JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 20, No. 1, January–February 1997

Conversion of Optimal Control Problems into Parameter Optimization Problems

David G. Hull
University of Texas at Austin, Austin, Texas 78712

Several methods exist for converting optimal control problems into parameter optimization problems, and they are categorized by the unknowns of the parameter optimization problem, the numerical integration technique, and the order of the integration technique. Conversion of optimal control problems into parameter optimization problems is accomplished by replacing the control and/or state histories by control and/or state parameters and forming the histories by interpolation. Four general classes of methods exist, and the unknowns in each class are 1) control parameters, 2) control parameters and some state parameters, 3) control parameters and state parameters, and 4) state parameters. In methods of classes 1 and 2, the state differential equations are integrated by explicit numerical integration, and in methods of classes 3 and 4, implicit numerical integration is used. All orders of numerical integration can be used in each method.

Introduction

A STANDARD method for solving an optimal control problem is to convert it into a parameter optimization problem and solve it using an existing nonlinear programming code. The conversion process begins by dividing the time interval of the optimal control problem into a prescribed number of subintervals, with the times at the ends of the subintervals being called nodes. Then, the basic approach is to choose the controls and/or the states at the nodes (called the control parameters and/or the state parameters) to be the unknowns and to form the control and/or state histories by interpolation. Next, the state equations of the optimal control problem are integrated, and the nonlinear programming code iterates on the unknown parameters until the parameter optimization problem is solved.

There are four general classes of methods for converting the optimal control problem into a parameter optimization problem. By class, the unknown parameters are 1) the control parameters, 2) the control parameters and the state parameters at some nodes, 3) the control parameters and the state parameters, and 4) the state parameters. An example of a class 1 method is given elsewhere.[1] Seemingly different examples of class 3 methods also are presented in the literature,[2–4] as is a discussion of the class 4 method.[5]

In methods of classes 1 and 2, the state differential equations are integrated explicitly. As many integration steps between nodes as needed can be used to obtain the required accuracy. Because the state parameters are the unknowns in methods of classes 3 and 4, the equations of motion can be integrated implicitly or explicitly. With implicit integration, each node represents an integration step.

Because the methods discussed in the literature have been developed over a long period of time and have been developed by different researchers,[1–5] the purpose of this paper is to present these methods from a single point of view. The methods differ principally by the unknowns (control and/or state parameters), the type of method used to integrate the state differential equations (explicit or implicit), and the order of the integration. First, the optimal control problem and the parameter optimization or nonlinear programming problem are defined. Next, numerical integration is reviewed with explicit and implicit integrators being discussed from the Runge–Kutta approach. Finally, the four classes of conversion methods are developed.

Optimization Problems

The baseline optimal control problem is to find the design parameter b and the control history $u(t)$ that minimize the scalar performance index

$$J = \phi(x_f, b) \tag{1}$$

subject to the differential constraint

$$\dot{x} = g(t, x, u, b) \tag{2}$$

the prescribed initial conditions

$$t_0 = 0, \qquad x_0 = x_{0_s} \tag{3}$$

and the prescribed final conditions

$$t_f = t_{f_s}, \qquad \psi(x_f, b) = 0, \qquad \theta(x_f, b) \geq 0 \tag{4}$$

The dimensions of b, u, x, ψ, and θ are $l \times 1$, $m \times 1$, $n \times 1$, $p \times 1$, and $q \times 1$, respectively, and the subscript s denotes a specified value. In this formulation, the final time is fixed. A problem with free final time can be transformed into this format by normalizing the time. Then, $t_{f_s} = 1$, and the actual final time is one of the elements of the design parameter b.

David G. Hull received a B.S. degree in aeronautical engineering from Purdue University, Lafayette, Indiana, in 1959, an M.S. degree in aeronautics and astronautics from the University of Washington, Seattle, Washington, in 1962, and a Ph.D. in mechanical engineering from Rice University, Houston, Texas, in 1967. He has been employed by the Boeing Company and Rice University, and he has spent several summers in Sandia National Laboratories. He has progressed through the professorial ranks at the University of Texas at Austin, Austin, Texas, and is currently the M. J. Thompson Regents Professor in Aerospace Engineering and Engineering Mechanics. His research interest are in the areas of optimization, flight mechanics, trajectory optimization, and guidance. He is an Associate Fellow of AIAA.

