 2n_u$ columns, and the blocks above the diagonal (dashed lines) have $n_e + n_u$ columns. The final column for both schemes will be nonzero because the variable t_f affects all constraint functions.

Figure 2 illustrates the sparsity pattern of the Hessian $\boldsymbol{H}$ for both schemes. For the trapezoidal scheme, the Hessian is block diagonal with blocks having $n_e + n_u$ rows and columns. Because the diagonal blocks are uncoupled (the dashed blocks are zero), the Lagrangian is referred to as separable. For the Hermite scheme, the diagonal blocks have $n_e + 2n_u$ rows and columns. The rectangular blocks above the diagonal (dashed lines) have $n_e + 2n_u$ rows, and $n_e + n_u$ columns, with symmetric blocks below the diagonal.

The trapezoidal scheme leads to a separable Lagrangian because the defects are the difference of two functions that have only the final time in common. Thus second partials of the constraints on each interval involve only second partials of the variables at the left endpoint or second partials of variables at the right endpoint. In contrast, the Hermite scheme evaluates defects at the midpoint of a grid by evaluating the right-hand side of the differential equation as a linear combination of the variables at both endpoints, thereby introducing a coupling (cross partials) between the variables at both ends of the interval. The separability induced by the trapezoidal method can be exploited when computing index sets. Specifically, we define $\bar{q} = (\theta_1^T, \vartheta_1^T, \ldots, \theta_f^T, \vartheta_f^T, f)^T$ and then construct index sets for a matrix $\bar{\boldsymbol{D}}$ with rows for each of the original Jacobian rows. In doing so, the number of index sets is reduced by nearly a factor of two, and the Jacobian information can be constructed by combining the derivatives at the end of the finite-difference process.

A comparison of the different schemes reveals a number of significant points. First, the number of index sets needed for the trapezoidal scheme is

$$\gamma_t = n_e + n_u + 1 \tag{45}$$

whereas Hermite interpolation requires

$$\gamma_h = 2\gamma_t + n_u - 1 \tag{46}$$

Since $\gamma(\gamma + 3)/2$ perturbations are needed to compute Hessian and Jacobian, it is apparent that Hermite interpolation is more expensive than trapezoidal by a factor of almost two for Jacobian evaluation and nearly a factor of four for the combined evaluation of Jacobian and Hessian. Second, for a problem with the same number of mesh points, the number of optimization variables for the Hermite formulation is $n_u n_s$ larger than the number of variables for a trapezoidal formulation. Since both schemes have the same number of optimization constraints (defects), the Hermite formulation will also have a larger number of degrees of freedom. Finally, since the Hessian for the trapezoidal formulation leads to a separable Lagrangian and the Hermite formulation does not, this suggests qualitatively that the trapezoidal approach might yield a more well-behaved optimization problem. Thus, one would expect the trapezoidal scheme to be superior to Hermite interpolation, provided acceptable accuracy can be achieved by both schemes using the same number of mesh points.

A more detailed analysis of the relationship between discretization method, grid size needed for a given solution accuracy, and the speed of optimization convergence is needed before one method can be determined to be more effective than another. In the results discussed in the following section, the main focus is on the speed of convergence of the optimization algorithm on the discretized problem rather than on the accuracy of the result as compared to the solution of the original differential equation.

VI. Computational Results

Computational results on a series of trajectory optimization problems are summarized in this section. Three classes of problems were solved—a minimum time powered flight ascent trajectory, and maximum downrange and maximum crossrange hypersonic re-entry trajectories. Detailed descriptions of the problems can be found in Refs. 6 and 16. Initial guesses were constructed by linearly interpolating between initial and terminal conditions. All solutions were obtained using an Alliant FX/8 without concurrency.

A. Linear Tangent Steering

The first problem, which must be considered academic, has four differential equations, one control variable, and an analytic solution in which the optimum thrust direction is given by a linear tangent law. Table 1 summarizes the results of four different cases—all solving the same problem using the trapezoidal discretization scheme but each with a different number of mesh points.

