JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 16, No. 1, January-February 1993

Path-Constrained Trajectory Optimization Using Sparse Sequential Quadratic Programming

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

One of the most effective numerical techniques for the solution of trajectory optimization and optimal control problems is the direct transcription method. This approach combines a nonlinear programming algorithm with discretization of the trajectory dynamics. The resulting mathematical programming problem is characterized by matrices that are large and sparse. Constraints on the path of the trajectory are then treated as algebraic inequalities to be satisfied by the nonlinear program. This paper describes a nonlinear programming algorithm that exploits the matrix sparsity produced by the transcription formulation. Numerical experience is reported for trajectories with both state and control variable equality and inequality path constraints.

I. Introduction

IT is well known that the solution of an optimal control or trajectory optimization problem can be posed as the solution of a two-point boundary value problem. This problem requires solving a set of nonlinear ordinary differential equations; the first set defined by the vehicle dynamics and the second set (of adjoint differential equations) by the optimality conditions. Boundary conditions are imposed from the problem requirements as well as the optimality criteria. By discretizing the dynamic variables, this boundary value problem can be reduced to the solution of a set of nonlinear algebraic equations. This approach has been successfully utilized[1-5] for applications without path constraints. Since the approach requires adjoint equations, it is subject to a number of difficulties. First, the adjoint equations are often very nonlinear and cumbersome to obtain for complex vehicle dynamics, especially when thrust and aerodynamic forces are given by tabular data. Second, the iterative procedure requires an initial guess for the adjoint variables, and this can be quite difficult because they lack a physical interpretation. Third, convergence of the iterations is often quite sensitive to the accuracy of the adjoint guess. Finally, the adjoint variables may be discontinuous when the solution enters or leaves an inequality path constraint.

Difficulties associated with adjoint equations are avoided by the direct transcription or collocation methods.[6-10] In this approach, the dynamic equations are discretized, and the optimal control problem is transformed into a nonlinear program, which can be solved directly. The nonlinear programming problem is large and sparse and a method for solving it is presented in Ref. 7. This paper extends the method of Ref. 7 to efficiently handle inequality constraints and presents a nonlinear programming algorithm designed to exploit the properties of the problem that results from direct transcription of the trajectory optimization application.

Received July 23, 1991; presented as Paper 91-2739 at the AIAA Guidance, Navigation, and Control Conference, New Orleans, LA, Aug. 12-14, 1991; revision received Dec. 10, 1991; accepted for publication Feb. 4, 1992.

*Senior Principal Scientist, Applied Mathematics and Statistics Department, Research and Technology Division, P.O. Box 24346, MS 7L-21.

†Applied Mathematician, Applied Mathematics and Statistics Department, Research and Technology Division, P.O. Box 24346, MS 7L-21.

II. Trajectory Optimization

A. Optimal Control Problem

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

$$\phi\left[y(t_f), t_f\right] \tag{1}$$

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

$$\dot{y} = h\left[y(t), u(t), t\right] \tag{2}$$

where y is the n_e-dimensional state vector. Initial conditions at time t_0 are defined by

$$\psi\left[y(t_0), u(t_0), t_0\right] \equiv \psi_0 = 0 \tag{3}$$

and terminal conditions at the final time t_f are defined by

$$\psi\left[y(t_f), u(t_f), t_f\right] \equiv \psi_f = 0 \tag{4}$$

In addition, the solution must satisfy path constraints of the form

$$\Psi_L \le \Psi\left[y(t), u(t), t\right] \le \Psi_U \tag{5}$$

where Ψ is a vector of size n_p, as well as simple bounds on the state variables

$$y_L \le y(t) \le y_U \tag{6}$$

and control variables

$$u_L \le u(t) \le u_U \tag{7}$$

Note that a path variable equality constraint can be imposed if the upper and lower bounds are equal, e.g., $(\Psi_L)_k = (\Psi_U)_k$ for some k. For simplicity in presentation, we have chosen to eliminate discussion of trajectories with more than one phase, optimization variables independent of time, and alternate forms of the objective function. However, some of the computational results do, in fact, involve each of these generalizations, and the reader is referred to Ref. 6 for details of the formulation.

B. Transcription Formulation

The basic approach for solving the optimal control problem by transcription has been presented in detail elsewhere[7-10] and will only be summarized here. All approaches divide the time interval into n_s segments

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

where the points are referred to as mesh or grid points. Let us introduce the notation $y_j \equiv y(t_j)$ to indicate the value of the state variable at a grid point. In like fashion, denote the control at a grid point by $u_j \equiv u(t_j)$. In addition, some discretization schemes require values for the control variable at the midpoint of an interval, and we denote this quantity by $\bar{u}_j \equiv u(\bar{t})$ with $\bar{t} = ½(t_j + t_{j-1})$. Numerical results will be presented for three different discretization schemes, namely, trapezoidal, Hermite-Simpson, and Runge-Kutta. Each scheme produces a distinct set of nonlinear programming (NLP) variables and constraints.

For the trapezoidal discretization, the NLP variables are

$$x = [y_0, u_0, y_1, u_1, \ldots, y_f, u_f, t_f]^{\mathsf{T}} \tag{8}$$

The state equations (2) are approximately satisfied by setting *defects*

$$\zeta_j = y_j - y_{j-1} - ½\kappa_j[h_j + h_{j-1}] \tag{9}$$

to zero for $j = 1, \ldots, n_s$. The step size is denoted by $\kappa_j \equiv t_j - t_{j-1}$, and the right-hand side of the differential equations (2) are given by $h_j \equiv h[y(t_j), u(t_j), t_j]$.

For the Hermite-Simpson discretization scheme, the NLP variables are

$$x = [y_0, u_0, \bar{u}_0, y_1, \bar{u}_1, \bar{u}_1, \ldots, y_f, u_f, t_f]^{\mathsf{T}} \tag{10}$$

The defects for this discretization are given by

$$\zeta_j = y_j - y_{j-1} - \frac{\kappa_j}{6}[h_j + 4\bar{h}_j + h_{j-1}] \tag{11}$$

where

$$\bar{h}_j = h[\bar{y}_j, \bar{u}_j, \bar{t}] \tag{12}$$

with

$$\bar{y}_j = ½[y_{j-1} + y_j] + \frac{\kappa_j}{8}[h_{j-1} - h_j] \tag{13}$$

for $j = 1, \ldots, n_s$.

For the fourth-order Runge-Kutta discretization scheme, the NLP variables are the same as those for the Hermite-Simpson method (10). The defects for this discretization are given by

$$\zeta_j = y_j - z_j^4 \tag{14}$$

where

$$z_j^1 = y_{j-1} + \frac{\kappa_j}{2} h(y_{j-1}, u_{j-1}, t_{j-1}) \tag{15}$$

$$z_j^2 = y_{j-1} + \frac{\kappa_j}{2} h(z_j^1, \bar{u}_j, \bar{t}) \tag{16}$$

$$z_j^3 = y_{j-1} + \kappa_j h(z_j^2, \bar{u}_j, \bar{t}) \tag{17}$$

$$z_j^4 = y_{j-1} + \frac{\kappa_j}{6}\Big[h(y_{j-1}, u_{j-1}, t_{j-1}) + 2h(z_j^1, \bar{u}_j, \bar{t}) + 2h(z_j^2, \bar{u}_j, \bar{t}) + h(z_j^3, u_j, t_j)\Big] \tag{18}$$

for $j = 1, \ldots, n_s$.

