robotics

MDPI

Article

Fast Approximation of Over-Determined Second-Order Linear Boundary Value Problems by Cubic and Quintic Spline Collocation

Philipp Seiwald * and Daniel J. Rixen

Department of Mechanical Engineering, Chair of Applied Mechanics, Technical University of Munich, Boltzmannstraße 15, 85748 Garching, Germany; rixen@tum.de

* Correspondence: philipp.seiwald@tum.de

Received: 3 May 2020; Accepted: 23 June 2020; Published: 25 June 2020

Abstract: We present an efficient and generic algorithm for approximating second-order linear boundary value problems through spline collocation. In contrast to the majority of other approaches, our algorithm is designed for over-determined problems. These typically occur in control theory, where a system, e.g., a robot, should be transferred from a certain initial state to a desired target state while respecting characteristic system dynamics. Our method uses polynomials of maximum degree three/five as base functions and generates a cubic/quintic spline, which is $\mathcal{C}^2/\mathcal{C}^4$ continuous and satisfies the underlying ordinary differential equation at user-defined collocation sites. Moreover, the approximation is forced to fulfill an over-determined set of two-point boundary conditions, which are specified by the given control problem. The algorithm is suitable for time-critical applications, where accuracy only plays a secondary role. For consistent boundary conditions, we experimentally validate convergence towards the analytic solution, while for inconsistent boundary conditions our algorithm is still able to find a "reasonable" approximation. However, to avoid divergence, collocation sites have to be appropriately chosen. The proposed scheme is evaluated experimentally through comparison with the analytical solution of a simple test system. Furthermore, a fully documented C++ implementation with unit tests as example applications is provided.

Keywords: cubic; quintic; spline; collocation; second-order; over-determined; boundary value problem

1. Introduction

1.1. Literature Review

In almost every field of natural science and engineering we face differential equations, which are typically used for modeling dynamical systems. Especially in engineering, the more specific case of boundary value problems (BVP) is very prominent. In other words, one often searches for the behavior of the investigated system between some kind of fixed, i.e., known, spatial or temporal boundaries. One can try to derive the analytical solution to the problem, however for complex systems this can be difficult or even impossible. In such cases, it can be sufficient to approximate the solution for which a variety of techniques exists.

Methods based on finite differences approximate the derivatives by difference quotients to obtain a system of equations which depends only on the primal function. This allows a solution to be found by formulating a linear system of (algebraic) equations representing the transformed system at certain grid points. In contrast, shooting methods aim at the iterative solution of an equivalent initial value problem (IVP), which is typically easier to handle than the original BVP. While for single shooting

the IVP is evaluated over the complete time interval, multiple shooting considers a partitioned time domain. Another technique is given by the finite element approach, which is mainly used for partial differential equations (PDE) as they occur, for example, in structural-, thermo-, and fluid dynamics. Typically finite element methods are based on a weak formulation of the residual and on splitting the considered domain into elements on which the local base functions to approximate the solution are defined. Although designed for PDEs, they can be applied to the simpler case of ordinary differential equations (ODE), which are the focus of this contribution (see, for example, [1]). A variety of other techniques exist. However, these three groups appear to be used the most in the field of engineering.

In the following, we approximate the solution through spline collocation using piecewise polynomial trial functions, which is a well-known technique to solve BVPs, see, for example, [2–4]. Over the past decades, various algorithms emerged, which can be classified into two types. The first type, also known as smoothest spline collocation (or just spline collocation as in [5]), aims at matching the differential equation at one collocation site per spline segment, which is typically a knot or the mid-point of the segment, while simultaneously forcing the spline to have maximum smoothness, i.e., highest possible continuity [5]. The second type, in [5] called Gaussian collocation, removes the constraints on higher order continuity and instead uses more collocation sites, which are typically chosen to be the Gaussian points of each segment. This class, which is also called orthogonal spline collocation [6], originates from [3] and aims at maximizing the order of convergence. In order to also provide a competitive convergence for the first type of methods, special variants for quadratic and quintic spline collocation have been proposed in [7] and [8], respectively. Optimal methods for quadratic and cubic splines on non-uniform grids, i.e., for an inhomogeneous segmentation of the spline, have been presented in [5].

Note that in addition to general purpose codes, such as COLSYS (FORTRAN) for non-linear mixed-order systems of multi-point boundary value ODEs [9], highly specialized algorithms, which aim at obtaining the best possible approximation for certain use cases, have also been published. Recent examples are methods for integro-differential equations [10] or fractional differential equations [11], which occur in certain material models exhibiting memory effects. Along with ODEs, various kinds of (multi-variable) PDEs have been investigated. See [6] for a survey on corresponding Gaussian collocation methods. Note that, for PDEs, the time domain is typically discretized using finite differences, e.g., by the Crank–Nicholson approach as used in [6], or the second-order backward difference in [10], while spline collocation is used for approximating the spacial variables. Lastly, methods for differential algebraic equations (DAE) have also been developed, e.g., the COLSYS extension COLDAE [12] for semi-explicit DAEs of index 2 and fully implicit DAEs of index 1.

In our opinion, despite the variety of techniques, there is still a lack of simple methods prioritizing execution time over approximation quality, which is essential for time critical control applications. The aim of this contribution is to provide an algorithm satisfying these needs while focusing on the special case of second-order linear ODEs, which are very common for dynamic systems. Moreover, we focus on over-determined BVPs, i.e., where more boundary conditions (BC) are given than necessary. This may at first seem to be a restriction of our algorithm since it needs more information than other implementations; however, it allows us to also consider inconsistent BVPs, i.e., the case where no exact solution to the problem exists.

1.2. Motivation

It might appear strange to search for an approximation of something that actually does not exist. However, we face exactly this situation during walking pattern generation for our humanoid robot LOLA, cf. Figure 1 left. In particular, we use a simplified model of the robot's multi-body dynamics, cf. Figure 1 right, to plan the center of mass (CoM) motion over a certain time horizon. The planned CoM motion resembles the dynamics of the model, which can be formulated as second-order linear ODE, and is constrained to certain values on position-, velocity- and acceleration-level at the boundaries

of the planning horizon. This leads to an overdetermined BVP of the type investigated in this contribution. The BCs at the beginning reflect the current state of the robot, e.g., the standing pose in front of a platform, while the BCs at the end represent the target state, e.g., the standing pose on the other side of the obstacle, cf. Figure 2. For a seamless motion of the robot it is crucial to guarantee the satisfaction of the BCs, i.e., a perfect match of the boundary states. In contrast, it is sufficient to approximate the dynamics of the underlying ODE since it is derived from a simplified model which is an approximation in itself. This is the key idea behind the formulation of an over-determined BVP, which may thus not have a proper "real" solution. To the authors knowledge, all comparable algorithms assume that a solution of the BVP exists, while most of them also require it to be unique and "sufficiently" smooth.

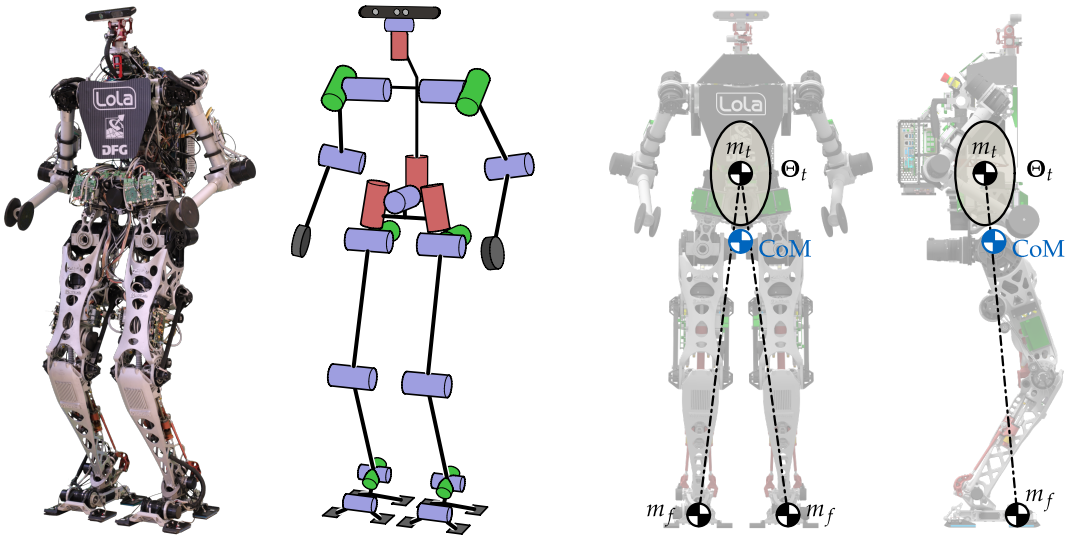

Figure 1. Humanoid robot LOLA developed at the Chair of Applied Mechanics, Technical University of Munich (TUM). The proposed algorithm is used within the walking pattern generation framework of LOLA, see [13] for details. The robot is 1.8 m tall and weights about 60 kg. Left: photo and kinematic configuration of the system with 24 actuated degrees of freedom. Right: simplified model of multi-body dynamics with torso mass m_t, foot mass m_f and torso mass moment of inertia Θ_t which (together with the ground reaction forces/torques) contribute to the ordinary differential equation (ODE) describing the center of mass (CoM) dynamics (blue).

Explaining the technical details of the walking pattern generation framework of LOLA goes beyond the scope of this paper. Instead, the interested reader is referred to [13], in which the application of the proposed algorithm is presented. The publication [13] comes with an accompanying video, see [14], which visualizes the sequential steps of the walking pattern generation pipeline of LOLA. The simulations and experiments presented in the video show the successful application of our algorithm. In the following, we do not further restrict ourselves to this special application. Since the proposed algorithm is generic, we derive it in the most general way since it may be useful also for different applications in robotics.

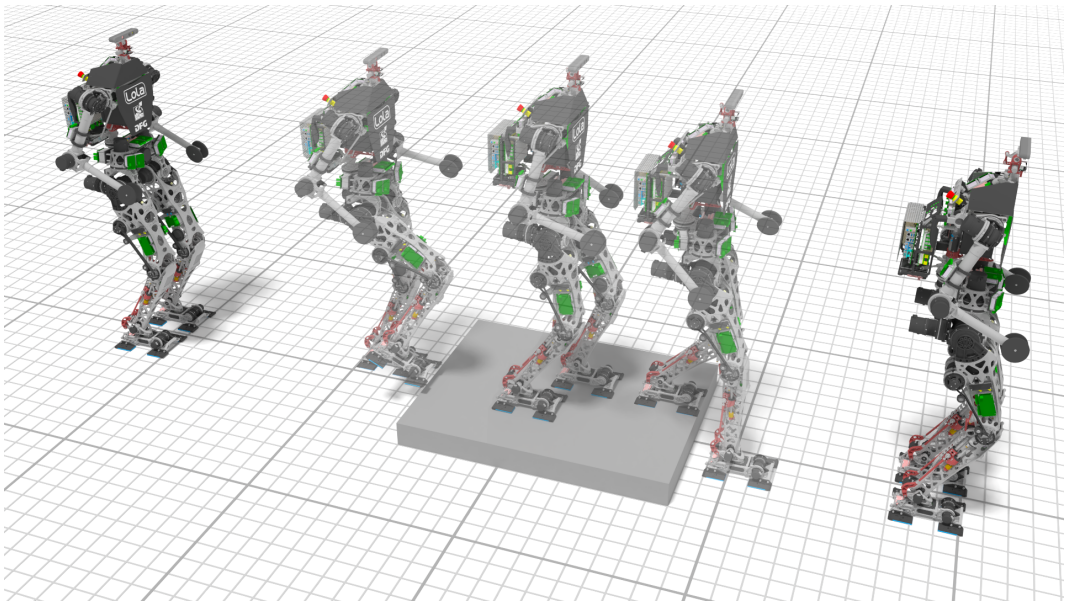

Figure 2. Visual interpretation of the over-determined boundary value problem (BVP): the start and end pose (**left**/**right**) represent the boundary conditions, while the intermediate motion (**transparent**) tries to approximate the inherent system dynamics of the simplified model.

1.3. Additional Remarks

Our method can be seen as smoothest spline collocation, i.e., it belongs to the "first" type as classified in Section 1.1. We do not apply special techniques to increase the order of convergence, but instead adhere to its basic form. This leads to a much simpler derivation and implementation. In addition it makes the algorithm faster by sacrificing approximation quality. This complies with the needs of our target application as explained in Section 1.2. We emphasize that our focus lies on simplicity, robustness, and efficiency. Thus we will not search for a mathematical formulation of the convergence order. Instead, the algorithm is evaluated mainly through experiments, where runtime performance is our primary concern.

As summarized in Section 1.1, there exist numerous methods for solving linear BVPs or spline interpolation/collocation in general. For most approaches, solving a large-sparse or small-dense linear system of equations (LSE) represents the main workload. To overcome this bottleneck, parallelized algorithms have been developed, which typically exploit the special structure of the involved matrices, e.g., the scheme proposed in [15] for staircase matrix structures. Although we aim at efficiency, we do not consider explicit parallelization as acceleration technique. This is because our algorithm is designed for embedded systems which typically feature only few physical CPU cores running also other time-critical tasks. Moreover, the use of GPUs, e.g., through CUDA or OPENCL, is often not feasible since the CPU–GPU interface lacks capabilities for hard real-time requirements. Finally, dedicated to our target application, we only consider (comparatively) small problems with runtimes $\approx 1\,\text{ms}$. For such systems the performance boost obtained through parallelization is likely to be canceled out by the synchronization overhead. Nevertheless, our algorithm may also be used for large scale problems. In this case an "off the shelf" parallel solver for dense LSEs may be used, cf. [16]. However, one should keep in mind that by using an iterative scheme execution time is not deterministic anymore, and, even worse, the solver might not converge. As an alternative, there exists an efficient way for the parallel solution of decoupled, multi-dimensional BVPs, which takes advantage of intermediate results, see Appendix C for details.

2. Materials and Methods

In the following, two versions of our algorithm are presented: one using a cubic spline and the other using a quintic spline for approximating the BVP. As the derivations and resulting algorithms are similar, we show the connecting links by presenting both methods in parallel. The version based on cubic splines is naturally simpler, although, using quintic splines leads to a smoother approximation. Indeed it is $\mathcal{C}^4$- instead of $\mathcal{C}^2$-continuous, which can be preferable for some use cases. Moreover, quintic splines allow us to directly preset first and second-order derivatives at both boundaries, which otherwise requires the introduction of virtual control points and in turn can lead to poor results, as discussed in Section 3. Although deriving the proposed collocation algorithm for quintic splines is more advanced than its cubic counterpart, overall performance is superior, because the same approximation quality can be obtained with less collocation sites and hence with less computational effort as shown in Section 3. However, this requires the underlying ODE to be sufficiently smooth. Note that, in contrast to our intention, most other investigations choose quintic splines for approximating fourth-order ODEs, e.g., in [8,10,11,17] (similar to choosing cubic splines for second-order ODEs).

We highlight that the proposed method is not only inspired by, but is also heavily based on the interpolation and collocation algorithms presented in [18,19], respectively. The main contribution of this paper is the combination, extension and runtime optimization of those methods. Moreover, we provide a detailed and self-contained derivation together with a fully documented open-source C++ reference implementation.

Having a background in mechanical engineering, the authors intention is to present a simple and self-contained derivation of the proposed algorithm, which can be easily understood, implemented, and extended by readers also lacking a dedicated mathematical background.

2.1. Problem Statement

Consider the second-order linear time-variant ODE

$$\alpha(t)\,\ddot{F}(t) + \beta(t)\,\dot{F}(t) + \gamma(t)\,F(t) = \tau(t) \qquad \text{for} \quad t \in [t_0,\, t_n] \tag{1}$$

where $\dot{F}(t)$ and $\ddot{F}(t)$ denote the first and second derivative of the unknown function $F(t)$ with respect to time t, i.e.,

$$\dot{F}(t) = \frac{\mathrm{d}F(t)}{\mathrm{d}t} \quad \text{and} \quad \ddot{F}(t) = \frac{\mathrm{d}^2\,F(t)}{\mathrm{d}t^2}\,.$$

Note that t does not have to represent time. However, this synonym is used in the following due to the typical appearance of (1) in dynamical systems. The coefficients α, β, γ, and the right-hand side τ are arbitrary, in general nonlinear, but known functions of t. Let system (1) be constrained by the BCs

$$\begin{aligned} F(t_0) &= F_0\,, \quad \dot{F}(t_0) = \dot{F}_0\,, \quad \ddot{F}(t_0) = \ddot{F}_0\,, \\ F(t_n) &= F_n\,, \quad \dot{F}(t_n) = \dot{F}_n\,, \quad \ddot{F}(t_n) = \ddot{F}_n\,, \end{aligned} \tag{2}$$

where t_0 and t_n define the considered time interval $t \in [t_0,\, t_n]$ and F_0, $\dot{F}_0$, $\ddot{F}_0$, F_n, $\dot{F}_n$, $\ddot{F}_n$ are user-defined constants. Then system (1) together with the BCs (2) represents the second-order linear two-point BVP for which an approximation is to be found. Note that (2) considers Dirichlet, Neumann, and second-order BCs independently of each other. In contrast, various other algorithms assume Robin BCs, i.e., a linear combination of Dirichlet and Neumann BCs, which is not equivalent to our approach. Due to (2), the BVP is over-determined and the existence of a solution $F(t)$ depends on the consistency of the BCs with the ODE.

In the following, we investigate the approximation of $F(t)$ through spline collocation, i.e., we generate a spline $y(t)$ which satisfies the underlying ODE (1) at a user-defined set of distinct

collocation sites $\{t_k\}$, numbered in increasing order, which lie within the considered interval (t_0, t_n), i.e.,

$$\alpha(t_k)\,\ddot{y}(t_k) + \beta(t_k)\,\dot{y}(t_k) + \gamma(t_k)\,y(t_k) = \tau(t_k) \qquad \text{for} \quad t_0 < t_k < t_{k+1} < t_n\,. \tag{3}$$

Moreover, $y(t)$ is forced to fulfill the BCs (2) at t_0 and t_n, i.e.,

$$\begin{aligned} y(t_0) = y_0 = F_0\,, \quad \dot{y}(t_0) = \dot{y}_0 = \dot{F}_0\,, \quad \ddot{y}(t_0) = \ddot{y}_0 = \ddot{F}_0\,, \\ y(t_n) = y_n = F_n\,, \quad \dot{y}(t_n) = \dot{y}_n = \dot{F}_n\,, \quad \ddot{y}(t_n) = \ddot{y}_n = \ddot{F}_n\,. \end{aligned} \tag{4}$$

Here we use $y(t)$ to denote the approximating spline while the exact solution is represented by $F(t)$. For clarity, we also use different denominations for $\{t_k\}$ and $\{y_k\}$ by using the terms collocation sites and collocation points, respectively. While $\{t_k\}$ are user-defined parameters, $\{y_k\}$ describe the solution to be found.

Since the proposed collocation algorithm is strongly related to the interpolation of cubic/quintic splines, which may not be common to some readers, spline interpolation is recapitulated in Section 2.3. Then, the proposed collocation method is derived in Section 2.7. Moreover, we reuse core elements of the interpolation algorithm during collocation, thus we cannot omit its derivation.

