Design of a General-Purpose Optimal Controller via the *hp*-Legendre-Gauss-Radau Collocation Method

Nicholas E. Hirsch*
University of Florida, Gainesville, FL, 32618

The design of a controller is a considerable investment in any engineering project, and the ability to reuse a general-purpose controller would be a significant time- and cost-saving measure. This paper outlines the design of a system-independent optimal controller using the Legendre-Gauss-Radau collocation method with *h*- and *p*-discretization. Given an arbitrary set of system dynamics and constraints, the controller can rapidly generate the optimal trajectory for the system with great accuracy. The theory behind classic optimal control is given, followed by a discussion of modern pseudospectral collocation methods used to solve optimal control problems numerically. The design of a software using these techniques in MATLAB is then outlined. Finally, this software is used to simulate the control of several dynamical systems — including the Bryson-Denham problem and a robotic arm — in order to demonstrate its efficacy.

I. Introduction

A significant portion of a control engineer's time is dedicated to the design of a controller suitable for a given system. While a wide variety of control schemes exist, these are often of limited applicability to systems with highly nonlinear dynamics or those subject to additional constraints [1–3]. One theoretical framework for combating these issues is known as optimal control. The control of an arbitrary system may be formulated as an optimal control problem (OCP) by representing it as an attempt at cost minimization. Informally, the cost of a system is a mapping from whatever aspects of a system should be minimized over a time interval to a scalar sum via an equation known as a cost functional [4]. In classic optimal control, a field of mathematics known as calculus of variations may then be employed to derive the minimizing control law in the form of a differential-algebraic equation (DAE) [3]. This DAE may then be solved numerically via a set of techniques referred to as indirect methods for the optimal control at any point in time.

While powerful, the variational approach to solving OCPs suffers from many of the same issues as other traditional control techniques. Namely, for each system, the corresponding DAE must be independently derived [3, 5]. In addition, any system constraints (including the system's dynamics) must be appended to the cost functional via a method known as Lagrange multipliers [1, 2], oftentimes greatly complicating the derivation. A new class of optimal control techniques known as direct methods circumvents these issues by simply guessing the optimal control law for the system and then optimizing it numerically [3, 5, 6]. This moves the burden of optimization from manual derivation via calculus of variations to numeric computation. As a result, the only things that must be manually formulated for a given system are its cost functional and corresponding constraints. This allows for the development of a universal framework which, given these two quantities, returns the optimal control of the system.

The remainder of this paper is dedicated to this development and is split into three parts. The first covers the mathematical formulation of the OCP and a direct method known as Legendre-Gauss-Radau collocation with *h*- and *p*-discretization (*hp*-LGR) which may be used to efficiently solve for the optimal control [4, 5]. Next, the implementation of these techniques in the design of a system-agnostic controller in MATLAB is detailed. Finally, the efficacy of this software is demonstrated on two distinct OCPs.

II. The Optimal Control Problem

Consider an arbitrary dynamical system governed by a set of equations,

$$\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) = \dot{\mathbf{x}}\,, \tag{1}$$

where $\mathbf{x}(t) \in \mathbb{R}^{n_x}$ and $\mathbf{u}(t) \in \mathbb{R}^{n_u}$ are covectors which represent the current state of, and control input to, the system respectively. t represents any direct time-dependence of the dynamics. The OCP is to find the $\mathbf{u}$ which minimizes

*Aerospace Engineering Undergraduate, Department of Mechanical and Aerospace Engineering, PO Box 116250, Gainesville FL, 32611, AIAA Student Member, 1810373.

the cost functional, J, over a fixed length of time, $t \in [t_0, t_f]$ (although infinite-horizon formulations do exist [7]). Traditionally, J may be written in what is known as Bolza form as,

$$J = M(\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_f; t_0, t_1, ..., t_f) + \int_{t_0}^{t_f} L(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}, t)\, dt \ , \tag{2}$$

where M is known as the Mayor/fixed cost and L is the Lagrange/running cost. M is paid at a finite number of discrete time steps whereas L is paid continuously over the entire trajectory.

Eq. (2) is constrained by the system dynamics given in Eq. (1) as well as two other primary forms of constraint*: boundary constraints, $\mathbf{b}$, and path constraints, $\mathbf{c}$. Boundary constraints fix $\mathbf{x}$, $\mathbf{u}$, and/or t at a set of discrete points in time and can be represented via the equality,

$$\mathbf{b}(\mathbf{x}_i, \mathbf{u}_i, t_i) = \mathbf{0},\ i \in \mathbb{Z}\ ; \tag{3}$$

whereas path constraints restrict the domains of $\mathbf{x}$ and/or $\mathbf{u}$ to a reduced range of values over an interval of time. The latter may appear in either equality or inequality form as,

$$\mathbf{c}_{\min}(\mathbf{x}, \mathbf{u}) \leq \mathbf{c}(\mathbf{x}, \mathbf{u}) \leq \mathbf{c}_{\max}(\mathbf{x}, \mathbf{u}) \ , \tag{4}$$

where in the case of an equality, $\mathbf{c}_{\min} = \mathbf{c}_{\max}$.

III. *hp*-Legendre-Gauss-Radau Collocation

A. Overview of Direct Methods

In general, the structure of a direct method for solving an OCP begins with making a guess as to the nature of the control law. It is common to fit $\mathbf{u}$ to a function of the form,

$$\mathbf{u}(t) = \sum_{i=1}^{N} \mathbf{C}_i \psi_i(t),\ N \in \mathbb{Z} \geq 1 \ , \tag{5}$$

where $\{\psi_i(t)\}$ represents a set of basis functions with a corresponding set of constant coefficients, $\{\mathbf{C}_i\}$. With the structure of $\mathbf{u}$ now known, it may be plugged into Eq. (2) to convert it from a functional to function†.