As a result of the transcription process, the optimal control constraints (2–5) are replaced by the NLP constraints

$$c_L \le c(x) \le c_U \tag{19}$$

where

$$c = [\zeta_1, \zeta_2, \ldots, \zeta_f, \psi_0, \psi_f, \Psi_0, \Psi_1, \ldots, \Psi_f]^{\mathsf{T}} \tag{20}$$

with

$$c_L = [0, \ldots, 0, \Psi_L, \ldots, \Psi_L]^{\mathsf{T}} \tag{21}$$

and a corresponding definition of c_U. The first $n_e n_s$ equality constraints require that the defect vectors from each of the n_s segments be zero, thereby approximately satisfying the differential equations (2). The boundary conditions are enforced directly by the equality constraints on ψ, and the nonlinear path constraints are imposed at the grid points. Note that nonlinear equality path constraints are accommodated by setting $c_L = c_U$.

In a similar fashion, the state and control variable bounds [Eqs. (6) and (7)] become simple bounds on the NLP variables. Observe that although the path constraints and variable bounds are always imposed at the grid points, for the Hermite-Simpson and Runge-Kutta discretization methods, additional bounds are imposed on the control variables at the interval midpoints.

III. Nonlinear Programming Problem

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

$$f(x) \tag{22}$$

subject to the *constraints*

$$c_L \le c(x) \le c_U \tag{23}$$

where $c(x)$ is an m vector of constraint functions, and the simple *bounds*

$$x_L \le x \le x_U \tag{24}$$

Equality constraints are imposed by setting $c_L = c_U$ and variables can be fixed by setting $x_L = x_U$.

The solution point x^* must satisfy the following Kuhn-Tucker necessary conditions for a local minimum:

1) The x^* is feasible, i.e., Eqs. (23) and (24) are satisfied.

2) There exist Lagrange multipliers λ and ν such that

$$g = G^{\mathsf{T}}\lambda + \nu \tag{25}$$

where $\nabla_x f(x) = g(x) = g$ is the N-dimensional gradient vector, and G is the $m \times N$ Jacobian matrix of constraint gradients.

3) The Lagrange multiplier for a constraint or variable active at its lower bound must be nonnegative.

4) The Lagrange multiplier for a constraint or variable active at its upper bound must be nonpositive.

5) The Lagrange multiplier for a strictly feasible constraint or free variable must be zero.

IV. Sparse Nonlinear Programming Algorithm

The solution of a nonlinear program can be accomplished in a wide variety of ways. In general, the design of an algorithm strategy entails a number of somewhat imprecise considerations with regard to speed, robustness, and general utility. The basic approach utilized by the algorithm is to solve a sequence of quadratic programming subproblems. The fundamental premise of the approach is to approximate the nonlinear constraint functions by a linear model and the general objective function by a quadratic model. First, background on the quadratic programming subproblem and the associated definition of a merit function are presented. Then a description of three distinct optimization strategies will be given.

A. Quadratic Programming Subproblem

A primary feature of the nonlinear programming algorithm to be described is the ability to solve a quadratic programming (QP) subproblem. Solution of the QP subproblem is used to define new estimates for the variables according to the formula

$$\bar{x} = x + \alpha p \tag{26}$$

where the vector p is referred to as the *search direction*. The scalar α determines the step length and is typically set to 1. The search direction p is found by minimizing the quadratic

$$g^{\mathrm{T}} p + \tfrac{1}{2} p^{\mathrm{T}} H p \tag{27}$$

subject to the linear constraints

$$b_\ell \le \begin{bmatrix} Gp \\ p \end{bmatrix} \le b_u \tag{28}$$

where H is a symmetric $N \times N$ positive definite approximation to the Hessian matrix. The upper-bound vector is defined by

$$b_u = \begin{bmatrix} c_U - c \\ x_U - x \end{bmatrix} \tag{29}$$

with a similar definition for the lower-bound vector b_ℓ. The technique for solving this quadratic program when the relevant matrices are large and sparse will be described in the next section.

B. Merit Function

When a quadratic program is used to approximate a general nonlinearly constrained problem, it may be necessary to adjust the steplength α to achieve "sufficient reduction" in a merit function that in some way combines the objective function and constraint violations.

The merit function we use is similar to that proposed by Gill et al.[11] and is related to the function given by Rockafeller[12]

$$M(x,\lambda,\nu,s,t) = f - \lambda^{\mathrm{T}}(c-s) - \nu^{\mathrm{T}}(x-t)$$
$$+ \tfrac{1}{2}(c-s)^{\mathrm{T}} Q (c-s) + \tfrac{1}{2}(x-t)^{\mathrm{T}} R (x-t) \tag{30}$$

The diagonal penalty matrices are defined by $Q_{ii} = \rho_i$ and $R_{ii} = \gamma_i$. For this merit function, the slack variables s at the beginning of a step are defined by

$$s_i = \max\left[c_{Li},\ \ \min\left(c_i - \frac{\lambda_i}{\rho_i},\ \ c_{Ui} \right) \right] \tag{31}$$

Similarly, the bound slacks at the beginning of the step are given by

$$t_i = \max\left[x_{Li},\ \ \min\left(x_i - \frac{\nu_i}{\gamma_i},\ \ x_{Ui} \right) \right] \tag{32}$$

The search direction in the real variables x as given by Eq. (26) is augmented to permit the multipliers and slack variables to vary according to

$$\begin{bmatrix} \bar{x} \\ \bar{\lambda} \\ \bar{\nu} \\ \bar{s} \\ \bar{t} \end{bmatrix} = \begin{bmatrix} x \\ \lambda \\ \nu \\ s \\ t \end{bmatrix} + \alpha \begin{bmatrix} p \\ \xi \\ \eta \\ q \\ \delta \end{bmatrix} \tag{33}$$

The multiplier search directions ξ and η are defined using the QP multipliers μ and ω according to

$$\xi \equiv \mu - \lambda \tag{34}$$

and

$$\eta \equiv \omega - \nu \tag{35}$$

From the QP Eqs. (27–29), the predicted slack variables are just

$$\bar{s} = Gp + c = s + q \tag{36}$$

Using this expression, define the slack vector step by

$$q = Gp + (c - s) \tag{37}$$

A similar technique defines the bound slack vector search direction

$$\delta = p + (x - t) \tag{38}$$

Note that when a full step is taken $\alpha = 1$, the updated estimate for the Lagrange multipliers $\bar{\lambda}$ and $\bar{\nu}$ are just the QP estimates μ and ω. The slack variables s and t are just the linear estimates of the constraints and the terms $(c-s)$ and $(x-t)$ in the merit function are measures of the *deviation* from linearity.

C. Parameter Definitions

In Ref. 11 it is shown that the penalty weights ρ_i and γ_i are finite, provided the Hessian matrix H used in the QP subproblem is positive definite. However, in general, the Hessian of the Lagrangian

$$H_L = \nabla_x^2 f - \sum_{i=1}^{m} \lambda_i \nabla_x^2 c_i \tag{39}$$

is not positive definite. In fact, it is only necessary that the projected Hessian be positive semidefinite at the solution with the correct active set of constraints.[13] Consequently, we use the modified matrix

$$H = H_L + \tau(|\sigma| + 1) I \tag{40}$$

The Levenberg parameter τ is chosen such that $0 \le \tau \le 1$ and is normalized using the Gerschgorin bound for the most negative eigenvalue of H_L, i.e.,

$$\sigma = \min_{1 \le i \le N} \left\{ h_{ii} - \sum_{i \ne j}^{N} |h_{ij}| \right\} \tag{41}$$

and h_{ij} is used to denote the nonzero elements of H_L.

