NUMERICAL SIMULATIONS OF A ROLLING BALL ROBOT ACTUATED BY INTERNAL POINT MASSES

VAKHTANG PUTKARADZE
Department of Mathematical and Statistical Sciences
University of Alberta, CAB 632
Edmonton, AB T6G 2G1, Canada
and
ATCO SpaceLab, 5302 Forand ST SW
Calgary, AB T3E 8B4, Canada

STUART ROGERS∗
Institute for Mathematics and its Applications
University of Minnesota, Twin Cities
207 Church ST SE, 306 Lind Hall
Minneapolis, MN 55455, USA

(Communicated by Stephen L. Campbell)

Abstract. The controlled motion of a rolling ball actuated by internal point masses that move along arbitrarily-shaped rails fixed within the ball is considered. The controlled equations of motion are solved numerically using a predictor-corrector continuation method, starting from an initial solution obtained via a direct method, to realize trajectory tracking and obstacle avoidance maneuvers.

1. Introduction.

1.1. Overview. This paper is a continuation of [58], providing numerical solutions of the controlled equations of motion for several special cases of the rolling disk and ball actuated by moving internal point masses. The paper [58] invokes Pontryagin’s minimum principle to derive the theoretical background for the optimal control of the rolling disk and ball having general performance indexes. This paper implements the theory derived in [58] to solve several practical examples, such as trajectory tracking for the rolling disk and obstacle avoidance for the rolling ball. The key contributions of this paper are listed below.

• The controlled equations of motion, for a rolling ball actuated by internal point masses that move along arbitrarily-shaped rails fixed within the ball, are solved numerically by a predictor-corrector continuation method, starting from an initial solution provided by a direct method.

2010 Mathematics Subject Classification. Primary: 37J60, 70E18, 70E60, 49J15; Secondary: 49K15, 34A34, 65L10.

Key words and phrases. Optimal control, rolling ball robots, trajectory tracking, obstacle avoidance, predictor-corrector continuation.

∗ Corresponding author: Stuart Rogers.
• Jacobians of the ordinary differential equations (ODEs) and boundary conditions (BCs) which constitute the controlled equations of motion (i.e. an ordinary differential equation two-point boundary value problem (ODE TPBVP)) corresponding to the optimal control of a dynamical system are derived. These Jacobians are useful for numerically solving the controlled equations of motion.
• Algorithms for solving an ODE TPBVP by predictor-corrector continuation methods are developed and were implemented in MATLAB to numerically solve the controlled equations of motion for the rolling ball. There are not very many predictor-corrector continuation methods publicly available for solving dynamical systems. The idea of using a monotonic continuation ODE TPBVP solver in conjunction with a predictor-corrector continuation method to advance (or “sweep”) as far along the tangent as possible is new, and this novel technique was used to obtain all the numerical results in this paper.

The paper is organized as follows. Subsections 1.2 and 1.3 review the specific types of rolling disk and ball considered, define coordinate systems and notation used to describe this rolling disk and ball, and present the uncontrolled equations of motion for this rolling disk and ball derived earlier in [60, 61]. In Sections 2 and 3, the controlled equations of motion for the rolling disk and ball are formulated and solved numerically via a predictor-corrector continuation method, starting from an initial solution provided by a direct method. Subsection 1.4 provides details of the numerical methods used to solve the controlled equations of motion. Section 4 summarizes the results of the paper and discusses topics for future work. The background material for this paper is contained in several appendices. In particular, Appendix A reviews the theory of optimal control needed to derive the controlled equations of motion for a generic dynamical system given initial and final conditions, given a performance index to be minimized, and in the absence of path inequality constraints. Appendix B provides details for numerically solving the controlled equations of motion. Appendices C and D develop two predictor-corrector continuation algorithms which numerically solve an ODE TPBVP, the latter of which is utilized to numerically solve the controlled equations of motion for the rolling disk and ball.

1.2. Rolling Ball. Consider a rigid ball of radius $r$ containing some static internal structure as well as $n$ point masses, where either $n$ is a positive integer denoting the number of moving masses or $n = 0$ if no moving masses are used and the structure of the ball is static. This ball rolls without slipping on a horizontal surface in the presence of a uniform gravitational field of magnitude $g$, as illustrated in Figure 1. The ball with its static internal structure has mass $m_0$ and the $i$th point mass has mass $m_i$ for $1 \leq i \leq n$. Let $M = \sum_{i=0}^{n} m_i$ denote the mass of the total system. The total mechanical system consisting of the ball with its static internal structure and the $n$ point masses is referred to as the ball or the rolling ball, the ball with its static internal structure but without the $n$ point masses may also be referred to as $m_0$, and the $i$th point mass may also be referred to as $m_i$ for $1 \leq i \leq n$. Note that this system is the Chaplygin ball [40] equipped with point masses.

Two coordinate systems, or frames of reference, will be used to describe the motion of the rolling ball, an inertial spatial coordinate system and a body coordinate system in which each particle within the ball is always fixed. For brevity, the spatial coordinate system will be referred to as the spatial frame and the body coordinate
Figure 1. A ball of radius $r$ and mass $m_0$ rolls without slipping on a horizontal surface in the presence of a uniform gravitational field of magnitude $g$. The ball’s geometric center, center of mass, and contact point with the horizontal surface are denoted by GC, $m_0$, and CP, respectively. The spatial frame has origin located at height $r$ above the horizontal surface and orthonormal axes $e_1$, $e_2$, and $e_3$. The body frame has origin located at the ball’s center of mass (denoted by $m_0$) and orthonormal axes $E_1$, $E_2$, and $E_3$. The ball’s motion is actuated by $n$ point masses, each of mass $m_i$, $1 \leq i \leq n$, and each moving along its own rail fixed inside the ball. The $i^{th}$ rail is depicted here by the dashed hoop. The trajectory of the $i^{th}$ rail, with respect to the body frame translated to the GC, is denoted by $\zeta_i$ and is parameterized by $\theta_i$. All vectors inside the ball are expressed with respect to the body frame, while all vectors outside the ball are expressed with respect to the spatial frame.
For $1 \leq i \leq n$, it is assumed that $m_i$ moves along its own 1-d rail. It is further assumed that the $i^{th}$ rail is parameterized by a 1-d parameter $\theta_i$, so that the trajectory $\zeta_i$ of the $i^{th}$ rail, in the body frame translated to the ball’s geometric center, as a function of $\theta_i$ is $\zeta_i(\theta_i)$. Refer to Figure 1 for an illustration. Therefore, the body frame vector from the ball’s geometric center to $m_i$’s center of mass is denoted by $\zeta_i(\theta_i(t))$. Since $m_0$ is stationary in the body frame and to be consistent with the positional notation for $m_i$ for $1 \leq i \leq n$, $\zeta_0 \equiv \zeta_0(\theta_0) \equiv \zeta_0(\theta_0(t))$ is the constant (time-independent) vector from the ball’s geometric center to $m_0$’s center of mass for any scalar-valued, time-varying function $\theta_0(t)$. In addition, suppose a time-varying external force $\mathbf{F}_e(t)$ acts at the ball’s geometric center.

Let $\mathbf{z}_i(t)$ denote the position of $m_i$’s center of mass in the spatial frame so that the position of $m_i$’s center of mass in the spatial frame is $\mathbf{z}_i(t) = \mathbf{z}_i(0) + \Lambda(t)\left[\zeta_i(\theta(t)) - \zeta_0\right]$. In general, a particle with position $\mathbf{w}(t)$ in the body frame has position $\mathbf{z}(t) = \mathbf{z}(0) + \Lambda(t)\mathbf{w}(t)$ in the spatial frame and has position $\mathbf{w}(t) + \zeta_0$ in the body frame translated to the ball’s geometric center.

For conciseness, the ball’s geometric center is often denoted GC, $m_0$’s center of mass is often denoted CM, and the ball’s contact point with the surface is often denoted CP. The GC is located at $\mathbf{z}_{GC}(t) = \mathbf{z}_0(t) - \Lambda(t)\zeta_0$ in the spatial frame, at $-\zeta_0$ in the body frame, and at $\mathbf{0} = [0 \ 0 \ 0]^T$ in the body frame translated to the GC. The CM is located at $\mathbf{z}_0(t)$ in the spatial frame, at $\mathbf{0}$ in the body frame, and at $\zeta_0$ in the body frame translated to the GC. The CP is located at $\mathbf{z}_{CP}(t) = \mathbf{z}_0(t) - \Lambda(t)[r\mathbf{\Gamma}(t) + \zeta_0]$ in the spatial frame, at $-r\mathbf{\Gamma}(t) + \zeta_0$ in the body frame, and at $-r\mathbf{\Gamma}(t)$ in the body frame translated to the GC, where $\mathbf{\Gamma}(t) = \Lambda^{-1}(t)e_3$. Since the third spatial coordinate of the ball’s GC is always $0$ and of the ball’s CP is always $-r$, only the first two spatial coordinates of the ball’s GC and CP, denoted by $\mathbf{z}(t)$, are needed to determine the spatial location of the ball’s GC and CP.

For succinctness, the explicit time dependence of variables is often dropped. That is, the orientation of the ball at time $t$ is denoted simply $\Lambda$ rather than $\Lambda(t)$, the position of $m_i$’s center of mass in the spatial frame at time $t$ is denoted $\mathbf{z}_i(t)$, rather than $\mathbf{z}_i(t)$, the position of $m_i$’s center of mass in the body frame translated to the GC at time $t$ is denoted $\zeta_i$ or $\zeta_i(\theta_i)$ rather than $\zeta_i(\theta_i(t))$, the spatial $e_1$-$e_2$ position of the ball’s GC at time $t$ is denoted $\mathbf{z}$ rather than $\mathbf{z}(t)$, and the external force is denoted $\mathbf{F}_e$ rather than $\mathbf{F}_e(t)$.

As shown in [60, 61], the uncontrolled equations of motion for this rolling ball are

$$\dot{\mathbf{z}} = (\mathbf{\Omega} \times \hat{\mathbf{e}}_3)_{12},$$

where $\mathbf{s}_i \equiv r\mathbf{\Gamma} + \zeta_i$ is the body frame vector from the CP to $m_i$ for $0 \leq i \leq n$, $\Omega \equiv \left(\Lambda^{-1}\dot{\Lambda}\right)^\top$ is the ball’s body angular velocity, $\mathbf{\Gamma} \equiv \Lambda^{-1}e_3$ is the spatial unit normal expressed in the body frame, and $\mathbf{\hat{e}} \equiv \Lambda^{-1}\mathbf{F}_e$ is the external force expressed in the body frame. For $\mathbf{v} = [v_1 \ v_2 \ v_3]^T \in \mathbb{R}^3$, $\mathbf{\hat{v}}^\top = \mathbf{\hat{v}}\mathbf{\hat{v}}$ is the symmetric matrix
given by
\[ \ddot{v}^2 = \begin{bmatrix} -(v_2^2 + v_3^2) & v_1 v_2 & v_1 v_3 \\ v_1 v_2 & -(v_1^2 + v_3^2) & v_2 v_3 \\ v_1 v_3 & v_2 v_3 & -(v_1^2 + v_2^2) \end{bmatrix} \] (4)
and \( \mathbf{v}_{12} \) is the projected vector consisting of the first two components of \( \mathbf{v} \) so that
\[ \mathbf{v}_{12} = [v_1 \ v_2]^T \in \mathbb{R}^2. \] (5)

Let \( N \) denote the magnitude of the normal force acting at the ball’s CP. Let \( f_s \) and \( \sigma \) denote the magnitude of and unit-length direction antiparallel to the static friction acting at the ball’s CP, respectively. As shown in [61], the magnitude of the normal force is
\[ N = Mg + \left( \sum_{i=0}^{n} m_i \left[ \mathbf{\Omega} \times \mathbf{s}_i + \mathbf{\Omega} \times \left( \mathbf{\Omega} \times \mathbf{\zeta}_i + 2\dot{\theta}_i \mathbf{\zeta}_i' \right) + \dot{\theta}_i^2 \mathbf{\zeta}_i'' + \ddot{\theta}_i \mathbf{\zeta}_i' \right], \mathbf{\Gamma} \right) - F_e,3 \] (6)
and the static friction is
\[ -f_s \sigma = \left[ \left( \Lambda \sum_{i=0}^{n} m_i \left[ \mathbf{\Omega} \times \mathbf{s}_i + \mathbf{\Omega} \times \left( \mathbf{\Omega} \times \mathbf{\zeta}_i + 2\dot{\theta}_i \mathbf{\zeta}_i' \right) + \dot{\theta}_i^2 \mathbf{\zeta}_i'' + \ddot{\theta}_i \mathbf{\zeta}_i' \right), \mathbf{\Gamma} \right] - F_e \right]_{12}. \] (7)

The dynamics encapsulated by (3) are valid only if the ball does not detach from the surface \((N > 0)\) and rolls without slipping \((\mu_s N \geq f_s)\), where \( \mu_s \) denotes the coefficient of static friction between the ball and the surface.

Figure 2. A disk of radius \( r \) and mass \( m_0 \) rolls without slipping in the \( e_1-e_3 \) plane. \( e_2 \) and \( E_2 \) are directed into the page and are omitted from the figure. The disk’s center of mass is denoted by \( m_0 \). The disk’s motion is actuated by \( n \) point masses, each of mass \( m_i \), \( 1 \leq i \leq n \), and each moving along its own rail fixed inside the disk. The point mass depicted here by \( m_i \) moves along a circular hoop in the disk that is not centered on the disk’s geometric center (GC). The disk’s orientation is determined by \( \phi \), the angle measured counterclockwise from \( e_1 \) to \( E_1 \).

1.3. Rolling Disk. Now suppose that the ball’s inertia is such that one of the ball’s principal axes, say the one labeled \( E_2 \), is orthogonal to the plane containing the GC and CM. Also assume that all the point masses move along 1-d rails which lie in the plane containing the GC and CM. Moreover, suppose that the ball is oriented initially so that the plane containing the GC and CM coincides with the \( e_1-e_3 \) plane and that the external force \( \mathbf{F}_e \) acts in the \( e_1-e_3 \) plane. Then for all time, the ball will remain oriented so that the plane containing the GC and CM coincides with the \( e_1-e_3 \) plane and the ball will only move in the \( e_1-e_3 \) plane, with the ball’s rotation
axis always parallel to $e_2$. Note that the dynamics of this system are equivalent to that of the Chaplygin disk [40], equipped with point masses, rolling in the $e_1$-$e_3$ plane, and where the Chaplygin disk (minus the point masses) has polar moment of inertia $d_2$. Therefore, henceforth, this particular ball with this special inertia, orientation, and placement of the rails and point masses, may be referred to as the disk or the rolling disk. Figure 2 depicts the rolling disk. Let $\phi$ denote the angle between $e_1$ and $E_1$, measured counterclockwise from $e_1$ to $E_1$. Thus, if $\phi > 0$, the disk rolls in the $-e_1$ direction and $\Omega$ has the same direction as $-e_2$, and if $\phi < 0$, the disk rolls in the $e_1$ direction and $\Omega$ has the same direction as $e_2$.

As shown in [60, 61], the uncontrolled equation of motion for this rolling disk is

$$
\dot{\phi} = \frac{-r F_{e,1} + \sum_{i=0}^{n} m_i K_i}{d_2 + \sum_{i=0}^{n} m_i \left( r \sin \phi + \zeta_{i,1} \right)^2 + \left( r \cos \phi + \zeta_{i,3} \right)^2} \equiv \kappa \left( t, \theta, \phi, \dot{\phi}, \dot{\theta} \right), \tag{8}
$$

where

$$
K_i = \left( g + r \dot{\phi}^2 \right) \left( \zeta_{i,3} \sin \phi - \zeta_{i,1} \cos \phi \right) + \left( r \cos \phi + \zeta_{i,3} \right) \left( -2 \dot{\phi} \dot{\phi}_i \zeta_{i,3} + \ddot{\phi}_i \zeta_{i,1} \right) \tag{9} + \left( r \sin \phi + \zeta_{i,1} \right) \left( 2 \dot{\phi} \dot{\phi}_i \zeta_{i,1} + \ddot{\phi}_i \zeta_{i,3} \right).
$$

In (8), $\kappa$ is a function that depends on time ($t$) through the possibly time-varying external force $F_{e,1}(t)$, on the point mass parameterized positions ($\theta$), velocities ($\dot{\theta}$), and accelerations ($\ddot{\theta}$), and on the disk’s orientation angle ($\phi$) and its time-derivative ($\dot{\phi}$). The spatial $e_1$ position $z$ of the disk’s GC is given by

$$
z = z_a - r (\phi - \phi_a), \tag{10}
$$

where $z_a$ is the spatial $e_1$ position of the disk’s GC at time $t = a$ and $\phi_a$ is the disk’s angle at time $t = a$. Let $N$ denote the magnitude of the normal force acting at the disk’s CP. Let $f_s$ and $\sigma$ denote the magnitude of and unit-length direction antiparallel to the static friction acting at the disk’s CP, respectively. As shown in [61], the magnitude of the normal force is

$$
N = Mg + \sum_{i=0}^{n} m_i \left[ \left( -\ddot{\phi}_i \zeta_{i,3} - \dot{\phi}^2 \zeta_{i,1} - 2 \dot{\phi} \dot{\phi}_i \zeta_{i,3} + \ddot{\phi}_i \zeta_{i,1} \right) \sin \phi \right. \tag{11}
$$

$$
\left. + \left( \ddot{\phi}_i \zeta_{i,1} - \dot{\phi}^2 \zeta_{i,3} + 2 \dot{\phi} \dot{\phi}_i \zeta_{i,1} + \ddot{\phi}_i \zeta_{i,3} \right) \cos \phi \right] - F_{e,3}
$$

and the static friction is

$$
-f_s \sigma = -\left\{ Mg \ddot{\phi} + \sum_{i=0}^{n} m_i \left[ \left( \ddot{\phi}_i \zeta_{i,3} + \dot{\phi}^2 \zeta_{i,1} + 2 \dot{\phi} \dot{\phi}_i \zeta_{i,3} - \ddot{\phi}_i \zeta_{i,1} \right) \cos \phi \right. \tag{12}
$$

$$
\left. + \left( \ddot{\phi}_i \zeta_{i,1} - \dot{\phi}^2 \zeta_{i,3} + 2 \dot{\phi} \dot{\phi}_i \zeta_{i,1} + \ddot{\phi}_i \zeta_{i,3} \right) \sin \phi \right] + F_{e,1} \right\} e_1.
$$

The dynamics encapsulated by (8) are valid only if the disk does not detach from the surface ($N > 0$) and rolls without slipping ($\mu_s N \geq f_s$), where $\mu_s$ denotes the coefficient of static friction between the disk and the surface.
1.4. Numerical Methods. In Sections 2 and 3, the motions of the rolling disk and ball are simulated in MATLAB R2019b by numerically solving the controlled equations of motion (28) and (56) corresponding to the optimal control problems (19) and (37) for the rolling disk and ball, respectively. Subsection 2.2 simulates the rolling disk, while Subsections 3.2 and 3.3 simulate the rolling ball. Because the controlled equations of motion have a very small radius of convergence [17, 16, 11], a direct method, namely the MATLAB toolbox GPOPS-II [57] version 2.5, is first used to construct a good initial guess. In these simulations, GPOPS-II is configured to use the NLP solver SNOPT [36, 35] version 7.6.0, though GPOPS-II can also be configured to use the NLP solver IPOPT [67]. For the rolling disk, the direct method is used to solve the rolling disk optimal control problem (19). When using the direct method to solve the rolling ball optimal control problem, the differential-algebraic equation (DAE) formulation (52) is solved first. The direct method solution to the DAE formulation is then used as an initial guess to solve the ODE formulation (37), by the direct method. The MATLAB automatic differentiation toolbox ADiGator [70, 69] version 1.5 is used to supply vectorized first derivatives (i.e. Jacobians) to the direct method solver GPOPS-II, since SNOPT accepts first, but not second, derivatives.

Starting from the initial guess provided by the direct method, the controlled equations of motion (28) and (56) are solved by predictor-corrector continuation in the parameter $\mu$, utilizing the algorithm described in Appendix D. The predictor-corrector continuation method uses the MATLAB global method ODE TPBVP solvers sbvp [6] version 1.0 or bvptwp [19] version 1.0. By vectorized automatic differentiation of $H_x$, $\pi$, and $f$, ADiGator is used to numerically construct the Jacobians of the normalized ODE velocity function (86) and (87). By non-vectorized automatic differentiation of the Hamiltonian $H$, the initial condition function $\sigma$, the final condition function $\psi$, and the endpoint function $G$, ADiGator is used to numerically construct the normalized BC function (121) and the Jacobians of the normalized BC function (123), (124), and (125). These functions are needed by the ODE TP-BVP solvers sbvp and bvptwp to solve the controlled equations of motion (28) and (56) by predictor-corrector continuation in the parameter $\mu$.

In contrast to the direct method, the controlled equations of motion obtained via the indirect method have a very small radius of convergence [17, 16, 11]. Therefore, the direct method is needed to initialize the predictor-corrector continuation of the controlled equations of motion. Predictor-corrector continuation is used in conjunction with the indirect, rather than direct, method, because a predictor-corrector continuation direct method requires a predictor-corrector continuation NLP solver. Even though predictor-corrector continuation NLP solver algorithms are provided in [71, 48], there do not seem to be any publicly available predictor-corrector continuation NLP solvers.

2. Trajectory Tracking for the Rolling Disk.

2.1. Optimal Control Problem and Controlled Equations of Motion. In the next subsection, numerical solutions of the controlled equations of motion for the rolling disk are presented, where the goal is to move the disk between a pair of points while the disk’s GC tracks a prescribed trajectory. A rolling disk of mass $m_0 = 4$, radius $r = 1$, polar moment of inertia $d_2 = 1$, and with the CM coinciding with the GC (i.e. $\zeta_0 = 0$) is simulated. These physical parameters for the disk are
consistent with the necessary and sufficient conditions stipulated by Inequality 3.2 in [63]; in that inequality, note that \(d_1 + d_3 = d_2 = 1\) since the disk is a planar distribution of mass. There are \(n = 4\) control masses, each of mass 1 so that \(m_1 = m_2 = m_3 = m_4 = 1\), located on concentric circular control rails centered on the GC of radii \(r_1 = .9\), \(r_2 = .63\), \(r_3 = .36\), and \(r_4 = .1\), as shown in Figure 3. For \(1 \leq i \leq 4\), the position of \(m_i\) in the body frame centered on the GC is

\[
\zeta_i(\theta_i) = r_i \begin{bmatrix} \cos \theta_i \\ 0 \\ \sin \theta_i \end{bmatrix}.
\]  
(13)

**Figure 3.** The disk of radius \(r = 1\) actuated by 4 control masses, \(m_1, m_2, m_3,\) and \(m_4\), each on its own circular control rail. The control rail radii are \(r_1 = .9, r_2 = .63, r_3 = .36,\) and \(r_4 = .1\). The location of the disk’s CM is denoted by \(m_0\).

The total mass of the system is \(M = 8\), and gravity is \(g = 9.81\). There is no external force acting on the disk’s GC, so that \(F_{e,1} = 0\) in (8). The initial time is fixed to \(a = 0\) and the final time is fixed to \(b = 12\). The disk’s GC starts at rest at \(z_a = 0\) at time \(a = 0\) and stops at rest at \(z_b = 1\) at time \(b = 12\). Table 1 shows parameter values used in the rolling disk’s initial conditions (22) and final conditions (23). Since the initial orientation of the disk is \(\phi_a = 0\) and since the initial configurations of the control masses are given by \(\theta_a = [\begin{array}{cccc} -\pi/2 & -\pi/2 & -\pi/2 & -\pi/2 \end{array}]^\top\), all the control masses are initially located directly below the GC. In order for the disk to start and stop at rest, \(\dot{\theta}_a = \dot{\theta}_b = [0\ 0\ 0\ 0]^\top\) and \(\dot{z}_a = \dot{z}_b = 0\). Table 2 shows parameter values used in the rolling disk’s final conditions (23).

The desired GC path \(z_d\) is depicted by the red curve in Figures 4a and 4b. \(z_d\) encourages the disk’s GC to track a sinusoidally-modulated linear trajectory connecting \(z(0) = 0\) with \(z(12) = 1\). That is, the disk is encouraged to roll right, then left, then right, then left, and finally to the right, with the amplitude of each successive roll increasing from the previous one. Specifically, \(z_d\) is given by

\[
z_d(t) \equiv [z_a w(t) + \tilde{z}_d(t)(1 - w(t))] \left[1 - y(t)\right] + z_b g(t),
\]  
(14)
Table 1. Initial condition parameter values for the rolling disk. Refer to (22) and (23).

| Parameter | Value |
|-----------|-------|
| $\theta_a$ | $[-\frac{\pi}{2}, -\frac{\pi}{2}, -\frac{\pi}{2}, -\frac{\pi}{2}]^T$ |
| $\dot{\theta}_a$ | $[0, 0, 0, 0]^T$ |
| $\phi_a$ | 0 |
| $z_a$ | 0 |
| $\dot{z}_a$ | 0 |

Table 2. Final condition parameter values for the rolling disk. Refer to (23).

| Parameter | Value |
|-----------|-------|
| $\dot{\theta}_b$ | $[0, 0, 0, 0]^T$ |
| $z_b$ | 1 |
| $\dot{z}_b$ | 0 |

where

$$S(t) \equiv \frac{1}{2} \left[ 1 + \tanh \left( \frac{-t}{\epsilon} \right) \right],$$ (15)

$$w(t) \equiv S(t-a),$$ (16)

$$y(t) \equiv S(-t+b),$$ (17)

and

$$\ddot{z}_d(t) \equiv \left[ z_a + (z_b - z_a) \frac{t-a}{b-a} \right] \sin \left( \frac{9\pi}{2} \frac{t-a}{b-a} \right),$$ (18)

with $\epsilon = .01$ in (15). The reader is referred to [58] for further details about the properties and construction of (14).