Before beginning optimization, the number and form of the trial functions is picked and guesses as to the values of the coefficients then made. Optimization is performed iteratively via an optimizer such as MATLAB's `fmincon()` [8], with the value of J and error in the constraints being computed for a given set of $\{\mathbf{C}_i\}$ along with any other free variables included in the problem — such the state or control at a discrete point in time, $\mathbf{X}_i = \mathbf{x}(t_i),\ t_i \in [t_0, t_f]$ and $\mathbf{U}_i = \mathbf{U}(t_i),\ t_i \in [t_0, t_f]$, or even t_0 and t_f themselves. Taken together, the set of variables being optimized can be collocated into a decision vector, $\mathbf{Z}$, of the form,

$$\mathbf{Z} = \begin{bmatrix} \mathbf{C}_0 & \cdots & \mathbf{C}_N & \mathbf{X}_0 & \cdots & \mathbf{X}_f & \mathbf{U}_0 & \cdots & \mathbf{U}_f & t_0 & t_f \end{bmatrix} \ . \tag{6}$$

With $\mathbf{Z}$ defined, the goal of an OCP is now to find the optimal decision vector, $\mathbf{Z}_*$, which may be written as,

$$\mathbf{Z}_* = \arg\min_{\mathbf{Z}} J \quad \text{s.t.} \quad \begin{aligned} &\mathbf{f}(\mathbf{x}, \mathbf{u}) - \dot{\mathbf{x}} = \mathbf{0} \ , \\ &\mathbf{b}(\mathbf{x}_i, \mathbf{u}_i, t_i) = \mathbf{0},\ i \in \mathbb{Z} \ , \\ &\mathbf{c}_{\min}(\mathbf{x}, \mathbf{u}) \leq \mathbf{c}(\mathbf{x}, \mathbf{u}) \leq \mathbf{c}_{\max}(\mathbf{x}, \mathbf{u}) \ . \end{aligned} \tag{7}$$

$\mathbf{Z}_*$ contains the set of optimized coefficients, $\{\mathbf{C}_i\}_*$, which together with $\{\psi_i(t)\}$ form the approximation for the optimal control, $\mathbf{u}_*$.

*Other forms of constraints, such as event constraints [1], exist but are beyond the scope of this treatment.

†In indirect formulations J is known as a functional because it takes in an arbitrary function, $\mathbf{u}(t)$, and returns a scalar value. However, in direct formulations the form of $\mathbf{u}(t)$ us known and as such J is reduced to a standard function which takes in a scalar, t, and returns a scalar.

B. Lagrange Polynomials

In the process of approximating $\mathbf{u}$ we first discretize its domain into k time points. In order to utilize integration techniques discussed in the next section, we also perform an affine transformation from the t-domain to the τ-domain, where $\tau \in [-1, 1]$. The two domains are related by the mapping,

$$\tau = 2\left(\frac{t - t_0}{t_f - t_0}\right) - 1 \,, \tag{8}$$

such that $\tau(t_0) = -1$ and $\tau(t_f) = 1$.

We then seek a set of $\{\psi_i(t)\}$ such that at these k points our approximation for $\mathbf{u}$, $\mathbf{U}(\tau)$, satisfies the property:

$$\mathbf{U}(\tau_k) \equiv \mathbf{u}(\tau_k), \; k \in \mathbb{Z}_{\geq 1} \,. \tag{9}$$

To this end, we select Lagrange polynomials of the form,

$$l_i(\tau) = \prod_{\substack{j=1 \\ j \neq i}}^{N} \frac{\tau - \tau_j}{\tau_i - \tau_j} \,, \tag{10}$$

to comprise our basis functions, where the set of $\{\tau_j\}$ are known as support points. Lagrange polynomials satisfy Eq. (9) via the fact that if we choose our discretization points, $\{\tau_k\}$, as our support points,

$$l_i(\tau_k) = \begin{cases} 1, \; i = k \\ 0, \; i \neq k \end{cases} \,, \tag{11}$$

which is known as the isolation property [3]. Thus,

$$\mathbf{U}(\tau_k) \equiv \mathbf{C}_k \,. \tag{12}$$

In addition, Lagrange polynomials are easily differentiable and provide a high level of accuracy when used in conjunction with pseudospectral methods for solving differential equations, both of which are important for later developments [3].

We may write our approximation for $\mathbf{u}$ using Eq. (10) and Eq. (12) as,

$$\mathbf{u}(\tau) \approx \mathbf{U}(\tau) = \sum_{i=1}^{N} \mathbf{U}_i l_i(\tau) \,, \tag{13}$$

where,

$$\mathbf{U}_i = \mathbf{U}(\tau_i) = \begin{bmatrix} U_1(\tau_i) & U_2(\tau_i) & \cdots & U_{n_u}(\tau_i) \end{bmatrix} \,. \tag{14}$$

This alters the form of Eq. (7) as $\{\mathbf{C}_i\}$ is now implicitly contained in $\mathbf{Z}$ via $\{\mathbf{U}_i\}$ such that $\{\mathbf{U}_i\}$ must now be optimized for at the k discrete time points. It is important to distinguish the fact that t is discretized into k τ-points while $\mathbf{u}$ is discretized at i τ-points. $\mathbf{U}_i(\tau_i) = \mathbf{U}_i(\tau_k)$ so long as $i = k$, and furthermore at that point $\tau_i = \tau_k$. While the two sets of τ may seem identical, the distinction is important because the two may be of different sizes.

