Chapter 1
Trajectory Optimization: A Survey

Anil V. Rao

Abstract A survey of numerical methods for trajectory optimization. The goal of this survey is to describe typical methods that have been developed over the years for optimal trajectory generation. In addition, this survey describes modern software tools that have been developed for solving trajectory optimization problems. Finally, a discussion is given on how to choose a method.

1.1 Introduction

Trajectory optimization is a process where it is desired to determine the path and the corresponding input (control) to a dynamical system that meet specified constraints on the system while optimizing a specified performance index. Typically, optimal trajectory generation is performed *off-line*, that is, such problems are not solved in real time nor in a closed-loop manner. Because of the complexity of most applications, optimal trajectories are typically generated using numerical methods. Numerical for trajectory optimization date back nearly five decades to the 1950s with the work of Bellman [5–10]. Because complexity of modern applications has has increased tremendously as compared to applications of the past, methods for trajectory optimization continue to evolve and the discipline is becoming increasingly relevant in a wide range of subject including virtually all branches of engineering, economics, and medicine.

Numerical methods for trajectory optimization are divided into two major classes: indirect methods and direct methods. In an indirect method, the first-order optimality conditions from variational calculus are employed. The trajectory optimization problem is then converted into a multiple-point Hamiltonian boundary-value problem. The HBVP is then solved numerically to determine candidate optimal trajectories

A. V. Rao (✉)
University of Florida, Gainesville, FL, USA
e-mail: anilvrao@ufl.edu

H. Waschl et al. (eds.), *Optimization and Optimal Control in Automotive Systems*,
Lecture Notes in Control and Information Sciences 455,
DOI: 10.1007/978-3-319-05371-4_1,

called *extremals*. Each extremal solution of the HBVP is then examined to see if it is a local minimum, maximum, or a saddle point, and the extremal with the lowest cost is chosen. In a direct method, the state and/or control of the original trajectory optimization problem is approximated by parameterizing the state and/or the control and the trajectory optimization problem is transcribed to a finite-dimensional nonlinear programming problem (NLP). The NLP is then solved using well known optimization techniques.

It is seen that indirect methods and direct method emanate from two different philosophies. On the one hand, the indirect approach solves the problem indirectly (thus the name, *indirect*) by converting the trajectory optimization problem to a boundary-value problem. As a result, in an indirect method the optimal solution is found by solving a system of differential equations that satisfies endpoint and/or interior point conditions. On the other hand, in a direct method the optimal solution is found by transcribing an infinite-dimensional optimization problem to a finite-dimensional optimization problem.

The two different philosophies of indirect and direct methods have led to a dichotomy in the trajectory optimization community. Researchers who focus on indirect methods are interested largely in the numerical solution of differential equations, while researchers who focus on direct methods are interested primarily in the numerical solution of optimization problems. While at first glance these two approaches may seem completely unrelated, they have a great deal in common. As will be described in the survey, recent years researchers have delved quite deeply into the connections between the indirect and direct forms. This research has uncovered that the optimality conditions from many direct methods have a well-defined meaningful relationship. Thus, these two classes of methods are merging as time goes by.

1.2 Trajectory Optimization Problem

A fairly general trajectory optimization problem is posed formally as follows. Typically, the problem is divided into P *phases* [15] and the phases are connected in some meaningful way. A multiple-phase trajectory optimization problem is posed as follows. Optimize the cost functional

$$J = \sum_{k=1}^{P} \left[\Phi^{(k)} \left[\mathbf{y}^{(k)}(t_0), t_0, \mathbf{y}^{(k)}\left(t_f\right), t_f; \mathbf{s} \right] + \int_{t_0^{(k)}}^{t_f^{(k)}} \mathscr{L}\left[\mathbf{y}^{(k)}\left(t\right), \mathbf{u}^{(k)}\left(t\right), t; \mathbf{s}^{(k)} \right] dt \right] \tag{1.1}$$

subject to the dynamic constraints

$$\dot{\mathbf{y}}^{(k)}\left(t\right) = \mathbf{f}\left(\mathbf{y}^{(k)}\left(t\right), \mathbf{u}^{(k)}\left(t\right), t; \mathbf{s}^{(k)} \right), \tag{1.2}$$

the *boundary conditions*,

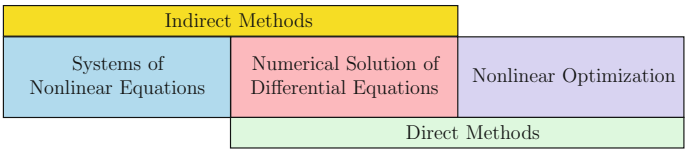

Fig. 1.1 The three major components of trajectory optimization and the class of methods that uses each component

$$\phi_{\min}^{(k)} \leq \phi^{(k)}\left(\mathbf{y}^{(k)}(t_0^{(k)}), t_0^{(k)}, \mathbf{y}^{(k)}(t_f^{(k)}), \mathbf{s}^{(k)}, t_f^{(k)}\right) \leq \phi_{\max}^{(k)}, \tag{1.3}$$

the algebraic *path constraints*

$$\mathbf{c}_{\min}^{(k)} \leq \mathbf{c}^{(k)}\left(\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{s}^{(k)}, t\right) \leq \mathbf{c}_{\max}^{(k)} \tag{1.4}$$

and the *linkage constraints* (also known as *phase continuity constraints*)

$$\mathbf{L}_{\min}^{(s)} \leq \mathbf{L}\left(\mathbf{y}^{(l_s)}\left(t_f^{(l_s)}\right), \mathbf{u}^{(l_s)}\left(t_f^{(l_s)}\right), \mathbf{s}^{(l_s)}, t_f^{(l_s)}, \mathbf{y}^{(r_s)}(t_f^{(r_s)}), \mathbf{u}^{(r_s)}\left(t_f^{(r_s)}\right), \mathbf{s}^{(r_s)}, t_f^{(r_s)}\right) \leq \mathbf{L}_{\max}^{(s)}, \tag{1.5}$$

where $s \in [1, \ldots, S]$ and S is the number of pairs of phases that are being linked. In Eq. (1.5) the parameter S is the number of pairs of phases to be linked, $r_s \in [1, \ldots, S]$ and $l_s \in [1, \ldots, S]$ are the *right phases* and *left phases*, respectively, of the linkage pairs, $r_s \neq l_s$ (implying that a phase cannot be linked to itself), and $s \in [1, \ldots, S]$.

