mathematics

MDPI

Article

A Theory of Functional Connections-Based *hp*-Adaptive Mesh Refinement Algorithm for Solving Hypersensitive Two-Point Boundary-Value Problems

Kristofer Drozd, Roberto Furfaro * and Andrea D'Ambrosio

System & Industrial Engineering, University of Arizona, Tucson, AZ 85721, USA; kdrozd@arizona.edu (K.D.); dambrosio@arizona.edu (A.D.)
* Correspondence: robertof@arizona.edu; Tel.: +1-520-621-2525

Abstract: This manuscript introduces the first *hp*-adaptive mesh refinement algorithm for the Theory of Functional Connections (TFC) to solve hypersensitive two-point boundary-value problems (TPBVPs). The TFC is a mathematical framework that analytically satisfies linear constraints using an approximation method called a constrained expression. The constrained expression utilized in this work is composed of two parts. The first part consists of Chebyshev orthogonal polynomials, which conform to the solution of differentiation variables. The second part is a summation of products between switching and projection functionals, which satisfy the boundary constraints. The mesh refinement algorithm relies on the truncation error of the constrained expressions to determine the ideal number of basis functions within a segment's polynomials. Whether to increase the number of basis functions in a segment or divide it is determined by the decay rate of the truncation error. The results show that the proposed algorithm is capable of solving hypersensitive TPBVPs more accurately than MATLAB R2021b's `bvp4c` routine and is much better than the standard TFC method that uses global constrained expressions. The proposed algorithm's main flaw is its long runtime due to the numerical approximation of the Jacobians.

Keywords: mesh refinement; hypersensitive boundary-value problems; functional interpolation; optimal control; theory of functional connections

MSC: 49M05

Citation: Drozd, K.; Furfaro, R.; D'Ambrosio, A. A Theory of Functional Connections-Based *hp*-Adaptive Mesh Refinement Algorithm for Solving Hypersensitive Two-Point Boundary-Value Problems. *Mathematics* **2024**, *12*, 1360. https://doi.org/10.3390/math12091360

Academic Editor: Alicia Cordero

Received: 15 March 2024
Revised: 20 April 2024
Accepted: 22 April 2024
Published: 29 April 2024

1. Introduction

In recent years, the Theory of Functional Connections (TFC) [1] has garnered much interest from researchers for obtaining numerical solutions to scientific and engineering problems. A few examples of the types of problems the TFC has been used to solve include those from transport theory [2], solid mechanics [3,4], and orbit transfer [5–7]. The increase in the TFC's popularity is due to its successful use of functional interpolation to analytically satisfy linear constraints exactly. Functional interpolation is performed by approximating a solution via a constrained expression, which comprises an arbitrary free function and a summation of products between switching functions and projection functionals. Some constraints that could be handled were immediately apparent during the TFC's inception by Daniele Mortari [8]. Examples include point, derivative, and relative constraints. The ability of the TFC to analytically satisfy these constraints exactly allowed it to be used to solve one-dimensional nonlinear ordinary differential equations (ODEs) in milliseconds and with machine-level accuracy [9]. In this work, we are concerned with solving linear and nonlinear hypersensitive two-point boundary-value problems (TPBVPs) stemming from optimal control problems (OCPs) involving point and derivative constraints. These problems are challenging because of steep gradients in their solutions, which are difficult to approximate with global constrained expressions. We overcome this

challenge by developing the first hp-adaptive mesh refinement algorithm for the TFC. The algorithm splits the constrained expressions into multiple segments and adaptively determines the length, location, and hyperparameters for each segment to accurately approximate a solution.

Three families of optimal control methods exist: dynamic programming (DP) [10], direct [11,12], and indirect methods [13,14]. DP solves an OCP by transforming it into the Hamilton–Jacobi–Bellman (HJB) partial differential equation, which gives the necessary and sufficient conditions for an optimal solution when solved and enables closed-loop control effort. Direct methods convert the OCP into a nonlinear programming (NLP) problem that can be solved by any of the available NLP solvers in the literature but are inherently open-loop. Indirect methods are also open-loop, but they analytically construct the necessary conditions for optimality via Pontryagin's Minimum Principle (PMP) and solve a resulting dual TPBVP to obtain the OCP solution. Indirect methods tend to solve OCPs more accurately than direct methods but are less likely to converge because they are more sensitive to an initial guess. Whether one class of methods should be used over another is problem-dependent. Hence, the TFC has already been used to solve OCPs via all three families, but some more than others. Schiassi et al. used a subset of the TFC, called the Extreme Theory of Functional Connections (X-TFC), to solve the HJB for finite and infinite horizon OCPs with linear and nonlinear dynamics [15]. For the direct method, Zhang et al. solved several coplanar orbit low-thrust multi-target visit problems, governed by aspherical gravity perturbation, with the TFC to analytically satisfy the orbital multi-target visit constraints in an NLP [16]. The TFC has been used to solve OCPs via the indirect method much more extensively in the literature. For example, by forming a TPBVP via PMP, the TFC has been used to solve many spacecraft energy optimal rendezvous problems in order to avoid keep-out-zones [17], planetary landing problems [18], and the famous Riccati problem of optimal control [19].

Even though the TFC has been shown to solve many OCPs very well via the indirect method, the method fails to converge on solutions that are not smooth (e.g., bang-bang problems). This occurs because the TFC initially utilizes global orthogonal polynomials for the free function, which only converge exponentially on smooth continuous functions [20] (p. 29). This issue was rectified by Johnston et al. by splitting the domain of a fuel-optimal planetary landing OCP into several intervals connected by knot points [18]. The knot points were placed where the discontinuity in the solution occurred, making each interval smooth. The total solution could then be converged upon by using constrained expressions for each segment and tying them together. The locations of the knot points were found via an optimization algorithm, which was feasible because the number of switching points in a fuel-optimal planetary landing OCP is known when the gravity is assumed to be constant [21]. A technique where the domains of TPBVPs could be split into segments and solved with the TFC was eventually formalized by Johnston and Mortari [22] for solving hybrid problems. Nonetheless, Johnston and Mortari provided no insight into how to determine the number of orthogonal polynomial basis terms in an interval and how to determine the number of intervals when they are unknown a priori. Hence, our desire to develop a TFC-based adaptive mesh refinement algorithm for determining these hyperparameters.

Researchers have heavily investigated the use of adaptive mesh refinement algorithms in conjunction with many OCP techniques. One of the more well-known applications in optimal control theory is the coupling with pseudospectral methods for solving OCPs directly [23–27], where the state and control are approximated using Lagrange orthogonal polynomials with Gaussian quadrature collocation points. Most adaptive mesh refinement methods for pseudospectral methods are referred to as *hp* methods because they achieve better convergence toward the solution by varying the number of intervals on the domain (the *h* method) and the degree of the orthogonal polynomial (the *p* method). A more in-depth explanation of the *h*, *p*, and *hp* methods can be found by studying finite-element methods [28–30], where they are also widely used. For the mesh refinement algorithm

to be adaptive, an a posteriori estimate of the error between the actual and approximate solutions must be quantified to determine the ideal number of intervals, the degree of the approximating polynomials, and the location of the knot points for each interval. For example, Darby et al. approximated the error by computing the residuals on and between collocation points [24]. When the error was above some tolerance, they increased the polynomial degree in intervals with low curvature or uniform error across the collocation points. An interval was divided into subintervals instead when the curvature was high or the error across the collocation points in an interval was not uniform. Unfortunately, the error estimates from Darby et al. [24] created much noise, making them computationally intractable when a highly accurate solution is desired. Patterson et al. developed a more accurate error estimate by using the difference between an interpolated value of the state and a quadrature approximation to the integral of the state dynamics [25]. They performed adaptive mesh refinement based on an error estimate ratio: when a set number of polynomial degrees was used to approximate the state, the degree number was increased by one for interpolation. The spectral convergence properties of pseudospectral methods could then be used to determine how many degrees to increase the polynomial by or to divide the interval into smaller subintervals.

The mesh refinement methods proposed in [24,25] only increased the mesh, which sometimes increased the computational complexity. Liu et al. [26] introduced a technique that both increased and decreased the mesh based on the error estimate from Patterson et al. [25] and used a ratio between the second derivative of the approximation on the current mesh and the previous one to determine the smoothness of an interval. Their algorithm used the power-series representation of the Lagrange polynomial to reduce the degree of the approximating polynomial within a mesh interval for smooth intervals when the estimated error was lower than the tolerance. Smooth neighboring mesh intervals were also combined when the estimated error of the larger interval, which extended into the smaller one, was similar. More recently, Liu et al. [27] discovered a new way to estimate the smoothness of mesh intervals according to the decay rate of a Legendre polynomial approximation and increased the mesh size based on the decay rate. The work performed in this manuscript is heavily inspired by [27].

This paper proposes an *hp*-adaptive mesh refinement algorithm that the TFC, with a truncated Chebyshev orthogonal polynomial-free function, can use to solve hypersensitive TPBVPs. The error between the actual solution and the constrained expression approximation is estimated using an exponential least-squares fit of the Chebyshev coefficients, the orthogonal property of Chebyshev polynomials, and a rule of thumb that correlates the error associated with Chebyshev polynomials when approximating functions with their truncation error. The decay rate of the exponential least-squares fit of the Chebyshev coefficients can be used to determine the smoothness of the solution on an interval. When the error estimate is greater than some tolerance and the interval is deemed smooth, the supergeometric convergence of the truncation error for Chebyshev polynomials can be used to relate the error to the ideal number of basis functions in the Chebyshev polynomial needed to satisfy the tolerance. If the interval is not smooth, it is divided into subintervals so that the solution in each subinterval is easier to approximate with a constrained expression. Smooth intervals can have the number of basis functions in the Chebyshev polynomial decreased when they already satisfy the error estimate. Furthermore, smooth intervals can also be combined if they are exceptionally smooth or if the number of basis functions in all their constrained expression approximations is some minimum number.

As the name of this manuscript suggests, we only demonstrate how the proposed *hp*-adaptive mesh refinement algorithm can be used to solve hypersensitive TPBVPs, even though it can also be used for more well-behaved problems. Hypersensitive problems have a very long time domain with respect to the rates of expansion and contraction of the dynamics in specific directions within a neighborhood of the optimal solution [31]. Thus, the TPBVPs solved in this work have sharp increases in the gradients near the boundary conditions and a smooth segment between them. These sharp increases concerning the

entire domain make hypersensitive TPBVPs challenging to solve with global constrained expression approximations. Although hypersensitive TPBVPs can stem from the indirect method of optimal control, the ones we solve in this work do not contain inequality constraints on the control or states. We do not solve these types of problems in this work because they require smoothing techniques along with a continuation procedure, drastically decreasing computational efficiency. The computation time of our *hp*-adaptive mesh refinement algorithm is already long (see Section 6 for more details). Therefore, we hope to speed up the computational runtime before coupling the proposed *hp*-adaptive mesh refinement algorithm with continuation.

This manuscript is outlined as follows. In the next section, we introduce how to solve a general TPBVP with the TFC, as well as how to split the domain into intervals and solve the solution over each with the TFC. The derivation of an error estimate for constrained expressions with orthogonal polynomial-free function approximations is highlighted in Section 3, along with an error analysis for smooth and nonsmooth functions. The TFC-based *hp*-adaptive mesh refinement algorithm that uses our derived error estimate is then proposed in Section 4. Numerical results for solving three hypersensitive TPBVPs are then provided and discussed in Section 5, along with a computational time study. Lastly, conclusions and an outlook on future work are presented in Section 7.

2. Theory of Functional Connections

Here, the TFC is first presented for solving a TPBVP with a single ODE and with a global constrained expression. We then show how global constrained expressions can be used to solve a TPBVP composed of a system of ODEs. Lastly, we demonstrate how global constrained expressions can be replaced with piecewise constrained expressions, where each only approximates a domain segment.

2.1. General TPBVP Outline

Solving a TPBVP via the TFC can be accomplished by following a simple set of procedures. Indeed, a "cookbook" guide can be summarized in three steps:

1. Define the loss equation(s), which are represented by the residuals of the differential equations (DEs).
2. Derive the constrained expression(s).
3. Map the problem domain to that of the free function (if necessary) or simply normalize it for numerical stability.

Assume a general TPBVP,

$$\ddot{y}(t) = f(t, y(t), \dot{y}(t)) \qquad \text{subject to:} \begin{cases} y(t_0) = y_0 \\ y(t_f) = y_f \end{cases}, \tag{1}$$

where f is some function in terms of time t and the differentiation variable y. As its name suggests, the TPBVP is constrained at the beginning and end of the time domain $t \in [t_0, t_f]$. With the TFC, we approximate the solution with a constrained expression Y,

$$y(t) \approx Y(t, g(t)) = g(t) + \sum_{l=1}^{N_{\text{const}}} \Omega_l(t)\, \varrho_l(t_l, g(t_l)), \tag{2}$$

where $g(t)$ is a free function, $\Omega_l(t)$ are switching functions, $\varrho_l(t_l, g(t_l))$ are projection functionals, and N_{const} is the number of linear constraints.

Defining the loss equation for any set of DEs, including the ODE in Equation (1), is easy. Simply bring all terms to one side of the equals sign, like so,

$$\ddot{y}(t) - f(t, y(t), \dot{y}(t)) = \hat{f}(t, y(t), \dot{y}(t), \ddot{y}(t)) = 0, \tag{3}$$

where $\hat{f}$ is the constrained DE residual. To turn the TPBVP into an unconstrained problem (i.e., $\hat{f}$ is no longer subject to y_0 and y_f), the constrained expressions for y and $\dot{y}$ must be derived.

We start the process of figuring out the constrained expressions by rewriting Equation (2) as

$$Y(t, g(t)) = g(t) + \Omega_1(t)\,\varrho_1(t_0, g(t_0)) + \Omega_2(t)\,\varrho_2(t_f, g(t_f)).$$

The projection functionals (i.e., ϱ_1 and ϱ_2) are simply the difference between the constraint value and the free function evaluated at the point where the constraint is defined:

$$\begin{aligned}\varrho_1 &= y_0 - g(t_0)\\ \varrho_2 &= y_f - g(t_f).\end{aligned}$$

The most cumbersome part of deriving the constrained expressions is the derivation of the switching functions, which are of the general forms

$$\Omega_1(t) = \sum_{i=1}^{N_{\text{const}}=2} s_i(t)\,\alpha_{i1} \quad \text{and} \quad \Omega_2(t) = \sum_{i=1}^{N_{\text{const}}=2} s_i(t)\,\alpha_{i2}.$$

Only two switching functions exist because the ODE has two boundary constraints. Combining the equations for the switching functions into a compact linear combination where the constraints occur yields

$$\begin{bmatrix} s_1(t_0) & s_2(t_0) \\ s_1(t_f) & s_2(t_f) \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} = \begin{bmatrix} \Omega_1(t_0) & \Omega_2(t_0) \\ \Omega_1(t_f) & \Omega_2(t_f) \end{bmatrix}.$$

To figure out the switching functions, compute the unknown coefficients α_{ij}. The support functions s_i do not have to be derived but can merely be chosen. However, to ensure the support matrix is invertible, the columns of the support matrix must be linearly independent. For point constraints, like in our case, the selection of $s_1(t) = 1$ and $s_2(t) = t$ satisfies this condition. A switching function is defined as being equal to 1 when evaluated at the constraint it is referencing and equal to 0 at all other constraints. Therefore, the system can be solved as

$$\begin{aligned}\begin{bmatrix} 1 & t_0 \\ 1 & t_f \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\\ \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} &= \begin{bmatrix} 1 & t_0 \\ 1 & t_f \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\\ &= \frac{1}{t_f - t_0} \begin{bmatrix} t_f & -t_0 \\ -1 & 1 \end{bmatrix}.\end{aligned}$$

Substituting the coefficients back into the switching functions and simplifying them gives

$$\Omega_1(t) = \frac{t_f\, s_1(t) - s_2(t)}{t_f - t_0} = \frac{t_f - t}{t_f - t_0}$$

and

$$\Omega_2(t) = \frac{s_2(t) - t_0\, s_1(t)}{t_f - t_0} = \frac{t - t_0}{t_f - t_0}.$$

With the switching and projection functions defined, all that is left to do to finish deriving the constrained expression is to decide what the free function should be. One

commonly selected free function among researchers who utilize the TFC method for solving ODEs is a Chebyshev polynomial expansion of the first kind,

$$g(t) = \sum_{j=N_{\text{const}}}^{N_{\text{const}}+L-1} \xi_j \mathcal{T}_j(\tau) = \boldsymbol{\xi}^\intercal \boldsymbol{\mathcal{T}}(\tau),$$

where ξ_j are the coefficients, $\mathcal{T}_j(\tau)$ are the Chebyshev basis functions, and L is the number of chosen basis functions. When one expresses a Chebyshev series, one often starts the summation from the zeroth order (i.e., $j = 0$). The TFC process requires the series summation to begin at the order equal to the number of constraints (i.e., $j = N_{\text{const}}$) because the Chebyshev series terms of order less than N_{const} are linearly dependent on the switching functions. The algebraic system of equations formed via the loss functions must be linearly independent to ensure that the Jacobian products are invertible in the Gauss–Newton algorithm and linear least squares give a unique solution. The loss function eventually derived is guaranteed to be linearly independent when the Chebyshev series begins at $j = N_{\text{const}}$.