Our choice of $\{\tau_k\}$, equivalently $\{\tau_j\}$, affects the accuracy of our polynomial interpolation. Discretizing the t-domain into a set of evenly-spaced points causes Runge phenomenon - a form of interpolation error which grows to infinity as the distance between interpolation points goes to zero. To avoid this, we instead choose $\{\tau_k\}$ to be the Legendre-Gauss-Radau (LGR) points in order to cluster the support points at the endpoints where oscillatory behavior is greatest [9]. These are the roots of the function, $R(x)$, given by,

$$R(x) = \frac{P_{N-1}(x) + P_N(x)}{1 + x} \,, \tag{15}$$

along with an additional LGR point at -1. $P_N(x)$ represents the N-degree Legendre polynomial. Notably, these roots are on the interval $[-1, 1)$ and hence $\mathbf{u}(t_f)$ is not included in the discretization of $\mathbf{u}$. While there are methods which can recover the control at t_f [10], they are outside the scope of this paper. It can thus be seen that for LGR quadrature $k \in [1, N + 1]$ whereas $i \in [1, N]$. Accordingly, $\{\tau_k\}$ may be referred to as the support or discretization points whereas $\{\tau_i\}$ are known as the LGR or collocation points; with the difference being that the former includes $\tau(t_f) = 1$ as the $(N + 1)$-th point while the later does not.

C. Legendre-Gauss-Radau Quadrature

In the optimization of Eq. (7) a method of numeric integration is needed to evaluate Eq. (2). Since LGR points are already being used in the approximation of $\mathbf{u}$, it is natural to choose LGR quadrature - a form of Legendre-Gauss (LG) quadrature where the quadrature points are chosen to be the LGR points. The general LG quadrature rule is as follows,

$$\int_{-1}^{1} f(\tau)\, d\tau \approx \sum_{k=1}^{N} w_k f(\tau_k) \,, \tag{16}$$

where $f(\tau)$ represents a generic function defined on $\tau \in [-1, 1]$. τ is discretized into k quadrature points which each have a corresponding weight, w_k, which indicates to what extent $f(\tau_k)$ contributes to the quadrature. When LGR points are used, $w_k \; \forall \; k \neq 1$, is given by the formula,

$$w_k = \frac{1 - x_k}{N^2 [P_{N-1}(x_k)]^2}, \; k \neq 1 \,, \tag{17}$$

and for $k = 1$,

$$w_1 = \frac{2}{N^2} \,, \tag{18}$$

as seen in Ref. [11]. Note that $P_N(x)$ represents the N-th degree Legendre polynomial [12].

Added benefits of LGR quadrature are its high accuracy[‡] and relatively low computational intensity [3, 5, 6, 11]. In addition, it can easily be incorporated into an integration scheme for solving sets differential equations such as Eq. (1).

In addition to Eq. (13), we complete our discretization of our OCP by defining an approximation for $\mathbf{x}$ of the same form via,

$$\mathbf{x}(\tau) \approx \mathbf{X}(\tau) = \sum_{i=1}^{N+1} \mathbf{X}_i l_i(\tau) \,, \tag{19}$$

where it is noted that $\mathbf{X}_i$ takes on a form equivalent to $\mathbf{U}_i$. $\{\mathbf{X}_i\}$ must be included in $\mathbf{Z}$ similar to the $\{\mathbf{U}_i\}$. Note that τ_{N+1}, i.e. $\tau(t_f)$, is included as a support point for $\mathbf{X}$ despite it not being an LGR point, and we have abused notation by allowing $i \in [0, N + 1]$ for the discretization of $\mathbf{X}$. This is a necessary sacrifice in order to be able to compute M in our discretized OCP. Using Eq. (16) in conjunction with Eq. (13) and Eq. (19), we may rewrite Eq. (2) as,

$$J \approx M(\mathbf{X}_1, ..., \mathbf{X}_{N+1}, \tau_1, ..., \tau_{N+1}) + \frac{t_f - t_0}{2} \sum_{i=1}^{N} w_i L(\mathbf{X}_i, \mathbf{U}_i, \dot{\mathbf{X}}_i, \tau_i) \,, \tag{20}$$

noting that τ is a function of t via its definition in Eq. (8) which necessitates the difference attached to the summation. The derivation of the discrete state derivative, $\dot{\mathbf{X}}$, is given in the next section.

D. Collocation

We may now combine Eq. (10) and Eq. (16) into an integration scheme for Eq. (1) known as the LGR pseudospectral method. This belongs to a subset of methods for integrating differential equations known as collocation methods. As opposed to traditional time-marching methods like Euler's formula, which integrate over the domain of a differential equation through a sequential series of intervals, collocation simultaneously integrates all intervals via the transformation of the integration into a root-finding problem. This is advantageous for OCPs because it helps minimize the curse of dimensionality that occurs when trajectories are long and comprised of many states [3]. To begin, we seek the value of both sides of Eq. (1) at $\{\tau_k\}$. We may take the derivative with respect to t of Eq. (19) to obtain,

$$\dot{\mathbf{x}}(\tau) = \frac{d}{d\tau}\mathbf{x}(\tau) \approx \dot{\mathbf{X}} = \frac{d}{d\tau} \sum_{i=1}^{N+1} \mathbf{X}_i l_i(\tau) = \sum_{i=1}^{N+1} \mathbf{X}_i \dot{l}_i(\tau) \,. \tag{21}$$

Equivalently, by discretizing Eq. (1) we may write that,

$$\dot{\mathbf{x}}(\tau) \approx \mathbf{F}[\mathbf{X}(\tau), \mathbf{U}(\tau), t(\tau)] \frac{dt}{d\tau} = \frac{t_f - t_0}{2} \mathbf{F}[\mathbf{X}(\tau), \mathbf{U}(\tau), t(\tau)] \,. \tag{22}$$

[‡]LGR quadrature is exact for a $2N - 1$-degree polynomial [11].