The proper choice for the Levenberg parameter τ can greatly affect the performance of the nonlinear programming algorithm. Quadratic convergence can only be obtained when $\tau = 0$ and the correct active set has been identified. On the other hand, if $\tau = 1$ in order to guarantee a positive definite Hessian, the search direction p is significantly biased toward a gradient direction and convergence is degraded. The strategy employed to adjust τ at a particular step in the nonlinear programming iteration is as follows:

1) *Inertia test.* If the inertia of the Kuhn-Tucker (KT) matrix K [see Eq. (60)] is correct, go to step 2; otherwise, increase τ and repeat step 1.

2) *Trust region strategy.*

a) Compute the ratio of actual reduction to predicted reduction

$$\varrho_1 = \frac{M^{(k)} - M^{(k-1)}}{\bar{M}^{(k)} - M^{(k-1)}} \tag{42}$$

b) Compute the rate of change in the projected gradient norm

$$\varrho_2 = \frac{\| \vartheta^{(k)} \|}{\| \vartheta^{(k-1)} \|} \tag{43}$$

where

$$\vartheta = g - G^{\mathsf{T}}\lambda - \nu \tag{44}$$

c) If $\varrho_1 \leq 0.25$, then reduce the trust radius, i.e., set $\tau^{(k+1)} = \min[2\tau^{(k)}, 1]$; otherwise,

d) If $\varrho_1 \geq 0.75$, then increase the trust radius, i.e., set $\tau^{(k+1)} = \tau^{(k)} \min(0.5, \varrho_2)$.

The inertia (i.e., the number of positive, negative, and zero eigenvalues) of the related KT matrix [Eq. (60)] described in the next section is used to infer that the projected Hessian is positive definite. Basically, the philosophy is to reduce the Levenberg parameter when the predicted reduction in the merit function agrees with the actual reduction and increase it when the agreement is poor. The process is accelerated by making the change in τ proportional to ϱ_2.

Although the Levenberg parameter is used to ensure that the projected Hessian approximation is positive definite, it is still necessary to define the penalty weights $\boldsymbol{Q}$ and $\boldsymbol{R}$. In Ref. 11 it is shown that convergence of the method requires choosing the weights such that

$$M_0' \leq -\tfrac{1}{2}\boldsymbol{p}^{\mathsf{T}}\boldsymbol{H}\boldsymbol{p} \tag{45}$$

where M_0' denotes the direction derivative of the merit function [Eq. (30)] with respect to the step length α evaluated at $\alpha = 0$. To achieve this, let us write the penalty parameters as

$$r_i = \begin{cases} \rho_i - \rho_0 & \text{if} \quad 1 \leq i \leq m \\ \gamma_{i-m} - \rho_0 & \text{if} \quad m < i \leq (m+N) \end{cases} \tag{46}$$

where ρ_0 is a strictly positive "threshold." Let us choose the penalty parameters r_i as the *minimum norm* solution to Eq. (45). After some lengthy algebra we find that

$$\boldsymbol{r} = \boldsymbol{a}(\boldsymbol{a}^{\mathsf{T}}\boldsymbol{a})^{-1}\zeta \tag{47}$$

where

$$a_i = \begin{cases} (c_i - s_i)^2 & \text{if} \quad 1 \leq i \leq m \\ (x_{i-m} - t_{i-m})^2 & \text{if} \quad m < i \leq (m+N) \end{cases} \tag{48}$$

and

$$\zeta = -\tfrac{1}{2}\boldsymbol{p}^{\mathsf{T}}\boldsymbol{H}\boldsymbol{p} + \boldsymbol{\mu}^{\mathsf{T}}\boldsymbol{q} + \boldsymbol{\omega}^{\mathsf{T}}\boldsymbol{\delta} - 2\boldsymbol{\xi}^{\mathsf{T}}(\boldsymbol{c}-\boldsymbol{s}) - 2\boldsymbol{\eta}^{\mathsf{T}}(\boldsymbol{x}-\boldsymbol{t})$$
$$- \rho_0(\boldsymbol{c}-\boldsymbol{s})^{\mathsf{T}}(\boldsymbol{c}-\boldsymbol{s}) - \rho_0(\boldsymbol{x}-\boldsymbol{t})^{\mathsf{T}}(\boldsymbol{x}-\boldsymbol{t}) \tag{49}$$

Typically, the threshold parameter ρ_0 is set at machine precision and only increased if the minimum norm solution is zero. In essence then, the penalty weights are chosen to be as small as possible, consistent with the descent condition [Eq. (45)].

D. Algorithm Strategy

The algorithm described in Ref. 7 is a "feasible region" method, since successive iterates maintain constraint feasibility. This philosophy is motivated by a number of considerations. The trajectory optimization applications of interest are characterized by a relatively large number of equality constraints (derived from the dynamics). When the constraints are satisfied, the variables describe a real (i.e., "flyable") trajectory for a fixed discretization history. Grid refinement to improve the accuracy of the discretization is meaningful when performed about a "real" trajectory and may not be elsewhere. Second, 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.

In contrast, the algorithm described in Ref. 14 does not produce a feasible point until the solution is obtained. For a well-posed problem, a strategy that takes direct steps toward the solution without recourse to intermediate constraint satisfaction may be more efficient. Furthermore, maintaining strict feasibility with respect to inequality constraints may be prohibitively slow when the number of inequalities is large and the active set is wrong. Correct identification of the active set by a quadratic programming subproblem may be the only effective way to deal with the combinatorial nature of the problem. However, for very nonlinear constraints it may be necessary to use very large penalty weights to achieve feasibility, thus introducing the possibilty of numerical instability.

To explore the conflicting benefits of these alternate strategies, three different approaches will be investigated:

1) "m" (minimize): Beginning at $\boldsymbol{x}^0$, solve a sequence of quadratic programs until the solution $\boldsymbol{x}^*$ is found.

2) "fm" (feasible point, then minimize): Beginning at $\boldsymbol{x}^0$, solve a sequence of quadratic programs to locate a feasible point $\boldsymbol{x}^f$, and then beginning from $\boldsymbol{x}^f$, solve a sequence of quadratic programs until the solution $\boldsymbol{x}^*$ is found.

3) "fme" (feasible point, then minimize subject to equalities): Beginning at $\boldsymbol{x}^0$, solve a sequence of quadratic programs to locate a feasible point $\boldsymbol{x}^f$, and then beginning from $\boldsymbol{x}^f$, solve a sequence of quadratic programs while maintaining feasible equalities until the solution $\boldsymbol{x}^*$ is found.

Philosophically, the first strategy is probably the most aggressive, whereas the last strategy is probably the most conservative.

E. Finding a Feasible Point

The first step in either the "fm" or "fme" strategy is to determine a point that is feasible with respect to the constraints. A fourth strategy "f," to just locate a feasible point, is also available in the software. The approach employed is to take a series of steps of the form given by Eq. (26), with the search direction computed to solve a *least distance program*. This can be accomplished if we impose the requirement that the search direction has a minimum norm, i.e., $\|\boldsymbol{p}\|$. The minimum norm solution can be obtained from Eq. (27) by setting $\boldsymbol{H} = \boldsymbol{I}$ and $\boldsymbol{g} = \boldsymbol{0}$.