The second line of the table presents the number of grid points, and the third and fourth lines present the corresponding number of optimization variables and constraints, respectively. The number of nonzero elements in the Jacobian matrix for both active and inactive constraints is given in the fifth line, followed by the number of nonzeros in the Hessian matrix on the sixth line. For this problem, the number of index

Table 1 Linear tangent steering

	Case no.			
	1	2	3	4
No. of grid points, $n_g (n_g = n_s + 1)$	10	20	200	2,000
No. of variables, N	44	94	994	9,994
No. of constraints, m_z	36	76	796	7,996
No. of Jacobian nonzeros, J_z	368	808	8,728	87,928
No. of Hessian nonzeros, H_z	168	368	3,968	39,968
No. of function evaluations, F_e	295	326	269	253
No. of function calls, F_c	106	110	92	76
No. of gradient calls, G_c	12	13	11	11
No. of Hessian calls, H_c	3	4	3	3
Solution time, s, T	10.54	23.79	223.09	3,232.88

Table 2 Maximum downrange—trapezoidal

	Case no.			
	5	6	7	8
n_g	51	101	201	401
N	249	499	999	1,999
m_z	200	400	800	1,600
J_z	2,172	4,372	8,772	17,572
H_z	988	1,988	3,988	7,988
F_e	911	1,932	2,216	780
F_c	398	978	1,166	321
G_c	29	47	50	27
H_c	11	26	30	9
T	181.35	698.78	1,588.56	1,290.14

sets $\gamma = 6$, and consequently a full Jacobian can be computed with 12 function evaluations, and a Jacobian and Hessian combined require 27 function evaluations.

The next five lines present the computational results for this problem. We define a function evaluation as a single evaluation of the objective function and constraint vector c. The number of function evaluations needed to solve the problem is shown on the seventh line of the table. Since the function evaluations are required both by the optimization algorithm and the sparse differencing process, the next three lines present a more detailed history of where the function evaluations were needed. In particular, function evaluations needed by the optimization algorithm itself (e.g., the univariate search) are tabulated as function calls. The number of times the Jacobian matrix was evaluated is summarized as a gradient call, and the number of Hessian matrix evaluations is presented as a Hessian call. Observe that there is hardly any change in any of these quantities from case 1 through 4. As predicted in the earlier sections, there is essentially no increase in the cost of computing gradient information because the number of index sets does not grow with the number of variables. Furthermore, the number of gradient and Hessian evaluations does not grow with the number of variables, which tends to corroborate the assertion that the optimization algorithm has second-order convergence. For example, case 4, which has 1998 degrees of freedom, could require as many as 1998 iterations if a recursive Hessian estimate had been used, in contrast to the three actually observed. Solution times (cpu seconds) are presented in the final line of the table.

B. Maximum Downrange

Unlike the linear tangent steering problem, the maximum downrange hypersonic re-entry trajectory optimization problem is extremely difficult to solve using a conventional shooting method. The solution to this class of problems is oscillatory, and numerical integration of the trajectory may require a sophisticated adaptive quadrature technique suitable for stiff systems to avoid unstable propagation of errors. This planar re-entry is described by four differential equations, with a single control variable (angle of attack) and vehicle aerodynamics representative of a simplified Space Shuttle Orbiter. Table 2 presents the results obtained using trapezoidal discretization, and Table 3 shows results using Hermite interpolation.

Table 3 Maximum downrange—Hermite

	Case no.			
	9	10	11	12
n_g	51	101	201	401
N	299	599	1,199	2,399
m_z	200	400	800	1,600
J_z	2,372	4,772	9,572	19,172
H_z	2,796	5,646	11,346	22,746
F_e	1,816	4,099	4,137	3,631
F_c	538	1,261	1,287	1,021
G_c	23	44	50	51
H_c	11	27	25	21
T	296.11	1,288.58	2,631.77	4,738.20

Table 4 Maximum crossrange—trapezoidal

	Case no.			
	13	14	15	16
n_g	51	101	201	1,430
N	350	700	1,400	10,003
m_z	250	500	1,000	7,145
J_z	4,508	9,058	18,158	129,997
H_z	1,735	3,485	6,985	50,000
F_e	486	528	752	2,875
F_c	118	112	144	331
G_c	16	19	31	152
H_c	4	4	4	4
T	138.98	310.11	941.42	34,147.17