Received Jan. 22, 1996; presented as Paper 96-3812 at the AIAA Guidance, Navigation, and Control Conference and Exhibit, San Diego, CA, July 29–31, 1996; revision received Aug. 15, 1996; accepted for publication Aug. 29, 1996.

The conversion of this optimal control problem into a parameter optimization problem begins with the definition of N fixed times

$$t_0 = t_1 < t_2 < \cdots < t_k < \cdots < t_{N-1} < t_N = t_f \tag{5}$$

called nodes, which usually are spaced equally. Then, functions of time $u(t)$ and $x(t)$ are replaced by their values at the nodes (u_k and x_k) and some form of interpolation. In general, the unknowns of the parameter optimization problem are the design parameter b and some combination of control parameters u_k and/or state parameters x_k. If X denotes the vector of unknown parameters, the corresponding parameter optimization problem is to find the value of X that minimizes the scalar performance index

$$J = F(X) \tag{6}$$

subject to the equality constraints

$$C(X) = 0 \tag{7}$$

and the inequality constraints

$$D(X) \geq 0 \tag{8}$$

The functions F and D are just different names for ϕ and θ. The function C contains ψ, but it also may contain equality constraints imposed by the numerical integration of the state equations (2). The specific forms of these functions are to be determined. In general, the solution process is to guess values for the unknown parameters and use a nonlinear programming code to find the values of the parameters that minimize the performance index and satisfy the constraints.

Before continuing, it is remarked that the optimal control problem can be made more complicated by including a free initial point with equality and inequality constraints, internal points with equality and inequality constraints, integral constraints, path equality constraints, and path inequality constraints. In all cases, conversion into a parameter optimization problem is possible. Path constraints can be converted to point constraints and included in the baseline problem by using penalty functions.[1] Alternatively, path constraints can be imposed at each node. However, if a path constraint is in effect at several consecutive nodes, it is not, in general, satisfied between the nodes. Satisfaction can be improved by adding more nodes in this area.

Numerical Integration

In this section, numerical integration is reviewed in terms of the differential equation

$$\dot{x} = f(t, x) \tag{9}$$

whose initial conditions t_1, x_1 are known. Relative to the optimal control problem, the design parameter b and the control history $u(t)$ are assumed to be known. Equation (9) is assumed to be scalar, but the results apply to vector equations. Only fixed-step integration is considered. Explicit integrators are derived from the Runge–Kutta approach, whereas implicit integrators are derived in several different ways: Runge–Kutta, finite difference, Newton–Coates, and collocation. The derivations are different, but the resulting integrators are the same.

The integration methods, explicit and implicit, normally used to convert optimal control problems into parameter optimization problems all can be derived from the Runge–Kutta approach.[6–9] Here, given the time t_i, the state $x_i = x(t_i)$, and the step size $h = t_{i+1} - t_i$, the state $x_{i+1} = x(t_i + h)$ can be obtained from the relation

$$x_{i+1} = x_i + h \sum_{j=1}^{p} c_j f_j \tag{10}$$

where

$$f_j = f\left(t_i + h\alpha_j, x_i + h \sum_{\lambda=1}^{p} \beta_{j\lambda} f_\lambda\right) \tag{11}$$

In these equations, p is the number of function evaluations to be made, and c, α, and β are constants whose values are to be determined. To obtain a particular integrator, p is specified; h is assumed to be a small quantity; and Eq. (10) is expanded in a Taylor series and compared with the general Taylor series expansion of x_{i+1}, that is,

$$x_{i+1} = x_i + \dot{x}_i h + (1/2!)\ddot{x}_i h^2 + (1/3!)\dddot{x}_i h^3 + \cdots \tag{12}$$

where

$$\begin{aligned} \dot{x} &= f \\ \ddot{x} &= f_t + f_x \dot{x} \\ \dddot{x} &= f_{tt} + 2f_{tx}\dot{x} + f_{xx}\dot{x}^2 + f_x \ddot{x} \\ &\vdots \end{aligned} \tag{13}$$

Equating corresponding terms of the two series leads to a set of equations of condition that can be solved for c, α, and β. An nth-order integrator is obtained if the two series match through terms of order h^n.