The domain of Chebyshev polynomials $\tau \in [-1, 1]$ does not coincide with the time domain $t \in [t_0, t_f]$ unless $t_0 = -1$ and $t_f = 1$. Thus, to ensure all terms within the constrained expressions are on the same domain, the switching functions can be linearly mapped to the τ domain using the equations,

$$\tau = \tau_0 + \frac{\tau_f - \tau_0}{t_f - t_0}(t - t_0) \quad \longleftrightarrow \quad t = t_0 + \frac{t_f - t_0}{\tau_f - \tau_0}(\tau - \tau_0).$$

The w-th derivative of the free function can then be defined as

$$\frac{\mathrm{d}^w g}{\mathrm{d}t^w} = \boldsymbol{\xi}^\intercal \frac{\mathrm{d}^w \boldsymbol{\mathcal{T}}(\tau)}{\mathrm{d}\tau^w} c^w, \tag{4}$$

where the mapping constant c is

$$c = \left(\frac{2}{t_f - t_0}\right). \tag{5}$$

With the free function and subsequent mapping defined, the constrained expressions for the y, $\dot{y}$, and $\ddot{y}(t)$ variables that make up the loss equation can be defined as

$$\begin{aligned} Y(t) = Y(\tau) &= \boldsymbol{\xi}^\intercal \boldsymbol{\mathcal{T}}(\tau) + \Omega_1(\tau)\Big(y_0 - \boldsymbol{\xi}^\intercal \boldsymbol{\mathcal{T}}(\tau_0)\Big) + \Omega_2(\tau)\Big(y_f - \boldsymbol{\xi}^\intercal \boldsymbol{\mathcal{T}}(\tau_f)\Big) \\ &= \Big(\boldsymbol{\mathcal{T}}(\tau) - \Omega_1(\tau)\,\boldsymbol{\mathcal{T}}(\tau_0) - \Omega_2(\tau)\,\boldsymbol{\mathcal{T}}(\tau_f)\Big)^\intercal \boldsymbol{\xi} + \Omega_1(\tau)\,y_0 + \Omega_2(\tau)\,y_f, \end{aligned}$$

$$\begin{aligned} \frac{\mathrm{d}Y(t)}{\mathrm{d}t} = c\,\frac{\mathrm{d}Y(\tau)}{\mathrm{d}\tau} = \left(\frac{2}{t_f - t_0}\right)\Big[\Big(&\frac{\mathrm{d}\boldsymbol{\mathcal{T}}(\tau)}{\mathrm{d}\tau} - \frac{\mathrm{d}\Omega_1(\tau)}{\mathrm{d}\tau}\,\boldsymbol{\mathcal{T}}(\tau_0) - \frac{\mathrm{d}\Omega_2(\tau)}{\mathrm{d}\tau}\,\boldsymbol{\mathcal{T}}(\tau_f)\Big)^\intercal \boldsymbol{\xi} \\ &+ \frac{\mathrm{d}\Omega_1(\tau)}{\mathrm{d}\tau}\,y_0 + \frac{\mathrm{d}\Omega_2(\tau)}{\mathrm{d}\tau}\,y_f\Big], \end{aligned}$$

and

$$\begin{aligned} \frac{\mathrm{d}^2Y(t)}{\mathrm{d}t^2} = c^2\,\frac{\mathrm{d}^2Y(\tau)}{\mathrm{d}\tau^2} = \left(\frac{2}{t_f - t_0}\right)\Big[\Big(&\frac{\mathrm{d}^2\boldsymbol{\mathcal{T}}(\tau)}{\mathrm{d}\tau^2} - \frac{\mathrm{d}^2\Omega_1(\tau)}{\mathrm{d}\tau^2}\,\boldsymbol{\mathcal{T}}(\tau_0) - \frac{\mathrm{d}^2\Omega_2(\tau)}{\mathrm{d}\tau^2}\,\boldsymbol{\mathcal{T}}(\tau_f)\Big)^\intercal \boldsymbol{\xi} \\ &+ \frac{\mathrm{d}^2\Omega_1(\tau)}{\mathrm{d}\tau^2}\,y_0 + \frac{\mathrm{d}^2\Omega_2(\tau)}{\mathrm{d}\tau^2}\,y_f\Big]. \end{aligned}$$

The constrained expressions can now be plugged into Equation (3), which transforms the constrained ODE $\hat{f}$ into an unconstrained ODE $\check{f}$ consisting of the basis domain τ,

unknown Chebyshev coefficients $\boldsymbol{\xi}$, endpoints of the time domain (t_0 and t_f), and the boundary conditions of the constrained ODE $\boldsymbol{\beta}$:

$$\check{f}(\tau, \boldsymbol{\xi}, \boldsymbol{\beta}, t_f, t_0) = c\,\frac{\mathrm{d}^2 Y(\tau)}{\mathrm{d}\tau^2} - f\left(\tau, Y(\tau), c\,\frac{\mathrm{d}Y(\tau)}{\mathrm{d}\tau}\right) = 0,$$

where $\boldsymbol{\beta} = \{y_0, \quad y_f\}^\intercal$.

With the constrained expressions plugged into the DE residual, an algebraic system of equations can be formed after the domain is discretized with Chebyshev–Gauss–Lobatto nodes,

$$\tau_d = -\cos\frac{d\pi}{N} \quad \text{for all } d = (0, \ldots, N-1),$$

where N is the number of discretization points, also called collocation points. Note that having $N \geq L$ ensures that the algebraic system of equations formed via the loss equations is not undetermined. Chebyshev–Gauss–Lobatto nodes are better than uniformly spaced points for discretizing the domain because they avoid the Runge phenomena. Once the domain is discretized, a vector of loss functions at each discretization point can be expressed as

$$\mathcal{L}(\boldsymbol{\xi}) = \begin{Bmatrix} \check{f}(\tau_0, \boldsymbol{\xi}, \boldsymbol{\beta}, t_f, t_0) \\ \vdots \\ \check{f}(\tau_d, \boldsymbol{\xi}, \boldsymbol{\beta}, t_f, t_0) \\ \vdots \\ \check{f}(\tau_{N-1}, \boldsymbol{\xi}, \boldsymbol{\beta}, t_f, t_0) \end{Bmatrix}, \tag{6}$$

where $\boldsymbol{\xi}$ is the only unknown variable, which is why we write the loss only in terms of it here. If f is linear, then $\boldsymbol{\xi}$ within $\check{f}$ will show up linearly. Equation (6) then yields the form

$$\boldsymbol{A}\boldsymbol{\xi} + \boldsymbol{b} = 0, \tag{7}$$

where the matrix $\boldsymbol{A}$ is a linear combination of terms multiplied by $\boldsymbol{\xi}$, while the vector $\boldsymbol{b}$ is the loss vector evaluated at $\boldsymbol{\xi} = 0$. Linear least squares can then be used to solve Equation (7) for $\boldsymbol{\xi}$.

In the case of nonlinear DEs, Equation (6) will be nonlinear in the unknown ξ_j coefficients. The Gauss–Newton method, also known as iterative least squares, can then be used to solve for the change in $\boldsymbol{\xi}$ at each p iteration step, which is denoted by

$$\Delta\boldsymbol{\xi}_p = \boldsymbol{\xi}_{p+1} + \boldsymbol{\xi}_p. \tag{8}$$

The Gauss–Newton technique is expressed as

$$\Delta\boldsymbol{\xi}_p = -\Big(\boldsymbol{J}(\boldsymbol{\xi}_p)^\intercal \boldsymbol{J}(\boldsymbol{\xi}_p)\Big)^{-1} \boldsymbol{J}(\boldsymbol{\xi}_p)^\intercal \mathcal{L}(\boldsymbol{\xi}_p),$$

where $\boldsymbol{J}(\boldsymbol{\xi}_p)$ is the Jacobian of the loss vector with respect to the $\boldsymbol{\xi}$ variables and is computed at $\boldsymbol{\xi}_p$. Iterations are repeated until some stopping criterion(s) is met, which in this work is

$$\|\mathcal{L}(\boldsymbol{\xi}_p)\|_\infty < \varepsilon_{\mathrm{GN}}, \qquad \|\mathcal{L}(\boldsymbol{\xi}_{p+1})\|_\infty - \|\mathcal{L}(\boldsymbol{\xi}_p)\|_\infty < \varepsilon_{\mathrm{GN}}, \quad \text{or} \quad p = p_{\max}, \tag{9}$$

where $\varepsilon_{\mathrm{GN}}$ defines some user-prescribed tolerance, $\|\cdot\|_\infty$ refers to the L_∞-norm, and $p_{\max}$ is the maximum number of iterations. When any stopping criterion is met, the solution is obtained by substituting $\boldsymbol{\xi}$ into the constrained expression and mapping back to the problem domain, which in this example is time. If the last two stopping criteria were met before the first, then the Gauss–Newton algorithm failed to converge toward an accurate solution.

2.2. Solving Systems of ODEs

The example of the general TPBVP solved via the TFC method in the previous walk-through only contained a single ODE. TPBVPs composed of n ODEs can also be solved via the TFC. Consider the following TPBVP:

$$\ddot{y}_i(t) = f_i(t, \boldsymbol{y}(t), \dot{\boldsymbol{y}}(t)) \qquad \text{subject to} \qquad y_i(t_{il}) = \beta_{il}, \tag{10}$$

where $i \in (1, \ldots, n)$, $l \in (1, \ldots, N_{\text{const}_i})$ and $\boldsymbol{y} \in \mathbb{R}^n$. The β_{il} constant in Equation (10) is the l-th point constraint on the i-th differentiation variable. Furthermore, t_{il} is the time at which the l-th constraint occurs on the i-th differentiation variable. To solve this problem with the TFC, a constrained expression may be written for each i-th differentiation variable,

$$y_i(t) \approx Y_i(t) = g_i(t) + \sum_{l=1}^{N_{\text{const}_i}} \Omega_{il}(t) \varrho_{il}(t_{il}, g_i(t_{il})),$$

where N_{const_i} represents the number of constraints on the i-th differentiation variable.

The free function, switching functions, and projection functionals can be derived/chosen following the same procedure as in the previous subsection. Likewise, each constrained expression can be discretized to form a system of algebraic equations,

$$\begin{aligned} \mathcal{L}_i(\boldsymbol{\Xi}, \boldsymbol{B}, t_0, t_f) &= \begin{Bmatrix} \check{f}_i(\tau_0, \boldsymbol{\Xi}, \boldsymbol{B}, t_0, t_f) \\ \vdots \\ \check{f}_i(\tau_{N-1}, \boldsymbol{\Xi}, \boldsymbol{B}, t_0, t_f) \end{Bmatrix} \\ &= \begin{Bmatrix} c^2 \frac{\mathrm{d}^2 Y_i(\tau_0)}{\mathrm{d}^2 \tau} - f_i\left(\tau_0, \boldsymbol{Y}(\tau_0), c \frac{\mathrm{d}\boldsymbol{Y}(\tau_0)}{\mathrm{d}\tau}\right) \\ \vdots \\ c \frac{\mathrm{d}Y_i(\tau_{N-1})}{\mathrm{d}\tau} - f_i\left(\tau_{N-1}, \boldsymbol{Y}(\tau_{N-1}), c^2 \frac{\mathrm{d}^2 \boldsymbol{Y}(\tau_{N-1})}{\mathrm{d}^2 \tau}\right) \end{Bmatrix}, \end{aligned} \tag{11}$$

Consider that the $\boldsymbol{\xi}_i$ and $\boldsymbol{\beta}_i$ variables refer to the vector of Chebyshev coefficients and vector of point constraints associated with the i-th differentiation variable, respectively. Then, the $\boldsymbol{\Xi}$ and $\boldsymbol{\beta}$ variables in Equation (11) represent the vectors of all Chebyshev coefficient vectors and point constraint vectors appended together as follows:

$$\boldsymbol{\Xi} = \left\{\boldsymbol{\xi}_1^\mathsf{T}, \quad \ldots, \quad \boldsymbol{\xi}_i^\mathsf{T}, \quad \ldots, \quad \boldsymbol{\xi}_n^\mathsf{T}\right\}^\mathsf{T} \tag{12}$$

and

$$\boldsymbol{B} = \left\{\boldsymbol{\beta}_1^\mathsf{T}, \quad \ldots, \quad \boldsymbol{\beta}_i^\mathsf{T}, \quad \ldots, \quad \boldsymbol{\beta}_n^\mathsf{T}\right\}^\mathsf{T}.$$

Note that the lengths of $\boldsymbol{\xi}_i$ and $\boldsymbol{\beta}_i$ do not have to be identical. Furthermore, $\boldsymbol{Y} \in \mathbb{R}^n$ in Equation (11) is a vector of constrained expressions, where the i-th element approximates the i-th differentiation variable. An augmented algebraic system of equations can be formed as

$$\mathbb{L}(\boldsymbol{\Xi}) = \begin{Bmatrix} \mathcal{L}_1(\boldsymbol{\Xi}) \\ \vdots \\ \mathcal{L}_i(\boldsymbol{\Xi}) \\ \vdots \\ \mathcal{L}_n(\boldsymbol{\Xi}) \end{Bmatrix}.$$

We wrote the final form of the loss only in terms of the unknowns, as in the previous subsection. Substituting $\boldsymbol{\xi}$ with $\boldsymbol{\Xi}$ and $\mathcal{L}(\boldsymbol{\xi})$ with $\mathbb{L}(\boldsymbol{\Xi})$ into Equations (8) and (9) then allows $\boldsymbol{\Xi}$ to be computed via the Gauss–Newton method. If all of the f_i functions are linear, then linear least squares can be used.

2.3. Domain Decomposition

The domain decomposition technique we employ is the same as that presented by Johnston and Mortari [22]. In their work, the authors developed a TFC algorithm that decomposes the domain for solving TPBVPs related to hybrid systems (i.e., systems where a time-sequence of DEs governs the dynamics), along with those that are not but have dynamics that contain transient or stiff behavior. One example is the fuel-optimal planetary landing problem whose control action is bang-bang [18] (i.e., switches between a maximum and minimum, and vice versa). However, the numerical examples provided only involved splitting the domain into two or three segments. This work uses the same procedure when the domain is split into many more segments. How the domain is decomposed is described here for the reader's convenience. If the reader has any lingering questions, we recommend they read Refs. [18,22].

Assume a TPBVP with one ODE and two boundary conditions, as in Equation (1). To solve this problem with the domain decomposed, separate constrained expressions must be derived for each segment S_k for $k = (1, \ldots, K)$, where K is the total number of segments. Between each segment, C^1 continuity must be enforced and computed together with the Chebyshev coefficients. Figure 1 depicts a solution to the TPBVP where the domain is decomposed into multiple segments. Be aware that Y_t in the figure refers to the derivative of Y with respect to t. First, let us talk about the segments at the ends of the domain, i.e., S_1 and S_K. The initial segment needs a constrained expression that analytically satisfies three constraints: $y(t_0) = y_0$, $y(t_1) = y_1$, and $\dot{y}(t_1) = \dot{y}_1$. The boundary condition gives the first constraint, y_0, while the last two, y_1 and $\dot{y}_1$, are continuity conditions with the next segment. Similar to the first segment, the last segment's constraint expression analytically satisfies three constraints as well: $y(t_{K-1}) = y_{K-1}$, $y(t_f) = y_f$, and $\dot{y}(t_{K-1}) = \dot{y}_{K-1}$. Following the TFC process already introduced, the constrained expression for the first and last segments can be written as

$$
\begin{aligned}
{}^{(1)}Y\left(t, {}^{(1)}g(t)\right) = {}& {}^{(1)}g(t) + {}^{(1)}\Omega_1(t)\left(y_0 - {}^{(1)}g(t_0)\right) \\
& + {}^{(1)}\Omega_2(t)\left(y_1 - {}^{(1)}g(t_1)\right) + {}^{(1)}\Omega_3(t)\left(\dot{y}_1 - {}^{(1)}\dot{g}(t_1)\right)
\end{aligned}
$$

and

$$
\begin{aligned}
{}^{(K)}Y\left(t, {}^{(K)}g(t)\right) = {}& {}^{(K)}g(t) + {}^{(K)}\Omega_1(t)\left(y_{K-1} - {}^{(K)}g(t_{K-1})\right) \\
& + {}^{(K)}\Omega_2(t)\left(y_f - {}^{(K)}g(t_f)\right) + {}^{(K)}\Omega_3(t)\left(\dot{y}_{K-1} - {}^{(K)}\dot{g}(t_{K-1})\right),
\end{aligned}
$$

where the switching functions are

$$
\begin{aligned}
{}^{(1)}\Omega_1(t) &= \frac{1}{(t_1 - t_0)^2}\left(t_1^2 - t_1\, t + t^2\right) \\
{}^{(1)}\Omega_2(t) &= \frac{1}{(t_1 - t_0)^2}\left(t_0(t_0 - 2\,t_1) + 2\,t_1\, t - t^2\right) \\
{}^{(1)}\Omega_3(t) &= \frac{1}{t_1 - t_0}\left(t_0\, t_1 - (t_0 + t_1)t + t^2\right) \\
{}^{(K)}\Omega_1(t) &= \frac{1}{\left(t_f - t_{K-1}\right)^2}\left(t_f\left(t_f - 2\,t_{K-1}\right) + 2\,t_{K-1}t - t^2\right) \\
{}^{(K)}\Omega_2(t) &= \frac{1}{\left(t_f - t_{K-1}\right)^2}\left(-t_f\, t_{K-1} + \left(t_f + t_{K-1}\right)t - t^2\right) \\
{}^{(K)}\Omega_3(t) &= \frac{1}{t_f - t_{K-1}}\left(t_{K-1}^2 - 2\,t_{K-1}\, t + t^2\right).
\end{aligned}
$$

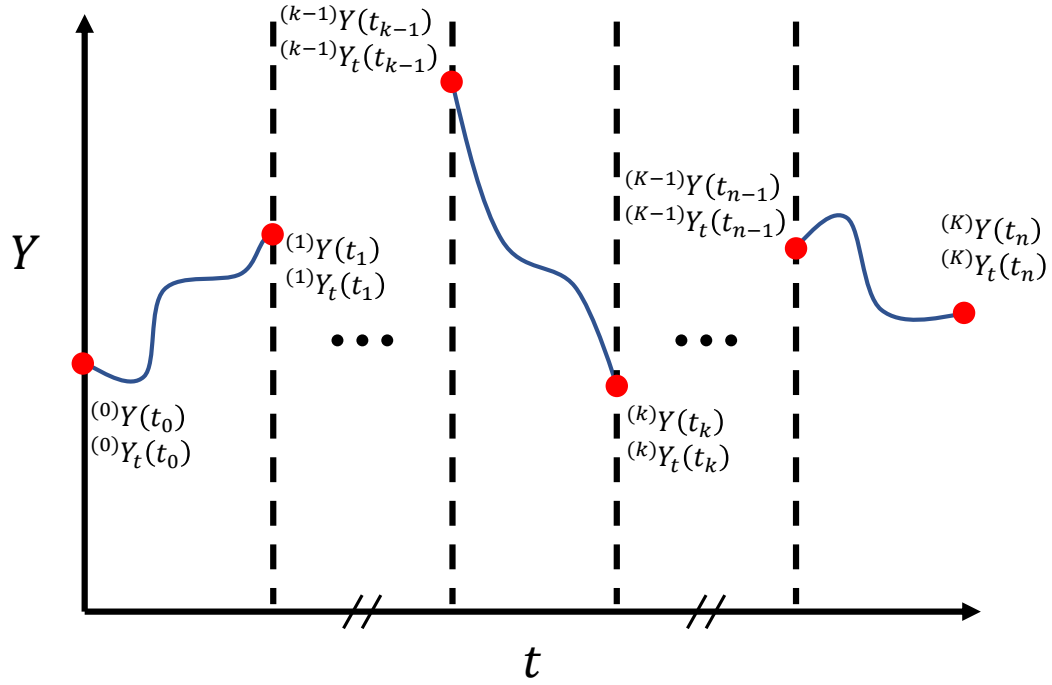


Figure 1. K-segment constrained expressions with C^1 continuity enforced. Red dots indicate the boundary/continuity conditions for each constrained expression.