Table 5 Maximum crossrange—Hermite

	Case no.			
	17	18	19	20
n_g	51	101	201	1,430
N	450	900	1,800	12,861
m_z	250	500	1,000	7,145
J_z	6,692	13,442	26,942	192,857
H_z	5,763	11,613	23,313	167,106
F_e	969	971	972	799
F_c	85	87	88	85
G_c	14	14	14	13
H_c	3	3	3	2
T	213.82	434.25	891.63	7,066.22

For this problem, the minimum number of index sets using the trapezoidal rule is 6, whereas Hermite interpolation requires 12. Cases 5–8 seem to corroborate the performance characteristic observed for the linear tangent steering problems—namely, increasing the problem size does not necessarily increase the number of function evaluations and iterations. Although the problems are derived by discretizing the same optimal control problem, the resulting nonlinear programming problem is different. However, it is expected that as the mesh is refined, the solution to these different optimization problems will converge to the same result. This asymptotic behavior was observed on all three examples.

The results presented in Table 3 highlight the differences between Hermite and trapezoidal discretization. Comparing cases 5–8 with cases 9–12, we note that Hermite interpolation produces a Hessian with substantially more nonzeros than the trapezoidal method. The number is different because of the off-diagonal blocks illustrated in Fig. 2. Furthermore, the number of optimization variables is somewhat larger, producing a problem with more degrees of freedom. Presumably because of the larger number of variables and the coupling between segments, the solutions of the Hermite cases were significantly more expensive to obtain than the comparable trapezoidal cases.

C. Maximum Crossrange

The maximum crossrange problem is considered the most difficult problem in the test set both because of the oscillatory solution and the size of the problem. This hypersonic re-entry

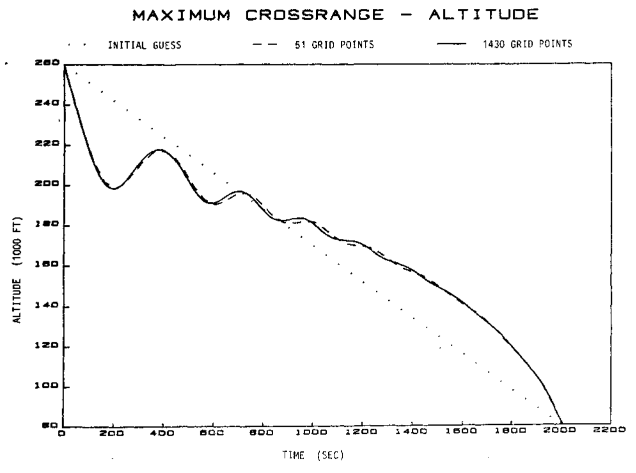

Fig. 3 Maximum crossrange—altitude.

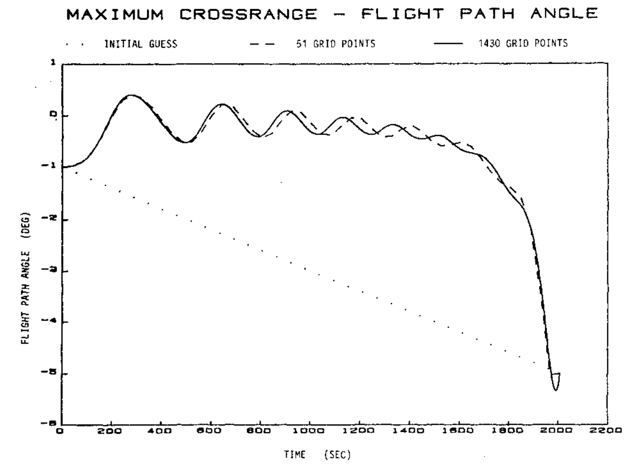

Fig. 4 Maximum crossrange—flight path angle.