For explicit integration, the value of x needed for an evaluation of f must be known. Hence, the constraints $\alpha_1 = 0$ and $\beta_{j\lambda} = 0$ for $j \leq \lambda$ are added to the equations of condition. Explicit Runge–Kutta integrators of all orders exist. Through order four, each additional function evaluation increases the order by one. A fifth-order integrator, however, requires 6 function evaluations, and an eighth-order integrator, 13. Explicit integration allows the integration of Eq. (9) from the initial time to the final time in one pass.

Although a fourth-order Runge–Kutta integrator makes only four function evaluations per integration step and is the integrator most often used, higher-order integrators are more efficient. The global truncation error (GTE) is one order of the step size less than the local truncation error. For fourth-order and eighth-order integrators, $\text{GTE}_4 = \mathcal{O}(h^4)$ and $\text{GTE}_8 = \mathcal{O}(h^8)$. If an integration over $t_f - t_0 = 1$ is required to produce a $\text{GTE} = 10^{-8}$, the step sizes would be $h_4 = 10^{-2}$ and $h_8 = 10^{-1}$. Then, the fourth-order integrator would take 100 integration steps and make 400 function evaluations, whereas the eighth-order integrator would take 10 integration steps and make 130 function evaluations. Hence, based on step-size considerations only, the eighth-order integrator requires much less computer time than the fourth-order integrator.

For implicit integration, the value of x needed for evaluating f is not known and, to perform one integration step, a predictor–corrector approach must be used (Ref. 6, p. 248). This local iteration approach can be replaced by a global iteration approach (Ref. 7, p. 194) if the value of x at the end of every integration step is guessed and then iterated upon until the correct values are obtained. This means that $t_1, \ldots, t_N$ are known, and values for $x_1, \ldots, x_N$ are available, t_1 and x_1 being the prescribed initial conditions. It also means that the integration formula (10) is not satisfied and that the residuals

$$R_i = x_{i+1} - x_i - h \sum_{j=1}^{p} c_j f_j, \qquad i = 1, N-1 \tag{14}$$

must be driven to zero in the iteration process.

To save function evaluations, it is desirable to have $f_p = f(t_{i+1}, x_{i+1})$ so that the last function evaluation of one integration step can be used as the first function evaluation of the next step. To do so, the constraints $\alpha_p = 1$ and $\beta_{p\lambda} = c_\lambda$, $\lambda = 1, p$ are added to the equations of condition.

For one function evaluation f_1, the highest-order integrator that can be obtained is the midpoint rule ($f_1 = f_m$), which is of second order. If it is required that $f_1 = f_{i+1}$, only a first-order integrator can be obtained (implicit Euler integration). However, if f_{i+1} is available on one integration step, then f_i is available for the next integration step. Hence, both f_i and f_{i+1} are available even though only one function evaluation is made on each integration step. With these two function evaluations ($f_1 = f_i$ and $f_2 = f_{i+1}$), the second-order trapezoid rule is obtained.

With two function evaluations f_1 and f_2, it is possible to develop a fourth-order integrator, but it is not possible to have $f_2 = f_{i+1}$. If

f_i is taken to be f_{i+1} and $f_1 \neq f_i$, only a third-order integrator can be derived. However, if f_{i+1} is computed on one integration step, it is available as f_i on the next integration step. If three function evaluations ($f_1 = f_i$, $f_2 = f_m$, and $f_3 = f_{i+1}$) are used, even though only two are made on each integration step, a fourth-order integrator can be obtained, which is called the Simpson one-third rule.

Higher-order implicit integrators are discussed elsewhere.[9,10]

Conversion with b, u_k as Unknowns

In this conversion method,[1] the design parameter b and the control parameters $u_1, \ldots, u_N$ (values of the controls at the nodes) are the unknowns, and the control history $u(t)$ is formed by interpolation. Because only the control history is known, the state differential equation must be integrated explicitly.