1.3 Numerical Methods for Trajectory Optimization

At the heart of a well-founded method for solving trajectory optimization problems are the following three fundamental components: (1) a method for solving differential equations and integrating functions; (2) a method for solving a system of nonlinear algebraic equations; and (3) a method for solving a nonlinear optimization problem. Methods for solving differential equations and integrating functions are required for all numerical methods in trajectory optimization optimal control. In an indirect method, the numerical solution of differential equations is combined with the numerical solution of systems of nonlinear equations while in a direct method the numerical solution of differential equations is combined with nonlinear optimization. A schematic with the breakdown of the components used by each class of optimal control methods is shown in Fig. 1.1.

1.4 Numerical Solution of Differential Equations

Consider the *initial-value problem* [30, 42, 92] (IVP)

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}(t), t), \qquad \mathbf{y}(t_0) = \mathbf{y}_0. \tag{1.6}$$

Next, let $[t_i, t_{i+1}]$ be a time interval over which the solution to Eq. (1.6) is desired. Integrating Eq. (1.6), we can write

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \int_{t_i}^{t_{i+1}} \dot{\mathbf{x}}(s)ds = \mathbf{y}_i + \int_{t_i}^{t_{i+1}} \mathbf{f}(\mathbf{y}(s), s)ds. \tag{1.7}$$

The two most common approaches for solving differential equations are time marching and collocation. In a time-marching method, the solution of the differential equation at each time step t_k is obtained sequentially using current and/or previous information about the solution. In a *multiple-step time marching method*, the solution at time t_{k+1} is obtained from a defined set of previous values $t_{k-j}, \ldots, t_k$ where j is the number of steps. The simplest multiple-step method is a *single-step method* (where $j = 1$). The most common single-step methods are *Euler methods*, while most commonly used multiple-step methods are the *Adams-Bashforth* and *Adams-Moulton* multiple-step methods [30]. Euler backward and Crank-Nicolson are examples of *implicit methods* whereas Euler forward is an example of an *explicit method*. When employing implicit method, the solution at t_{k+1} is obtained using a *predictor-corrector* where the predictor is typically an explicit method (that is, Euler-forward) while the corrector is the implicit formula. Implicit methods methods are more stable than explicit methods [42], but an implicit method requires more computation at each step due to the need to implement a predictor-corrector.

An alternative to a multiple-step time marching method is a *multiple-stage* method. In a multiple-stage method, the interval $[t_i, t_{i+1}]$ into K subintervals $[\tau_j, \tau_{j+1}]$ where

$$\tau_j = t_i + h_i\alpha_j, \quad (j = 1, \ldots, K), \quad h_i = t_{i+1} - t_i, \tag{1.8}$$

and $0 \leq \alpha_j \leq 1$, $(j = 1, \ldots, K)$. Each value τ_j is called a *stage* . The integral from t_i to t_{i+1} can be approximated via *quadrature* as

$$\int_{t_i}^{t_{i+1}} \mathbf{f}(\mathbf{y}(s), s)ds \approx h_i \sum_{j=1}^{K} \beta_j \mathbf{f}(\mathbf{y}_j, \tau_j) \tag{1.9}$$

where $\mathbf{y}_j \equiv \mathbf{y}(\tau_j)$. It is seen in Eq. (1.9) that the values of the state at each stage are required in order to evaluate the quadrature approximation. These intermediate values are obtained as

$$\mathbf{y}(\tau_j) - \mathbf{y}(t_i) = \int_{t_i}^{\tau_j} \mathbf{f}(\mathbf{y}(s), s)ds \approx h_i \sum_{l=1}^{K} \gamma_{jl}\mathbf{f}(\mathbf{y}_l, \tau_l) \tag{1.10}$$

The combination of Eqs. (1.9) and (1.10) leads to the family of $K-$stage *Runge-Kutta* methods [15, 24, 25, 30, 50, 51, 92]. A Runge-Kutta method is called *explicit* if $\gamma_{jl} = 0$ for all $l \geq j$ and is called *implicit* otherwise. In an explicit Runge-Kutta method, the approximation at t_{k+1} is computed using information prior to t_{k+1} whereas in an implicit Runge-Kutta method $\mathbf{y}(t_{k+1})$ is required in order to determine the solution at t_{k+1}. In the latter case, the solution is updated using a predictor-corrector approach.

1.4.1 Collocation

Another way to solve differential equations is as follows. Suppose over a subinterval $[t_i, t_{i+1}]$ we choose to approximate the state using the following Kth-degree piecewise polynomial:

$$\mathbf{Y}(t) \approx \sum_{k=0}^{K} \mathbf{a}_i (t - t_i)^k, \quad t \in [t_i, t_{i+1}]. \tag{1.11}$$

Suppose further that the coefficients $(\mathbf{a}_0, \ldots, \mathbf{a}_K)$ of the piecewise polynomial are chosen to match the value of the function at the beginning of the step, that is,

$$\mathbf{Y}(t_i) = \mathbf{y}_i. \tag{1.12}$$

Finally, suppose we choose to match the derivative of the state at the points defined in Eq. (1.8), that is,

$$\dot{\mathbf{y}}(\tau_j) = \mathbf{f}(\mathbf{y}(\tau_j), \tau_j), \quad (j = 1, \ldots, K). \tag{1.13}$$

Equation (1.13) is called a *collocation condition* because the approximation to the derivative is set equal to the right-hand side of the differential equation evaluated at each of the intermediate points $(\tau_1, \ldots, \tau_K)$. Collocation methods fall into three general categories [15]: Gauss methods, Radau methods, and Lobatto methods. In a *Gauss method*, neither of the endpoints t_k or t_{k+1} are collocation points. In a *Radau method*, at most one of the endpoints t_k or t_{k+1} is a collocation point. In a *Lobatto method*, both of the endpoints t_k and t_{k+1} are collocation points.

As it turns out, Euler and Runge-Kutta methods can be thought of equivalently as either time-marching or collocation methods. When an Euler or a Runge-Kutta method is employed in the form of collocation, the differential equation is said to be solved *simultaneously* because all of the unknown parameters are determined at

the same time. Furthermore, collocation methods are said to simulate the dynamics of the system *implicitly* because the values of the state at each collocation point are obtained at the same time (as opposed to solving for the state sequentially as in a time-marching method). In order to implement simultaneous simulation, the discretized dynamics are written as *defect constraints* of the form

$$\boldsymbol{\zeta}_j = \dot{\mathbf{y}}(\tau_j) - \mathbf{f}(\mathbf{y}(\tau_j), \tau_j). \tag{1.14}$$

As an example, the defect constraints for the Crank-Nicolson method are given as

$$\boldsymbol{\zeta}_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2}\left(\mathbf{f}_k + \mathbf{f}_{k+1}\right). \tag{1.15}$$

In collocation (that is, implicit simulation) it is desired to find a solution such that all of the defect constraints are zero. Finally, one of the key differences between collocation and time-marching is that in collocation it is not necessary to use a predictor-corrector because the values of the state at each discretization point are being solve for simultaneously.