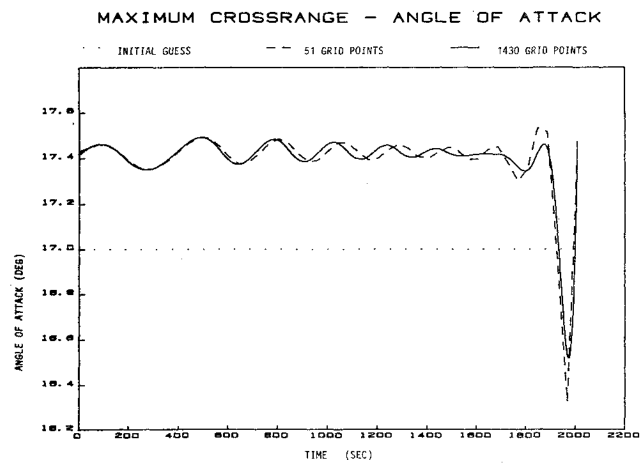

Fig. 5 Maximum crossrange—angle of attack.

is described by five differential equations, with two control variables (angle of attack and bank angle) and simplified vehicle aerodynamics representative of a Space Shuttle Orbiter. Table 4 presents the results obtained using trapezoidal discretization. For this method, eight index sets were needed, and consequently a Jacobian evaluation required 16 function evaluations, and a combined Jacobian/Hessian evaluation can be completed using 44 function evaluations. Table 5 presents similar results using Hermite interpolation. Figures 3–5 illustrate the significant solution parameters obtained for 51 and 1430 grid points as well as the linear initial guess. More extensive results are summarized in Ref. 17.

Table 6 Dense vs sparse

	Case no.			
	1	2	5	9
N	44	94	249	299
m_z	36	76	200	200
F_{SL}	1,650	2,079	4,034	1,824†
F_{SP}	295	326	911	1,816
T_{SL}	160.5	1,621.	52,681.	12,517.†
T_{SP}	10.54	23.79	181.35	296.11
R	15.23	68.14	290.49	†

Computational experience on this problem seems to confirm trends observed on the other test cases. Only the largest trapezoidal cases (15 and 16) exhibited unpredicted behavior. Although the number of Hessian evaluations remained the same for different cases, the number of Jacobian evaluations increased. This in turn caused an increase in the total number of function evaluations and the resulting solution time increased because more linear systems had to be solved. Examination of the results showed that increases occurred during the initial iterations when finding a feasible point—and could be significantly altered by scaling the problem. It also is apparent that the nonlinear programming approach to solving optimal control problems is quite robust even for very nonlinear problems, thus corroborating the experience reported in Ref. 8.

D. Comparable Dense Results

For comparison, a subset of the smaller problems were also solved using a dense optimization algorithm, specifically the successive quadratic programming algorithm NPSOL (Version 4).[18] The method is currently used in a number of trajectory programs including OTIS[8] and represents the state of the art in dense nonlinear programming. The Jacobian matrix was computed using the same sparse differencing technique; however, unlike the new sparse nonlinear programming (SPR) algorithm, the Hessian matrix was approximated recursively by NPSOL. Table 6 presents the total number of function evaluations required by NPSOL and by the SPR algorithm (F_{SL} and F_{SP}), as well as the corresponding solution times (T_{SL} and T_{SP}). The final line of the table presents the ratio of solution times ($R = T_{SL}/T_{SP}$). NPSOL failed to solve the last problem, as indicated by the symbol † in the table. For this limited test set, the number of NPSOL iterations ranged between two and five times the number of degrees of freedom, in constrast to the SPR algorithm where the number of iterations were not related to the number of degrees of freedom. Larger problems were not attempted with NPSOL because the cost appeared to be prohibitive.

VII. Summary and Conclusions

This paper presents a method for solving trajectory optimization problems using a sparse nonlinear programming algorithm. An algorithm that constructs finite-difference Jacobian and Hessian matrices using sparse differencing is given. Although quadratic convergence is not proven, computational experience with the algorithm seems to confirm the conjecture that the number of function evaluations and number of iterations needed to obtain a solution does not grow with the size of the problem. Furthermore, the results suggest that the problem sparsity and algorithm performance can be exploited by appropriately choosing the method used to discretize the original optimal control problem.