The optimal control problem for the rolling disk is

$$\min_u J \text{ s.t. } \begin{cases} \dot{x} = f(x, u), \\ \sigma(x(a)) = 0, \\ \psi(x(b)) = 0. \end{cases}$$ (19)

In (19), the system state $x$ and control $u$ are

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix} \quad \text{and} \quad u = \ddot{\theta},$$ (20)

where $\theta, \dot{\theta}, \ddot{\theta} \in \mathbb{R}^4$ and $\phi, \dot{\phi} \in \mathbb{R}$. In (19), the system dynamics defined for $a \leq t \leq b$ are

$$\dot{x} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix} = f(x, u) \equiv \begin{bmatrix} \dot{\theta} \\ u \\ \dot{\phi} \end{bmatrix},$$ (21)
where $\kappa (x, u)$ is given by the right-hand side of (8). In (21), the time-dependence of $\kappa$ is dropped since $F_{v,1} = 0$ in (8) for these simulations. In (19), the prescribed initial conditions at time $t = a$ are

$$\sigma (x(a)) = \begin{bmatrix} \theta(a) - \theta_a \\ \theta(a) - \dot{\theta}_a \\ \phi(a) - \phi_a \\ -r\phi(a) - \dot{\gamma}_a \end{bmatrix} = 0, \quad (22)$$

and the prescribed final conditions at time $t = b$ are

$$\psi (x(b)) = \begin{bmatrix} \Pi \left( \hat{\Lambda}(\phi(b)) \left[ \frac{1}{M} \sum_{i=0}^{4} m_i \xi_i (\theta_i(b)) \right] \right) \\ \theta(b) - \dot{\theta}_b \\ z_a - r(\phi(b) - \phi_a) - z_b \\ -r\phi(b) - \dot{\gamma}_b \end{bmatrix} = 0. \quad (23)$$

In (23),

$$\frac{1}{M} \sum_{i=0}^{4} m_i \xi_i (\theta_i(t)) \quad (24)$$

is the total system CM expressed in the body frame translated to the disk’s GC at time $t$,

$$\hat{\Lambda}(\phi(t)) = \Lambda(t) = \begin{bmatrix} \cos \phi(t) & 0 & -\sin \phi(t) \\ 0 & 1 & 0 \\ \sin \phi(t) & 0 & \cos \phi(t) \end{bmatrix} \quad (25)$$

is the rotation matrix that maps the body to spatial frame at time $t$, and $\Pi$ is the projection onto the first component. Therefore, the first constraint in (23) ensures that the total system CM is above or below the disk’s GC in the spatial frame at the final time $t = b$, so that, in conjunction with the final condition parameter values given in Table 2, the disk stops at rest. In (19), the performance index is

$$J \equiv \int_a^b L (t, x, u, \mu) \, dt = \int_a^b \left[ \frac{\alpha(\mu)}{2} (z_a - r(\phi - \phi_a) - z_d) + \sum_{i=1}^{4} \frac{\gamma_i}{2} \theta_i^2 \right] \, dt, \quad (26)$$

where the integrand cost function is

$$L (t, x, u, \mu) = \frac{\alpha(\mu)}{2} (z_a - r(\phi - \phi_a) - z_d)^2 + \sum_{i=1}^{4} \frac{\gamma_i}{2} \theta_i^2, \quad (27)$$

for positive coefficients $\alpha(\mu)$ and $\gamma_i, \quad 1 \leq i \leq 4$. The first summand $\frac{\alpha(\mu)}{2} (z_a - r(\phi - \phi_a) - z_d)^2$ in $L$ encourages the disk’s GC to track the desired spatial $e_1$ path $z_d$, and $\mu$ is a scalar continuation parameter used to construct a sequence of optimal control problems. The next 4 summands $\frac{\gamma_i}{2} \theta_i^2, \quad 1 \leq i \leq 4$, in $L$ limit the magnitude of the acceleration of the $i^{th}$ control mass parameterization.

Table 3 shows the values set for the integrand cost function coefficients in (27).

As explained in Appendix A, the controlled equations of motion for the rolling disk’s optimal control problem (19) are encapsulated by the ODE TPBVP:

$$\dot{x} = \hat{H}_x (t, x, \lambda, \mu) = \tilde{f}(x, \lambda) = \left. \tilde{f}(x, \pi (x, \lambda)) \right|_{\mu},$$

$$\dot{\lambda} = -\hat{H}_{xx}^T (t, x, \lambda, \mu) = -H_x^T (t, x, \lambda, \pi (x, \lambda), \mu),$$

$$\lambda|_{t=a} = -G_x^T (x(a)), \quad G_{xx}^T = \sigma (x(a)) = 0,$$

$$\lambda|_{t=b} = G_x^T (x(b)), \quad G_{xx}^T = \psi (x(b)) = 0. \quad (28)$$
Subappendix A.1 of [58] derives the formulas for constructing $H^T_x$. In (28), $G$ is the endpoint function

$$G(x(a), \xi, x(b), \nu) \equiv \xi^T \sigma(x(a)) + \nu^T \psi(x(b))$$

$$= \xi^T \begin{bmatrix} \theta(a) - \theta_a & 0 & 0 & 0 & 0 & \phi(a) - \phi_a & 0 & -r\phi(a) - z_a \end{bmatrix}$$

$$+ \nu^T \begin{bmatrix} \Pi \left( \Lambda \phi(b) \right) \left[ \frac{1}{M} \sum_{i=0}^4 m_i \zeta_i(\theta(b)) \right] \\ \theta(b) - \theta_b \\ z_a - r(\phi(b) - \phi_a) - z_b \\ -r\phi(b) - z_b \end{bmatrix},$$

where $\xi \in \mathbb{R}^{10}$ and $\nu \in \mathbb{R}^7$ are constant Lagrange multiplier vectors enforcing the initial and final conditions, (22) and (23), respectively. In (28), $H$ is the Hamiltonian

$$H(t, x, \lambda, u, \mu) \equiv L(t, x, u, \mu) + \lambda^T f(x, u)$$

$$= \frac{\alpha(\mu)}{2} (z_a - r(\phi - \phi_a) - z_d)^2 + \sum_{i=1}^4 \frac{\gamma_i}{2} \dot{\theta}_i^2 + \lambda^T \begin{bmatrix} \dot{\theta} & u & \phi & \kappa(x, u) \end{bmatrix},$$

where $\lambda \in \mathbb{R}^{10}$ is a time-varying Lagrange multiplier vector enforcing the dynamics (21). In (28), $\pi$ is an analytical formula expressing the control $u$ as a function of the state $x$ and the costate $\lambda$. The components of $\pi$ are given by

$$\ddot{\pi}_i = \pi_i(x, \lambda) \equiv -\gamma_i^{-1} \times$$

$$\begin{cases} 
\lambda_{4+i} + \lambda_{10} \frac{m_i \left[ (r \cos \phi + \zeta_{i,3}) \zeta_{i,1}' - (r \sin \phi + \zeta_{i,1}) \zeta_{i,3}' \right]}{d_2 + \sum_{k=0}^4 m_k \left[ (r \sin \phi + \zeta_{k,1})^2 + (r \cos \phi + \zeta_{k,3})^2 \right]}, 
\end{cases}$$

for $1 \leq i \leq 4$. In (28), $\dot{H}$ is the regular Hamiltonian

$$\dot{H}(t, x, \lambda, \mu) \equiv H(t, x, \lambda, \pi(x, \lambda), \mu)$$

$$= \frac{\alpha(\mu)}{2} (z_a - r(\phi - \phi_a) - z_d)^2$$

$$+ \sum_{i=1}^4 \frac{\gamma_i}{2} \pi_i^2(x, \lambda) + \lambda^T \begin{bmatrix} \dot{\theta} & \pi(x, \lambda) & \kappa(x, \pi(x, \lambda)) \end{bmatrix}. $$

**Table 3.** Integrand cost function coefficient values for the rolling disk when predictor-corrector continuation is performed in $\alpha$. Refer to (27).
The reader is referred to [58] for a more general description of the rolling disk’s optimal control problem (19) and the associated controlled equations of motion (28).

2.2. Numerical Solutions: Trajectory Tracking. The direct method solver GPOPS-II is used to solve the optimal control problem (19) when the integrand cost function coefficient is \( \alpha = 1 \). Predictor-corrector continuation is then used to solve the controlled equations of motion (28), starting from the direct method solution. The continuation parameter is \( \mu \), which is used to adjust \( \alpha \) according to the linear homotopy given in Table 3, so that \( \alpha = 1 \) when \( \mu = .95 \) and \( \alpha \approx 272 \) when \( \mu \approx .8983 \). The predictor-corrector continuation begins at \( \mu = .95 \), which is consistent with the direct method solution obtained at \( \alpha = 1 \).

For the direct method, GPOPS-II is run using the NLP solver SNOPT. The GPOPS-II mesh error tolerance is \( 1e^{-6} \) and the SNOPT error tolerance is \( 1e^{-7} \). In order to encourage convergence of SNOPT, a constant \( C = 50 \) is added to the integrand cost function \( L \) in (27). The sweep predictor-corrector continuation method discussed in Appendix D is used by the indirect method. For the sweep predictor-corrector continuation method, the maximum tangent steplength \( \sigma_{\text{max}} \) is adjusted according to Figure 6d over the course of 6 predictor-corrector steps, the maximum tangent steplength in each step is \( \sigma_{\text{max}} = [40 \ 40 \ 40 \ 5 \ 1 \ 1] \), the direction of the initial unit tangent is determined by setting \( d = -2 \) to force the continuation parameter \( \mu \) to initially decrease, the relative error tolerance is \( 1e^{-8} \), the unit tangent solver is \( \text{tpvbpvpc.m} \), and the monotonic “sweep” continuation solver is \( \text{acdcc} \). The numerical results are shown in Figures 4, 5, and 6. As \( \mu \) decreases from .95 down to .8983 during continuation (see Figure 6a), \( \alpha \) increases from .1 up to 272 (see Figure 6b). Since \( \alpha \) is ratcheted up during continuation, thereby increasing the penalty in the integrand cost function (27) for deviation between the disk’s GC and \( z_d \), by the end of continuation, the disk’s GC tracks \( z_d \) very accurately (compare Figures 4a vs 4b), at the expense of more serpentine control mass trajectories (compare Figures 4c vs 4d) and larger magnitude controls (compare Figures 4e vs 4f). The disk does not detach from the surface since the magnitude of the normal force is always positive (see Figures 5a and 5b). The disk rolls without slipping if the coefficient of static friction \( \mu_s \) is at least \( \hat{\mu}_s \approx .07799 \) for the direct method solution (see Figure 5c) and if \( \mu_s \) is at least \( \hat{\mu}_s \approx .3502 \) for the indirect method solution (see Figure 5d). That is, the indirect method solution requires a much larger coefficient of static friction.

2.2.1. Turning Points. The previous sweep predictor-corrector continuation indirect method is repeated, but this time with 22 steps, where the maximum tangent steplength in each step is

\[
\sigma_{\text{max}} = [40 \ 40 \ 40 \ 5 \ 1 \ 1 \ 60 \ 2 \ 5 \ 2 \ 2 \ 5 \ 10 \ 10 \ 5 \ .1 \ .1 \ 1 \ 10 \ 10 \ 10].
\] (33)

Note that the first 6 maximum tangent steplengths in (33) agree with those used in the previous simulation, so that the two simulations agree for the first 7 solutions. Figure 7 shows the evolution of the continuation parameter \( \mu \), GC path weighting factor \( \alpha \), performance index \( J \), GC tracking error \( \|z - z_d\|^2 \), and tangent steplength \( \sigma \) over the course of the 23 solutions (the first solution is initialized by the direct method) constructed by the sweep predictor-corrector continuation indirect method. Note the turning points (local maxima or minima) at solutions 7, 10, and 18 in Figures 7a-7d. Figure 7d shows that the GC tracking error realizes a minimum at
The GC tracks the desired path crudely when $\alpha = 0.1$.

The GC tracks the desired path very accurately when $\alpha \approx 2.72$.

The motions of the center of masses are modest when $\alpha = 0.1$.

The motions of the center of masses are more serpentine when $\alpha \approx 2.72$.

The controls have relatively small magnitudes when $\alpha = 0.1$.

The controls have relatively larger magnitudes when $\alpha \approx 2.72$.

Figure 4. Numerical solutions of the rolling disk optimal control problem (19) using 4 control masses for $\gamma_1 = \gamma_2 = \gamma_3 = \gamma_4 = 0.1$ and for fixed initial and final times. The direct method results for $\alpha = 0.1$ are shown in the left column, while the predictor-corrector continuation indirect method results for $\alpha \approx 2.72$ are shown in the right column. The direct method solution tracks the desired GC path crudely, whereas the indirect method solution tracks the desired GC path very accurately at the expense of larger magnitude controls.
Figure 5. Numerical solutions of the rolling disk optimal control problem (19) using 4 control masses for $\gamma_1 = \gamma_2 = \gamma_3 = \gamma_4 = .1$ and for fixed initial and final times. The direct method results for $\alpha = .1$ are shown in the left column, while the predictor-corrector continuation indirect method results for $\alpha \approx 272$ are shown in the right column. The disk does not detach from the surface since the magnitude of the normal force is always positive. The disk rolls without slipping if $\mu_s \geq .07799$ for the direct method solution and if $\mu_s \geq .3502$ for the indirect method solution. That is, the indirect method solution requires a much larger coefficient of static friction.

solution 7, which explains how the stopping point for the previous simulation was selected.

3. Obstacle Avoidance for the Rolling Ball.

3.1. Optimal Control Problem and Controlled Equations of Motion. In the next two subsections, numerical solutions of the controlled equations of motion for the rolling ball are presented, where the goal is to move the ball between a
Figure 6. Evolution of various parameters and variables during the predictor-corrector continuation indirect method, which starts from the direct method solution, used to solve the rolling disk optimal control problem (19). \( \mu \) decreases monotonically, while \( \alpha \) and \( J \) increase monotonically.

pair of points while the ball’s GC avoids a pair of obstacles. A rolling ball of mass \( m_0 = 4 \), radius \( r = 1 \), principal moments of inertia \( d_1 = d_2 = d_3 = 1 \), and with the CM coinciding with the GC (i.e. \( \zeta_0 = 0 \)) is simulated. These physical parameters for the ball are consistent with the necessary and sufficient conditions stipulated by Inequalities 3.1 and 3.2 in [63]. There are \( n = 3 \) control masses, each of mass 1 so that \( m_1 = m_2 = m_3 = 1 \), located on circular control rails centered on the GC of radii \( r_1 = .95 \), \( r_2 = .9 \), and \( r_3 = .85 \), oriented as shown in Figure 8. For \( 1 \leq i \leq 3 \), the position of \( m_i \) in the body frame centered on the GC is

\[
\zeta_i (\theta_i) = r_i B_i (\varsigma (v_i)) \begin{bmatrix} \cos \theta_i \\ 0 \\ \sin \theta_i \end{bmatrix},
\]

where \( B_i (n) \in SO(3) \) is a rotation matrix whose columns are the right-handed orthonormal basis constructed from the unit vector \( n \in \mathbb{R}^3 \) based on the algorithm.
(a) Evolution of the continuation parameter $\mu$.

(b) Evolution of the GC path weighting factor $\alpha$.

(c) Evolution of the performance index $J$.

(d) Evolution of the GC tracking error, measured by the square of the $L^2$ norm.

(e) Evolution of the tangent steplength $\sigma$.

**Figure 7.** Evolution of various parameters and variables during an extended run of the predictor-corrector continuation indirect method, which starts from the direct method solution, used to solve the rolling disk optimal control problem (19). Note the turning points at solutions 7, 10, and 18. The minimum of the GC tracking error occurs at solution 7.
given in Section 4 and Listing 2 of [34], \( \varsigma : \mathbb{R}^3 \to \mathbb{R}^3 \) maps spherical coordinates to Cartesian coordinates:

\[
\varsigma \left( \begin{bmatrix} \phi \\ \theta \\ \rho \end{bmatrix} \right) = \begin{bmatrix} \rho \cos \theta \cos \phi \\ \rho \cos \theta \sin \phi \\ \rho \sin \theta \end{bmatrix}, \tag{35}
\]

and

\[
\mathbf{v}_1 = [0 \ 0 \ 1]^T, \quad \mathbf{v}_2 = \left[ \frac{\pi}{4} \ 0 \ 1 \right]^T, \quad \text{and} \quad \mathbf{v}_3 = \left[ \frac{\pi}{4} \ \frac{\pi}{4} \ 1 \right]^T \tag{36}
\]

are spherical coordinates of unit vectors in \( \mathbb{R}^3 \).

**Ball, Control Masses, and Control Rails**

in the Body Frame Translated to the GC

![Diagram of ball and control rails](image)

**Figure 8.** The ball of radius \( r = 1 \) actuated by 3 control masses, \( m_1, m_2, \) and \( m_3 \), each on its own circular control rail. The control rail radii are \( r_1 = .95, r_2 = .9, \) and \( r_3 = .85 \). The location of the ball’s CM is denoted by \( m_0 \).

The total mass of the system is \( M = 7 \), and gravity is \( g = 9.81 \). There is no external force acting on the ball’s GC, so that \( \mathbf{\Gamma} = \Lambda^{-1} \mathbf{F}_e = \mathbf{0} \) in (3). The initial time is fixed to \( a = 0 \) and the final time is fixed to \( b = 5 \). The ball’s GC starts at rest at \( \mathbf{z}_a = [0 \ 0]^T \) at time \( a = 0 \) and stops at rest at \( \mathbf{z}_b = [1 \ 1]^T \) at time \( b = 5 \). The ball’s GC should avoid 2 circular obstacles, depicted in Figures 9a and 9b. The obstacles each have radius \( \rho_1 = \rho_2 = .282 \) and are centered at \( \mathbf{v}_1 = [.2 \ .2]^T \) and \( \mathbf{v}_2 = [.8 \ .8]^T \).

The ODE formulation of the optimal control problem for the rolling ball is

\[
\min_u J \quad \text{s.t.} \begin{cases} 
\dot{x} = f(x, u), \\
\sigma(x(a)) = 0, \\
\psi(x(b)) = 0. 
\end{cases} \tag{37}
\]

In (37), the system state \( x \) and control \( u \) are

\[
x \equiv \begin{bmatrix} \theta \\ \dot{\theta} \\ q \\
\Omega \\ z \end{bmatrix} \quad \text{and} \quad u \equiv \dot{\theta}, \tag{38}
\]
where \( \theta, \dot{\theta}, \ddot{\theta} \in \mathbb{R}^3, q \in \mathbb{S} \subset \mathbb{R}^4, \Omega \in \mathbb{R}^3 \), and \( z \in \mathbb{R}^2 \). In the system state defined in (38), the rolling ball’s orientation matrix \( \Lambda \in SO(3) \) is parameterized by \( q \in \mathbb{S} \), where \( \mathbb{S} \) denotes the set of versors (i.e. unit quaternions) \([40, 37, 66, 9]\). The properties of versors and the notation used to manipulate versors are explained in Appendix D of [60]. Recall from [60] that given a column vector \( v \in \mathbb{R}^3 \), \( v^\sharp \) is the quaternion \( v^\sharp = [0 \ v] \) \((39)\) and given a quaternion \( p \in \mathbb{H} \), \( p^\flat \) is the column vector such that \( p = [p_0 \ p^\flat] \) \((40)\).

As explained in [60], the transformation of a body frame vector \( Y \in \mathbb{R}^3 \) into the spatial frame by the ball’s orientation matrix \( \Lambda \in SO(3) \) can be realized using the versor \( q \in \mathbb{S} \) via the Euler-Rodrigues formula \( \Lambda Y = [qY^\sharp q^{-1}]^\flat \) \((41)\).

In (37), the system dynamics defined for \( a \leq t \leq b \) are
\[
\dot{x} = \begin{bmatrix} \dot{\theta} \\ \dot{\hat{q}} \\ \dot{\Omega} \\ \dot{z} \end{bmatrix} = f(x, u) \equiv \begin{bmatrix} \dot{\theta} \\ u \\ \frac{1}{2} q\Omega^\sharp \\ \kappa(x, u) \end{bmatrix},
\]
where \( \kappa(x, u) \) is given by the right-hand side of the formula for \( \dot{\Omega} \) in (3). In (42), the time-dependence of \( \kappa \) is dropped since \( \tilde{\Gamma} = \Lambda^{-1} F_e = 0 \) in (3) for these simulations.

In (37), the prescribed initial conditions at time \( t = a \) are
\[
\sigma(x(a)) \equiv \begin{bmatrix} \theta(a) - \theta_a \\ \dot{\theta}(a) - \dot{\theta}_a \\ q(a) - q_a \\ \Omega(a) - \Omega_a \\ z(a) - z_a \end{bmatrix} = 0,
\]
and the prescribed final conditions at time \( t = b \) are
\[
\psi(x(b)) \equiv \Pi \left( \left[ \frac{1}{M} \sum_{i=0}^{3} m_i \zeta_i (\theta_i(t)) \right]^\flat \right)^\flat = 0.
\]

Table 4 shows the parameter values used in the rolling ball’s initial conditions \((43)\). The initial configurations of the control masses are selected so that the total system CM in the spatial frame is initially located above or below the ball’s GC. Hence, in conjunction with the other initial condition parameter values given in Table 4, the ball starts at rest. In (44),
\[
\frac{1}{M} \sum_{i=0}^{3} m_i \zeta_i (\theta_i(t))
\]
is the total system CM in the body frame translated to the ball’s GC at time \( t \),

\[
\begin{bmatrix}
q(t) \\
\frac{1}{M} \sum_{i=0}^{3} m_i \zeta_i (\theta_i(t))
\end{bmatrix}^2 = \Lambda(t) \begin{bmatrix}
\frac{1}{M} \sum_{i=0}^{3} m_i \zeta_i (\theta_i(t))
\end{bmatrix}
\]

(46)

is the total system CM in the spatial frame translated to the ball’s GC at time \( t \), and \( \Pi \) is the projection onto the first two components. Therefore, the first two constraints in (44) ensure that the total system CM in the spatial frame is above or below the ball’s GC at the final time \( t = b \). Hence, in conjunction with the final condition parameter values given in Table 5, the ball stops at rest. In (37), the performance index is

\[
J = \int_a^b L(x, u, \mu) \, dt = \int_a^b \left[ \sum_{i=1}^{3} \frac{\gamma_i(\mu)}{2} \ddot{\theta}_i^2 + \sum_{j=1}^{2} h_j(\mu) S(|z - v_j| - \rho_j) \right] \, dt,
\]

(47)

where the integrand cost function is

\[
L(x, u, \mu) \equiv \sum_{i=1}^{3} \frac{\gamma_i(\mu)}{2} \ddot{\theta}_i^2 + \sum_{j=1}^{2} h_j(\mu) S(|z - v_j| - \rho_j),
\]

(48)

for positive coefficients \( \gamma_i(\mu), 1 \leq i \leq 3 \), and \( h_j(\mu), 1 \leq j \leq 2 \). The first 3 summands \( \frac{\gamma_i(\mu)}{2} \ddot{\theta}_i^2 \), \( 1 \leq i \leq 3 \), in \( L \) limit the magnitude of the acceleration of the \( i^{\text{th}} \) control mass parameterization and the final 2 summands \( h_j(\mu) S(|z - v_j| - \rho_j) \), \( 1 \leq j \leq 2 \), in \( L \) encourage the ball’s GC to avoid the pair of obstacles. For the obstacle avoidance function in \( L \), \( S \) is either the time-reversed sigmoid function (15) or the \( C^2 \) cutoff function

\[
S(y) \equiv [\max \{0, -y\}]^4 = \text{ReLU}^4(-y).
\]

(49)

In (49), ReLU is the rectified linear unit function frequently used in the machine learning literature [50]. In (48), the coefficients \( \gamma_i(\mu), 1 \leq i \leq 3 \), and \( h_j(\mu), 1 \leq j \leq 2 \), depend on the scalar continuation parameter \( \mu \) so that a sequence of optimal control problems may be constructed. Note that a solution obtained by the optimal control procedure, that minimizes (26) for the rolling disk or (47) for the rolling ball, is a “compromise” between several, often conflicting, components, where some components of the performance index can be made more prominent by making their coefficients appropriately larger. The minimization of the performance index does not guarantee the minimization of each component individually.