The switching functions for the first and last segments are different because the continuity conditions are at the end of the first segment and the beginning of the last segment. On the other hand, the constrained expressions for the interior segments, S_k for all $k = (2, \ldots, K-1)$, have constrained expressions with identical switching functions because they always analytically satisfy three continuity constraints at both ends of their domain: $y(t_{k-1}) = y_{k-1}$, $y(t_k) = y_k$, $\dot{y}(t_{k-1}) = \dot{y}_{k-1}$, and $\dot{y}(t_k) = \dot{y}_k$. Unlike the end segments, none of the constraints for the interior segments are associated with the boundary conditions. Thus, the interior segments only have to deal with enforcing continuity. The constrained expressions for these segments can be written as

$$
\begin{aligned}
{}^{(k)}y\left(t, {}^{(k)}g(t)\right) = {}^{(1)}g(t) &+ {}^{(k)}\Omega_1(t)\left(y_k - {}^{(k)}g(t_{k-1})\right) + {}^{(k)}\Omega_2(t)\left(y_k - {}^{(k)}g(t_k)\right) \\
&+ {}^{(k)}\Omega_3(t)\left(\dot{y}_{k-1} - {}^{(k)}\dot{g}(t_{k-1})\right) + {}^{(k)}\Omega_4(t)\left(\dot{y}_k - {}^{(k)}\dot{g}(t_k)\right)
\end{aligned}
$$

where the switching functions are

$$
\begin{aligned}
{}^{(k)}\Omega_1(t) &= \frac{1}{(t_k - t_{k-1})^3}\left(-t_k^2(2t_{k-1} - t_k) + 6\,t_{k-1}\,t_k\,t - 3(t_{k-1} + t_k)t^2 + 2t^3\right) \\
{}^{(k)}\Omega_2(t) &= \frac{1}{(t_k - t_{k-1})^3}\left(-t_{k-1}^2(t_{k-1} - 3\,t_k) - 6\,t_{k-1}\,t_k\,t + 3(t_{k-1} + t_k)t^2 - 2\,t^3\right) \\
{}^{(k)}\Omega_3(t) &= \frac{1}{(t_k - t_{k-1})^2}\left(-t_{k-1}\,t_k^2 + t_k(2\,t_{k-1} + t_k)t - (t_{k-1} + 2\,t_k)t^2 + t^3\right) \\
{}^{(k)}\Omega_4(t) &= \frac{1}{(t_k - t_{k-1})^2}\left(-t_{k-1}^2\,t_k + t_{k-1}(t_{k-1} + 2\,t_k)t - (2\,t_{k-1} + t_k)t^2 + t^3\right).
\end{aligned}
$$

Now, assume the TPBVP is composed of n ODEs. The free functions within the various constrained expressions can be expressed as truncated Chebyshev polynomials on the domain $\tau \in [-1, 1]$. Using Equations (4) and (5), we have

$$ {}^{(k)}g_i(t) = {}^{(k)}g_i(\tau), \qquad \frac{\mathrm{d}^{(k)}g_i(\tau)}{\mathrm{d}t} = {}^{(k)}c\frac{\mathrm{d}^{(k)}g_i(\tau)}{\mathrm{d}\tau}, \qquad \text{and} \qquad c_k = \frac{2}{t_k - t_{k-1}}. $$

Then, by discretizing the domains, the augmented loss vector that combines the loss vectors for each differentiation variable in each segment can be written as

$$ {}^{(k)}\mathbb{L}\Big({}^{(k)}\boldsymbol{\Xi}, {}^{(k)}\boldsymbol{B}, t_{k-1}, t_k\Big) = \begin{Bmatrix} {}^{(k)}\mathcal{L}_1\Big({}^{(k)}\boldsymbol{\Xi}, {}^{(k)}\boldsymbol{B}, t_{k-1}, t_k\Big) \\ \vdots \\ {}^{(k)}\mathcal{L}_i\Big({}^{(k)}\boldsymbol{\Xi}, {}^{(k)}\boldsymbol{B}, t_{k-1}, t_k\Big) \\ \vdots \\ {}^{(k)}\mathcal{L}_n\Big({}^{(k)}\boldsymbol{\Xi}, {}^{(k)}\boldsymbol{B}, t_{k-1}, t_k\Big) \end{Bmatrix} $$

where

$$ \begin{aligned} {}^{(k)}\mathcal{L}_i\Big({}^{(k)}\boldsymbol{\Xi}, {}^{(k)}\boldsymbol{B}, t_{k-1}, t_k\Big) &= \begin{Bmatrix} \check{f}_i\Big(\tau_0, {}^{(k)}\boldsymbol{\Xi}, {}^{(k)}\boldsymbol{B}, t_{k-1}, t_k\Big) \\ \vdots \\ \check{f}_i\Big(\tau_{N-1}, {}^{(k)}\boldsymbol{\Xi}, {}^{(k)}\boldsymbol{B}, t_{k-1}, t_k\Big) \end{Bmatrix} \\ &= \begin{Bmatrix} c_k^2\,\frac{\mathrm{d}^{2(k)}Y_i(\tau_0)}{\mathrm{d}\tau^2} - f_i\Big(\tau_0, {}^{(k)}\boldsymbol{Y}(\tau_0), c_k\,\frac{\mathrm{d}^{(k)}\boldsymbol{Y}(\tau_0)}{\mathrm{d}\tau}\Big) \\ \vdots \\ c_k^2\,\frac{\mathrm{d}^{2(k)}Y_i(\tau_{N-1})}{\mathrm{d}\tau^2} - f_i\Big(\tau_{N-1}, {}^{(k)}\boldsymbol{Y}(\tau_{N-1}), c_k\,\frac{\mathrm{d}^{(k)}\boldsymbol{Y}(\tau_{N-1})}{\mathrm{d}\tau}\Big) \end{Bmatrix}. \end{aligned} $$

The unknown variable ${}^{(k)}\boldsymbol{\Xi}$ represents the k-th segment's appended unknown Chebyshev coefficient vector,

$$ {}^{(k)}\boldsymbol{\Xi} = \Big\{ {}^{(k)}\boldsymbol{\xi}_1^\intercal, \quad \ldots, \quad {}^{(k)}\boldsymbol{\xi}_i^\intercal, \quad \ldots, \quad {}^{(k)}\boldsymbol{\xi}_n^\intercal \Big\}^\intercal. $$

Likewise, ${}^{(k)}\boldsymbol{B}$ refers to the augmented vector of point constraints for the k-th segment and can be written as

$$ {}^{(k)}\boldsymbol{B} = \Big\{ {}^{(k)}\boldsymbol{\beta}_1^\intercal, \quad \ldots, \quad {}^{(k)}\boldsymbol{\beta}_i^\intercal, \quad \ldots, \quad {}^{(k)}\boldsymbol{\beta}_n^\intercal \Big\}^\intercal. $$

Not all point constraints are known in constrained expressions that approximate only a segment of the domain. Only the initial conditions on S_1 and final conditions on S_K are known. All continuity conditions within a k-th segment and for an i-th differentiation variable are unknown and are represented by ${}^{(k)}\hat{\boldsymbol{\beta}}_i$. Grouping the continuity conditions in a segment gives

$$ {}^{(k)}\hat{\boldsymbol{B}} = \Big\{ {}^{(k)}\hat{\boldsymbol{\beta}}_1{}^\intercal, \quad \ldots, \quad {}^{(k)}\hat{\boldsymbol{\beta}}_i{}^\intercal, \quad \ldots, \quad {}^{(k)}\hat{\boldsymbol{\beta}}_n{}^\intercal \Big\}^\intercal. $$

Likewise, grouping the continuity conditions in all segments gives

$$ \hat{\boldsymbol{B}} = \mathrm{unique}\Big(\Big\{ {}^{(1)}\hat{\boldsymbol{B}}^\intercal, \quad \ldots, \quad {}^{(k)}\hat{\boldsymbol{B}}^\intercal, \quad \ldots, \quad {}^{(K)}\hat{\boldsymbol{B}}^\intercal \Big\}^\intercal \Big), $$

where $\mathrm{unique}(\cdot)$ means only consider the unique elements of the $\cdot$ vector. All unknowns can then be grouped to form

$$ \boldsymbol{Z} = \Big\{ {}^{(1)}\boldsymbol{\Xi}^\intercal, \quad \ldots, \quad {}^{(k)}\boldsymbol{\Xi}^\intercal, \quad \ldots, \quad {}^{(K)}\boldsymbol{\Xi}^\intercal, \hat{\boldsymbol{B}}^\intercal \Big\}^\intercal. $$

Exactly how $\hat{\boldsymbol{B}}$ can be built for a specific TPBVP is shown in Appendix A.

The complete augmented loss vector that contains all segments and forms the system of algebraic equations to be solved is

$$\mathbb{L}(\boldsymbol{Z}) = \begin{Bmatrix} {}^{(1)}\mathbb{L}(\boldsymbol{Z}) \\ \vdots \\ {}^{(k)}\mathbb{L}(\boldsymbol{Z}) \\ \vdots \\ {}^{(K)}\mathbb{L}(\boldsymbol{Z}) \end{Bmatrix}.$$

Once again, $\mathbb{L}$ is only explicitly written in terms of $\boldsymbol{Z}$ because the boundary conditions within ${}^{(1)}\boldsymbol{B}$ and ${}^{(K)}\boldsymbol{B}$ are known, and so too are t_k for all $k = (1, \ldots, K)$. The unknown $\boldsymbol{Z}$ values can be computed using linear least squares when every f_i is linear or the Gauss–Newton algorithm when any f_i is nonlinear. The $\mathbb{L}$ algebraic system will be taken in a block diagonal form due to its segment-wise construction, allowing efficient algorithms for computing sparse matrices to be exploited.

3. Error Estimation

The most critical aspect of an *hp*-adaptive mesh refinement algorithm is how it relates the approximation error to when the order of the approximating polynomial should be increased (*p*-adaptive mesh refinement) or when an approximating polynomial should be divided into more segments (*h*-adaptive mesh refinement). For the TFC, increasing the approximating polynomial order corresponds to adding more basis functions to the constrained expression's free function (a truncated expansion of Chebyshev polynomials). In the previous section, it was shown that the TFC minimizes $\|\mathbb{L}\|_\infty$ to obtain an accurate solution. However, a criterion for when to increase the number of basis functions in a constrained expression or split the constrained expression into multiple segments can be made by examining the constrained expression's truncation error. Therefore, that criterion is presented in this section, along with an error analysis of a few problems to demonstrate that a low truncation error corresponds with a small $\|\mathbb{L}\|_\infty$.

3.1. Truncation Error of the Constrained Expression

Let $y(\tau)$ be a piecewise smooth bounded function on the interval $\tau \in [-1, +1]$ that has N_{const} point constraints at τ_l with $l = (1, \ldots, N_{\text{const}})$. Then, $y(\tau)$ can be approximated with a constrained expression $Y(\tau)$, where the free function is a linear combination of L Chebyshev polynomials, $\mathcal{T}_j$ for all $j = (N_{\text{const}}, \ldots, L + N_{\text{const}} - 1)$. Assuming point constraints, the projection functionals within the constrained expression take the form

$$\varrho_j(\tau_l) = y(\tau_l) - \sum_{j=N_{\text{const}}}^{L+N_{\text{const}}-1} \xi_j \, \mathcal{T}_j(\tau_l).$$

Hence, the approximation (constrained expression) takes the form

$$y(\tau) \approx Y(\tau) = \sum_{j=N_{\text{const}}}^{L+N_{\text{const}}-1} \xi_j \, \mathcal{T}_j(\tau) + \sum_{l=1}^{N_{\text{const}}} \Omega_l(\tau) \left(y(\tau_l) - \sum_{j=N_{\text{const}}}^{L+N_{\text{const}}-1} \xi_j \mathcal{T}_j(\tau_l) \right), \tag{13}$$

where Ω_l is the switching function for the l-th constraint. Since $y(\tau)$ is piecewise smooth and bounded, it can be represented exactly by an infinite Chebyshev polynomial series,

$$y(\tau) = \sum_{j=0}^{\infty} a_j \, \mathcal{T}_j(\tau) = \sum_{j=0}^{N_{\text{const}}-1} a_j \, \mathcal{T}_j(\tau) + \sum_{j=N_{\text{const}}}^{L+N_{\text{const}}-1} a_j \, \mathcal{T}_j(\tau) + \sum_{j=L+N_{\text{const}}}^{\infty} a_j \, \mathcal{T}_j(\tau),$$

where a_j for all $j = (0, \ldots, \infty)$ are the Chebyshev coefficients of the infinite series. The Euclidean norm, otherwise called the L_2-norm, of the error between $y(\tau)$ and $Y(\tau)$ satisfies the following inequality:

$$\begin{aligned} e &= \|y(\tau) - Y(\tau)\|_2 \\ &= \left\| \sum_{j=0}^{\infty} a_j \mathcal{T}_j(\tau) - \left(\sum_{j=N_{\text{const}}}^{L+N_{\text{const}}-1} \xi_j \mathcal{T}_j(\tau) + \sum_{l=1}^{N_{\text{const}}} \Omega_l(\tau) \left(y(\tau_l) - \sum_{j=N_{\text{const}}}^{L+N_{\text{const}}-1} \xi_j \mathcal{T}_j(\tau_l) \right) \right) \right\|_2 \\ &\leq e_{\text{T}} + e_{\text{A}} + e_{\text{C}}, \end{aligned}$$

where e_{T} is the truncation error, e_{A} is the aliasing error, and e_{C} is what we call the constraint error. They are given as

$$\begin{aligned} e_{\text{T}} &= \left\| \sum_{j=L+N_{\text{const}}}^{\infty} a_j \mathcal{T}_j(\tau) \right\|_2, \\ e_{\text{A}} &= \left\| \sum_{j=N_{\text{const}}}^{L+N_{\text{const}}-1} (a_j - \xi_j) \mathcal{T}_j(\tau) \right\|_2, \end{aligned} \tag{14}$$

and

$$e_{\text{C}} = \left\| \sum_{j=0}^{N_{\text{const}}-1} a_j \mathcal{T}_j(\tau) - \sum_{l=1}^{N_{\text{const}}} \Omega_l(\tau) \left(y(\tau_l) - \sum_{j=N_{\text{const}}}^{L+N_{\text{const}}-1} \xi_j \mathcal{T}_j(\tau_l) \right) \right\|_2.$$

The aliasing and constraint errors can only be calculated if the function being approximated is known. A critical conclusion about the magnitude of the truncation error can be made from Rule of Thumb 1, as presented by Boyd [32] (p. 32):

Rule of Thumb 1. *e_{A} will be of the same order of magnitude as e_{T}.*

Furthermore, through testing, we found that the constraint error is of a similar magnitude to the aliasing error. Thus, if the truncation error is low, so too should be the aliasing and constraint errors.