2.2. Spline Parametrization

Before diving into the derivation of algorithms, one first has to decide which spline representation to use. In the literature, formulations such as basis splines (B-splines) are common, since they feature inherent continuity and local control, which typically leads to banded systems [20]. In general, B-splines do not pass through their control points, which seems to make interpolation difficult at first sight; however, efficient algorithms for interpolation and collocation exist, see, for example, [21]. In [20], it has been shown that B-splines might not be as stable and efficient as other representations, namely monomial and Hermite type bases, especially when it comes to implementation. In particular, monomial bases have been recommended due to their superior condition, and thus lower roundoff errors. For all three forms, B-spline, Hermite type, and monomial, the core operation during Gaussian collocation is typically the solution of an almost block diagonal (ABD) linear system of equations [6,21]. A generic solver for these type of systems is SOLVEBLOK [22], while the special structure occurring for monomial bases is exploited by ABDPACK [23] with increased speed and lower memory consumption. Unfortunately, for smoothest spline collocation as presented in the following, the corresponding collocation matrix is dense, thus we cannot apply these algorithms. However, when compared to Gaussian collocation, the count of collocation sites and thus the dimension of the corresponding LSE is much smaller, which can lead to comparable performance.

Despite the popularity of B-splines, we use the so-called piecewise polynomial (PP) form [21], which describes the spline through the coefficients of interconnected, but independently defined, polynomial segments. We use a special type of monomial bases, namely the canonical form of the polynomials, which may not be as efficient as the choice in [20]; however, it makes our algorithm much simpler. By using this formulation, continuity between the spline segments needs to be explicitly established. The evaluation of the resulting spline, however, boils down to the evaluation of a single polynomial belonging to the corresponding segment, which is in general much quicker than evaluating the equivalent B-spline form. The evaluation of B-splines of degree p with the well-known de Boor's algorithm [24] takes $\mathcal{O}(p^2) + \mathcal{O}(p)$ operations [25]. There are optimized versions of it as proposed in [25,26]; however, these are numerically less stable [27]. In contrast, evaluating a polynomial of degree p with Horner's method [28] takes only $2\,p$, i.e., $\mathcal{O}(p)$, operations [29]. This is essential for time-critical applications, where the resulting spline has to be evaluated as quickly as possible. Note that we are free to construct the spline in B-spline formulation and convert it to the corresponding PP form in a post-processing step, see [21]. However, this introduces an additional (expensive) step which we try to avoid since, in our case, not only the evaluation but also the construction of the spline is time critical.

Let the spline $y(t)$ be defined as

$$y(t) = s_i(\xi_i(t)) \quad \text{for} \quad t_0 \leq t_i \leq t < t_{i+1} \leq t_n \quad \text{with} \quad i = 0, \ldots, n-1\,, \tag{5}$$

where s_i represents the i-th of the $n > 1$ spline segments parametrized by the normalized interpolation parameter ξ_i. We call $t \in [t_0, t_n]$ and $\xi_i \in [0, 1]$ the global and local interpolation parameters, respectively, for which we choose the linear mapping

$$\xi_i(t) = \frac{1}{h_i}\, t - \frac{t_i}{h_i} = (t - t_i)\, g_i \tag{6}$$

with $h_i > 0$ as the duration of the i-th segment $h_i = t_{i+1} - t_i$ and its reciprocal $g_i = 1/h_i$. The partitioning of the spline into n segments is visualized in Figure 3. In the following, we predominantly derive expressions in local segment space, i.e., with respect to ξ_i, since this makes the notation clearer, especially in Section 2.7. Note that, in contrast to some other approaches, we do not require homogeneous partitioning, thus the segmentation can be chosen arbitrarily as long as spline knots do not coincide. However, in Section 2.3, we show that for best numerical stability uniform partitioning should be used.

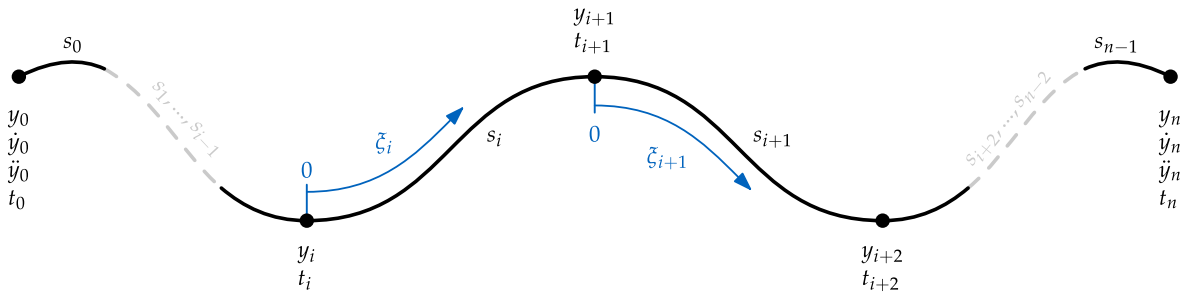

Figure 3. Segmentation and parametrization of the investigated spline $y(t)$. The spline consists of n interconnected segments, which share the interior knots with their neighbors. Each segment is described through the local interpolation parameter $\xi_i \in [0, 1]$ (blue).

In the case of cubic splines (left subscript C), each segment ${}_\mathrm{C}s_i$ represents a polynomial of degree three,

$${}_\mathrm{C}s_i(\xi_i) = {}_\mathrm{C}a_i\, \xi_i^3 + {}_\mathrm{C}b_i\, \xi_i^2 + {}_\mathrm{C}c_i\, \xi_i + {}_\mathrm{C}d_i \quad \text{with} \quad i = 0, \ldots, n-1 \tag{7}$$

where ${}_\mathrm{C}a_i$, ${}_\mathrm{C}b_i$, ${}_\mathrm{C}c_i$, and ${}_\mathrm{C}d_i$ are its constant coefficients. The first two derivatives of ${}_\mathrm{C}s_i(\xi_i(t))$ with respect to t are obtained by applying the chain rule:

$${}_\mathrm{C}\dot{s}_i(\xi_i) = \left(\frac{\mathrm{d}\,{}_\mathrm{C}s_i}{\mathrm{d}t}\right) = \left(\frac{\partial\,{}_\mathrm{C}s_i}{\partial \xi_i}\right)\left(\frac{\mathrm{d}\xi_i}{\mathrm{d}t}\right) = \frac{3}{h_i}\,{}_\mathrm{C}a_i\, \xi_i^2 + \frac{2}{h_i}\,{}_\mathrm{C}b_i\, \xi_i + \frac{1}{h_i}\,{}_\mathrm{C}c_i\,, \tag{8}$$

$${}_\mathrm{C}\ddot{s}_i(\xi_i) = \left(\frac{\mathrm{d}^2\,{}_\mathrm{C}s_i}{\mathrm{d}t^2}\right) = \left(\frac{\partial^2\,{}_\mathrm{C}s_i}{\partial \xi_i^2}\right)\left(\frac{\mathrm{d}\xi_i}{\mathrm{d}t}\right)^2 + \left(\frac{\partial\,{}_\mathrm{C}s_i}{\partial \xi_i}\right)\underbrace{\left(\frac{\mathrm{d}^2\xi_i}{\mathrm{d}t^2}\right)}_{=0} = \frac{6}{h_i^2}\,{}_\mathrm{C}a_i\, \xi_i + \frac{2}{h_i^2}\,{}_\mathrm{C}b_i\,. \tag{9}$$

For quintic splines (left subscript Q), each segment ${}_\mathrm{Q}s_i$ represents a polynomial of degree five,

$${}_\mathrm{Q}s_i(\xi_i) = {}_\mathrm{Q}a_i\, \xi_i^5 + {}_\mathrm{Q}b_i\, \xi_i^4 + {}_\mathrm{Q}c_i\, \xi_i^3 + {}_\mathrm{Q}d_i\, \xi_i^2 + {}_\mathrm{Q}e_i\, \xi_i + {}_\mathrm{Q}f_i \quad \text{with} \quad i = 0, \ldots, n-1 \tag{10}$$

where ${}_{\mathrm{Q}}a_i$, ${}_{\mathrm{Q}}b_i$, ${}_{\mathrm{Q}}c_i$, ${}_{\mathrm{Q}}d_i$, ${}_{\mathrm{Q}}e_i$, and ${}_{\mathrm{Q}}f_i$ are its constant coefficients. As in the cubic case, we obtain the first four derivatives of ${}_{\mathrm{Q}}s_i(\xi_i(t))$ with respect to t through the chain rule:

$$ {}_{\mathrm{Q}}\dot{s}_i(\xi_i) = \left(\frac{\mathrm{d}\,{}_{\mathrm{Q}}s_i}{\mathrm{d}t}\right) = \frac{5}{h_i}\,{}_{\mathrm{Q}}a_i\,\xi_i^4 + \frac{4}{h_i}\,{}_{\mathrm{Q}}b_i\,\xi_i^3 + \frac{3}{h_i}\,{}_{\mathrm{Q}}c_i\,\xi_i^2 + \frac{2}{h_i}\,{}_{\mathrm{Q}}d_i\,\xi_i + \frac{1}{h_i}\,{}_{\mathrm{Q}}e_i\,, \tag{11} $$

$$ {}_{\mathrm{Q}}\ddot{s}_i(\xi_i) = \left(\frac{\mathrm{d}^2\,{}_{\mathrm{Q}}s_i}{\mathrm{d}t^2}\right) = \frac{20}{h_i^2}\,{}_{\mathrm{Q}}a_i\,\xi_i^3 + \frac{12}{h_i^2}\,{}_{\mathrm{Q}}b_i\,\xi_i^2 + \frac{6}{h_i^2}\,{}_{\mathrm{Q}}c_i\,\xi_i + \frac{2}{h_i^2}\,{}_{\mathrm{Q}}d_i\,, \tag{12} $$

$$ {}_{\mathrm{Q}}s_i^{(3)}(\xi_i) = \left(\frac{\mathrm{d}^3\,{}_{\mathrm{Q}}s_i}{\mathrm{d}t^3}\right) = \frac{60}{h_i^3}\,{}_{\mathrm{Q}}a_i\,\xi_i^2 + \frac{24}{h_i^3}\,{}_{\mathrm{Q}}b_i\,\xi_i + \frac{6}{h_i^3}\,{}_{\mathrm{Q}}c_i\,, \tag{13} $$

$$ {}_{\mathrm{Q}}s_i^{(4)}(\xi_i) = \left(\frac{\mathrm{d}^4\,{}_{\mathrm{Q}}s_i}{\mathrm{d}t^4}\right) = \frac{120}{h_i^4}\,{}_{\mathrm{Q}}a_i\,\xi_i + \frac{24}{h_i^4}\,{}_{\mathrm{Q}}b_i\,. \tag{14} $$

2.3. Spline Interpolation: Preliminaries

In the following, we recall how a given set of $n+1$ data points $\{(t_i, y_i)\}$ with $i = 0, \ldots, n$ can be interpolated with a $\mathcal{C}^2$ or $\mathcal{C}^4$ smooth cubic or quintic spline, respectively. The derivation explicitly uses the PP form of the spline and leads to the same algorithm as presented in [18], except for slight modifications in notation. Note that [18] only deals with quintic splines. However, the method for cubic segments presented in this paper is a simplified version of the same scheme. Moreover, the derivation is investigated in more detail than it is in [18]. Readers not interested in these details are referred to Algorithm 1 which summarizes the results of this section.

In contrast to [18], we only consider the case of predefined first- and second-order derivatives at the boundaries of the quintic spline, i.e., we assume $\dot{y}_0$, $\ddot{y}_0$, $\dot{y}_n$, and $\ddot{y}_n$ to be given (as indicated in Figure 3). For the cubic counterpart, we lose two degrees of freedom, allowing us to predefine only two constraints out of $\{\dot{y}_0, \ddot{y}_0, \dot{y}_n, \ddot{y}_n\}$. For the remainder of this section, we restrict ourselves to the case of predefined second-order time derivatives $\ddot{y}_0$ and $\ddot{y}_n$ as this allows an efficient algorithm similar to the one presented for quintic splines. Note that this choice includes the common case of natural cubic splines, i.e., $\ddot{y}_0 = \ddot{y}_n = 0$. For cubic splines, we postpone the enforcement of the remaining boundary conditions, i.e., $\dot{y}_0$ and $\dot{y}_n$, to the end of Section 2.7.

In the following, we only consider distinct and ascending interpolation sites, i.e., $t_0 \leq t_i < t_{i+1} \leq t_n$ for $i = 0, \ldots, n-1$.

2.4. Cubic Spline Interpolation: Derivation

Since our goal is a spline which passes through the given data points $\{(t_i, y_i)\}$, we enforce the interpolation constraints

$$ \begin{aligned} {}_{\mathrm{C}}s_i(\xi_i)\Big|_{\xi_i=0} &\overset{!}{=} y_i \qquad \text{for } i = 0, \ldots, n-1 \qquad (n \text{ eqs.})\,, \\ {}_{\mathrm{C}}s_i(\xi_i)\Big|_{\xi_i=1} &\overset{!}{=} y_{i+1} \qquad \text{for } i = 0, \ldots, n-1 \qquad (n \text{ eqs.})\,. \end{aligned} \tag{15} $$

Furthermore, we aim at $\mathcal{C}^2$ continuity, thus we further require that

$$ \begin{aligned} {}_{\mathrm{C}}\dot{s}_i(\xi_i)\Big|_{\xi_i=1} &\overset{!}{=} {}_{\mathrm{C}}\dot{s}_{i+1}(\xi_{i+1})\Big|_{\xi_{i+1}=0} = \dot{y}_{i+1} \quad \text{for } i = 0, \ldots, n-2 \qquad (n-1 \text{ eqs.})\,, \\ {}_{\mathrm{C}}\ddot{s}_i(\xi_i)\Big|_{\xi_i=1} &\overset{!}{=} {}_{\mathrm{C}}\ddot{s}_{i+1}(\xi_{i+1})\Big|_{\xi_{i+1}=0} = \ddot{y}_{i+1} \quad \text{for } i = 0, \ldots, n-2 \qquad (n-1 \text{ eqs.})\,. \end{aligned} \tag{16} $$

Inserting (7) and (9) into (15) and in the second row of (16) allows us to reformulate the spline coefficients as

$$\begin{aligned} {}_{\mathrm{C}}a_i &= && && -\tfrac{1}{6}\,\ddot{y}_i\,h_i^2 && +\tfrac{1}{6}\,\ddot{y}_{i+1}\,h_i^2\,, \\ {}_{\mathrm{C}}b_i &= && && +\tfrac{1}{2}\,\ddot{y}_i\,h_i^2\,, && \\ {}_{\mathrm{C}}c_i &= && -y_i && +y_{i+1} && -\tfrac{1}{3}\,\ddot{y}_i\,h_i^2 && -\tfrac{1}{6}\,\ddot{y}_{i+1}\,h_i^2\,, \\ {}_{\mathrm{C}}d_i &= && +y_i\,, && && && \end{aligned} \tag{17}$$

where $\ddot{y}_i$ for $i = 1, \ldots, n-1$ are the $n-1$ unknowns which have to be computed using the remaining $n-1$ equations given by the first row of (16). For this purpose, we expand the first row of (16) with (8) and insert ${}_{\mathrm{C}}a_i$, ${}_{\mathrm{C}}b_i$, ${}_{\mathrm{C}}c_i$ and ${}_{\mathrm{C}}c_{i+1}$ from (17) to obtain

$$h_{i-1}\,\ddot{y}_{i-1} + 2\,(h_{i-1}+h_i)\,\ddot{y}_i + h_i\ddot{y}_{i+1} = \frac{6}{h_i}\,(y_{i+1}-y_i) - \frac{6}{h_{i-1}}\,(y_i - y_{i-1})\,, \tag{18}$$

which holds for $i = 1, \ldots, n-1$. This equation can be considered as defining intermediate unknowns $\ddot{y}_i$ and can be written as LSE of the form

$${}_{\mathrm{C}}\boldsymbol{A}\,{}_{\mathrm{C}}\boldsymbol{X} = {}_{\mathrm{C}}\boldsymbol{B} \tag{19}$$

with ${}_{\mathrm{C}}\boldsymbol{A} \in \mathbb{R}^{(n-1)\times(n-1)}$ and ${}_{\mathrm{C}}\boldsymbol{X}, {}_{\mathrm{C}}\boldsymbol{B} \in \mathbb{R}^{n-1}$ given by

$${}_{\mathrm{C}}\boldsymbol{A} = \begin{bmatrix} {}_{\mathrm{C}}D_1 & {}_{\mathrm{C}}U_1 & & & & & 0 \\ {}_{\mathrm{C}}L_2 & {}_{\mathrm{C}}D_2 & {}_{\mathrm{C}}U_2 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & {}_{\mathrm{C}}L_i & {}_{\mathrm{C}}D_i & {}_{\mathrm{C}}U_i & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & {}_{\mathrm{C}}L_{n-2} & {}_{\mathrm{C}}D_{n-2} & {}_{\mathrm{C}}U_{n-2} \\ 0 & & & & & {}_{\mathrm{C}}L_{n-1} & {}_{\mathrm{C}}D_{n-1} \end{bmatrix}, \quad {}_{\mathrm{C}}\boldsymbol{X} = \begin{bmatrix} {}_{\mathrm{C}}X_1 \\ {}_{\mathrm{C}}X_2 \\ \vdots \\ {}_{\mathrm{C}}X_i \\ \vdots \\ {}_{\mathrm{C}}X_{n-2} \\ {}_{\mathrm{C}}X_{n-1} \end{bmatrix}, \quad {}_{\mathrm{C}}\boldsymbol{B} = \begin{bmatrix} {}_{\mathrm{C}}B_1 \\ {}_{\mathrm{C}}B_2 \\ \vdots \\ {}_{\mathrm{C}}B_i \\ \vdots \\ {}_{\mathrm{C}}B_{n-2} \\ {}_{\mathrm{C}}B_{n-1} \end{bmatrix}, \tag{20}$$

and their elements

$${}_{\mathrm{C}}D_i = 2\,(h_{i-1}+h_i)\,, \quad {}_{\mathrm{C}}X_i = \ddot{y}_i \qquad \text{for } i = 1, \ldots, n-1\,, \tag{21}$$

$${}_{\mathrm{C}}U_i = h_i\,, \quad {}_{\mathrm{C}}L_{i+1} = {}_{\mathrm{C}}U_i \qquad \text{for } i = 1, \ldots, n-2\,, \tag{22}$$

$${}_{\mathrm{C}}B_i = \frac{6}{h_i}\,(y_{i+1}-y_i) - \frac{6}{h_{i-1}}\,(y_i - y_{i-1}) + {}_{\mathrm{C}}\Lambda_i \qquad \text{for } i = 1, \ldots, n-1\,, \tag{23}$$

$${}_{\mathrm{C}}\Lambda_i = \begin{cases} -h_0\,\ddot{y}_0 - h_{n-1}\,\ddot{y}_n & \text{for } n = 2\,, \quad (\equiv -{}_{\mathrm{C}}L_1\,{}_{\mathrm{C}}X_0 - {}_{\mathrm{C}}U_{n-1}\,{}_{\mathrm{C}}X_n) \\ -h_0\,\ddot{y}_0 & \text{for } n > 2 \wedge i = 1\,, \quad (\equiv -{}_{\mathrm{C}}L_1\,{}_{\mathrm{C}}X_0) \\ -h_{n-1}\,\ddot{y}_n & \text{for } n > 2 \wedge i = n-1\,, \quad (\equiv -{}_{\mathrm{C}}U_{n-1}\,{}_{\mathrm{C}}X_n) \\ 0 & \text{else} \end{cases}. \tag{24}$$

Herein ${}_{\mathrm{C}}L_i$, ${}_{\mathrm{C}}D_i$, and ${}_{\mathrm{C}}U_i \in \mathbb{R}$ represent the lower diagonal, diagonal, and upper diagonal elements of ${}_{\mathrm{C}}\boldsymbol{A}$, respectively. Note that ${}_{\mathrm{C}}\boldsymbol{A}$ only depends on the choice of $\{t_i\}$, thus it can be reused in case we need to perform further interpolations with the same segmentation $\{t_i\}$. While ${}_{\mathrm{C}}\boldsymbol{A}$ represents the interpolation sites $\{t_i\}$, the interpolation points $\{y_i\}$ are encoded in ${}_{\mathrm{C}}\boldsymbol{B}$. The additional term ${}_{\mathrm{C}}\Lambda_i$ incorporates the BCs such that we can write the system in the neat form of (20). From the solution ${}_{\mathrm{C}}\boldsymbol{X}$ we can directly obtain the unknowns $\ddot{y}_i$ which in turn can be used together with the data points

$\{(t_i, y_i)\}$ and BCs to compute the segment coefficients (17) such that the spline $y(t)$ is fully defined. Note that ${}_{\mathrm{C}}\boldsymbol{A}$ is symmetric, i.e., ${}_{\mathrm{C}}\boldsymbol{A} = {}_{\mathrm{C}}\boldsymbol{A}^T$; however, we do not make use of this property.

