Sparse Jacobian Updates in the Collocation Method for Optimal Control Problems

John T. Betts*
Boeing Computer Services, Seattle, Washington 98124

Advanced guidance algorithms for the control of aerospace vehicles can require rapid solution of an optimal control problem. The necessary conditions for the solution of an optimal control problem result in a two-point boundary-value problem. For guidance applications, the boundary-value problem must be solved rapidly in order to reflect real-time navigation input. The collocation method has been proposed by a number of authors as a robust approach to the problem. By introducing piecewise cubic polynomial interpolation of the dynamic variables, the boundary-value problem is reduced to solving a system of nonlinear algebraic equations. The resulting iteration equation involves a large sparse matrix. This paper demonstrates the application of sparse Broyden updates for the iteration matrix that results in significant time savings. The viability of the approach for real-time optimal control applications is illustrated with computational results for maximum downrange and crossrange Shuttle re-entry trajectories and for simpler powered flight ascent trajectories. A substantial reduction in computation cost has been observed for typical cases.

Introduction

THE design of a trajectory for many aerospace vehicles can be formulated as an optimal control problem. The effective solution of a general optimal control problem is a notoriously challenging computational task. When a vehicle actually flies a trajectory, it is subjected to random perturbations that cannot be predicted in preflight or nominal mission design. Consequently, even though the nominal mission design may be optimally configured, the actual trajectory flown will exhibit a degradation in the performance index. The role of a guidance algorithm is to maintain control of a vehicle in the presence of errors and, when possible, optimize the trajectory.

Since the perturbing forces on a trajectory are assumed to be small, most guidance techniques attempt to exploit this by introducing approximations that are valid in a small neighborhood of the nominal trajectory. For example, it is common to approximate the nonlinear control problem using linear dynamics and a quadratic performance index. The papers by Breakwell et al.[1] and Kelley[2] describe the fundamentals of this technique. The primary reason for introducing approximations is to avoid the computational burden of solving a general nonlinear optimal control problem. With the increased power of onboard computers, it has become desirable to solve the trajectory design problem in real-time in order to maintain vehicle autonomy and enhance mission flexibility. The Inertial Upper Stage Gamma Guidance algorithm[3] represents a first step in this direction, since it effectively solves an impulsive orbit transfer problem to determine steering commands. The method described in Bradt et al.[15] extends the philosophy of Ref. 3, to solve a general optimal control problem without introducing approximations about a preflight nominal trajectory. The method introduced in Ref. 15 uses a general nonlinear programming algorithm in conjunction with collocation as proposed for preflight trajectory design by Hargraves and Paris.[16]

The guidance application is characterized by the need to repeatedly solve a similar problem. Furthermore, the solution obtained on one guidance call should provide a good prediction for the solution on the next guidance call. It is important to note, however, that successive solutions may deviate substantially from a preflight nominal trajectory, thus precluding linearization about a nominal trajectory profile.

The necessary conditions for the optimal solution of a control problem result in a nonlinear two-point boundary-value problem. The shooting method for solving a boundary-value problem, although straightforward, is not sufficiently robust, primarily because of inherent instability in the adjoint differential equations. A number of authors[4-6] have proposed using the collocation method to solve the boundary-value problem. In this paper, the approach of Dickmanns[4] is extended to exploit the characteristics of the guidance environment. Specifically, the use of a sparse Broyden update for the Jacobian matrix is introduced, thereby significantly reducing the cost of an iteration. Computational experience on a number of problems, including a Shuttle re-entry, demonstrates the viability of the technique.

Optimal Control Problem

The general optimal control problem can be stated in a number of ways. We choose a form that is well-suited to guidance applications. Let us find the m-dimensional control vector $u(t)$ to minimize the performance index

$$J = \phi[x(t_f), t_f] \tag{1}$$

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

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

where the n_e dimension state vector x has given initial conditions at time t_0; i.e., given

$$x(t_0) \tag{3}$$

Presented as Paper 88-4150 at AIAA Guidance, Navigation and Control Conference, Aug. 15-17, 1988; received Nov. 14, 1989; revision received March 20, 1989.

*Applied Mathematician, Scientific Computing and Analysis Division.