Note that $\|f\|_2 = \langle f, f \rangle^{1/2}$ is the norm induced by the inner product

$$\langle f, g \rangle = \int_{-1}^{1} f(\tau)\, g(\tau)\, \mathrm{d}\tau. \tag{15}$$

Using the definition of the inner product in Equation (15), the orthogonal property of Chebyshev polynomials is

$$\langle \mathcal{T}_m, \mathcal{T}_n \rangle = \int_{-1}^{1} \mathcal{T}_m(\tau)\, \mathcal{T}_n(\tau) \left(1 - \tau^2\right)^{-1/2} \mathrm{d}\tau = \frac{\pi}{2} \delta_{mn}, \tag{16}$$

where δ_{mn} is the Kronecker delta function. Multiplying $\left(1 - \tau^2\right)^{-1/4}$ by Equation (14) and applying the absolute homogeneity of norms gives

$$\begin{aligned} \left(1 - \tau^2\right)^{-1/4} e_{\text{T}} &= \left(1 - \tau^2\right)^{-1/4} \left\| \sum_{j=L+N_{\text{const}}}^{\infty} a_j \mathcal{T}_j(\tau) \right\|_2 \\ &= \left\| \sum_{j=L+N_{\text{const}}}^{\infty} a_j \mathcal{T}_j(\tau) \left(1 - \tau^2\right)^{-1/4} \right\|_2. \end{aligned} \tag{17}$$

Expanding the norm in Equation (17), substituting in Equation (15), and plugging in Equation (16) gives a new expression for e_{T},

$$\begin{aligned} e_{\mathrm{T}} &= \left(1-\tau^2\right)^{1/4} \left\| \sum_{j=L+N_{\text{const}}}^{\infty} a_j \, \mathcal{T}_j(\tau) \left(1-\tau^2\right)^{-1/4} \right\|_2 \\ &= \left(1-\tau^2\right)^{1/4} \left[\sum_{j=L+N_{\text{const}}}^{\infty} \int_{-1}^{1} a_j^2 \, \mathcal{T}_j^2(\tau) \left(1-\tau^2\right)^{-1/2} \mathrm{d}\tau \right]^{1/2} \\ &= \left(1-\tau^2\right)^{1/4} \left[\sum_{j=L+N_{\text{const}}}^{\infty} a_j^2 \int_{-1}^{1} \mathcal{T}_j^2(\tau) \left(1-\tau^2\right)^{-1/2} \mathrm{d}\tau \right]^{1/2} \\ &= \left(\frac{\pi}{2}\right)^{1/2} \left(1-\tau^2\right)^{1/4} \left[\sum_{j=L+N_{\text{const}}}^{\infty} a_j^2 \right]^{1/2}. \end{aligned} \tag{18}$$

It has been shown that the Chebyshev series has super-geometric convergence when $y(\tau)$ is analytic in the neighborhood of $\tau \in [-1, +1]$ [32,33]. Hence, the Chebyshev coefficient values from the infinite expansion a_j for all $j = (0, \ldots, \infty)$ decay as

$$|a_j| \sim \mathcal{O}\left(s \, e^{-\tilde{q} \, j^r}\right),$$

with constants $s, \tilde{q} > 0$, and $r > 1$. Following in the spirit of Lie et al. [27], we note that a slightly smaller value $q > 0$ exists, such that

$$s \, e^{-\tilde{q} \, j^r} \leq \left(s - \min_j\left(s \, e^{-q \, j} - |a_j|\right)\right) e^{-q \, j}. \tag{19}$$

The exponential upper bound is more convenient because it allows the Chebyshev coefficients to be approximated with an exponential least-squares fit. The minimum in Equation (19) represents a translation that ensures every value of a_j is below the bound. Since the solution of the problem being solved is often not known, the $|a_j|$ values cannot be computed to perform the regression, but the truncated coefficients from the constrained expression $|\xi_j|$ can. Based on Rule of Thumb 1, $a_j \approx \xi_j$ when the truncation error is low. Thus, simply replace a_j with ξ_j in Equation (19) to obtain an upper bound estimate for a_j for all $j = (N_{\text{const}}, \ldots, L + N_{\text{const}} - 1)$. Extrapolation can then be used to obtain an upper bound for $j = (0, \ldots, \infty)$. As a result, in this work, the upper bound for $|a_j|$ is approximated by

$$\ln\left(|a_j|\right) \approx \ln\left(|\xi_j|\right) \leq \ln\left(s\right) - \delta - q \, j, \tag{20}$$

with $s, q > 0$, $j = (N_{\text{const}}, \ldots, L + N_{\text{const}} - 1)$, and

$$\delta = \min_j\left(\ln\left(s\right) - q \, j - \ln\left(|\xi_j|\right)\right).$$

The s and q coefficients are obtained by performing an exponential least-squares fit on $|\xi_j|$,

$$\ln\left(|\xi_j|\right) = \ln\left(s\right) - q \, j. \tag{21}$$

Note that q represents the exponential decay of the Chebyshev coefficients. Furthermore, an upper bound of e_{T} can be derived by plugging Equation (20) into (18),

$$\begin{aligned} e_{\mathrm{T}} &\leq \left(\frac{\pi}{2}\right)^{1/2} \left(1-\tau^2\right)^{1/4} \left[\sum_{j=L+N_{\text{const}}}^{\infty} e^{2\left(\ln\left(s\right) - \delta - q \, j\right)} \right]^{1/2} \\ &\leq \left(\frac{\pi}{2}\right)^{1/2} \left(1-\tau^2\right)^{1/4} \left[\sum_{j=L+N_{\text{const}}}^{\infty} \left(s^2 \, e^{-2\,\delta}\right) e^{-2\,q\,j} \right]^{1/2}. \end{aligned} \tag{22}$$

The summation in Equation (22) is a geometric series with the common ratio e^{-2q},

$$\sum_{j=L+N_{\text{const}}}^{\infty} \left(s^2 e^{-2\delta}\right) e^{-2qj} = \frac{\left(s^2 e^{-2\delta}\right) e^{-2q(L+N_{\text{const}})}}{1-e^{-2q}}.$$

Thus, Equation (22) becomes

$$\begin{aligned} e_{\text{T}} &\leq \left(\frac{\pi}{2}\right)^{1/2} \left(1-\tau^2\right)^{1/4} \left[\frac{\left(s^2 e^{-2\delta}\right) e^{-2q(L+N_{\text{const}})}}{1-e^{-2q}}\right]^{1/2} \\ &\leq \frac{\pi^{1/2} \left(1-\tau^2\right)^{1/4} \left(s\, e^{-\delta}\right) e^{-q(L+N_{\text{const}})}}{\left(2\left(1-e^{-2q}\right)\right)^{1/2}} \\ &\leq \frac{\pi^{1/2}\, s\, e^{(-\delta-q(L+N_{\text{const}}))}}{\left(2\left(1-e^{-2q}\right)\right)^{1/2}} = \hat{e}_{\text{T}}. \end{aligned} \tag{23}$$

From Equations (20) and (23), one can see that the coefficients $|\xi_j|$ and the approximate truncation error upper bound $\hat{e}_{\text{T}}$ decrease at the same exponential rate q, a result similar to that seen in Liu et al. [27]. Consequently, the Chebyshev coefficients ξ_j for all $j = (N_{\text{const}}, \ldots, L + N_{\text{const}} - 1)$ can be used to estimate the decay rate, q, of the truncation error as a function of the free function's Chebyshev polynomial degree given in Equation (13).

Before moving on, the reader may have noticed that the L_2-norm is used to derive the analytical expression for the truncation error (see Equation (14)). The L_2-norm had to be used to derive the expression using the orthogonal property of Chebyshev polynomials (Equation (16)). On the other hand, the accuracy of the TFC is quantified by taking the L_∞-norm of the losses (see Equation (9)). In general, the L_2-norm of the loss can also be used to quantify the accuracy of the TFC, but we chose to use the L_∞-norm in order to align with the only textbook on the TFC [1]. Although the L_2-norm of the truncation error and the L_∞-norm of the loss are two separate measurements, for one to be used for *hp*-adaptive mesh refinement and the other used to be used as a stopping criterion, it is required that both correlate. When one decreases, so must the other, and vice versa. This correlation is demonstrated in Sections 3.3 and 3.4.

3.2. Determining Function Smoothness

How to determine when to add basis functions to a constrained expression within a segment or split it up is a critical aspect of our mesh refinement algorithm. In the same vein as Liu et al. [27], this decision can be made based on the decay rate of the ξ_j coefficients and the truncation error, i.e., the q value from Equations (21) and (23). The main distinction between our work and [27] is that our decay rate comes from an exponential fit on the truncated Chebyshev expansion coefficients within an approximating constrained expression, whereas the decay rate from [27] comes from an exponential fit on the truncated Legendre expansion coefficients computed from an approximating Lagrange polynomial. Similar to [27], the function is smooth enough to be approximated with a single polynomial if q is large. In other words, only basis functions need to be added to bring the error down to some desired tolerance. If q is sufficiently small, then the function is not smooth enough to be approximated with a single polynomial, and a piecewise polynomial should be employed to approximate it. The cutoff decay rate value $\bar{q}$, which decides when to add basis functions or split a segment, is a design choice determined by the engineer. When $q \geq \bar{q}$, basis functions should be added to the free function of the constrained expression to improve the approximation. On the other hand, when $q < \bar{q}$, the segment the constrained expression approximates should be divided to improve the approximation. In the next two subsections, error analysis studies are carried out on a normal TPBVP with a smooth solution and a hybrid TPBVP with a nonsmooth solution to demonstrate agreement between the decay rate of the ξ_j coefficients and the truncation error's maximum upper

bound $\hat{e}_{\mathrm{T}}$, as well as determine how they correspond to the aliasing error, constraint error, actual error, and L_∞-norm of the loss.

3.3. Case 1 Error Analysis: TPBVP with a Smooth Solution

Consider the following TPBVP:

$$2\,\ddot{y}_1(t) - y_1(t) = 4\,\sin(2\,t), \quad \text{s.t.} \begin{cases} y(0) = 0 \\ y(1) = 0 \end{cases}, \tag{24}$$

which has the analytical solution

$$y_1(t) = c_1\,e^{t/\sqrt{2}} + c_2\,e^{-t/\sqrt{2}} - \frac{4}{19}\,\sin(3\,t),$$

where

$$c_1 = -\frac{4\,e^{1/\sqrt{2}}\,\sin(3)}{19\left(-e^{\sqrt{2}}+1\right)}$$

and

$$c_2 = \frac{4\,e^{1/\sqrt{2}}\,\sin(3)}{19\left(-e^{\sqrt{2}}+1\right)}.$$

It can be seen that $y_1(t)$ is a smooth function. Using the TFC procedure outlined in Section 2, Equation (24) can be solved by approximating y_1 as a constrained expression with the truncated Chebyshev coefficients ξ_j. Using the computed ξ_j values via the TFC, a least-squares exponential fit of the form shown in Equation (21) can be constructed. Equation (20) can then be used to build an upper bound on the ξ_j values. Subsequently, an upper bound on the truncation error $\hat{e}_{\mathrm{T}}$ can be computed, as shown in Equation (23). The aliasing error e_{A} and the constraint error e_{C} can also be computed because the true solution is known. Only the infinite Chebyshev expansion coefficients a_j have to be computed, which we performed using the open-source MATLAB toolbox Chebfun [34]. Figure 2 shows how the actual error; two variants of $\hat{e}_{\mathrm{T}}$, e_{A}, e_{C}; two estimated error bounds ($e = \hat{e}_{\mathrm{T}} + e_{\mathrm{A}} + e_{\mathrm{C}}$); the L_∞-norm of the loss shown in Equation (6); and the final Chebyshev coefficient $\xi_{L+N_{\mathrm{const}}-1}$ changed as the number of Chebyshev basis functions L increased. The number of collocation points N was set equal to $L+3$. Note that the results in Figure 2 correspond to Rule of Thumb 2 from Boyd [32] (p. 51), which states the following:

Rule of Thumb 2. *The truncation error is the same order of magnitude as the last coefficient retained in the truncation for a series with geometric convergence.*

For $L \geq 15$, the $\xi_{L+N_{\mathrm{const}}-1}$ coefficients are so low in value that they are negligible. Hence, the accuracy of the solution will not improve by adding more than 15 basis functions to the Chebyshev polynomial. This can be observed by following the $\|\mathbb{L}\|_\infty$ and e curves in Figure 2. All coefficients with $j > L_R$, where $L_R = 15$, are called the roundoff plateau [32] (pp. 30–31) because their values are less than the machine-level error, which is 1×10^{-16}. Since the inclusion of Chebyshev series terms with their coefficients on the roundoff plateau has no effect on the accuracy of the solution, they do not have to be included in the exponential regression. Not including them results in an improved upper bound (IUB) for the non-negligible ξ_j coefficients, i.e., those that are off the roundoff plateau. Including the ξ_j coefficients that are on the roundoff plateau in the exponential regression results in a conservative upper bound (CUB) for them. The resulting truncation errors from using a CUB or IUB on the ξ_j coefficients are referred to as CUB $\hat{e}_{\mathrm{T}}$ and IUB $\hat{e}_{\mathrm{T}}$ in Figure 2. The CUB and IUB $\hat{e}_{\mathrm{T}}$ curves are practically identical until $L > L_R$. The CUB causes $\hat{e}_{\mathrm{T}}$ to level out, while the IUB causes $\hat{e}_{\mathrm{T}}$ to keep decreasing for larger values of L. One of the main takeaways from Figure 2 is that it validates the truncation error bound because the actual error is always below the estimated total error bounds with a CUB and IUB $\hat{e}_{\mathrm{T}}$. Figure 2 also

shows that when $\hat{e}_{\mathrm{T}}$ is low, so too is $\|\mathbb{L}\|_\infty$. Lastly, both truncation errors follow $\xi_{L+N_{\mathrm{const}}-1}$, demonstrating that the decay rate of the ξ_j coefficients is the same as the truncation error for a smooth analytic function.

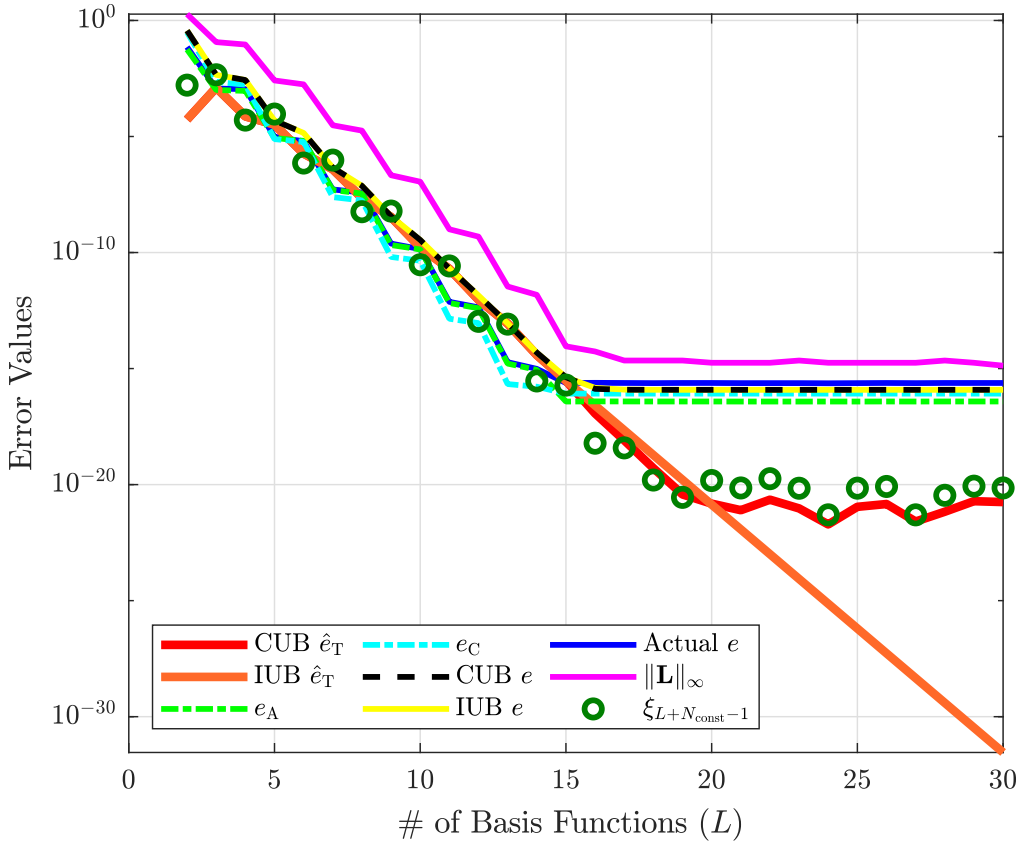


Figure 2. Error analysis of the TFC solution to Equation (24).

Figure 3 aids in the analysis of the exponential regression performed on the ξ_j values computed via the TFC for solving Equation (24). Subplot (**a**) shows the ξ_j coefficients for $j = (2, \ldots, L)$ when $L = 30$, the exponential fit performed on all coefficients (Exp. Fit 1), the exponential fit performed on only the coefficients of the roundoff plateau (Exp. Fit 2), the CUB, and the IUB. From the subplot, one can clearly see that the IUB bounds the ξ_j coefficients off the roundoff plateau tighter than the CUB. Furthermore, it is clear that geometric convergence is maintained until the roundoff plateau is reached. Subplot (**b**) shows the goodness of the exponential fit according to the coefficient of determination R^2 as L increases. R^2 can be expressed as

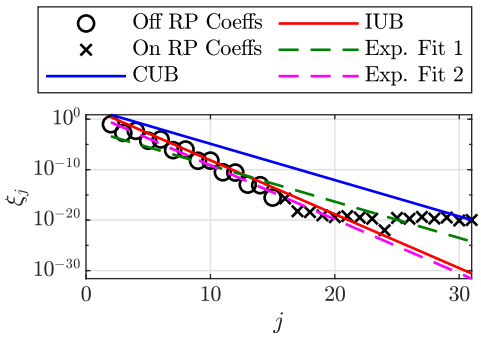
$$R^2 = 1 - \frac{\sum_{j=N_{\mathrm{const}}}^{L+N_{\mathrm{const}}} \left(|\xi_j| - f_j\right)^2}{\sum_{i=N_{\mathrm{const}}}^{L+N_{\mathrm{const}}} \left(|\xi_j| - \mathrm{mean}(\xi)\right)^2},$$

where f_j is the value of the fit for the j-th coefficient and

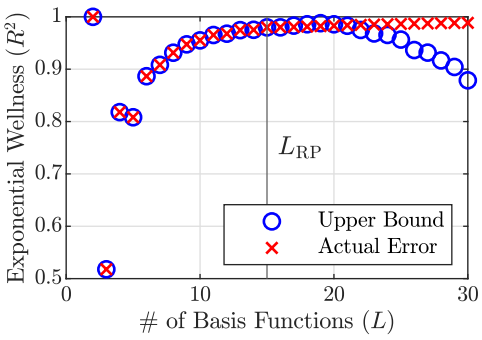
$$\mathrm{mean}(\xi) = \frac{1}{L+1} \sum_{i=N_{\mathrm{const}}}^{L+N_{\mathrm{const}}} |\xi_j|.$$

From subplot (**a**), one can see that the exponential fit is not very good when all ξ_j coefficients are included in the regression. This is confirmed by subplot (**b**) because R^2 for Exp. Fit 1 moves further from 1 when more ξ_j coefficients with $j > L_R$ are included in the regression. However, when only the ξ_j coefficients off the roundoff plateau are fitted,

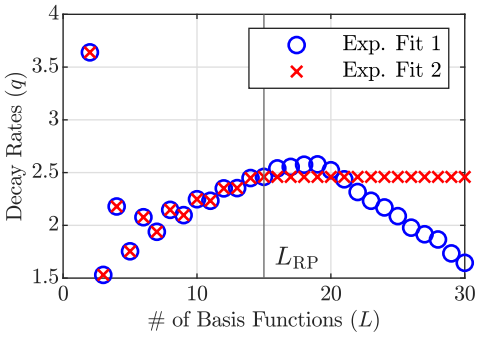
$R^2 \approx 1$ as j increases. Subplot (**c**) shows the decay rates q computed via Exp. Fit 1 and Exp. Fit 2 as L increases. One can see that all decay rates are greater than 1.5, which is emblematic of the y_1 function's smoothness. The different decay rates are identical until $L = L_{\text{RP}} = 15$. The Exp. Fit 2 decay rates for $L > L_{\text{RP}}$ are constant because the ξ_j coefficients included in the fit are only for $j = 2, \ldots, L_{\text{RP}}$. One can see that the Exp. Fit 1 decay rates are greater than those for the Exp. Fit 2 directly after L_{RP} is reached but then start to decrease quickly as L is further increased. Hence, when $L = 50$, the decay rate for Exp. Fit 1 could be close to 0. This means that a smooth function could have a very low decay rate if the round-off plateau is included in the exponential regression of the ξ_j coefficients. Therefore, Exp. Fit 2 is preferred over Exp. Fit 1.