To discretize and transform Eq. (1) into a root-finding form suitable for collocation we rewrite it as,

$$\dot{\mathbf{X}} - \mathbf{F}(\mathbf{X}, \mathbf{U}, \tau) = \mathbf{0} \, . \tag{23}$$

Substituting in Eq. (21) and Eq. (22) at a single τ_k we now have,

$$\mathbf{\Delta}_k = \sum_{i=1}^{N+1} \mathbf{X}_i \dot{l}_i(\tau_k) - \frac{t_f - t_0}{2} \mathbf{F}(\mathbf{X}_k, \mathbf{U}_k, \tau_k) = \mathbf{0} \, , \tag{24}$$

where $\mathbf{\Delta}_k$ is known as the defect constraint at τ_k. To expand the collocation scheme to the entirety of the τ-domain we define the differentiation matrix, $\mathbf{D}$, such that,

$$D_{ki} = \dot{l}(\tau_k) \, , \tag{25}$$

and hence we may rewrite Eq. (24) for all k as,

$$\begin{bmatrix} \mathbf{\Delta}_1 \\ \vdots \\ \mathbf{\Delta}_N \end{bmatrix} = \begin{bmatrix} D_{11} & \cdots & D_{1(N+1)} \\ \vdots & & \vdots \\ D_{N1} & \cdots & D_{N(N+1)} \end{bmatrix} \begin{bmatrix} X_{11} & \cdots & X_{1n_x} \\ \vdots & & \vdots \\ X_{(N+1)1} & \cdots & X_{(N+1)n_x} \end{bmatrix} - \frac{t_f - t_0}{2} \begin{bmatrix} F_{11} & \cdots & F_{1n_x} \\ \vdots & & \vdots \\ F_{N1} & \cdots & F_{Nn_x} \end{bmatrix} = \mathbf{0} \, , \tag{26}$$

where, $F_{kj} = F_j(\mathbf{X}_k, \mathbf{U}_k, \tau_k), \ j \in [0, n_x] \in \mathbb{Z}$. This may also be written as:

$$\mathbf{\Delta} = \mathbf{D}\mathbf{X} - \frac{t_f - t_0}{2} \mathbf{F} = \mathbf{0} \, . \tag{27}$$

Note that because the set of $\{\tau_i\}$ for $\mathbf{U}$ is missing the final point from $\{\tau_k\}$ there are only N defect constraints.

Eq. (27) in conjunction with Eq. (20) and any discretized boundary and/or path constraints[§] provides the complete LGR discretization of an OCP.

E. *hp*-Discretization

The number of discretization points used for $\{l_i\}$ may be varied to increase the accuracy $\mathbf{X}$ and $\mathbf{U}$. The ability to vary this parameter is known as p-discretization. However, if the degree of $\{l_i\}$ grows top large it can lead to over-fitting and other interpolation errors [13][9]. To combat this, the total trajectory may be split into a series of intervals, each with their own set of$\{l_i\}$, and then patched back together. This process is known as h-discretization, first developed for *hp*-LGR collocation in Ref. [5].

To split the domain of the trajectory into intervals we define a new variable, $s \in [-1, 1]$. We may then convert a τ-interval $\tau \in [\tau_{k-1}, \tau_k]$ to the s-domain using the conversion,

$$s = 2 \left(\frac{\tau - \tau_{k-1}}{\tau_k - \tau_{k-1}} \right) - 1 \, . \tag{28}$$

In effect, we have mapped our original t-domain to $[-1, 1]$ via the τ-domain. We then used the s-domain to further map each interval, $\tau \in [\tau_{k-1}, \tau_k]$, in the τ-domain (which are notably not on $[-1, 1]$) back onto $[-1, 1]$. Let N_k be the number of LGR points in grid interval, $k \in [1, N]$, such that each interval contains a set of $\{s_p^{(k)}\}$ discretization points, where $p \in [1, N_k + 1]$. We may then rewrite Eq. (23) and Eq. (25) for a single interval, k, in the s-domain as,

$$\dot{\mathbf{x}}^{(k)}(s) \approx \mathbf{F}^{(k)}[\mathbf{X}^{(k)}(s), \mathbf{U}^{(k)}(s), t(\tau(s))] \frac{ds}{d\tau} \frac{d\tau}{dt} = \frac{\tau_k - \tau_k}{2} \frac{t_f - t_0}{2} \mathbf{F}^{(k)}[\mathbf{X}^{(k)}(s), \mathbf{U}^{(k)}(s), t(\tau(s))] \, , \tag{29}$$

where a superscript, (k), denotes which s-interval a term belongs to; and,

$$D_{pi}^{(k)} = \dot{l}(s_p^{(k)}) \, , \tag{30}$$

[§]To discretize a constraint simply apply it only at the k discrete time points which fall within the continuous t-interval in which it was defined.

respectively. With this, we may rewrite Eq. (27) as,

$$\begin{bmatrix} \boldsymbol{\Delta}^{(1)} \\ \vdots \\ \boldsymbol{\Delta}^{(N)} \end{bmatrix} = \begin{bmatrix} \mathbf{D}^{(1)} & 0 & \cdots & 0 \\ 0 & \mathbf{D}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \mathbf{D}^{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{X}_1^{(1)} \\ \vdots \\ \mathbf{X}_{N_1}^{(1)} \\ \mathbf{X}_1^{(2)} \\ \vdots \\ \mathbf{X}_{N_k}^{(N)} \\ \mathbf{X}_{N_k+1}^{(N)} \end{bmatrix} - \frac{\tau_k - \tau_{k-1}}{2} \frac{t_f - t_0}{2} \begin{bmatrix} \mathbf{F}_1^{(1)} \\ \vdots \\ \mathbf{F}_{N_1}^{(1)} \\ \vdots \\ \mathbf{F}_{N_k}^{(N)} \end{bmatrix} = \mathbf{0} \,, \tag{31}$$