Furthermore, at the final time we impose the n_q constraints

$$\psi[x(t_f),t_f] = 0 \tag{4}$$

The Hamiltonian function is

$$H = \lambda^T f \tag{5}$$

where λ is an n_e vector of adjoint variables. The adjoint equations

$$\dot{\lambda} = -\left(\frac{\partial H}{\partial x}\right)^T = -\left(\frac{\partial f}{\partial x}\right)^T \lambda \tag{6}$$

must satisfy the transversality conditions

$$T = \lambda(t_f) - \left(\frac{\partial \phi}{\partial x} + \nu^T \frac{\partial \psi}{\partial x}\right)^T_{t=t_f} = 0 \tag{7}$$

as well as

$$R = \left[\frac{\partial \phi}{\partial t} + \nu^T \frac{\partial \psi}{\partial t} + \left(\frac{\partial \phi}{\partial x} + \nu^T \frac{\partial \psi}{\partial x}\right) f\right]_{t=t_f} = 0 \tag{8}$$

where ν is an n_q-vector of constants. The optimal control vector is defined by the maximum principle

$$0 = -\left(\frac{\partial H}{\partial u}\right)^T = -\left(\frac{\partial f}{\partial u}\right)^T \lambda \tag{9}$$

The solution of the $2n_e$ differential equations (2) and (6) and the choice of the $n_q + 1$ parameters ν and t_f are determined by the $2n_e + n_q + 1$ boundary conditions [Eqs. (3), (4), (7), and (8)]. State and control variable path constraints will not be addressed in this paper, although they could be incorporated by including either a predetermined number of constrained phases or a penalty function.

Collocation Using Hermite Interpolation

Solution of the two-point boundary value problem is achieved by approximating the functions using a finite set of parameters. The boundary-value problem is then reduced to solving a set of nonlinear algebraic equations using an iteration procedure. This section describes how the boundary-value problem is approximated with a finite set of parameters using Hermite interpolation.

All dynamic variables $y = (x,\lambda)$ are approximated using piecewise cubic polynomials. We begin by dividing the time interval into n_s segments

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

Let us define the constants

$$\tau_{j+1} = (t_{j+1} - t_j)/(t_f - t_0) \tag{10}$$

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

$$\hat{y}_j = \tfrac{1}{2}(y_j + y_{j+1}) + \tfrac{1}{8}\tau_{j+1}(t_f - t_0)(\dot{y}_j - \dot{y}_{j+1}) \tag{11}$$

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

$$\dot{\hat{y}}_j = \frac{-3(y_j - y_{j+1})}{2\tau_{j+1}(t_f - t_0)} - \frac{1}{4}(\dot{y}_j - \dot{y}_{j+1}) \tag{12}$$

Since $\dot{y} = g(y,t)$ by construction, and g is given by the right-hand sides of Eqs. (2) and (6), 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

$$\Delta = \dot{\hat{y}} - g[\hat{y},t_m] \tag{13}$$

where t_m is the time at the midpoint of the segment. There is a $2n_e$ dimension defect vector for each of the n_s segments.

Let us now define the iteration variables

$$z = \left[\lambda(t_0),x(t_1),\lambda(t_1),\ldots,x(t_f),\lambda(t_f),\nu,t_f\right]^T \tag{14}$$

Note that z, which has dimension

$$n_d = 2n_e n_s + n_e + n_q + 1 \tag{15}$$

consists of the dynamic variables x and λ at all of the mesh points, as well as the constants ν, and final time t_f. Since the state vector $x(t_0)$ is given by Eq. (3), it is not included in z.

The iteration variables must be chosen, such that the constraint vector

$$c(z) = \begin{bmatrix} \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_{n_s} \\ \psi \\ T \\ R \end{bmatrix} = 0 \tag{16}$$

The total constraint vector is made of the defect vectors from each of the n_s segments, the n_q terminal constraints [Eq. (4)], the n_e transversality conditions [Eq. (7)], and the condition in Eq. (8). When the constraints $c(z)$ are satisfied, not only are the boundary conditions ψ, T, and R satisfied, but also the differential equations (2) and (6) are 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. The general problem of mesh refinement and its relation to integration accuracy is discussed in Refs. 4 and 6. For a guidance application, it is assumed that the mesh points have been selected to provide sufficient accuracy in the solution.