(**a**) Exponential regression



(**b**) Exponential wellness



(**c**) Decay rates

Figure 3. Three plots that aid in the analysis of the exponential fits performed on the truncated Chebyshev coefficients computed with TFC for the TPBVP shown in Equation (24). Subplot (**a**) shows Exp. Fit 1 for $L = 30$, where all truncated Chebyshev coefficients are included in the regression, and Exp. Fit 2 for $L = 30$, where only the truncated Chebyshev coefficients off the roundoff plateau (RP) are included in the regression. The resulting conservative upper bound (CUB) and improved upper bound (IUB) when Exp. Fit 1 and Exp. Fit 2 are used, respectively, are also shown. Subplot (**b**,**c**) shows the goodness and decay rates of each fit for increasing values of L.

3.4. Case 2 Error Analysis: Hybrid TPBVP with a Nonsmooth Solution

Consider the hybrid TPBVP

$$\ddot{y}(t) = t^2 + a, \quad \text{subject to: } \begin{cases} y(0) = 0 \\ y(1) = 0 \end{cases} \quad \text{where: } \begin{cases} a = 0 & \text{for } t \leq 0.5 \\ a = 1 & \text{for } t > 0.5 \end{cases}. \tag{25}$$

which has a discontinuous solution. We instead find a smoothed solution by transforming Equation (25) into

$$\ddot{y}_2(t) = t^2 + 1\left(\frac{1}{2} - \frac{1}{2}\tanh\left(\frac{\gamma - t}{\alpha}\right)\right), \quad \text{subject to: } \begin{cases} y(0) = 0 \\ y(1) = 0 \end{cases}, \tag{26}$$

which has the analytical solution

$$y_2(t) = \frac{1}{12}\left(-3\alpha^2 \mathrm{Li}_2\left(-e^{\frac{-2(\gamma-t)}{\alpha}}\right) + 6\alpha\ln(2)(\gamma - t) - 3\gamma^2 + 6\gamma t + t^4\right) + c_1 t + c2,$$

where

$$c_1 = \frac{1}{4}\alpha^2 \mathrm{Li}_2\left(-e^{\frac{-2(\gamma-1)}{\alpha}}\right) - \frac{1}{4}\alpha^2 \mathrm{Li}_2\left(-e^{\frac{-2\gamma}{\alpha}}\right) - \frac{\gamma}{2} + \frac{\alpha\ln(2)}{2} + \frac{11}{12},$$

$$c_2 = \frac{1}{4}\alpha^2 \mathrm{Li}_2\left(-e^{\frac{-2\gamma}{\alpha}}\right) - \frac{\gamma\alpha\ln(2)}{2} + \frac{\gamma^2}{4},$$

and

$$\mathrm{Li}_2(x) = \int_2^x \frac{\mathrm{d}\tau}{\ln(\tau)}.$$

The γ variable represents the discontinuous point, which is set to $\gamma = 0.5$. The α variable determines how smooth the region around $t = \gamma$ is. The closer α is to 0, the sharper the gradient at $t = \gamma$, and the closer the solution of Equation (26) to Equation (25). Thus, we set $\alpha = 1.5 \times 10^{-3}$. An error analysis study, similar to that shown in Section 3.3, was then performed. The main difference from the previous study is that y_2 is nearly not smooth. This results in the ξ_j coefficients, computed via the TFC, oscillating severely between local maximums and local minimums. Hence, in this study, Exp. Fit 2 refers to performing an exponential regression on only the local maximums of the ξ_j coefficients. Exp. Fit 1 refers to performing an exponential regression on all ξ_j coefficients, just like in Section 3.3. The resulting CUB and IUB are from using Equation (20) with Exp. Fit 1 and Exp. Fit 2, respectively.

Error curves of the TFC solutions to Equation (26) as L increases are shown in Figure 4. Compared with Figure 2, many more basis functions are required for the actual error to converge. Furthermore, the L_∞-norm of the loss is still decreasing and greater than the actual error. The reason for this is that the sharp gradient that replaces the discontinuity in Equation (25) appears for $\dot{y}_2$, which contributes to the loss equation, while the actual error is only for the constrained expression of y_2. Another important observation from Figure 4 is that both truncation error bounds are well above the actual error, meaning they do not accurately represent the total error. Rule of Thumb 1, as described in Section 3.1, does not seem to apply. However, this makes sense because our selection of $\alpha = 1.5 \times 10^{-3}$ creates such a sharp gradient that the solution has a difficult time being approximated by a global polynomial, i.e., it is nearly not smooth. However, the IUB $\hat{e}_\mathrm{T}$ is much closer to the actual error and the L_∞-norm of the loss curve than the CUB $\hat{e}_\mathrm{T}$. This is because the IUB $\hat{e}_\mathrm{T}$ is, as its name suggests, an improved upper bound, considering only the local maximums of the ξ_j coefficients during the exponential regression, as shown in Figure 5.

Subplot (**a**) in Figure 5 shows the ξ_j coefficients for $j = (2, \ldots, L)$ belonging to the TFC solution when $L = 4000$, the exponential fit performed on all coefficients (Exp. Fit 1), the exponential fit performed on only the local maximums (Exp. Fit 2), the CUB, and the IUB. As can be clearly seen, the coefficients fluctuate significantly. This results from the Gibbs Phenomenon, which is the oscillatory behavior of an approximation around a jump discontinuity, or in our case, a sharp gradient nearly identical to a jump discontinuity. The oscillatory behavior of the coefficients causes their exponential fit to be poor, as seen in subplot (**b**), which shows the R^2 value as L increases. When an exponential fit of only the local maximum ξ_j coefficients is performed, the R^2 values are much closer to 1, and the resulting upper bound is much less conservative while still bounding all coefficients. Comparing Figure 3a with Figure 5a, one can see that the local maximum ξ_j coefficients for approximating y_2 yield algebraic convergence instead of geometric convergence. This is a sign that y_2 is very close to not being smooth, if not already. Indeed, geometric convergence is only a given for a truncated series of Chebyshev polynomials if the function they approximate is definitively continuous and smooth.

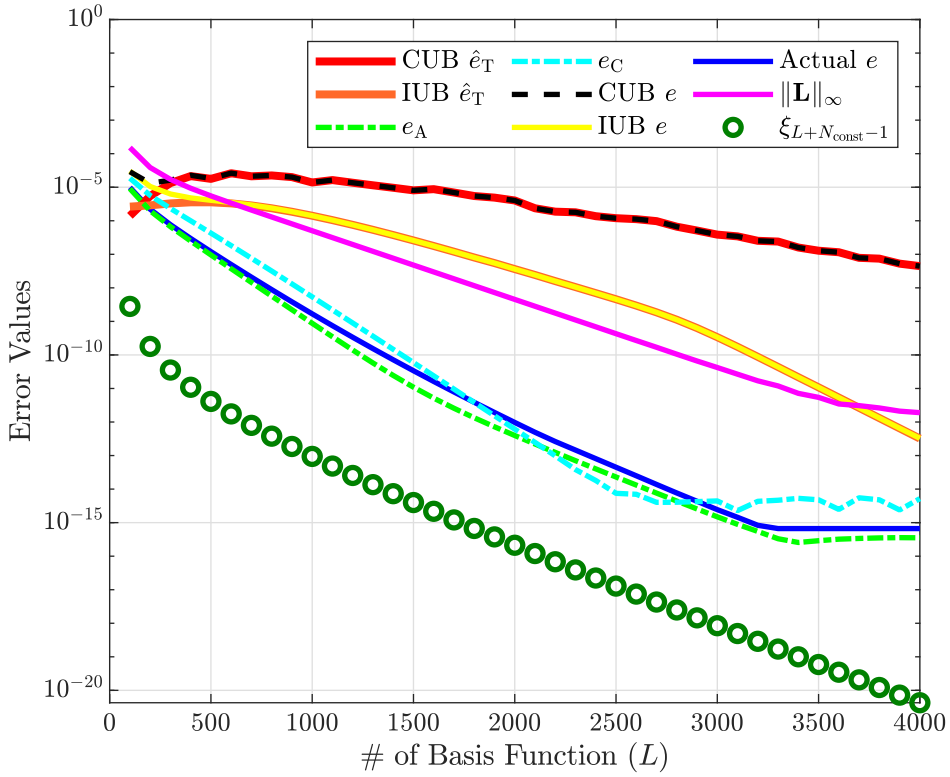
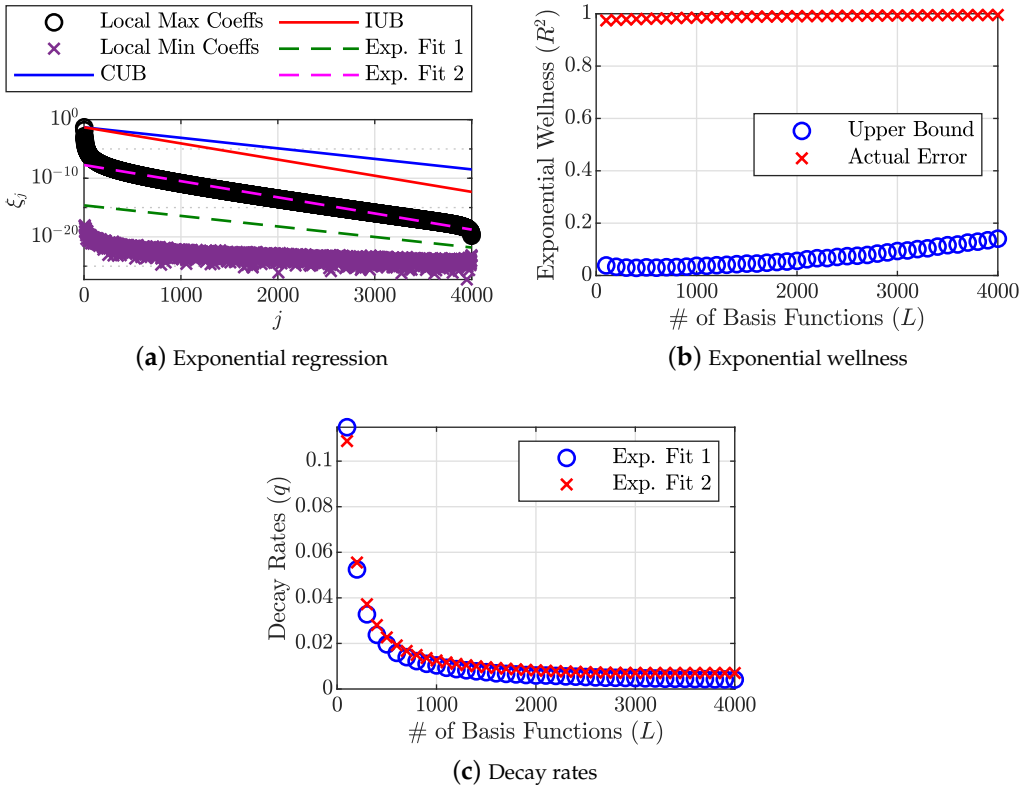


Figure 4. Error analysis of the TFC solution to Equation (26).

Since the local maximums of the ξ_j coefficients for approximating y_2 yield algebraic convergence, Rule of Thumb 2 does not apply (which can be seen in Figure 4). Also, the IUB in subplot (**a**) has a lot of room between it and the local maximum ξ_j coefficients as j increases. Hence, the translated exponential fit shown in Equation (20) is not the best representation for nonsmooth functions. Nonetheless, the translated exponential fit is still a sufficient upper bound, meaning the resulting IUB $\hat{e}_T$ is still valid. Other than the global polynomial not approximating the solution well, another reason why Rule of Thumb 1 is not valid in this case is that the IUB $\hat{e}_T$ is still a conservative upper bound, although not as conservative as the CUB $\hat{e}_T$. Therefore, Exp. Fit 2 is better than Exp. Fit 1 because it leads to the IUB $\hat{e}_T$. Figure 4 shows that the IUB $\hat{e}_T$ is closer to the L_∞-norm of the loss and, as such, correlates better with how the TFC method determines the accuracy of a solution.

An indication that the approximated function is not smooth, other than the local maximums of the ξ_j coefficients yielding algebraic convergence, is the decay rate of the exponential fits, as shown in subplot (**c**). The decay rate of the exponential fits as L increases are all very close to 0. Thus, since the solution to Equation (26) is very close to not being smooth (i.e., $\alpha = 1.5 \times 10^{-3}$), the decay rate of the ξ_j coefficients alone can be used to determine that the actual solution is not smooth, or very close to not being smooth. When this is the case, the segment of the domain the constrained expression approximates should be divided.



(a) Exponential regression

(b) Exponential wellness

(c) Decay rates

Figure 5. Three plots that aid in the analysis of the exponential fits performed on the truncated Chebyshev coefficients computed with TFC for the TPBVP shown in Equation (26). Subplot (**a**) shows Exp. Fit 1 for $L = 4000$, where all truncated Chebyshev coefficients are included in the regression, and Exp. Fit 2 for $L = 4000$, where only the local maximums of the truncated Chebyshev coefficients are included in the regression. The resulting conservative upper bound (CUB) and improved upper bound (IUB) when Exp. Fit 1 and Exp. Fit 2 are used, respectively, are also shown. Subplot (**b**,**c**) shows the goodness and decay rates of each fit for increasing values of L.

4. *hp*-Adaptive Mesh Refinement Algorithm

The proposed *hp*-adaptive algorithm for solving hypersensitive TPBVPs begins by constructing an initial mesh $M_{h=0}$ and formulating the unconstrained system of ODEs via the TFC. All meshes M_h are composed of mesh segments $S_k = [t_k, t_{k+1}]$ for all $k = (1, \ldots, K)$, the number of basis functions for each constrained expression on each segment ${}^{(h,k)}L_i$, and the number of collocation points on each segment of each mesh ${}^{(h,k)}N$. Note that the subscript h refers to an iteration of the *hp*-adaptive algorithm. Hence, each iteration of the algorithm has its own mesh. Furthermore, the number of basis functions does not have to be the same among the constrained expressions on S_k belonging to M_h. However, the number of collocation points for all constrained expressions on the same segment must be identical. Therefore, we express ${}^{(h,k)}N$ as

$$ {}^{(h,k)}N = \max_i \left({}^{(h,k)}L_i \right) + N^+,$$

where N^+ is a user-defined parameter.

With M_0 and the unconstrained system of ODEs formed via the TFC, an iteration of the *hp*-adaptive mesh refinement algorithm begins by solving the algebraic system of equations representing the unconstrained system of ODEs on M_0 with the Gauss–Newton method or linear least squares. Afterward, the truncated Chebyshev series terms/coefficients on the roundoff plateau or the local minimums of significant oscillation are first eliminated for the decay rate and bounded truncation error computations of the constrained expressions.

The reason for this was explained in Sections 3.3 and 3.4, and the method for accomplishing this is explained in Section 4.1. The estimated truncation error bound and decay rates, as described in Section 3, for each constrained expression in each interval can then be computed. If a k segment's maximum absolute loss on M_h, $\|^{(h,k)}\mathbb{L}\|_\infty$, is greater than some prescribed tolerance, ϵ_{MR}, then ${}^{(h+1,k)}L_i$ for all $i = (1, \ldots, n)$ are increased, decreased, or the mesh interval is divided into subintervals for the next mesh iteration (see Sections 4.2 and 4.3). The computational efficiency of subsequent mesh iterations can be improved by combining neighboring pairwise segments that are "super smooth" or by combining all neighboring segments that contain the minimum number of basis functions in their constrained expressions, $L_{\min}$ (see Section 4.4). With the next mesh interval defined as M_{h+1}, the process can be repeated until a maximum number of mesh iterations is reached, $h = H$, or $\|^{(h,k)}\mathbb{L}\|_\infty \leq \epsilon_{\text{MR}}$ for all $k = (1, \ldots, K)$. An outline of the process is given in Section 4.5.

4.1. Disregarding Unnecessary Truncated Chebyshev Series Terms for Bounded Truncation Error Computation

In Section 3.1, we showed that a bounded exponential least-squares approximation of the truncated Chebyshev coefficients is necessary to obtain an expression for the maximum bound of a constrained expression's truncation error. If the bounded exponential least-squares approximation of the Chebyshev coefficients is bad, then so too is the computation for the truncation error bound of the constrained expression. By "bad", we do not only mean that the bounds are not valid. Instead, we mean that the Chebyshev coefficients and actual truncation error are well below their respective bounds (i.e., too conservative). In the next subsection, it is mathematically shown that conservative bounds lead to overly large mesh sizes. Hence, conservative bounds are undesirable. Obviously, bounds that are not valid are also undesirable.

In Sections 3.3 and 3.4, it was shown that performing an exponential least-squares fit on all ξ_j coefficients can lead to an incorrect or highly conservative bound for ξ_j. However, it was also shown that performing an exponential least-squares fit without the ξ_j values that are on the roundoff plateau and without including the local minimums of significant oscillation leads to much better bounds. Representing the set of indices corresponding to the coefficients not on the roundoff plateau and not local minimums of significant oscillation as ς, the exponential least-squares fit equation for computing q and s can then be given as

$$\ln\left(|\xi_\varsigma|\right) = \ln(s) - q\,\varsigma.$$

To determine the j indices not included in ς, we refer to the coefficients on the roundoff plateau as those that are less than double machine epsilon. Furthermore, coefficients are believed to be a local minimum of significant oscillation when they are smaller than 3 orders of magnitude from their neighboring coefficients.