Since the solution of this subproblem is based on a linear model of the constraint functions, it may be necessary to adjust the step length α in Eq. (26) to produce a reduction in the constraint error. Specifically, a quadratic line search is used to adjust α so that $\Upsilon(\bar{\boldsymbol{x}}) \leq \Upsilon(\boldsymbol{x})$, where the constraint violation is defined by

$$\Upsilon(\boldsymbol{x}) = \sum_{i=1}^{m} \left[2c_i - \min(c_{Ui}, c_i) - \max(c_{Li}, c_i)\right]^2$$
$$+ \sum_{i=1}^{N} \left[2x_i - \min(x_{Ui}, x_i) - \max(x_{Li}, x_i)\right]^2 \tag{50}$$

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.

F. Minimization Process

All three strategies—"m," "fm," and "fme"—execute a series of steps to minimize the merit function [Eq. (30)]. In the case of strategy "m," the iteration begins from the arbitrary and possibly infeasible point $\boldsymbol{x}^0$. On the other hand, strategy "fm" begins the minimization of the merit function from a feasible point $\boldsymbol{x}^f$. Finally, the "fme" strategy not only begins the minimization at a feasible point, but also *maintains* feasibility with respect to the equality constraints. Let us denote the equalities by $\boldsymbol{e}$, with Jacobian $\boldsymbol{E}$.

The iteration begins at the point $\boldsymbol{x}$ and proceeds as follows:

1) Evaluate gradient information $\boldsymbol{g}$ and $\boldsymbol{G}$.
   a) Then terminate if the Kuhn-Tucker conditions are satisfied.
   b) Otherwise, define τ from Eqs. (42–44), computing $\boldsymbol{H}_L$ from Eq. (39).

2) Construct the optimization search direction.
   a) Compute H from Eq. (40).
   b) Compute p by solving the QP subproblem (27) and (28).
   c) If inertia of H is incorrect, increase τ and return to step a.
   d) Compute ξ and η from Eqs. (34) and (35).
   e) Compute q and δ from Eqs. (37) and (38).
   f) Compute penalty parameters to satisfy Eq. (45), using Eqs. (46–49).
   g) And initialize $p^{(2)}=p^{(3)}=0$, $\alpha=1$, $\tilde{\alpha}=0$.
3) Compute the predicted point for
   a) Variables from

$$\bar{x} = x + \alpha p + \alpha^2 p^{(2)} + \alpha^3 p^{(3)} \tag{51}$$

   b) Multipliers and slacks from Eq. (33).
   c) Then evaluate the constraints $\bar{c}=c(\bar{x})$ at the predicted point.
   d) Then if $\|\bar{e}\| \le \epsilon$ or not, strategy "fme," set $\hat{x} \Leftarrow \bar{x}$ and go to step 7.
4) Solve the underdetermined system

$$Ed = \bar{e} \tag{52}$$

for the direction d and initialize $v=1$.
5) Compute the corrected point

$$\hat{x} = \bar{x} - vd \tag{53}$$

6) Evaluate the constraints $\hat{e}$ at the corrected point $\hat{x}$.
   a) Then if $\|\hat{e}\| \le \epsilon$ and $\tilde{\alpha}=0$, compute

$$p^{(2)} = 1/\alpha^2[\hat{x} - x - \alpha p] \tag{54}$$

   Save the corrected point (set $\tilde{x} \Leftarrow \hat{x}$ and $\tilde{\alpha} \Leftarrow \alpha$), and then go to step 7.
   b) Else if $\|\hat{e}\| \le \epsilon$ and $\tilde{\alpha} \ne 0$, compute the elements of $p_i^{(2)}$ and $p_i^{(3)}$ for $i=1,\ldots,N$ from the system

$$\begin{bmatrix} \alpha^2 & \alpha^3 \\ \tilde{\alpha}^2 & \tilde{\alpha}^3 \end{bmatrix} \begin{bmatrix} p_i^{(2)} \\ p_i^{(3)} \end{bmatrix} = \begin{bmatrix} \hat{x}_i - x_i - \alpha p_i \\ \tilde{x}_i - x_i - \tilde{\alpha} p_i \end{bmatrix} \tag{55}$$

   Save the corrected point (set $\tilde{x} \Leftarrow \hat{x}$ and $\tilde{\alpha} \Leftarrow \alpha$) and then go to step 7.
   c) Else if $\|\hat{e}\| \le \|\bar{e}\|$, update the corrected point (set $\bar{x} \Leftarrow \hat{x}$ and $\bar{e} \Leftarrow \hat{e}$) and return to step 4.
   d) Else reduce the step length v to achieve constraint reduction and return to step 5.
7) Evaluate the merit function $M(\hat{x},\bar{\lambda},\bar{v},\bar{s},\bar{t})=\hat{M}$.
   a) If the merit function $\hat{M}$ is "sufficiently" less than M, then $\hat{x}$ is an improved point. Update all quantities and return to step 1.
   b) Else change the step length α to minimize M and return to step 3.

The steps outlined describe the fundamental elements of the optimization process; however, a number of points deserve additional clarification. Note that the algorithm consists of an "outer" loop (steps 1–7) to minimize the merit function and an "inner" loop (steps 3–6) to eliminate the error in the binding constraints. The outer loop can be viewed as a univariate (line) search in the direction p, with the step length α adjusted to minimize the merit function. The inner loop can be viewed as a nonlinear root-solving process designed to eliminate the error in the equality constraints for the specified value of α. Note that steps 4–6 are only executed for the "fme" strategy. The inner constraint elimination process must be initiated with an estimate of the variables, and this estimate is given by the expression (51). Notice that the first prediction is based on a linear model for the constraints since $p^{(2)}=p^{(3)}=0$ in step 2. However, after the constraint error has been eliminated, the value of $p^{(2)}$ is updated by Eq. (54) and the second prediction is based on a quadratic model of the constraints. After the second corrected point is obtained, subsequent predictions utilize the cubic model defined by solving Eq. (55). Adjusting the value of the step length α as required in step 7b is accomplished using a univariate search procedure described in Ref. 15 that constructs a quadratic model of the merit function.

Because the constraint elimination process requires the solution of the underdetermined system of Eq. (52), 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

$$\begin{bmatrix} I & E^{\mathrm{T}} \\ E & 0 \end{bmatrix} \begin{bmatrix} d \\ -\underline{\omega} \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{e} \end{bmatrix} \tag{56}$$

Notice that the Jacobian E is evaluated at the reference point x and not re-evaluated during the inner loop iteration, even though the right-hand side $\bar{e}$ 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 of the following:

1) The coefficient matrix can be factored only once per outer-optimization iteration, thereby significantly reducing the linear algebra expense.

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.

To evaluate the Hessian matrix of Eq. (39), an estimate of the Lagrange multipliers is needed. The values obtained by solving the QP with $H=I$ are used for the first iteration, and thereafter the values $\bar{\lambda}$ from Eq. (33) are used. Furthermore, for the very first iteration the multiplier search direction $\xi=0$ so that the multipliers will be initialized to the QP estimates μ. The Levenberg parameter τ in Eq. (40) and penalty weights r_i in Eq. (46) are initialized to zero, and consequently, the merit function is initially just the Lagrangian.

Gradient and Hessian information can either be computed analytically or using finite difference estimates. All numerical results construct this information using sparse finite differencing, as described in Refs. 6, 7, and 16. Although central difference estimates must be used during optimization, forward-difference estimates are used when finding a feasible point. Numerical experience also suggests that it is not necessary to compute a Hessian matrix for every iteration (step 1b) when the algorithm is progressing well. Instead, we simply use the current (presumably best) estimate, until it is necessary to recompute the estimate to maintain good progress.