Assume that values for the design parameter and the control parameters are given and placed in the unknown parameter vector X, that is,

$$X = \begin{bmatrix} b^T & u_1^T & \cdots & u_N^T \end{bmatrix}^T \tag{15}$$

which contains $l + mN$ elements. Because the design parameter b and the control history $u(t)$ are known, the state equation (2) can be integrated from t_0, x_0 to t_f to obtain the final state

$$x_f = x_f(X) \tag{16}$$

meaning that the final state is a function only of the unknown parameters. Substitution of Eq. (16) into the performance index (1) and constraints (4) reduces the optimal control problem to the parameter optimization problem defined by Eqs. (6–8). In other words, the performance index function is defined as

$$F(X) = \phi[x_f(X), b] \tag{17}$$

The equality constraint function is

$$C(X) = \psi[x_f(X), b] \tag{18}$$

and the inequality constraint function is

$$D(X) = \theta[x_f(X), b] \tag{19}$$

because b is contained in X.

Examples of methods for interpolating the control table $u_1, \ldots, u_N$ are high-order polynomials and piecewise polynomials such as linear and cubic splines. High-order polynomial curve fits can be excessively wavy and can affect numerical integration. Cubic splines also can produce wavy fits. For the cubic spline to be done correctly, the initial and final slopes should be included as optimization parameters. A linear spline or linear interpolation does not produce wavy fits. Also, in the computation of numerical derivatives, it is possible to save computer time by integrating only from the node previous to the perturbed node to the final time. This is not possible with the other two methods because changing one parameter changes the entire curve fit.

Any order of integrator can be used, but considerable accuracy is needed for computing the partial derivatives needed by nonlinear programming. Accuracy is achieved by a tradeoff between order and step size. Derivatives are computed by finite differences, and fixed-step integration is employed on the notion that the difference used in computing a derivative cancels GTE. This can be explained by considering a nominal path and a neighboring path obtained by perturbing one of the parameters. If fixed-step integration is used, the GTE along each path is roughly the same so that differencing the final values of the states cancels the truncation error. Variable-step integration does not control the truncation error; it just keeps it below a certain tolerance. Hence, if something should cause the step pattern on the perturbed path to be substantially different from that on the nominal path, the global truncation errors on the two paths would be different. Then, the difference used in computing the numerical derivative would not cancel the error, and the derivative would be less accurate.

In the optimization literature, this conversion method is beginning to be called the direct shooting method because once the guess is made the differential equations are integrated from t_0 to t_f in one pass.

Conversion with b, u_k, x_j as Unknowns

The method defined in the preceding section is characterized by a single integration over the time interval $[t_0, t_f]$. If the time interval is very long, the accuracy of the integration is affected, and the accuracy of the numerical derivatives for a given perturbation size is even worse. One way to reduce this problem is to guess the value of the state x at the end of every ν intervals ($N - 1$ must be an integer multiple of ν) and to start a new integration. Then, as a part of the optimization process, the difference of the computed and guessed values of x at each of these nodes is driven to zero. This method could be called direct multiple shooting.

For this method, the unknown parameter vector (15) becomes

$$X = \begin{bmatrix} b^T & u_1^T & \cdots & u_N^T & x_{\nu+1}^T & x_{2\nu+1}^T & \cdots & x_{N-\nu}^T \end{bmatrix}^T \tag{20}$$

so that there are now $l + mN + n\{[(N-1)/\nu] - 1\}$ unknown parameters. If $\bar{x}$ denotes the computed value and

$$R = x - \bar{x} \tag{21}$$

denotes the difference between the guessed and the computed values of x, the equality constraint (18) becomes the following:

$$C(X) = \begin{bmatrix} \psi[x_f(X), b] \\ R_1(x_{\nu+1}) \\ R_2(x_{2\nu+1}) \\ \vdots \\ R_{[(N-1)/\nu]-1}(x_{N-\nu}) \end{bmatrix} = 0 \tag{22}$$

and contains $p + n\{[(N-1)/\nu] - 1\}$ elements. There is no change in $F(X)$ or $D(X)$.