1.4.2 Integration of Functions

Because the objective is to solve a trajectory optimization problem, it is necessary to approximate the cost function of Eq. (1.1). Typically, the cost is approximated using a quadrature that is consistent with the numerical method for solving the differential equation (for example, if one is using an Euler-forward rule for solving the differential equation, the cost would also be approximated using Euler-forward integration). The requirement for consistency in the approximation of the differential equations and the cost can be thought of in another manner. Consider a one-phase trajectory optimization problem. The cost functional

$$J = \Phi(\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f) + \int_{t_0}^{t_f} \mathscr{L}[\mathbf{y}(t), \mathbf{u}(t), t; \mathbf{s}]dt \tag{1.16}$$

can be converted to a Mayer problem by adding a state y_{n+1} and adding the differential equation

$$\dot{y}_{n+1} = g[\mathbf{y}(t), \mathbf{u}(t), t; \mathbf{s}] \tag{1.17}$$

with the initial condition

$$y_{n+1}(t_0) = 0. \tag{1.18}$$

The cost functional of Eq. (1.16) would be given as

$$J = \Phi[\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f; \mathbf{s}] + y_{n+1}(t_f) \tag{1.19}$$

and the resulting augmented system of differential equations would then be written as

$$\begin{aligned}\dot{\mathbf{y}}(t) &= \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t; \mathbf{s}],\\ \dot{y}_{n+1} &= \mathscr{L}[\mathbf{x}(t), \mathbf{u}(t), t; \mathbf{s}].\end{aligned} \tag{1.20}$$

Equation (1.20) could then be solved using any well established numerical integration method. Using this approach, is it seen that the method used to integrate Eq. (1.17) must be the same method that is used to integrate

$$\mathscr{L}[\mathbf{y}(t), \mathbf{u}(t), t; \mathbf{s}].$$

1.5 Nonlinear Optimization

A key ingredient to solving trajectory optimization problems is the ability to solve *nonlinear optimization* or *nonlinear programming problems* [4, 11, 21, 44] (NLPs). An NLP takes the following general mathematical form. Determine the decision vector $\mathbf{z} \in \mathbb{R}^n$ that minimizes the cost function

$$f(\mathbf{z}) \tag{1.21}$$

subject to the algebraic constraints

$$\mathbf{g}(\mathbf{z}) = \mathbf{0}, \tag{1.22}$$

$$\mathbf{h}(\mathbf{z}) \leq \mathbf{0}, \tag{1.23}$$

where $\mathbf{g}(\mathbf{z}) \in \mathbb{R}^m$ and $\mathbf{h}(\mathbf{z}) \in \mathbb{R}^p$. The NLP may either be *dense* (that is, a large percentage of the derivatives of the objective function and the constraint functions with respect to the components of $\mathbf{z}$ are nonzero) or may be *sparse* (that is, a large percentage of the derivatives of the objective function and the constraint functions with respect to the components of $\mathbf{z}$ are zero). Dense NLPs typically are small (consisting of at most a few hundred variables and constraints) while sparse NLPs are often extremely large (ranging anywhere from thousands of variables and constraints to millions of variables and constraints).

1.6 Methods for Solving Trajectory Optimization Problems

With the exception of simple problems, trajectory optimization problems must be solved numerically. The need for solving optimal control problems numerically has given rise to a wide range of numerical approaches. These numerical approaches are

divided into two broad categories: (1) indirect methods and (2) direct methods. The major methods that fall into each of these two broad categories are described in the next two sections.

1.6.1 Indirect Methods

In an indirect method, the *calculus of variations* [3, 20, 23, 40, 56, 62, 67, 68, 91, 94, 97] is used to determine the first-order optimality conditions of the trajectory optimization problem given in Eqs. (1.1)–(1.5). Unlike ordinary calculus (where the objective is to determine points that optimize a function), the calculus of variations is the subject of determining *functions* that optimize a *function of a function* (also known as *functional optimization*). Applying the calculus of variations to the functional optimization problem given in Eqs. (1.1)–(1.5) leads to the *first-order necessary conditions* for an extremal trajectory. The first-order optimality conditions for a single-phase continuous-time trajectory optimization problem with no static parameters are given as

$$\dot{\mathbf{y}} = \mathscr{H}_{\boldsymbol{\lambda}}^{\mathsf{T}}, \quad \dot{\boldsymbol{\lambda}} = -\mathscr{H}_{\mathbf{y}}^{\mathsf{T}}, \tag{1.24}$$

$$\mathbf{u}^* = \arg\min_{\mathbf{u}\in\mathscr{U}} \mathscr{H}, \tag{1.25}$$

$$\phi(\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f) = \mathbf{0}, \tag{1.26}$$

$$\lambda(t_0) = -\Phi_{\mathbf{y}}(t_0) + \boldsymbol{\nu}^{\mathsf{T}}\phi_{\mathbf{y}(t_0)}, \quad \lambda(t_f) = \Phi_{\mathbf{y}}(t_f) - \boldsymbol{\nu}^{\mathsf{T}}\phi_{\mathbf{y}(t_f)}, \tag{1.27}$$

$$\mathscr{H}(t_0) = \Phi_{t_0} - \boldsymbol{\nu}^{\mathsf{T}}\phi_{t_0}, \quad \mathscr{H}(t_f) = -\Phi_{t_f} + \boldsymbol{\nu}^{\mathsf{T}}\phi_{t_f}, \tag{1.28}$$

$$\begin{aligned} &\mu_j(t) = 0, \text{ when } \quad C_j(\mathbf{x}, \mathbf{u}, t) < 0, \quad j = 1, \ldots, c, \\ &\mu_j(t) \leq 0, \text{ when } \quad C_j(\mathbf{x}, \mathbf{u}, t) = 0, \quad j = 1, \ldots, c, \end{aligned} \tag{1.29}$$

where $\mathscr{H} = \mathscr{L} + \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{f} - \mu^{\mathsf{T}}\mathbf{C}$ is the augmented Hamiltonian, $\mathscr{U}$ is the feasible control set and $\boldsymbol{\nu} \in \mathbb{R}^q$ is the Lagrange multiplier associated with the boundary condition ϕ. Finally, it is noted that the solution to the optimal control problem may lie along a singular arc [23] where the control cannot be determined from the first-order optimality conditions. If a singular arc is a possibility, additional conditions must be derived to determine the control along the singular arc.