V. Sparse Quadratic 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 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 $A=LDL^{\mathrm{T}}$, 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. 17 and 18.

B. Schur Complement Method

The quadratic programming algorithm used is based on a method proposed by Gill et al.[19] The implementation is described in Ref. 20 and exploits the multifrontal algorithm for the solution of sparse linear systems. The QP Eqs. (27–29) is first stated in the following *standard form*. Minimize

$$c^{\mathrm{T}}x + \tfrac{1}{2}x^{\mathrm{T}}Hx \tag{57}$$

subject to the linear constraints

$$Ax = b \tag{58}$$

and simple bounds

$$x_L \leq x \leq x_U \tag{59}$$

Note that the variables x include slack variables to replace the general inequality constraints and do not correspond to the notation used elsewhere in the paper. Similar modifications are necessary to define the other quantities in the standard form QP. When written in this form, variables are either fixed at their bounds or free to move within the bounds. In keeping with the notation of Ref. 19, we denote the n_{FR} "free" quantities by FR and the fixed by FX. For a specific estimate of the active set of constraints, the step to the constrained minimum is obtained by solving the Kuhn-Tucker or KT system[7]

$$\begin{bmatrix} H_{FR} & A_{FR}^{\mathsf{T}} \\ A_{FR} & 0 \end{bmatrix} \begin{bmatrix} -p_{FR} \\ \pi \end{bmatrix} \equiv K_0 \begin{bmatrix} -p_{FR} \\ \pi \end{bmatrix} = \begin{bmatrix} g_{FR} \\ 0 \end{bmatrix} \tag{60}$$

In Ref. 19 it is shown that the inertia of K is $(N_{FR}, m, 0)$ when the projected Hessian is positive definite and the constraints are full rank. If the estimate of the active set is correct, the solution of the KT system defines the solution of the QP. However, in general, it will be necessary to change the active set and solve a series of equality-constrained problems. In Ref. 19 it is demonstrated that the solution to a problem with a new active set can be obtained by the symmetric addition of a row and column to the original K_0, with a corresponding augmentation of the right-hand side. In fact, after k iterations the KT system is of dimension $n_0 + k$ and has the form

$$\begin{bmatrix} K_0 & U \\ U^{\mathsf{T}} & V \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} f_0 \\ w \end{bmatrix} \tag{61}$$

where U is $n_0 \times k$ and V is $k \times k$. The initial right-hand side of Eq. (60) is denoted by the n_0 vector f_0, and the k vector w defines the additions to the right-hand side to reflect changes in the active set.

The fundamental feature of the method is that the system of Eq. (61) can be solved using factorizations of K_0 and C, the $k \times k$ *Schur complement* of K_0

$$C \equiv V - U^{\mathsf{T}} K_0^{-1} U \tag{62}$$

Using the Schur complement, we can compute the values for y and z by solving, in turn,

$$K_0 v_0 = f_0 \tag{63}$$

$$Cz = w - U^{\mathsf{T}} v_0 \tag{64}$$

$$K_0 y = f - Uz \tag{65}$$

Thus, each iteration of the QP requires one solve with the factorization of K_0 and one solve with the dense symmetric indefinite factorization of C. The solve for v_0 only needs to be done once at the first iteration. Each change in the active set adds a new row and column to C, and it is relatively straightforward to update both C and its factorization to accommodate the change. Corresponding changes to the vectors f and w are also made. It is important to keep C small enough to maintain a stable, dense factorization, and this is achieved by refactoring the entire KT matrix whenever $k > 100$. In general, the penalty for refactoring may be substantial. However, when the QP algorithm is used within the general NLP algorithm, it is possible to exploit previous estimates of the active set to give the QP a "warm start." In fact, as the NLP algorithm approaches a solution, it is expected that the active set will be correctly identified, and the resulting number of iterations k for the QP subproblem will become small.

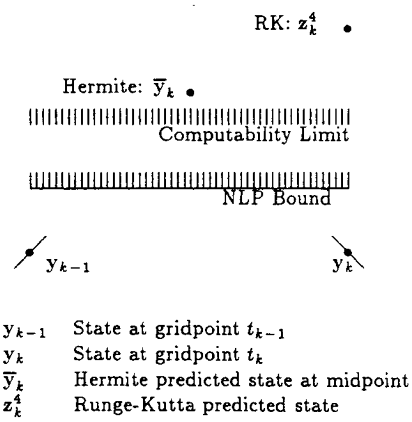

Fig. 1 Transcription robustness: y_{k-1} = state at grid point t_{k-1}; y_k = state at grid point t_k; $\bar{y}_k$ = Hermite predicted state at midpoint; and z_k^4 = Runge-Kutta predicted state.

Table 1 Transcription comparison

	Trapezoidal	Hermite	RK-4
Γ	$n_d + 1$	$2n_d + 1$	$n_d + 1$
J_z	$n_e n_d n_s$	$2n_e n_d n_s$	$n_e n_d n_s$
H_z	$n_g n_d^2/2$	$2n_g n_d^2$	$n_g n_d^2/2$
Error	$O(\kappa^2)$	$O(\kappa^4)$	$O(\kappa^4)$
RHS evaluation	n_g	$2n_g - 1$	$4n_g - 4$
Robust	Very	Less	Less

Γ = no. index sets; n_g = no. grid points ($n_g = n_s + 1$); n_d = no. dynamic variables ($n_d = n_e + n_u$); J_z = no. Jacobian nonzeros; and H_z = no. Hessian nonzeros.

VI. Grid Refinement

Section II described how an optimal control problem is transcribed into a large sparse nonlinear programming problem. Sections III–V described how the sparse NLP is solved. The accuracy of the direct transcription process is related to the number and location of the grid points, as well as the order of the discretization formula. The purpose of *grid refinement* is to determine 1) the number of grid points, 2) the location of the grid points, and 3) the discretization formula necessary to achieve a specified accuracy in the solution of the system of differential equations (2) by solving a *sequence* of nonlinear programming problems. This section presents a method for achieving these goals, which addresses the interaction between the grid refinement and nonlinear programming methods.

Table 1 summarizes six different attributes of the trapezoidal, Hermite-Simpson, and Runge-Kutta discretizations. The first three rows present attributes related to the sparsity pattern of the various schemes. When sparse finite difference derivatives are used to construct the gradient and Hessian information, the cost of this process is proportional to the number of index sets, and the Hermite discretization is nearly twice as expensive in this measure. Hermite discretization also produces a denser Jacobian and Hessian matrix, as indicated by the number of nonzero elements shown in the next two rows. The fourth and fifth rows present attributes related to the accuracy of the methods. The discretization errors for the different schemes are proportional to the quantities shown in the fourth row of the table. Another measure of cost is the number of times the right-hand sides of the differential equation are evaluated, with trapezoidal requiring the least and Runge-Kutta the most. The final attribute labeled "robust" refers to the observed behavior of the NLP when using the discretization method at a very poor initial guess. Both Hermite-Simpson and Runge-Kutta discretizations require nonlinear predictions of the state variables (either the midpoint or endpoint) of an interval. Since the predicted values are *not* NLP variables, they cannot be limited to reasonable values by

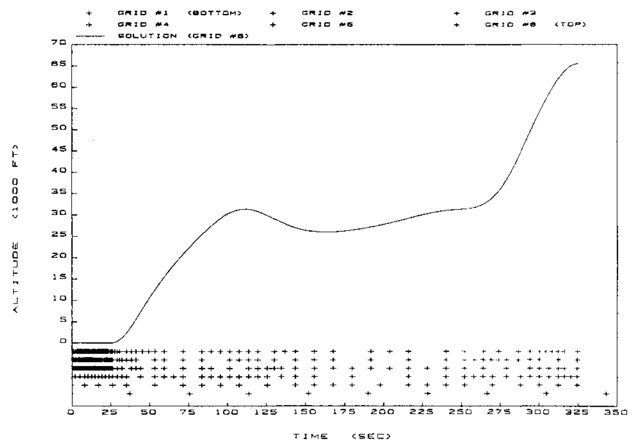

Fig. 2 Minimum time to climb: altitude vs time.