In the limit, the value of x can be guessed at every node, with explicit integration taking place between the nodes. If the nodes are sufficiently close together, a single integration step can be taken. In this case, it is possible to use implicit integration, which has more accuracy per function evaluation than explicit integration.

Conversion with b, u_k, x_k as Unknowns

For this conversion method,[2–4] the design parameter b, the control parameters $u_1, \ldots, u_N$, and the state parameters $x_2, \ldots, x_N$, are the unknowns. Note that $x_1 = x_0$ from the prescribed initial conditions (3). Again, the symbol z_k denotes the vector z at the node t_k. The vector of unknown parameters is given by

$$X = \begin{bmatrix} b^T & u_1^T & \cdots & u_N^T & x_2^T & \cdots & x_N^T \end{bmatrix}^T \tag{23}$$

and has $l + mN + n(N-1)$ elements. Whereas b and u_k are normal optimization variables, $n(N-1)$ additional constraints must be imposed to compute $x_2, \ldots, x_N$.

Implicit integration of Eq. (2) is performed by calculating the residuals (one for each interval) and driving them to zero as a part of the optimization process. For the second-order midpoint rule, which has one function evaluation at t_m, the integration formula is applied in the form

$$R_k = x_{k+1} - x_k - g_m(t_{k+1} - t_k) \tag{24}$$

where

$$g_m = g(t_m, x_m, u_m, b) \tag{25}$$

and

$$t_m = \frac{t_k + t_{k+1}}{2}, \qquad x_m = \frac{x_k + x_{k+1}}{2}, \qquad u_m = \frac{u_k + u_{k+1}}{2} \tag{26}$$

For the second-order trapezoid rule, which uses values of g at t_k and t_{k+1}, the residual is given by

$$R_k = x_{k+1} - x_k - \tfrac{1}{2}(g_k + g_{k+1})(t_{k+1} - t_k) \tag{27}$$

For the fourth-order Simpson one-third rule, which uses the function evaluations at t_k, t_m, and t_{k+1}, the residual is

$$R_k = x_{k+1} - x_k - \tfrac{1}{6}(g_k + 4g_m + g_{k+1})(t_{k+1} - t_k) \tag{28}$$

where g_m is evaluated at

$$t_m = \frac{t_k + t_{k+1}}{2}, \qquad u_m = \frac{u_k + u_{k+1}}{2}$$
$$x_m = \frac{x_k + x_{k+1}}{2} - \tfrac{1}{8}(g_{k+1} - g_k)(t_{k+1} - t_k) \tag{29}$$

The use of higher-order rules is discussed elsewhere.[10]

Note that, when u_m is required, it has been computed by linear interpolation of the node values. In the development of a numerical integrator, it is assumed that $u(t)$ is known so that the way in which u_m is computed does not affect the order of the integrator. However, it may affect the overall accuracy of the integration. Hence, more accurate interpolation can be used, or u_m can be made an optimization parameter. For the midpoint rule, the u_m would be the parameters instead of the u_k.

Regardless of the integration method used, the parameter optimization problem is the same. The performance index and inequality constraint function are given by

$$F(X) = \phi(x_N, b), \qquad D(X) = \theta(x_N, b) \tag{30}$$

while the equality constraint function becomes

$$C(X) = \begin{bmatrix} \psi(x_N, b) \\ R_1(b, u_1, u_2, x_2) \\ R_2(b, u_2, u_3, x_2, x_3) \\ \vdots \\ R_{N-1}(b, u_{N-1}, u_N, x_{N-1}, x_N) \end{bmatrix} \tag{31}$$

and now contains $p + n(N-1)$ elements.

In the optimization literature, the Simpson one-third rule is used in collocation.[2] The use of any rule is called direct transcription.[3,4] The use of higher-order rules[10] is called collocation. Hence, what is being called collocation and what is being called direct transcription are the same thing.