Because the dynamics of Eq. (1.24) arise from differentiation a Hamiltonian, Eq. (1.24) is called a *Hamiltonian system* [3, 66, 67]. Furthermore, Eq. (1.25) is known as *Pontryagin's Minimum Principle* [75] (PMP) and is a classical result to determine theoptimal control. Finally, the conditions on the initial and final costate

given in Eq. (1.27) are called *transversality conditions* [3, 23, 40, 62, 66, 68, 91, 93, 94] while the conditions on the Lagrange multipliers of the path constraints given in Eq. (1.29) are called *complementary slackness conditions* [4, 11, 21]. The Hamiltonian system, together with the boundary conditions, transversality conditions, and complementary slackness conditions, is called a *Hamiltonian boundary-value problem* (HBVP) [2, 3, 66]. Any solution $(\mathbf{y}(t), \mathbf{u}(t), \lambda(t), \mu(t), \nu)$ is called an *extremal solution* and consists of the state, costate, and any Lagrange multipliers that satisfy the boundary conditions and any interior-point constraints on the state and costate. In an indirect method extremal trajectories (that is, solutions of the HBVP) are determined numerically. Because an indirect method requires solving a multiple-point boundary-value problem, the original trajectory optimization problem is turned into the problem of solving a system of nonlinear equations of the form

$$\begin{aligned} \mathbf{f}(\mathbf{z}) &= \mathbf{0}, \\ \mathbf{g}_{\min} \leq \mathbf{g}(\mathbf{z}) &\leq \mathbf{g}_{\max}. \end{aligned} \tag{1.30}$$

The three two most common indirect methods are the shooting method, the multiple-shooting method, and collocation methods. Each of these approaches is now described.

1.6.1.1 Indirect Shooting Method

Perhaps the most basic indirect method is the *shooting method* [65]. In a typical shooting method, an initial guess is made of the unknown boundary conditions at one end of the interval. Using this guess, together with the known initial conditions, the Hamiltonian system Eq. (1.24) is integrated to the other end (that is, either forward from t_0 to t_f or backward from t_f to t_0). Upon reaching t_f, the terminal conditions obtained from the numerical integration are compared to the known terminal conditions given in Eqs. (1.26) and (1.27). If the integrated terminal conditions differ from the known terminal conditions by more than a specified tolerance ε, the unknown initial conditions are adjusted and the process is repeated until the difference between the integrated terminal conditions and the required terminal conditions is less than some specified threshold.

1.6.1.2 Indirect Multiple-Shooting Method

While a simple shooting method is appealing due to its simplicity, it presents significant numerical difficulties due to ill-conditioning of the Hamiltonian dynamics. The reason for this ill-conditioning is that Hamiltonian systems have the property that the divergence of the flow of trajectories must be zero, that is

$$\sum_{i=1}^{n}\left[\frac{\partial}{\partial x_i}\left(\frac{\partial \mathscr{H}}{\partial \lambda_i}\right)+\frac{\partial}{\partial \lambda_i}\left(-\frac{\partial \mathscr{H}}{\partial x_i}\right)\right]\equiv 0. \tag{1.31}$$

Equation (1.31) implies that, in a neighborhood of the optimal solution, there exist an equal number of directions along which the solution will contract and expand and this expansion and contraction takes place at the same rate (the simultaneous expanding and contracting behavior is due to the fact that many Hamiltonian systems admit an exponential *dichotomy* [2]). As a result, errors made in the unknown boundary conditions will amplify as the dynamics are integrated in either direction of time. The shooting method poses particularly poor characteristics when the trajectory optimization problem is *hyper-sensitive* [76, 78, 79, 81, 82] (that is, when time interval of interest is long in comparison with the time-scales of the Hamiltonian system in a neighborhood of the optimal solution).

In order to overcome the numerical difficulties of the simple shooting method, a modified method, called the *multiple-shooting method* [92], has been developed. In a multiple-shooting method, the time interval $[t_0, t_f]$ is divided into $M+1$ subintervals. The shooting method is then applied over each subinterval $[t_i, t_{i+1}]$ with the initial values of the state and adjoint of the interior intervals being the unknowns that need to be determined. In order to enforce continuity, the following conditions are enforced at the interface of each subinterval:

$$\mathbf{p}(t_i^-)=\mathbf{p}(t_i^+) \Longleftrightarrow \mathbf{p}(t_i^-)-\mathbf{p}(t_i^+)=\mathbf{0}, \tag{1.32}$$

where $\mathbf{p}(t)$ is the combined state-costate vector, that is,

$$\mathbf{p}(t)=\begin{bmatrix}\mathbf{x}(t)\\ \lambda(t)\end{bmatrix}.$$

The continuity conditions of Eq. (1.32) result in vector root-finding problem where it is desired to drive the values of the difference between $\mathbf{p}(t_i^-)-\mathbf{p}(t_i^+)$ to zero. It is seen that the multiple-shooting method requires extra variables be introduced into the problem (that is, the values of the state and adjoint at the interface points). Despite the increased size of the problem due to these extra variables, the multiple-shooting method is an improvement over the shooting method because the sensitivity to errors in the unknown initial conditions is reduced by integrating over subintervals of the original time domain $t \in [t_0, t_f]$. Nevertheless, even multiple-shooting can present issues if a sufficiently good guess of the costate is not used [48].

1.6.1.3 Indirect Collocation Methods

In an indirect collocation method, the state and costate are parameterized using piecewise polynomials as described in Sect. 1.4.1. The collocation procedure leads to a root-finding problem where the vector of unknown coefficients $\mathbf{z}$ consists of the

coefficients of the piecewise polynomial. This system of nonlinear equations is then solved using a root-finding technique (for example, Newton's method).

1.6.2 Direct Methods

Direct methods are fundamentally different from indirect methods. In a direct method, the state and/or control of the original optimal control problem are approximated in some appropriate manner. In the case where only the control is approximated, the method is called a *control parameterization method* [46]. When both the state and control are approximated the method is called a *state and control parameterization method*. In either a control parameterization method or a state and control parameterization method, the optimal control problem is transcribed to a *nonlinear optimization problem* or *nonlinear programming problem* [4, 11, 15, 21, 44] (NLP).