NLP-bound constraints. Consequently, it is possible that the differential equations cannot be evaluated at the predicted point, even though they can be evaluated at the grid points. A typical situation is illustrated in Fig. 1.

The basic goal of the grid refinement strategy is to begin with a relatively crude grid, i.e., a small number of points, and low-order method. After obtaining a solution for the coarse grid, the accuracy is refined by redistributing (both adding and deleting) grid points, as well as altering the order of the method. To quantify the refinement procedure, it is necessary to have a measure of the discretization error. Specifically, consider the interval $t_k \le t \le t_{k+1}$. From the known values of the dynamic variables at the ends of the interval, y_k and y_{k+1}, and their derivatives h_k and h_{k+1}, form the unique cubic interpolating polynomial. Evaluate this polynomial at three interior points $t_{k+1/4}$, $t_{k+1/2}$, and $t_{k+3/4}$ to obtain $y_{k+1/4}$, $y_{k+1/2}$, and $y_{k+3/4}$. Next evaluate the right-hand side of the differential equation at these points to give $h_{k+1/4}$, $h_{k+1/2}$, and $h_{k+3/4}$. Then from Simpson's rule, we obtain the expression

$$\hat{y}_{k+1} - y_k = \int_{t_k}^{t_{k+1}} h(t)\,\mathrm{d}t \approx \frac{\kappa_{k+1}}{3}\big(h_k + 4h_{k+1/4} + 2h_{k+1/2} + 4h_{k+3/4} + h_{k+1}\big) \tag{66}$$

where $\hat{y}_{k+1}$ is the *predicted* value of the dynamic variable at t_{k+1}, and the arguments h are just the right-hand sides of the state equations (2) evaluated at the points in the interval. Note that this formula must be applied to each component in the state vector. Now define the relative error for interval k by

$$\epsilon_k = \max_{i=1,\ldots,n_e} \frac{|\hat{y}^i_{k+1} - y^i_{k+1}|}{|y^i_{k+1}| + 1} \tag{67}$$

where the superscript refers to component i of the state variable.

From the relative error estimates, we first determine the *number* of grid points using the following criteria:

1) Equidistribute the error. Form the ratio $\bar{r}_k$ of the relative error ϵ_k to the mean of the relative error over all intervals. If $\bar{r}_k > 4$, then add an extra grid point to the interval. If $\bar{r}_k < 1/10$, then remove a grid point (right endpoint).

2) Reduce the relative error magnitude. Form the ratio r_k of the relative error ϵ_k to the targeted error level, which is usually a tenth of the user input relative error tolerance. If $r_k > 4$, add two new grid points to the interval. If $1/2 \le r_k \le 4$, add one grid point to the interval.

After the total number of new points has been defined by the refinement criteria, they are distributed uniformly over the interval.

In addition to adjusting the number and location of grid points, limited numerical experience suggests the following simple procedure for selecting the discretization method:

1) Use trapezoidal discretization for the first refinement iteration.

2) Use Hermite-Simpson discretization for subsequent refinement iterations.

Trapezoidal discretization is preferred for the first iteration because of its previously described robustness. Thereafter, the Hermite-Simpson method is desired because it is a fourth-order technique and quickly achieves the requested accuracy with a limited number of refinement iterations (usually, one or two). Although the Runge-Kutta scheme also is a fourth-order method, it was generally less efficient because of the additional right-hand-side evaluations needed in each interval. Although computational experience with this simple strategy has been quite successful, incorporation of more sophisticated refinement strategies and/or extrapolation techniques[21] may provide additional benefits. In particular, the current strategy does not explicitly control the discretization error for the path constraints, and techniques used for solving differential algebraic equations may be appropriate.

VII. Computational Results

Computational results on a series of trajectory optimization problems are summarized in this section. The test set consists of 74 problems with no grid refinement and 10 problems with refinement. Space does not permit a complete presentation of the results for all problems. Instead, we will first describe the problem set and present a summary of significant algorithm performance parameters. Second, the nonlinear programming algorithm performance will be presented for a maximum crossrange re-entry trajectory problem with heating constraint. Third, the grid refinement performance will be presented for a minimum time-to-climb problem. Finally, the method will be compared with results generated by the OTIS software.[10] Initial guesses were constructed by linearly interpolating between initial and terminal conditions. All solutions were obtained using a Sun Sparcstation IPC.

A. Trajectory Test Set

The trajectory test set consists of the following problems:

1) Quadratic-linear[22]
   a) Minimum terminal error with linear differential equations
   b) Minimum terminal error with quadratic differential equations
2) Linear tangent steering[2,22]
   a) Minimum time: adjoint variable formulation
   b) Minimum time-powered flight
3) Spherical, nonrotating Earth trajectories
   a) Minimum fuel two-burn orbit transfer
   b) Maximum downrange Shuttle re-entry[2,23]
   c) Maximum crossrange Shuttle re-entry[2,23]
   d) Maximum crossrange Shuttle re-entry with bank-angle limits
   e) Maximum crossrange Shuttle re-entry with aerodynamic heating limits[23]
4) Goddard rocket problem
   a) Maximum final velocity: vertical flight
   b) Maximum final altitude: singular arc[22,24]
5) NLQR guidance problem[25]
   a) Minimum deviation: simple aerodynamics, straight-line path
   b) Minimum deviation: complex aerodynamics, atmosphere, and path
6) Minimum time to climb[24,26]
   a) Three-degree-of-freedom formulation
   b) Planar formulation

A set of 74 different NLP problems was constructed by solving each of the problem classes using the three different discretization methods and various numbers of grid points. A summary of the results for the test problem set is given in Ref. 27. All 74 problems were run using the three optimization strategies. The algorithm performance was measured in terms

Table 2 Grid refinement test set summary

P	I_r	n_g	N	FE	T
1b	2	25	322	2920	105.3
2b	2	28	168	387	10.6
3a	2	60	448	2811	193.8
3b	2	147	882	897	118.9
3c	2	102	917	791	109.0
3d	2	101	908	1700	262.1
3e	2	78	701	2039	307.8
4a	2	25	125	260	12.7
6a	6	99	891	3093	451.2
6b	6	99	693	2092	282.9

P = problem no. and class; I_r = no. of refinement iterations; n_g = no. grid points ($n_g = n_s + 1$); N = no. of NLP variables; FE = no. function evaluations; and T = solution time, s.