Iteration Technique

The basic iterative technique used for computing a new estimate of the variables $\bar{z}$ from an old estimate z is given by the formula

$$\bar{z} = z - \rho s \tag{17}$$

where ρ is a scalar defining the step length taken in the direction s. The step length is adjusted using a quadratic interpolation method when required to insure a reduction in the constraint error, i.e.,

$$\| c(\bar{z}) \| \leq \| c(z) \| \tag{18}$$

The search direction vector is computed by solving the linear system

$$Gs = c \tag{19}$$

where G is the $n_d \times n_d$ Jacobian matrix

$$G = \begin{bmatrix} G_{11} & G_{12} & 0 & \cdots & 0 & 0 & G_{t1} \\ 0 & G_{22} & G_{23} & \cdots & 0 & 0 & G_{t2} \\ \vdots & & \ddots & \ddots & & & \vdots \\ & & & G_{n_s n_s} & G_{n_s n_s+1} & 0 & G_{tn_s} \\ 0 & \cdots & & & G_{bz} & G_{bv} & G_{bt} \end{bmatrix} \tag{20}$$

The submatrices have the following dimensions

$$\begin{aligned} G_{11} &: n_b \times n_e \\ G_{ij} &: n_b \times n_b \\ G_{ti} &: n_b \times 1 \\ G_{bz} &: (n_e + n_q + 1) \times n_b \\ G_{bv} &: (n_e + n_q + 1) \times n_q \\ G_{bt} &: (n_e + n_q + 1) \times 1 \end{aligned}$$

where the block size $n_b = 2n_e$. Because the dimension of G depends on the number of segements n_s, it is quite likely to be large. Consequently, it is very important to exploit the sparsity of G, not only to reduce storage requirements, but also to reduce the computation times.

There are two principal contributions to the computational cost of an iteration. The first is the numerical solution of the linear system [Eq. (19)], and the second is the evaluation of the Jacobian matrix G. In Ref. 7, Dickmanns describes a method for solving Eq. (19), which is essentially a Gaussian elimination technique applied to the blocks of G. The approach requires solving an n_b dimensional dense system $(n_s - 1)$ times, and then solving a dense $(3n_e + n_q + 1)$ system. Since the cost of solving a dense system is $O(n^3)$, the computational overhead associated with this sparse technique is $O[(n_s - 1)n_b^3 + (3n_e + n_q + 1)^3]$. By comparison, solving Eq. (19) directly would require $O(n_d^3)$ operations, where n_d is given by Eq. (15). For a typical example, such as the Shuttle re-entry to be described, $n_s = 50$, $n_e = 6$, $n_q = 3$, and the dense solution would require nearly 2400 times as many numerical operations. In other words, the sparse method is nearly four orders of magnitude faster than the dense method. A Householder decomposition technique is used in the current implementation to solve the small blocks in the sparse system. Although the current implementation is acceptable, further time savings may be possible by incorporating an alternate sparse solution technique and/or utilizing vector or parallel processing methods.

Sparse Broyden Update

Computing the Jacobian matrix G in Eq. (19) is often the most costly operation, especially if it is constructed using finite differencing. When a forward or backward difference technique is used, the constraint vector c must be evaluated an additional n_d times to construct the Jacobian, and, if a central difference method is used, at least $2n_d$ evaluations must be made. An alternate approach is to construct the Jacobian by updating a previous estimate, using only a single constraint evaluation. Thus, we are interested in a formula of the form

$$\bar{G} = G + B(\delta c, \delta z) \tag{21}$$

where

$$\delta c = c(\bar{z}) - c(z) \tag{22}$$

$$\delta z = \bar{z} - z \tag{23}$$

For a dense Jacobian matrix, the Broyden update formula is just

$$B(\delta c, \delta z) = [(\delta c - G\delta z)\delta z^T]/\delta z^T \delta z \tag{24}$$

Schubert[8] and Broyden[9] suggested a generalization of the update, which maintains sparsity. Specifically,

$$B_s(\delta c, \delta z) = P_z[D^+(\delta c - G\delta z)\delta z^T] \tag{25}$$

where D^+ is the pseudoinverse of a diagonal positive semidefinite matrix with

$$D_{ii} = \delta \hat{z}_i^T \delta \hat{z}_i \tag{26}$$

The vector $\delta \hat{z}_i$ is constructed by imposing the sparsity pattern for the ith row of G on δz. The projection operator P_z zeros out the elements corresponding to the zeros of G.

Sparse Finite-Difference Jacobian

Another way for computing the Jacobian of a sparse matrix was suggested by Curtis et al.[10] Essentially, the approach requires partitioning the column indices of the Jacobian G into subsets Γ^k. Each subset of the columns of G has the property that there is at most one nonzero element in each *row*. Then we define the perturbation direction vector by

$$\delta z^k = \sum_{j \in \Gamma^k} h_j e_j \tag{27}$$

where h_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 $j \in \Gamma^k$, we can approximate element ij of the Jacobian by

$$G_{ij} \approx \frac{c_i(z + \delta z^k) - c_i(z)}{h_j} \tag{28}$$

The total number of evaluations required is just the total number of index sets Γ^k needed to span the columns of G.