1.6.2.1 Direct Shooting Method

The most basic direct method for solving trajectory optimization problems is the *direct shooting method*. The direct shooting method is a control parameterization method where the control is parameterized using a specified functional form. For example, the control could be parameterized as

$$\mathbf{u}(t) \approx \sum_{i=1}^{m} \mathbf{a}_i \psi_i(t), \tag{1.33}$$

where $\psi_i(t),\ (i = 1, \ldots, m)$ are known functions and $\mathbf{a}_i,\ (i = 1, \ldots, m)$ are the parameters to be determined from the optimization. The dynamics are then satisfied by integrating the differential equations using a time-marching algorithm. Similarly, the cost function of Eq. (1.1) is determined using a quadrature approximation that is consistent with the numerical integrator used to solve the differential equations. The NLP that arises from direct shooting then minimizes the cost subject to any path and interior-point constraints.

1.6.2.2 Direct Multiple-Shooting Method

In a manner similar to that for indirect methods, in a *direct multiple-shooting method*, the time interval $[t_0, t_f]$ is divided into $M+1$ subintervals. The aforementioned direct shooting method is then used over each subinterval $[t_i, t_{i+1}]$ with the values of the state at the beginning of each subinterval and the unknown coefficients in the control parameterization being unknowns in the optimization. In order to enforce continuity, the following conditions are enforced at the interface of each subinterval:

$$\mathbf{x}(t_i^-) = \mathbf{x}(t_i^+) \Longleftrightarrow \mathbf{x}(t_i^-) - \mathbf{x}(t_i^+) = \mathbf{0}. \tag{1.34}$$

The continuity conditions of Eq. (1.34) result in vector root-finding problem where it is desired to drive the values of the difference between $\mathbf{x}(t_i^-) - \mathbf{x}(t_i^+)$ to zero. It is seen that the direct multiple-shooting method increases the size of the optimization problem because the values of the state at the beginning of each subinterval are parameters in the optimization. As with indirect multiple-shooting, the direct multiple-shooting method is an improvement over the direct shooting method because the sensitivity to errors in the unknown initial conditions is reduced by integrating over subintervals of the original time domain $t \in [t_0, t_f]$.

1.6.2.3 Direct Collocation Methods

Arguably the most powerful methods for solving general trajectory optimization problems are *direct collocation methods*. A direct collocation method is a state and control parameterization method where the state and control are approximated using a specified functional form. The two most common forms of collocation are *local collocation* and *global collocation*. A *local collocation method* follows a procedure similar to that of Sect. 1.4.1 in that the time interval $[t_0, t_f]$ is divided into S subintervals $[t_{s-1}, t_s]$, $(s = 1, \ldots, S)$ where $t_S = t_f$. In order to ensure continuity in the state across subintervals, the following *compatibility constraint* is enforced at the interface of each subinterval:

$$\mathbf{y}(t_i^-) = \mathbf{y}(t_i^+), \quad (s = 2, \ldots, S-1). \tag{1.35}$$

In the context of trajectory optimization, local collocation has been employed using one of two categories of discretization: Runge-Kutta methods and orthogonal collocation methods. Nearly all Runge-Kutta methods used are *implicit* [31–36, 49, 69, 88, 89] because the stability properties of implicit Runge-Kutta methods are better than those of explicit methods. The seminal work on orthogonal collocation methods in trajectory optimization is due to Reddien [84], where Legendre-Gauss points were used together with cubic splines. Following on Reddien's work, Cuthrell and Biegler used LG points together with Lagrange polynomials [28, 29]. Interestingly, Cuthrell [29] showed mathematically that the indirect transcription using LG points was equivalent to the KKT conditions obtained from the NLP of the direct formulation. In the 1990s, orthogonal collocation methods were developed using higher-order Gauss-Lobatto collocation methods [38, 39, 54, 55]. Finally, the convergence rates of an orthogonal collocation method using Legendre-Gauss-Radau (LGR) points was studied [64].

Generally, employing direct local collocation leads to a *large sparse* NLP, where the NLP contains potentially thousands to hundreds of thousands of variables and constraints. Moreover, such large NLPs arise from trajectory optimization problems that consist of hundreds of of states and controls. Because the NLP is sparse, however, many of the derivatives of the constraint Jacobian are zero. This feature of local direct

collocation makes it possible to solve such problem efficiently using appropriate NLP solvers such as *SNOPT* [43, 45], *SPRNLP* [17], and *KNITRO* [26].

1.7 Software for Solving Trajectory Optimization Problems

A wide variety of software tools have been developed for solving trajectory optimization problems. Most of these software programs use direct methods. One well known software program employing indirect methods is *BNDSCO* [71] which employs a multiple-shooting method. Perhaps the oldest software tool that employs direct methods is the *Program to Simulate and Optimize Trajectories* [22] (*POST*). *POST* was originally developed to solve problems in launch vehicle trajectory optimization and it still in use today for such applications.

The late 1980s saw a transformation in the available tools for solving trajectory optimization problems. This transformation was coincident with the observation of the power of direct collocation methods. The first well-known direct collocation software was *Optimal Trajectories by Implicit Simulation* [95] (*OTIS*). *OTIS* is a FORTRAN software that has general-purpose capabilities for problems in aeronautics and astronautics. OTIS has been used widely in the aerospace and defense industries and its theoretical foundations are found in Ref. [52]. Following shortly after the development of *OTIS* is the program *Sparse Optimal Control Software* [18] (*SOCS*). *SOCS* is a highly powerful FORTRAN software that is capable of solving many highly challenging trajectory optimization problems (see Ref. [15] for highly complex optimal control problems solved with *SOCS*). Some of the applications solved using *SOCS* are found in Refs. [12–14, 16, 77]. Finally, three other direct collocation FORTRAN programs are *MISER* [47], *Direct Collocation* [96] (*DIRCOL*), *Graphical Environment for Simulation and Optimization* [1] (*GESOP*), and *Nonlinear Trajectory Generation* [70] (*NTG*). Like *OTIS* and *SOCS*, *DIRCOL* and *GESOP* use local direct collocation techniques while *NTG* is designed for rapid trajectory generation of differentially flat systems.