Table 3 Maximum crossrange iteration history

i	Υ	$\lVert \vartheta \rVert$	τ	$\lVert r \rVert$
	Refinement Iteration 1: Trapezoidal; "fm"			
	Find Feasible Point			
1	0.63	—	—	—
2	0.58	—	—	—
3	0.15	—	—	—
4	0.57E-1	—	—	—
5	0.10E-1	—	—	—
6	0.32E-3	—	—	—
7	0.31E-6	—	—	—
8	0.23E-10	—	—	—
	Minimize Merit Function			
1	0.44E-10	0.20E-1	0.14E-1	0.14E-7
2	0.72E-2	0.21E-1	0.74E-2	11.17
3	0.11E-1	0.38E-1	0.37E-2	0.14E-7
4	0.35E-2	0.72E-2	0.71E-3	0.14E-7
5	0.58E-2	0.26E-1	0.35E-3	0.14E-7
6	0.25E-3	0.32E-2	0.42E-4	0.14E-7
7	0.41E-5	0.71E-4	0.95E-6	0.14E-7
8	0.20E-7	0.31E-6	0.0	0.14E-7
9	0.17E-12	0.36E-10	0.0	0.14E-7
	Refinement Iteration 2: Hermite; "m"			
1	0.70E-2	0.10E-1	0.14E-1	0.14E-7
2	0.72E-3	0.31E-2	0.47E-2	250.30
3	0.10E-3	0.57E-3	0.85E-3	0.14E-7
4	0.34E-4	0.19E-3	0.29E-3	0.14E-7
5	0.13E-4	0.59E-4	0.87E-4	0.14E-7
6	0.24E-5	0.12E-4	0.19E-4	0.14E-7
7	0.26E-5	0.20E-5	0.30E-5	0.14E-7
8	0.12E-5	0.41E-6	0.62E-6	0.14E-7
9	0.54E-7	0.21E-7	0.31E-7	0.14E-7
10	0.14E-9	0.57E-10	0.31E-7	0.14E-7

i = iteration no.; Υ = constraint error Eq. (50); $\lVert \vartheta \rVert$ = projected gradient norm Eq. (44); τ = Levenberg parameter; and $\lVert r \rVert$ = penalty weight norm Eq. (46).

of the number of function evaluations [the number of times $f(x)$ and $c(x)$ are evaluated] and the solution time. For the test set the best results were obtained for 50 problems using the default "fm" strategy. Strategy "fme" was the most efficient on 14 problems. Strategy "m" performed best on 9 problems, and 1 problem was solved best using strategy "f." Only the most conservative "fme" strategy successfully solved all problems; however, the strategy "fm" was sufficiently robust and slightly more efficient to warrant making it the default.

Table 2 presents a summary of the grid refinement test set, with a requested relative error of 10^{-3}. All problems were initiated with an equally spaced grid corresponding to the smallest problem size in the class. Trapezoidal discretization was used for the first iteration, and subsequent grid refinement iterations used a Hermite-Simpson discretization with grid points located by the refinement algorithm. The default NLP "fm" strategy was used for the first refinement iteration, and all successive refinement iterations used strategy "m." The results in Table 2 define the converged case summary. For example, problem 1b required two refinement iterations, with the final grid consisting of 25 points and 322 NLP variables. This case required a total of 2920 function evaluations and terminated after 105.3 s.

B. Maximum Crossrange

A typical optimization iteration history is illustrated by the behavior on a maximum crossrange trajectory problem with aerodynamic heating path constraint. This problem[23] is extremely nonlinear and because of the path constraints is quite difficult to solve using an indirect formulation that requires guesses for the adjoint variables. The first NLP produced by trapezoidal discretization with 51 grid points has 358 variables and 301 nonlinear constraints. Of the 358 variables, 307 are limited by bounds and 8 are fixed by equalities, and of the 301 constraints, 250 are equalities and 51 are inequalities. The second NLP that results from Hermite-Simpson discretization has 701 variables with 8 fixed and 623 bounded. The problem has 463 nonlinear constraints, consisting of 385 equalities and 78 inequalities (55 active at the solution). Table 3 illustrates the iteration history for this problem that corresponds to problem 3e in Table 2. After locating a feasible point using the technique described in Sec. IV.E, the merit function is minimized. Note that the constraint error grows during the first few iterations as steps move away from the constraint surface, and the penalty weight was increased in step 2 to avoid excessive violation. Accelerated convergence is marked by reduction in the Levenberg parameter, which is dictated by the projected gradient norm. During the second refinement iteration that uses the "m" strategy, optimization begins directly from the initial (infeasible) point. Figures 2 and 3 in Ref. 27 present the altitude and heating profiles observed on the optimal trajectory.

C. Minimum Time to Climb

The three-degree-of-freedom formulation of the minimum time-to-climb problem[24,26] illustrates a number of features of the new technique. Table 4 presents a summary of the grid refinement iterations for this problem.

Observe that most of the NLP optimization iterations were performed on the first refinement iteration, with the number of Hessian calls dropping to one for the final refinement iterations. Also note that the number of right-hand-side evaluations increases significantly for iterations 2–6 because Hermite quadrature is used. In comparison, suppose that grid refinement was not used, and instead, the solution with 100 equally spaced grid points was computed [see Table 4, problem 6a, with $n_g = 100$ in. (Ref. 27)]. Not only is the total time required less with grid refinement (672.58 vs 451.15), but the equally spaced configuration *does not* satisfy the relative error in the differential equation. Limited computational experience seems to confirm that it is usually better to utilize the grid refinement strategy when no a priori knowledge of the solution is available. Figure 2 illustrates the altitude-time profile for this problem, including the placement of the grid points during the refinement process.

The results obtained for this problem can also be compared with a standard dense optimization method. The OTIS computer program[10] uses the sequential quadratic programming code NPSOL[14] and a slightly different Hermite transcription approach. Specifically, the NPSOL/OTIS transcription uses the values of the control derivatives at the grid points, instead of the values of the control at the interval midpoint (10). The same atmosphere, thrust, and aerodynamic data were used for the comparison. As a reference value, we assume that the grid refinement solution with 99 grid points is "truth" and $\phi^* = 324.9912565$ s. Table 5 summarizes the results of this comparison. It is important to note that the first line of the table that shows the difference between the OTIS solution and ϕ^* does *not* imply the solution is wrong: It is simply *different* because of the aforementioned modeling difference. Instead, it seems clear that the results are "close" to each other from an engineering standpoint and are both converging to similar values. A comparison of the values T_s and T indicates that when

Table 4 Minimum time-to-climb iteration summary

i_r	n_g	HC	FE	RHS	T
1	10	11	886	8,860	27.38
2	25	5	1,005	49,245	71.94
3	47	1	319	29,667	49.32
4	71	1	319	44,979	84.51
5	92	1	282	51,606	101.68
6	99	1	282	55,554	116.32
		20	3,093	239,911	451.15

i_r = refinement iteration no; n_g = no. grid points ($n_g = n_s + 1$); HC = no. Hessian calls; FE = no. function evaluations; RHS = no. right-hand-side evaluations; and T = solution time, s.