Although one can compute ${}_{\mathrm{C}}\boldsymbol{X}$ from (19) using an arbitrary solver for linear systems of equations, there is a more efficient way for doing so: since ${}_{\mathrm{C}}\boldsymbol{A}$ is tridiagonal, we can solve (19) with the Thomas algorithm [30,31]. Derived from an LU decomposition of ${}_{\mathrm{C}}\boldsymbol{A}$, one performs a recursive forward elimination

$$ {}_{\mathrm{C}}H_i := \begin{cases} \dfrac{{}_{\mathrm{C}}U_1}{{}_{\mathrm{C}}D_1} & \text{for } i = 1\,, \\ \dfrac{{}_{\mathrm{C}}U_i}{{}_{\mathrm{C}}D_i - {}_{\mathrm{C}}L_i\,{}_{\mathrm{C}}H_{i-1}} & \text{for } i = 2, \ldots, n-2 \end{cases}, \tag{25} $$

$$ {}_{\mathrm{C}}P_i := \begin{cases} \dfrac{{}_{\mathrm{C}}B_1}{{}_{\mathrm{C}}D_1} & \text{for } i = 1\,, \\ \dfrac{{}_{\mathrm{C}}B_i - {}_{\mathrm{C}}L_i\,{}_{\mathrm{C}}P_{i-1}}{{}_{\mathrm{C}}D_i - {}_{\mathrm{C}}L_i\,{}_{\mathrm{C}}H_{i-1}} & \text{for } i = 2, \ldots, n-1 \end{cases} \tag{26} $$

followed by a backward substitution

$$ {}_{\mathrm{C}}X_i = \begin{cases} {}_{\mathrm{C}}P_{n-1} & \text{for } i = n-1\,, \\ {}_{\mathrm{C}}P_i - {}_{\mathrm{C}}H_i\,{}_{\mathrm{C}}X_{i+1} & \text{for } i = n-2,\, n-3, \ldots, 1 \end{cases}. \tag{27} $$

Computing ${}_{\mathrm{C}}\boldsymbol{X}$ out of ${}_{\mathrm{C}}\boldsymbol{A}$ and ${}_{\mathrm{C}}\boldsymbol{B}$ thus boils down to $5\,n - 9$ operations. Note that in contrast to [31] where $A \in \mathbb{R}^{n\times n}$ in our case ${}_{\mathrm{C}}\boldsymbol{A} \in \mathbb{R}^{(n-1)\times(n-1)}$, thus n changes to $n-1$ for computing the count of operations in total [31]. Since ${}_{\mathrm{C}}\boldsymbol{A}$ is symmetric and positive definite, one may think of using an algorithm based on Cholesky factorization instead of LU decomposition, as this has proven to be approximately twice as efficient where applicable. However, this rule of thumb seems to be no longer valid for the special case of tridiagonal matrices: the Cholesky factorization $T = L\,D_L^{-1}\,L^T$ in [32], where the computation of the lower diagonal matrix L exploits the special structure and the diagonal matrix D_L is used to avoid evaluating expensive square roots, leads to $7\,n - 10$ operations in total.

The numerical stability of cubic spline interpolation is shown in Appendix A.1.

2.5. Quintic Spline Interpolation: Derivation

As previously mentioned, the following is a detailed version of the derivation given in [18]. Just as in the cubic case, our task is to pass through the given data points $\{(t_i, y_i)\}$, thus we enforce the interpolation constraints

$$ \begin{aligned} {}_{\mathrm{Q}}s_i(\xi_i)\Big|_{\xi_i=0} &\overset{!}{=} y_i \qquad \text{for } i = 0, \ldots, n-1 \qquad (n \text{ eqs.})\,, \\ {}_{\mathrm{Q}}s_i(\xi_i)\Big|_{\xi_i=1} &\overset{!}{=} y_{i+1} \qquad \text{for } i = 0, \ldots, n-1 \qquad (n \text{ eqs.})\,. \end{aligned} \tag{28} $$

We use the additional degrees of freedom to enforce not only $\mathcal{C}^2$, but instead $\mathcal{C}^4$ continuity with

$$ \begin{aligned} {}_{\mathrm{Q}}\dot{s}_i(\xi_i)\Big|_{\xi_i=1} &\overset{!}{=} {}_{\mathrm{Q}}\dot{s}_{i+1}(\xi_{i+1})\Big|_{\xi_{i+1}=0} = \dot{y}_{i+1} \qquad \text{for } i = 0, \ldots, n-2 \qquad (n-1 \text{ eqs.})\,, \\ {}_{\mathrm{Q}}\ddot{s}_i(\xi_i)\Big|_{\xi_i=1} &\overset{!}{=} {}_{\mathrm{Q}}\ddot{s}_{i+1}(\xi_{i+1})\Big|_{\xi_{i+1}=0} = \ddot{y}_{i+1} \qquad \text{for } i = 0, \ldots, n-2 \qquad (n-1 \text{ eqs.})\,, \\ {}_{\mathrm{Q}}s_i^{(3)}(\xi_i)\Big|_{\xi_i=1} &\overset{!}{=} {}_{\mathrm{Q}}s_{i+1}^{(3)}(\xi_{i+1})\Big|_{\xi_{i+1}=0} = y_{i+1}^{(3)} \qquad \text{for } i = 0, \ldots, n-2 \qquad (n-1 \text{ eqs.})\,, \\ {}_{\mathrm{Q}}s_i^{(4)}(\xi_i)\Big|_{\xi_i=1} &\overset{!}{=} {}_{\mathrm{Q}}s_{i+1}^{(4)}(\xi_{i+1})\Big|_{\xi_{i+1}=0} = y_{i+1}^{(4)} \qquad \text{for } i = 0, \ldots, n-2 \qquad (n-1 \text{ eqs.})\,. \end{aligned} \tag{29} $$

Inserting (10), (11), and (12) into (28) and into the first two rows of (29) allows us to reformulate the spline coefficients to

$$\begin{aligned}
{}_{\mathrm{Q}}a_i &= -6\,y_i \;\; +6\,y_{i+1} \;\; -3\,\dot{y}_i\,h_i \;\; -3\,\dot{y}_{i+1}\,h_i \;\; -\tfrac{1}{2}\,\ddot{y}_i\,h_i^2 \;\; +\tfrac{1}{2}\,\ddot{y}_{i+1}\,h_i^2\,, \\
{}_{\mathrm{Q}}b_i &= +15\,y_i \;\; -15\,y_{i+1} \;\; +8\,\dot{y}_i\,h_i \;\; +7\,\dot{y}_{i+1}\,h_i \;\; +\tfrac{3}{2}\,\ddot{y}_i\,h_i^2 \;\; -\ddot{y}_{i+1}\,h_i^2\,, \\
{}_{\mathrm{Q}}c_i &= -10\,y_i \;\; +10\,y_{i+1} \;\; -6\,\dot{y}_i\,h_i \;\; -4\,\dot{y}_{i+1}\,h_i \;\; -\tfrac{3}{2}\,\ddot{y}_i\,h_i^2 \;\; +\tfrac{1}{2}\,\ddot{y}_{i+1}\,h_i^2\,, \\
{}_{\mathrm{Q}}d_i &= +\tfrac{1}{2}\,\ddot{y}_i\,h_i^2\,, \\
{}_{\mathrm{Q}}e_i &= +\dot{y}_i\,h_i\,, \\
{}_{\mathrm{Q}}f_i &= +y_i\,,
\end{aligned} \tag{30}$$

where $\dot{y}_i$ and $\ddot{y}_i$ for $i = 1, \ldots, n-1$ are the $2\,(n-1)$ unknowns which we still have to determine using the remaining $2\,(n-1)$ equations given by the last two rows of (29). In particular, we expand the fourth row of (29) with (14) and insert ${}_{\mathrm{Q}}a_i$, ${}_{\mathrm{Q}}b_i$, and ${}_{\mathrm{Q}}b_{i+1}$ from (30) to obtain

$$\begin{aligned}
&-56\,g_{i-1}^3\,\dot{y}_{i-1} - 8\,g_{i-1}^2\,\ddot{y}_{i-1} - 64\,\dot{y}_i\,(g_{i-1}^3 + g_i^3) + 12\,\ddot{y}_i(g_{i-1}^2 - g_i^2) - 56\,g_i^3\,\dot{y}_{i+1} + 8\,g_i^2\,\ddot{y}_{i+1} = \\
&\quad = -120\,g_i^4\,(y_{i+1} - y_i) - 120\,g_{i-1}^4\,(y_i - y_{i-1})\,.
\end{aligned} \tag{31}$$

In the same manner, we expand the third row of (29) with (13) and insert ${}_{\mathrm{Q}}a_i$, ${}_{\mathrm{Q}}b_i$, ${}_{\mathrm{Q}}c_i$ and ${}_{\mathrm{Q}}c_{i+1}$ from (30) which leads to

$$\begin{aligned}
&-8\,g_{i-1}^2\,\dot{y}_{i-1} - g_{i-1}\,\ddot{y}_{i-1} - 12\,\dot{y}_i\,(g_{i-1}^2 - g_i^2) + 3\,\ddot{y}_i(g_{i-1} + g_i) + 8\,g_i^2\,\dot{y}_{i+1} - g_i\,\ddot{y}_{i+1} = \\
&\quad = 20\,g_i^3\,(y_{i+1} - y_i) - 20\,g_{i-1}^3\,(y_i - y_{i-1})\,.
\end{aligned} \tag{32}$$

Both (31) and (32) hold for $i = 1, \ldots, n-1$ which allows casting them in the form of

$${}_{\mathrm{Q}}\boldsymbol{A}\,{}_{\mathrm{Q}}\boldsymbol{X} = {}_{\mathrm{Q}}\boldsymbol{B} \tag{33}$$

with ${}_{\mathrm{Q}}\boldsymbol{A} \in \mathbb{R}^{2(n-1)\times 2(n-1)}$ and ${}_{\mathrm{Q}}\boldsymbol{X}$, ${}_{\mathrm{Q}}\boldsymbol{B} \in \mathbb{R}^{2(n-1)}$ given by

$${}_{\mathrm{Q}}\boldsymbol{A} = \begin{bmatrix}
{}_{\mathrm{Q}}\boldsymbol{D}_1 & {}_{\mathrm{Q}}\boldsymbol{U}_1 & & & & & \boldsymbol{0} \\
{}_{\mathrm{Q}}\boldsymbol{L}_2 & {}_{\mathrm{Q}}\boldsymbol{D}_2 & {}_{\mathrm{Q}}\boldsymbol{U}_2 & & & & \\
& \ddots & \ddots & \ddots & & & \\
& & {}_{\mathrm{Q}}\boldsymbol{L}_i & {}_{\mathrm{Q}}\boldsymbol{D}_i & {}_{\mathrm{Q}}\boldsymbol{U}_i & & \\
& & & \ddots & \ddots & \ddots & \\
& & & & {}_{\mathrm{Q}}\boldsymbol{L}_{n-2} & {}_{\mathrm{Q}}\boldsymbol{D}_{n-2} & {}_{\mathrm{Q}}\boldsymbol{U}_{n-2} \\
\boldsymbol{0} & & & & & {}_{\mathrm{Q}}\boldsymbol{L}_{n-1} & {}_{\mathrm{Q}}\boldsymbol{D}_{n-1}
\end{bmatrix}, {}_{\mathrm{Q}}\boldsymbol{X} = \begin{bmatrix}
{}_{\mathrm{Q}}\boldsymbol{X}_1 \\ {}_{\mathrm{Q}}\boldsymbol{X}_2 \\ \vdots \\ {}_{\mathrm{Q}}\boldsymbol{X}_i \\ \vdots \\ {}_{\mathrm{Q}}\boldsymbol{X}_{n-2} \\ {}_{\mathrm{Q}}\boldsymbol{X}_{n-1}
\end{bmatrix}, {}_{\mathrm{Q}}\boldsymbol{B} = \begin{bmatrix}
{}_{\mathrm{Q}}\boldsymbol{B}_1 \\ {}_{\mathrm{Q}}\boldsymbol{B}_2 \\ \vdots \\ {}_{\mathrm{Q}}\boldsymbol{B}_i \\ \vdots \\ {}_{\mathrm{Q}}\boldsymbol{B}_{n-2} \\ {}_{\mathrm{Q}}\boldsymbol{B}_{n-1}
\end{bmatrix}, \tag{34}$$

and their block components

$${}_{\mathrm{Q}}\boldsymbol{D}_i = \begin{bmatrix}
64\,(g_{i-1}^3 + g_i^3) & 12\,\kappa\,(g_{i-1}^2 - g_i^2)\,g_i \\
12\,\kappa\,(g_{i-1}^2 - g_i^2)\,g_i & 3\,\kappa^2\,(g_{i-1} + g_i)\,g_i^2
\end{bmatrix}, \quad {}_{\mathrm{Q}}\boldsymbol{X}_i = \begin{bmatrix}
-\dot{y}_i \\ \frac{1}{\kappa\,g_i}\,\ddot{y}_i
\end{bmatrix} \qquad \text{for } i = 1, \ldots, n-1\,, \tag{35}$$

$${}_{\mathrm{Q}}\boldsymbol{U}_i = \begin{bmatrix}
56\,g_i^3 & 8\,\kappa\,g_i^2\,g_{i+1} \\
-8\,\kappa\,g_i^3 & -\kappa^2\,g_i^2\,g_{i+1}
\end{bmatrix}, \quad {}_{\mathrm{Q}}\boldsymbol{L}_{i+1} = {}_{\mathrm{Q}}\boldsymbol{U}_i^T \qquad \text{for } i = 1, \ldots, n-2\,, \tag{36}$$

$$ {}_{\mathrm{Q}}\boldsymbol{B}_i = \begin{bmatrix} -120\, g_i^4 (y_{i+1} - y_i) - 120\, g_{i-1}^4 (y_i - y_{i-1}) \\ 20\,\kappa\, g_i^4 (y_{i+1} - y_i) - 20\,\kappa\, g_{i-1}^3\, g_i\, (y_i - y_{i-1}) \end{bmatrix} + {}_{\mathrm{Q}}\boldsymbol{\Lambda}_i \qquad \text{for } i = 1, \ldots, n-1\,, \tag{37} $$

$$ {}_{\mathrm{Q}}\boldsymbol{\Lambda}_i = \begin{cases} \begin{bmatrix} 56\, g_0^3\, \dot{y}_0 + 8\, g_0^2\, \ddot{y}_0 + 56\, g_{n-1}^3\, \dot{y}_n - 8\, g_{n-1}^2\, \ddot{y}_n \\ 8\,\kappa\, g_0^2\, g_1\, \dot{y}_0 + \kappa\, g_0\, g_1\, \ddot{y}_0 - 8\,\kappa\, g_{n-1}^3\, \dot{y}_n + \kappa\, g_{n-1}^2\, \ddot{y}_n \end{bmatrix} & \text{for } n = 2 \\ & (\equiv -{}_{\mathrm{Q}}\boldsymbol{L}_1\, {}_{\mathrm{Q}}\boldsymbol{X}_0 - {}_{\mathrm{Q}}\boldsymbol{U}_{n-1}\, {}_{\mathrm{Q}}\boldsymbol{X}_n)\,, \\ \begin{bmatrix} 56\, g_0^3\, \dot{y}_0 + 8\, g_0^2\, \ddot{y}_0 \\ 8\,\kappa\, g_0^2\, g_1\, \dot{y}_0 + \kappa\, g_0\, g_1\, \ddot{y}_0 \end{bmatrix} & \text{for } n > 2 \wedge i = 1 \\ & (\equiv -{}_{\mathrm{Q}}\boldsymbol{L}_1\, {}_{\mathrm{Q}}\boldsymbol{X}_0)\,, \\ \begin{bmatrix} 56\, g_{n-1}^3\, \dot{y}_n - 8\, g_{n-1}^2\, \ddot{y}_n \\ -8\,\kappa\, g_{n-1}^3\, \dot{y}_n + \kappa\, g_{n-1}^2\, \ddot{y}_n \end{bmatrix} & \text{for } n > 2 \wedge i = n-1 \\ & (\equiv -{}_{\mathrm{Q}}\boldsymbol{U}_{n-1}\, {}_{\mathrm{Q}}\boldsymbol{X}_n)\,, \\ \boldsymbol{0} & \text{else} \end{cases} \tag{38} $$

Just as in the cubic case, ${}_{\mathrm{Q}}\boldsymbol{L}_i$, ${}_{\mathrm{Q}}\boldsymbol{D}_i$, and ${}_{\mathrm{Q}}\boldsymbol{U}_i \in \mathbb{R}^{2\times 2}$ represent the lower diagonal, diagonal, and upper diagonal blocks of ${}_{\mathrm{Q}}\boldsymbol{A}$, respectively. As suggested in [18], the additional parameter κ in the definitions (35)–(38) is chosen as

$$ \kappa = \sqrt{\frac{64}{3}}\,. \tag{39} $$

From an analytical point of view, κ has no influence on the solution ${}_{\mathrm{Q}}\boldsymbol{X}$ (at least if $\kappa \neq 0$). However, it improves numerical stability which is verified in Appendix A.2.

As for the cubic spline ${}_{\mathrm{Q}}\boldsymbol{A}$ is symmetric and only depends on the choice of $\{t_i\}$, while the interpolation points $\{y_i\}$ are contained in ${}_{\mathrm{Q}}\boldsymbol{B}$. As before, the additional term ${}_{\mathrm{Q}}\boldsymbol{\Lambda}_i$ incorporates the BCs such that we can write the system in the form of (34). In contrast to the cubic case, the solution ${}_{\mathrm{Q}}\boldsymbol{X}$ represents not only $\ddot{y}_i$, but also $\dot{y}_i$, which is now additionally required to compute the segment coefficients from (30).