There is also a DAE formulation of the optimal control problem for the rolling ball which explicitly enforces the algebraic versor constraint on \( q \) and which is mathematically equivalent to (37). In the DAE formulation, the first component, \( q_0 \), of the versor \( q \) is moved from the state \( x \) to the control \( u \) and an imitator state, \( \hat{q}_0 \), is used to replace \( q_0 \) in \( x \). \( \hat{q}_{a,0} = q_{a,0} \), so that with perfect integration (i.e. no numerical integration errors), \( \hat{q}_0(t) = q_0(t) \) for \( a \leq t \leq b \). Defining

\[
\hat{q} \equiv \begin{bmatrix}
\hat{q}_0 \\
q^b
\end{bmatrix},
\]

(50)

then with perfect integration,

\[
\hat{q}(t) = \begin{bmatrix}
\hat{q}_0(t) \\
q^b(t)
\end{bmatrix} = \begin{bmatrix}
q_0(t) \\
q^b(t)
\end{bmatrix} = q(t)
\]

(51)
Table 4. Initial condition parameter values for the rolling ball. Refer to (43).

| Parameter | Value |
|-----------|-------|
| $\theta_a$ | $\begin{bmatrix} 0 & 2.0369 & .7044 \end{bmatrix}^\top$ |
| $\dot{\theta}_a$ | $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\top$ |
| $q_a$ | $\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^\top$ |
| $\Omega_a$ | $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ |
| $z_a$ | $\begin{bmatrix} 0 & 0 \end{bmatrix}$ |

Table 5. Final condition parameter values for the rolling ball. Refer to (44).

| Parameter | Value |
|-----------|-------|
| $\dot{\theta}_b$ | $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\top$ |
| $\Omega_b$ | $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\top$ |
| $z_b$ | $\begin{bmatrix} 1 & 1 \end{bmatrix}$ |

for $a \leq t \leq b$. $\tilde{q}_0$ is added to the state since the final conditions require knowledge of $q_0$, which is unavailable if it has been moved to the control since the final conditions are not a function of the control. The DAE formulation of the rolling ball’s optimal control problem is

$$
\min_{u_2} J \quad \text{s.t.} \quad \begin{cases} 
\dot{x}_2 = f_2(x_2, u_2), \\
h_2(x_2, u_2) = 1, \\
\sigma_2(x_2(a)) = 0, \\
\psi_2(x_2(b)) = 0,
\end{cases}
$$

(52)

where

$$
x_2 \equiv \begin{bmatrix} \theta \\ \dot{\theta} \\ \tilde{q} \\ \Omega \\ z \end{bmatrix}, \quad u_2 \equiv \begin{bmatrix} \dot{\theta} \\ q_0 \end{bmatrix},
$$

$$
f_2(x_2, u_2) \equiv \begin{bmatrix} \dot{\theta} \\ \dot{\theta} \\ \frac{1}{2} q \Omega^\top \\ \kappa(x, \dot{\theta}) \\ \left[q \Omega q^{-1} \right]_{12} \times r e_3 \end{bmatrix}, \quad h_2(x_2, u_2) \equiv |q|^2,
$$

(53)
\[ \sigma_2(x_2(a)) = \begin{bmatrix} \theta(a) - \theta_a \\ \dot{\theta}(a) - \dot{\theta}_a \\ \Omega(a) - \Omega_a \\ z(a) - z_a \end{bmatrix}, \]  

\[ \psi_2(x_2(b)) = \begin{bmatrix} \Pi \left( \begin{bmatrix} \ddot{q}(b) [\frac{1}{M} \sum_{i=0}^{3} m_i \zeta_i (\theta_i(b))]^2 \ddot{q}(b)^{-1} ] \right) \end{bmatrix} \right) \\ \dot{\theta}(b) - \dot{\theta}_b \\ \Omega(b) - \Omega_b \\ z(b) - z_b \end{bmatrix}. \]  

Even though the DAE formulation (52) is mathematically equivalent to the ODE formulation (37), the DAE formulation (52) tends to be numerically more stable to solve than the ODE formulation (37), as explained in Example 6.12 “Reorientation of an Asymmetric Rigid Body” of [11].

As explained in Appendix A, the controlled equations of motion for the ODE formulation (37) of the rolling ball’s optimal control problem are encapsulated by the ODE TPBVP:

\[ \begin{align*}
\dot{x} &= \dot{H}_x^T(x, \lambda, \mu) = \dot{f}(x, \lambda) \equiv f(x, \pi(x, \lambda)), \\
\lambda &= -H_x^T(x, \lambda, \mu) = -H_x^T(x, \lambda, \pi(x, \lambda), \mu), \\
\lambda_{t=a} &= -G_{x(a)}^T, \quad G_{\xi}^T = \sigma(x(a)) = 0, \\
\lambda_{t=b} &= G_{x(b)}^T, \quad G_{\nu}^T = \psi(x(b)) = 0.
\end{align*} \]  

Subappendix A.2 of [58] derives the formulas for constructing \( H_x^T \). In (56), \( G \) is the endpoint function

\[ G(x(a), \xi, x(b), \nu) \equiv \xi^T \sigma(x(a)) + \nu^T \psi(x(b)) \]

\[ \begin{bmatrix} \theta(a) - \theta_a \\ \dot{\theta}(a) - \dot{\theta}_a \\ \Omega(a) - \Omega_a \\ z(a) - z_a \end{bmatrix} + \begin{bmatrix} \Pi \left( \begin{bmatrix} \ddot{q}(b) [\frac{1}{M} \sum_{i=0}^{3} m_i \zeta_i (\theta_i(b))]^2 \ddot{q}(b)^{-1} ] \right) \end{bmatrix} \right) \\ \dot{\theta}(b) - \dot{\theta}_b \\ \Omega(b) - \Omega_b \\ z(b) - z_b \end{bmatrix}. \]  

where \( \xi \in \mathbb{R}^{15} \) and \( \nu \in \mathbb{R}^{10} \) are constant Lagrange multiplier vectors enforcing the initial and final conditions, (43) and (44), respectively. In (56), \( H \) is the Hamiltonian

\[ H(x, \lambda, u, \mu) = L(x, u, \mu) + \lambda^T f(x, u) \]

\[ = \frac{3}{2} \sum_{i=1}^{3} \gamma_i(\mu) \tilde{\theta}_i^2 + \frac{2}{2} \sum_{j=1}^{2} h_j(\mu) S(|z - v_j| - \rho_j) + \lambda^T \begin{bmatrix} \dot{\theta} \\ \dot{u} \end{bmatrix} \]

\[ \begin{bmatrix} \frac{\theta}{2\Omega^2} \\ \frac{3}{2} q \Omega^2 \\ \kappa(x, u) \end{bmatrix}, \]

where \( \lambda \in \mathbb{R}^{15} \) is a time-varying Lagrange multiplier vector enforcing the dynamics (42). In (56), \( \pi \) is an analytical formula expressing the control \( u \) as a function of
the state $x$ and the costate $\lambda$. The components of $\pi$ are given by

$$\dot{\theta}_i = \pi_i (x, \lambda) \equiv -\gamma_i^{-1} \left\{ \lambda_{3+i} + \lambda_{13}^T \left[ \sum_{k=0}^{3} m_k s_k^2 - I \right]^{-1} \left[ m_i s_i \times \zeta_i \right] \right\},$$

(59)

for $1 \leq i \leq 3$ and where $\lambda_{13} \equiv [\lambda_{11} \quad \lambda_{12} \quad \lambda_{13}]^T$. In (56), $\hat{H}$ is the regular Hamiltonian

$$\hat{H} (x, \lambda, \mu) \equiv H (x, \lambda, \pi (x, \lambda), \mu)$$

$$= \sum_{i=1}^{3} \frac{\gamma_i (\mu)}{2} \pi_i^2 (x, \lambda) + \sum_{j=1}^{2} h_j (\mu) s \left( |z - v_j| - \rho_j \right) + \lambda^T \left[ \begin{array}{c} \dot{\theta} \\ \pi (x, \lambda) \\ \frac{1}{2} q \Omega^2 \kappa (x, \pi (x, \lambda)) \\ \left( \frac{q \Omega^2}{e_3} \right)^{12} x \end{array} \right].$$

(60)

The reader is referred to [58] for a more general description of the ODE and DAE formulations, (37) and (52), of the rolling ball’s optimal control problem and the controlled equations of motion (56) corresponding to (37). The DAE TPBVP encapsulating the controlled equations of motion corresponding to the DAE formulation (52) of the rolling ball’s optimal control problem were not investigated since a robust DAE TPBVP solver is not readily available in MATLAB. COLDAE is a robust DAE TPBVP solver that uses collocation [5]; however, COLDAE is only available in Fortran and thus was not used in our calculations.

### 3.2. Numerical Solutions: Sigmoid Obstacle Avoidance.

The controlled equations of motion (56) for the rolling ball are solved numerically to move the ball between the pair of points while avoiding the pair of obstacles, where the obstacle avoidance function $S$ in (48) is realized via the time-reversed sigmoid function (15) with $\epsilon = .01$. The direct method solver GPOPS-II is used to solve the DAE formulation (52) of the optimal control problem, where the obstacle heights appearing in the integrand cost function (48) are $h_1 = h_2 = 0$ and where the values for the other integrand cost function coefficients are given in Table 6. Using this direct method solution as an initial guess, GPOPS-II is used again to solve the ODE formulation (37) of the same optimal control problem. Predictor-corrector continuation is then used to solve the controlled equations of motion (56), starting from the second direct method solution. The continuation parameter is $\mu$, which is used to adjust $h_1 = h_2$ according to the linear homotopy shown in Table 6, so that $h_1 = h_2 = 0$ when $\mu = .95$ and $h_1 = h_2 = 1,000$ when $\mu = .00001$. The predictor-corrector continuation begins at $\mu = .95$, which is consistent with the direct method solution obtained at $h_1 = h_2 = 0$.

For the direct method, GPOPS-II is run using the NLP solver SNOPT. The GPOPS-II mesh error tolerance is $1e-6$ and the SNOPT error tolerance is $1e-7$. In order to encourage convergence of SNOPT, a constant $C = 50$ is added to the integrand cost function $L$ in (48). The sweep predictor-corrector continuation method discussed in Appendix D is used by the indirect method. For the sweep predictor-corrector continuation method, there are 4 predictor-corrector steps, the maximum tangent steplength in each step is $\sigma_{\text{max}} = 500$, the direction of the initial unit tangent is determined by setting $d = -2$ to force the continuation parameter $\mu$
to initially decrease, the relative error tolerance is $1e^{-6}$, the unit tangent solver is \texttt{twpbvpc.m}, and the monotonic “sweep” continuation solver is \texttt{acdcc}. The numerical results are shown in Figures 9, 10, and 11. As $\mu$ decreases from .95 down to .9406 during continuation (see Figure 11a), $h_1 = h_2$ increases from 0 up to 9.93 (see Figure 11b). Since $h_1 = h_2$ is ratcheted up during continuation, thereby increasing the penalty in the integrand cost function (48) when the GC intrudes into the obstacles, by the end of continuation, the ball’s GC avoids both obstacles while veering smartly around the first obstacle (compare Figures 9a vs 9b), at the expense of slightly larger magnitude controls (compare Figures 9e vs 9f). The ball does not detach from the surface since the magnitude of the normal force is always positive (see Figures 10a and 10b). The ball rolls without slipping if the coefficient of static friction $\mu_s$ is at least $\hat{\mu}_s \approx 0.1055$ for the direct method solution (see Figure 10c) and if $\mu_s$ is at least $\hat{\mu}_s \approx 0.0988$ for the indirect method solution (see Figure 10d). As shown in Figures 11a-11c, the sweep predictor-corrector continuation indirect method encounters turning points at solutions 3 and 4.

![Parameter Value](image)

**Table 6.** Integrand cost function coefficient values for the rolling ball when predictor-corrector continuation is performed in the obstacle heights. Refer to (48).

3.3. Numerical Solutions: ReLU Obstacle Avoidance. The controlled equations of motion (56) for the rolling ball are solved numerically again to move the ball between the pair of points while avoiding the pair of obstacles, but this time the obstacle avoidance function $S$ in (48) is realized via the $C^2$ cutoff function (49). With the obstacle heights appearing in the integrand cost function (48) set to $h_1 = h_2 = 0$ and with the values for the other integrand cost function coefficients set according to Table 6, the same double pass (DAE formulation followed by ODE formulation) direct method, with all the same settings, parameters, and initial conditions, is used as in the previous subsection to generate the same initial solution. Starting from the direct method solution, two rounds of the same sweep predictor-corrector continuation method that was used in the previous subsection are again used here to solve the controlled equations of motion (56), with all the same settings and parameters, except for the number of predictor-corrector steps and maximum tangent steplengths used. In the first round, the continuation parameter $\mu$ is used to adjust $h_1 = h_2$ according to the linear homotopy shown in Table 6, so that $h_1 = h_2 = 0$ when $\mu = .95$ and $h_1 = h_2 = 1.00$ when $\mu = .00001$. This predictor-corrector continuation begins at $\mu = .95$, which is consistent with the direct method solution obtained at $h_1 = h_2 = 0$. In the first round, two predictor-corrector steps are made with the maximum tangent steplengths $\sigma_{\text{max}} = [200000 \ 100000]$. Starting from the predictor-corrector continuation solution obtained in the first round,
Figure 9. Numerical solutions of the rolling ball optimal control problem (37) for sigmoid obstacle avoidance using 3 control masses for γ1 = γ2 = γ3 = 10 and for fixed initial and final times. The obstacle centers are located at \( v_1 = \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = \begin{bmatrix} 2 \\ .2 \end{bmatrix}^T \) and \( v_2 = \begin{bmatrix} v_{2,1} \\ v_{2,2} \end{bmatrix} = \begin{bmatrix} .8 \\ .8 \end{bmatrix}^T \) and the obstacle radii are \( \rho_1 = \rho_2 = .282 \). The direct method results for obstacle heights at \( h_1 = h_2 = 0 \) are shown in the left column, while the predictor-corrector continuation indirect method results for obstacle heights at \( h_1 = h_2 \approx 9.93 \) are shown in the right column.
Figure 10. Numerical solutions of the rolling ball optimal control problem (37) for sigmoid obstacle avoidance using 3 control masses for \( \gamma_1 = \gamma_2 = \gamma_3 = 10 \) and for fixed initial and final times. The direct method results for obstacle heights at \( h_1 = h_2 = 0 \) are shown in the left column, while the predictor-corrector continuation indirect method results for obstacle heights at \( h_1 = h_2 \approx 9.93 \) are shown in the right column. The ball does not detach from the surface since the magnitude of the normal force is always positive. The ball rolls without slipping if \( \mu_s \geq 0.1055 \) for the direct method solution and if \( \mu_s \geq 0.0988 \) for the indirect method solution.

A second predictor-corrector continuation is used to solve the controlled equations of motion (56), where the continuation parameter is \( \mu \), which is now used to adjust \( \gamma_1 = \gamma_2 = \gamma_3 \) according to the linear homotopy shown in Table 7, so that \( \gamma_1 = \gamma_2 = \gamma_3 = 10 \) when \( \mu = .95 \) and \( \gamma_1 = \gamma_2 = \gamma_3 = -1,000 \) when \( \mu = .00001 \). The second predictor-corrector continuation begins at \( \mu = .95 \), which is consistent with the first predictor-corrector continuation solution obtained at \( \gamma_1 = \gamma_2 = \gamma_3 = 10 \).
Moreover, during the second predictor-corrector continuation, the obstacle heights \( h_1 = h_2 \) are fixed at 7.846e8, which is consistent with the final obstacle heights obtained by the first predictor-corrector continuation solution. In the second continuation, four predictor-corrector steps are made with the maximum tangent steplengths \( \sigma_{\text{max}} = [5000 \ 200 \ 1 \ 1] \).

The numerical results are shown in Figures 12, 13, and 14. As \( \mu \) decreases from .95 down to −7.453e5 during the first round of predictor-corrector continuation, \( h_1 = h_2 \) increases from 0 up to 7.846e8 (see Figure 14a). Since \( h_1 = h_2 \) is ratcheted up during continuation, thereby increasing the penalty in the integrand cost function (48) when the GC intrudes into the obstacles, by the end of continuation, the ball’s GC completely exits the second obstacle and approaches the boundary of the first obstacle (compare Figures 12a vs 12b). As \( \mu \) decreases from .95 down to .9406 during the second round of predictor-corrector continuation, \( \gamma_1 = \gamma_2 = \gamma_3 \) decreases from 10 down to 3.602e−5 (see Figure 14b). Since \( \gamma_1 = \gamma_2 = \gamma_3 \) is ratcheted down during continuation, thereby decreasing the penalty in the integrand cost function (48) for
large magnitude accelerations of the control mass parameterizations, and since the obstacle heights are held fixed at 7.846e8, by the end of continuation, the ball’s GC avoids both obstacles while veering smartly around the first obstacle (compare Figures 12b vs 12c). Figure 14c shows that the performance index $J$ increases from 284.2 up to 289.7 as the obstacle heights are ramped up in the first round of predictor-corrector continuation; Figure 14d shows that the performance index $J$ then decreases down to 250 as the control coefficients are ramped down and the ball’s GC fully departs the tall obstacles in the second round of predictor-corrector continuation.

| Parameter                  | Value                                      |
|----------------------------|--------------------------------------------|
| $\gamma_1(\mu) = \gamma_2(\mu) = \gamma_3(\mu)$ | $10 + \frac{35-\mu}{95-300001} (-1000 - 10)$ |
| $h_1 = h_2$                | $7.846e8$                                  |
| $v_1$                      | $0.2$                                      |
| $v_2$                      | $0.8$                                      |
| $\rho_1 = \rho_2$         | $0.282$                                    |

Table 7. Integrand cost function coefficient values for the rolling ball when a second round of predictor-corrector continuation is performed in the control coefficients. Refer to (48).

4. Summary, Discussion, and Future Work. The controlled equations of motion for the rolling disk and ball were solved numerically using predictor-corrector continuation, starting from an initial solution obtained via a direct method, to solve trajectory tracking problems for the rolling disk and obstacle avoidance problems for the rolling ball. These optimal control maneuvers were achieved by performing predictor-corrector continuation in weighting factors that scale penalty functions in the integrand cost function of the performance index.

This paper focused on the indirect, rather than direct, method to numerically solve the optimal control problems. Because the indirect and direct methods only converge to a local minimum solution near the initial guess, a robust continuation algorithm capable of handling turning points is needed to obtain indirect and direct method solutions of complicated, nonconvex optimal control problems. A continuation indirect method requires a continuation ODE or DAE TPBVP solver, while a continuation direct method requires a continuation NLP solver. Predictor-corrector continuation ODE TPBVP algorithms were presented in Appendices C and D and implemented in MATLAB to realize the continuation indirect method used to solve the rolling disk and ball optimal control problems. Even though predictor-corrector continuation NLP solver algorithms are provided in the literature (e.g. see [71, 48]), there do not seem to be any publicly available predictor-corrector continuation NLP solvers, which inhibited the use of a continuation direct method in this paper. When compared against the direct method, the indirect method suffers from two major deficiencies:

1. Unlike the direct method, the indirect method has a very small radius of convergence and therefore requires a very accurate initial solution guess [11,
Figure 12. Numerical solutions of the rolling ball optimal control problem (37) for ReLU obstacle avoidance using 3 control masses. First a direct method (left column), followed by predictor-corrector continuation in the obstacle heights \( h_1 = h_2 \) (middle column), and finally predictor-corrector continuation in the control coefficients \( \gamma_1 = \gamma_2 = \gamma_3 \) (right column).
Figure 13. A direct method (left column) is followed by two rounds of a predictor-corrector continuation indirect method to realize a ReLU obstacle avoidance maneuver for the rolling ball. The first round (middle column) of predictor-corrector continuation increases the obstacle heights $h_1 = h_2$, and the second round (right column) of predictor-corrector continuation decreases the control coefficients $\gamma_1 = \gamma_2 = \gamma_3$. The ball does not detach from the surface since the magnitude of the normal force is always positive. The ball rolls without slipping if $\mu_s \geq 0.1055$ for the direct method solution, if $\mu_s \geq 0.09942$ for the first indirect method solution, and if $\mu_s \geq 0.09917$ for the second indirect method solution.
Figure 14. Predictor-corrector continuation in the obstacle heights \( h_1 = h_2 \) (left column) is followed by predictor-corrector continuation in the control coefficients \( \gamma_1 = \gamma_2 = \gamma_3 \) (right column) to realize a ReLU obstacle avoidance maneuver for the rolling ball.
Moreover, unlike the direct method, the indirect method requires a guess of the costates, which are unphysical.

2. Unlike the direct method, the indirect method is unable to construct the switching structure (i.e. the times when the states and/or controls enter and exit the boundary) of an optimal control problem having path inequality constraints.

Since predictor-corrector continuation was used in this paper, the first deficiency in the indirect method only applied when constructing the solution of the initial ODE TPBVP, and this deficiency was circumvented by using a direct method to solve the optimal control problem corresponding to that initial ODE TPBVP. To circumvent the second deficiency in the indirect method, path inequality constraints were incorporated into the optimal control problems as soft constraints through penalty functions in the integrand cost functions.

The predictor-corrector continuation methods presented in Appendices C and D work if the control can be expressed analytically as a function of the state and costate, e.g. if the Hamiltonian is quadratic in the control. That is, those methods perform continuation only in the state and costate after the original optimal control DAE TPBVP is transformed into an ODE TPBVP through elimination of the control. For more complicated Hamiltonians (such as when penalty functions are added to the integrand cost function to softly enforce bounded body frame accelerations of the control masses, the no-detachment constraint, and the no-slip constraint), numerical methods (such as Newton’s method) must be used to construct the control numerically from the state, costate, and a good initial guess of the control. In these cases, the predictor-corrector continuation method of Appendix C must be extended to perform continuation in the state, costate, and control by solving the optimal control DAE TPBVP. This will be investigated in subsequent work.

In future work, instead of using MATLAB, the simulation code could be reimplemented in the higher performance programming languages Julia or C++, while relying on Fortran routines like COLNEW [7], COLMOD [22], TWPBVP(C) [23, 20], TWPBVPL(C) [10, 21], and ACDC(C) [22, 19] to solve the underlying ODE TPBVPs, to obtain faster numerical results. Julia and C++ feature several mature and efficient automatic differentiation libraries [1] capable of constructing the Jacobians and Hessians needed by the ODE TPBVP solvers. In addition, a more efficient and robust predictor-corrector adaptive tangent steplength algorithm, such as described in [8, 29], could be implemented.

Another avenue for future investigation is to use a neighboring extremal optimal control (NEOC) method [38], which constructs a homotopy between the controlled equations of motion and their linearization about a nominal solution; however, the NEOC method in [38] could be made more robust by using predictor-corrector, rather than monotonic, continuation in the homotopy parameter. Yet another avenue for future investigation is to perform predictor-corrector continuation in a weighting factor that scales a term in the endpoint cost function measuring the deviation between the actual and prescribed final conditions.

Additionally, throughout the paper we have kept the initial and final times fixed. It would be interesting to perform additional studies when the time duration is free, as outlined in Appendix A, especially regarding problems of navigation over complex terrains and slippery surfaces. Complex terrains will affect both the uncontrolled equations of motion, due to gravity, and the performance index, for example, through potential energy penalty functions that discourage the ball from ascending
steep slopes. If the substrate is slippery, for example, due to the presence of moisture, one can imagine situations where the valleys are wet and the slopes are dry and thus less slippery. Then, one can introduce an additional term in the performance index penalizing motion through the valleys where the possibility of a slip is high. The interplay between the terms in the performance index discouraging and encouraging motion up the slopes will give a very interesting control system. These and other interesting questions will be considered in future work.

Acknowledgements. We are indebted to our colleagues A.M. Bloch, D.M. de Diego, F. Gay-Balmaz, D.D. Holm, M. Leok, A. Lewis, T. Ohsawa, and D.V. Zenkov for useful and fruitful discussions. M.J. Weinstein provided copious advice on using the MATLAB automatic differentiation toolbox ADiGator and fixed numerous bugs in ADiGator that were revealed in the course of this research. A.V. Rao provided a free license to use the MATLAB direct method optimal control solver GPOPS-II for some of this research. P. Tallapragada observed that the reaction forces exerted on the ball by the accelerating internal point masses may cause the ball to detach from the surface, which prompted the inclusion of plots depicting the normal force and minimum coefficient of static friction. G.M. Rozenblat pointed out the necessary and sufficient conditions that must be satisfied by a ball’s physical parameters in [63].