where $\boldsymbol{\Delta}^{(k)}$ and $\mathbf{D}^{(k)}$ represent the collocated defect constraints and differentiation matrices respectively for the k-th s-interval. Each s-interval could be though of as its own single-domain collocation, and Eq. (31) is hence the concatenation of all of these individual problems. The inter-linkage of each interval comes from the overlap between the last s-point in the $(k-1)$-th interval and first s-point in the k-th interval. This is enforced by the nature of the collocated $\mathbf{X}$ (and $\mathbf{F}$) vector in Eq. (31), where it can be seen that,

$$\mathbf{X}_{N_{k-1}+1}^{(k-1)} = \mathbf{X}_1^{(k)} \,, \tag{32}$$

which is known as a consistency constraint. In addition, the last column of $\mathbf{D}^{(k-1)}$ is positioned directly above the first of $\mathbf{D}^{(k)}$ so that collocation still functions properly with this imposed constraint. Note also that there is no consistency constraint on $\mathbf{X}$ or $\mathbf{F}$ at $s_1^{(1)} = s(\tau(t_0))$ or $s_{N_k+1}^{(N)} = s(\tau(t_f))$. Finally, the h-discretized collocation is still efficient because of how sparse the collocated differentiation matrices are [5].

Eq. (31), along with the multi-domain cost function,

$$J = M(\mathbf{X}_1^{(1)}, ..., \mathbf{X}_{N_k+1}^{(N)}; s_1^{(1)}, ..., s_{N_k+1}^{(N)}) + \frac{\tau_k - \tau_{k-1}}{2} \frac{t_f - t_0}{2} \sum_{k=1}^{N} \sum_{i=1}^{N_k} w_i^{(k)} L_i^{(k)}(\mathbf{X}_i^{(k)}, \mathbf{U}_i^{(k)}, \dot{\mathbf{X}}_i^{(k)}, s_i^{(k)}) \,, \tag{33}$$

and any discretized constraints form the *hp*-LGR collocation method for solving OCPs.

IV. Implementation

We now turn to the implementation of the *hp*-LGR collocation method in MATLAB. As formulated in Eq. (7), our OCP may be thought of as a nonlinear programming problem (NLP) [3]. This is a consequence of the fact that our OCP is an optimization problem involving a nonlinear function to optimize, J, subject to a set of nonlinear constraints, $\mathbf{f}$, and potentially $\mathbf{c}$. For this paper the implementation was performed in MATLAB, with its inbuilt function *fmincon()* serving as the NLP solver. It was chosen because of its ability to handle not just nonlinear objective functions but also nonlinear constraints as well as crucially, its auto-differentiation capability [8]. At each step of optimization, *fmincon)()* guesses a value for $\mathbf{Z}$ and then computes the new value of J as well as the error in the constraints. It continues this process until it reaches any of the user specified tolerances, normally the allowed error for $\boldsymbol{\Delta}$ and $\mathbf{c}$. The only major downside of *fmincon()* is that $\mathbf{Z}$ must be a vector, so any time $\mathbf{X}$ and/or $\mathbf{U}$ are appended to or extracted from $\mathbf{Z}$ they must be altered via functions such as *reshape()*.

In order to make the software truly general-purpose, it must be capable of taking in a "NLP-file" containing all necessary information about the problem and from there returning the optimal control law and corresponding trajectory. This file must contain the following:

1) The number of intervals desired for h-discretization.
2) The degree of the Lagrange polynomial in each interval, ie. how many discretization points the interval should have.
3) An initial guess for $\mathbf{Z}$ comprised of $\mathbf{X}$, $\mathbf{U}$, t_0, and t_f.
4) The dynamics of the system.
5) Any boundary and/or constant path constraints.
6) Any non-constant path constraints.

When using *fmincon()* constant and non-constant constraints must be handled differently due to how *fmincon()* performs optimization. Constant constraints can simply be implemented as bounds on $\mathbf{Z}$ which are specified during initialization

of *fmincon()*. However, as a consequence they can not be updated during optimization and as such non-constant constraints must be handled separately. *fmincon()* allows the user to specify a series of nonlinear constraints, and in addition to $\mathbf{\Delta}$ any other non-constant constraints may also be passed through here.

Once the NLP-file is constructed it is passed into the software, at which point $\mathbf{D}$, $\{w_k\}$, and the τ-domain and s-domain discretization points are computed. Since these do not vary with $\mathbf{Z}$, they can be calculated before *fmincon()* is called, significantly reducing the number of floating-point operations needed on each optimization cycle. From there, all the necessary information from the NLP-file as well as any newly computed quantities is passed to *fmincon()* and optimization ensues. When finished $\mathbf{Z}$ is returned and its components automatically extracted — returning the optimal control law to the user. The logic flow for this program is shown in Figure 1.

V. Examples

A. The Bryson-Denham Problem

Consider a double-integrator system with a constraint on the first state of the form:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ u \end{bmatrix}, \tag{34}$$

$$\mathbf{b} = \begin{bmatrix} x_1(t_0) & x_2(t_0) & t_0 & x_1(t_f) & x_2(t_f) & t_f \end{bmatrix}^\intercal = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 & 1 \end{bmatrix}^\intercal, \tag{35}$$

$$x_1 \leq c_{\max} = 1/9\,, \tag{36}$$

where the goal is to minimize the squared control effort such that,

$$J = \frac{1}{2}\int_{t_0}^{t_f} u^2\, dt\,, \tag{37}$$

as discussed in Ref. [14]. While the dynamic constraints are linear, the added path constant restricts the domain of x_1 and adds addition hurdles to a classical variational approach to solving the problem. However, using our implementation of direct methods an upper bound is simply added to all components of $\mathbf{Z}$ corresponding to x_1 to accurately incorporate the path constraint.

An NLP file was constructed for a 10 interval t-grid with each interval containing a 10-th degree Lagrange polynomial. Figure 2 shows the corresponding optimal control and Figure 3 the accompanying trajectory. It can seen that the path constraint was enforced accurately to within its stated tolerance, in this case, 10^{-8}.