Since ${}_{\mathrm{Q}}\boldsymbol{A}$ is block-tridiagonal, we can again solve (33) efficiently with the generalization of the Thomas algorithm to block-tridiagonal matrices [31]. Based on an *LU* decomposition of ${}_{\mathrm{Q}}\boldsymbol{A}$, we first run a recursive forward elimination

$$ {}_{\mathrm{Q}}\boldsymbol{H}_i := \begin{cases} {}_{\mathrm{Q}}\boldsymbol{D}_1^{-1}\, {}_{\mathrm{Q}}\boldsymbol{U}_1 & \text{for } i = 1\,, \\ \left({}_{\mathrm{Q}}\boldsymbol{D}_i - {}_{\mathrm{Q}}\boldsymbol{L}_i\, {}_{\mathrm{Q}}\boldsymbol{H}_{i-1}\right)^{-1} {}_{\mathrm{Q}}\boldsymbol{U}_i & \text{for } i = 2, \ldots, n-2 \end{cases}, \tag{40} $$

$$ {}_{\mathrm{Q}}\boldsymbol{P}_i := \begin{cases} {}_{\mathrm{Q}}\boldsymbol{D}_1^{-1}\, {}_{\mathrm{Q}}\boldsymbol{B}_1 & \text{for } i = 1\,, \\ \left({}_{\mathrm{Q}}\boldsymbol{D}_i - {}_{\mathrm{Q}}\boldsymbol{L}_i\, {}_{\mathrm{Q}}\boldsymbol{H}_{i-1}\right)^{-1} \left({}_{\mathrm{Q}}\boldsymbol{B}_i - {}_{\mathrm{Q}}\boldsymbol{L}_i\, {}_{\mathrm{Q}}\boldsymbol{P}_{i-1}\right) & \text{for } i = 2, \ldots, n-1 \end{cases} \tag{41} $$

followed by a backward substitution

$$ {}_{\mathrm{Q}}\boldsymbol{X}_i = \begin{cases} {}_{\mathrm{Q}}\boldsymbol{P}_{n-1} & \text{for } i = n-1\,, \\ {}_{\mathrm{Q}}\boldsymbol{P}_i - {}_{\mathrm{Q}}\boldsymbol{H}_i\, {}_{\mathrm{Q}}\boldsymbol{X}_{i+1} & \text{for } i = n-2, n-3, \ldots, 1 \end{cases}. \tag{42} $$

Computing ${}_{\mathrm{Q}}\boldsymbol{X}$ out of ${}_{\mathrm{Q}}\boldsymbol{A}$ and ${}_{\mathrm{Q}}\boldsymbol{B}$ requires at a maximum $36\,n - 60$ operations. Again [31] uses $A \in \mathbb{R}^{2n\times 2n}$ while in our case ${}_{\mathrm{Q}}\boldsymbol{A} \in \mathbb{R}^{2(n-1)\times 2(n-1)}$ holds, thus n changes to $n-1$ for computing the count of operations in total [31]. For this upper bound, explicit computation of the inverse of ${}_{\mathrm{Q}}\boldsymbol{D}_1$ and $\left({}_{\mathrm{Q}}\boldsymbol{D}_i - {}_{\mathrm{Q}}\boldsymbol{L}_i\, {}_{\mathrm{Q}}\boldsymbol{H}_{i-1}\right)$ by Gaussian elimination is assumed, which in practice should be avoided by

solving a 2×2 LSE instead [31]. Thus, a corresponding implementation can be expected to require even less operations.

In Appendix A.2, the interested reader can find considerations on the numerical stability of quintic spline interpolation.

2.6. Algorithm for Cubic/Quintic Spline Interpolation

Since the presented derivation is rather lengthy, the key steps for interpolating cubic/quintic splines following the proposed method are summarized in Algorithm 1.

Algorithm 1: Cubic/Quintic Spline Interpolation.

Input: Interpolation parameters consisting of

- $n+1$ distinct and ascending interpolation sites $\{t_i\}$,
- their corresponding data values $\{y_i\}$,
- BCs $\{\ddot{y}_0, \ddot{y}_n\}$ (cubic) or $\{\dot{y}_0, \ddot{y}_0, \dot{y}_n, \ddot{y}_n\}$ (quintic) at t_0 and t_n

where $i = 0, \ldots, n$ and $n > 1$.

Output: Cubic/quintic spline, consisting of n interconnected segments in PP form, which

- interpolates the given data values at all spline knots $\{t_0, \ldots, t_n\}$,
- is $\mathcal{C}^2$ (cubic) or $\mathcal{C}^4$ (quintic) continuous at interior spline knots $\{t_1, \ldots, t_{n-1}\}$,
- satisfies the BCs.

begin

Cubic: Setup ${}_{\mathrm{C}}L_i$, ${}_{\mathrm{C}}D_i$, ${}_{\mathrm{C}}U_i$, and ${}_{\mathrm{C}}B_i$ from (21), (22), (23), and (24)
Quintic: Setup ${}_{\mathrm{Q}}\boldsymbol{L}_i$, ${}_{\mathrm{Q}}\boldsymbol{D}_i$, ${}_{\mathrm{Q}}\boldsymbol{U}_i$, and ${}_{\mathrm{Q}}\boldsymbol{B}_i$ from (35), (36), (37), and (38)

Cubic: Compute ${}_{\mathrm{C}}X_i$ using the recursive scheme given in (25), (26), and (27)
Quintic: Compute ${}_{\mathrm{Q}}\boldsymbol{X}_i$ using the recursive scheme given in (40), (41), and (42)

Cubic: Extract $\ddot{y}_i$ from ${}_{\mathrm{C}}X_i$ and compute segment coefficients ${}_{\mathrm{C}}a_i$, ${}_{\mathrm{C}}b_i$, ${}_{\mathrm{C}}c_i$, and ${}_{\mathrm{C}}d_i$ using (17)
Quintic: Extract $\dot{y}_i$, $\ddot{y}_i$ from ${}_{\mathrm{Q}}\boldsymbol{X}_i$ and compute segment coefficients ${}_{\mathrm{Q}}a_i$, ${}_{\mathrm{Q}}b_i$, ${}_{\mathrm{Q}}c_i$, ${}_{\mathrm{Q}}d_i$, ${}_{\mathrm{Q}}e_i$, and ${}_{\mathrm{Q}}f_i$ using (30)

end

For details on convergence order and approximation error of quintic spline interpolation, the interested reader is referred to [18] where these issues have been experimentally investigated for various examples.

2.7. Spline Collocation: Derivation

The following is based on the collocation algorithm presented in [19,33]. However, we extend the method from cubic to quintic splines. Moreover, we do not use natural splines, but instead integrate the boundary conditions directly into the scheme. Lastly, in contrast to [19,33], we do not need to modify the right-hand side of (1), thus leading to a "true" collocation of the ODE for all collocation sites, which are chosen to be the interior spline knots. As runtime performance is of the highest priority for our application, we choose smoothest spline collocation. This minimizes the count of (expensive) collocation sites, thus reduces the count of equations to solve, and instead uses the available degrees of freedom to force $\mathcal{C}^2$ (cubic spline) or $\mathcal{C}^4$ (quintic spline) continuity. Moreover, in our application, $y(t)$ is used as input for controlling the motion of a robot, thus a smooth $y(t)$ is equivalent to small changes in joint accelerations, i.e., motor jerks, which in turn improves overall stability during locomotion.

As stated in Section 2.1, we require the approximation $y(t)$ to fulfill the underlying ODE at certain collocation sites $\{t_k\}$, see (3), while simultaneously satisfying the boundary conditions as specified in (4). Note that we use the index k instead of i to highlight that our new task consists in collocating the ODE at the interior knots, i.e., $k = 1, \ldots, n-1$ rather than the previously investigated interpolation at all knots, i.e., $i = 0, \ldots, n$. Furthermore, it should be pointed out that although (3) holds, this does not imply that $y(t_k) = F(t_k)$, $\dot{y}(t_k) = \dot{F}(t_k)$, or $\ddot{y}(t_k) = \ddot{F}(t_k)$. In other words, $y(t)$ will not coincide with the real solution $F(t)$ at the collocation sites $\{t_k\}$. However, it will behave similarly at these spots (meaning that they will satisfy the same Equation (1)), which is illustrated in Figure 4.

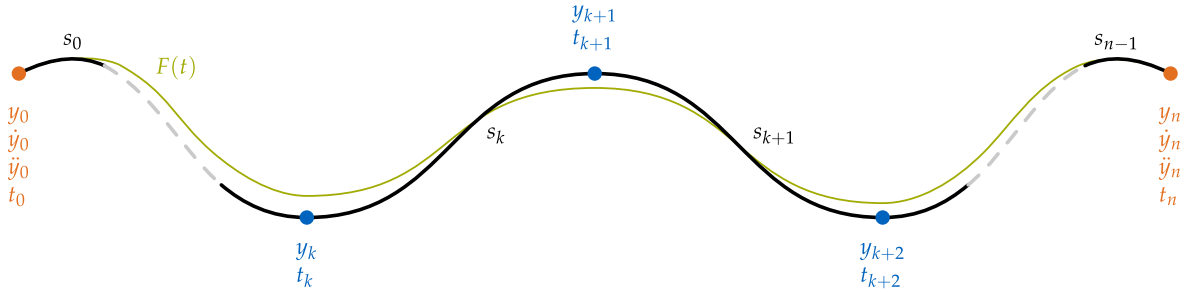

Figure 4. The computed spline $y(t)$ (black) approximating the real solution $F(t)$ (green). The approximation satisfies the underlying ODE at the specified collocation sites $\{t_k\}$ (blue), and fulfills the boundary conditions (BC) at t_0 and t_n (orange), but do not necessarily coincide at the collocation points.

As first step, we introduce the auxiliary variables $\boldsymbol{\lambda}$, $\boldsymbol{\eta}$, and $\boldsymbol{r}$ which are defined as

$$\begin{aligned} {}_{\mathrm{C}}\boldsymbol{\lambda} &:= \begin{bmatrix} y_1, \ldots, y_{n-1} \end{bmatrix}^T, \quad {}_{\mathrm{C}}\boldsymbol{\eta} := \begin{bmatrix} \ddot{y}_1, \ldots, \ddot{y}_{n-1} \end{bmatrix}^T, \qquad {}_{\mathrm{C}}\boldsymbol{r} := \begin{bmatrix} y_0, \ddot{y}_0, y_n, \ddot{y}_n \end{bmatrix}^T, \\ {}_{\mathrm{Q}}\boldsymbol{\lambda} &:= \begin{bmatrix} y_1, \ldots, y_{n-1} \end{bmatrix}^T, \quad {}_{\mathrm{Q}}\boldsymbol{\eta} := \begin{bmatrix} \dot{y}_1, \ddot{y}_1, \ldots, \dot{y}_{n-1}, \ddot{y}_{n-1} \end{bmatrix}^T, \quad {}_{\mathrm{Q}}\boldsymbol{r} := \begin{bmatrix} y_0, \dot{y}_0, \ddot{y}_0, y_n, \dot{y}_n, \ddot{y}_n \end{bmatrix}^T, \end{aligned} \tag{43}$$

where ${}_{\mathrm{C}}\boldsymbol{\lambda}$, ${}_{\mathrm{C}}\boldsymbol{\eta}$, ${}_{\mathrm{C}}\boldsymbol{r}$ and ${}_{\mathrm{Q}}\boldsymbol{\lambda}$, ${}_{\mathrm{Q}}\boldsymbol{\eta}$, ${}_{\mathrm{Q}}\boldsymbol{r}$ are the corresponding counterparts for the case of cubic and quintic splines, respectively. While $\boldsymbol{\lambda}$ represents the (yet unknown) collocation points $\{y_k\}$, $\boldsymbol{\eta}$ contains their corresponding first (and second) time derivatives, which can be seen as "internal" unknowns, as they will be implicitly defined through an embedded spline interpolation. Lastly, $\boldsymbol{r}$ depicts the BCs, where we lack $\dot{y}_0$ and $\dot{y}_n$ in the case of cubic splines as has been previously explained.

From (17), (30), and (43) we observe that the spline segments s_i are linear with respect to $\boldsymbol{\lambda}$, $\boldsymbol{\eta}$, and $\boldsymbol{r}$, i.e.,

$$s_i(\xi_i) = \underbrace{\left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{\lambda}}\right)}_{\text{known}} \boldsymbol{\lambda} + \underbrace{\left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{\eta}}\right)}_{\text{known}} \boldsymbol{\eta} + \underbrace{\left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{r}}\right)}_{\text{known}} \boldsymbol{r} \qquad \text{for } i = 0, \ldots, n-1 \tag{44}$$

holds. The gradients are fully defined by the spline partitioning $\{t_i\}$, which is assumed to be known. Thus the construction of the spline $y(t)$ is equivalent to the search for a corresponding $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$. Note that to obtain (44), we used (17) and (30), which in turn were derived from fulfilling the interpolation condition together with enforcing continuity of the second time derivative (cubic spline), or first and second time derivative (quintic spline) at the interior knots. In order to accomplish full $\mathcal{C}^2$ and $\mathcal{C}^4$ continuity, we further make use of $\boldsymbol{A}\,\boldsymbol{X} = \boldsymbol{B}$ from (19) and (33), which represents continuity of the first time derivative (cubic spline) or third and fourth time derivative (quintic spline), respectively. In particular, we observe from (23), (24), (37), and (38), that $\boldsymbol{B}$ is linear with respect to $\boldsymbol{\lambda}$ and $\boldsymbol{r}$, thus

$$\boldsymbol{A}\,\boldsymbol{X} = \boldsymbol{B} = \underbrace{\left(\frac{\partial \boldsymbol{B}}{\partial \boldsymbol{\lambda}}\right)}_{\text{known}} \boldsymbol{\lambda} + \underbrace{\left(\frac{\partial \boldsymbol{B}}{\partial \boldsymbol{r}}\right)}_{\text{known}} \boldsymbol{r} \tag{45}$$

holds, where the gradients again depend only on the known partitioning $\{t_i\}$. We further observe that, according to the definitions (21), (35) and (43), we can write the mapping

$$\boldsymbol{X} = \boldsymbol{S}\,\boldsymbol{\eta} \quad \text{with} \quad {}_{\mathrm{C}}\boldsymbol{S} := \boldsymbol{I} \quad \text{and} \quad {}_{\mathrm{Q}}\boldsymbol{S} := \operatorname{diag}\left({}_{\mathrm{Q}}\boldsymbol{S}_1, \ldots, {}_{\mathrm{Q}}\boldsymbol{S}_{n-1}\right) \quad \text{where} \quad {}_{\mathrm{Q}}\boldsymbol{S}_i := \begin{bmatrix} -1 & 0 \\ 0 & \frac{1}{\kappa g_i} \end{bmatrix} \tag{46}$$

with $\boldsymbol{I}$ being the identity matrix of appropriate size. Since $\boldsymbol{A}$ and $\boldsymbol{S}$ are constant, by "constant" we mean in this context that an expression does not depend on the yet unknown $\boldsymbol{\lambda}$ or $\boldsymbol{\eta}$, and assumed to be non-singular, it is clear from (45) that not only $\boldsymbol{B}$, but also $\boldsymbol{X}$ and thus $\boldsymbol{\eta}$ are linear with respect to $\boldsymbol{\lambda}$ and $\boldsymbol{r}$. Hence one can write

$$\boldsymbol{X} = \left(\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{\lambda}}\right)\boldsymbol{\lambda} + \left(\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{r}}\right)\boldsymbol{r} \qquad \text{and} \qquad \boldsymbol{\eta} = \left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\lambda}}\right)\boldsymbol{\lambda} + \left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{r}}\right)\boldsymbol{r}\,. \tag{47}$$

Note that $\boldsymbol{A}$ and ${}_{\mathrm{Q}}\boldsymbol{S}$ only depend on the known $\{t_i\}$, which allows us to safely differentiate (45) with respect to $\boldsymbol{\lambda}$ and $\boldsymbol{r}$ to obtain

$$\underbrace{\boldsymbol{A}\left(\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{\lambda}}\right)}_{\boldsymbol{S}\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\lambda}}\right)} = \underbrace{\left(\frac{\partial \boldsymbol{B}}{\partial \boldsymbol{\lambda}}\right)}_{\text{known}} \qquad \text{and} \qquad \underbrace{\boldsymbol{A}\left(\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{r}}\right)}_{\boldsymbol{S}\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{r}}\right)} = \underbrace{\left(\frac{\partial \boldsymbol{B}}{\partial \boldsymbol{r}}\right)}_{\text{known}}\,. \tag{48}$$

which we can use to compute the yet unknown gradients in (47). Note that this can be done very efficiently due to the (block-)tridiagonal form of $\boldsymbol{A}$ as already discussed in Section 2.3. Since $\boldsymbol{S}$ is diagonal, this property also holds for the product $\boldsymbol{A}\,\boldsymbol{S}$. However, for best numerical stability, one should solve for the gradients of $\boldsymbol{X}$ first and use the mapping (46) to obtain the gradients of $\boldsymbol{\eta}$ afterwards, which is of negligible cost since $\boldsymbol{S}$, and thus also $\boldsymbol{S}^{-1}$, is diagonal. The right-hand sides necessary to solve (48) only depend on $\{t_i\}$ and are derived in Appendix B. Lastly, we insert $\boldsymbol{\eta}$ from (47) into (44) and obtain

$$s_i(\xi_i) = \left[\left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{\lambda}}\right) + \left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{\eta}}\right)\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\lambda}}\right)\right]\boldsymbol{\lambda} + \left[\left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{r}}\right) + \left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{\eta}}\right)\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{r}}\right)\right]\boldsymbol{r} \tag{49}$$

or equivalently

$$s_i(\xi_i) = \nabla_{\boldsymbol{\lambda}} s_i(\xi_i)\,\boldsymbol{\lambda} + \nabla_{\boldsymbol{r}} s_i(\xi_i)\,\boldsymbol{r} \tag{50}$$

with the known spline gradients

$$\nabla_{\boldsymbol{\lambda}} s_i(\xi_i) := \left[\left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{\lambda}}\right) + \left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{\eta}}\right)\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\lambda}}\right)\right], \quad \nabla_{\boldsymbol{r}} s_i(\xi_i) := \left[\left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{r}}\right) + \left(\frac{\partial s_i(\xi_i)}{\partial \boldsymbol{\eta}}\right)\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{r}}\right)\right]. \tag{51}$$

We can obtain the corresponding expressions for the first and second time derivatives by following the exact same scheme. In particular, we get

$$\dot{s}_i(\xi_i) = \nabla_{\boldsymbol{\lambda}} \dot{s}_i(\xi_i)\,\boldsymbol{\lambda} + \nabla_{\boldsymbol{r}} \dot{s}_i(\xi_i)\,\boldsymbol{r}\,, \quad \ddot{s}_i(\xi_i) = \nabla_{\boldsymbol{\lambda}} \ddot{s}_i(\xi_i)\,\boldsymbol{\lambda} + \nabla_{\boldsymbol{r}} \ddot{s}_i(\xi_i)\,\boldsymbol{r} \tag{52}$$

with

$$\nabla_{\boldsymbol{\lambda}} \dot{s}_i(\xi_i) := \left[\left(\frac{\partial \dot{s}_i(\xi_i)}{\partial \boldsymbol{\lambda}}\right) + \left(\frac{\partial \dot{s}_i(\xi_i)}{\partial \boldsymbol{\eta}}\right)\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\lambda}}\right)\right], \quad \nabla_{\boldsymbol{r}} \dot{s}_i(\xi_i) := \left[\left(\frac{\partial \dot{s}_i(\xi_i)}{\partial \boldsymbol{r}}\right) + \left(\frac{\partial \dot{s}_i(\xi_i)}{\partial \boldsymbol{\eta}}\right)\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{r}}\right)\right], \tag{53}$$

$$\nabla_{\boldsymbol{\lambda}} \ddot{s}_i(\xi_i) := \left[\left(\frac{\partial \ddot{s}_i(\xi_i)}{\partial \boldsymbol{\lambda}}\right) + \left(\frac{\partial \ddot{s}_i(\xi_i)}{\partial \boldsymbol{\eta}}\right)\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\lambda}}\right)\right], \quad \nabla_{\boldsymbol{r}} \ddot{s}_i(\xi_i) := \left[\left(\frac{\partial \ddot{s}_i(\xi_i)}{\partial \boldsymbol{r}}\right) + \left(\frac{\partial \ddot{s}_i(\xi_i)}{\partial \boldsymbol{\eta}}\right)\left(\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{r}}\right)\right]. \tag{54}$$

Although computing the spline gradients can be done very efficiently, their mathematical representation is rather lengthy. A formulation of the gradients, which is ready for implementation, is given in Appendix B.