This research was partially supported by the NSERC Discovery Grant, the University of Alberta Centennial Fund, and the Alberta Innovates Technology Funding (AITF) which came through the Alberta Centre for Earth Observation Sciences (CEOS). S.M. Rogers also received support from the University of Alberta Doctoral Recruitment Scholarship, the FGSR Graduate Travel Award, the IGR Travel Award, the GSA Academic Travel Award, the AMS Fall Sectional Graduate Student Travel Grant, Target Corporation, and the Institute for Mathematics and its Applications at the University of Minnesota, Twin Cities.

Appendix A. Optimal Control: Variational Pontryagin’s Minimum Principle. This appendix presents necessary conditions, called the variational Pontryagin’s minimum principle, which a solution to an optimal control problem lacking path inequality constraints must satisfy; there is a more general version of Pontryagin’s minimum principle that applies to optimal control problems possessing path inequality constraints. In this paper, these necessary conditions, in the context of describing the optimal control of the rolling ball, are referred to as the controlled equations of motion. In the literature, application of Pontryagin’s minimum principle to solve an optimal control problem is called the indirect method. Let \( n, m \in \mathbb{N} \). Let \( a \) be a prescribed or free initial time and let \( k_1 \in \mathbb{N}^0 \) be such that \( 0 \leq k_1 \leq n \) if \( a \) is prescribed and \( 1 \leq k_1 \leq n + 1 \) if \( a \) is free. Let \( b \) be a prescribed or free final time and let \( k_2 \in \mathbb{N}^0 \) be such that \( 0 \leq k_2 \leq n \) if \( b \) is prescribed and \( 1 \leq k_2 \leq n + 1 \) if \( b \) is free. Suppose a dynamical system has state \( x \in \mathbb{R}^n \) and control \( u \in \mathbb{R}^m \) and the control \( u \) is sought that minimizes the performance index

\[
J \equiv p(a, x(a), b, x(b), \mu) + \int_a^b L(t, x, u, \mu) \, dt
\]  

subject to the system dynamics defined for \( a \leq t \leq b \)

\[
\dot{x} = f(t, x, u, \mu),
\]
the prescribed initial conditions at time $t = a$

$$\sigma (a, x(a), \mu) = 0_{k_1 \times 1}, \quad (63)$$

and the prescribed final conditions at time $t = b$

$$\psi (b, x(b), \mu) = 0_{k_2 \times 1}. \quad (64)$$

$p$ is a scalar-valued function called the endpoint cost function, $L$ is a scalar-valued function called the integrand cost function, $x$ and $f$ are $n \times 1$ vector-valued functions, $u$ is an $m \times 1$ vector-valued function, $\sigma$ is a $k_1 \times 1$ vector-valued function, and $\psi$ is a $k_2 \times 1$ vector-valued function. $\mu$ is a prescribed scalar parameter which may be exploited to numerically solve this problem via continuation. More concisely, this optimal control problem may be stated as

$$\min_{a, x(a), b, u} J \text{ s.t. } \begin{cases} \dot{x} = f(t, x, u, \mu), \\ \sigma (a, x(a), \mu) = 0_{k_1 \times 1}, \\ \psi (b, x(b), \mu) = 0_{k_2 \times 1}. \end{cases} \quad (65)$$

Observe that the optimal control problem encapsulated by (65) ignores path inequality constraints such as $D(t, x, u, \mu) \leq 0$, where $D$ is an $r \times 1$ vector-valued function for $r \in \mathbb{N}^0$. Path inequality constraints can be incorporated into (65) as soft constraints through penalty functions in the integrand cost function $L$ or the endpoint cost function $p$. By omitting hard path inequality constraints from (65), a solution of (65) does not lie on the boundary of a compact set and the calculus of variations may be applied to derive necessary conditions, called the variational Pontryagin’s minimum principle, which a solution of (65) must satisfy.

Define the endpoint function $G$ and the Hamiltonian $H$ by

$$G (a, x(a), \xi, b, x(b), \nu, \mu) \equiv p (a, x(a), b, x(b), \mu) + \xi^T \sigma (a, x(a), \mu) + \nu^T \psi (b, x(b), \mu), \quad (66)$$

$$H (t, x, \lambda, u, \mu) \equiv L (t, x, u, \mu) + \lambda^T f (t, x, u, \mu),$$

where $\xi$ is a $k_1 \times 1$ constant Lagrange multiplier vector, $\nu$ is a $k_2 \times 1$ constant Lagrange multiplier vector, and $\lambda$ is an $n \times 1$ time-varying Lagrange multiplier vector. In the literature, the time-varying Lagrange multiplier vector used to adjoin the system dynamics to the integrand cost function is often called the adjoint variable or the costate. Henceforth, the time-varying Lagrange multiplier vector is referred to as the costate and the elements in this vector are referred to as the costates. The necessary conditions [41] on $x$, $\lambda$, and $u$ which a solution of (65) must satisfy are the DAEs defined for $a \leq t \leq b$

$$\dot{x} = H_\lambda^T (t, x, \lambda, u, \mu) = f(t, x, u, \mu), \quad \dot{\lambda} = -H_x^T (t, x, \lambda, u, \mu), \quad (67)$$

and the initial conditions defined at time $t = a$

$$H|_{t=a} = G_a, \quad \lambda|_{t=a} = -G_{\xi}^T x(a), \quad G_\xi^T = \sigma (a, x(a), \mu) = 0_{k_1 \times 1}, \quad (68)$$

and the right boundary conditions defined at time $t = b$

$$H|_{t=b} = -G_b, \quad \lambda|_{t=b} = G_\xi^T x(b), \quad G_\nu^T = \psi (b, x(b), \mu) = 0_{k_2 \times 1}. \quad (69)$$

If the initial time $a$ is prescribed, then the left boundary condition $H|_{t=a} = G_a$ is dropped. If the final time $b$ is prescribed, then the right boundary condition

If $H_{uu}$ is nonsingular, then the optimal control problem is said to be regular or nonsingular; otherwise if $H_{uu}$ is singular, then the optimal control problem is said to be singular. If $H_{uu}$ is nonsingular, then by the implicit function theorem, the algebraic equation $H_u = 0_{1 \times m}$ in (67) guarantees the existence of a unique function, say $\pi$, for which

$$u = \pi (t, x, \lambda, \mu).$$

(70)

If $H_{uu}$ is nonsingular, it may be possible to solve the algebraic equation $H_u = 0_{1 \times m}$ in (67) analytically for $u$ in terms of $t, x, \lambda,$ and $\mu$ to construct $\pi$ explicitly in (70); otherwise, the value $u$ of $\pi$ in (70) may be constructed numerically in an efficient and accurate manner (with quadratic convergence) via a few iterations of Newton’s method applied to $H_u = 0_{1 \times m}$ starting from an initial guess $u_0$ of $u$:

$$u_{i+1} = u_i - H_{uu}^{-1} (t, x, \lambda, u_i, \mu) H_u^T (t, x, \lambda, u_i, \mu).$$

(71)

Using (70), the Hamiltonian may be re-expressed as a function of $t, x, \lambda,$ and $\mu$ via the regular or reduced Hamiltonian

$$\hat{H} (t, x, \lambda, \mu) \equiv H (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu).$$

(72)

Note that by construction of $\pi$,

$$H_u (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) = 0_{1 \times m}. \quad \text{(73)}$$

By using the definition (72) of the regular Hamiltonian $\hat{H}$, invoking (73), and defining

$$\hat{f} (t, x, \lambda, \mu) \equiv f (t, x, \pi (t, x, \lambda, \mu), \mu),$$

(74)

it follows from the chain rule that

$$\hat{H}_t (t, x, \lambda, \mu) = H_t (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$$

$$+ H_u (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_t (t, x, \lambda, \mu),$$

(75)

$$\hat{H}_x (t, x, \lambda, \mu) = H_x (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$$

$$+ H_u (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_x (t, x, \lambda, \mu),$$

(76)

$$\hat{H}_\lambda (t, x, \lambda, \mu) = H_\lambda (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$$

$$+ H_u (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_\lambda (t, x, \lambda, \mu)$$

$$= H_\lambda (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$$

(77)

$$= \hat{f}_x (t, x, \pi (t, x, \lambda, \mu), \mu)$$

$$= \hat{f}_x (t, x, \lambda, \mu),$$

and

$$\hat{H}_\mu (t, x, \lambda, \mu) = H_\mu (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$$

$$+ H_u (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_\mu (t, x, \lambda, \mu).$$

(78)

By using (70) to eliminate the algebraic equation $H_u = 0_{1 \times m}$ from (67), by plugging (77) and (76) into the right hand sides of the ODEs in (67), and by plugging the definition (72) into the left and right boundary conditions (68) and (69), the
necessary conditions on \( x \) and \( \lambda \) which a solution of (65) must satisfy are the ODEs defined for \( a \leq t \leq b \)

\[
\dot{x} = \hat{H} \lambda^T (t, x, \lambda, \mu),
\]

\[
\dot{\lambda} = -\hat{H} \lambda^T (t, x, \lambda, \mu),
\]

the left boundary conditions defined at time \( t = a \)

\[
\hat{H} \bigg|_{t=a} = G_a, \quad \lambda \bigg|_{t=a} = -G_x^T (a), \quad G_T^\sigma (a, x(a), \mu) = 0_{k_1 \times 1},
\]

and the right boundary conditions defined at time \( t = b \)

\[
\hat{H} \bigg|_{t=b} = -G_b, \quad \lambda \bigg|_{t=b} = G_x^T (b), \quad G_T^\psi (b, x(b), \mu) = 0_{k_2 \times 1}.
\]

If the initial time \( a \) is prescribed, then the left boundary condition \( \hat{H} \bigg|_{t=a} = G_a \) is dropped. If the final time \( b \) is prescribed, then the right boundary condition \( \hat{H} \bigg|_{t=b} = -G_b \) is dropped. The necessary conditions (79), (80), and (81) constitute an ODE TPBVP. Appendix B provides implementation details for numerically solving the ODE TPBVP (79), (80), and (81).

A solution of the DAE TPBVP (67), (68), and (69) or of the ODE TPBVP (79), (80), and (81) is said to be an extremal solution of the optimal control problem (65). Note that an extremal solution only satisfies necessary conditions for a minimum of the optimal control problem (65), so that an extremal solution is not guaranteed to be a local minimum of (65).

Since the DAE TPBVP (67), (68), and (69) and the ODE TPBVP (79), (80), and (81) have small convergence radii [17, 16, 11], a continuation method (performing continuation in the parameter \( \mu \)) is often required to numerically solve them starting from a solution to a simpler optimal control problem [4]. The solution to the simpler optimal control problem might be obtained via analytics, the gradient method [17, 16], the method of successive approximations [46, 47, 51, 24], or the direct method [11, 57]. For example in [59], the continuation parameter \( \mu \) is used to vary integrand cost function coefficients in \( L \) in order to numerically solve the optimal control ODE TPBVPs for Suslov’s problem via monotonic continuation, starting from an analytical solution to a singular optimal control problem. In Sections 2 and 3, the continuation parameter \( \mu \) is used to vary integrand cost function coefficients in \( L \) in order to numerically solve the optimal control ODE TPBVPs for the rolling disk and ball via predictor-corrector continuation, starting from a direct method solution to a simpler optimal control problem. Appendices C and D describe predictor-corrector continuation methods for solving ODE TPBVPs and which are used to solve the optimal control ODE TPBVPs for the rolling disk and ball in Sections 2 and 3.

**Appendix B. Implementation Details for Solving the ODE TPBVP for a Regular Optimal Control Problem.** Details for numerically solving the ODE TPBVP (79), (80), and (81) associated with the indirect method solution of a regular optimal control problem are presented here. There are two general methods, initial value and global, for numerically solving an ODE TPBVP. An initial value method, such as single or multiple shooting, subdivides the integration interval \([a, b]\) into a fixed, finite mesh and integrates the ODE on each mesh subinterval using a guess of the unknown initial conditions at one endpoint in each mesh subinterval. A root-finder is used to iteratively adjust the guesses of the unknown initial conditions until the solution segments are continuous at the internal mesh points and until the
boundary conditions at the endpoints $a$ and $b$ are satisfied. A global method, such as
a Runge-Kutta, collocation, or finite-difference scheme, subdivides the integration
interval $[a, b]$ into a finite, adaptive mesh and solves a large nonlinear system of
algebraic equations obtained by imposing the ODE constraints at a finite set of
points in each mesh subinterval, by imposing continuity of the solution at internal
mesh points, and by imposing the boundary conditions at the endpoints $a$ and $b$. By
estimating the error in each mesh subinterval, a global method iteratively refines or
adapts the mesh until a prescribed error tolerance is satisfied. Because initial value
methods cannot integrate unstable ODEs, global methods are preferred [4, 54, 13].

B.1. Normalization and ODE Velocity Function. There are many solvers
available to numerically solve the ODE TPBVP (79), (80), and (81). For example,
bvp4c [64], bvp5c [43], sbvp [6], and bvptwp [19] (which encapsulates twpbvp.m,
twpbvpc.m, twpbvpl.m, twpbvpc1.m, acdc, and acdcc) are MATLAB Runge-Kutta or
collocation ODE TPBVP solvers, while COLSYS [3], COLNEW [7], COLMOD
[22], COLCON [8], BVP.M-2 [14], TWPBVP [23], TWPBVPC [20], TWPBVPL
[10], TWPBVPLC [21], ACDC [22], and ACDC [19] are Fortran Runge-Kutta or
collocation ODE TPBVP solvers. The reader is referred to the Appendix in [59]
for a comprehensive list of ODE TPBVP solvers. In order to numerically solve
the ODE TPBVP (79), (80), and (81), many solvers (such as the global method
MATLAB and Fortran solvers just listed) require that the ODE TPBVP be defined
on a fixed time interval and any unknown parameters, such as $\xi$, $\nu$, $a$, and $b$, must
often be modeled as dummy constant dependent variables with zero derivatives.
In addition, to aid convergence, many solvers can exploit Jacobians of the ODE
velocity function and of the two-point boundary condition function. Thus, (79) is
redefined on the normalized time interval $[0, 1]$ through the change of independent
variable $s \equiv \frac{t - a}{T}$, where $T \equiv b - a$. Note that $t(s) = Ts + a$. Define the normalized
state $\tilde{x}(s) \equiv x(t(s))$ and normalized costate $\tilde{\lambda}(s) \equiv \lambda(t(s))$. Define the expanded
un-normalized ODE TPBVP dependent variable vector

$$ z(t) \equiv \begin{bmatrix} x(t) \\ \lambda(t) \\ \xi \\ \nu \\ a \\ b \end{bmatrix} . $$

(82)

Defining $\tilde{z}(s) \equiv z(t(s))$, the expanded normalized ODE TPBVP dependent variable vector is

$$ \tilde{z}(s) \equiv z(t(s)) = \begin{bmatrix} x(t(s)) \\ \lambda(t(s)) \\ \xi \\ \nu \\ a \\ b \end{bmatrix} = \begin{bmatrix} \tilde{x}(s) \\ \tilde{\lambda}(s) \\ \xi \\ \nu \\ a \\ b \end{bmatrix} . $$

(83)
By the chain rule, (79), and since \( \frac{d(x)s}{ds} = T \),
\[
\dot{z}(s) = \frac{d\dot{z}(s)}{ds} = \left[ \begin{array}{c} \dot{\lambda}(s) \\ \dot{\lambda}(s) \end{array} \right] = \left[ \begin{array}{c} \frac{dz(t(s))}{dt} \\ \frac{d\lambda(t(s))}{dt} \end{array} \right] = \left[ \begin{array}{c} \frac{d\lambda(t(s))}{dt} \\ \frac{d\lambda(t(s))}{dt} \end{array} \right] \frac{dt(s)}{ds} = \left[ \begin{array}{c} 0_{(k_1+k_2+2)\times 1} \\ 0_{(k_1+k_2+2)\times 1} \end{array} \right] \frac{dt(s)}{ds}
\]
\[
\dot{z}(s) = \left[ \begin{array}{c} \dot{f}(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \\ \dot{H}_x(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \end{array} \right] T = \left[ \begin{array}{c} \dot{f}(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \\ \dot{H}_x(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \end{array} \right] T.
\]

Define \( \Phi(s, \hat{z}(s), \mu) \) to be the right-hand side of (84), i.e. the normalized ODE velocity function, so that
\[
\Phi(s, \hat{z}(s), \mu) = \left[ \begin{array}{c} \dot{f}(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \\ -\dot{H}_x(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \end{array} \right] T.
\]

The Jacobian of \( \Phi \) with respect to \( \hat{z}(s) \) is
\[
\Phi_{\hat{z}}(s, \hat{z}(s), \mu) = \left[ \begin{array}{c} \dot{f}(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \\ -\dot{H}_x(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \end{array} \right] T.
\]

and the Jacobian of \( \Phi \) with respect to \( \mu \) is
\[
\Phi_{\mu}(s, \hat{z}(s), \mu) = \left[ \begin{array}{c} \dot{f}(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \\ -\dot{H}_x(t(s), \hat{\mu}(t(s), \lambda(t(s)), \mu)) \end{array} \right] T.
\]

In (86) and (87), shorthand notation is used for conciseness and all zeroth and first derivatives of \( \dot{f} \) and all first and second derivatives of \( H \) are evaluated at \( s, \hat{z}(s), \mu \).

An explanation of the meaning of the shorthand notation used to express all zeroth and first derivatives of \( \dot{f} \) and all first and second derivatives of \( H \) is given in Table 8.

In rows \( n+1 \) through \( 2n \) and columns \( n+1 \) through \( 2n \) of (86), Clairaut’s Theorem was used to obtain \( \dot{H}_{\lambda} = \dot{H}_{x\lambda} = \dot{f}_x \), recalling from (77) that \( H_{\lambda} = \dot{f} \).

| Shorthand | Extended Shorthand | Normalized | Un-Normalized |
|------------|-------------------|-------------|---------------|
| \( \hat{f} \) | \( \hat{f}_{(s, \lambda(t(s)), \mu)} \) | \( \hat{f}(t(s), \lambda(t(s)), \mu) \) | \( \hat{f}(t(s), \lambda(t(s)), \mu) \) |
| \( \hat{f}_\lambda \) | \( \hat{f}_{(s, \lambda(t(s)), \mu)} \) | \( \hat{f}_\lambda(t(s), \lambda(t(s)), \mu) \) | \( \hat{f}_\lambda(t(s), \lambda(t(s)), \mu) \) |
| \( \hat{f}_\mu \) | \( \hat{f}_{(s, \lambda(t(s)), \mu)} \) | \( \hat{f}_\mu(t(s), \lambda(t(s)), \mu) \) | \( \hat{f}_\mu(t(s), \lambda(t(s)), \mu) \) |
| \( H_\lambda \) | \( H_{x\lambda} \) | \( H_{x\lambda}(t(s), \lambda(t(s)), \mu) \) | \( H_{x\lambda}(t(s), \lambda(t(s)), \mu) \) |
| \( H_{\mu\mu} \) | \( H_{x\mu} \) | \( H_{x\mu}(t(s), \lambda(t(s)), \mu) \) | \( H_{x\mu}(t(s), \lambda(t(s)), \mu) \) |
| \( H_{x\lambda} \) | \( H_{x\mu} \) | \( H_{x\mu}(t(s), \lambda(t(s)), \mu) \) | \( H_{x\mu}(t(s), \lambda(t(s)), \mu) \) |

Table 8. Explanation of shorthand notation for zeroth and first derivatives of \( \dot{f} \) and first and second derivatives of \( H \) used in (86) and (87).
Recall that \( \hat{f} \) is defined in (74) in terms of \( f \) and \( \pi \). By using the chain rule, the first derivatives of \( \hat{f} \) that appear in (86), (87), and Table 8 may be computed from first derivatives of \( f \) and \( \pi \) as follows:

\[
\hat{f}_{\lambda} (t, x, \lambda, \mu) = f_u (t, x, \pi (t, x, \lambda, \mu), \mu) \pi_{\lambda} (t, x, \lambda, \mu), \tag{88}
\]

\[
\hat{f}_x (t, x, \lambda, \mu) = f_x (t, x, \pi (t, x, \lambda, \mu), \mu) + f_u (t, x, \pi (t, x, \lambda, \mu), \mu) \pi_x (t, x, \lambda, \mu), \tag{89}
\]

\[
\hat{f}_t (t, x, \lambda, \mu) = f_t (t, x, \pi (t, x, \lambda, \mu), \mu) + f_u (t, x, \pi (t, x, \lambda, \mu), \mu) \pi_t (t, x, \lambda, \mu), \tag{90}
\]

and

\[
\hat{f}_\mu (t, x, \lambda, \mu) = f_\mu (t, x, \pi (t, x, \lambda, \mu), \mu) + f_u (t, x, \pi (t, x, \lambda, \mu), \mu) \pi_\mu (t, x, \lambda, \mu). \tag{91}
\]

Recall that by construction of \( \pi \), \( H_u (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) = 0_{1 \times m} \), as stated previously in (73). Differentiating \( H_u (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) = 0_{1 \times m} \) with respect to \( \lambda, \mu, x, t, \lambda, \mu, x \), and \( \mu \), in turn, and using the chain rule gives

\[
H_{u\lambda} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) + H_{uu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_{\lambda} (t, x, \lambda, \mu) = 0_{m \times m}, \tag{92}
\]

\[
H_{ux} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) + H_{uu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_x (t, x, \lambda, \mu) = 0_{m \times n}, \tag{93}
\]

\[
H_{ut} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) + H_{uu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_t (t, x, \lambda, \mu) = 0_{m \times 1}, \tag{94}
\]

and

\[
H_{\mu u} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) + H_{uu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_\mu (t, x, \lambda, \mu) = 0_{m \times 1}. \tag{95}
\]

(92), (93), (94), and (95) may be solved for \( \pi_{\lambda}, \pi_x, \pi_t, \) and \( \pi_\mu \), respectively:

\[
\pi_{\lambda} (t, x, \lambda, \mu) = -H^{-1}_{u\lambda} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) H_{u\lambda} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) = -H^{-1}_{u\lambda} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) f_{u\lambda}^T (t, x, \pi (t, x, \lambda, \mu), \mu), \tag{96}
\]

\[
\pi_x (t, x, \lambda, \mu) = -H^{-1}_{ux} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) H_{ux} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu), \tag{97}
\]

\[
\pi_t (t, x, \lambda, \mu) = -H^{-1}_{ut} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) H_{ut} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu), \tag{98}
\]

and

\[
\pi_\mu (t, x, \lambda, \mu) = -H^{-1}_{\mu u} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) H_{\mu u} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu). \tag{99}
\]

In (96), Clairaut’s Theorem was used to obtain \( H_{u\lambda} = H_{\lambda u}^T = f_{u\lambda}^T \), since \( H_{\lambda} = f_\lambda^T \).

As will be stated again later, (96), (97), (98), and (99) are especially useful if the value of \( \pi \) is constructed numerically via Newton’s method as in (71); if \( \pi \) is given analytically, then it should be possible to construct \( \pi_{\lambda}, \pi_x, \pi_t, \pi_\mu \) via manual, symbolic, or automatic differentiation of the analytical formula for \( \pi \).

Since Clairaut’s Theorem guarantees that

\[
H_{xu} (t, x, \lambda, u, \mu) = H_{ux}^T (t, x, \lambda, u, \mu) \tag{100}
\]

and

\[
H_{\mu u} (t, x, \lambda, u, \mu) = H_{u\mu}^T (t, x, \lambda, u, \mu), \tag{101}
\]
(93) may be solved for $H_{xu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$:

$$H_{xu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) = H_{uu}^T (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_x (t, x, \lambda, \mu)$$

$$= -\pi_{x}^T (t, x, \lambda, \mu) H_{uu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$$

$$= -\pi_{x}^T (t, x, \lambda, \mu) H_{uu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) .$$

(102)

By differentiating (76) with respect to $x$, $t$, and $\mu$, using the chain rule, and exploiting (102), the second derivatives of $\hat{H}$ that appear in (86), (87), and Table 8 may be computed from first derivatives of $\pi$ and second derivatives of $H$ as follows:

$$\hat{H}_{xx} (t, x, \lambda, \mu)$$

$$= H_{xx} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) + H_{xu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_x (t, x, \lambda, \mu)$$

$$= H_{xx} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$$

$$- \pi_{x}^T (t, x, \lambda, \mu) H_{uu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_x (t, x, \lambda, \mu) ,$$

(103)

$$\hat{H}_{xt} (t, x, \lambda, \mu)$$

$$= H_{xt} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) + H_{xu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_t (t, x, \lambda, \mu)$$

$$= H_{xt} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$$

$$- \pi_{x}^T (t, x, \lambda, \mu) H_{uu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_t (t, x, \lambda, \mu) ,$$

(104)

and

$$\hat{H}_{x\mu} (t, x, \lambda, \mu)$$

$$= H_{x\mu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) + H_{xu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_\mu (t, x, \lambda, \mu)$$

$$= H_{x\mu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu)$$

$$- \pi_{x}^T (t, x, \lambda, \mu) H_{uu} (t, x, \lambda, \pi (t, x, \lambda, \mu), \mu) \pi_\mu (t, x, \lambda, \mu) .$$

(105)

If the value of $\pi$ is constructed numerically via Newton’s method as in (71) rather than analytically, then (96), (97), (98), and (99) should be used to evaluate $\pi_\lambda$, $\pi_x$, $\pi_t$, and $\pi_\mu$, which appear in the formulas for $\hat{f}_\lambda$ given in (88), $\hat{f}_x$ given in (89), $\hat{f}_t$ given in (90), $\hat{f}_\mu$ given in (91), $\hat{H}_{xx}$ given in (103), $\hat{H}_{xt}$ given in (104), and $\hat{H}_{x\mu}$ given in (105). The second equation in (103), (104), and (105) is given because it may be more computationally efficient than the first equation if $\pi$ is given analytically, so that (96), (97), (98), and (99) need not be used to evaluate $\pi_\lambda$, $\pi_x$, $\pi_t$, and $\pi_\mu$.