B. The Robotic Arm Problem

Consider a robotic arm which must move between two configurations as quickly as possible, modeled by the following system:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} x_2 \\ u_1/5 \\ x_4 \\ 3u_2/[(5-x_3)^3 + x_1^3] \\ x_6 \\ 3u_3/\{[(5-x_3)^3 + x_1^3]\sin^2(x_5)\} \end{bmatrix}, \tag{38}$$

$$\mathbf{b}(t_0) = \begin{bmatrix} x_1(t_0) & x_2(t_0) & x_3(t_0) & x_4(t_0) & x_5(t_0) & x_6(t_0) & t_0 \end{bmatrix}^\intercal = \begin{bmatrix} 4.5 & 0 & 0 & 0 & \frac{\pi}{4} & 0 & 0 \end{bmatrix}^\intercal, \tag{39}$$

$$\mathbf{b}(t_f) = \begin{bmatrix} x_1(t_f) & x_2(t_f) & x_3(t_f) & x_4(t_f) & x_5(t_f) & x_6(t_f) \end{bmatrix}^\intercal = \begin{bmatrix} 4.5 & 0 & \frac{2\pi}{3} & 0 & \frac{\pi}{4} & 0 \end{bmatrix}^\intercal, \tag{40}$$

$$\begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} = \mathbf{c}_{\min} \leq \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \leq \mathbf{c}_{\max} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{41}$$

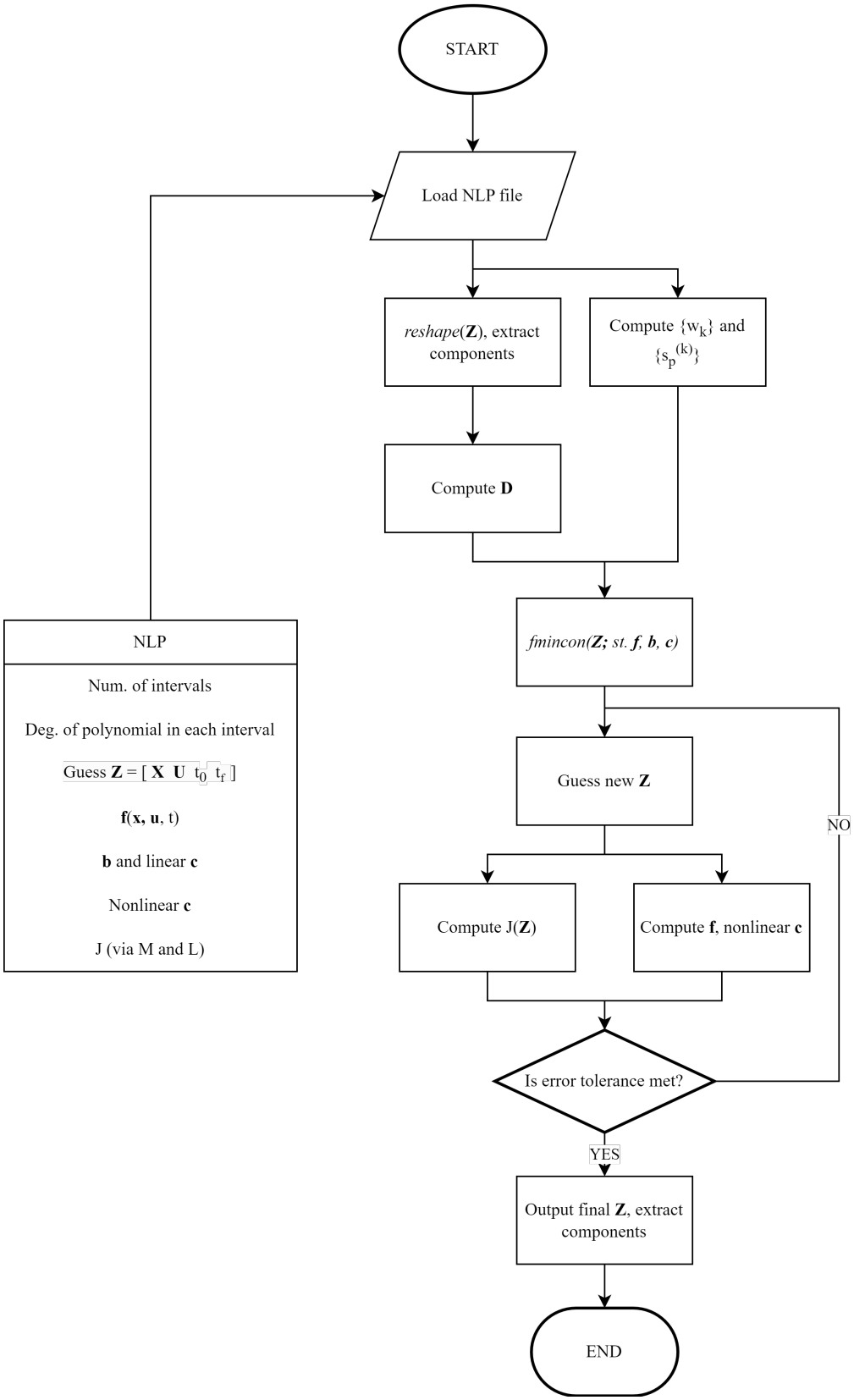

Fig. 1 Program logic flowchart for implementing *hp*-LGR collocation in MATLAB.