Conversion with b, x_k as Unknowns

If the controls are eliminated from the optimization problem,[5] the unknown parameter vector becomes

$$X = \begin{bmatrix} b^T & x_2^T & \cdots & x_N^T \end{bmatrix}^T \tag{32}$$

and contains only $l + n(N-1)$ elements. Here, m of the n state equations

$$\dot{x} = g(t, x, u, b) \tag{33}$$

are solved for the controls, which are then eliminated from the remaining equations. This process results in $n - m$ constraints of the form

$$F(t, x, \dot{x}, b) = 0 \tag{34}$$

These constraints can be imposed at the midpoint of each interval as

$$R_k = F(t_m, x_m, \dot{x}_m, b) = 0 \tag{35}$$

where

$$t_m = \frac{t_k + t_{k+1}}{2}, \qquad x_m = \frac{x_k + x_{k+1}}{2}, \qquad \dot{x}_m = \frac{x_{k+1} - x_k}{t_{k+1} - t_k} \tag{36}$$

Upon convergence, the integration has been performed by the midpoint rule as can be seen from the expression for $\dot{x}_m$. The use of higher-order integrators has yet to be discussed in the literature.

Relative to the parameter optimization problem, $F(X)$ and $D(X)$ are the same as those for the previous method, but the equality constraint function becomes

$$C(X) = \begin{bmatrix} \psi(x_N, b) \\ R_1(b, x_2) \\ R_2(b, x_2, x_3) \\ \vdots \\ R_{N-1}(b, x_{N-1}, x_N) \end{bmatrix} \tag{37}$$

and has $p + (n-m)(N-1)$ elements. Once the design and state parameters are known, the control parameters are obtained from their defining equations. In the optimization literature, this method is called differential inclusion.

To form Eq. (35), it may not be possible to eliminate the control analytically. However, if the control must be eliminated numerically, the attractiveness of this method is reduced.

Conclusions

In recent years, several methods have been proposed for converting optimal control problems into parameter optimization problems. Because these methods have a lot of similarities not immediately obvious from reading the papers, the purpose of this paper has been to summarize these methods and to categorize them in terms of the choice of the unknowns (control and/or state parameters), the numerical integration technique (explicit or implicit) used to integrate the state differential equation, and the order of the numerical integration technique. Four general classes of methods exist, depending on the selection of the unknowns: 1) control parameters, 2) control parameters and some state parameters, 3) control parameters and state parameters, and 4) state parameters. In the methods of the first two classes, the state differential equations are integrated explicitly. In the remaining two classes, the equations are integrated implicitly because the state parameters are available. Finally, all orders of numerical integration can be used in each class of methods.

References

[1]Hull, D. G., and Speyer, J. L., "Optimal Reentry and Plane-Chang Trajectories," *Journal of the Astronautical Sciences*, Vol. 30, No. 2, 1982, pp. 117–130.

[2]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.

[3]Enright, P. J., and Conway, B. A., "Discrete Approximations to Optimal Trajectories Using Direct Transcription and Nonlinear Programming," *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 4, 1992, pp. 994–1001.

[4]Betts, J. T., "Using Sparse Nonlinear Programming to Compute Low Thrust Orbit Transfers," *Journal of the Astronautical Sciences*, Vol. 41, No. 3, 1993, pp. 349–371.

[5]Seywald, H., "Trajectory Optimization Based on Differential Inclusion," *Journal of Guidance, Control, and Dynamics*, Vol. 17, No. 3, 1994, pp. 480–487.

[6]Hoffman, J. D., *Numerical Methods for Engineers and Scientists*, McGraw–Hill, New York, 1992.

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

[8]Butcher, J. C., "Implicit Runge–Kutta Processes," *Mathematics of Computation*, Vol. 18, No. 85, 1964, pp. 50–64.

[9]Hempel, P., "Application of Implicit Runge–Kutta Methods," AIAA Paper 76-818, Aug. 1976.

[10]Herman, A. L., and Conway, B. A., "Direct Optimization Using Collocation Based on High-Order Gauss–Lobatto Quadrature Rules," *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 3, 1996, pp. 592–599.