B.2. Two-Point Boundary Condition Function. Now the boundary conditions (80)-(81) are considered. Letting

$$Y_1 (z(a), z(b), \mu) \equiv \begin{bmatrix} \hat{H} (a, x(a), \lambda(a), \mu) \\ \lambda(a) \\ \sigma (a, x(a), \mu) \\ \hat{H} (b, x(b), \lambda(b), \mu) \\ \lambda(b) \\ \psi (b, x(b), \mu) \end{bmatrix} ,$$

(106)
\( \mathcal{Y}_2(z(a), z(b), \mu) \equiv \begin{bmatrix} G_a (a, x(a), \xi, b, x(b), \nu, \mu) \\ -G^T_{x(a)} (a, x(a), \xi, b, x(b), \nu, \mu) \\ 0_{k_1 \times 1} \\ -G_b (a, x(a), \xi, b, x(b), \nu, \mu) \\ G^T_{x(b)} (a, x(a), \xi, b, x(b), \nu, \mu) \\ 0_{k_2 \times 1} \end{bmatrix} \), \hspace{1cm} (107)

and

\[
\mathcal{Y} (z(a), z(b), \mu) \equiv \mathcal{Y}_1 (z(a), z(b), \mu) - \mathcal{Y}_2 (z(a), z(b), \mu)
\]

\[
\begin{bmatrix} \hat{H} (a, x(a), \lambda(a), \mu) \\ \sigma (a, x(a), \mu) \\ \hat{H} (b, x(b), \lambda(b), \mu) \\ \psi (b, x(b), \mu) \end{bmatrix} = \begin{bmatrix} G_a (a, x(a), \xi, b, x(b), \nu, \mu) \\ -G^T_{x(a)} (a, x(a), \xi, b, x(b), \nu, \mu) \\ 0_{k_1 \times 1} \\ -G_b (a, x(a), \xi, b, x(b), \nu, \mu) \\ G^T_{x(b)} (a, x(a), \xi, b, x(b), \nu, \mu) \\ 0_{k_2 \times 1} \end{bmatrix} \), \hspace{1cm} (108)

the boundary conditions (80)-(81) in un-normalized dependent variables are given by the two-point boundary condition function

\[
\mathcal{Y} (z(a), z(b), \mu) = 0_{(2n+k_1+k_2+2) \times 1}. \hspace{1cm} (109)
\]

The Jacobians of \( \mathcal{Y} \) with respect to \( z(a), z(b), \) and \( \mu \) are

\[
\mathcal{Y}_{z(a)} (z(a), z(b), \mu) = \mathcal{Y}_{1,z(a)} (z(a), z(b), \mu) - \mathcal{Y}_{2,z(a)} (z(a), z(b), \mu), \hspace{1cm} (110)
\]

\[
\mathcal{Y}_{z(b)} (z(a), z(b), \mu) = \mathcal{Y}_{1,z(b)} (z(a), z(b), \mu) - \mathcal{Y}_{2,z(b)} (z(a), z(b), \mu), \hspace{1cm} (111)
\]

and

\[
\mathcal{Y}_{\mu} (z(a), z(b), \mu) = \mathcal{Y}_{1,\mu} (z(a), z(b), \mu) - \mathcal{Y}_{2,\mu} (z(a), z(b), \mu), \hspace{1cm} (112)
\]

where

\[
\mathcal{Y}_{1,z(a)} (z(a), z(b), \mu) = \begin{bmatrix} \hat{H}_x \big|_a \\ \sigma \big|_a \\ 0_{n \times n} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} \hat{f}^T \big|_a \\ \sigma a \big|_a \\ 0_{n \times n} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} 0_{1 \times k_1} \\ 0_{k_1 \times k_2} \\ 0_{n \times n} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} 0_{1 \times k_1} \\ 0_{k_1 \times k_2} \\ 0_{n \times n} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} \hat{H}_t \big|_a \\ 0 \end{bmatrix}, \hspace{1cm} (113)
\]

\[
\mathcal{Y}_{1,z(b)} (z(a), z(b), \mu) = \begin{bmatrix} \hat{H}_z \big|_b \\ \sigma \big|_b \\ 0_{n \times n} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} \hat{f}^T \big|_b \\ \sigma a \big|_b \\ 0_{n \times n} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} 0_{1 \times k_1} \\ 0_{k_1 \times k_2} \\ 0_{n \times n} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} 0_{1 \times k_1} \\ 0_{k_1 \times k_2} \\ 0_{n \times n} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} \hat{H}_t \big|_b \\ 0 \end{bmatrix}, \hspace{1cm} (114)
\]
\[ \Upsilon_{1,\mu}(\mathbf{z}(a), \mathbf{z}(b), \mu) = \begin{bmatrix} \dot{H}_\mu |_{a} \\ 0_{n \times 1} \\ \sigma_{\mu} \\ \dot{H}_\mu |_{b} \\ 0_{n \times 1} \end{bmatrix}, \]  

(115)

\[ \Upsilon_{2,\mu}(\mathbf{z}(a), \mathbf{z}(b), \mu) = \begin{bmatrix} G_{a\xi} \\ -G_{\xi(a)} \\ 0_{1 \times n} \\ 0_{k_1 \times n} \\ 0_{k_1 \times k_1} \end{bmatrix} \begin{bmatrix} \sigma_{\mu}^T \\ -\sigma_\xi \\ 0_{n \times n} \\ 0_{k_1 \times n} \\ 0_{k_1 \times k_1} \end{bmatrix} \begin{bmatrix} 0_{1 \times k_2} \\ 0_{k_1 \times k_2} \\ 0_{k_1 \times n} \\ 0_{k_1 \times k_2} \\ 0_{k_1 \times k_1} \end{bmatrix} \begin{bmatrix} G_{a\nu} \\ -G_{\xi(a)\nu} \\ G_{\xi(b)} \\ -G_{\xi(b)\nu} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} G_{aa} \\ -G_{\xi(a)a} \\ G_{\xi(b)a} \\ -G_{\xi(b)a} \\ 0_{n \times n} \end{bmatrix} \begin{bmatrix} G_{ab} \\ -G_{\xi(a)b} \\ G_{\xi(b)b} \\ -G_{\xi(b)b} \end{bmatrix} \]  

(116)

\[ \Upsilon_{2,\mu}(\mathbf{z}(a), \mathbf{z}(b), \mu) = \begin{bmatrix} G_{a\beta} \\ -G_{\beta(a)} \\ 0_{1 \times n} \\ 0_{k_1 \times n} \\ 0_{k_1 \times k_1} \end{bmatrix} \begin{bmatrix} \sigma_{\mu}^T \\ -\sigma_\beta \\ 0_{n \times n} \\ 0_{k_1 \times n} \\ 0_{k_1 \times k_1} \end{bmatrix} \begin{bmatrix} 0_{1 \times k_2} \\ 0_{k_1 \times k_2} \\ 0_{k_1 \times n} \\ 0_{k_1 \times k_2} \\ 0_{k_1 \times k_1} \end{bmatrix} \begin{bmatrix} G_{a\nu} \\ -G_{\beta(a)\nu} \\ G_{\beta(b)} \\ -G_{\beta(b)\nu} \end{bmatrix} \begin{bmatrix} G_{aa} \\ -G_{\beta(a)a} \\ G_{\beta(b)a} \\ -G_{\beta(b)a} \end{bmatrix} \begin{bmatrix} G_{ab} \\ -G_{\beta(a)b} \\ G_{\beta(b)b} \\ -G_{\beta(b)b} \end{bmatrix} \]  

(117)

In equations (113), (114), and (115), \( \dot{f} \) and all first derivatives of \( \dot{H} \) in row 1 are evaluated at \((a, \mathbf{x}(a), \lambda(a), \mu)\) as shown in Table 9, all first derivatives of \( \sigma \) in rows \( n + 2 \) through \( n + 1 + k_1 \) are evaluated at \((a, \mathbf{x}(a), \mu)\) as shown in Table 10, \( \dot{f} \) and all first derivatives of \( \dot{H} \) in row \( n + 2 + k_1 \) are evaluated at \((b, \mathbf{x}(b), \lambda(b), \mu)\) as shown in Table 11, and all first derivatives of \( \psi \) in rows \( 2n + 3 + k_1 \) through \( 2n + 2 + k_1 + k_2 \) are evaluated at \((b, \mathbf{x}(b), \mu)\) as shown in Table 12. Since \( \dot{H}_{\lambda} = \dot{\mathbf{f}}^T, \dot{H}_{\lambda} |_{a} = \dot{\mathbf{f}}^T |_{a} \) in row 1.
and columns \( n+1 \) through \( 2n \) of (113) and \( \hat{H}_\lambda \big|_b = \hat{f}_T \big|_b \) in row \( n+2+k_1 \) and columns \( n+1 \) through \( 2n \) of (114). In equations (116), (117), and (118), all second derivatives of \( G \) are evaluated at \( (a, x(a), x(b), \nu, \mu) \), while all first derivatives of \( \sigma \) and \( \psi \) are evaluated at \( (a, x(a), \mu) \) and \( (b, x(b), \mu) \), respectively, as shown in Tables 10 and 12. To simplify (116) and (117), Clairaut’s Theorem, \( G_\xi^T = \sigma \), and \( G_\nu^T = \psi \) are used to get \( G_a \xi = G_\xi^a = \sigma_\nu^a, G_{a \nu} = G_\nu^a = \psi_\nu^a = 0_{1 \times k_2} \), \( G_{x(a) \xi} = G_\xi^a = \sigma_\xi^a, \)

\[
G_{x(a) \nu} = G_\nu^a = \psi_\nu^a = 0_{1 \times k_2}, \quad G_{b \xi} = G_\xi^b = \sigma_\xi^b = 0_{1 \times k_1}, \quad G_{b \nu} = G_\nu^b = \psi_\nu^b, \]

\[
G_{x(b) \xi} = G_\xi^b = \sigma_\xi^b = 0_{1 \times k_1}, \quad G_{x(b) \nu} = G_\nu^b = \psi_\nu^b, \]

| Shorthand | Meaning |
|-----------|---------|
| \( H_a \big|_a = H_a (a, x(a), \lambda(a), \mu) \) | \( H_a (a, x(a), \lambda(a), \mu) \) |
| \( \hat{f}_T \big|_a = \hat{f}_T (a, x(a), \lambda(a), \mu) \) | \( \hat{f}_T (a, x(a), \lambda(a), \mu) \) |
| \( \hat{H}_\lambda \big|_a = \hat{H}_\lambda (a, x(a), \lambda(a), \mu) \) | \( \hat{H}_\lambda (a, x(a), \lambda(a), \mu) \) |

Table 9. Explanation of shorthand notation for \( \hat{f} \) and first derivatives of \( \hat{H} \) evaluated at \( a \) used in (113), (114), and (115). Note that \( \hat{H}_\lambda \big|_a = \hat{H}_\lambda (a, x(a), \lambda(a), \mu) = \hat{f}_T \big|_a \).

| Shorthand | Meaning |
|-----------|---------|
| \( \sigma_{x(a)} = \sigma_{x(a)} (a, x(a), \mu) \) | \( \sigma_{x(a)} (a, x(a), \mu) \) |
| \( \sigma_a = \sigma_a (a, x(a), \mu) \) | \( \sigma_a (a, x(a), \mu) \) |
| \( \sigma_\mu = \sigma_\mu (a, x(a), \mu) \) | \( \sigma_\mu (a, x(a), \mu) \) |

Table 10. Explanation of shorthand notation for first derivatives of \( \sigma \) used in (113), (114), and (115).

| Shorthand | Meaning |
|-----------|---------|
| \( H_b \big|_b = H_b (b, x(b), \lambda(b), \mu) \) | \( H_b (b, x(b), \lambda(b), \mu) \) |
| \( \hat{f}_T \big|_b = \hat{f}_T (b, x(b), \lambda(b), \mu) \) | \( \hat{f}_T (b, x(b), \lambda(b), \mu) \) |
| \( \hat{H}_\lambda \big|_b = \hat{H}_\lambda (b, x(b), \lambda(b), \mu) \) | \( \hat{H}_\lambda (b, x(b), \lambda(b), \mu) \) |

Table 11. Explanation of shorthand notation for \( \hat{f} \) and first derivatives of \( \hat{H} \) evaluated at \( b \) used in (113), (114), and (115). Note that \( \hat{H}_\lambda \big|_b = \hat{H}_\lambda (b, x(b), \lambda(b), \mu) = \hat{f}_T \big|_b \).

To express the boundary conditions (80)-(81) in terms of normalized dependent variables, let \( \Upsilon_1 (\hat{z}(0), \hat{z}(1), \mu) \equiv \Upsilon_1 (z(a), z(b), \mu), \quad \Upsilon_2 (\hat{z}(0), \hat{z}(1), \mu) \equiv \)
Table 12. Explanation of shorthand notation for first derivatives of $\psi$ used in (113), (114), and (115).

$$
\psi_{x(b)} = \psi_{x(b)}(b, x(b), \mu)
$$

$$
\psi_b = \psi_b(b, x(b), \mu)
$$

$$
\psi_\mu = \psi_\mu(b, x(b), \mu)
$$

The Jacobians of $\hat{Y}$ with respect to $\tilde{z}(0)$, $\tilde{z}(1)$, and $\mu$ are

$$
\hat{Y}_{\tilde{z}(0)} (\tilde{z}(0), \tilde{z}(1), \mu) = \hat{Y}_{1,\tilde{z}(0)} (\tilde{z}(0), \tilde{z}(1), \mu) - \hat{Y}_{2,\tilde{z}(0)} (\tilde{z}(0), \tilde{z}(1), \mu),
$$

$$
\hat{Y}_{\tilde{z}(1)} (\tilde{z}(0), \tilde{z}(1), \mu) = \hat{Y}_{1,\tilde{z}(1)} (\tilde{z}(0), \tilde{z}(1), \mu) - \hat{Y}_{2,\tilde{z}(1)} (\tilde{z}(0), \tilde{z}(1), \mu),
$$

and

$$
\hat{Y}_\mu (\tilde{z}(0), \tilde{z}(1), \mu) = \hat{Y}_{1,\mu} (\tilde{z}(0), \tilde{z}(1), \mu) - \hat{Y}_{2,\mu} (\tilde{z}(0), \tilde{z}(1), \mu),
$$

where the equality between the Jacobians of $\hat{Y}$, $\hat{Y}_1$, and $\hat{Y}_2$ with respect to $\tilde{z}(0)$, $\tilde{z}(1)$, and $\mu$ and the Jacobians of $\hat{Y}$, $\hat{Y}_1$, and $\hat{Y}_2$ with respect to $z(0)$, $z(1)$, and $\mu$ is given in Table 13.
Since the unknown constants $\xi$, $\nu$, $a$, and $b$ appear at the end of both $\tilde{z}(0)$ and $\tilde{z}(1)$, the unknown constants from only one of $\tilde{z}(0)$ and $\tilde{z}(1)$ are actually used to construct each term in $\tilde{\Upsilon}$ involving $\xi$, $\nu$, $a$, and $b$. The trailing columns in (123) are actually the Jacobian of $\tilde{\Upsilon}$ with respect to $\xi$, $\nu$, $a$, and $b$ in $\tilde{z}(0)$, while the trailing columns in (124) are actually the Jacobian of $\tilde{\Upsilon}$ with respect to $\xi$, $\nu$, $a$, and $b$ in $\tilde{z}(1)$. Thus, the trailing columns in (123) and (124) corresponding to the Jacobian of $\tilde{\Upsilon}$ with respect to $\xi$, $\nu$, $a$, and $b$ should not coincide in a software implementation. For example, if the unknown constants are extracted from $\tilde{z}(0)$ to construct $\tilde{\Upsilon}$, $\tilde{\Upsilon}_{\tilde{z}(0)}$ is as shown in (123) while the trailing columns in (124) corresponding to the Jacobian of $\tilde{\Upsilon}$ with respect to the unknown constants in $\tilde{z}(0)$ should be all zeros. Alternatively, if the unknown constants are extracted from $\tilde{z}(1)$ to construct $\tilde{\Upsilon}$, $\tilde{\Upsilon}_{\tilde{z}(1)}$ is as shown in (124) while the trailing columns in (123) corresponding to the Jacobian of $\tilde{\Upsilon}$ with respect to the unknown constants in $\tilde{z}(1)$ should be all zeros.

**B.3. Final Details.** In equations (82), (83), (84), (85), and (87), the second to last row is needed only if the initial time $a$ is free and the last row and column are needed only if the final time $b$ is free. In equation (86), the second to last row and column are needed only if the initial time $a$ is free and the last row and column are needed only if the final time $b$ is free.

In equations (106), (107), (108), (109), (112), (115), (118), (119), (120), (121), (122), and (125) the first row is needed only if the initial time $a$ is free and row $n + k_1 + 2$ is needed only if the final time $b$ is free. In equations (110), (111), (113), (114), (116), (117), (123), and (124) the first row and second to last column are needed only if the initial time $a$ is free and row $n + k_1 + 2$ and the last column are needed only if the final time $b$ is free.

In order to numerically solve the ODE TPBVP (79), (80), and (81) without continuation or with a monotonic continuation solver (such as acdc or acdcc), the solver should be provided (85), (86), (121), (123), and (124). In order to numerically solve the ODE TPBVP (79), (80), and (81) with a non-monotonic continuation solver (such as the predictor-corrector methods discussed in Appendices C and D), the solver should be provided (85), (86), (87), (121), (123), (124), and (125).

| Normalized | Un-Normalized |
|------------|---------------|
| $\tilde{\Upsilon}_{\tilde{z}(0)} (\tilde{z}(0), \tilde{z}(1), \mu)$ | $\tilde{\Upsilon}_{\tilde{z}(a)} (\tilde{z}(a), \tilde{z}(b), \mu)$ |
| $\tilde{\Upsilon}_{1,\tilde{z}(0)} (\tilde{z}(0), \tilde{z}(1), \mu)$ | $\tilde{\Upsilon}_{1,\tilde{z}(a)} (\tilde{z}(a), \tilde{z}(b), \mu)$ |
| $\tilde{\Upsilon}_{2,\tilde{z}(0)} (\tilde{z}(0), \tilde{z}(1), \mu)$ | $\tilde{\Upsilon}_{2,\tilde{z}(a)} (\tilde{z}(a), \tilde{z}(b), \mu)$ |
| $\tilde{\Upsilon}_{\tilde{z}(1)} (\tilde{z}(0), \tilde{z}(1), \mu)$ | $\tilde{\Upsilon}_{\tilde{z}(b)} (\tilde{z}(a), \tilde{z}(b), \mu)$ |
| $\tilde{\Upsilon}_{1,\tilde{z}(1)} (\tilde{z}(0), \tilde{z}(1), \mu)$ | $\tilde{\Upsilon}_{1,\tilde{z}(b)} (\tilde{z}(a), \tilde{z}(b), \mu)$ |
| $\tilde{\Upsilon}_{2,\tilde{z}(1)} (\tilde{z}(0), \tilde{z}(1), \mu)$ | $\tilde{\Upsilon}_{2,\tilde{z}(b)} (\tilde{z}(a), \tilde{z}(b), \mu)$ |
| $\tilde{\Upsilon}_{\mu} (\tilde{z}(0), \tilde{z}(1), \mu)$ | $\tilde{\Upsilon}_{\mu} (\tilde{z}(a), \tilde{z}(b), \mu)$ |
| $\tilde{\Upsilon}_{1,\mu} (\tilde{z}(0), \tilde{z}(1), \mu)$ | $\tilde{\Upsilon}_{1,\mu} (\tilde{z}(a), \tilde{z}(b), \mu)$ |
| $\tilde{\Upsilon}_{2,\mu} (\tilde{z}(0), \tilde{z}(1), \mu)$ | $\tilde{\Upsilon}_{2,\mu} (\tilde{z}(a), \tilde{z}(b), \mu)$ |

Table 13. Equality between Jacobians of two-point boundary condition functions in normalized and un-normalized coordinates.
The first and second derivatives required to construct (85), (86), (87), (121), (123), (124), and (125) are generally quite tedious to derive manually. Instead, symbolic differentiation [39], complex/bicomplex step differentiation [65, 52, 53, 49], dual/hyper-dual numbers [31, 33, 32, 56], and automatic differentiation [25, 55] are computational alternatives. If fact, it may be shown that the use of dual/hyper-dual numbers [31, 33, 32, 56], and automatic differentiation [25, 55] are more robust and broadly-applicable.

Therefore, while (85), (86), (87), (121), (123), (124), and (125) are complicated, they may be readily constructed numerically through automatic differentiation of \( H, \pi, f, G, \sigma, \) and \( \psi \) if \( \pi \) is given analytically and of \( H, f, G, \sigma, \) and \( \psi \) if the value of \( \pi \) is constructed numerically via Newton’s method as in (71). There are many free automatic differentiation toolboxes available [1], such as the MATLAB automatic differentiation toolbox ADiGator [70, 69]. Moreover, ADiGator is able to construct vectorized automatic derivatives, which is extremely useful for realizing the vectorized version of (85), (86), and (87), as the non-vectorized version of these equations execute too slowly in MATLAB to solve the ODE TPBVP (79), (80), and (81) in a timely manner.

Appendix C. Predictor-Corrector Continuation Method for Solving an ODE TPBVP.

C.1. Introduction. Suppose it is desired to solve the ODE TPBVP:

\[
\frac{d}{ds} y(s) = F(s, y(s), \lambda),
\]

\[
G(y(a), y(b), \lambda) = 0_{n \times 1},
\]  

where \( a, b \in \mathbb{R} \) are prescribed with \( a < b \), \( s \in [a, b] \subset \mathbb{R} \) is the independent variable, \( n \in \mathbb{N} \) is the prescribed number of dependent variables in \( y, y: [a, b] \to \mathbb{R}^n \) is an unknown function which must be solved for, \( \lambda \in \mathbb{R} \) is a prescribed scalar parameter, \( F: [a, b] \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n \) is a prescribed ODE velocity function defining the velocity of \( y \), and \( G: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n \) is a prescribed two-point boundary condition function. Observe that if \( n = 1, y, F, \) and \( G \) are scalar-valued functions, while if \( n > 1, y, F, \) and \( G \) are vector-valued functions. The Jacobian of \( F \) with respect to \( y \) is \( F_Y: [a, b] \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times n} \) and the Jacobian of \( F \) with respect to \( \lambda \) is \( F_\lambda: [a, b] \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times 1} \). The Jacobian of \( G \) with respect to \( y(a) \) is \( G_{y(a)}: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times 1} \), the Jacobian of \( G \) with respect to \( y(b) \) is \( G_{y(b)}: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times 1} \), and the Jacobian of \( G \) with respect to \( \lambda \) is \( G_\lambda: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times 1} \). If \( F \) is linear in \( y \) and \( G \) is linear in \( y(a) \) and \( y(b) \), then (126) is said to be a linear ODE TPBVP; otherwise, (126) is said to be a nonlinear ODE TPBVP.

Note that a solution \( y \) to (126) depends on the given value of the scalar parameter \( \lambda \), so a solution to (126) will be denoted by the pair \((y, \lambda)\). Usually it is not possible to solve (126) analytically. Instead, a numerical method such as a shooting, finite-difference, or Runge-Kutta method (collocation is a special kind of Runge-Kutta method) must be utilized to construct an approximate solution to (126). All such numerical methods require an initial solution guess and convergence to a solution is guaranteed only if the initial solution guess is sufficiently near the solution. Thus, solving (126) numerically requires construction of a good initial solution guess.
One way to construct a good initial solution guess for (126) is through continuation in the scalar parameter $\lambda$. If $(y_1, \lambda_1)$ solves (126) and it is desired to solve (126) for $\lambda = \lambda_F$, it may be possible to construct a finite sequence of solutions $\{(y_j, \lambda_j)\}_{j=1}^{J+1}$ starting at the known solution $(y_1, \lambda_1) = (y_1, \lambda_1)$ and ending at the desired solution $(y_{j+1}, \lambda_{j+1}) = (y_F, \lambda_F)$, using the previous solution $(y_j, \lambda_j)$ as an initial solution guess for the numerical solver to obtain the next solution $(y_{j+1}, \lambda_{j+1})$, $1 \leq j \leq J$, in the sequence. $J \in \mathbb{N}$ denotes the number of solutions in the sequence after the known solution.