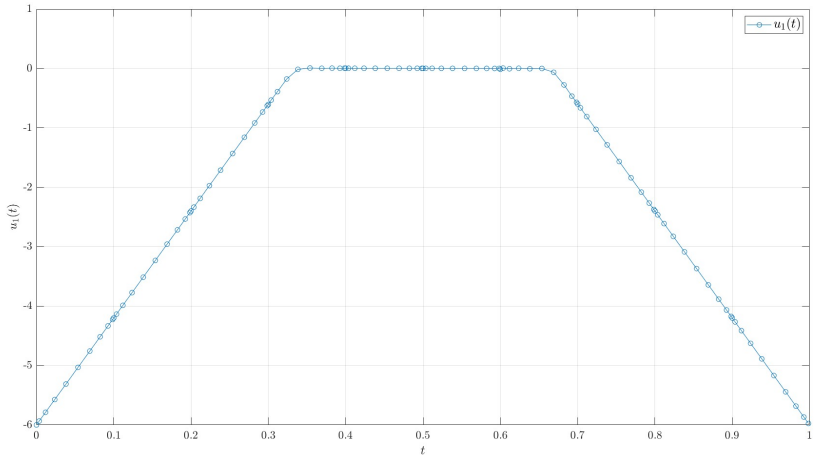

Fig. 2 Optimal control for the Bryson-Denham problem.

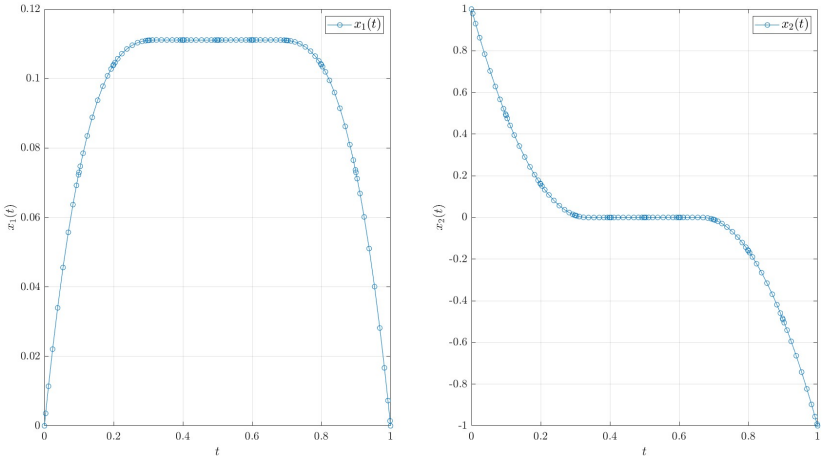

Fig. 3 Optimal trajectory for the Bryson-Denham problem.

The corresponding cost-function is simply given by:

$$J = t_f \ , \tag{42}$$

as described in Ref. [4]. In addition to being an overall increase in complexity when compared to the Bryson-Denham problem, the robotic arm problem expands the scope of optimization from the control to also include a variable final time. Regardless of these changes, the software still rapidly and accurately generates the optimal control[¶] and trajectory, as shown in Figure 2 and Figure 3 respectively, again with a constraint tolerance of 10^{-8}, and this time over a 3 interval t-grid with each interval containing a 10-th degree Lagrange polynomial.

[¶]The exception to this occurs at near discontinuous changes in the control, known as a bang-bang control scheme. The placement of the LGR points prevents high accuracy at these instances and more advanced methods are needed to resolve these inaccuracies, such as those detailed in Ref. [4].

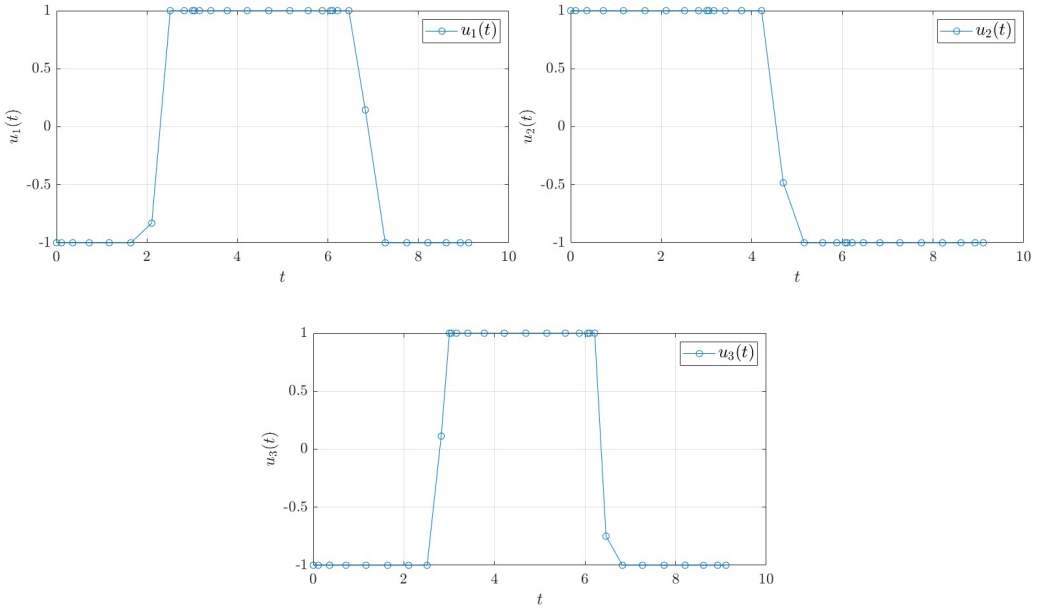

Fig. 4 Optimal control for the robotic arm problem.