For a matrix with the sparsity structure of Eq. (20), the sparse differencing can be implemented as follows, assuming n_s is an even number. Set

$$\Gamma^k = \left[k + n_e, k + n_e + 2n_b, \ldots, k + n_e + 2n_b[(n_s/2) - 1]\right] \tag{29}$$

for $k = 1, 2, \ldots, 2n_b$;

$$\Gamma^k = [k - 2n_b, k - 2n_b + n_e + n_s n_b] \tag{30}$$

for $k = (2n_b + 1), \ldots, (2n_b + n_q)$;

$$\Gamma^k = [k - 2n_b] \tag{31}$$

for $k = (2n_b + n_q + 1), \ldots, (2n_b + n_e)$;

$$\Gamma^k = [n_d] \tag{32}$$

for $k = 5n_e + 1$. The index sets given by Eq. (29) compute all of the blocks along the diagonal of Eq. (20), namely G_{12}, G_{22} through $G_{n_s n_s+1}$, and G_{bz}. The index sets [Eq. (30)] produce the n_q columns of G_{bv} and the first n_q columns of G_{11}. The remaining $n_e - n_q$ columns of G_{11} are defined by the sets [Eq. (31)], and the final column of G is produced by set [Eq. (32)]. The perturbation scheme is modified slightly when n_s is odd. Notice that the total number of perturbations is given by

$$n^* = \begin{cases} 5n_e + 1 & n_s \text{ even} \\ 7n_e + 1 & n_s \text{ odd} \end{cases} \tag{33}$$

Note that the maximum number of perturbations is independent of the number of segments n_s!

Although the partitioning scheme defined by Eqs. (29-32) is quite efficient, it may not be the optimal partitioning. Coleman and Moré[11] 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 and will be pursued in future research.

Numerical Results

To develop an efficient computational algorithm, a series of numerical test cases have been solved that incorporate the techniques discussed. There are a number of criteria that should be used to measure the efficiency of a computational algorithm. Clearly, the two principal areas of concern are computer storage and run time. Unfortunately, absolute answers to the questions, "How fast does it run?" and "How much memory does it use?" must be deferred until specific hardware is determined. Instead, we must be satisfied with *relative* answers. The goal of the paper has been to reduce storage and run time by exploiting the sparsity of the Jacobian matrix. For most trajectory applications the computational cost and storage requirements are dominated by the cost of a function evaluation. Consequently the principal figure of merit used to assess an algorithm will be the number of function evaluations, where a "function evaluation" consists of a single evaluation of $c(z)$. All results presented were obtained using an SCS-40 computer. The iteration was terminated when $|c_i| \leq 10^{-10}$ for all i.

Brachistochrone

The first series of tests were run on the classical Brachistochrone problem, as described by Bryson and Ho.[12] The formulation used has one control and three state variables, and initial guesses were constructed as multiples of the known solution, i.e., $z^{(0)} = kz^*$. The first set of cases were designed to compare the sparse Broyden update [Eq. (25)] with the dense version of the update [Eq. (24)]. A single update was done per iteration and the initial Jacobian was computed using a central differencing technique with perturbation sizes adjusted for accuracy. Table 1 presents the results of four cases, beginning from different initial guesses, i.e., different multiples k.

For all cases $n_s = 4$ and $n_d = 29$. For all four cases the sparse update required more iterations and, hence, more function evaluations than the dense update. Based on the number of iterations there would seem to be no reason to choose the sparse update. However, a sparse iteration should require $O(1979)$ operations, whereas a dense iteration should require $O(24{,}389)$ calculations. Consequently, the ultimate choice is dictated by factors not addressed, namely the cost of a function evaluation relative to the cost of an iteration.

Table 2 presents a collection of cases designed to demonstrate various algorithmic features and options. The initial Jacobian matrix was computed three different ways. The first way was central differencing with perturbation size adjustment, and is denoted as C. Forward difference estimates are indicated by F, and when the sparse differencing method of Eqs. (27-32) was used it was denoted by the symbol S. After the initial Jacobian was constructed, two options for computing subsequent Jacobians were investigated. The first approach is to update the Jacobian using the sparse Broyden formula [Eq. (25)], and this is denoted by the symbol B. The second alternative is to recompute the Jacobian at each iteration, using the same perturbation technique used to construct G^0. Since this is just a Newton method, we denote it by the symbol N.