This appendix describes a particular such continuation method, called predictor-corrector continuation, for solving (126). The treatment given here follows [12]. In the literature, predictor-corrector continuation is also called path-following [45], predictor-corrector path-following [2], and differential path-following [18]. AUTO [30], COLCON [8], and the algorithm presented in [68] are Fortran predictor-corrector continuation codes, while Chebfun's followpath [12], and COCO [26] are MATLAB predictor-corrector continuation codes. All these codes rely on global methods for solving ODE BVPs (e.g. Runge-Kutta, collocation, and finite-difference schemes), which are more robust than initial value methods for solving ODE BVPs (i.e. single and multiple shooting) because initial value methods cannot integrate unstable ODEs [4, 54, 13].

Before delving into the details, some functional analysis is reviewed which is necessary to understand how the predictor-corrector continuation method is applied to solve (126).

### C.2. A Hilbert Space

Let $\mathcal{H} = \{(y, \lambda) : y \in L^2([a,b], \mathbb{R}^n), \lambda \in \mathbb{R}\}$. $\mathcal{H}$ is a Hilbert space over $\mathbb{R}$. If $\alpha, \beta \in \mathbb{R}$ and $(y, \lambda), (\tilde{y}, \tilde{\lambda}) \in \mathcal{H}$, then

$$\alpha (y, \lambda) + \beta (\tilde{y}, \tilde{\lambda}) = (\alpha y + \beta \tilde{y}, \alpha \lambda + \beta \tilde{\lambda}),$$

(127)

the inner product on $\mathcal{H}$ is

$$\langle (y, \lambda), (\tilde{y}, \tilde{\lambda}) \rangle = \int_a^b y^T(s)\tilde{y}(s)ds + \lambda \tilde{\lambda},$$

(128)

and the norm on $\mathcal{H}$, induced by the inner product, is

$$\| (y, \lambda) \| = \langle (y, \lambda), (y, \lambda) \rangle^{\frac{1}{2}} = \left[ \int_a^b y^T(s)y(s)ds + \lambda^2 \right]^{\frac{1}{2}}.$$  

(129)

$(y, \lambda) \in \mathcal{H}$ and $(\tilde{y}, \tilde{\lambda}) \in \mathcal{H}$ are said to be orthogonal iff $\langle (y, \lambda), (\tilde{y}, \tilde{\lambda}) \rangle = 0$, and $(y, \lambda) \in \mathcal{H}$ is said to be of unit length iff $\| (y, \lambda) \| = 1$.

### C.3. The Fréchet Derivative and Newton’s Method

Given a function $F : \mathbb{R}^n \to \mathbb{R}^m$, recall that ordinary vector calculus defines the Jacobian of $F$ as the function $F' : \mathbb{R}^n \to \mathbb{R}^{m \times n}$ such that $F'(x)$ is the linearization of $F$ at $x \in \mathbb{R}^n$. Given normed vector spaces $V$ and $W$ and an open subset $U$ of $V$, the Fréchet derivative is an extension of the Jacobian to an operator $F : U \to W$. Before giving the definition of the Fréchet derivative, recall that $L(V, W)$ denotes the space of continuous linear operators from $V$ to $W$. Now for the definition of the Fréchet derivative, which comes from Definition 2.2.4 of [12].
**Definition C.1.** Suppose that $V$ and $W$ are normed vector spaces, and let $U$ be an open subset of $V$. Then the operator $F : U \rightarrow W$ is said to be Fréchet differentiable at $u \in U$ if and only if there exists an operator $L \in L(V,W)$ such that

$$
\lim_{\|h\|_V \to 0} \frac{\|F(u + h) - F(u) - Lh\|_W}{\|h\|_V} = 0.
$$

(130)

The operator $L$ is then called the Fréchet derivative of $F$ at $u$, often denoted by $F'(u)$. If $F$ is Fréchet differentiable at all points in $U$, $F$ is said to be Fréchet differentiable in $U$.

Given a function $H : \mathbb{R}^m \rightarrow \mathbb{R}^m$, Newton’s method is an algorithm to solve $H(x) = 0$ for $x \in \mathbb{R}^m$ and $0 \in \mathbb{R}^m$ when $H$ satisfies certain mild conditions. Starting from an initial solution guess $x_0 \in \mathbb{R}^m$ sufficiently close to a solution, Newton’s method converges to a solution of $H(x) = 0$ by iteratively solving the equations

$$
H'(x_k) \delta x_k = -H(x_k), \quad x_{k+1} = x_k + \delta x_k,
$$

(131)

starting at $k = 0$, where $H'$ denotes the Jacobian of $H$ and $x_k, \delta x_k \in \mathbb{R}^m$ for $k \geq 0$. The iteration in (131) continues until $H(x_k) \approx 0$ (or $\delta x_k \approx 0$) or until $k$ exceeds a maximum iteration threshold.

Now consider an operator $\mathcal{H} : U \subset V \rightarrow W$, where $V$ and $W$ are Banach spaces and $U$ is an open subset of $V$. Kantorovich [42] provided an extension of Newton’s method to solve $\mathcal{H}(u) = 0$ for $u \in U$ and $0 \in W$ when $\mathcal{H}$ satisfies certain mild conditions. Starting from an initial solution guess $u_0 \in U$ sufficiently close to a solution, Kantorovich’s extension of Newton’s method converges to a solution of $\mathcal{H}(u) = 0$ by iteratively solving the equations

$$
\mathcal{H}'(u_k) \delta u_k = -\mathcal{H}(u_k), \quad u_{k+1} = u_k + \delta u_k,
$$

(132)

starting at $k = 0$, where $\mathcal{H}'$ denotes the Fréchet derivative of $\mathcal{H}$ and $u_k, \delta u_k \in U$ for $k \geq 0$. The iteration in (132) continues until $\mathcal{H}(u_k) \approx 0$ (or $\delta u_k \approx 0$) or until $k$ exceeds a maximum iteration threshold.

**C.4. The Davidenko ODE IVP.** To motivate the predictor-corrector continuation method, the Davidenko ODE IVP is first presented. Let $C = \{(y, \lambda) : (y, \lambda) \text{ solves (126)}\}$ denote the solution manifold of (126). Suppose the solution manifold $C$ is parameterized by arclength $\nu$, so that an element of $C$ is $(y(\nu), \lambda(\nu))$, the tangent $(v(\nu), \tau(\nu))$ to $C$ at $(y(\nu), \lambda(\nu))$ satisfies $\|v(\nu), \tau(\nu)\|^2 = \int_a^b \sqrt{v'(s, \nu) v(s, \nu) ds} + [\tau(\nu)]^2 = 1$ (i.e. $(v(\nu), \tau(\nu))$ is a unit tangent), and the solution manifold $C$ can be described as a solution curve. With this arclength parameterization, $y : [a, b] \times \mathbb{R} \rightarrow \mathbb{R}^n$, $\lambda : \mathbb{R} \rightarrow \mathbb{R}$, $v : [a, b] \times \mathbb{R} \rightarrow \mathbb{R}^n$, $\tau : \mathbb{R} \rightarrow \mathbb{R}$, $y(\nu)$ is shorthand for $y(\cdot, \nu) : [a, b] \rightarrow \mathbb{R}^n$, and $v(\nu)$ is shorthand for $v(\cdot, \nu) : [a, b] \rightarrow \mathbb{R}^n$. Note that the components of the unit tangent $(v(\nu), \tau(\nu))$ to $C$ at $(y(\nu), \lambda(\nu))$ are given explicitly by $v(s, \nu) = \frac{\partial y(s, \nu)}{\partial \nu}$ and $\tau(\nu) = \frac{\partial \lambda(\nu)}{\partial \nu}$.

The Fréchet derivative of the ODE TPBVP (126) with respect to $\nu$ about the solution $(y(\nu), \lambda(\nu))$, in conjunction with the arclength constraint and the initial
condition \((y_1, \lambda_1)\), gives the nonlinear ODE IVP in the independent arclength variable \(\nu\):

\[
\frac{d}{ds} v(s, \nu) = F_y(s, y(s, \nu), \lambda(\nu)) v(s, \nu) + F_\lambda(s, y(s, \nu), \lambda(\nu)) \tau(\nu),
\]

\[
0_{n \times 1} = G_{y(a)}(y(a, \nu), y(b, \nu), \lambda(\nu)) v(a, \nu)
+ G_{y(b)}(y(a, \nu), y(b, \nu), \lambda(\nu)) v(b, \nu)
+ G_\lambda(y(a, \nu), y(b, \nu), \lambda(\nu)) \tau(\nu),
\]

\[
||\langle v(\nu), \tau(\nu) \rangle||^2 = \langle v(\nu), \tau(\nu) \rangle, \langle v(\nu), \tau(\nu) \rangle = \int_a^b v^T(s, \nu)v(s, \nu)ds + ||\tau(\nu)||^2 = 1,
\]

\[
(y(\nu_0), \lambda(\nu_0)) = (y_1, \lambda_1),
\]

(133)

which must be solved for \((y(\nu), \lambda(\nu))\) starting at \(\nu_0\) from an initial solution \((y_1, \lambda_1)\) of (126). (133) is called the Davidenko ODE IVP and its solution is called the Davidenko flow [27, 28, 62, 29, 12, 15]. The first two equations in (133) constitute the Fréchet derivative of the ODE TPBVP (126), the third equation is the arclength constraint, and the final equation is the initial condition. By introducing a dummy scalar-valued function \(w\) to represent the integrand of the arclength constraint, (133) can be re-written:

\[
\frac{d}{ds} v(s, \nu) = F_y(s, y(s, \nu), \lambda(\nu)) v(s, \nu) + F_\lambda(s, y(s, \nu), \lambda(\nu)) \tau(\nu),
\]

\[
\frac{d}{ds} w(s, \nu) = v^T(s, \nu)v(s, \nu),
\]

\[
0_{n \times 1} = G_{y(a)}(y(a, \nu), y(b, \nu), \lambda(\nu)) v(a, \nu)
+ G_{y(b)}(y(a, \nu), y(b, \nu), \lambda(\nu)) v(b, \nu)
+ G_\lambda(y(a, \nu), y(b, \nu), \lambda(\nu)) \tau(\nu),
\]

\[
w(a, \nu) = 0,
\]

\[
w(b, \nu) + ||\tau(\nu)||^2 - 1 = 0,
\]

\[
(y(\nu_0), \lambda(\nu_0)) = (y_1, \lambda_1).
\]

(134)

Again, letting \(\nu\) vary, (134) is a nonlinear ODE IVP which must be solved for \((y(\nu), \lambda(\nu))\) (i.e. \(y: [a, b] \times \mathbb{R} \rightarrow \mathbb{R}^n\) and \(\lambda: \mathbb{R} \rightarrow \mathbb{R}\) starting at \(\nu_0\) from an initial solution \((y_1, \lambda_1)\) of (126). However, for a fixed \(\nu\), (134) is a nonlinear ODE TPBVP which must be solved for \(v(\cdot, \nu): [a, b] \rightarrow \mathbb{R}^n, \tau(\nu) \in \mathbb{R}\), and \(w(\cdot, \nu): [a, b] \rightarrow \mathbb{R}\) and where the independent variable is \(s \in [a, b]\).

As explained in Chapter 5 of [29], it is inadvisable to integrate the Davidenko ODE IVP (133), or equivalently (134). Instead, a predictor-corrector continuation method, depicted in Figure 15 and explained in detail in the following subappendices, is used to generate a solution sequence \(\{(y_j, \lambda_j)\}_{j=1}^J\) which is a discrete subset of the Davidenko flow such that \((y_1, \lambda_1) = (y_1, \lambda_1)\).

C.5. Construct the Tangent. Given a solution \((y_j, \lambda_j)\) to (126) and a unit tangent \((v_{j-1}, \tau_{j-1})\) to the previous solution \((y_{j-1}, \lambda_{j-1})\) to (126), we seek to construct a tangent \((v_j, \tau_j)\) to the solution curve \(C\) at \((y_j, \lambda_j)\) which is roughly of unit length.
The arclength constraint is
\[ \| (v_j, \tau_j) \|^2 = \langle (v_j, \tau_j), (v_j, \tau_j) \rangle = \int_a^b v_j^T(s)v_j(s)ds + \tau_j^2 = 1, \] (135)
which is nonlinear in the tangent \((v_j, \tau_j)\). An alternative constraint, the pseudo-arclength constraint, is
\[ \langle (v_{j-1}, \tau_{j-1}), (v_j, \tau_j) \rangle = \int_a^b v_{j-1}^T(s)v_j(s)ds + \tau_{j-1} \tau_j = 1, \] (136)
which, in contrast to the arclength constraint (135), is linear in the tangent \((v_j, \tau_j)\). The linearization (i.e. Fréchet derivative) of the ODE TPBVP (126) about the solution \((y_j, \lambda_j)\), in conjunction with the pseudo-arclength condition (136), gives the linear ODE TPBVP:
\[
\frac{d}{ds} v_j(s) = F_y(s, y_j(s), \lambda_j) v_j(s) \\
+ F_\lambda(s, y_j(s), \lambda_j) \tau_j, \\
\frac{d}{ds} \tau_j = 0, \\
\frac{d}{ds} w(s) = v_{j-1}^T(s)v_j(s), \\
G_{y(a)}(y_j(a), y_j(b), \lambda_j) v_j(a) \\
+ G_{y(b)}(y_j(a), y_j(b), \lambda_j) v_j(b) \\
+ G_\lambda(y_j(a), y_j(b), \lambda_j) \tau_j = 0_{n \times 1}, \\
w(a) = 0, \\
w(b) + \tau_{j-1} \tau_j - 1 = 0, 
\] (137)
which must be solved for \(v_j: [a, b] \rightarrow \mathbb{R}^n, \tau_j \in \mathbb{R}\), and \(w: [a, b] \rightarrow \mathbb{R}\) and where \((v_j, \tau_j)\) is a tangent to \(C\) at \((y_j, \lambda_j)\). Note that the first, second, and third equations in (137) are the ODEs, while the fourth, fifth, and sixth equations constitute the boundary conditions. The first, second, and fourth equations in (137) are the linearization (i.e. Fréchet derivative) of (126) about the solution \((y_j, \lambda_j)\) and ensure that a tangent is produced, while the third, fifth, and sixth equations in (137) enforce the pseudo-arclength condition (136). The initial solution guess to solve (137) is \((v_j, \tau_j) = (v_{j-1}, \tau_{j-1})\) and \(w(s) = \int_a^s v_{j-1}(\tilde{s})v_{j-1}(\tilde{s})d\tilde{s}, s \in [a, b], \) for \(j \geq 1\). For
$j = 1$, define $(v_0, \tau_0) = (0_{n \times 1}, 1)$. Note that the construction of the initial guess for $w$ can be realized efficiently via the MATLAB routine cumtrapz.

Note that the linear ODE TPBVP (137) can be solved numerically via the MATLAB routines sbvp or bvp4c, which offers 4 algorithms: twpbvp_m, twpbvpc_m, twpbvp_l, and twpbvpc_l; moreover, sbvp and bvp4c have special algorithms to solve linear ODE TPBVPs. Since $y_j$ and $v_{j-1}$ are usually only known at a discrete set of points in $[a, b]$, the values of these functions at the other points in $[a, b]$ must be obtained through interpolation in order to numerically solve (137). The MATLAB routine interp1 performs linear, cubic, pchip, makima, and spline interpolation and may be utilized to interpolate $y_j$ and $v_{j-1}$ while solving (137).

Because the numerical solvers usually converge faster when provided Jacobians of the ODE velocity function and of the two-point boundary condition function, these are computed below. Let

$$x = \begin{bmatrix} v_j \\ \tau_j \\ w \end{bmatrix}. \quad (138)$$

The ODE velocity function in (137) is

$$H^t(s, x(s)) = H^t(s, v_j(s), \tau_j, w(s)) = \begin{bmatrix} F_y(s, y_j(s), \lambda_j) v_j(s) + F_\lambda(s, y_j(s), \lambda_j) \tau_j \\ 0 \\ v_{j-1}^T(s) v_j(s) \end{bmatrix}. \quad (139)$$

The Jacobian of the ODE velocity function $H^t$ with respect to $x$ is

$$H^t_x(s, x(s)) = H^t_x(s, v_j(s), \tau_j, w(s)) = \begin{bmatrix} F_y(s, y_j(s), \lambda_j) & F_\lambda(s, y_j(s), \lambda_j) & 0_{n \times 1} \\ 0_{1 \times n} & 0 & 0 \\ v_{j-1}^T(s) & 0 & 0 \end{bmatrix}. \quad (140)$$

The two-point boundary condition in (137) is

$$K^t(x(a), x(b)) = 0_{(n+2) \times 1}, \quad (141)$$

where $K^t$ is the two-point boundary condition function

$$K^t(x(a), x(b)) = \begin{bmatrix} G_y (y_j(a), y_j(b), \lambda_j) v_j(a) + G_y (y_j(a), y_j(b), \lambda_j) v_j(b) + G_\lambda (y_j(a), y_j(b), \lambda_j) \tau_j \\ w(a) \\ w(b) + \tau_{j-1} \tau_j - 1 \end{bmatrix}. \quad (142)$$

The Jacobians of the two-point boundary condition function $K^t$ with respect to $x(a)$ and $x(b)$ are

$$K^t_x(x(a), x(b)) = \begin{bmatrix} G_y (y_j(a), y_j(b), \lambda_j) & G_\lambda (y_j(a), y_j(b), \lambda_j) & 0_{n \times 1} \\ 0_{1 \times n} \\ 0_{1 \times n} \end{bmatrix}, \quad (143)$$

and

$$K^t_x(x(a), x(b)) = \begin{bmatrix} G_y (y_j(a), y_j(b), \lambda_j) & G_\lambda (y_j(a), y_j(b), \lambda_j) & 0_{n \times 1} \\ 0_{1 \times n} \\ 0_{1 \times n} \end{bmatrix}. \quad (144)$$
Special care must be taken when implementing the Jacobians (143) and (144). Since the unknown constant \( \tau_j \) appears as the second to last element of both \( x(a) \) and \( x(b) \), \( \tau_j \) from only one of \( x(a) \) and \( x(b) \) is actually used to construct each term in \( K^t \) involving \( \tau_j \). The middle column of (143) is actually the derivative of \( K^t \) with respect to the \( \tau_j \) in \( x(a) \), while the middle column of (144) is actually the derivative of \( K^t \) with respect to the \( \tau_j \) in \( x(b) \). Thus, the middle columns in (143) and (144) corresponding to the derivative of \( K^t \) with respect to \( \tau_j \) should not coincide in a software implementation. For example, if \( K^t \) is constructed from the \( \tau_j \) in \( x(a) \), \( K^t_{x(a)} \) is as shown in (143) while the middle column of (144) corresponding to the derivative of \( K^t \) with respect to the \( \tau_j \) in \( x(b) \) is all zeros. Alternatively, if \( K^t \) is constructed from the \( \tau_j \) in \( x(b) \), \( K^t_{x(b)} \) is as shown in (144) while the middle column of (143) corresponding to the derivative of \( K^t \) with respect to the \( \tau_j \) appearing in \( x(a) \) is all zeros.

C.6. **Normalize the Tangent.** The tangent \((v_j, \tau_j)\) at \((y_j, \lambda_j)\) obtained by solving (137) in the previous step is only roughly of unit length. A unit tangent at \((y_j, \lambda_j)\) is obtained from \((v_j, \tau_j)\) through normalization:

\[
(v_j, \tau_j) \leftarrow \frac{1}{\kappa} (v_j, \tau_j),
\]

(145)

where

\[
\kappa = \|(v_j, \tau_j)\| = \langle (v_j, \tau_j), (v_j, \tau_j) \rangle^{\frac{1}{2}} = \left[ \int_a^b v_j^T(s)v_j(s)ds + \tau_j^2 \right]^{\frac{1}{2}}.
\]

(146)

The integration operator to construct the normalization scalar \( \kappa \) in (146) can be realized via the MATLAB routine `trapz`.

C.7. **Construct the Tangent Predictor.** The unit tangent \((v_j, \tau_j)\) constructed in (145) is used to obtain a guess (the so-called “tangent predictor”) \((y^*_j, \lambda^*_j)\) for the next solution \((y_{j+1}, \lambda_{j+1})\) as follows:

\[
(y^*_j, \lambda^*_j) = (y_j, \lambda_j) + \sigma (v_j, \tau_j),
\]

(147)

where \( \sigma \in [\sigma_{\text{min}}, \sigma_{\text{max}}] \) is a steplength and where \( 0 < \sigma_{\text{min}} \leq \sigma_{\text{max}} \). Concretely, \( \sigma_{\text{min}} \) might be .0001 and \( \sigma_{\text{max}} \) might be \( \frac{1}{2} \). \( \sigma \) is adapted during the predictor-corrector continuation method based on the corrector step, discussed in the next subappendix. Initially, the value of \( \sigma \) is set to \( \sigma_{\text{init}} \in [\sigma_{\text{min}}, \sigma_{\text{max}}] \). The notation \((y^*_j, \lambda^*_j)\) is used to denote the tangent predictor in (147) because, as discussed in the next subappendix, the tangent predictor is used as the initial corrector in an iterative Newton’s method that projects the tangent predictor onto \( C \).

C.8. **Construct the Corrector.** Since the tangent predictor \((y^*_j, \lambda^*_j)\) constructed in (147) does not necessarily lie on \( C \), \((y^*_j, \lambda^*_j)\) must be projected onto \( C \) to obtain the next solution (the so-called “corrector”) \((y_{j+1}, \lambda_{j+1})\). This projection process is the corrector step. In order to perform the projection efficiently, the difference between the next solution and the tangent predictor, \((y_{j+1}, \lambda_{j+1}) - (y^*_j, \lambda^*_j)\), should
be orthogonal to the unit tangent \((v_j, \tau_j)\). That is, the orthogonality constraint is
\[
((v_j, \tau_j), (v_{j+1}, \lambda_{j+1}) - (y^c_j, \lambda_j)) = \langle (v_j, \tau_j), (y^c_j - y^c_j) \rangle = \int_a^b v_j^T(s) [y_{j+1}(s) - y^c_j(s)] ds + \tau_j [\lambda_{j+1} - \lambda^c_j] = 0.
\]

(148)

The tangent predictor \((y^c_j, \lambda_j)\) can be iteratively corrected by applying Newton’s method to (126), while enforcing the orthogonality constraint (148), to generate a sequence of correctors \(\{(y^c_k, \lambda^c_k)\}_{k=1}^K\). Applying Newton’s method to the ODE TPBVP (126) about the current corrector \((y^c_k, \lambda^c_k)\), in conjunction with the orthogonality constraint (148), gives the linear ODE TPBVP:
\[
\frac{d}{ds} \delta y^c_k(s) = F_y(s, y^c_k(s), \lambda^c_k) \delta y^c_k(s) + F_\lambda(s, y^c_k(s), \lambda^c_k) \delta \lambda^c_k - \frac{d}{ds} y^c_k(s) + F(s, y^c_k(s), \lambda^c_k),
\]
\[
\frac{d}{ds} \delta \lambda^c_k = 0,
\]
\[
\frac{d}{ds} w(s) = v_j^T(s) \delta y^c_k(s),
\]

(149)

which must be solved for \(\delta y^c_k: [a, b] \rightarrow \mathbb{R}^n, \delta \lambda^c_k \in \mathbb{R}\), and \(w: [a, b] \rightarrow \mathbb{R}\) and where \((\delta y^c_k, \delta \lambda^c_k)\) represents a correction to the current corrector \((y^c_k, \lambda^c_k)\). Note that the first, second, and third equations in (149) are the ODEs, while the fourth, fifth, and sixth equations constitute the boundary conditions. The first, second, and fourth equations in (149) are the result of applying Newton’s method to (126) about the current corrector \((y^c_k, \lambda^c_k)\), while the third, fifth, and sixth equations in (149) enforce the orthogonality constraint (148). (149) must be solved iteratively for at most \(K\) iterations, so that \(1 \leq k \leq K\). The initial solution guess to solve (149) at the beginning of each iteration is \((\delta y^c_k, \delta \lambda^c_k) = (0_{n \times 1}, 0)\) and \(w(s) = 0, s \in [a, b]\). The initial corrector about which Newton’s method is applied in the first iteration is the tangent predictor \((y^c_1, \lambda^c_1)\). At the end of each iteration, the corrector about which Newton’s method is applied for the next iteration is updated via \((y^c_{k+1}, \lambda^c_{k+1}) = (y^c_k, \lambda^c_k) + (\delta y^c_k, \delta \lambda^c_k)\). At the end of each iteration, convergence to \(\mathcal{L}\) should be tested via:
\[
\frac{\|\delta y^c_k, \delta \lambda^c_k\|}{\|y^c_1, \lambda^c_1\|} = \left[\int_a^b [\delta y^c_k(s)]^T \delta y^c_k(s) ds + [\delta \lambda^c_k]^2\right]^{1/2} < \gamma,
\]

(150)
where $\gamma$ is a small threshold such as .001. Since Newotn’s method enjoys quadratic convergence near a solution, only a few (say $K = 5$) iterative solves of (149) should be attempted. If convergence has not been attained in $K$ iterations, the steplength $\sigma$ should be reduced:

$$\sigma \leftarrow \sigma_i \sigma,$$

(151)

where $\sigma_i$ is a reduction scale factor such as $\frac{1}{4}$ and the corrector step should be restarted at the new tangent predictor \((\mathbf{y}_j, \lambda^c_j) = (\mathbf{y}_j, \lambda_j) + \sigma (\mathbf{v}_j, \tau_j)\), based on the updated value of $\sigma$ realized in (151). If, as a result of the reduction realized in (151), $\sigma < \sigma_{\text{min}}$, the algorithm should halt and predictor-corrector continuation failed. However, if convergence has been achieved in $k \leq K$ iterations, the next solution can be taken to be \((\mathbf{y}_{j+1}, \lambda_{j+1}) = (\mathbf{y}_{k+1}, \lambda_{k+1})\) or the corrector can be further polished as explained in the next subappendix. Moreover, if convergence has been achieved rapidly in no more than $k_{\text{fast}}$ iterations, where $1 \leq k_{\text{fast}} \leq K$ and, concretely, $k_{\text{fast}}$ might be 3, then the steplength $\sigma$ may be increased:

$$\sigma \leftarrow \min \{\sigma; \sigma, \sigma_{\text{max}}\},$$

(152)

where $\sigma_i$ is an increase scale factor such as 2.