Lastly, we fulfill the dynamics of the ODE by inserting (50) and (52) into (3), which leads to

$$\alpha(t_k)\,\ddot{s}_k(\xi_k = 0) + \beta(t_k)\,\dot{s}_k(\xi_k = 0) + \gamma(t_k)\,s_k(\xi_k = 0) = \tau(t_k) \tag{55}$$

for $t_0 < t_k < t_n$ and $t_k < t_{k+1}$ with $k = 1, \ldots, n-1$. This can be formulated as LSE

$$\boldsymbol{A}_{\text{coll}}\boldsymbol{\lambda} = \boldsymbol{B}_{\text{coll}} \tag{56}$$

with

$$\boldsymbol{A}_{\text{coll}} = \begin{bmatrix} \boldsymbol{A}_{\text{coll},1} \\ \vdots \\ \boldsymbol{A}_{\text{coll},k} \\ \vdots \\ \boldsymbol{A}_{\text{coll},n-1} \end{bmatrix} \in \mathbb{R}^{(n-1)\times(n-1)}\,, \qquad \boldsymbol{B}_{\text{coll}} = \begin{bmatrix} B_{\text{coll},1} \\ \vdots \\ B_{\text{coll},k} \\ \vdots \\ B_{\text{coll},n-1} \end{bmatrix} \in \mathbb{R}^{(n-1)} \tag{57}$$

where the k-th row of $\boldsymbol{A}_{\text{coll}}$ and $\boldsymbol{B}_{\text{coll}}$ corresponds to the collocation site t_k and is given by

$$\boldsymbol{A}_{\text{coll},k} = \alpha(t_k)\,\nabla_{\boldsymbol{\lambda}}\ddot{s}_k(0) + \beta(t_k)\,\nabla_{\boldsymbol{\lambda}}\dot{s}_k(0) + \gamma(t_k)\,\nabla_{\boldsymbol{\lambda}}s_k(0) \quad \in \mathbb{R}^{1\times(n-1)}\,, \tag{58}$$

$$B_{\text{coll},k} = \tau(t_k) - \left[\alpha(t_k)\,\nabla_{\boldsymbol{r}}\ddot{s}_k(0) + \beta(t_k)\,\nabla_{\boldsymbol{r}}\dot{s}_k(0) + \gamma(t_k)\,\nabla_{\boldsymbol{r}}s_k(0)\right]\boldsymbol{r} \quad \in \mathbb{R}\,. \tag{59}$$

Note that we choose a partitioning of the spline such that the collocation sites coincide with the starting ("left") knot of each segment, i.e., $\xi_k = 0$, see Figure 4. This allows us to skip the computation of certain spline gradients. Since the underlying ODE is of order two, only gradients of the last three coefficients, i.e., ${}_{\text{C}}b_i$, ${}_{\text{C}}c_i$, ${}_{\text{C}}d_i$ (cubic) or ${}_{\text{Q}}d_i$, ${}_{\text{Q}}e_i$, ${}_{\text{Q}}f_i$ (quintic), have to be computed, for details see Appendix B. This simplifies the implementation and improves the overall performance. Solving (56) for $\boldsymbol{\lambda}$ represents the key operation (i.e., bottleneck for large n) of the proposed collocation method, since $\boldsymbol{A}_{\text{coll}}$ is in general dense while all other operations are either simple explicit expressions or linear systems in (block-)tridiagonal form, which can be solved efficiently. This justifies our strategy to minimize the count of collocation points (and thus the dimension of $\boldsymbol{A}_{\text{coll}}$) and instead force high order continuity.

As soon as $\boldsymbol{\lambda}$ has been obtained, we can compute $\boldsymbol{\eta}$ directly from (47), since the gradients of $\boldsymbol{\eta}$ with respect to $\boldsymbol{\lambda}$ and $\boldsymbol{r}$ are already available as by-products of computing $\boldsymbol{\lambda}$, see (48). From $\boldsymbol{\lambda}$, $\boldsymbol{\eta}$ and $\boldsymbol{r}$ the segment coefficients can finally be computed using (17) or (30).

2.8. Satisfying First Order Boundary Conditions for Cubic Splines

The presented method for cubic spline collocation respects y_0, $\ddot{y}_0$, y_n, and $\ddot{y}_n$ as BCs. However, our task was to fulfill all BCs given in (4), which seems at first to be only possible with quintic spline collocation. Also satisfying the first-order BCs, i.e., $\dot{y}_0$ and $\dot{y}_n$, can be achieved with moderate effort. For that purpose we insert two auxiliary knots, the so-called virtual control points $t_{\text{virt},1}$ and $t_{\text{virt},2}$, which give us the necessary degrees of freedom to introduce additional constraints. The term "virtual" highlights that these points are not used as collocation sites. Both, $t_{\text{virt},1}$ and $t_{\text{virt},2}$, have to lie within the specified start- and endtime and must not coincide with the collocation sites. This way the spline remains properly partitioned. For simplicity, we place the virtual control points at the centers of the (originally) first and last segments, i.e.,

$$t_{\text{virt},1} := \frac{t_{\text{orig},0} + t_{\text{orig},1}}{2} \quad \text{and} \quad t_{\text{virt},2} := \frac{t_{\text{orig},n-1} + t_{\text{orig},n}}{2}\,. \tag{60}$$

Obviously, inserting two knots leads to a different segmentation of the spline, i.e., $n \to n+2$; however, the boundaries and collocation sites remain unchanged. If we adapt the indexing such that $t_1 := t_{\text{virt},1}$ and $t_{n-1} := t_{\text{virt},2}$, all findings derived so far are also valid for this case. The only difference is that we do not force $y(t)$ to fulfill the underlying ODE at t_1 and t_{n-1} anymore. Instead, we satisfy the BCs $\dot{y}_0$ and $\dot{y}_n$ by replacing the first and last row of $\boldsymbol{A}_{\text{coll}}$ and $\boldsymbol{B}_{\text{coll}}$ with

$$\begin{aligned} \boldsymbol{A}_{\text{coll},1} &= \nabla_{\boldsymbol{\lambda}} \dot{s}_0(\xi_0 = 0) \,, & B_{\text{coll},1} &= \dot{y}_0 - \nabla_{\boldsymbol{r}} \dot{s}_0(\xi_0 = 0)\boldsymbol{r} \,, \\ \boldsymbol{A}_{\text{coll},n-1} &= \nabla_{\boldsymbol{\lambda}} \dot{s}_{n-1}(\xi_{n-1} = 1) \,, & B_{\text{coll},n-1} &= \dot{y}_n - \nabla_{\boldsymbol{r}} \dot{s}_{n-1}(\xi_{n-1} = 1)\boldsymbol{r} \,. \end{aligned} \tag{61}$$

In this way the resulting cubic spline fulfills all BCs given in (4), satisfies the underlying ODE at the specified collocation sites, and is $\mathcal{C}^2$ continuous at the interior spline knots (which include $t_{\text{virt},1}$ and $t_{\text{virt},2}$).

2.9. Algorithm for Cubic/Quintic Spline Collocation

We summarize our findings in Algorithm 2. Note that, for the case of cubic splines, the modification necessary to satisfy first-order BCs is already integrated.

Algorithm 2: Cubic/Quintic Spline Collocation.

Input: Collocation parameters consisting of

- starttime t_0 and endtime t_n with $t_0 < t_n$ defining the boundaries of the spline,
- $n-1$ distinct and ascending collocation sites $\{t_k\}$ with $t_0 < t_k < t_n$,
- coefficients $\alpha(t_k)$, $\beta(t_k)$, $\gamma(t_k)$, and right-hand side $\tau(t_k)$ of the underlying ODE at the collocation sites,
- BCs $\{y_0, \dot{y}_0, \ddot{y}_0, y_n, \dot{y}_n, \ddot{y}_n\}$ at t_0 and t_n

where $k = 1, \ldots, n-1$ and $n > 1$.

Output: Cubic/quintic spline, consisting of $n = n_{\text{orig}} + 2$ (cubic) or $n = n_{\text{orig}}$ (quintic) interconnected segments in PP form, which

- satisfies the underlying ODE (1) at the given collocation sites,
- is $\mathcal{C}^2$ (cubic) or $\mathcal{C}^4$ (quintic) continuous at interior spline knots,
- satisfies the BCs.

begin

Cubic: Compute virtual control points using (60) and remap indices $n \to n+2$

Cubic: Setup ${}_{\text{C}}L_i$, ${}_{\text{C}}D_i$, ${}_{\text{C}}U_i$, $(\partial_{\text{C}}B_i/\partial_{\text{C}}\boldsymbol{\lambda})$, and $(\partial_{\text{C}}B_i/\partial_{\text{C}}\boldsymbol{r})$ from (21), (22), and (A16)
Quintic: Setup ${}_{\text{Q}}\boldsymbol{L}_i$, ${}_{\text{Q}}\boldsymbol{D}_i$, ${}_{\text{Q}}\boldsymbol{U}_i$, $(\partial_{\text{Q}}\boldsymbol{B}_i/\partial_{\text{Q}}\boldsymbol{\lambda})$, and $(\partial_{\text{Q}}\boldsymbol{B}_i/\partial_{\text{Q}}\boldsymbol{r})$ from (35), (36), (A31), and (A32)

Solve (48) for $(\partial\boldsymbol{X}/\partial\boldsymbol{\lambda})$ and $(\partial\boldsymbol{X}/\partial\boldsymbol{r})$ through the recursive scheme given in (25), (26), and (27) (cubic) or (40), (41), and (42) (quintic) while using $(\partial\boldsymbol{B}/\partial\boldsymbol{\lambda})$, and $(\partial\boldsymbol{B}/\partial\boldsymbol{r})$ instead of $\boldsymbol{B}$ as right-hand sides
(Note that the recursive scheme allows block-wise operations on the right-hand side, i.e., computing all columns of $(\partial\boldsymbol{X}/\partial\boldsymbol{\lambda})$ and $(\partial\boldsymbol{X}/\partial\boldsymbol{r})$ in parallel. For quintic splines this can be done very efficiently through *LU* decomposition of $({}_{\text{Q}}\boldsymbol{D}_i - {}_{\text{Q}}\boldsymbol{L}_i\,{}_{\text{Q}}\boldsymbol{H}_{i-1})$.)

Compute $(\partial\boldsymbol{\eta}/\partial\boldsymbol{\lambda})$ and $(\partial\boldsymbol{\eta}/\partial\boldsymbol{r})$ from $\boldsymbol{S}^{-1}\,(\partial\boldsymbol{X}/\partial\boldsymbol{\lambda})$ and $\boldsymbol{S}^{-1}\,(\partial\boldsymbol{X}/\partial\boldsymbol{r})$

Compute the spline gradients $\nabla_{\boldsymbol{\lambda}} s_k(0)$, $\nabla_{\boldsymbol{\lambda}} \dot{s}_k(0)$, $\nabla_{\boldsymbol{\lambda}} \ddot{s}_k(0)$, $\nabla_{\boldsymbol{r}} s_k(0)$, $\nabla_{\boldsymbol{r}} \dot{s}_k(0)$, and $\nabla_{\boldsymbol{r}} \ddot{s}_k(0)$ from (51), (53), (54) and Appendix B for all collocation sites
Cubic: Additionally compute the spline gradients $\nabla_{\boldsymbol{\lambda}} \dot{s}_0(0)$, $\nabla_{\boldsymbol{r}} \dot{s}_0(0)$, $\nabla_{\boldsymbol{\lambda}} \dot{s}_{n-1}(1)$, and $\nabla_{\boldsymbol{r}} \dot{s}_{n-1}(1)$

Assemble $\boldsymbol{A}_{\text{coll}}$ and $\boldsymbol{B}_{\text{coll}}$ using (57), (58), (59) (and (61) in cubic case) and solve (56) for $\boldsymbol{\lambda}$

Compute $\boldsymbol{\eta}$ from (47)

Cubic: Extract y_i from ${}_{\text{C}}\boldsymbol{\lambda}$ and $\ddot{y}_i$ from ${}_{\text{C}}\boldsymbol{\eta}$ and compute ${}_{\text{C}}a_i$, ${}_{\text{C}}b_i$, ${}_{\text{C}}c_i$, and ${}_{\text{C}}d_i$ using (17)
Quintic: Extract y_i from ${}_{\text{Q}}\boldsymbol{\lambda}$ and $\dot{y}_i$, $\ddot{y}_i$ from ${}_{\text{Q}}\boldsymbol{\eta}$ and compute ${}_{\text{Q}}a_i$, ${}_{\text{Q}}b_i$, ${}_{\text{Q}}c_i$, ${}_{\text{Q}}d_i$, ${}_{\text{Q}}e_i$, and ${}_{\text{Q}}f_i$ using (30)

end

3. Results

3.1. Implementation

Algorithm 2 delivers a detailed description of the proposed collocation method which can be used as a reference during implementation. However, we also provide a fully documented C++ implementation through our free and open-source library BROCCOLI [34] (see Supplementary

Materials). For best usability, the library is designed to be header-only and uses the (also header-only) cross-platform linear algebra system EIGEN [35] as sole dependency. Although BROCCOLI also has other dependencies, only EIGEN is required for the functionality discussed in this paper. Aside from basic matrix and vector operations, we use EIGEN to solve dense linear systems of equations such as $\boldsymbol{A}_{\text{coll}}^{-1}\,\boldsymbol{B}_{\text{coll}}$ in (56), but also ${}_{\text{Q}}\boldsymbol{D}_1^{-1}(\cdot)$ and $({}_{\text{Q}}\boldsymbol{D}_i - {}_{\text{Q}}\boldsymbol{L}_i\,{}_{\text{Q}}\boldsymbol{H}_{i-1})^{-1}(\cdot)$ in (40) and (41). In particular, we use a Householder rank-revealing QR decomposition with column-pivoting, see `ColPivHouseholderQR` in EIGEN, since it provides the best trade-off between speed, accuracy, and robustness for our use case. Note that for large scale systems solving $\boldsymbol{A}_{\text{coll}}^{-1}\,\boldsymbol{B}_{\text{coll}}$ turns out to be the bottleneck, cf. Figure 11. Thus, in this case one might choose a parallel solver for (56) instead, for example `PartialPivLU` in EIGEN with OPENMP enabled. In addition to the variants `CubicSplineCollocator` and `QuinticSplineCollocator` for spline collocation, see `ode` module of BROCCOLI, cubic and quintic spline interpolation, see `curve` module of BROCCOLI, as specified in Algorithm 1, is also implemented. An overview over the structure of the source code related to our collocation method is given in Appendix C. Lastly, unit tests, similar to the evaluation presented in the following, are shipped with BROCCOLI. These can be used as example applications.

3.2. Test System

In order to evaluate the proposed method and its variants, a simple mass-spring-damper system is considered, see Figure 5 (left). For the sake of simplicity, we do not consider external excitation, i.e., $\tau(t) := 0$. The corresponding ODE (1) describing the system dynamics simplifies to

$$\alpha\,\ddot{F}(t) + \beta\,\dot{F}(t) + \gamma\,F(t) = 0 \tag{62}$$

where α, β, and γ are constants representing the mass and (linear) damping/stiffness coefficients of the system, respectively.

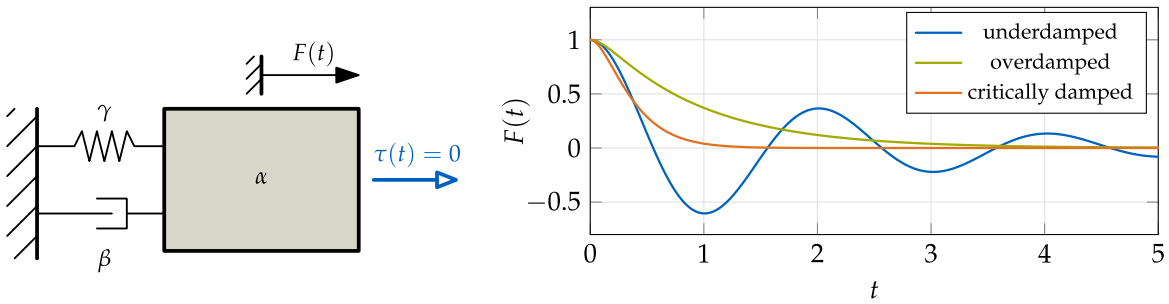

Figure 5. Left: mass-spring-damper system used for validation. Right: analytical solution for the underdamped case ($\alpha = 1$, $\beta = 1$, $\gamma = 10$), the overdamped case ($\alpha = 1$, $\beta = 10$, $\gamma = 10$) and the critically damped case ($\alpha = 1$, $\beta = 10$, $\gamma = 25$). The solution is plotted for the initial conditions $F_0 = 1$, $\dot{F}_0 = 0$.

Using textbook mathematics, we can find the analytical solution given by

$$F(t) = \begin{cases} e^{\sigma_1 t}\left[F_0\,\cos(\sigma_2 t) + \left(\dfrac{\dot{F}_0 - F_0\,\sigma_1}{\sigma_2}\right)\sin(\sigma_2 t)\right] & \text{for } \dfrac{\beta^2}{4\,\alpha^2} - \dfrac{\gamma}{\alpha} < 0 \quad \text{(underdamped)} \\ \left(\dfrac{\dot{F}_0 - F_0\,(\sigma_1 - \sigma_2)}{2\,\sigma_2}\right) e^{(\sigma_1+\sigma_2)t} + \left(\dfrac{F_0\,(\sigma_1 + \sigma_2) - \dot{F}_0}{2\,\sigma_2}\right) e^{(\sigma_1-\sigma_2)t} & \text{for } \dfrac{\beta^2}{4\,\alpha^2} - \dfrac{\gamma}{\alpha} > 0 \quad \text{(overdamped)} \\ e^{\sigma_1 t}\left[F_0 + \left(\dot{F}_0 - F_0\,\sigma_1\right) t\right] & \text{for } \dfrac{\beta^2}{4\,\alpha^2} - \dfrac{\gamma}{\alpha} = 0 \quad \text{(critically damped)} \end{cases}, \tag{63}$$

where we used the initial conditions $F(t_0 = 0) = F_0$ and $\dot{F}(t_0 = 0) = \dot{F}_0$, and the abbreviations

$$\sigma_1 := -\frac{\beta}{2\,\alpha} \quad \text{and} \quad \sigma_2 := \sqrt{\left|\frac{\beta^2}{4\,\alpha^2} - \frac{\gamma}{\alpha}\right|}\,. \tag{64}$$

The characteristic shape of each branch of (63) is visualized in Figure 5 (right) for the parametrization $\alpha = 1$, $\beta = 1$, $\gamma = 10$ in the underdamped case, $\alpha = 1$, $\beta = 10$, $\gamma = 10$ in the overdamped case, and $\alpha = 1$, $\beta = 10$, $\gamma = 25$ in the critically damped case. For the rest of this paper we will adhere to this parametrization. Moreover, we assume the initial conditions to be given with $F_0 = 1$, $\dot{F}_0 = 0$. Note that by choosing $\sigma_1 < 0$ we obtain asymptotically stable behavior. As we never constrained the underlying ODE to be stable, our algorithm can also be used to approximate instable systems. Since tests with $\beta = -1$ showed results comparable to the underdamped case (with $\beta = 1$), we omit an explicit discussion of this case for brevity. Lastly, we point out that although (62) describes a very simple system, we also successfully applied the proposed algorithm to the much more complex walking-pattern generation system of our humanoid robot LOLA, cf. [13]. However, the properties and characteristics of the proposed method can be better investigated and explained by means of a less complex test system.