The first case in Table 2 is just a repeat of the first case in Table 1. In case 1, the initial Jacobian was computed using central differences, with the perturbation size adjusted to balance truncation and roundoff error in the estimates. In case 2, the initial Jacobian was computed using forward differences, with no perturbation size adjustment, thereby reducing the number of function evaluations from 122 to 30. However, since the perturbation sizes were not chosen optimally, the initial Jacobian G^0 was not as accurate, and the number of iterations increased from 27 to 29. The remaining four cases all had 10 segments and 65 iteration variables. Comparing case 2 with case 3, we notice that only two additional iterations were required and that most of the additional function evaluations were necessary to compute G^0. When the Jacobian was initialized with sparse differences in case 4, the cost was reduced from 66 to 17 function evaluations, and produced no change in the number of iterations. In case 5, the number of iterations was reduced from 31 to 5, by recomputing a sparse difference Jacobian at each iteration. Note, however, that the total number of function evaluations increased to 70 from 47, even though there were fewer iterations. For comparison with case 5, a Newton method employing forward difference Jacobians was included. Note that the same number of iterations were required; however, the number of function evaluations (265) was much larger. The results of case 6 should be analogous to the method proposed by Dickmanns.[4,7]

Table 1 Sparse vs dense Broyden update

Guess	Function evaluation for G^0	Iterations, sparse/dense	Total function evaluations sparse/dense
0.5	122	27/18	148/139
0.75	122	15/13	136/134
1.25	125	14/12	138/136
1.5	119	21/17	139/135

Table 2 Brachistochrone results

Case	n_s	n_d	Function evaluations for G^0	Iterations	Total function evaluations
1 *C-B*	4	29	122	27	148
2 *F-B*	4	29	30	29	58
3 *F-B*	10	65	66	31	96
4 *S-B*	10	65	17	31	47
5 *S-N*	10	65	17	5	70
6 *F-N*	10	65	66	5	265

Table 3 Powered flight results

Case	n_s	n_d	Function evaluations for G^0	Iterations	Total function evaluations
1 Linear tangent	4	40	41	24	69
2 Bilinear tangent	4	41	42	35	90
3 Bilinear tangent	8	73	74	24	101

Table 4 Maximum downrange results

Case	Function evaluations for G^0	Iterations	Total function evaluations
1 Good guess, *F-B*	611	6	616
2 Good guess, *S-N*	32	3	66
3 Good guess, *S-B*	32	6	37
4 Bad guess, *F-B*	611	211	980
5 Bad guess, *S-N*	32	14	421
6 Bad guess, *S-B*	32	211	401

Powered Flight

Two simple planar powered flight cases were solved to demonstrate viability of the method to ascent vehicle guidance. The problems are defined in Appendix A. For both problems the objective is to minimize the final time t_f, and the analytic solution can be found in Ref. 12. When all four of the state variables are constrained at the terminal point, the optimal control obeys a bilinear tangent law, and when the final range (x_1) is free, the optimal control is given by a linear tangent expression. The initial guess for the iteration variables z was constructed by linearly interpolating between the end conditions for the analytic solution (cf. Ref. 12, p. 61), with $t_f = 0.5$. All cases used a forward difference initial Jacobian with Broyden updating (F-B) algorithm strategy. Table 3 summarizes the results.

The final case in Table 3 incorporated an asymmetric grid distribution, that is, the constants τ_i given by Eq. (10) were not just set to $1/n_s$, as in all other examples. Instead, $\tau_1 = \tau_8 = 1/4$, $\tau_2 = \tau_7 = 1/8$, and $\tau_3 = \tau_4 = \tau_5 = \tau_6 = 1/16$. Notice that even though the final case had more variables than the second case, the method converged in fewer iterations because of the improved accuracy produced by the asymmetric grid.

Optimal Aerodynamic Control of Shuttle Re-Entry

The final collection of cases involve the optimal control of the re-entry trajectory for a vehicle that represents a simplified model of the space shuttle. The problems are defined in Appendix B. A spherical, nonrotating Earth, with an exponential atmosphere and simplified vehicle aerodynamics, was used. In addition, the dynamics equations were written in intrinsic coordinates, using characteristic or canonical units, as defined in Ref. 13. A similar set of maximum downrange and crossrange problems were solved in Ref. 14, although aerodynamic heating constraints are omitted here for simplicity. These problems are extremely difficult to solve using a shooting method because the state and adjoint equations are numerically unstable. Furthermore, a large number of mesh points are necessary to accurately integrate the differential equations, because the solutions are oscillatory in nature.