References

[1]Bauer, T., Betts, J., Hallman, W., Huffman, W., and Zondervan, K., "Solving the Optimal Control Problem Using a Nonlinear Programming Technique, Part 2: Optimal Shuttle Ascent Trajectories," AIAA Paper 84-2038, Aug. 1984.

[2]Betts, J. T., Bauer, T., Huffman, W., and Zondervan, K., "Solving the Optimal Control Problem Using a Nonlinear Programming Technique, Part 1: General Formulation," AIAA Paper 84-2037, Aug. 1984.

[3]Betts, J. T., "Optimal Three-Burn Orbit Transfer," *AIAA Journal,* Vol. 15, No. 6, 1977, pp. 861-864.

[4]Brauer, G. L., Cornick, D. E., and Stevenson, R., "Capabilities and Applications of the Program to Optimize Simulated Trajectories (POST)," NASA CR-2770, Feb. 1977.

[5]Meder, D. S., and Searcy, J. L., "Generalized Trajectory Simulation (GTS)," The Aerospace Corp., SAMSO TR-15-255, Los Angeles, CA, Vols. I-V, Jan. 1975.

[6]Betts, J. T., "Sparse Jacobian Updates in the Collocation Method for Optimal Control Problems," *Journal of Guidance, Control, and Dynamics,* Vol. 13, No. 3, 1990, pp. 409-415.

[7]Betts, J. T., and Huffman, W. P., "Trajectory Optimization on a Parallel Processor," *Journal of Guidance, Control, and Dynamics,* Vol. 14, No. 2, pp. 431-439.

[8]Hargraves, C. R., and Paris, S.W., "Direct Trajectory Optimization Using Nonlinear Programming and Collocation," *Journal of Guidance, Control, and Dynamics,* Vol. 10, No. 4, 1987, p. 388.

[9]Ashcraft, C. C., "A Vector Implementation of the Multifrontal Method for Large Sparse, Symmetric Positive Definite Linear Systems," Boeing Computer Services, Seattle, WA, ETA-TR-51, 1987.

[10]Ashcraft, C. C., and Grimes, R. G., "The Influence of Relaxed Supernode Partitions on the Multifrontal Method," Boeing Computer Services, Seattle, WA, ETA-TR-60, 1988.

[11]Betts, J. T., and Hallman, W. P., "NLP2 Optimization Algorithm Documentation," The Aerospace Corp., TOR-0089 (4464-06)-1, Los Angeles, CA, Aug. 1989.

[12]Curtis, A., Powell, M. J. D., and Reid, J. K., "On the Estimation of Sparse Jacobian Matrices," *Journal of the Institute of Mathematical Applications,* Vol. 13, 1974, pp. 117-120.

[13]Coleman, T. F., and Moré, J. J., "Estimation of Sparse Jacobian Matrices and Graph Coloring Problems," *SIAM Journal of Numerical Analysis,* Vol. 20, 1983, pp. 187-209.

[14]Powell, M. J. D., and Toint, Ph. L., "On Estimation of Sparse Hessian Matrices," *SIAM Journal of Numerical Analysis,* Vol. 16, No. 6, 1979, pp. 1060-1074.

[15]Hallman, W. P., "Numerical Derivative Techniques for Trajectory Optimization," *Proceedings of the Third Air Force/NASA Symposium on Recent Advances in Multidisciplinary Analysis and Optimization,* San Francisco, CA, Sept. 1990, pp. 418-424.

[16]Zondervan, K. P., Bauer, T., Betts, J., and Huffman, W., "Solving the Optimal Control Problem Using a Nonlinear Programming Technique, Part 3: Optimal Shuttle Reentry Trajectories," AIAA Paper 84-2039, Aug. 1984.

[17]Betts, J. T., and Huffman, W. P., "The Application of Sparse Nonlinear Programming to Trajecotry Optimization," Boeing Computer Services, ECA-TR-141, Seattle, WA, June 1990.

[18]Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H., "User's Guide for NPSOL (Version 4.0): A Fortran Package for Nonlinear Programming," Dept. of Operation Research, Stanford University, Stanford, CA, Rept. SOL 86-2.