3.3. Convergence for Consistent and Inconsistent Boundary Conditions

As already mentioned, we focus on over-determined BVPs. Thus, we consider two cases for evaluating our algorithm. For the first analysis, we use the initial conditions $y_0 = F_0 = 1$, $\dot{y}_0 = \dot{F}_0 = 0$ and correspondingly $\ddot{y}_0 = \ddot{F}_0 = -\gamma/\alpha$, see (62), together with the analytical solution given in (63) to compute the BCs $y_n = F_n$, $\dot{y}_n = \dot{F}_n$, and $\ddot{y}_n = \ddot{F}_n$ at $t_n = 5$. In this way, the BCs are guaranteed to be consistent because they belong to the same analytic solution $F(t)$. In the second case, we keep the initial BCs unchanged, but force $y_n = \dot{y}_n = \ddot{y}_n = 0$ for $t_n = 5$ which deviates from the previous solution F_n, $\dot{F}_n$, and $\ddot{F}_n$, especially in the underdamped case, as can be clearly seen in Figure 5 right. Thus, the second analysis handles inconsistent BCs.

In the following, we only focus on the underdamped and overdamped case, since the critically damped case can be seen as a special form of these with $\sigma_2 \to 0$. By evaluating both cases, we aim at covering oscillating and non-oscillating dynamics. Moreover, we run tests for different counts of collocation sites $\nu := |\{t_k\}|$, where we use a uniform segmentation of the spline, i.e., $h_i = h_{i+1} \forall\, i$. For cubic and quintic spline collocation without virtual control points $\nu = n - 2$ holds. In contrast, $\nu = n - 4$ holds for cubic spline collocation with virtual control points. Note that an inhomogeneous partitioning $h_i \neq h_{i+1}$ is evaluated per default in the unit tests provided with BROCCOLI. The approximation of the BVP by spline collocation with consistent BCs and $\nu = 1, \ldots, 100$ is depicted in Figure 6. Note that for cubic spline collocation without virtual control points, the BCs $\dot{y}_0$ and $\dot{y}_n$ are violated (see Section 2.7).

For the case of consistent BCs, we observe that the approximation $y(t)$ indeed converges for increasing ν to the analytical solution $F(t)$. From a qualitative point of view, the convergence order using a quintic spline (Figure 6 right column) is clearly higher than the corresponding cubic counterparts (Figure 6 left and center column). In order to compare convergence using a quantitative measure, we use the root mean square (RMS) of the approximation error $e(t)$ and of the residual $r(t)$, defined as

$$\mathrm{RMS}(e) := \sqrt{\frac{1}{t_n - t_0}\int_{t_0}^{t_n} [e(t)]^2\,\mathrm{d}t} \qquad \text{with} \quad e(t) := F(t) - y(t)\,, \tag{65}$$

$$\mathrm{RMS}(r) := \sqrt{\frac{1}{t_n - t_0}\int_{t_0}^{t_n} [r(t)]^2\,\mathrm{d}t} \qquad \text{with} \quad r(t) := \alpha\,\ddot{y}(t) + \beta\,\dot{y}(t) + \gamma\,y(t)\,. \tag{66}$$

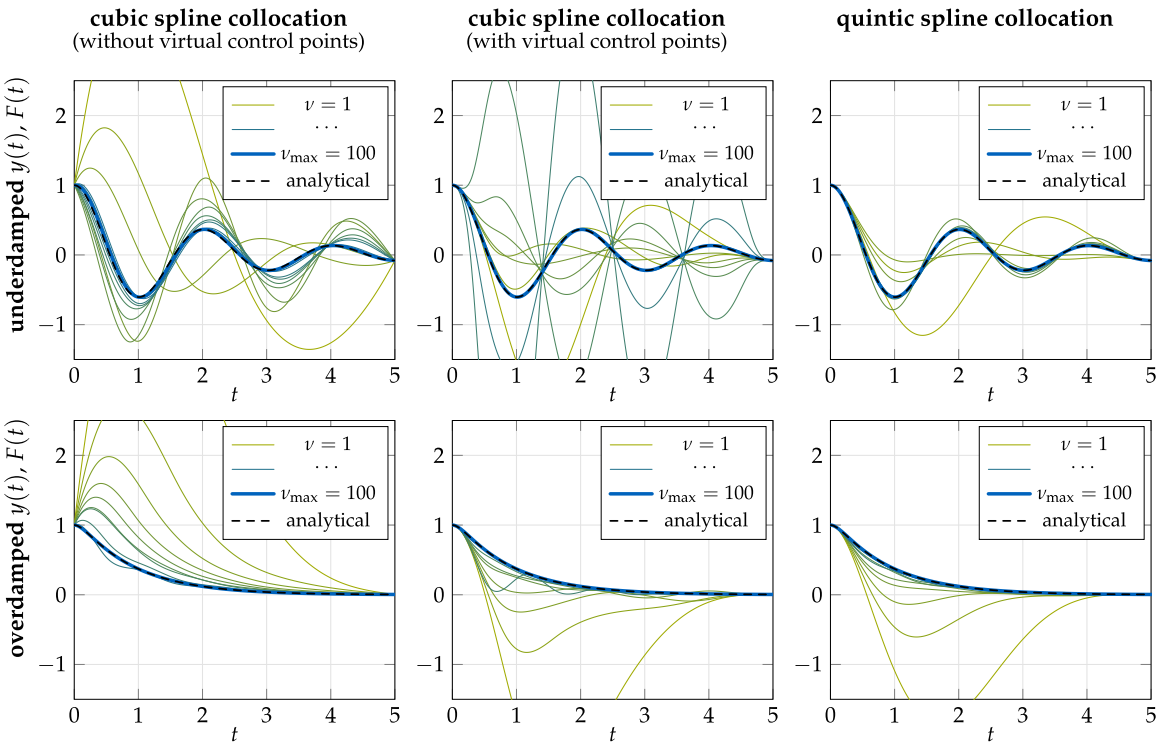

Figure 6. Consistent BCs: convergence of the approximation $y(t)$ (blue and green) towards the analytic solution $F(t)$ (black, dashed) for $\nu = 1, \ldots, 9, 30, 50, 70, 100$. The top row belongs to the underdamped case while the bottom row represents the overdamped case. From left to right: approximation with cubic spline without virtual control points (**left**), cubic spline with virtual control points (**center**) and quintic spline (**right**). The corresponding best approximation $\nu_{\max}$ is drawn in bold blue.

For numerical evaluation of (65) and (66), we discretize the embedded integral using a time step size of $\Delta t = 0.01$ for which we obtain the results presented in Figure 7 left. Since we stop the test series at $\nu_{\max} = 100$, we find the optimum for all variants to be at $\nu_{\text{opt}} = \nu_{\max}$. However, due to convergence, we expect the theoretical optimum to be at $\nu_{\text{opt}} \to \infty$. We observe that also from an quantitative point of view, quintic spline collocation clearly outperforms the cubic variants for the same ν. Furthermore, forcing first-order BCs through virtual control points of the cubic spline seems to have a negative influence, especially for increasing ν. The influence of virtual control points on the residual is visualized in Figure 8 where peaks in $r(t)$ occur at these spots. Note that without virtual control points, the first-order boundary conditions at $t_0 = 0$ and $t_n = 5$ are missed. Thus, in contrast to the collocation sites, the residual does not drop to zero at the boundaries. By comparing the left and right plot of Figure 8, we observe that $r(t)$ behaves similarly for consistent and inconsistent BCs.

For consistent BCs, we can state that, at least for the investigated test system, the proposed algorithm leads to an approximation which converges to the real solution where the "speed" of convergence depends on the chosen variant of Algorithm 2. For inconsistent BCs, however, our analysis draws a different picture: while we can find an optimal count of collocation sites ν_{opt} for each variant and test case, the approximation diverges for $\nu \to \infty$, see Figure 7 right and Figure 9. Note that this was expected since we are attempting to find an approximation of a solution which does not actually exist. For small ν, we still find a "reasonable" $y(t)$ where the spline is still able to smooth out the wrong BCs. However, as we refine the segmentation of the spline, it gets harder to compensate the error, which in turn leads to undesired oscillations of $y(t)$. Note that although we call this behavior divergence, our collocation algorithm still satisfies all constraints that we defined: using a given partitioning, it fulfills the BCs and satisfies the underlying ODE at the given collocation sites. Thus, the divergence is not a fault of the proposed algorithm, but rather is due to the non-existence of the solution $F(t)$. Moreover, the sensitivity to undesired oscillations depends heavily on the underlying BVP: for our

target application of real-time motion planning, we did not observe any oscillations up to $\nu = 10^4$. The critical value for ν is expected to be even higher. However, we could not determine it due to memory limitations of our test system.

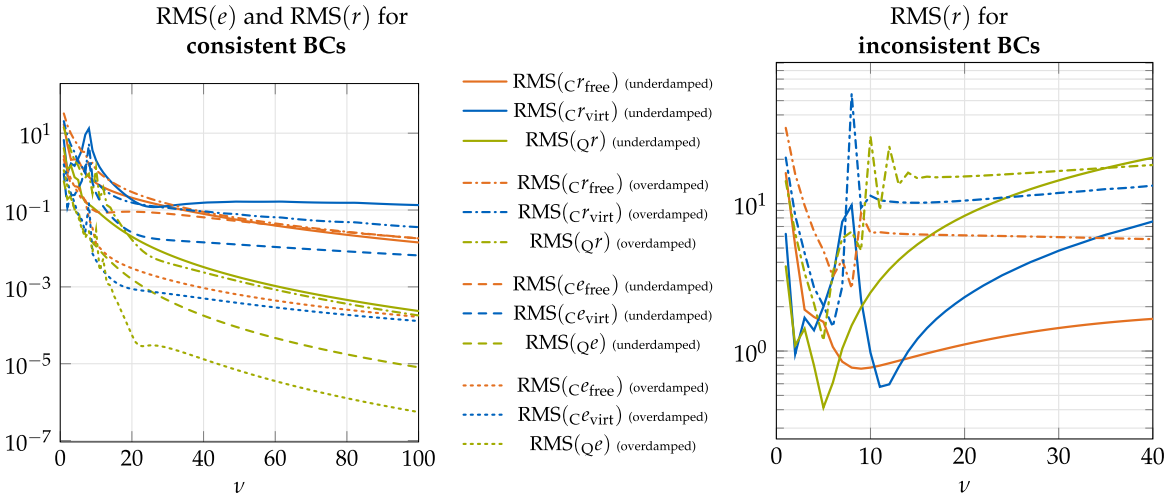

Figure 7. Root mean square (RMS) of error $e(t)$ and residual $r(t)$ as defined in (65) and (66). The left subscript differs between cubic C and quintic Q spline collocation. Moreover, for cubic spline collocation, we identify the variants without virtual control points (i.e., free first-order boundaries $\dot{y}_0$, $\dot{y}_n$) and with virtual control points by the right subscript free and virt, respectively. The left plot belongs to the case of consistent BCs while the right one was obtained using inconsistent BCs. Note that for inconsistent BCs an analytic solution does not exist, thus the error $e(t)$ is not defined and we consider only the residual $r(t)$.

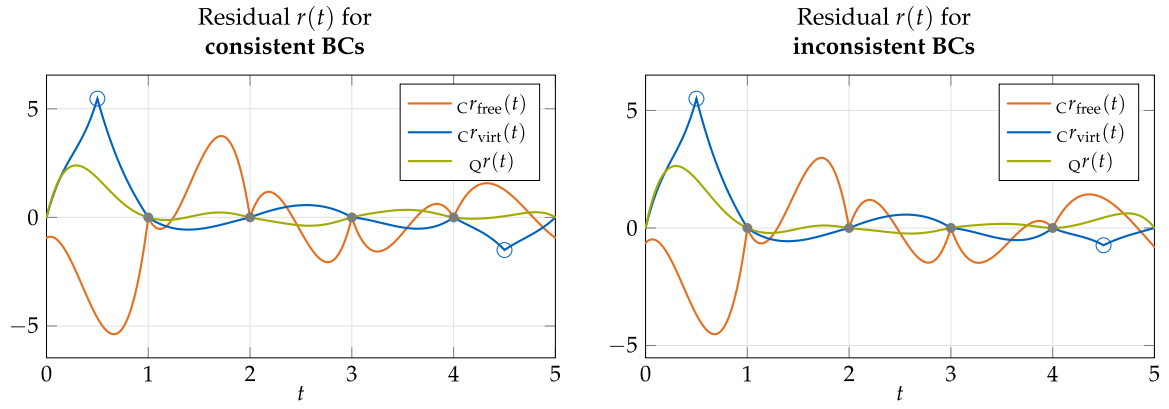

Figure 8. Residual $r(t)$ as defined in (66) for consistent (left) and inconsistent (right) BCs in the underdamped case. For best presentation, the count of collocation sites is chosen as $\nu = 4$ such that the collocation sites (dots) are given by $\{t_k\} = \{1, 2, 3, 4\}$. For cubic spline collocation, the left subscript C and for the quintic counterpart Q is used. Moreover, the right subscripts free and virt are used to differentiate between the variants without and with virtual control points, respectively. The virtual control points are highlighted with circles.

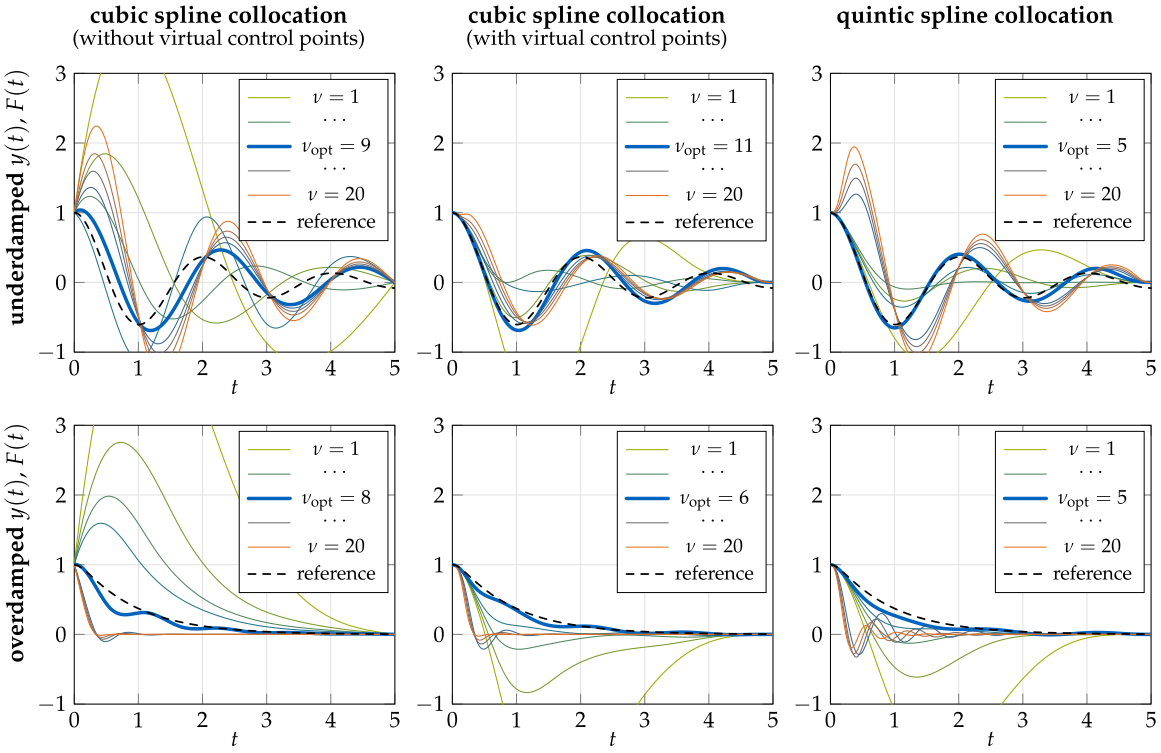

Figure 9. Inconsistent BCs: approximation $y(t)$ (blue, green, and orange) and reference system $F(t)$ (black, dashed) for $\nu = 1, \ldots, 4, \nu_{\text{opt}}, 13, 15, 17, 20$. The top row belongs to the underdamped case while the bottom row represents the overdamped case. From left to right: approximation with cubic spline (no virtual control points, left), cubic spline (with virtual control points, center) and quintic spline (right). The corresponding best approximation ν_{opt} is drawn in bold blue. Diverging approximations for $\nu > \nu_{\text{opt}}$ are colored orange.

In order to avoid undesired oscillations, an optimal ν has to be chosen, which seems to be difficult at the first sight, since in practice we do not know the analytical solution, and thus cannot evaluate $e(t)$. However, one can use the residual $r(t)$ as measure for the approximation error and use this to formulate a governing optimization for finding ν_{opt}. Moreover, one can also use a non-uniform segmentation to give specific sections more weight. Such adaptive techniques for automatic mesh refinement have also been developed for other collocation methods, see, for example, [36]. However, for our application it is sufficient to choose h_i once (fixed), thus we withhold this idea for future investigations.

3.4. Runtime Analysis

In the following, we present measurements, which have been made by using the implementation given in BROCCOLI with the version primo (v1.0.0 - commit 89280c69). We used an AMD Ryzen 7 1700X 8x (16x) @3.40GHz with 32GB DDR4 RAM @2133MHz as hardware backend and Ubuntu 18.04.2 LTS 64bit (Linux kernel 4.15.0-51) together with Clang (version 6.0.0-1) on optimization level 3 as software basis. Although our algorithm is sequential, we run different test cases on four physical cores of the CPU in parallel. For all tests simultaneous multi-threading (SMT) was disabled in the BIOS. For runtime evaluation, we take 1000 samples for every code section in Algorithm 2 and choose the minimum execution time as reference to minimize the risk of wrong measurements due to high system load and context switching effects.

In Figure 10 (left), we present runtime measurements for all three variants of our algorithm. We restrict our analysis to the case of an underdamped parametrization and consistent BCs since we expect comparable results for the other test cases. Since our algorithm has a deterministic runtime which only depends on the chosen variant (cubic/quintic, with/without virtual control points) and ν, there is

no reason why it should be faster or slower with another parametrization. We observe that quintic spline collocation is more expensive than the cubic counterparts for the same ν, which complies with theory since the (block-)tridiagonal system for quintic splines is twice the size of the tridiagonal system for cubic splines. However, for increasing ν, the gap becomes smaller since solving the collocation equations, where $\boldsymbol{A}_{\text{coll}}$ is of the same dimension for cubic and quintic splines, becomes the bottleneck, see Figure 11. Note that there is only a small difference in runtime between cubic spline collocation with and without virtual control points. This also complies with theory, since the only difference is two additional spline knots, which increases the dimension of $\boldsymbol{A}_{\text{coll}}$ by two. Lastly, the small ripples in Figure 10 (left) are due to vectorization and SIMD optimizations handled by EIGEN and the compiler, which give slightly better performance if the dimension of arrays in memory are a multiple of two.