Two different initial guesses were used to begin the iteration for a maximum downrange solution. A "good guess" was obtained by numerically integrating the state and adjoint equations, using the optimal values found in Ref. 14. Even though state and adjoint variables were accurate to more than 10 figures, the integrated solution at the final time exhibits an altitude error of 361 ft, a velocity error of 19.4 fps, and flight-path angle error of 0.01 deg. In essence, the good guess results from numerical instability in reproducing a known optimal solution. The so-called "bad guess" was obtained by perturbing the good guess approximately 10%, i.e., $z^0 = 1.1z^*$. The maximum downrange subject to the terminal constraints is 187.5079° longitude at a trajectory time of 3633.667 s. The maximum crossrange solution yields a latitude of 34.14114°, at a longitude of 75.31535° and trajectory time of 2008.584 s. The optimal angle of attack and bank angle histories for these cases are very similar to those presented in Ref. 14. All results are characterized by the parameters $m = 2$, $n_e = 6$, $n_q = 3$, $n_s = 50$, and $n_d = 610$.

Examination of the results presented in Table 4 leads to conclusions similar to those seen for the Brachistochrone example. It is clearly preferable to compute the initial Jacobian using the sparse finite-difference approach. It also seems clear that the basic Newton method with sparse difference Jacobian estimates requires fewer iterations to converge than the Broyden update approach. However, the fewest number of function evaluations are needed by the Broyden method with sparse difference G^0.

Table 5 Sparse difference Newton method

Case	Iterations	Total function evaluations
1	3	66
2	3	66
3	3	66
4	3	66
5	3	66
6	3	66
7	3	66
8	4	98
9	4	98
10	4	98
11	4	98
12	4	98
13	4	98
14	4	98
15	4	98
16	4	98
17	4	98
18	5	130
19	5	130
20	19	590
21	10	293

Table 6 Sparse Broyden method

Case	Iterations	Total function evaluations	Case	Iterations	Total function evaluations
1	6/6	37/37	22	7/20	38/21
2	5/6	36/6	23	8/22	39/23
3	5/7	36/7	24	8/25	39/29
4	5/8	36/8	25	8/25	39/26
5	5/8	36/8	26	8/33	39/38
6	5/9	36/9	27	8/31	39/36
7	6/9	37/9	28	9/31	40/37
8	6/10	37/10	29	9/42	40/53
9	6/11	37/11	30	9/42	40/54
10	6/10	37/10	31	9/55	40/77
11	6/11	37/11	32	10/51	41/68
12	6/11	37/11	33	10/47	41/60
13	6/13	37/13	34	11/47	42/62
14	6/13	37/13	35	11/62	42/94
15	6/13	37/13	36	12/79	43/125
16	6/14	37/14	37	14/97	45/157
17	6/15	37/15	38	18/62	49/140
18	6/16	37/16	39	41/54	98/94
19	7/16	38/16	40	51/37	113/97
20	7/17	38/17	41	23/24	55/27
21	7/18	38/18			

The final set of results are presented in Tables 5 and 6. The primary goal was to solve a series of similar problems, much as would be done by an onboard guidance algorithm. To this end we minimize

$$J = -(1-\epsilon)\tilde{x}_2(t_f) - \epsilon\tilde{x}_3(t_f) \tag{34}$$

where $\tilde{x}_2(t_f)$ is the longitude, $\tilde{x}_3(t_f)$ the latitude, and the embedding parameter

$$\epsilon = (i-1)/(N-1) \tag{35}$$

where i is the case number and N the total number of cases. Notice that for case 1, $i = 1$ and $\epsilon = 0$, so that the overall problem is a maximum downrange trajectory. The maximum crossrange trajectory occurs for case N, when $\epsilon = 1$. Table 5 summarizes the results of 21 different cases, beginning with a maximum downrange trajectory and ending with a maximum cross-range trajectory. A sparse finite-difference Newton method was used for all cases. The solution for case k was used as the initial guess for case $k+1$. The initial guess for case 1 was the "good guess," so that case 1 on Table 5 is the same as case 2 on Table 4. Except for the last two cases, all solutions were obtained in five iterations or less.