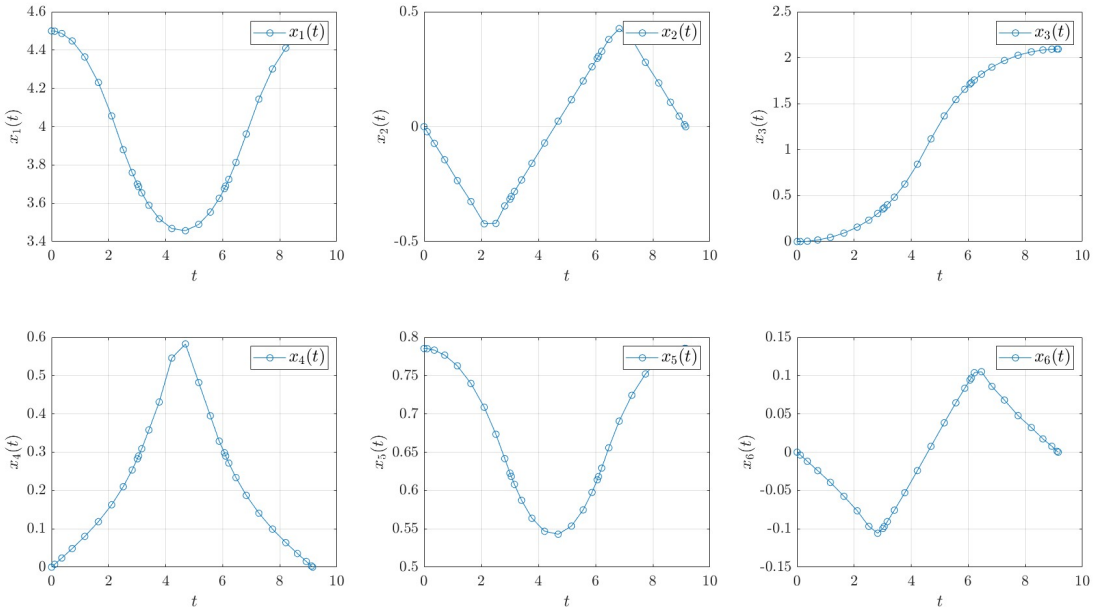

Fig. 5 Optimal trajectory for the robotic arm problem.

VI. Conclusion

The fundamental formulation of an OCP was introduced. The direct method of solving an OCP via *hp*-LGR collocation has been demonstrated and some the benefits and limitations of such a technique discussed. This was then utilized in the design of a general-purpose controller, which given an arbitrary dynamically system, is capable of automatically generating the optimal control law and corresponding trajectory with respect to a minimized cost

function subject to an array of constraints. The efficacy of said software was then demonstrated on two optimal control problems. Future research could include the development of an adaptive mesh refinement scheme to increase accuracy, costate estimation, or methods for handling path constraints which induce bang-bang control schemes or other large discontinuities.

Acknowledgments

The author would like to thank Dr. Anil V. Rao of the University of Florida's Vehicle Dynamics and Optimization Laboratory as well as Alexander Davies, Katrina Winkler, Dr. Cale Byczkowski, Gabriela Abadia-Doyle, George Haman III, and Emily Pager for their invaluable mentorship.

References

[1] Bryson, A. E., and Ho, Y.-C., *Applied Optimal Control: Optimization, Estimation, and Control*, CRC Press, 1975, pp. 19, 136.

[2] Stengel, R. F., *Optimal Control and Estimation*, Dover Books on Mathematics, Dover Publications, 1994, pp. 184–247.

[3] Rao, A. V., "A Survey of Numerical Methods for Optimal Control," *Advances in the Astronautical Sciences*, Vol. 135, No. 1, 2010.

[4] Pager, E. R., and Rao, A. V., "Method for Solving Bang-Bang and Singular Optimal Control Problem Using Adaptive Radau Collocation," *Computational Optimization and Applications*, Vol. 81, No. 3, 2022, pp. 857–887. https://doi.org/10.1007/s10589-022-00350-6.

[5] Rao, A. V., Benson, D. A., Darby, C. L., Patterson, M. A., Francolin, C., Sanders, I., and Huntington, G. T., "Algorithm 902: GPOPS, A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using the Gauss Pseudospectral Method," *Association for Computing Machinery Transactions on Mathematical Software*, Vol. 37, No. 2, 2010, pp. 22:1–22:39. https://doi.org/10.1145/1731022.1731032.

[6] Huntington, G. T., and Rao, A. V., "A Comparison of Accuracy and Computational Efficiency of Three Pseudospectral Methods," *Journal of Guidance, Control, and Dynamics*, Vol. 31, No. 2, 2008. https://doi.org/10.2514/1.30915.

[7] Carson, H. A. B., Dean A, and Leizarowitz, A., *Infinite Horizon Optimal Control: Deterministic and Stochastic Systems*, Springer Verlag, 1991.

[8] "Constrained Nonlinear Optimization Algorithms," *Mathworks Help Center*, Mathworks, retrieved 26 Feburary 2025. URL https://www.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html.

[9] Berrut, J.-P., and Trefethen, L. N., "Barycentric Lagrange Interpolation," *Society for Industrial and Applied Mathematics Review*, Vol. 46, No. 3, 2004, pp. 501–517. https://doi.org/10.1137/S0036144502417715.

[10] Abadia-Doyle, G., and Rao, A. V., "Modified Legendre-Gauss Collocation Method for Solving Optimal Control Problems with Nonsmooth Solutions," *IEEE Conference on Decision and Control*, 16–19 December 2024.

[11] Weisstein, E. W., "Radau Quadrature," *Wolfram Mathworld*, Wolfram Web, retrieved 17 Feburary 2025. URL https://mathworld.wolfram.com/RadauQuadrature.html.

[12] Lanczos, C., "Trigonometric Interpolation of Empirical and Analytical Functions," *Journal of Mathematics and Physics*, Vol. 17, No. 1-4, 1938. https://doi.org/10.1002/sapm1938171123.

[13] Archer, B., and Weisstein, E. W., "Lagrange Interpolating Polynomial," *Wolfram Mathworld*, Wolfram Web, retrieved 17 Feburary 2025. URL https://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html.

[14] "Bryson-Denham Problem," *GPOPS-II*, RP Optimization Research, retrieved 26 Feburary 2025. URL https://gpops2.com/Examples/Bryson-Denham.html.