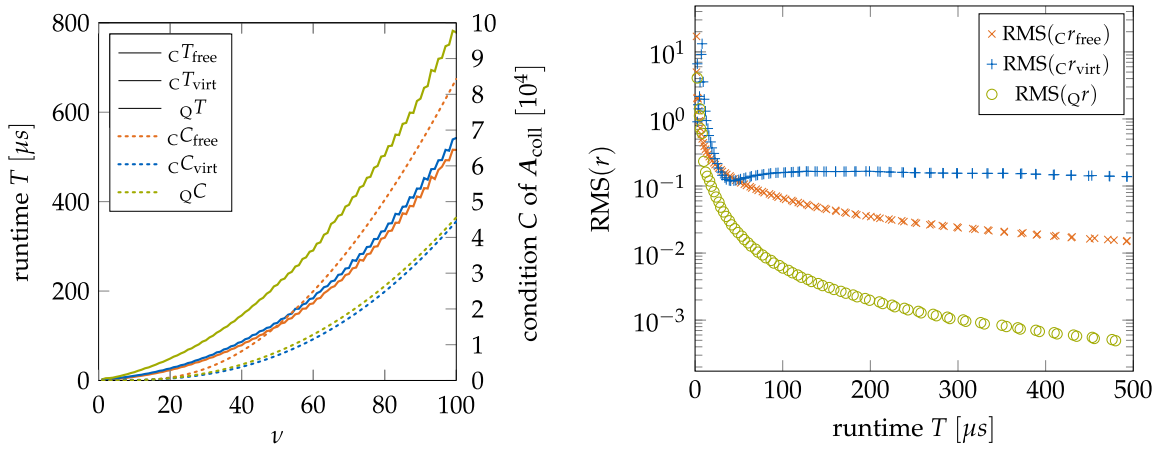

Figure 10. Left: (minimum) runtime T and condition C of $\boldsymbol{A}_{\text{coll}}$ for running Algorithm 2 over count of collocation sites ν. Right: root mean square (RMS) of residual $r(t)$ as defined in (66) over (minimum) runtime T. The left subscript C and Q belong to the cubic or quintic spline version of the algorithm, respectively. Moreover, the right subscript indicates if cubic spline collocation was performed without (free) or with (virt) virtual control points. All measurements were performed using the underdamped parametrization and consistent BCs.

Comparing runtimes for the same ν might not be a meaningful basis for choosing a method. Instead, one is typically interested in getting the best approximation in the shortest time. For this purpose, the RMS of the residual $r(t)$ is plotted over runtime in Figure 10 (right). As can be seen, quintic spline collocation significantly outperforms both other variants. Moreover, bad convergence of cubic spline collocation with virtual control points is also visible in this comparison. In addition to measuring total runtime, we also performed a detailed analysis on the relative cost of each code section of Algorithm 2 during quintic spline collocation. Figure 11 demonstrates that in the vicinity of $\nu \approx 160$, evaluating the block-tridiagonal systems to enforce BCs and continuity, and solving $\boldsymbol{A}_{\text{coll}}\,\boldsymbol{\lambda} = \boldsymbol{B}_{\text{coll}}$ for actual collocation share approximately the same portion of total runtime. With increasing ν, solving for $\boldsymbol{\lambda}$ becomes more relevant since the corresponding system is dense while the block-tridiagonal LSE can be solved efficiently using the recursive scheme discussed in Section 2.3.

Lastly, we want to point out that the condition of $\boldsymbol{A}_{\text{coll}}$ gets worse for increasing count of collocation sites ν, see Figure 10 left. Thus, there might be an upper limit of ν for the proposed algorithm.

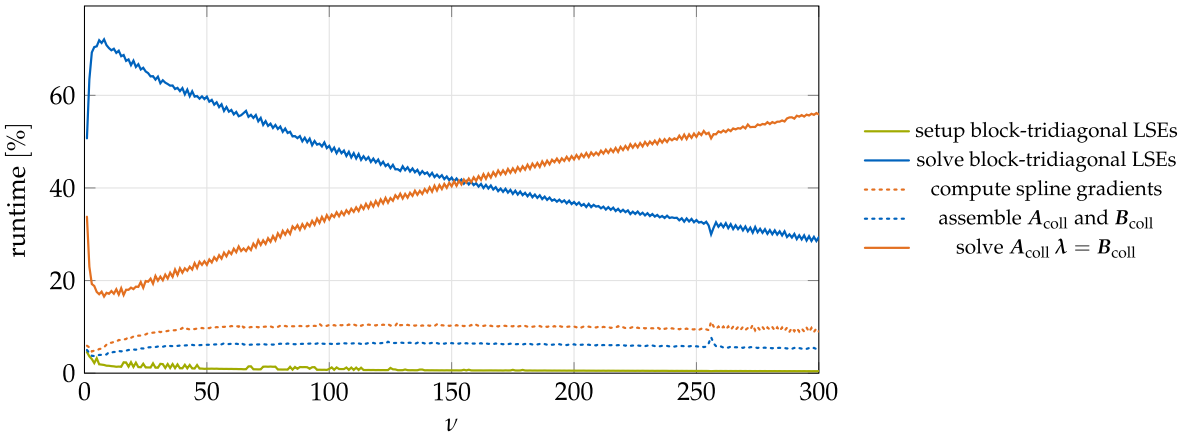

Figure 11. Runtime (percentile) of relevant steps of Algorithm 2 relative to total runtime for quintic spline collocation. Code sections with negligible execution time are not plotted and also not accounted for total runtime. The measurements were obtained by using an extended time horizon $t_n = 50$ and the parametrization $\alpha = 1$, $\beta = 0.1$, $\gamma = 10$ (underdamped) to allow a better representation of high counts of collocation sites of up to $\nu = 300$.

4. Discussion

As shown in the previous section, the presented algorithm performs well if the BCs are consistent and fully known. Even in the case of inconsistent BCs, we still obtain a reasonable approximation as long as we carefully pick the collocation sites. However, if we exceed the optimum, undesired oscillations may occur, which is an indicator for putting too much emphasis on satisfying the underlying ODE while simultaneously trying to compensate the "broken" BCs. In order to automatically determine an optimal partitioning of the spline, a higher level optimization may be applied, which will be the focus of further investigations.

Obviously, if the investigated BVP is well-posed, i.e., if it is not over-determined, other techniques should be preferred over our approach. However, for applications where the enforcement of certain BCs is more important than approximating the underlying dynamics, the proposed method seems to be a valid approach. Moreover, we want to emphasize that no variable recursion or iteration is involved. This makes execution time predictable, which is especially relevant for real-time applications such as in our use case.

Comparing different variants of our algorithm showed that collocation using a quintic spline is in general superior to using the somewhat simpler cubic splines. Note that although its derivation is more involved, the final implementation is of approximately the same complexity, since virtual control points have to be introduced in the cubic case. At this point, we also want to emphasize that, based on the results of our study, we do not recommend to use cubic spline collocation with virtual control points. Although the full set of BCs is satisfied, the additional knots seem to significantly downgrade convergence. However, other choices of $t_{\text{virt},1}$ and $t_{\text{virt},2}$ may lead to different results.

Within this contribution, we only considered second-order BVPs. An extension to ODEs of higher order seems to be straightforward, since only the collocation equations, see (58) and (59) have to be extended by the corresponding gradients while the overall dimension of $\boldsymbol{A}_{\text{coll}}$ and $\boldsymbol{B}_{\text{coll}}$ stays the same. Moreover, our approach may be applied to nonlinear systems as well, by using the common approach of linearization and embedding the scheme into a Newton iteration.

Lastly, we want to highlight that our focus lies on runtime efficiency. However, for embedded systems especially, memory consumption may also be a limiting factor. Although we expect our algorithm to have similar requirements when compared to other techniques, we have not looked into this issue so far.

Supplementary Materials: An archived version of the free and open-source C++ header-only library BROCCOLI in the version *primo* (v1.0.0) which includes the proposed algorithm is available online at http://www.mdpi.com/2218-6581/xx/1/5/s1. For the most recent version of BROCCOLI visit https://gitlab.lrz.de/AM/broccoli.

Author Contributions: Conceptualization, P.S.; Data curation, P.S.; Formal analysis, P.S.; Funding acquisition, D.J.R.; Investigation, P.S.; Methodology, P.S.; Project administration, D.J.R.; Resources, P.S.; Software, P.S.; Supervision, D.J.R.; Validation, P.S.; Visualization, P.S.; Writing—original draft, P.S.; Writing—review & editing, P.S. and D.J.R. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the German Research Foundation (DFG), project number 407378162. Moreover, this work was supported by the DFG and the Technical University of Munich (TUM) in the framework of the Open Access Publishing Program.

Acknowledgments: We would like to express special thanks to Nora-Sophie Staufenberg and Felix Sygulla for the constructive discussions during development, implementation and analysis of the presented method.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

BVP	Boundary Value Problem
IVP	Initial Value Problem
PDE	Partial Differential Equation
ODE	Ordinary Differential Equation
DAE	Differential Algebraic Equation
BC	Boundary Condition
CoM	Center of Mass
ABD	Almost Block Diagonal
LSE	Linear System of Equations
PP	Piecewise Polynomial
RMS	Root Mean Square
SMT	Simultaneous Multi-Threading
DFG	German Research Foundation (Deutsche Forschungsgemeinschaft)
TUM	Technical University of Munich

Appendix A. Considerations on Numerical Stability

Appendix A.1. Cubic Spline Interpolation

The Thomas algorithm is guaranteed to be numerically stable if ${}_{\mathrm{C}}\boldsymbol{A}$ is diagonally dominant, i.e., if

$$|{}_{\mathrm{C}}D_i| = |2\,(h_{i-1} + h_i)| > |{}_{\mathrm{C}}L_i| + |{}_{\mathrm{C}}U_i| = |h_{i-1}| + |h_i| \quad \text{for } i = 1, \ldots, n-1 \tag{A1}$$

holds [30], whereas for $i = 1$ and $i = n-1$ we use ${}_{\mathrm{C}}L_1 = 0$ and ${}_{\mathrm{C}}U_{n-1} = 0$, respectively. As is easily verified with (A1), this holds true for any choice of $h_{i-1} > 0$ and $h_i > 0$ (i.e., for distinct and ascending interpolation sites), thus the presented method is always stable.

Appendix A.2. Quintic Spline Interpolation

The presented scheme for solving block-tridiagonal systems is a special form of block-Gaussian elimination without pivoting and is guaranteed to be numerically stable if ${}_{\mathrm{Q}}\boldsymbol{A}$ is block-diagonally dominant, i.e., if

$$\Gamma_i := \left\| {}_{\mathrm{Q}}\boldsymbol{D}_i^{-1} \right\| \left(\|{}_{\mathrm{Q}}\boldsymbol{L}_i\| + \|{}_{\mathrm{Q}}\boldsymbol{U}_i\| \right) \leq 1 \quad \text{for } i = 1, \ldots, n-1 \tag{A2}$$

holds for an arbitrary matrix norm $\|\cdot\|$ [37]. Note that for $i = 1$ and $i = n-1$ we set ${}_{\mathrm{Q}}\boldsymbol{L}_1 = \boldsymbol{0}$ and ${}_{\mathrm{Q}}\boldsymbol{U}_{n-1} = \boldsymbol{0}$, respectively. In order to verify (A2) for ${}_{\mathrm{Q}}\boldsymbol{D}_i$, ${}_{\mathrm{Q}}\boldsymbol{L}_i$, and ${}_{\mathrm{Q}}\boldsymbol{U}_i$ as specified in (35) and (36), we assume a constant ratio $\omega = g_i / g_{i-1} = g_{i+1} / g_i$ to obtain

$$ {}_{\mathrm{Q}}\boldsymbol{D}_i = g_i^3 \underbrace{\begin{bmatrix} 64\left(\frac{1}{\omega^3}+1\right) & 12\,\kappa\left(\frac{1}{\omega^2}-1\right) \\ 12\,\kappa\left(\frac{1}{\omega^2}-1\right) & 3\,\kappa^2\left(\frac{1}{\omega}+1\right) \end{bmatrix}}_{=:{}_{\mathrm{Q}}\hat{\boldsymbol{D}}_i(\omega)}, \; {}_{\mathrm{Q}}\boldsymbol{L}_i = g_i^3 \underbrace{\begin{bmatrix} 56\,\frac{1}{\omega^3} & -8\,\kappa\,\frac{1}{\omega^3} \\ 8\,\kappa\,\frac{1}{\omega^2} & -\kappa^2\,\frac{1}{\omega^2} \end{bmatrix}}_{=:{}_{\mathrm{Q}}\hat{\boldsymbol{L}}_i(\omega)}, \; {}_{\mathrm{Q}}\boldsymbol{U}_i = g_i^3 \underbrace{\begin{bmatrix} 56 & 8\,\kappa\,\omega \\ -8\,\kappa & -\kappa^2\,\omega \end{bmatrix}}_{=:{}_{\mathrm{Q}}\hat{\boldsymbol{U}}_i(\omega)} \tag{A3} $$

and further

$$ \Gamma_i(\omega) = \left\| {}_{\mathrm{Q}}\boldsymbol{D}_i^{-1}(\omega) \right\| \left(\| {}_{\mathrm{Q}}\boldsymbol{L}_i(\omega) \| + \| {}_{\mathrm{Q}}\boldsymbol{U}_i(\omega) \| \right) = \left\| {}_{\mathrm{Q}}\hat{\boldsymbol{D}}_i^{-1}(\omega) \right\| \left(\left\| {}_{\mathrm{Q}}\hat{\boldsymbol{L}}_i(\omega) \right\| + \left\| {}_{\mathrm{Q}}\hat{\boldsymbol{U}}_i(\omega) \right\| \right) \tag{A4} $$

which shows that Γ_i does not depend on g_i for a constant ratio ω. It can be easily verified through numerical evaluation of (A4) that a homogeneous partitioning of the spline, i.e., $\omega = 1$, results in the lowest value for Γ_i. Using the spectral matrix norm and the special choice of κ given in (39) leads to

$$ \Gamma_i(\omega = 1) \approx 1.24 \not\leq 1 \quad \text{for } i = 2, \ldots, n-2\,. \tag{A5} $$

Unfortunately, condition (A2) is violated, even for the ideal case of a homogeneously partitioned spline. However, (A2) represents a sufficient but not necessary condition for numerical stability, thus we can still use Γ_i as a measure for the pivotal growth [37] which should be minimized. In doing so, we can conclude that for best numerical stability one should use a "reasonable" ratio ω which is close to 1.

Note that the special choice of κ in (39) has been suggested in [18] to minimize Γ_i and thus optimize numerical stability. This intention becomes clear when observing that

$$ \left\| {}_{\mathrm{Q}}\hat{\boldsymbol{D}}_i^{-1}(\omega = 1) \right\|_2 = \sqrt{\lambda_{\max}\left({}_{\mathrm{Q}}\hat{\boldsymbol{D}}_i^{-T}(\omega = 1)\, {}_{\mathrm{Q}}\hat{\boldsymbol{D}}_i^{-1}(\omega = 1) \right)} = \begin{cases} \dfrac{1}{6\,\kappa^2} > \dfrac{1}{128} & \text{for } \kappa^2 < \dfrac{64}{3}\,, \\ \dfrac{1}{128} & \text{for } \kappa^2 \geq \dfrac{64}{3} \end{cases} $$

where $\lambda_{\max}(\dots)$ denotes the maximum eigenvalue of a given matrix. Since both $\left\| {}_{\mathrm{Q}}\hat{\boldsymbol{L}}_i(\omega) \right\|_2$ and $\left\| {}_{\mathrm{Q}}\hat{\boldsymbol{U}}_i(\omega) \right\|_2$, increase with growing $|\kappa|$, the optimum (39) is chosen. This finding can be easily verified through numerical investigation. Note that numerical stability only depends on $|\kappa|$, thus one could also choose the negative form $\kappa = -\sqrt{64/3}$.

Appendix B. Spline Gradients

In the following, explicit expressions for the spline gradients used in (51), (53), and (54) are given. Note that the gradients differ depending on the type of the underlying spline (cubic/quintic).