4.2. Increasing or Decreasing the Number of Basis Functions

Suppose $\|^{(h,k)}\mathbb{L}\|_\infty > \epsilon_{\text{MR}}$ and $q \geq \bar{q}$. Then, all constrained expressions in S_k are considered to be smooth. Every ${}^{(h+1,k)}L_i$ can then be increased to reduce $\|^{(h+1,k)}\mathbb{L}\|_\infty$. Let ${}^{(h,k)}\hat{e}_{\text{T}_i}$ denote the constrained expression's estimate for the upper truncation error bound over interval S_k, on mesh h, and for the i-th differentiation variable. Then, we have the relationship

$${}^{(h,k)}\hat{e}_{\text{T}_i} = \frac{\pi^{1/2}\, s\, \exp\left(-\delta - q\left({}^{(h,k)}L_i + {}^{(h,k)}N_{\text{const}_i} + 1\right)\right)}{\left(2\left(1 - \exp\left(-2\,q\right)\right)\right)^{1/2}}, \tag{27}$$

where ${}^{(h,k)}N_{\text{const}_i}$ is the number of constraints on the i-th differentiation variable in S_k belonging to M_h. On the next mesh, M_{h+1}, it is desired to achieve ${}^{(h+1,k)}\hat{e}_{\text{T}_i} = \varepsilon_{\text{T}} \approx 0$

because, as Section 3.3 shows, a small ${}^{(h+1,k)}\hat{e}_{\mathrm{T}_i}$ corresponds to a small $\|{}^{(h+1,k)}\mathcal{L}_i\|_\infty$. There is a specific value of ${}^{(h+1,k)}L_i$ that yields ${}^{(h+1,k)}\hat{e}_{\mathrm{T}_i} = \varepsilon_{\mathrm{T}}$,

$$\varepsilon_{\mathrm{T}} = \frac{\pi^{1/2}\, s\, \exp\left(-\delta - q\left({}^{(h+1,k)}L_i + {}^{(h+1,k)}N_{\mathrm{const}_i} + 1\right)\right)}{2^{1/2}\left(1 - \exp\left(-2\,q\right)\right)^{1/2}}. \tag{28}$$

Dividing Equations (27) and (28), recognizing that ${}^{(h,k)}N_{\mathrm{const}_i} = {}^{(h+1,k)}N_{\mathrm{const}_i}$ when the intervals are not changed, and solving for ${}^{(h+1,k)}L_i$ gives

$${}^{(h+1,k)}L_i = \left\lceil {}^{(h,k)}L_i + \frac{1}{q}\ln\left(\frac{{}^{(h,k)}\hat{e}_{\mathrm{T}_i}}{\varepsilon_{\mathrm{T}}}\right)\right\rceil, \tag{29}$$

where q is calculated as shown in Section 4.1. Note that $\lceil\cdot\rceil$ rounds the $\cdot$ inside it up to the next highest integer, which is necessary because the number of basis functions has to be an integer that strictly increases for a lower ${}^{(h+1,k)}\hat{e}_{\mathrm{T}_i}$. In Equation (29), one can see that if ${}^{(h,k)}\hat{e}_{\mathrm{T}_i} < \varepsilon_{\mathrm{T}}$, then the number of basis functions decreases (i.e., ${}^{(h+1,k)}L_i < {}^{(h,k)}L_i$). This is ideal because it allows the number of basis functions for specific constrained expressions in a specific segment to decrease and still have $\|{}^{(h+1,k)}\mathbb{L}\|_\infty \leq \epsilon_{\mathrm{MR}}$. Allowing smoother constrained expressions to decrease their ${}^{(h,k)}L_i$ values reduces computational complexity when obtaining the Jacobians and optimizing for the Chebyshev coefficients. Lastly, note that ε_{T} is a user-defined value but should be as low as possible.

4.3. Dividing a Mesh Interval

Let $\|{}^{(h,k)}\mathbb{L}\|_\infty > \epsilon_{\mathrm{MR}}$ and $q < \bar{q}$. Then, computing ${}^{(h+1,k)}L_i$ with Equation (29) will be very large for some small ε_{T}. Large ${}^{(h+1,k)}L_i$ values cause large inversion matrices in the Gauss–Newton and linear least-squares algorithms, which raises the condition number. This can cause both algorithms to become numerically unstable and fail. Instead of increasing the number of basis functions in a mesh interval, it can instead be divided. The number of subintervals, V_k, into which S_k is divided should equal the predicted number of basis functions in the constrained expression for the next mesh in Equation (29) using $\bar{q}$,

$${}^{(h,k)}\bar{L}_i = {}^{(h,k)}L_i + \frac{1}{\bar{q}}\ln\left(\frac{{}^{(h,k)}\hat{e}_{\mathrm{T}_i}}{\varepsilon_{\mathrm{T}}}\right).$$

Thus, we have

$$V_k = \max_i\left(\left\lceil\frac{{}^{(h,k)}\bar{L}_i}{{}^{(h,k)}L_i}\right\rceil\right).$$

The locations of the knot points for all V_k intervals within S_k are equidistant.

4.4. Combining Mesh Intervals

The prescribed decay rate $\bar{q}$ indicates whether a function is smooth ($q \geq \bar{q}$) or nonsmooth ($q < \bar{q}$). Here, a new prescribed decay rate $\hat{q} > \bar{q}$ is introduced to determine whether a function is super-smooth ($q \geq \hat{q}$). If two and only two adjacent mesh intervals have $q > \hat{q}$ in all of their constrained expressions, then the intervals are combined. The number of basis functions on the new interval is equal to the sum of the basis functions in the intervals being combined. Only two neighboring mesh intervals are combined in a mesh iteration at a time to reduce the chance that multiple super-smooth segments are combined into one nonsmooth segment. Furthermore, neighboring pairwise super-smooth segments can only be combined in a mesh iteration if they were not divided in the same iteration.

Another way in which segments are combined is if adjacent segments have

$$\max_i\left({}^{(h,k)}L_i\right) = L_{\min}, \tag{30}$$

which means every constrained expression in a segment has the minimum number of basis functions. Hypersensitive TPBVPs have a solution that typically consists of constant values for long periods of time, followed or proceeded by sharp changes in its gradient. When the equality in Equation (30) holds true, the solution on the k segment is often constant. Thus, all neighboring segments where Equation (30) holds true are combined. We do not have to worry about combining a collection of super-smooth segments into a nonsmooth segment because the solution on the combined segments consists of constant values. Segments can only be combined if they were not divided in the same mesh iteration as well.

4.5. hp-Adaptive Mesh Refinement Algorithm Outline

For the reader's convenience, a brief step-by-step procedure of the *hp*-adaptive mesh refinement algorithm is given as follows:

Step 1: Set $h = 0$ and provide an initial mesh M_0 that is composed of initial segments ${}^{(0,k)}S$ and an initial number of basis functions for each constrained expression within each segment, ${}^{(0,k)}L_i$. Furthermore, make sure to set the hyperparameters, which are listed in Table 1.

Step 2: Formulate the unconstrained system of ODEs from the TPBVP by building the constrained expressions for the differentiation variables on M_h.

Step 3: Solve the unconstrained system of ODEs on mesh M_h, as described in Section 2.

Step 4: If $\|{}^{(h,k)}\mathbb{L}\|_\infty \leq \epsilon_{\text{MR}}$ for all $k = (1, \ldots, K)$, then the TPBVP is solved with the user's desired accuracy, and the mesh refinement is complete. If $h = H$, then the mesh refinement is also complete, even though the TPBVP may not be solved at the desired accuracy. Otherwise, for every S_k where $k = (1, \ldots, K)$ on M_h:

- Proceed to the next segment if $\|{}^{(h,k)}\mathbb{L}\|_\infty \leq \epsilon_{\text{MR}}$.
- Calculate q_i and $\hat{e}_{T_i}$ for every $i = 1, \ldots, n$ in the segment, as shown in Section 3.1. Make sure to disregard the terms from the Chebyshev polynomial within the constrained expressions that contain negligible coefficients on the roundoff plateau or are a local minimum of significant oscillation, as described in Section 4.1.
- If $\max_i(q_i) > \bar{q}$, modify the segment for the next mesh iteration by either increasing/decreasing the number of basis functions in any of the segment's constrained expressions (Section 4.2) or by dividing the segment (Section 4.3).

Step 5: For neighboring segments that were not divided for the next mesh iteration, combine them for the next mesh iteration, as shown in Section 4.4).

Step 6: Set $h = h + 1$ and return to Step 2.

Table 1. Hyperparameters.

Symbol	Value	Description
ϵ_{GN}	1×10^{-15}	Gauss–Newton error tolerance.
ϵ_{MR}	1×10^{-13}	Mesh refinement error tolerance for an acceptable solution.
ε_{T}	1×10^{-15}	Desired truncation error for a constrained expression.
$p_{\max}$	40	Maximum number of Gauss–Newton iterations.
H	20	Maximum number of mesh refinement iterations.
N^+	3	Number of collocation points more than the number of basis functions.
$\bar{q}$	0.8	Cutoff decay rate that determines whether a function is smooth or not.
$\grave{q}$	1.2	Cutoff decay rate that determines whether a function is super-smooth or just smooth
$L_{\min}$	3	Minimum number of basis functions in a constrained expression.

5. Results

We solved three hypersensitive TPBVPs with our proposed TFC *hp*-adaptive mesh refinement algorithm. Each TPBVP was derived by applying PMP to its respective OCP.

A brief procedure for how to use PMP to formulate TPBVPs from OCPs is given in Appendix B. All programming was performed in MATLAB® R2021b and run on an Intel Core i7-9700 CPU PC with 64 GB of RAM. The hyperparameters used for the algorithm are given in Table 1, except that ϵ_{MR} equaled 1^{-15} for the first test problem.

5.1. Linear Hypersensitive Problem

The first OCP we attempted to solve was

$$\begin{aligned} \min_{u} \quad & J = \frac{1}{2}\int_{t_0}^{t_f}\left(x^2+u^2\right)\mathrm{d}t \\ \text{s.t.} \quad & \dot{x} = -x+u \end{aligned}$$

with the boundary constraints

$$\begin{cases} x(t_0) = 1.5 \\ x(t_f) = 1 \end{cases}. \tag{31}$$

The initial and final times were $t_0 = 0$ and t_f = 10,000, respectively. After applying PMP, the dual TPBVP ODEs were

$$\begin{cases} \dot{x} = -x-\lambda \\ \dot{\lambda} = -x+\lambda \end{cases}, \tag{32}$$

where λ is the costate variable that can be used to compute the control,

$$u = -\lambda.$$

As its name suggests, the TPBVP is subject to the same boundary conditions as the OCP (Equation (31)). The main reason we decided to solve this hypersensitive TPBVP was because its linearity allows a true analytical solution to be found quite easily:

$$x^*(t) = c_1\,\exp\left(t/\sqrt{2}\right) + c_2\,\exp\left(-t/\sqrt{2}\right)$$

and

$$u^*(t) = \dot{x}^*(t) + x^*(t)$$

with

$$c_1 = \frac{1}{\exp\left(-t_f/\sqrt{2}\right) - \exp\left(t_f/\sqrt{2}\right)}\left(1.5\,\exp\left(-t_f/\sqrt{2}\right) - 1\right)$$

and

$$c_2 = \frac{1}{\exp\left(-t_f/\sqrt{2}\right) - \exp\left(t_f/\sqrt{2}\right)}\left(1 - 1.5\,\exp\left(t_f/\sqrt{2}\right)\right).$$

To quantify the benefits of using our TFC-based *hp*-adaptive mesh refinement algorithm, we first solved this problem with the standard TFC procedure that uses global constrained expressions to approximate x and u with a hard-coded number of basis functions and collocation points. Multiple solutions were calculated, each having the same number of basis functions in the constrained expressions approximating x and u, but increased by one for each run. The number of collocation points for each solution was fixed at 1050. Figure 6 shows the L_∞-norm of the loss and actual error for the solutions. As can be seen, the TFC with global constrained expressions was very capable of solving this problem, but around 600 basis functions were required for the minimum loss and minimum actual error to be reached. With the *hp*-adaptive mesh refinement algorithm, this problem should be able to be solved with a much fewer number of total basis functions, and even with better accuracy.

The TFC-based *hp*-adaptive mesh refinement algorithm began with an initial mesh of only two knot points: one at t_0 and another at t_f, i.e., global constrained expressions.

The number of basis functions in the x and λ constrained expressions on the initial mesh was set to 50. The algorithm did not find a solution that satisfied $\|\mathbb{L}\|_\infty \leq \epsilon_{\text{MR}} = 1 \times 10^{-15}$, meaning it ran for the maximum number of iterations, but the solution it did find was still very accurate. Figure 7 shows the final solutions for x and u, each overlapping with the truth. The subplots in Figure 8 show how the loss decreased and the mesh knots changed over the mesh iterations. The final mesh consisted of five segments, i.e., six knots at $t_0 = 0$, $t_1 = 19.5$, $t_2 = 29.3$, $t_3 = 9970.7$, $t_4 = 9980.5$, and $t_5 = t_f$ = 10,000. The $\|\mathbb{L}\|_\infty$ of the final solution was 9.3×10^{-15}. Summing the number of basis functions for all segments of the constrained expressions for x and λ yielded 100 and 103, respectively.

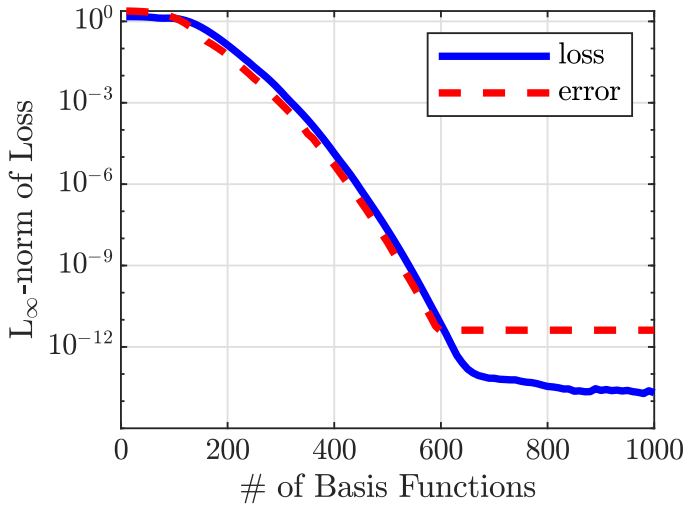


Figure 6. L_∞-norm curves for the error and loss of the TFC's solution to the TPBVP represented by Equations (31) and (32) when global constrained expressions are used as the number of basis functions increases.

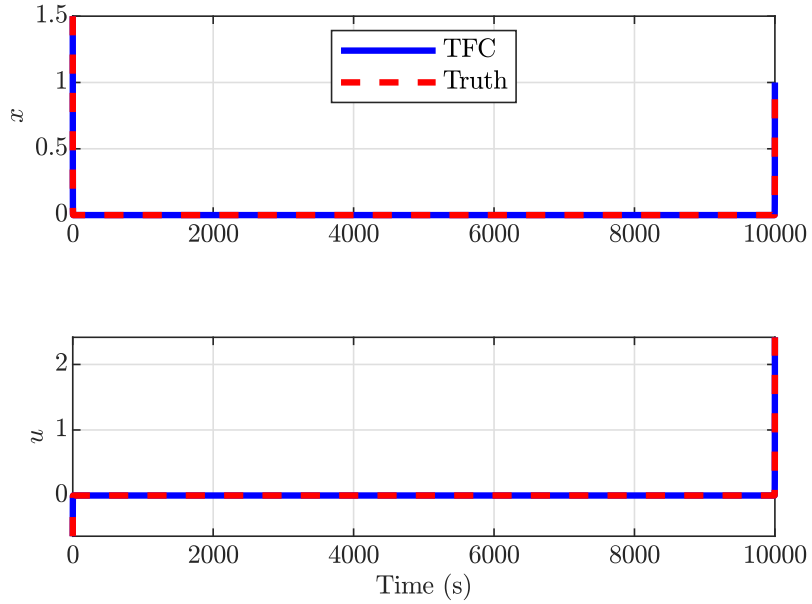
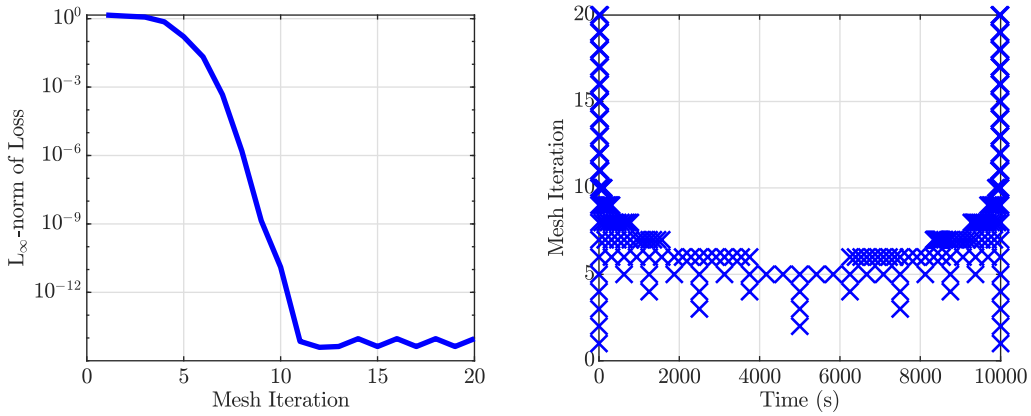


Figure 7. Solutions to the linear hypersensitive problem.



(a) Loss per iteration **(b)** Mesh knot points

Figure 8. Evolution of the mesh by the proposed *hp*-adaptive mesh refinement algorithm for solving the linear hypersensitive problem. Subplot (**a**) shows the L_∞-norm of the loss for each mesh iteration. Subplot (**b**) shows the knot points for each mesh iteration.