In recent years, interest in the particular application of optimal control to space flight has led to the development of several useful programs. One such program is *Mission Design and Analysis Software* [87] (*MIDAS*) which is designed to solve complex ballistic heliocentric transfer trajectories for interplanetary space flight missions. Another tool that has been recently developed is the NASA *Generalized Mission Analysis Tool* [61] (*GMAT*). Another tool that has been widely used in the last several years is *COPERNICUS* [72, 73]. Both *GMAT* and *COPERNICUS* are designed to solve trajectory optimization problems where the maneuvers can be treated as either impulsive or finite-thrust burns.

While earlier software programs used compiled languages such as FORTRAN, in recent years, MATLAB® has become increasingly popular for solving optimization problems. The increased appeal for MATLAB emanates from the fact that MATLAB is an extremely easy environment in which to program along with the fact that many of today's most powerful NLP solvers are now available for use in MATLAB® (for

example, standalone MATLAB mex versions are now available for the NLP solvers *SNOPT* [43, 45] and *KNITRO* [26]). In addition, the TOMLAB [19, 37, 57–60] package has facilitated additional solvers for use in MATLAB. In addition, because of major computational improvements, the computational efficiency between MATLAB and compiled languages is growing ever closer. Examples of MATLAB-based trajectory optimization software programs include RIOTS_95 [90], *DIDO* [85], *DIRECT* [98], *PROPT* [86], *OPTCONTROLCENTRE* [63], *GPOPS* [80], and *GPOPS-II* [74].

It is important to note that all of the trajectory optimization software programs described above incorporate gradient methods for solving the NLP. In a less formal manner, heuristic methods have also been used to solve trajectory optimization problems. For example, interplanetary trajectory optimization problems using a genetic algorithm have been considered in Refs. [41, 53] while low-thrust orbit transfers using a genetic algorithm have been studied in Refs. [27] and [83]. In addition, a calculus of variations technique has been used together with a genetic algorithm to optimize low-thrust Mars-to-Earth trajectories for the Mars Sample Return Mission [99]. Thus, while gradient methods are somewhat the de facto standard for trajectory optimization, the aforementioned research demonstrates that genetic algorithms may be well-suited for some applications.

1.8 Choosing a Method

Choosing a method for solving a trajectory optimization problem is based largely on the type of problem to be solved and the amount of time that can be invested in coding. An indirect shooting method has the advantage that it is simple to understand and produces highly accurate solutions when it converges. Unfortunately, indirect shooting is extremely sensitive to the unknown boundary conditions. In addition, indirect shooting requires the derivation of the first-order optimality conditions of the trajectory optimization problem [see Eqs. (1.24)–(1.29]! While for simple problems it may be possible to derive the first-order optimality conditions, deriving such conditions for complex optimal control problems is tedious, error-prone, and sometimes impossible (for example, problem with table lookups). Furthermore, the need to derive the optimality conditions makes implementing indirect shooting difficult in a general-purpose software program. For example, if it was required to derive first-order optimality conditions, a program such as *POST* would become nearly impossible to use because every new problem would require the derivation of these conditions! A multiple-shooting method overcomes some of the numerical difficulties of standard shooting, but does not avoid the issue of having to derive the optimality conditions.

The accuracy and robustness of a direct method is highly dependent upon the form of direct method used. Direct shooting methods are very good for problems where the control can be parameterized in a simple manner (for example, piecewise linear functions of time) and the problem can be characterized accurately using a small number optimization parameters. Software programs such as *POST* perform well

on launch vehicle ascent trajectories because these problems can be approximated accurately using simple control parameterizations. As the complexity of the problem increases, it becomes more and more apparent that the workhorse for solving trajectory optimization problems is the direct collocation method. The two main reasons that direct collocation methods work so well is because highly complex problems can be formulated and solved with today's NLP solvers. The reason that the NLP solvers can handle such complex problems is because they are designed to converge with poor initial guesses (for example, straight line guesses in the state and control) and are extremely computationally efficient because they exploit the sparsity of the derivatives in the constraints and objective function.

In many cases, the solution of a trajectory optimization problem is a means to an end, that is, the user does not want to know all of the details about a method, but simply wants to use a software program to provide results so that a particular problem of interest can be solved. If one does not wish to become an expert in the technologies associated with trajectory optimization, it is advisable to obtain a canned software package that allows a user to input the problem in an intuitive manner. Then the software can simply be run on the problem of interest. It is always important to understand, however, that canned software can have its issues when things go wrong because the user may often not understand why.

1.9 Applications to Automotive Systems

The numerical methods provided in this survey are designed to generate reference trajectories and corresponding reference controls for systems that have well developed deterministic models. In the context of automotive systems, the methods described in this paper would be of relevance to optimal control in systems where performance is important. For example, state-of-the-art direct collocation software such as GPOPS-II or SOCS could be employed to generate highly accurate trajectories to determine the minimum lap time required in a in high-speed race car problem (for example, Formula One racing). In addition, the indirect methods described in this paper could be the starting point for developing near-optimal feedback controllers for use in engine design or in autonomous ground vehicles. Thus, the numerical methods described in this survey could be used to generate solutions to a wide variety of problems in automotive systems, and the particular numerical method employed would depend upon the intended use of the solution.

1.10 Conclusions

A survey of numerical methods for solving trajectory optimization problems has been given. The problem of solving optimal control problems has been decomposed into the three key components of solving differential equations and integrating functions,

solving nonlinear optimization problems, and solving systems of nonlinear algebraic equations. Using these components, the two classes of indirect and direct methods for solving optimal control problems have been described. Subsequently, important computational issues have been discussed and several different software tools for solving optimal control problems have been described. Finally, a brief discussion has been given on how to choose a method.