Table 5 NPSOL/OTIS comparison

n_g	10	20	40
$\Delta\phi_0$	3.4704	4.22E-1	4.18E-2
$\Delta\phi$	1.70E-1	1.66E-1	−3.63E-2
T_0	73.72	759.32	7794.65
T_s	140.76	276.81	550.35
T	67.83	165.31	295.47
R_1	0.52	2.74	14.16
R_2	1.08	4.59	26.38

n_g = no. grid points ($n_g = n_s + 1$); $\Delta\phi_0$ = OTIS: $\Delta\phi = \phi - \phi^*$; T_0 = NPSOL/OTIS: solution time, s; T_s = SPR/OTIS: solution time, s; T = solution time, s; R_1 = speedup $R_1 = T_0/T_s$; and R_2 = speedup $R_2 = T_0/T$.

the sparse optimization algorithm was used to solve the same problem on OTIS vs in a "stand-alone" test environnment, there is an overhead cost incurred. In OTIS a number of auxiliary quantities are computed at every right-hand-side evaluation that are not essential to the optimization problem solution. This overhead increases the cost of a function evalution in the OTIS implementation by a factor (≈ 1.87) relative to the stand-alone test value T. Note that for the smallest problem, the overhead actually causes the SPR/OTIS time T_s to be worse than the NPSOL/OTIS time T_0, which is primarily due to the fact the sparse algorithm uses central-difference gradients, whereas NPSOL uses forward-difference estimates. However, as the number of grid points increases, the computational cost is dominated by linear algebra, which for NPSOL is proportional to N^3, and the overhead penalty becomes less significant. The NPSOL/OTIS software was not run for the case with 99 grid points because it required more *storage* than was available. Although storage comparisons are difficult, experience suggests that the dense method is limited to approximately 350 variables and constraints for the Sun IPC workstation configuration. In contrast, the sparse algorithm can easily accommodate problems with 10 times as many variables and constraints within the same memory limitations. Furthermore, previously reported results[7] have demonstrated that problems with nearly 13,000 variables can be solved on larger computers. Finally, we mention that the results presented reflect computation times on a scalar processor. We would expect significant additional speedups from the sparse optimization algorithm on a vector processor, since the underlying linear algebra routines have been designed to exploit vectorization.

As a final practical observation, note that the accuracy of the computed objective function value for all cases (even with 10 grid points) is probably acceptable for preliminary analyses. Unfortunately, the corresponding state and control variable histories may not be accurate enough to permit the results to be reproduced by conventional methods, e.g., by numerically integrating the trajectory from the beginning to the end. In general, high-fidelity applications of direct transcription methods require a large number of grid points. Grid refinement and sparse optimization methods make these applications feasible.

VIII. Summary and Conclusions

This paper presents a method for solving path-constrained trajectory optimization problems using a sparse nonlinear programming algorithm. A comparison of three different strategies for using a sparse quadratic programming algorithm suggests that an approach that first locates a feasible point and then stays "near" the constraints produces a reasonable compromise between speed and robustness. A grid refinement strategy has been proposed to exploit the benefits of trapezoidal discretization, followed by refinement with a higher-order Hermite-Simpson approach. When coupled with the sparse nonlinear programming algorithm, the technique has demonstrated both speed and reliability when solving optimal control problems with both equality and inequality constraints.

Acknowledgment

The authors wish to acknowledge the insightful contributions of Paul Frank during the development and implementation of this trajectory optimization method, especially in the development of an efficient, robust sparse quadratic programming algorithm.

References

[1]Ascher, U., Christiansen, J., and Russell, R., "Collocation Software for Boundary-Value ODEs," *ACM Transactions on Mathematical Software*, Vol. 7, No. 2, June 1981, pp. 209–222.

[2]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.

[3]Dickmanns, E. D., "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), Sept. 15–17, 1980.

[4]Dickmanns, E. D., "Kollozierte Hermite-Approximation dritter und fünfter Ordnung mit automatischer Gitteranpassung zur Lösung von Randwertproblemen der optimalen Steuerungen," Hochschule der Bundeswehr München, LRT/WE, 13a/FB79-1.

[5]Russell, R. D., and Shampine, L. F., "A Collocation Method for Boundary Value Problems," *Numerical Mathematics*, Vol. 19, 1972, pp. 1–28.

[6]Betts, J. T., and Huffman, W. P., "Trajectory Optimization on a Parallel Processor," *Journal of Guidance, Control, and Dynamics*, Vol. 14, No. 2, 1991, pp. 431–439.

[7]Betts, J. T., and Huffman, W. P., "The Application of Sparse Nonlinear Programming to Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 1, 1992, pp. 198–206.

[8]Enright, P. J., "Optimal Finite-Thrust Spacecraft Trajectories Using Direct Transcription and Nonlinear Programming," Ph.D. Dissertation, Univ. of Illinois, 1991.

[9]Enright, P. J., and Conway, B. A., "Optimal Finite-Thrust Spacecraft Trajectories Using Collocation and Nonlinear Programming," AAS/AIAA Astrodynamics Conference (Stowe, VT), 1989 (Paper 89-350).

[10]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. 338.

[11]Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H., "Some Theoretical Properties of an Augmented Lagrangian Merit Function," Rept. SOL 86-6, Dept. of Operations Research, Stanford Univ., Stanford, CA.

[12]Rockafeller, R. T., "The Multiplier Method of Hestenes and Powell Applied to Convex Programming," *Journal of Optimization Theory and Applications*, Vol. 12, 1973, pp. 555–562.

[13]Gill, P. E., Murray, W., and Wright, M. H., *Practical Optimization*, Academic, New York, 1981.

[14]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," Rept. SOL 86-2, Dept. of Operations Research, Stanford Univ., Stanford, CA.

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

[16]Huffman, W., and Carter, M., "Software for Sparse Finite Difference Derivatives," ECA-LR-71, Boeing Computer Services, Seattle, WA, 1991.

[17]Ashcraft, C. C., "A Vector Implementation of the Multifrontal

Method for Large Sparse, Symmetric Positive Definite Linear Systems," Boeing Computer Services Tech. Rept. ETA-TR-51, Seattle, WA, 1987.

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

[19] Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H., "A Schur-Complement Method for Sparse Quadratic Programming," Rept. SOL 87-12, Dept. of Operations Research, Stanford Univ., Stanford, CA.

[20] Frank, P. D., and Betts, J. T., "A Quadratic Programming Code for Large Sparse Problems," ECA-TR (to appear).

[21] Ascher, U., Mattheij, R., and Russell, R. D., *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations,* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[22] Bryson, A. E., and Ho, Y. C., *Applied Optimal Control,* Wiley, New York, 1975.

[23] Zondervan, K. P., Bauer, T., Betts, J., and Huffman, W., "Solving the Optimal Control Problem Using a Nonlinear Programming Technique, Part 3: Optimal Shuttle Re-entry Trajectories," *Proceedings of AIAA/AAS Astrodynamics Conference* (Seattle, WA), AIAA, New York, 1984 (AIAA Paper 84-2039).

[24] Hargraves, C. R., Johnson, F., Paris, S., and Rettie, I., "Numerical Computation of Optimal Atmospheric Trajectories," *Journal of Guidance and Control,* Vol. 4, No. 4, 1981, pp. 406–414.

[25] Betts, J. T., "The Nonlinear Quadratic Regulator Formulation for Guidance Targeting and SAIR," Boeing Computer Services Tech. Rept. G-3713-JTB-060, Seattle, WA, Sept. 4, 1990.

[26] Bryson, A. E., Desai, M. N., and Hoffman, W. C., "Energy-State Approximation in Performance Optimization of Supersonic Aircraft," *Journal of Aircraft,* Vol. 6, No. 6, 1969, pp. 481–487.

[27] Betts, J. T., and Huffman, W. P., "Path Constrained Trajectory Optimization Using Sparse Sequential Quadratic Programming," *Proceedings of the AIAA Guidance, Navigation, and Control Conference* (New Orleans, LA), AIAA, Washington, DC, 1991, pp. 1236–1259.