5.2. Nonlinear Hypersensitive Problem

The nonlinear hypersensitive OCP we attempted to solve was

$$\begin{aligned} \min_{u} \quad & J = \frac{1}{2}\int_{t_0}^{t_f}\left(x^2+u^2\right)\mathrm{d}t \\ \text{s.t.} \quad & \dot{x} = -x^3+u \end{aligned}$$

with the boundary constraints

$$\begin{cases} x(t_0) = 1.5 \\ x(t_f) = 1 \end{cases}. \tag{33}$$

As before, the initial and final times were $t_0 = 0$ and t_f = 10,000. After applying PMP, the dual TPBVP ODEs were

$$\begin{cases} \dot{x} = -x^3 - \lambda \\ \dot{\lambda} = -x + 3\,x^2\,\lambda \end{cases}, \tag{34}$$

where the control is related to the costate by

$$u = -\lambda.$$

The TPBVP given by Equations (33) and (34) does not have a true analytical solution due to its nonlinearity. Due to this nonlinearity, an initial guess needed to be provided for the Gauss–Newton step of the TFC procedure. The initial guess we provided for x and λ was simply zero over the entire time domain. Unlike for the linear hypersensitive problem, here, we do not provide a plot showing how $\|\mathbb{L}\|_\infty$ decreases when the number of basis functions is increased and global constrained expressions are used because $\|\mathbb{L}\|_\infty$ was greater than 1×10^{-4} when 1000 basis functions were used.

Similar to the approach used to solve the previous problem, the TFC-based *hp*-adaptive mesh refinement algorithm began with an initial mesh of only one segment and with the number of basis functions in the x and λ constrained expressions set to 50. The algorithm ran for the maximum number of mesh iterations, i.e., $h = H = 20$. Thus, $\|\mathbb{L}\|_\infty \leq \epsilon_{\text{MR}} = 1 \times 10^{-13}$ was not satisfied by the final loss, even though it was still exceptionally low. Table 2 gives the L_∞-norm of the loss for the TFC solution at the final mesh iteration (TFC Split), the TFC when only global constrained expressions were used with $L = 300$ (TFC Global), and MATLAB's `bvp4c` routine. Note that the `bvp4c` routine is a finite-difference implementation of the three-stage Lobatto IIIa formula [35,36] and can be regarded as an *h* method. The same initial guess was used for each mesh. Figure 9

shows the final solutions for x and u near the boundaries of the time domain. The proposed mesh refinement algorithm's and `bvp4c` curves overlap, but it is clear that the proposed approach's solutions are better based on its L_∞-norm of the loss being nine orders of magnitude lower. The subplots in Figure 10 show how the loss decreased in the proposed algorithm over its mesh iterations and how the mesh knots changed. The final mesh consisted of nine segments, one long segment in the center of the time domain, and several much smaller segments clustered near the boundaries to handle the gradients.

Table 2. L_∞-norm of the loss for different methods for the nonlinear hypersensitive problem.

TFC Split	TFC Global	`bvp4c`
1.285×10^{-13}	4.969×10^{-1}	4.128×10^{-4}

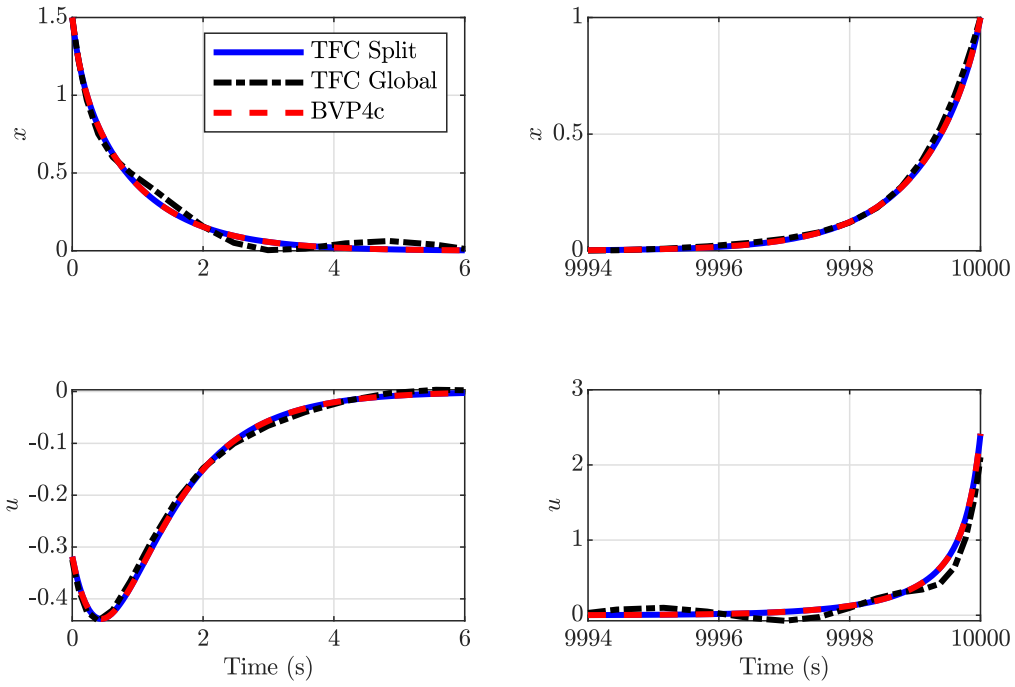
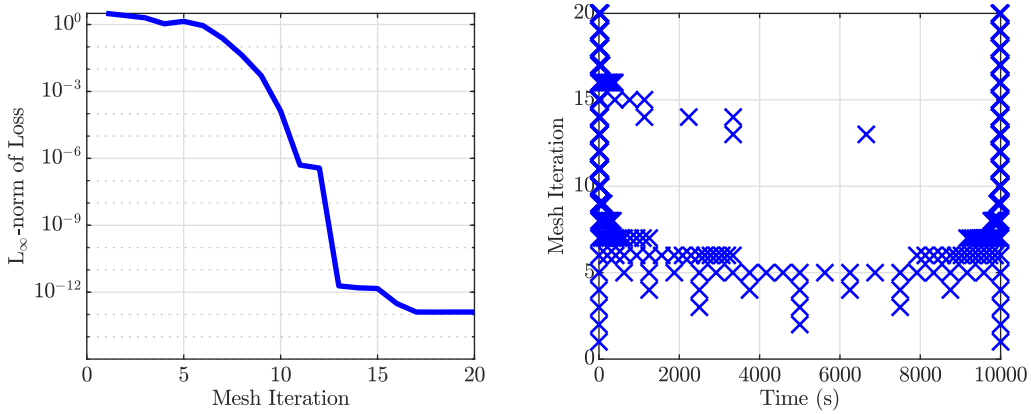


Figure 9. Solutions to the nonlinear hypersensitive problem.



(**a**) Loss per iteration (**b**) Mesh knot points

Figure 10. Evolution of the mesh by the proposed *hp*-adaptive mesh refinement algorithm for solving the nonlinear hypersensitive problem. Subplot (**a**) shows the L_∞-norm of the loss for each mesh iteration. Subplot (**b**) shows the knot points for each mesh iteration.

5.3. Mass Spring Problem

The last problem solved was a common mass spring system OCP,

$$\begin{aligned} \min_{u} \quad & J = \frac{1}{2}\int_{t_0}^{t_f}\left(x^2+u^2\right)\mathrm{d}t \\ \text{s.t.} \quad & \dot{x}_1 = x_2 \\ & \dot{x}_2 = -x_1 - x_1^3 - u - 1, \end{aligned}$$

with the boundary constraints

$$\begin{cases} x_1(t_0) = 1 \\ x_2(t_0) = 0.75 \\ x_1(t_f) = 0 \\ x_2(t_f) = 0 \end{cases}. \tag{35}$$

The initial and final times were $t_0 = 0$ and t_f = 10,000, respectively. After applying PMP, the dual TPBVP ODEs were

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -x_1 - x_1^3 - u - 1 \\ \dot{\lambda}_1 = -x_1 + \lambda_2\left(1 + 3\,x_1^2\right) \\ \dot{\lambda}_2 = -x_2 - \lambda_1 \end{cases}, \tag{36}$$

where the control is related to the costate by

$$u = -\lambda.$$

Like the previous TPBVP solved, the TPBVP given by Equations (35) and (36) does not have a true analytical solution due to its nonlinearity. The initial mesh and initial guess of the solution were identical to what was used to solve the nonlinear hypersensitive problem. Table 3 shows the L_∞-norm of the loss for the proposed approach and the other approaches shown in Table 2. Once again, the proposed approach performed the best. Figure 11 shows the trajectories of the various solutions. The subplots in Figure 12 show how the loss decreased in the proposed algorithm over its mesh iterations and portray how the mesh knots changed. The final mesh consisted of 20 segments, and most of them were near the boundary, while one long segment was used to approximate the constant part of the solution, as shown in Figure 12b.

Table 3. L_∞-norm of the loss for different methods for the mass spring problem.

TFC Split	TFC Global	`bvp4c`
2.345×10^{-12}	5.455×10^{-1}	9.878×10^{-4}

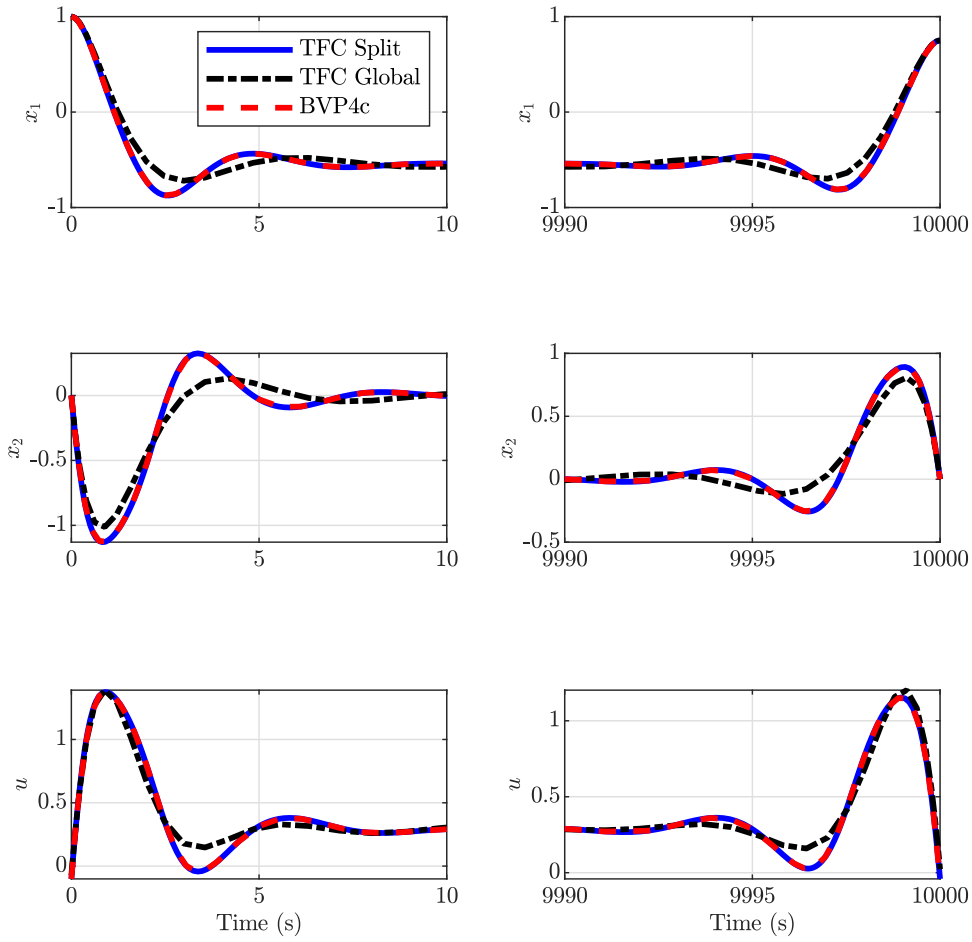
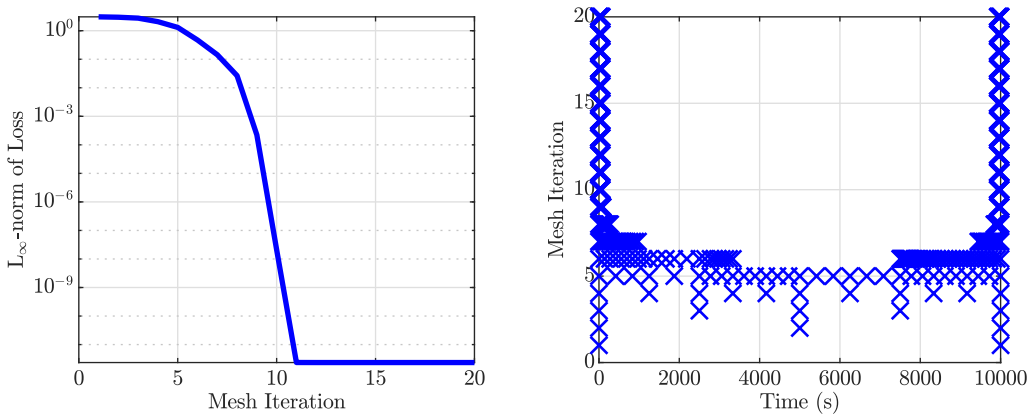


Figure 11. Solutions to the mass spring problem.



(a) Loss per iteration **(b)** Mesh knot points

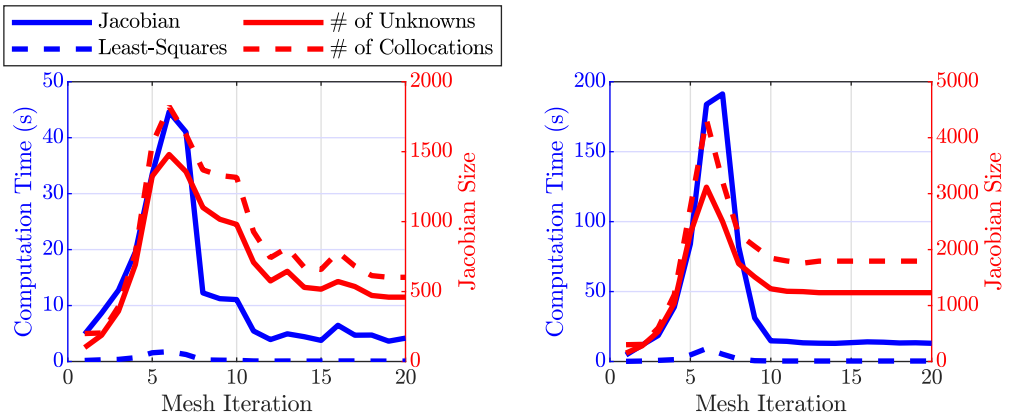
Figure 12. Evolution of the mesh by the proposed *hp*-adaptive mesh refinement algorithm for solving the mass spring problem. Subplot (**a**) shows the L_∞-norm of the loss for each mesh iteration. Subplot (**b**) shows the knot points for each mesh iteration.

6. A Discussion on Computation Time

The main drawback of the proposed algorithm is its computation time when the problem is nonlinear. Figure 13 shows how the size of the mesh affects the most computationally

intensive aspects of the proposed algorithm for the nonlinear hypersensitive problem (**a**) and mass spring problem (**b**). The steps that took the longest to complete were the numerical approximations of the Jacobian and the least-square computations performed within the Gauss–Newton algorithm. Hence, Figure 13 shows the total accumulated runtime for performing these calculations at each mesh iteration. The size of the Jacobian is related to the number of basis functions (columns) and the number of collocations (rows) being solved. As can clearly be seen, a larger mesh correlates to a longer runtime. The reason why all accumulated least-squares computations were so small was due to MATLAB's efficient storage of sparse matrices. The numerical approximation of the Jacobian was time-consuming because it required a number of computations equal to the number of columns in the Jacobian, i.e., the derivatives of the loss with respect to all Chebyshev coefficients in all constrained expressions in all segments were needed. Hence, the proposed algorithm has a long runtime because of the need to numerically approximate the Jacobian at each iteration of the Gauss–Newton algorithm within each mesh iteration.

The best way to reduce the time needed to numerically compute the Jacobian would be to break apart the partial derivatives of the loss with respect to the unknown coefficients into two parts using the chain rule: the partial derivatives of the loss with respect to the differentiation variables and the partial derivatives of the constrained expressions with respect to the unknown coefficients. The partial derivatives of the constrained expressions with respect to the unknown coefficients could be computed by hand a priori. Thus, only the partial derivatives of the loss with respect to the differentiation variables would need to be computed numerically, which would drastically reduce the number of loss evaluations needed. Patterson and Rao [37] demonstrated a similar approach for the pseudospectral method when Lagrange polynomials are used to approximate the differentiation variables. We hope to do the same but for the TFC's constrained expressions in future work.



(**a**) Nonlinear hypersensitive problem (**b**) Mass spring problem

Figure 13. The computation time of a mesh iteration versus the size of the Jacobian pertaining to the same mesh iteration for the nonlinear hypersensitive problem (**a**) and the mass spring problem (**b**).

7. Conclusions

An *hp*-adaptive mesh refinement algorithm has been developed for the TFC method that utilizes Chebyshev polynomial-free functions. The algorithm relies on the correlation between the L_∞-norm of the loss and the truncation error to determine the ideal number of basis functions in a constrained expression. Whether to add basis functions to constrained expressions within a segment or divide the segment is determined by the decay rates of the Chebyshev coefficients in the segment's constrained expressions. Smooth segments have larger decay rates, while nonsmooth segments have small decay rates. The error is decreased by adding basis functions to the smooth segments and dividing the segments that are not smooth. To improve computational performance, the number of basis functions is decreased in constrained expressions on smooth segments that already satisfy the L_∞-norm

of the loss tolerance well. Lastly, segments that are very smooth are combined to further improve computational performance.