References

1. GESOP & ASTOS (2003) The new generation of optimization software. Institute of Flight Mechanics and Control of Stuttgart University
2. Ascher UM, Mattheij RM, Russell RD (1996) Numerical solution of boundary-value problems in ordinary differential equations. SIAM Press, Philadelphia
3. Athans MA, Falb PL (2006) Optimal control: an introduction to the theory and its applications. Dover Publications, Mineola, New York
4. Bazaraa MS, Sherali HD, Shetty CM (2006) Nonlinear programming: theory and algorithms, 3 edn. Wiley-Interscience, New Jersey
5. Bellman R (1957) Dynamic programming. Princeton University Press, Princeton
6. Bellman R (1962) Dynamic programming treatment of the travelling salesman problem. J Assoc Comput Mach 9(1):61–63
7. Bellman R (1966) Dynamic programming. Science 1, 153(3731):34–37
8. Bellman R, Dreyfus S (1959) Functional approximations and dynamic programming. Math Tables Other Aids Comput 13(68):247–251
9. Bellman R, Kalaba R, Kotkin B (1963) Polynomial approximation—a new computational technique in dynamic programming: allocation processes. Math Comput 17(82):155–161
10. Bellman RE, Dreyfus SE (1971) Applied dynamic programming. Princeton University Press, Princeton
11. Bertsekas D (2004) Nonlinear programming. Athena Scientific Publishers, Belmont, Massachusetts
12. Betts JT (1993) Using sparse nonlinear programming to compute low thrust orbit transfers. J Astronaut Sci 41:349–371
13. Betts JT (1994) Optimal interplanetary orbit transfers by direct transcription. J Astronaut Sci 42:247–268
14. Betts JT (2000) Very low thrust trajectory optimization using a direct sqp method. J Comput Appl Math 120:27–40
15. Betts JT (2001) Practical methods for optimal control using nonlinear programming. SIAM Press, Philadelphia
16. Betts JT (2007) Optimal lunar swingby trajectories. J Astronaut Sci 55:349–371
17. Betts JT, Huffman WP (1994) A sparse nonlinear optimization algorithm. J Optim Theory Appl 82(3):519–541
18. Betts JT, Huffman WP (1997) Sparse optimal control software—socs. Technical report MEA-LR-085, Boeing information and support services, Seattle, Washington, July 1997
19. Björkman M, Holmström K (1999) Global optimization with the direct user interface for nonlinear programming. Adv Model Simul 2:17–37
20. Bliss GA (1946) Lectures on the calculus of variations. University of Chicago Press, Chicago, IL
21. Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press, Cambridge
22. Brauer GL, Cornick DE, Stevenson R (1977) Capabilities and applications of the program to optimize and simulate trajectories. Technical report NASA-CR-2770, National Aeronautics and Space Administration

23. Bryson AE, Ho Y-C (1975) Applied optimal control. Hemisphere Publishing, New York
24. Butcher JC (1964) Implicit runge-kutta processes. Math Comput 18(85):50–64
25. Butcher JC (2008) Numerical methods for ordinary differential equations. Wiley, New York
26. Byrd RH, Nocedal J, Waltz RA (2006) Knitro: an integrated package for nonlinear optimization. In: Large scale nonlinear optimization. Springer, Berlin, pp 35–59
27. Coverstone-Carroll VL, Hartmann JW, Mason WJ (2000) Optimal multi-objective low-thrust spacecraft trajectories. Comput Methods Appl Mech Eng 186(2–4):387–402
28. Cuthrell JE, Biegler LT (1987) On the optimization of differential-algebraic processes. AIChe J 33(8):1257–1270
29. Cuthrell JE, Biegler LT (1989) Simultaneous optimization and solution methods for batch reactor control profiles. Comput Chem Eng 13(1/2):49–62
30. Dahlquist G, Björck A (2003) Numerical methods. Dover Publications, Mineola, New York
31. Dontchev AL, Hager WW (1998) Lipschitzian stability for state constrained nonlinear optimal control. SIAM J Control Optim 36:696–718
32. Dontchev AL, Hager WW (1998) A new approach to lipschitz continuity in state constrained optimal control. Syst Control Lett 35:137–143
33. Dontchev AL, Hager WW (2001) The euler approximation in state constrained optimal control. Math Comput 70:173–203
34. Dontchev AL, Hager WW, Malanowski K (2000) Error bounds for the euler approximation and control constrained optimal control problem. Numer Funct Anal Appl 21:653–682
35. Dontchev AL, Hager WW, Veliov VM (2000) Second-order runge-kutta approximations in constrained optimal control. SIAM J Numer Anal 38:202–226
36. Dontchev AL, Hager WW, Veliov VM (2000) Uniform convergence and mesh independence of newton's method for discretized variational problems. SIAM J Control Optim 39:961–980
37. Dotzauer E, Holmström K (1999) The tomlab graphical user interface for nonlinear programming. Adv Model Simul 2:9–16
38. Enright PJ (1991) Optimal finite—thrust spacecraft trajectories using direct transcription and nonlinear programming. PhD thesis, Department of Aerospace Engineering, University of Illinois at Urbana-Champaign
39. Enright PJ, Conway BA (1996) Discrete approximations to optimal trajectories using direct transcription and nonlinear programming. J Guidance Control Dyn 19(4):994–1002, Jul–Aug 1996
40. Fleming WH, Rishel RW (1982) Deterministic and stochastic optimal control. Springer, Heidelberg
41. Gage PJ, Braun RD, Kroo IM (1995) Interplanetary trajectory optimization using a genetic algorithm. J Astronaut Sci 43(1):59–75
42. Gear WC (1971) Numerical initial-value problems in ordinary differential equations. Prentice-Hall, Englewood Cliffs, New Jersey
43. Gill PE, Murray W, Saunders MA (2002) Snopt: an sqp algorithm for large-scale constrained optimization. SIAM Rev 47(1):99–131
44. Gill PE, Murray W, Wright MH (1981) Practical optimization. Academic Press, London
45. Gill PE, Murray W, Saunders MA (2006) User's guide for SNOPT version 7: software for large scale nonlinear programming, Feb 2006
46. Goh CJ, Teo KL (1988) Control parameterization: a unified approach to optimal control problems with general constraints. Automatica 24(1):3–18
47. Goh CJ, Teo KL (1988) Miser: a fortran program for solving optimal control problems. Adv Eng Softw 10(2):90–99
48. Grimm W, Markl A (1997) Adjoint estimation from a multiple-shooting method. J Optim Theory Appl 92(2):263–283
49. Hager WW (2000) Runge-kutta methods in optimal control and the transformed adjoint system. Numerische Mathematik 87:247–282
50. Hairer E, Norsett SP, Wanner G (1993) Solving ordinary differential equations I: nonstiff problems. Springer, New York