Note that the linear ODE TPBVP (149) can be solved numerically via the MATLAB routines sbvp or bvp1wp, which offers 4 algorithms: twpbvp.m, twpbvpcc.m, twpbvpc_1, and twpbvpcc_1; moreover, sbvp and bvp1wp have special algorithms to solve linear ODE TPBVPs. Since $\mathbf{y}_k^c$, $\frac{d}{ds}\mathbf{y}_k^c$, and $\mathbf{v}_j$ are usually only known at a discrete set of points in $[a, b]$, the values of these functions at the other points in $[a, b]$ must be obtained through interpolation in order to numerically solve (149). The MATLAB routine interp1 performs linear, cubic, pchip, makima, and spline interpolation and may be utilized to interpolate $\mathbf{y}_k^c$, $\frac{d}{ds}\mathbf{y}_k^c$, and $\mathbf{v}_j$ while solving (149).

Because the numerical solvers usually converge faster when provided Jacobians of the ODE velocity function and of the two-point boundary condition function, these are computed below. Let

$$\mathbf{x} = \begin{bmatrix} \delta \mathbf{y}_k^c \\ \delta \lambda_k^c \\ w \end{bmatrix}. $$

(153)

The ODE velocity function in (149) is

$$\mathbf{H}^e (s, \mathbf{x}(s)) = \mathbf{H}^e (s, \delta \mathbf{y}_k^c(s), \delta \lambda_k^c, w(s))$$

$$= \begin{bmatrix} \mathbf{F}_y (s, \mathbf{y}_k^c(s), \lambda_k^c) \delta \mathbf{y}_k^c(s) + \mathbf{F}_\lambda (s, \mathbf{y}_k^c(s), \lambda_k^c) \delta \lambda_k^c - \frac{d}{ds} \mathbf{y}_k^c(s) + \mathbf{F} (s, \mathbf{y}_k^c(s), \lambda_k^c) \\ 0 \\ \mathbf{v}_j(s) \end{bmatrix}. $$

(154)

The Jacobian of the ODE velocity function $\mathbf{H}^e$ with respect to $\mathbf{x}$ is

$$\mathbf{H}_x^e (s, \mathbf{x}(s)) = \mathbf{H}_x^e (s, \delta \mathbf{y}_k^c(s), \delta \lambda_k^c, w(s))$$

$$= \begin{bmatrix} \mathbf{F}_y (s, \mathbf{y}_k^c(s), \lambda_k^c) & \mathbf{F}_\lambda (s, \mathbf{y}_k^c(s), \lambda_k^c) & \mathbf{0}_{n \times 1} \\ \mathbf{0}_{1 \times n} & \mathbf{0} & \mathbf{0} \\ \mathbf{v}_j(s) & 0 & 0 \end{bmatrix}. $$

(155)

The two-point boundary condition in (149) is

$$\mathbf{K}^c (\mathbf{x}(a), \mathbf{x}(b)) = \mathbf{0}_{(n+2) \times 1}, $$

(156)
where $K^c$ is the two-point boundary condition function

$$K^c(x(a), x(b)) = \begin{bmatrix} G_{y(a)} (y_k^c(a), y_k^c(b), \lambda_k^c) \delta y_k^c(a) + G_{y(b)} (y_k^c(a), y_k^c(b), \lambda_k^c) \delta y_k^c(b) \\ + G_{\lambda} (y_k^c(a), y_k^c(b), \lambda_k^c) \delta \lambda_k^c + G (y_k^c(a), y_k^c(b), \lambda_k^c) \\ w(a) \\ w(b) + \tau_j \delta \lambda_k^c \end{bmatrix}. \quad (157)$$

The Jacobians of the two-point boundary condition function $K^c$ with respect to $x(a)$ and $x(b)$ are

$$K^c_{x(a)} (x(a), x(b)) = \begin{bmatrix} G_{y(a)} (y_k^c(a), y_k^c(b), \lambda_k^c) & G_{\lambda} (y_k^c(a), y_k^c(b), \lambda_k^c) & 0_{n \times 1} \\ 0_{1 \times n} & 0 & 1 \\ 0_{1 \times n} & \tau_j & 0 \end{bmatrix} \quad (158)$$

and

$$K^c_{x(b)} (x(a), x(b)) = \begin{bmatrix} G_{y(b)} (y_k^c(a), y_k^c(b), \lambda_k^c) & G_{\lambda} (y_k^c(a), y_k^c(b), \lambda_k^c) & 0_{n \times 1} \\ 0_{1 \times n} & 0 & 0 \\ 0_{1 \times n} & \tau_j & 1 \end{bmatrix}. \quad (159)$$

Special care must be taken when implementing the Jacobians (158) and (159). Since the unknown constant $\delta \lambda_k^c$ appears as the second to last element of both $x(a)$ and $x(b)$, $\delta \lambda_k^c$ from only one of $x(a)$ and $x(b)$ is actually used to construct each term in $K^c$ involving $\delta \lambda_k^c$. The middle column of (158) is actually the derivative of $K^c$ with respect to the $\delta \lambda_k^c$ in $x(a)$, while the middle column of (159) is actually the derivative of $K^c$ with respect to the $\delta \lambda_k^c$ in $x(b)$. Thus, the middle columns in (158) and (159) corresponding to the derivative of $K^c$ with respect to $\delta \lambda_k^c$ should not coincide in a software implementation. For example, if $K^c$ is constructed from the $\delta \lambda_k^c$ in $x(a)$, $K^c_{x(a)}$ is as shown in (158) while the middle column of (159) corresponding to the derivative of $K^c$ with respect to the $\delta \lambda_k^c$ in $x(b)$ is all zeros. Alternatively, if $K^c$ is constructed from the $\delta \lambda_k^c$ in $x(b)$, $K^c_{x(b)}$ is as shown in (159) while the middle column of (158) corresponding to the derivative of $K^c$ with respect to the $\delta \lambda_k^c$ in $x(a)$ is all zeros.

C.9. Polish the Corrector. The final corrector $(y_{k+1}^c, \lambda_{k+1}^c)$ from the previous step can be further polished by finding $(y_{j+1}, \lambda_{j+1})$ that solves (126) while satisfying the orthogonality constraint (148). This yields the ODE TPBVP:

$$\frac{d}{ds} y_{j+1} (s) = F (s, y_{j+1} (s), \lambda_{j+1}),$$

$$\frac{d}{ds} \lambda_{j+1} = 0,$$

$$\frac{d}{ds} w(s) = v_j^T (s) [y_{j+1} (s) - y_j^1 (s)],$$

$$G (y_{j+1} (a), y_{j+1} (b), \lambda_{j+1}) = 0_{n \times 1},$$

$$w(a) = 0,$$

$$w(b) + \tau_j [\lambda_{j+1} - \lambda_j^c] = 0,$$

which must be solved for $(y_{j+1}, [a, b] \rightarrow \mathbb{R}^n, \lambda_{j+1} \in \mathbb{R}$, and $w : [a, b] \rightarrow \mathbb{R}$. Note that the first, second, and third equations in (160) are the ODEs, while the fourth, fifth, and sixth equations constitute the boundary conditions. The first, second, and fourth equations in (160) ensure that the solution lies on $C$ (i.e., satisfies (126)), while the third, fifth, and sixth equations in (160) enforce the orthogonality constraint.
The initial solution guess to solve (160) is the final corrector \( (y_{k+1}, \lambda_{k+1}) \) from the previous step and \( w(s) = 0, s \in [a, b] \).

Note that the ODE TPBVP (160) can be solved numerically via the MATLAB routines \texttt{sbvp} or \texttt{bvptwp}, which offers 4 algorithms: \texttt{twpbvp.m}, \texttt{twpbvpcc.m}, \texttt{twpbvp2.m}, and \texttt{twpbvp2.1}. Since \( y_j^r \) and \( v_j \) are usually only known at a discrete set of points in \( [a, b] \), the values of these functions at the other points in \( [a, b] \) must be obtained through interpolation in order to numerically solve (160). The MATLAB routine \texttt{interp1} performs linear, cubic, pchip, makima, and spline interpolation and may be utilized to interpolate \( y_j^r \) and \( v_j \) while solving (160).

Because the numerical solvers usually converge faster when provided Jacobians of the ODE velocity function and of the two-point boundary condition function, these are computed below. Let

\[
\mathbf{x} = \begin{bmatrix} y_{j+1} \\ \lambda_{j+1} \\ w \end{bmatrix}.
\]  
(161)

The ODE velocity function in (160) is

\[
H^p(s, \mathbf{x}(s)) = H^p(s, y_{j+1}(s), \lambda_{j+1}, w(s)) = \begin{bmatrix} F(s, y_{j+1}(s), \lambda_{j+1}) \\ 0 \\ v_j^T(s) [y_{j+1}(s) - y_j^r(s)] \end{bmatrix}.
\]  
(162)

The Jacobian of the ODE velocity function \( H^p \) with respect to \( \mathbf{x} \) is

\[
H^p_x(s, \mathbf{x}(s)) = \begin{bmatrix} F_y(s, y_{j+1}(s), \lambda_{j+1}) & F_\lambda(s, y_{j+1}(s), \lambda_{j+1}) & 0_{n \times 1} \\ 0_{1 \times n} & 0 & 0 \\ v_j^T(s) & 0 & 0 \end{bmatrix}.
\]  
(163)

The two-point boundary condition in (160) is

\[
K^p(\mathbf{x}(a), \mathbf{x}(b)) = 0_{(n+2) \times 1},
\]  
(164)

where \( K^p \) is the two-point boundary condition function

\[
K^p(\mathbf{x}(a), \mathbf{x}(b)) = \begin{bmatrix} G_y(y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) & w(a) \\ w(b) + \tau_j [\lambda_{j+1} - \lambda_j^r] & 0_{n \times 1} \end{bmatrix}.
\]  
(165)

The Jacobians of the two-point boundary condition function \( K^p \) with respect to \( \mathbf{x}(a) \) and \( \mathbf{x}(b) \) are

\[
K^p_{\mathbf{x}(a)}(\mathbf{x}(a), \mathbf{x}(b)) = \begin{bmatrix} G_y(y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) & G_\lambda(y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) & 0_{n \times 1} \\ 0_{1 \times n} & 0 & 1 \\ 0_{1 \times n} & \tau_j & 0 \end{bmatrix},
\]  
(166)

and

\[
K^p_{\mathbf{x}(b)}(\mathbf{x}(a), \mathbf{x}(b)) = \begin{bmatrix} G_y(b, y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) & G_\lambda(b, y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) & 0_{n \times 1} \\ 0_{1 \times n} & 0 & 0 \\ 0_{1 \times n} & \tau_j & 1 \end{bmatrix},
\]  
(167)

Special care must be taken when implementing the Jacobians (166) and (167). Since the unknown constant \( \lambda_{j+1} \) appears as the second to last element of both \( \mathbf{x}(a) \) and \( \mathbf{x}(b) \), \( \lambda_{j+1} \) from only one of \( \mathbf{x}(a) \) and \( \mathbf{x}(b) \) is actually used to construct each term in \( K^p \) involving \( \lambda_{j+1} \). The middle column of (166) is actually the derivative of \( K^p \) with respect to the \( \lambda_{j+1} \) in \( \mathbf{x}(a) \), while the middle column of (167) is actually the derivative of \( K^p \) with respect to the \( \lambda_{j+1} \) in \( \mathbf{x}(b) \). Thus, the middle columns in
(166) and (167) corresponding to the derivative of $K$ with respect to $\lambda_{j+1}$ should not coincide in a software implementation. For example, if $K$ is constructed from the $\lambda_{j+1}$ in $x(a)$, $K^p_{x(a)}$ is as shown in (166) while the middle column of (167) corresponding to the derivative of $K$ with respect to the $\lambda_{j+1}$ in $x(b)$ is all zeros. Alternatively, if $K$ is constructed from the $\lambda_{j+1}$ in $x(b)$, $K^p_{x(b)}$ is as shown in (167) while the middle column of (166) corresponding to the derivative of $K$ with respect to the $\lambda_{j+1}$ appearing in $x(a)$ is all zeros.

C.10. Pseudocode for Predictor-Corrector Continuation. Below is pseudocode, split into two parts, that realizes the predictor-corrector continuation method.

**Algorithm 1** Predictor-Corrector Continuation for Nonlinear ODE TPBVPs. Part 1.

**Input:** ODE velocity function $F: [a, b] \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, two-point boundary condition function $G: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, and their Jacobians $F_y: [a, b] \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times n}$, $F_{\lambda}: [a, b] \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times 1}$, $G_y(a): \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times n}$, $G_{y(b)}: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times n}$, and $G_{\lambda}: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times 1}$. Initial point on the solution curve $C$, $(y_1, \lambda_1)$. Maximum number of points not including the initial point to be computed on $C$, $J$. Initial tangent steplength, $\sigma_{\text{init}}$. Minimum and maximum tangent steplengths permitted, $\sigma_{\text{min}}$ and $\sigma_{\text{max}}$. Tangent steplength reduction and increase scale factors, $\sigma_r$ and $\sigma_t$. Maximum number of Newton correction steps permitted, $K$. Maximum number of Newton correction steps for which a tangent steplength increase may occur if convergence is obtained, $k_{\text{fast}}$. Newton correction convergence threshold, $\gamma$. Tangent direction at the first solution, $d$. $d$ may be $-1$ or $1$. If $d$ is $-1$ or $1$, the first tangent is scaled by $d$. $\text{polish}$ is a Boolean that determines whether the Newton corrector solution is polished by solving (160).

**Output:** A solution curve $c$ or a flag indicating that the curve could not be traced.

**function** $\text{PAC,BVP}$

\[
(F, G, F_y, F_{\lambda}, G_{y(a)}, G_{y(b)}, G_{\lambda}, (y_1, \lambda_1), J, \sigma_{\text{init}}, \sigma_{\text{min}}, \sigma_{\text{max}}, \sigma_r, \sigma_t, K, k_{\text{fast}}, \\
\gamma, d, \text{polish})
\]

1: $\sigma \leftarrow \sigma_{\text{init}}$ \hspace{10mm} $\triangleright$ Set the initial tangent steplength.

2: $c(1) \leftarrow (y_1, \lambda_1)$ \hspace{10mm} $\triangleright$ Store the initial solution on $C$.

3: (v0, τ0) \leftarrow (0_n \times 1, 1)$ \hspace{10mm} $\triangleright$ Select an initial unit tangent. This choice forces $\tau_1 = 1$.

4: for $j = 1$ to $J$ \hspace{10mm} $\triangleright$ Trace the solution curve $C$.

5: Obtain a tangent $(v_j, \tau_j)$ to $C$ at $(y_j, \lambda_j)$ by solving (137) \hspace{10mm} $\triangleright$ starting from $(v_{j-1}, \tau_{j-1})$.

6: $\kappa \leftarrow \|(v_j, \tau_j)\|$ \hspace{10mm} $\triangleright$ Choose the direction of the tangent at the initial solution, based on $d$.

7: if $\kappa = 1$ then \hspace{10mm} $\triangleright$ Normalize the tangent.

8: $\kappa \leftarrow \text{sgn}(d) \kappa$

9: $\sigma \leftarrow \text{polish}(d) \kappa$

10: end if

11: $(v_j, \tau_j) \leftarrow \frac{1}{\kappa} (v_j, \tau_j)$

12: reject $\leftarrow$ TRUE
Algorithm 1: Predictor-Corrector Continuation for Nonlinear ODE TPBVPs. Part 2.

13: while reject do
14:   \((y_j^1, \lambda_j^1) \leftarrow (y_j, \lambda_j) + \sigma (v_j, \tau_j)\) \(\triangleright\) Take a tangent step of length \(\sigma\).
15:   for \(k = 1\) to \(K\) do
16:     Obtain a Newton correction \((\delta y_k^c, \delta \lambda_k^c)\) to \((y_k^c, \lambda_k^c)\) by solving \((149)\).
17:     Construct the Newton corrector.
18:     \((y_{k+1}^c, \lambda_{k+1}^c) \leftarrow (y_k^c, \lambda_k^c) + (\delta y_k^c, \delta \lambda_k^c)\)
19:     if \(\frac{\|\delta y_k^c, \delta \lambda_k^c\|}{\|y_k^c, \lambda_k^c\|} < \gamma\) then \(\triangleright\) Test for convergence to \(C\).
20:       reject \leftarrow FALSE
21:     if polish then
22:       Obtain the next solution \((y_{j+1}, \lambda_{j+1})\) on \(C\) by solving \((160)\) starting from \((y_{k+1}^c, \lambda_{k+1}^c)\).
23:     else \(\triangleright\) Accept the Newton corrector solution.
24:       \((y_{j+1}, \lambda_{j+1}) \leftarrow (y_{k+1}^c, \lambda_{k+1}^c)\)
25:     end if
26:     \(c(j+1) \leftarrow (y_{j+1}, \lambda_{j+1})\) \(\triangleright\) Store the new solution on \(C\).
27:   if \(k < k_{\text{fast}}\) then \(\triangleright\) Test for rapid Newton convergence.
28:     \(\triangleright\) Rapid Newton convergence, so increase the tangent steplength.
29:     \(\sigma \leftarrow \min\{\sigma, \sigma_{\text{max}}\}\)
30:   end if
31:   \(\triangleright\) Break out of the for loop since convergence to \(C\) has been achieved.
32:   break
33: end if
34: end for
35: if reject then \(\triangleright\) Too many Newton steps taken, so reduce the tangent steplength.
36:   \(\sigma \leftarrow \sigma_{\text{min}}\)
37:   if \(\sigma < \sigma_{\text{min}}\) then \(\triangleright\) Impossible to trace \(C\) because the tangent steplength is too small.
38:     print "Unable to trace \(C\) because the tangent steplength is too small: \(\sigma < \sigma_{\text{min}}\)."
39:     return flag
40: end if
41: end while
42: return \(c\)
43: end function

Appendix D. Sweep Predictor-Corrector Continuation Method for Solving an ODE TPBVP.

D.1. Introduction. In this appendix, an alternative predictor-corrector continuation method is presented that exploits a monotonic continuation ODE TPBVP solver, such as bvptwp’s acdc or acdcc, to monotonically increase (i.e. sweep) the
tangent steplength $\sigma$ from 0 up until a maximum threshold $\sigma_{\text{max}}$ is reached or until the next turning point is reached.

**D.2. Construct the Tangent.** Given a solution $(y_j, \lambda_j)$ to (126), we seek to construct a unit tangent $(v_j, \tau_j)$ to the solution curve $C$ at $(y_j, \lambda_j)$. Recall the arclength constraint

$$\| (v_j, \tau_j) \|^2 = \langle (v_j, \tau_j), (v_j, \tau_j) \rangle = \int_a^b v_j^T(s)v_j(s)ds + \tau_j^2 = 1. \quad (168)$$

The linearization (i.e. Fréchet derivative) of the ODE TPBVP (126) about the solution $(y_j, \lambda_j)$, in conjunction with the arclength constraint (168), gives the nonlinear ODE TPBVP:

$$\frac{d}{ds}v_j(s) = F_y(s, y_j(s), \lambda_j)v_j(s) + F_\lambda(s, y_j(s), \lambda_j)\tau_j,$$

$$\frac{d}{ds}\tau_j = 0,$$

$$\frac{d}{ds}w(s) = v_j^T(s)v_j(s),$$

$$G_y(y_j(a), y_j(b), \lambda_j) v_j(a) + G_y(y_j(b), y_j(b), \lambda_j) v_j(b) + G_\lambda(y_j(a), y_j(b), \lambda_j) \tau_j = 0_{n \times 1},$$

$$w(a) = 0,$$

$$w(b) + \tau_j^2 - 1 = 0,$$

which must be solved for $v_j : [a, b] \to \mathbb{R}^n$, $\tau_j \in \mathbb{R}$, and $w : [a, b] \to \mathbb{R}$ and where $(v_j, \tau_j)$ is a unit tangent to $C$ at $(y_j, \lambda_j)$. Note that the first, second, and third equations in (169) are the ODEs, while the fourth, fifth, and sixth equations constitute the boundary conditions. The first, second, and fourth equations in (169) are the linearization (i.e. Fréchet derivative) of (126) about the solution $(y_j, \lambda_j)$ and ensure that a tangent is produced, while the third, fifth, and sixth equations in (169) enforce the arclength constraint (168) ensuring that the tangent is of unit length. The initial solution guess to solve (169) is $(v_j, \tau_j) = (0_{n \times 1}, 1)$ and $w(s) = 0$, $s \in [a, b]$.

Note that the ODE TPBVP (169) can be solved numerically via the MATLAB routines sbvp or bvptwp, which offers 4 algorithms: twpbvp.m, twpbvpc.m, twpbvp.l, and twpbvpc.l. Since $y_j$ is usually only known at a discrete set of points in $[a, b]$, the values of this function at the other points in $[a, b]$ must be obtained through interpolation in order to numerically solve (169). The MATLAB routine interp1 performs linear, cubic, pchip, makima, and spline interpolation and may be utilized to interpolate $y_j$ while solving (169).