Table 6 presents similar results, when 41 cases were solved using the embedding approach. Two different versions of the iteration algorithm were employed. Both versions used a sparse Broyden update and carried over the initial guess for z from the preceding case. One version of the algorithm constructed the initial Jacobian using sparse finite differencing. The second version carried over the final Jacobian from case k to use as the initial Jacobian for case $k+1$. For the second version, only case 1 used the sparse difference estimate for G^0. The results for each version are presented in adjoining columns. For example, case 3 required 5 iterations and 36 function evaluations to converge when the Jacobian was initialized by sparse differences. In contrast, when the Jacobian from case 2 was used to initialize case 3, 7 iterations and 7 evaluations were required for convergence. In fact, the Jacobian

carryover strategy provided better results for the first 28 cases and the last 3 cases. Only cases 29–38 showed sufficient degradation to warrant recomputing G^0 by differencing. It should be observed, however, that the carryover strategy nearly always requires more iterations.

Summary and Conclusions

Advanced guidance algorithms often require the rapid solution of two-point boundary-value problems. Toward this end, the collocation method has been specialized by incorporating a sparse Broyden update in conjunction with a sparse finite-difference estimation technique. Computational experience with a series of test cases demonstrates the overall viability of the approach. In general, a sparse Broyden method requires the fewest number of function evaluations to converge, provided that the iteration begins with a reasonable estimate for the Jacobian matrix. For many guidance applications, a reasonable estimate can be provided by simply using the result of an earlier guidance call, i.e., carry over the Jacobian. After many updates, the accuracy of the Jacobian may degrade. In this case, or when beginning a new problem, an efficient alternative is to compute the Jacobian using a sparse finite-difference method. Generally speaking, if the Jacobian matrix is more accurate, the number of iterations is reduced. Consequently, the ultimate choice of method will be dictated by the computational cost of a function evaluation compared to the cost of an iteration, i.e., solving a sparse linear system. An answer to this question must be deferred until both the hardware and the function evaluation process are more precisely defined.

Although the basic thrust of the paper was oriented toward improving the efficiency of the collocation method for guidance applications, the approach presented is also applicable when solving optimal control problems in other environments. It has been demonstrated that a combination of the sparse Broyden update with the sparse finite differencing significantly reduces the solution time when compared to a standard finite-difference Newton method.

Appendix A: Simplified Powered Flight Examples

State Equations

The equations of motion for a point mass acted upon by a thrust force of magnitude ma are

$$\begin{aligned}\dot{x}_1 &= x_3\\ \dot{x}_2 &= x_4\\ \dot{x}_3 &= a\cos u_1\\ \dot{x}_4 &= a\sin u_1\end{aligned}$$

where $a = 100$.

Linear Tangent Case

When the end conditions are

$$\begin{aligned}x_1(0) &= 0\\ x_2(0) &= 0\\ x_3(0) &= 0\\ x_4(0) &= 0\\ x_2(t_f) &= 5\\ x_3(t_f) &= \sqrt{4000}\\ x_4(t_f) &= 0\end{aligned}$$

the optimal control is given by the "linear tangent" law, $\tan u_1 = c_1 - c_2 t$.

Bilinear Tangent Case

When the end conditions are

$$\begin{aligned}x_1(0) &= 0\\ x_2(0) &= 0\\ x_3(0) &= 5\\ x_4(0) &= 0\\ x_1(t_f) &= 5\\ x_2(t_f) &= 5\\ x_3(t_f) &= 0\\ x_4(t_f) &= 5\end{aligned}$$

the optimal control is given by the "bilinear tangent" law, $\tan u_1 = (c_1 - c_2 t)(c_3 - c_4 t)^{-1}$.

Appendix B: Simplified Aerodynamic Re-Entry

Equations of Motion

The basic dynamic variables are defined, as follows:

x_1 = altitude, ft
x_2 = longitude, rad
x_3 = geocentric latitude, rad
x_4 = air relative velocity, fps
x_5 = air relative flight-path angle, rad
x_6 = air relative azimuth, rad
u_1 = angle of attack, rad
u_2 = bank angle, rad

The equations of motion in terms of these variables are

$$\begin{aligned}\dot{x}_1 &= x_4 \sin x_5\\ \dot{x}_2 &= (x_4 \cos x_5 \sin x_6)/(r\cos x_3)\\ \dot{x}_3 &= (x_4 \cos x_5 \cos x_6)/r\\ \dot{x}_4 &= -(D/m) - g\sin x_5\\ \dot{x}_5 &= (L\cos u_2/m x_4) + (\cos x_5/x_4)[(x_4^2/r) - g]\\ \dot{x}_6 &= (L\sin u_2)/(m x_4 \cos x_5) + (x_4/r)\cos x_5 \sin x_6 \tan x_3\end{aligned}$$

where

$$\begin{aligned}r &= x_1 + c_e\\ g &= \mu/r^2\\ L &= \tfrac{1}{2}\rho x_4^2 C_L S\\ D &= \tfrac{1}{2}\rho x_4^2 C_D S\\ \rho &= \rho_0 \exp(-x_1/h_r)\end{aligned}$$