51. Hairer E, Wanner G (1996) Solving ordinary differential equations II: stiff differential-algebraic problems. Springer, New York
52. Hargraves CR, Paris SW (1987) Direct trajectory optimization using nonlinear programming techniques. J Guidance Control Dyn 10(4):338–342
53. Hartmann JW, Coverstone-Carroll VL (1998) Optimal interplanetary spacecraft trajectories via a pareto genetic algorithm. J Astronaut Sci 46(3):267–282
54. Herman AL (1995) Improved collocation methods with application to direct trajectory optimization. PhD thesis, Department of Aerospace Engineering, University of Illinois at Urbana-Champaign
55. Herman AL, Conway BA (1996) Direct optimization using collocation based on high-order gauss-lobatto quadrature rules. J Guidance Control Dyn 19(3):592–599
56. Hildebrand FB (1992) Methods of applied mathematics. Dover Publications, Mineola, New York
57. Holmström K (1999) New optimization algorithms and software. Theor Stoch Process 1–2:55–63
58. Holmström K (1999) The tomlab optimization environment in matlab. Adv Model Simul 1:47–69
59. Holmström K, Björkman M (1999) The tomlab nlplib toolbox for nonlinear programming. Adv Model Simul 1:70–86
60. Holmström K, Björkman M, Dotzauer E (1999) The tomlab opera toolbox for linear and discrete optimization. Adv Model Simul 2:1–8
61. Hughes S (2008) Gmat—generalized mission analysis tool. Technical report, NASA Goddard Space Flight Center, http://sourceforge.net/projects/gmat
62. Hull DG (2003) Optimal control theory for applications. Springer, New York
63. Jockenhovel T (2002) Optcontrolcentre, software package for dynamic optimization. http://OptControlCentre.com/
64. Kameswaran S, Biegler LT (2008) Convergence rates for direct transcription of optimal control problems using collocation at radau points. Comput Optim Appl 41(1):81–126
65. Keller HB (1976) Numerical solution of two point boundary value problems. SIAM, Philadelphia
66. Kirk DE (2004) Optimal control theory: an introduction. Dover Publications, Mineola, New York
67. Leitmann G (1981) The calculus of variations and optimal control. Springer, New York
68. Lewis FL, Syrmos VL (1995) Optimal control, 2nd edn. Wiley, New York
69. Logsdon JS, Biegler LT (1989) Accurate solution of differential-algebraic optimization problems. Ind Eng Chem Res 28:1628–1639
70. Milam MB (2003) Real-time optimal trajectory generation for constrained dynamical systems. PhD thesis, California Institute of Technology, Pasadena, California, May 2003
71. Oberle HJ, Grimm W (1990) Bndsco: a program for the numerical solution of optimal control problems. Technical report, Institute of Flight Systems Dynamics, German Aerospace Research Establishment DLR, IB 515–89/22, Oberpfaffenhofen, Germany
72. Ocampo C (2003) An architecture for a generalized spacecraft trajectory design and optimization system. In: Proceedings of the international conference on libration point missions and applications, Aug 2003
73. Ocampo C (2004) Finite burn maneuver modeling for a generalized spacecraft trajectory design and optimization system. Ann NY Acad Sci 1017:210–233
74. Patterson MA, Rao AV (2013) GPOPS-II, a matlab software for solving multiple-phase optimal control problems *hp*—adaptive gaussian quadrature collocation methods and sparse nonlinear programming. ACM Trans Math Softw(in Revision) Sep 2013
75. Pontryagin LS (1962) Mathematical theory of optimal processes. Wiley, New York
76. Rao AV (1996) Extension of the computational singular perturbation method to optimal control. PhD thesis, Princeton University
77. Rao AV, Tang S, Hallman WP (2002) Numerical optimization study of multiple-pass aeroassisted orbital transfer. Optim Control Appl Methods 23(4):215–238

78. Rao AV (2000) Application of a dichotomic basis method to performance optimization of supersonic aircraft. J Guidance Control Dyn 23(3):570–573
79. Rao AV (2003) Riccati dichotomic basis method for solving hyper-sensitive optimal control problems. J Guidance Control Dyn 26(1):185–189
80. Rao AV, Benson DA, Darby CL, Francolin C, Patterson MA, Sanders I, Huntington GT (2010) Algorithm 902: GPOPS, a MATLAB software for solving multiple-phase optimal control problems using the gauss pseudospectral method. ACM Trans Math Softw 37(2, Article 22):39 p
81. Rao AV, Mease KD (1999) Dichotomic basis approach to solving hyper-sensitive optimal control problems. Automatica 35(4):633–642
82. Rao AV, Mease KD (2000) Eigenvector approximate dichotomic basis method for solving hyper-sensitive optimal control problems. Optim Control Appl Methods 21(1):1–19
83. Rauwolf GA, Coverstone-Carroll VL (1996) Near-optimal low-thrust orbit transfers generated by a genetic algorithm. J Spacecraft Rockets 33(6):859–862
84. Reddien GW (1979) Collocation at gauss points as a discretization in optimal control. SIAM J Control Optim 17(2):298–306
85. Ross IM, Fahroo F (2001) User's manual for DIDO 2001 α: a MATLAB application for solving optimal control problems. Technical Report AAS-01-03, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, California
86. Rutquist P, Edvall M (2008) PROPT: MATLAB optimal control software. Tomlab Optimization Inc, Pullman, Washington
87. Sauer CG (1989) Midas—mission design and analysis software for the optimization of ballistic interplanetary trajectories. J Astronaut Sci 37(3):251–259
88. Schwartz A (1996) Theory and implementation of numerical methods based on runge-kutta integration for solving optimal control problems. PhD thesis, Department of Electrical Engineering, University of California, Berkeley
89. Schwartz A, Polak E (1996) Consistent approximations for optimal control problems based on runge-kutta integration. SIAM J Control Optim 34(4):1235–1269
90. Schwartz A, Polak E, Chen Y (1997) Recursive integration optimal trajectory solver (RIOTS_95)
91. Stengel RF (1994) Optimal control and estimation. Dover Publications, Mineola, New York
92. Stoer J, Bulirsch R (2002) Introduction to numerical analysis. Springer, Berlin
93. Vinh N-X (1981) Optimal trajectories in atmospheric flight. Elsevier Science, New York
94. Vintner R (2000) Optimal control (systems and control: foundations and applications). Birkhäuser, Boston
95. Vlases WG, Paris SW, Lajoie RM, Martens MJ, Hargraves CR (1990) Optimal trajectories by implicit simulation. Technical report WRDC-TR-90-3056, Boeing Aerospace and Electronics, Wright-Patterson Air Force Base, Ohio
96. von Stryk O (1999) User's guide for DIRCOL version 2.1: a direct collocation method for the numerical solution of optimal control problems. Technische Universitat Darmstadt, Darmstadt, Germany
97. Weinstock R (1974) Calculus of variations. Dover Publications, Mineola, New York
98. Williams P (2008) User's guide for DIRECT 2.0. Royal Melbourne Institute of Technology, Melbourne, Australia
99. Wuerl A, Crain T, Braden E (2003) Genetic algorithm and calculus of variations-based trajectory optimization technique. J Spacecraft Rockets 40(6):882–888