Because the numerical solvers usually converge faster when provided Jacobians of the ODE velocity function and of the two-point boundary condition function, these are computed below. Let

$$x = \begin{bmatrix} v_j \\ \tau_j \\ w \end{bmatrix}. \quad (170)$$
The ODE velocity function in (169) is

\[ \mathbf{H}^t (s, \mathbf{x}(s)) = \mathbf{H}^t (s, \mathbf{v}_j(s), \tau_j, w(s)) \]

\[ = \begin{bmatrix} \mathbf{F}_y (s, \mathbf{y}_j(s), \lambda_j) \mathbf{v}_j(s) + \mathbf{F}_\lambda (s, \mathbf{y}_j(s), \lambda_j) \tau_j \\ 0 \\ \mathbf{v}_j^T(s) \mathbf{v}_j(s) \end{bmatrix}. \]  

(171)

The Jacobian of the ODE velocity function \( \mathbf{H}^t \) with respect to \( \mathbf{x} \) is

\[ \mathbf{H}_x^t (s, \mathbf{x}(s)) = \mathbf{H}_x^t (s, \mathbf{v}_j(s), \tau_j, w(s)) \]

\[ = \begin{bmatrix} \mathbf{F}_y (s, \mathbf{y}_j(s), \lambda_j) & \mathbf{F}_\lambda (s, \mathbf{y}_j(s), \lambda_j) & \mathbf{0}_{n \times 1} \\ \mathbf{0}_{1 \times n} & 0 & 0 \\ 2 \mathbf{v}_j^T(s) & 0 & 0 \end{bmatrix}. \]  

(172)

The two-point boundary condition in (169) is

\[ \mathbf{K}^t (\mathbf{x}(a), \mathbf{x}(b)) = \mathbf{0}_{(n+2) \times 1}, \]  

(173)

where \( \mathbf{K}^t \) is the two-point boundary condition function

\[ \mathbf{K}^t (\mathbf{x}(a), \mathbf{x}(b)) = \begin{bmatrix} \mathbf{G}_y (a) (\mathbf{y}_j(a), \mathbf{y}_j(b), \lambda_j) \mathbf{v}_j(a) + \mathbf{G}_y (b) (\mathbf{y}_j(a), \mathbf{y}_j(b), \lambda_j) \mathbf{v}_j(b) + \mathbf{G}_\lambda (\mathbf{y}_j(a), \mathbf{y}_j(b), \lambda_j) \tau_j \\ \mathbf{0}_{1 \times n} \\ \mathbf{0}_{1 \times n} \\ \mathbf{w}(a) \\ \mathbf{w}(b) + \tau_j^2 - 1 \end{bmatrix}. \]  

(174)

The Jacobians of the two-point boundary condition function \( \mathbf{K}^t \) with respect to \( \mathbf{x}(a) \) and \( \mathbf{x}(b) \) are

\[ \mathbf{K}_{x(a)}^t (\mathbf{x}(a), \mathbf{x}(b)) = \begin{bmatrix} \mathbf{G}_y (a) (\mathbf{y}_j(a), \mathbf{y}_j(b), \lambda_j) & \mathbf{G}_\lambda (\mathbf{y}_j(a), \mathbf{y}_j(b), \lambda_j) & \mathbf{0}_{n \times 1} \\ \mathbf{0}_{1 \times n} \\ \mathbf{0}_{1 \times n} \\ 2 \tau_j \\ 1 \end{bmatrix}. \]  

(175)

and

\[ \mathbf{K}_{x(b)}^t (\mathbf{x}(a), \mathbf{x}(b)) = \begin{bmatrix} \mathbf{G}_y (b) (\mathbf{y}_j(a), \mathbf{y}_j(b), \lambda_j) & \mathbf{G}_\lambda (\mathbf{y}_j(a), \mathbf{y}_j(b), \lambda_j) & \mathbf{0}_{n \times 1} \\ \mathbf{0}_{1 \times n} \\ \mathbf{0}_{1 \times n} \\ 2 \tau_j \\ 1 \end{bmatrix}. \]  

(176)

Special care must be taken when implementing the Jacobians (175) and (176). Since the unknown constant \( \tau_j \) appears as the second to last element of both \( \mathbf{x}(a) \) and \( \mathbf{x}(b) \), \( \tau_j \) from only one of \( \mathbf{x}(a) \) and \( \mathbf{x}(b) \) is actually used to construct each term in \( \mathbf{K}^t \) involving \( \tau_j \). The middle column of (175) is actually the derivative of \( \mathbf{K}^t \) with respect to the \( \tau_j \) in \( \mathbf{x}(a) \), while the middle column of (176) is actually the derivative of \( \mathbf{K}^t \) with respect to the \( \tau_j \) in \( \mathbf{x}(b) \). Thus, the middle columns in (175) and (176) corresponding to the derivative of \( \mathbf{K}^t \) with respect to \( \tau_j \) should not coincide in a software implementation. For example, if \( \mathbf{K}^t \) is constructed from the \( \tau_j \) in \( \mathbf{x}(a) \), \( \mathbf{K}_{x(a)}^t \) is as shown in (175) while the middle column of (176) corresponding to the derivative of \( \mathbf{K}^t \) with respect to the \( \tau_j \) in \( \mathbf{x}(b) \) is all zeros. Alternatively, if \( \mathbf{K}^t \) is constructed from the \( \tau_j \) in \( \mathbf{x}(b) \), \( \mathbf{K}_{x(b)}^t \) is as shown in (176) while the middle column of (175) corresponding to the derivative of \( \mathbf{K}^t \) with respect to the \( \tau_j \) appearing in \( \mathbf{x}(a) \) is all zeros.
D.3. Determine the Tangent Direction. The unit tangent \((v_j, \tau_j)\) at \((y_j, \lambda_j)\) obtained by solving (169) must be scaled so that the sweep predictor-corrector continuation method does not reverse direction. As shown in [45], the correct direction for the unit tangent is obtained via:

\[
(v_j, \tau_j) \leftarrow \text{sgn}(\kappa)(v_j, \tau_j),
\]

where \(\kappa\) is the inner product of the previous and current unit tangents:

\[
\kappa = \langle (v_{j-1}, \tau_{j-1}), (v_j, \tau_j) \rangle = \int_a^b v_j^T(s)v_j(s)ds + \tau_{j-1}\tau_j.
\]

The integration operator to construct the inner product \(\kappa\) in (178) can be realized via the MATLAB routine trapez. With the sign direction selected by (177), the inner product of the previous and current unit tangents is positive:

\[
\langle (v_{j-1}, \tau_{j-1}), (v_j, \tau_j) \rangle = \int_a^b v_j^T(s)v_j(s)ds + \tau_{j-1}\tau_j > 0.
\]

D.4. Sweep along the Tangent. By monotonically increasing (or sweeping) the tangent step length \(\sigma\) from 0, the current solution \((y_j, \lambda_j)\) and its unit tangent \((v_j, \tau_j)\) can be used to find the next solution \((y_{j+1}, \lambda_{j+1})\) that solves (126) while satisfying the orthogonality constraint:

\[
\langle (v_j, \tau_j), (y_{j+1}, \lambda_{j+1}) - ((y_j, \lambda_j) + \sigma (v_j, \tau_j)) \rangle = \langle (v_j, \tau_j), (y_{j+1} - (y_j + \sigma v_j), \lambda_{j+1} - (\lambda_j + \sigma \tau_j)) \rangle
\]

\[
= \int_a^b v_j^T(s) [y_{j+1}(s) - (y_j(s) + \sigma v_j(s))] ds + \tau_j [\lambda_{j+1} - (\lambda_j + \sigma \tau_j)] = 0.
\]

This yields the ODE TPBVP:

\[
\begin{align*}
\frac{d}{ds} y_{j+1}(s) &= F(s, y_{j+1}(s), \lambda_{j+1}), \\
\frac{d}{ds} \lambda_{j+1} &= 0, \\
\frac{d}{ds} w(s) &= v_j^T(s) [y_{j+1}(s) - (y_j(s) + \sigma v_j(s))], \\
G(y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) &= 0_{n \times 1}, \\
w(a) &= 0, \\
w(b) + \tau_j [\lambda_{j+1} - (\lambda_j + \sigma \tau_j)] &= 0,
\end{align*}
\]

which must be solved for \(y_{j+1}: [a, b] \to \mathbb{R}^n, \lambda_{j+1} \in \mathbb{R}\), and \(w: [a, b] \to \mathbb{R}\) by monotonically increasing (or sweeping) \(\sigma\). Note that the first, second, and third equations in (181) are the ODEs, while the fourth, fifth, and sixth equations constitute the boundary conditions. The first, second, and fourth equations in (181) ensure that the solution lies on \(C\) (i.e. satisfies (126)), while the third, fifth, and sixth equations in (181) enforce the orthogonality constraint (180). The initial solution guess to solve (181) is the current solution \((y_j, \lambda_j)\) and \(w(s) = 0, s \in [a, b]\). \(\sigma\) starts at 0, since the initial solution guess for \((y_{j+1}, \lambda_{j+1})\) is \((y_j, \lambda_j)\), and increases monotonically until the maximum threshold \(\sigma_{\text{max}}\) is reached or until the ODE TPBVP solver halts (due to reaching a turning point).

Note that the ODE TPBVP (181) can be solved numerically via the MATLAB routine bvptwp, which offers 2 continuation algorithms: \texttt{acdc} and \texttt{acdcc}. The continuation algorithms \texttt{acdc} and \texttt{acdcc} assume that the continuation parameter...
The ODE velocity function in (181) is
\[ y \] at a turning point in the continuation parameter. Since \( y_j \) and \( v_j \) are usually only known at a discrete set of points in \([a, b]\), the values of these functions at the other points in \([a, b]\) must be obtained through interpolation in order to numerically solve (181). The MATLAB routine \texttt{interp1} performs linear, cubic, pchip, makima, and spline interpolation and may be utilized to interpolate \( y_j \) and \( v_j \) while solving (181).

Because the numerical solvers usually converge faster when provided Jacobians of the ODE velocity function and of the two-point boundary condition function, these are computed below. Let
\[
\mathbf{x} = \begin{bmatrix} y_{j+1} \\ \lambda_{j+1} \\ w \end{bmatrix}.
\] (182)

The ODE velocity function in (181) is
\[
\mathbf{H}^q(s, \mathbf{x}(s), \sigma) = \mathbf{H}^q(s, y_{j+1}(s), \lambda_{j+1}, w(s), \sigma)
\]
\[= \begin{bmatrix} F(s, y_{j+1}(s), \lambda_{j+1}) \\ 0 \\ v_j^T(s)[y_{j+1}(s) - (y_j(s) + \sigma v_j(s))] \end{bmatrix}. \] (183)

The Jacobian of the ODE velocity function \( \mathbf{H}^q \) with respect to \( \mathbf{x} \) is
\[
\mathbf{H}_x^q(s, \mathbf{x}(s), \sigma) = \mathbf{H}_x^q(s, y_{j+1}(s), \lambda_{j+1}, w(s), \sigma)
\]
\[= \begin{bmatrix} F_y(s, y_{j+1}(s), \lambda_{j+1}) & F_{\lambda}(s, y_{j+1}(s), \lambda_{j+1}) & 0_{n \times 1} \\ 0_{1 \times n} & 0 & 0 \\ v_j^T(s) & 0 & 0 \end{bmatrix}. \] (184)

The two-point boundary condition in (181) is
\[
\mathbf{K}^q(\mathbf{x}(a), \mathbf{x}(b), \sigma) = 0_{(n+2) \times 1}, \] (185)
where \( \mathbf{K}^q \) is the two-point boundary condition function
\[
\mathbf{K}^q(\mathbf{x}(a), \mathbf{x}(b), \sigma) = \begin{bmatrix} G(y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) \\ w(a) \\ w(b) + \tau_j[\lambda_{j+1} - (\lambda_j + \sigma \tau_j)] \end{bmatrix}. \] (186)

The Jacobians of the two-point boundary condition function \( \mathbf{K}^q \) with respect to \( \mathbf{x}(a) \) and \( \mathbf{x}(b) \) are
\[
\mathbf{K}^q_{\mathbf{x}(a)}(\mathbf{x}(a), \mathbf{x}(b), \sigma)
\]
\[= \begin{bmatrix} G_{y}(y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) & G_{\lambda}(y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) & 0_{n \times 1} \\ 0_{1 \times n} & 0 & 1 \\ 0_{1 \times n} & \tau_j & 0 \end{bmatrix}. \] (187)

and
\[
\mathbf{K}^q_{\mathbf{x}(b)}(\mathbf{x}(a), \mathbf{x}(b), \sigma)
\]
\[= \begin{bmatrix} G_{y}(y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) & G_{\lambda}(y_{j+1}(a), y_{j+1}(b), \lambda_{j+1}) & 0_{n \times 1} \\ 0_{1 \times n} & 0 & 0 \\ 0_{1 \times n} & \tau_j & 1 \end{bmatrix}. \] (188)

Special care must be taken when implementing the Jacobians (187) and (188). Since the unknown constant \( \lambda_{j+1} \) appears as the second to last element of both \( \mathbf{x}(a) \) and \( \mathbf{x}(b) \), \( \lambda_{j+1} \) from only one of \( \mathbf{x}(a) \) and \( \mathbf{x}(b) \) is actually used to construct each
Algorithm 2 Sweep Predictor-Corrector Continuation for Nonlinear ODE TPBVPs.

Input: ODE velocity function $F: [a, b] \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, two-point boundary condition function $G: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, and their Jacobians $F_y: [a, b] \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times n}$, $F_\lambda: [a, b] \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times 1}$, $G_{y(a)}: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times n}$, $G_{y(b)}: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times n}$, and $G_\lambda: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n \times 1}$. Initial point on the solution curve $C$, $(y_1, \lambda_1)$.

Maximum number of points not including the initial point to be computed on $C$, $J$. $\sigma_{\text{max}}$ is a vector of length $J$ such that $\sigma_{\text{max}}(j)$ is the maximum tangent steplength permitted to obtain solution $j + 1$. Tangent direction at the first solution, $d$. $d$ may be $-2$, $-1$, 1, or 2. If $d$ is $-1$ or 1, the first tangent is scaled by $d$. If $d$ is $-2$ (2), the first tangent is scaled so that $\lambda$ decreases (increases) in the first step.

Output: A solution curve $c$.

1: function PACs3_BVP($F, G, F_y, F_\lambda, G_{y(a)}, G_{y(b)}, G_\lambda, (y_1, \lambda_1), J, \sigma_{\text{max}}, d$)
2:     $c(1) \leftarrow (y_1, \lambda_1)$ \Comment{Store the initial solution on $C$.}
3:     for $j = 1$ to $J$ do
4:         \Comment{Trace the solution curve $C$.}
5:             if $j = 1$ then
6:                 \Comment{Choose the direction of the tangent at the initial solution, based on $d$.}
7:                     if ($d == -2$ OR $d == 2$) AND $\tau_1 < 0$ then
8:                         $d \leftarrow -d$ \Comment{Flip the sign of $d$ to get the desired tangent direction.}
9:                     \end if
10:                 \else
11:                     \Comment{Ensure that the traced solution does not reverse direction.}
12:                     $\kappa \leftarrow \langle (\nu_{j-1}, \tau_{j-1}), (\nu_j, \tau_j) \rangle$
13:                 \end if
14:             \Comment{Choose the correct tangent direction.}
15:             $(\nu_j, \tau_j) \leftarrow \text{sgn}(\kappa) (\nu_j, \tau_j)$
16:             \Comment{Obtain the next solution $(y_{j+1}, \lambda_{j+1})$ on $C$ by solving (181) starting from $(y_j, \lambda_j)$ and monotonically increasing $\sigma$ starting from 0 and without exceeding $\sigma_{\text{max}}(j)$.}
17:             $c(j + 1) \leftarrow (y_{j+1}, \lambda_{j+1})$ \Comment{Store the new solution on $C$.}
18:         \end for
19:     return $c$
20: \end function

term in $K^q$ involving $\lambda_{j+1}$. The middle column of (187) is actually the derivative of $K^q$ with respect to $\lambda_{j+1}$ in $x(a)$, while the middle column of (188) is actually the derivative of $K^q$ with respect to $\lambda_{j+1}$ in $x(b)$. Thus, the middle columns in (187) and (188) corresponding to the derivative of $K^q$ with respect to $\lambda_{j+1}$ should not coincide in a software implementation. For example, if $K^q$ is constructed from the $\lambda_{j+1}$ in $x(a)$, $K^q_{x(a)}$ is as shown in (187) while the middle column of (188) corresponding to the derivative of $K^q$ with respect to the $\lambda_{j+1}$ in $x(b)$ is all zeros. Alternatively, if $K^q$ is constructed from the $\lambda_{j+1}$ in $x(b)$, $K^q_{x(b)}$ is as shown in (188) while the middle column of (187) corresponding to the derivative of $K^q$ with respect to the $\lambda_{j+1}$ appearing in $x(a)$ is all zeros.
D.5. Pseudocode for Sweep Predictor-Corrector Continuation. Above is pseudocode that realizes the sweep predictor-corrector continuation method.

REFERENCES

[1] Community portal for automatic differentiation, 2016, Available at: http://www.autodiff.org/.
[2] E. Allgower and K. Georg, Continuation and path following, Acta Numerica, 2 (1993), 1–64.
[3] U. Ascher, J. Christiansen and R. Russell, Algorithm 569: Colsys: Collocation software for boundary-value odes [d2], ACM Transactions on Mathematical Software (TOMS), 7 (1981), 223–229.
[4] U. Ascher, R. Mattheij and R. Russell, Numerical Solution of Boundary Value Problems for Ordinary Differential Equations, Vol. 13, SIAM, 1994.
[5] U. Ascher and R. Spiteri, Collocation software for boundary value differential-algebraic equations, SIAM Journal on Scientific Computing, 15 (1994), 938–952.
[6] W. Auzinger, G. Kneisl, O. Koch and E. Weinmüller, A collocation code for singular boundary value problems in ordinary differential equations, Numerical Algorithms, 33 (2003), 27–39.
[7] G. Bader and U. Ascher, A new basis implementation for a mixed order boundary value ode solver, SIAM Journal on Scientific and Statistical Computing, 8 (1987), 483–500.
[8] G. Bader and P. Kunkel, Continuation and collocation for parameter-dependent boundary value problems, SIAM Journal on Scientific and Statistical Computing, 10 (1989), 72–88.
[9] D. Baraff, Physically based modeling: Rigid body simulation, SIGGRAPH Course Notes, ACM SIGGRAPH, 2 (2001), 1–2.
[10] Z. Bashir-Ali, J. Cash and H. Silva, Lobatto deferred correction for stiff two-point boundary value problems, Computers & Mathematics with Applications, 36 (1998), 59–69.
[11] J. Betts, Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, Vol. 19, SIAM, 2010.
[12] Á. Birkisson, Numerical Solution of Nonlinear Boundary Value Problems for Ordinary Differential Equations in the Continuous Framework, PhD thesis, University of Oxford, 2013.
[13] J. Boisvert, A Problem-Solving Environment for the Numerical Solution of Boundary Value Problems, PhD thesis, University of Saskatchewan, 2011.
[14] J. Boisvert, P. Muir and R. Spiteri, A runge-kutta bvode solver with global error and defect control, ACM Transactions on Mathematical Software (TOMS), 39 (2013), 11.
[15] J. Boyd, Solving Transcendental Equations: The Chebyshev Polynomial Proxy and Other Numerical Rootfinders, Perturbation Series, and Oracles, Vol. 139, SIAM, 2014.
[16] A. Bryson, Dynamic Optimization, Vol. 1, Prentice Hall, 1999.
[17] A. Bryson and Y.-C. Ho, Applied Optimal Control: Optimization, Estimation and Control, CRC Press, 1975.
[18] J.-B. Caillau, O. Cots and J. Gergaud, Differential continuation for regular optimal control problems, Optimization Methods and Software, 27 (2012), 177–196.
[19] J. Cash, D. Hollevoet, F. Mazzia and A. Nagy, Algorithm 927: The matlab code bvptwp.m for the numerical solution of two point boundary value problems, ACM Transactions on Mathematical Software (TOMS), 39 (2013), 15.
[20] J. Cash and F. Mazzia, A new mesh selection algorithm, based on conditioning, for two-point boundary value codes, Journal of Computational and Applied Mathematics, 184 (2005), 362–381.
[21] J. Cash and F. Mazzia, Hybrid mesh selection algorithms based on conditioning for two-point boundary value problems, JNAIAM J. Numer. Anal. Indust. Appl. Math, 1 (2006), 81–90.
[22] J. Cash, G. Moore and R. Wright, An automatic continuation strategy for the solution of singularly perturbed nonlinear boundary value problems, ACM Transactions on Mathematical Software (TOMS), 27 (2001), 245–266.
[23] J. Cash and M. Wright, A deferred correction method for nonlinear two-point boundary value problems: implementation and numerical evaluation, SIAM Journal on Scientific and Statistical Computing, 12 (1991), 971–989.
[24] F. Chernousko and A. Lyubushin, Method of successive approximations for solution of optimal control problems, Optimal Control Applications and Methods, 3 (1982), 101–114.
[25] G. Corliss, C. Faure, A. Griewank, L. Hascoet and U. Naumann, Automatic Differentiation Of Algorithms: From Simulation To Optimization, Vol. 1, Springer Science & Business Media, 2002.
[26] H. Dankowicz and F. Schilder, Recipes for Continuation, SIAM, 2013.
[27] D. Davidenko, On a new method of numerical solution of systems of nonlinear equations, in Dokl. Akad. Nauk SSSR, 88 (1953), 601–602.
[28] D. Davidenko, The approximate solution of sets of nonlinear equations, Ukr. Mat. Zh, 5 (1953), 196–206.
[29] P. Deuflhard, Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms, Vol. 35, Springer Science & Business Media, 2011.
[30] E. Doedel, T. Fairgrieve, B. Sandstede, A. Champneys, Y. Kuznetsov and X. Wang, Auto-07p: Continuation and bifurcation software for ordinary differential equations.
[31] J. Fike and J. Alonso, The development of hyper-dual numbers for exact second-derivative calculations, AIAA paper, 886 (2011), 124.
[32] J. Fike and J. Alonso, Automatic differentiation through the use of hyper-dual numbers for second derivatives, in Recent Advances in Algorithmic Differentiation, Springer, (2012), 163–173.
[33] J. Fike, S. Jongsma, J. Alonso and E. Van Der Weide, Optimization with gradient and hessian information calculated using hyper-dual numbers, AIAA paper, 3807 (2011), 2011.
[34] J. Frisvad, Building an orthonormal basis from a 3d unit vector without normalization, Journal of Graphics Tools, 16 (2012), 151–159.
[35] P. Gill, W. Murray and M. Saunders, SNOPT: An SQP algorithm for large-scale constrained optimization, SIAM Rev., 47 (2005), 99–131.
[36] P. Gill, W. Murray, M. Saunders and E. Wong, User’s Guide for SNOPT 7.6: Software for Large-Scale Nonlinear Programming, Center for Computational Mathematics Report CCoM 17-1, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2017.
[37] B. Graf, Quaternions and dynamics, arXiv preprint arXiv:0811.2889.
[38] R. Gupta, A. Bloch and I. Kolmanovskiy, Combined homotopy and neighboring extremal optimal control, Optimal Control Applications and Methods, 38 (2017), 459–469.
[39] Y. Hardy, K. Tan and W.-H. Steeb, Computer Algebra with SymbolicC++, World Scientific Publishing Company, 2008.
[40] D. Holm, Geometric Mechanics: Rotating, Translating, and Rolling, 2nd edition, Geometric Mechanics, Imperial College Press, 2011.
[41] D. Hull, Optimal Control Theory for Applications, Springer Science & Business Media, 2013.
[42] V. Kungurtsev and J. Jäschke, A predictor-corrector path-following algorithm for dual-degenerate parametric optimization problems, SIAM Journal on Optimization, 27 (2017), 538–564.
[43] G. Lantoine, R. Russell and T. Dargent, Using multicomplex variables for automatic computation of high-order derivatives, ACM Transactions on Mathematical Software (TOMS), 38 (2012), 16.
[44] Y. LeCun, Y. Bengio and G. Hinton, Deep learning, Nature, 521 (2015), 436–444.
[45] A. Lyubushin, Modifications of the method of successive approximations for solving optimal control problems, USSR Computational Mathematics and Mathematical Physics, 22 (1982), 29–34.
[46] J. Martins, P. Sturdza and J. Alonso, The connection between the complex-step derivative approximation and algorithmic differentiation, AIAA Paper, 921 (2001), 2001.
[53] J. Martins, P. Sturdza and J. Alonso, The complex-step derivative approximation, ACM Transactions on Mathematical Software (TOMS), 29 (2003), 245–262.
[54] P. Muir, Optimal discrete and continuous mono-implicit runge–kutta schemes for bvodes, Advances in Computational Mathematics, 10 (1999), 135–167.
[55] U. Naumann, The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation, Vol. 24, SIAM, 2012.
[56] M. Neuenhofen, Review of theory and implementation of hyper-dual numbers for first and second order automatic differentiation, arXiv preprint arXiv:1801.03614.
[57] M. Patterson and A. Rao, Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming, ACM Transactions on Mathematical Software (TOMS), 41 (2014), 1.
[58] V. Putkaradze and S. Rogers, On the optimal control of a rolling ball robot actuated by internal point masses, Journal of Dynamic Systems, Measurement, and Control, 142 (2020), 051002, 22 pages.
[59] V. Putkaradze and S. Rogers, Constraint control of nonholonomic mechanical systems, Journal of Nonlinear Science, 28 (2018), 193–234.
[60] V. Putkaradze and S. Rogers, On the dynamics of a rolling ball actuated by internal point masses, Meccanica, 53 (2018), 3839–3868.
[61] V. Putkaradze and S. Rogers, On the normal force and static friction acting on a rolling ball actuated by internal point masses, Regular and Chaotic Dynamics, 24 (2019), 145–170.
[62] L. Rall, Davidenko’s Method for the Solution of Nonlinear Operator Equations, Technical report, University of Wisconsin, Madison, Mathematics Research Center, 1968.
[63] G. Rozenblat, On the choice of physically realizable parameters when studying the dynamics of spherical and ellipsoidal rigid bodies, Mechanics of Solids, 51 (2016), 415–423.
[64] L. Shampine, J. Kierzenka and M. Reichelt, Solving boundary value problems for ordinary differential equations in matlab with bvp4c, Tutorial notes, 437–448.
[65] W. Squire and G. Trapp, Using complex variables to estimate derivatives of real functions, Siam Review, 40 (1998), 110–112.
[66] B. Stevens, F. Lewis and E. Johnson, Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems, John Wiley & Sons, 2015.
[67] A. Wächter and L. Biegler, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, Mathematical Programming, 106 (2006), 25–57.
[68] E. Weinmüller and R. Winkler, Pathfollowing algorithm for singular boundary value problems, ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik, 68 (1988), 527–537.
[69] M. Weinstein, M. Patterson and A. Rao, Utilizing the algorithmic differentiation package adigator for solving optimal control problems using direct collocation, in AIAA Guidance, Navigation, and Control Conference, (2015), 1085.
[70] M. Weinstein and A. Rao, Algorithm 984: Adigator, a toolbox for the algorithmic differentiation of mathematical functions in matlab using source transformation via operator overloading, ACM Transactions on Mathematical Software (TOMS), 44 (2017), 21.
[71] W. Zangwill and C. Garcia, Pathways to Solutions, Fixed Points, and Equilibria, Prentice Hall, 1981.

Received June 2019; 1st revision January 2020; Final revision January 2020.

E-mail address: putkarad@ualberta.ca
E-mail address: srogers@umn.edu