The vehicle aerodynamics are given by

$$\begin{aligned}C_L &= C_{L0} + C_{L1}\sigma u_1\\ C_D &= C_{D0} + C_{D1}\sigma u_1 + C_{D2}\sigma^2 u_1^2\end{aligned}$$

Constants

For the specific case of interest define the constant values, as follows:

$$\begin{aligned}c_e &= 0.209029 \times 10^8\\ \mu &= 0.14076539 \times 10^{17}\\ m &= 6309.432886\\ S &= 2690\\ \rho_0 &= 0.002378\\ h_r &= 23{,}800\\ C_{L0} &= -0.20704\\ C_{L1} &= 0.029244\\ C_{D0} &= 0.07854\\ C_{D1} &= -6.1592 \times 10^{-3}\\ C_{D2} &= 6.21408 \times 10^{-4}\\ \sigma &= 180/\pi\end{aligned}$$

Initial Conditions

The conditions at $t_0 = 0$, which specify the vehicle state at the re-entry point, are

$$x_1 = 260{,}000$$
$$x_2 = 0$$
$$x_3 = 0$$
$$x_4 = 25{,}600$$
$$x_5 = -1/\sigma$$
$$x_6 = 90/\sigma$$

Terminal Conditions

The conditions at the final time t_f, which is free, are

$$x_1 = 80{,}000$$
$$x_4 = 2500$$
$$x_5 = -5/\sigma$$

Transformation to Characteristic Units

When written in characteristic or canonical units, distance is measured in Earth radii (er), and "modified time" is measured in units of $(er)^{3/2}$. The modified time $\tilde{t}$ is related to the time t (in seconds) by the formula

$$\tilde{t} = a_1(t - t_0)$$

where

$$a_1 = \sqrt{\mu/c_e^3}$$

The state vector in characteristic units is related to the state vector in engineering units by the formula

$$\tilde{x}(\tilde{t}) = Ax(t)$$

where A is a diagonal matrix with $A_{22} = A_{33} = A_{55} = A_{66} = 1$, $A_{11} = c_e^{-1}$, and $A_{44} = (a_1 c_e)^{-1}$. The state equations are obtained by differentiating the preceding transformation with respect to $\tilde{t}$ and applying the chain rule. The result is

$$\frac{d\tilde{x}}{d\tilde{t}} = \tilde{f}(\tilde{x}, u, \tilde{t})$$

$$= \frac{A}{a_1} f\left[A^{-1}\tilde{x}, u, \frac{\tilde{t}}{a_1} + t_0\right]$$

References

[1]Breakwell, J. V., Speyer, J. L., and Bryson, A. E., "Optimization and Control of Nonlinear Systems Using the Second Variation," *SIAM Journal,* Vol. 1, 1963, p. 193.

[2]Kelley, H. J., "Guidance Theory and Extremal Fields," *Proceedings of the Institute of Radio Engineers,* 1962, p. 75.

[3]Hardtla, John W., "Gamma Guidance for the Inertial Upper Stage (IUS)," AIAA Guidance and Control Conference, 1978.

[4]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 Optimization, Oberpfaffenhofen, Sept. 15–17, 1980.

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

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

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

[8]Schubert, L. K., "Modification of a Quasi-Newton Method for Nonlinear Equations with a Sparse Jacobian," *Mathematical Computations,* Vol. 24, 1970, pp. 27–30.

[9]Broyden, C. G., "The Convergence of an Algorithm for Solving Sparse Nonlinear Systems," *Mathematics of Computation,* Vol. 25, 1971, pp. 285–294.

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

[11]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.

[12]Bryson, A. E., and Ho, Y. C., *Applied Optimal Control,* Wiley, New York, 1975, Chap. 2.

[13]Escobal, P. R., *Methods of Orbit Determination,* Wiley, New York, 1965, Chap. 1.

[14]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," AIAA-84-2039, *Proceedings of the AIAA/AAS Astrodynamics Conference,* AIAA, New York, 1984.

[15]Bradt, J. E., Jessick, M. V., and Hardtla, J. W., "Optimal Guidance for Future Space Applications," *Proceedings of the AIAA Guidance, Navigation and Control Conference,* AIAA, New York, 1987.

[16]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, pp. 338–342.