The only problems solved in this work were TPBVPs generated by performing PMP on hypersensitive OCPs without inequality constraints. All problems were solved using the proposed approach with a maximum loss on the order of 1×10^{-12} or better. In contrast, MATLAB's `bvp4c` solved each nonlinear problem with a maximum loss on the order of 1×10^{-4}. Hypersensitive OCPs with inequality constraints will be solved in future work, which will require the continuation of regularization or smoothing parameters. Since the continuation process will be iterative, increased computational performance will be required. As the algorithm is currently constructed, it is too slow to be coupled with continuation due to the cost of numerically approximating the Jacobian. Hence, a major challenge of future work will be discovering new ways to approximate the Jacobian with fewer calculations, similar to what has already been accomplished with other collocation procedures (e.g., the pseudospectral method for optimal control).

Author Contributions: Conceptualization, K.D.; methodology, K.D.; software, K.D.; validation, K.D.; formal analysis, K.D.; investigation, K.D.; resources, K.D. and R.F.; writing—original draft preparation, K.D. and A.D.; writing—review and editing, K.D., A.D. and R.F.; visualization, K.D. and A.D.; supervision, R.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data will be made available by the authors on request.

Acknowledgments: The authors would like to thank Enrico Schiassi and Mario De Florio for sharing problems to test the proposed algorithm with at the beginning stages of this research.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Constraint Vector Generalization

The constraint vector for the end segments (i.e., S_1 and S_K) depends on the boundary conditions of the TPBVP being solved. For example, suppose we are solving the following TPBVP:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = 2x_1 \\ \dot{\lambda}_1 = 3\lambda_2 \\ \dot{\lambda}_2 = x_1 + x_2 + 3\lambda_2 - \lambda_1 \end{cases} \quad \text{subject to:} \begin{cases} x_1(t_0) = x_{1_0} \\ x_2(t_0) = x_{2_0} \\ x_1(t_f) = x_{1_f} \\ \lambda_2(t_f) = 0 \end{cases} . \tag{A1}$$

The constraint vectors for the initial and final segments when solving the TPBVP are

$${}^{(1)}\boldsymbol{B} = \begin{Bmatrix} x_1(t_0) \\ x_1(t_1) \\ \frac{\mathrm{d}x_1(t_1)}{\mathrm{d}t} \\ x_2(t_0) \\ x_2(t_1) \\ \frac{\mathrm{d}x_2(t_1)}{\mathrm{d}t} \\ \lambda_1(t_1) \\ \frac{\mathrm{d}\lambda_1(t_1)}{\mathrm{d}t} \\ \lambda_2(t_1) \\ \frac{\mathrm{d}\lambda_2(t_1)}{\mathrm{d}t} \end{Bmatrix} \quad \text{and} \quad {}^{(K)}\boldsymbol{B} = \begin{Bmatrix} x_1(t_{K-1}) \\ x_1(t_f) \\ \frac{\mathrm{d}x_1(t_{K-1})}{\mathrm{d}t} \\ x_2(t_{K-1}) \\ \frac{\mathrm{d}x_2(t_{K-1})}{\mathrm{d}t} \\ \lambda_1(t_{K-1}) \\ \frac{\mathrm{d}\lambda_1(t_{K-1})}{\mathrm{d}t} \\ \lambda_2(t_{K-1}) \\ \lambda_2(t_f) \\ \frac{\mathrm{d}\lambda_2(t_{K-1})}{\mathrm{d}t} \end{Bmatrix},$$

respectively. The continuity conditions in the first and last segments, ${}^{(1)}\hat{\boldsymbol{B}}$ and ${}^{(K)}\hat{\boldsymbol{B}}$, are identical to ${}^{(1)}\boldsymbol{B}$ and ${}^{(K)}\boldsymbol{B}$ without $x_1(t_0)$, $x_2(t_0)$, $x_1(t_f)$, and $\lambda_2(t_f)$. Unlike the constraint

vectors for the end segments, the constraint vectors for the interior segments are not unique. Each differentiation variable has four constraints, two on each end of the segment, to enforce continuity. Hence, the constraint vectors for the interior segments when solving Equation (A1) are

$$ {}^{(k)}\boldsymbol{B} = \begin{Bmatrix} x_1(t_{k-1}) \\ x_1(t_k) \\ \frac{\mathrm{d}x_1(t_{k-1})}{\mathrm{d}t} \\ \frac{\mathrm{d}x_1(t_k)}{\mathrm{d}t} \\ x_2(t_{k-1}) \\ x_2(t_k) \\ \frac{\mathrm{d}x_2(t_{k-1})}{\mathrm{d}t} \\ \frac{\mathrm{d}x_2(t_k)}{\mathrm{d}t} \\ \lambda_1(t_{k-1}) \\ \lambda_1(t_k) \\ \frac{\mathrm{d}\lambda_1(t_{k-1})}{\mathrm{d}t} \\ \frac{\mathrm{d}\lambda_1(t_k)}{\mathrm{d}t} \\ \lambda_2(t_{k-1}) \\ \lambda_2(t_k) \\ \frac{\mathrm{d}\lambda_2(t_{k-1})}{\mathrm{d}t} \\ \frac{\mathrm{d}\lambda_2(t_k)}{\mathrm{d}t} \end{Bmatrix}, \quad \text{for } k = (2, \dots K-1). $$

Since the constraint vectors in the interior segments are made up of only continuity conditions, we have ${}^{(k)}\boldsymbol{B} = {}^{(k)}\hat{\boldsymbol{B}}$ when $k = (2, \dots K-1)$.

Appendix B. Deriving a Dual Two-Point Boundary-Value Problem from an Optimal Control Problem

The goal of an optimal control problem is to minimize a cost functional $J(\boldsymbol{x}(t), \boldsymbol{u}(t), t_f)$ through the selection of the state vector $\boldsymbol{x}(t) \in \mathbb{R}^n$ and control vector $\boldsymbol{u}(t) \in \mathbb{R}^m$, along with the initial time t_0 and the final time t_f when both are free. The general form of a fixed final time OCP can be given as

$$ \min_{\boldsymbol{x}, \boldsymbol{u}, t_0, t_f} \quad J(t_0, t_f, \boldsymbol{x}(t), \boldsymbol{u}(t)) = \Phi(t_0, t_f, \boldsymbol{x}(t_0), \boldsymbol{x}(t_f)) + \int_{t_0}^{t_f} L(t, \boldsymbol{x}(t), \boldsymbol{u}(t))\, \mathrm{d}t \tag{A2a} $$

$$ \text{s.t.} \quad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t)\boldsymbol{u}(t)) \tag{A2b} $$

$$ \boldsymbol{u}(t) \in U \tag{A2c} $$

$$ \boldsymbol{\phi}(t_0, \boldsymbol{x}(t_0), t_f, \boldsymbol{x}(t_f)) = 0 \tag{A2d} $$

where J is made up of a scalar penalty term $\Phi(t_0, t_f, \boldsymbol{x}(t_0), \boldsymbol{x}(t_f))$, called the Meyer cost, and a scalar integral term, called the running or Lagrangian cost, with the integrand $L(t, \boldsymbol{x}(t), \boldsymbol{u}(t))$. Furthermore, the problem is subject to equations of motion $\boldsymbol{f}(t, \boldsymbol{x}(t)\boldsymbol{u}(t))$, an admissible set of controls U, and boundary conditions $\boldsymbol{\phi}(t_0, \boldsymbol{x}(t_0), t_f, \boldsymbol{x}(t_f))$. Note that inequality constraints on the state can exist but were not considered in the above OCP. The same is true for the control. Hence, the admissible control and its derivative are continuous, $U \in C^1\Big([t_0, t_f]\Big)$.

Following the indirect method, the OCP can be transformed into a TPBVP whose solution is the solution to the OCP. Formulating the TPBVP first requires the Hamiltonian equation $H(t, \boldsymbol{x}(t), \boldsymbol{u}(t))$,

$$ H(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) = L(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) + \boldsymbol{\lambda}^{\intercal}(t)\, f(t, \boldsymbol{x}(t), \boldsymbol{u}(t)), $$

where $\boldsymbol{\lambda}(t) \in \mathbb{R}^n$ is the costate vector. Minimizing the Hamiltonian with respect to the control, $\frac{\partial H}{\partial \boldsymbol{u}} = 0$, allows an equation for the optimal control with respect to the costate to be derived, $\boldsymbol{u}^*(t, \boldsymbol{\lambda}(t))$. When $\boldsymbol{u}$ does not appear linearly in the Hamiltonian, and it is not transcendental, substituting it with $\boldsymbol{u}^*$ in the Hamiltonian allows it to be written in terms of the costates instead of the controls, $H(t, \boldsymbol{x}(t), \boldsymbol{u}^*(t), \lambda(t)) = H(t, \boldsymbol{x}(t), \lambda(t))$. When $\boldsymbol{u}$ does appear linearly in the Hamiltonian, the equation for $\boldsymbol{u}^*$ must be added to the final TPBVP. By Pontryagin's minimum principle, the first-order necessary conditions that minimize the Hamiltonian and solve the OCP are given by

$$\dot{\boldsymbol{x}} = \frac{\partial H}{\partial \boldsymbol{\lambda}}. \tag{A3}$$

and

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{x}} \tag{A4}$$

The states are still constrained by the boundary condition shown in Equation (A2d). Likewise, the costates have boundary conditions given by

$$\boldsymbol{\lambda}(t_0) = -\frac{\partial \Phi}{\partial \boldsymbol{x}(t_0)} + \boldsymbol{v}^\intercal \frac{\partial \phi}{\partial \boldsymbol{x}(t_0)} \tag{A5}$$

and

$$\boldsymbol{\lambda}(t_f) = \frac{\partial \Phi}{\partial \boldsymbol{x}(t_f)} + \boldsymbol{v}^\intercal \frac{\partial \phi}{\partial \boldsymbol{x}(t_f))}, \tag{A6}$$

where $\boldsymbol{v} \in \mathbb{R}^q$ is the Lagrange multiplier associated with the state boundary condition ϕ. If t_f is unspecified, then the transversality condition must also hold,

$$H(t_f, \boldsymbol{x}(t_f), \boldsymbol{\lambda}(t_f)) + \frac{\partial \Phi}{\partial t_f} = 0. \tag{A7}$$

If t_0 is unspecified, then the following transversality condition must also hold:

$$H(t_0, \boldsymbol{x}(t_0), \boldsymbol{\lambda}(t_0)) - \frac{\partial \Phi}{\partial t_0} = 0. \tag{A8}$$

Equations (A2d) and (A3)–(A8) then form the dual TPBVP.

References

1. Leake, C.; Johnston, H.; Daniele, M. *The Theory of Functional Connections: A Functional Interpolation Framework with Applications*; Lulu: Morrisville, NC, USA, 2022.
2. De Florio, M.; Schiassi, E.; Furfaro, R.; Ganapol, B.D.; Mostacci, D. Solutions of Chandrasekhar's basic problem in radiative transfer via theory of functional connections. *J. Quant. Spectrosc. Radiat. Transf.* **2021**, *259*, 107384. [CrossRef]
3. Yassopoulos, C.; Leake, C.; Reddy, J.; Mortari, D. Analysis of Timoshenko–Ehrenfest beam problems using the theory of functional connections. *Eng. Anal. Bound. Elem.* **2021**, *132*, 271–280. [CrossRef]
4. Yassopoulos, C.; Reddy, J.N.; Mortari, D. Analysis of nonlinear Timoshenko–Ehrenfest beam problems with von Kármán nonlinearity using the Theory of Functional Connections. *Math. Comput. Sim.* **2023**, *205*, 709–744. [CrossRef]
5. Schiassi, E.; D'Ambrosio, A.; Drozd, K.; Curti, F.; Furfaro, R. Physics-informed veural vetworks for optimal planar orbit transfers. *J. Spacecr. Rockets* **2022**, *59*, 834–849. [CrossRef]
6. De Almeida, A.K., Jr.; Johnston, H.; Leake, C.; Mortari, D. Fast 2-impulse non-Keplerian orbit Transfer using the theory of functional connections. *Eur. Phys. J. Plus* **2021**, *136*, 223. [CrossRef]
7. de Almeida Junior, A.K.; Prado, A.F.; Mortari, D. Using the theory of functional connections to create periodic orbits with a linear variable thrust. *New Astron.* **2023**, *104*, 102068. [CrossRef]
8. Mortari, D. The theory of connections: Connecting points. *Mathematics* **2017**, *5*, 57. [CrossRef]
9. Mortari, D.; Johnston, H.; Smith, L. High accuracy least-squares solutions of nonlinear differential equations. *J. Comput. Appl. Math.* **2019**, *352*, 293–307. [CrossRef]
10. Bertsekas, D. *Dynamic Programming and Optimal Control*, 4th ed.; Athena Scientific: Nashua, NH, USA, 2012; Volume 2.
11. Enright, P.J.; Conway, B.A. Discrete approximations to optimal trajectories using direct transcription and nonlinear programming. *J. Guid. Control Dyn.* **1992**, *15*, 994–1002. [CrossRef]

12. Kelly, M. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Rev.* **2017**, *59*, 849–904. [CrossRef]
13. Bryson, A.E.; Ho, Y.C. *Applied Optimal Control: Optimization, Estimation, and Control*, rev. printing ed.; Taylor & Francis Group, LLC: New York, NY, USA, 1975.
14. Longuski, J.M.; Guzmán, J.J.; Prussing, J.E. *Optimal Control with Aerospace Applications*; Springer: New York, NY, USA, 2014. [CrossRef]
15. Schiassi, E.; D'Ambrosio, A.; Furfaro, R. Bellman neural networks for the class of optimal control problems with integral quadratic cost. *IEEE TAI* **2022**, *5*, 1016–1025. [CrossRef]
16. Zhang, H.; Zhou, S.; Zhang, G. Shaping low-thrust multi-target visit trajectories via theory of functional connections. *Adv. Space Res.* **2023**, *72* 257–269. [CrossRef]
17. Drozd, K.; Furfaro, R.; Mortari, D. Rapidly Exploring Random Trees with Physics-Informed Neural Networks for Constrained Energy-Optimal Rendezvous Problems. *J. Astronaut. Sci.* **2024**, *71*, 361–382. [CrossRef]
18. Johnston, H.; Schiassi, E.; Furfaro, R.; Mortari, D. Fuel-efficient powered descent guidance on large planetary bodies via theory of functional connections. *J. Astronaut. Sci.* **2020**, *67*, 1521–1552. [CrossRef]
19. Drozd, K.; Furfaro, R.; Schiassi, E.; D'Ambrosio, A. Physics-informed neural networks and functional interpolation for solving the matrix differential riccati equation. *Mathematics* **2023**, *11*, 3635. [CrossRef]
20. Trefethen, L.N. Spectral methods in MATLAB. In *Optimal Control with Aerospace Applications;* SIAM: Philadelphia, PA, USA, 2014; pp. 29–39. [CrossRef]
21. Lu, P. Propellant-optimal powered descent guidance. *J. Guid. Control Dyn.* **2018**, *41*, 813–826. [CrossRef]
22. Johnston, H.; Mortari, D. Least-squares solutions of boundary-value problems in hybrid systems. *J. Comput. Appl. Math.* **2021**, *393*, 113524. [CrossRef]
23. Darby, C.L.; Hager, W.W.; Rao, A.V. An hp-adaptive pseudospectral method for solving optimal control problems. *Optim. Control Appl. Methods* **2011**, *32*, 476–502. [CrossRef]
24. Darby, C.L.; Hager, W.W.; Rao, A.V. Direct trajectory optimization using a variable low-order adaptive pseudospectral method. *J. Spacecr. Rockets* **2011**, *48*, 433–445. [CrossRef]
25. Patterson, M.A.; Hager, W.W.; Rao, A.V. A ph mesh refinement method for optimal control. *Optim. Control Appl. Methods* **2015**, *36*, 398–421. [CrossRef]
26. Liu, F.; Hager, W.W.; Rao, A.V. Adaptive mesh refinement method for optimal control using nonsmoothness detection and mesh size reduction. *J. Frankl. Inst.* **2015**, *352*, 4081–4106. [CrossRef]
27. Liu, F.; Hager, W.W.; Rao, A.V. Adaptive mesh refinement method for optimal control using decay rates of Legendre polynomial coefficients. *IEEE Trans. Control Syst. Technol.* **2017**, *26*, 1475–1483. [CrossRef]
28. Gui, W.; Babuška, I. The h, p, and h-p versions of the finite element method in 1 dimension. I. the error analysis of the p-version. *Numer. Math.* **1986**, *49*, 577–612. [CrossRef]
29. Gui, W.; Babuška, I. The h, p, and h-p versions of the finite element method in 1 dimension. II. The error analysis of the h- and h-p versions. *Numer. Math.* **1986**, *49*, 613–657. [CrossRef]
30. Gui, W.; Babuška, I. The h, p, and h-p versions of the finite element method in 1 dimension. III. The adaptive h-p version. *Numer. Math.* **1986**, *49*, 659–683. [CrossRef]
31. Pan, B.; Wang, Y.; Tian, S. A high-precision single shooting method for solving hypersensitive optimal control problems. *Math. Probl. Eng.* **2018**, *2018*, 7908378. [CrossRef]
32. Boyd, J.P. *Chebyshev and Fourier Spectral Methods*, 2nd ed.; Dover Publishing: New York, NY, USA, 2001; pp. 19–57.
33. Trefethen, L.N. Is Gauss quadrature better than Clenshaw–Curtis? *SIAM Rev.* **2008**, *50*, 67–87. [CrossRef]
34. Driscoll, T.A.; Hale, N.; Trefethen, L.N. *Chebfun Guide*; Pafnuty Publications: Oxford, UK, 2014. Available online: http://www.chebfun.org/docs/guide/ (accessed on 21 April 2024).
35. Kierzenka, J.; Shampine, L.F. A BVP solver based on residual control and the Maltab PSE. *ACM Trans. Math. Softw.* **2001**, *27*, 299–316. [CrossRef]
36. Shampine, L.F.; Kierzenka, J.; Reichelt, M.W. Solving boundary value problems for ordinary differential equations in MATLAB with bvp4c. *Tutor. Notes* **2000**, *2000*, 1–27.
37. Patterson, M.A.; Rao, A.V. Exploiting sparsity in direct collocation pseudospectral methods for solving optimal control problems. *J. Spacecr. Rockets* **2012**, *49*, 354–377. [CrossRef]