Appendix B.1. Cubic Spline Gradients

From (7), (8), and (9) we obtain for each variable ${}_{\mathrm{C}}\boldsymbol{\rho} \in \{ {}_{\mathrm{C}}\boldsymbol{\lambda}, {}_{\mathrm{C}}\boldsymbol{\eta}, {}_{\mathrm{C}}\boldsymbol{r} \}$ and each segment ${}_{\mathrm{C}}s_i$ with $i = 0, \ldots, n-1$

$$ \left(\frac{\partial {}_{\mathrm{C}}s_i(\xi_i)}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right) = \left(\frac{\partial {}_{\mathrm{C}}a_i}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right) \xi_i^3 + \left(\frac{\partial {}_{\mathrm{C}}b_i}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right) \xi_i^2 + \left(\frac{\partial {}_{\mathrm{C}}c_i}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right) \xi_i + \left(\frac{\partial {}_{\mathrm{C}}d_i}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right)\,, \tag{A6} $$

$$ \left(\frac{\partial {}_{\mathrm{C}}\dot{s}_i(\xi_i)}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right) = \frac{3}{h_i} \left(\frac{\partial {}_{\mathrm{C}}a_i}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right) \xi_i^2 + \frac{2}{h_i} \left(\frac{\partial {}_{\mathrm{C}}b_i}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right) \xi_i + \frac{1}{h_i} \left(\frac{\partial {}_{\mathrm{C}}c_i}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right)\,, \tag{A7} $$

$$ \left(\frac{\partial {}_{\mathrm{C}}\ddot{s}_i(\xi_i)}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right) = \frac{6}{h_i^2} \left(\frac{\partial {}_{\mathrm{C}}a_i}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right) \xi_i + \frac{2}{h_i^2} \left(\frac{\partial {}_{\mathrm{C}}b_i}{\partial {}_{\mathrm{C}}\boldsymbol{\rho}} \right)\,. \tag{A8} $$

Moreover, by using the indices $u = 1, \ldots, n-1$, $v = 1, \ldots, n-1$, and $w = 1, \ldots, 4$ to specify the elements of

$$\begin{aligned} {}_{\mathrm{C}}\boldsymbol{\lambda} &= [{}_{\mathrm{C}}\lambda_1, \ldots, {}_{\mathrm{C}}\lambda_u, \ldots, {}_{\mathrm{C}}\lambda_{n-1}]^T &&= [y_1, \ldots, y_u, \ldots, y_{n-1}]^T \\ {}_{\mathrm{C}}\boldsymbol{\eta} &= [{}_{\mathrm{C}}\eta_1, \ldots, {}_{\mathrm{C}}\eta_v, \ldots, {}_{\mathrm{C}}\eta_{n-1}]^T &&= [\ddot{y}_1, \ldots, \ddot{y}_v, \ldots, \ddot{y}_{n-1}]^T \\ {}_{\mathrm{C}}\boldsymbol{r} &= [{}_{\mathrm{C}}r_1, \ldots, {}_{\mathrm{C}}r_w, \ldots, {}_{\mathrm{C}}r_4]^T &&= [y_0, \ddot{y}_0, y_n, \ddot{y}_n]^T \end{aligned} \tag{A9}$$

we obtain

$$\left(\frac{\partial_{\mathrm{C}} a_i}{\partial_{\mathrm{C}}\boldsymbol{\lambda}}\right) = \mathbf{0}\,, \left(\frac{\partial_{\mathrm{C}} b_i}{\partial_{\mathrm{C}}\boldsymbol{\lambda}}\right) = \mathbf{0}\,, \left(\frac{\partial_{\mathrm{C}} c_i}{\partial_{\mathrm{C}}\lambda_u}\right) = \begin{cases} -1 & \text{for } u = i\,, \\ 1 & \text{for } u = i+1\,, \\ 0 & \text{else} \end{cases} \quad \left(\frac{\partial_{\mathrm{C}} d_i}{\partial_{\mathrm{C}}\lambda_u}\right) = \begin{cases} 1 & \text{for } u = i\,, \\ 0 & \text{else} \end{cases}\,, \tag{A10}$$

$$\left(\frac{\partial_{\mathrm{C}} a_i}{\partial_{\mathrm{C}}\eta_v}\right) = \begin{cases} -\frac{h_i^2}{6} & \text{for } v = i\,, \\ \frac{h_i^2}{6} & \text{for } v = i+1\,, \\ 0 & \text{else} \end{cases}\,, \qquad \left(\frac{\partial_{\mathrm{C}} c_i}{\partial_{\mathrm{C}}\eta_v}\right) = \begin{cases} -\frac{h_i^2}{3} & \text{for } v = i\,, \\ -\frac{h_i^2}{6} & \text{for } v = i+1\,, \\ 0 & \text{else} \end{cases} \tag{A11}$$

$$\left(\frac{\partial_{\mathrm{C}} b_i}{\partial_{\mathrm{C}}\eta_v}\right) = \begin{cases} \frac{h_i^2}{2} & \text{for } v = i\,, \\ 0 & \text{else} \end{cases}\,, \qquad \left(\frac{\partial_{\mathrm{C}} d_i}{\partial_{\mathrm{C}}\boldsymbol{\eta}}\right) = \mathbf{0}\,, \tag{A12}$$

$$\left(\frac{\partial_{\mathrm{C}} a_i}{\partial_{\mathrm{C}} r_w}\right) = \begin{cases} -\frac{h_i^2}{6} & \text{for } i = 0 \wedge w = 2\,, \\ \frac{h_i^2}{6} & \text{for } i = n-1 \wedge w = 4\,, \\ 0 & \text{else} \end{cases}\,, \qquad \left(\frac{\partial_{\mathrm{C}} c_i}{\partial_{\mathrm{C}} r_w}\right) = \begin{cases} -1 & \text{for } i = 0 \wedge w = 1\,, \\ -\frac{h_i^2}{3} & \text{for } i = 0 \wedge w = 2\,, \\ 1 & \text{for } i = n-1 \wedge w = 3\,, \\ -\frac{h_i^2}{6} & \text{for } i = n-1 \wedge w = 4\,, \\ 0 & \text{else} \end{cases}\,, \tag{A13}$$

$$\left(\frac{\partial_{\mathrm{C}} b_i}{\partial_{\mathrm{C}} r_w}\right) = \begin{cases} \frac{h_i^2}{2} & \text{for } i = 0 \wedge w = 2\,, \\ 0 & \text{else} \end{cases}\,, \qquad \left(\frac{\partial_{\mathrm{C}} d_i}{\partial_{\mathrm{C}} r_w}\right) = \begin{cases} 1 & \text{for } i = 0 \wedge w = 1\,, \\ 0 & \text{else} \end{cases}\,, \tag{A14}$$

which can easily be derived from (17) and the definition of ${}_{\mathrm{C}}\boldsymbol{\lambda}$, ${}_{\mathrm{C}}\boldsymbol{\eta}$, and ${}_{\mathrm{C}}\boldsymbol{r}$. Note that for $\xi_i = 0$ the computation of

$$\left(\frac{\partial_{\mathrm{C}} a_i}{\partial_{\mathrm{C}}\boldsymbol{\lambda}}\right)\,, \left(\frac{\partial_{\mathrm{C}} a_i}{\partial_{\mathrm{C}}\boldsymbol{\eta}}\right)\,, \text{ and } \left(\frac{\partial_{\mathrm{C}} a_i}{\partial_{\mathrm{C}}\boldsymbol{r}}\right) \tag{A15}$$

can be skipped entirely since up to the second time derivative of the gradients of $s_i(\xi_i = 0)$ these terms are multiplied with zero and have no effect anyway. In order to solve (48), we have to further compute the corresponding right-hand sides which are given by

$$\left(\frac{\partial_{\mathrm{C}} B_i}{\partial_{\mathrm{C}}\lambda_u}\right) = \begin{cases} \frac{6}{h_{i-1}} & \text{for } u = i-1\,, \\ -\frac{6}{h_i} - \frac{6}{h_{i-1}} & \text{for } u = i\,, \\ \frac{6}{h_i} & \text{for } u = i+1\,, \\ 0 & \text{else} \end{cases}\,, \qquad \left(\frac{\partial_{\mathrm{C}} B_i}{\partial_{\mathrm{C}} r_w}\right) = \begin{cases} \frac{6}{h_{i-1}} & \text{for } i = 1 \wedge w = 1\,, \\ -h_{i-1} & \text{for } i = 1 \wedge w = 2\,, \\ \frac{6}{h_i} & \text{for } i = n-1 \wedge w = 3\,, \\ -h_i & \text{for } i = n-1 \wedge w = 4\,, \\ 0 & \text{else} \end{cases} \tag{A16}$$

for $i = 1, \ldots, n-1$, where we used (23), (24), and the definition of ${}_{\mathrm{C}}\boldsymbol{\lambda}$ and ${}_{\mathrm{C}}\boldsymbol{r}$.

Appendix B.2. Quintic Spline Gradients

From (10), (11), and (12) we obtain for each variable ${}_{\mathrm{Q}}\boldsymbol{\rho} \in \{{}_{\mathrm{Q}}\boldsymbol{\lambda}, {}_{\mathrm{Q}}\boldsymbol{\eta}, {}_{\mathrm{Q}}\boldsymbol{r}\}$ and each segment ${}_{\mathrm{Q}}s_i$ with $i = 0, \ldots, n-1$

$$\left(\frac{\partial {}_{\mathrm{Q}}s_i(\xi_i)}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right) = \left(\frac{\partial {}_{\mathrm{Q}}a_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i^5 + \left(\frac{\partial {}_{\mathrm{Q}}b_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i^4 + \left(\frac{\partial {}_{\mathrm{Q}}c_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i^3 + \left(\frac{\partial {}_{\mathrm{Q}}d_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i^2 + \left(\frac{\partial {}_{\mathrm{Q}}e_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i + \left(\frac{\partial {}_{\mathrm{Q}}f_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right), \tag{A17}$$

$$\left(\frac{\partial {}_{\mathrm{Q}}\dot{s}_i(\xi_i)}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right) = \frac{5}{h_i}\left(\frac{\partial {}_{\mathrm{Q}}a_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i^4 + \frac{4}{h_i}\left(\frac{\partial {}_{\mathrm{Q}}b_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i^3 + \frac{3}{h_i}\left(\frac{\partial {}_{\mathrm{Q}}c_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i^2 + \frac{2}{h_i}\left(\frac{\partial {}_{\mathrm{Q}}d_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i + \frac{1}{h_i}\left(\frac{\partial {}_{\mathrm{Q}}e_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right), \tag{A18}$$

$$\left(\frac{\partial {}_{\mathrm{Q}}\ddot{s}_i(\xi_i)}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right) = \frac{20}{h_i^2}\left(\frac{\partial {}_{\mathrm{Q}}a_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i^3 + \frac{12}{h_i^2}\left(\frac{\partial {}_{\mathrm{Q}}b_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i^2 + \frac{6}{h_i^2}\left(\frac{\partial {}_{\mathrm{Q}}c_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right)\xi_i + \frac{2}{h_i^2}\left(\frac{\partial {}_{\mathrm{Q}}d_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\rho}}\right). \tag{A19}$$

Moreover, by using the indices $u = 1, \ldots, n-1$, $v = 1, \ldots, 2(n-1)$, and $w = 1, \ldots, 4$ to specify the elements of

$$\begin{aligned} {}_{\mathrm{Q}}\boldsymbol{\lambda} &= [{}_{\mathrm{Q}}\lambda_1, \ldots, {}_{\mathrm{Q}}\lambda_u, \ldots, {}_{\mathrm{Q}}\lambda_{n-1}]^T &&= [y_1, \ldots, y_u, \ldots, y_{n-1}]^T \\ {}_{\mathrm{Q}}\boldsymbol{\eta} &= [{}_{\mathrm{Q}}\eta_1, \ldots, {}_{\mathrm{Q}}\eta_v, \ldots, {}_{\mathrm{Q}}\eta_{2(n-1)}]^T &&= [\dot{y}_1, \ddot{y}_1, \ldots, \dot{y}_{(v+1)/2}, \ddot{y}_{v/2}, \ldots, \dot{y}_{n-1}, \ddot{y}_{n-1}]^T \\ {}_{\mathrm{Q}}\boldsymbol{r} &= [{}_{\mathrm{Q}}r_1, \ldots, {}_{\mathrm{Q}}r_w, \ldots, {}_{\mathrm{Q}}r_6]^T &&= [y_0, \dot{y}_0, \ddot{y}_0, y_n, \dot{y}_n, \ddot{y}_n]^T \end{aligned} \tag{A20}$$

we obtain

$$\left(\frac{\partial {}_{\mathrm{Q}}a_i}{\partial {}_{\mathrm{Q}}\lambda_u}\right) = \begin{cases} -6 & \text{for } u = i, \\ 6 & \text{for } u = i+1, \\ 0 & \text{else} \end{cases}, \qquad \left(\frac{\partial {}_{\mathrm{Q}}d_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\lambda}}\right) = \mathbf{0}, \tag{A21}$$

$$\left(\frac{\partial {}_{\mathrm{Q}}b_i}{\partial {}_{\mathrm{Q}}\lambda_u}\right) = \begin{cases} 15 & \text{for } u = i, \\ -15 & \text{for } u = i+1, \\ 0 & \text{else} \end{cases}, \qquad \left(\frac{\partial {}_{\mathrm{Q}}e_i}{\partial {}_{\mathrm{Q}}\boldsymbol{\lambda}}\right) = \mathbf{0}, \tag{A22}$$

$$\left(\frac{\partial {}_{\mathrm{Q}}c_i}{\partial {}_{\mathrm{Q}}\lambda_u}\right) = \begin{cases} -10 & \text{for } u = i, \\ 10 & \text{for } u = i+1, \\ 0 & \text{else} \end{cases}, \qquad \left(\frac{\partial {}_{\mathrm{Q}}f_i}{\partial {}_{\mathrm{Q}}\lambda_u}\right) = \begin{cases} 1 & \text{for } u = i, \\ 0 & \text{else} \end{cases}, \tag{A23}$$

$$\left(\frac{\partial {}_{\mathrm{Q}}a_i}{\partial {}_{\mathrm{Q}}\eta_v}\right) = \begin{cases} -3\,h_i & \text{for } v = 2\,i-1, \\ -\frac{h_i^2}{2} & \text{for } v = 2\,i, \\ -3\,h_i & \text{for } v = 2\,i+1, \\ \frac{h_i^2}{2} & \text{for } v = 2\,i+2, \\ 0 & \text{else} \end{cases}, \qquad \left(\frac{\partial {}_{\mathrm{Q}}d_i}{\partial {}_{\mathrm{Q}}\eta_v}\right) = \begin{cases} \frac{h_i^2}{2} & \text{for } v = 2\,i, \\ 0 & \text{else} \end{cases}, \tag{A24}$$

$$\left(\frac{\partial {}_{\mathrm{Q}}b_i}{\partial {}_{\mathrm{Q}}\eta_v}\right) = \begin{cases} 8\,h_i & \text{for } v = 2\,i-1, \\ \frac{3\,h_i^2}{2} & \text{for } v = 2\,i, \\ 7\,h_i & \text{for } v = 2\,i+1, \\ -h_i^2 & \text{for } v = 2\,i+2, \\ 0 & \text{else} \end{cases}, \qquad \left(\frac{\partial {}_{\mathrm{Q}}e_i}{\partial {}_{\mathrm{Q}}\eta_v}\right) = \begin{cases} h_i & \text{for } v = 2\,i-1, \\ 0 & \text{else} \end{cases}, \tag{A25}$$

$$\left(\frac{\partial_{\mathrm{Q}} c_i}{\partial_{\mathrm{Q}} \eta_v}\right) = \begin{cases} -6\,h_i & \text{for } v = 2\,i-1\,, \\ -\frac{3\,h_i^2}{2} & \text{for } v = 2\,i\,, \\ -4\,h_i & \text{for } v = 2\,i+1\,, \\ \frac{h_i^2}{2} & \text{for } v = 2\,i+2\,, \\ 0 & \text{else} \end{cases}, \qquad \left(\frac{\partial_{\mathrm{Q}} f_i}{\partial_{\mathrm{Q}} \boldsymbol{\eta}}\right) = \mathbf{0}\,, \tag{A26}$$

$$\left(\frac{\partial_{\mathrm{Q}} a_i}{\partial_{\mathrm{Q}} r_w}\right) = \begin{cases} -6 & \text{for } i = 0 \wedge w = 1\,, \\ -3\,h_i & \text{for } i = 0 \wedge w = 2\,, \\ -\frac{h_i^2}{2} & \text{for } i = 0 \wedge w = 3\,, \\ 6 & \text{for } i = n-1 \wedge w = 4\,, \\ -3\,h_i & \text{for } i = n-1 \wedge w = 5\,, \\ \frac{h_i^2}{2} & \text{for } i = n-1 \wedge w = 6\,, \\ 0 & \text{else} \end{cases}, \quad \left(\frac{\partial_{\mathrm{Q}} d_i}{\partial_{\mathrm{Q}} r_w}\right) = \begin{cases} \frac{h_i^2}{2} & \text{for } i = 0 \wedge w = 3\,, \\ 0 & \text{else} \end{cases}, \tag{A27}$$

$$\left(\frac{\partial_{\mathrm{Q}} b_i}{\partial_{\mathrm{Q}} r_w}\right) = \begin{cases} 15 & \text{for } i = 0 \wedge w = 1\,, \\ 8\,h_i & \text{for } i = 0 \wedge w = 2\,, \\ \frac{3\,h_i^2}{2} & \text{for } i = 0 \wedge w = 3\,, \\ -15 & \text{for } i = n-1 \wedge w = 4\,, \\ 7\,h_i & \text{for } i = n-1 \wedge w = 5\,, \\ -h_i^2 & \text{for } i = n-1 \wedge w = 6\,, \\ 0 & \text{else} \end{cases}, \quad \left(\frac{\partial_{\mathrm{Q}} e_i}{\partial_{\mathrm{Q}} r_w}\right) = \begin{cases} h_i & \text{for } i = 0 \wedge w = 2\,, \\ 0 & \text{else} \end{cases}, \tag{A28}$$

$$\left(\frac{\partial_{\mathrm{Q}} c_i}{\partial_{\mathrm{Q}} r_w}\right) = \begin{cases} -10 & \text{for } i = 0 \wedge w = 1\,, \\ -6\,h_i & \text{for } i = 0 \wedge w = 2\,, \\ -\frac{3\,h_i^2}{2} & \text{for } i = 0 \wedge w = 3\,, \\ 10 & \text{for } i = n-1 \wedge w = 4\,, \\ -4\,h_i & \text{for } i = n-1 \wedge w = 5\,, \\ \frac{h_i^2}{2} & \text{for } i = n-1 \wedge w = 6\,, \\ 0 & \text{else} \end{cases}, \quad \left(\frac{\partial_{\mathrm{Q}} f_i}{\partial_{\mathrm{Q}} r_w}\right) = \begin{cases} 1 & \text{for } i = 0 \wedge w = 1\,, \\ 0 & \text{else} \end{cases}, \tag{A29}$$

which can easily be derived from (30) and the definition of ${}_{\mathrm{Q}}\boldsymbol{\lambda}$, ${}_{\mathrm{Q}}\boldsymbol{\eta}$, and ${}_{\mathrm{Q}}\boldsymbol{r}$. Note that for $\xi_i = 0$ the computation of

$$\left(\frac{\partial_{\mathrm{Q}} a_i}{\partial_{\mathrm{Q}} \lambda}\right), \left(\frac{\partial_{\mathrm{Q}} a_i}{\partial_{\mathrm{Q}} \boldsymbol{\eta}}\right), \left(\frac{\partial_{\mathrm{Q}} a_i}{\partial_{\mathrm{Q}} \boldsymbol{r}}\right), \left(\frac{\partial_{\mathrm{Q}} b_i}{\partial_{\mathrm{Q}} \lambda}\right), \left(\frac{\partial_{\mathrm{Q}} b_i}{\partial_{\mathrm{Q}} \boldsymbol{\eta}}\right), \left(\frac{\partial_{\mathrm{Q}} b_i}{\partial_{\mathrm{Q}} \boldsymbol{r}}\right), \left(\frac{\partial_{\mathrm{Q}} c_i}{\partial_{\mathrm{Q}} \lambda}\right), \left(\frac{\partial_{\mathrm{Q}} c_i}{\partial_{\mathrm{Q}} \boldsymbol{\eta}}\right), \left(\frac{\partial_{\mathrm{Q}} c_i}{\partial_{\mathrm{Q}} \boldsymbol{r}}\right), \tag{A30}$$

can be skipped entirely since up to the second time derivative of the gradients of $s_i(\xi_i = 0)$ these terms are multiplied with zero and have no effect anyway. In order to solve (48), we further have to compute the corresponding right-hand sides, which are given by

$$\left(\frac{\partial_{\mathrm{Q}} \boldsymbol{B}_i}{\partial_{\mathrm{Q}} \lambda_u}\right) = \begin{cases} \begin{bmatrix} 120\,g_{i-1}^4 \\ 20\,\kappa\,g_{i-1}^3\,g_i \end{bmatrix} & \text{for } u = i-1\,, \quad \begin{bmatrix} -120\,g_i^4 \\ 20\,\kappa\,g_i^4 \end{bmatrix} \quad \text{for } u = i+1\,, \\ \begin{bmatrix} 120\left(g_i^4 - g_{i-1}^4\right) \\ -20\,\kappa\,g_i\left(g_i^3 + g_{i-1}^3\right) \end{bmatrix} & \text{for } u = i\,, \qquad \mathbf{0} \qquad \text{else} \end{cases}, \tag{A31}$$

$$\left(\frac{\partial_{\mathrm{Q}}\boldsymbol{B}_i}{\partial_{\mathrm{Q}} r_w}\right) = \begin{cases} \begin{bmatrix} 120\, g_{i-1}^4 \\ 20\,\kappa\, g_{i-1}^3\, g_i \end{bmatrix} & \text{for } i = 1 \wedge w = 1\,, \quad \begin{bmatrix} -120\, g_i^4 \\ 20\,\kappa\, g_i^4 \end{bmatrix} \quad \text{for } i = n-1 \wedge w = 4\,, \\ \begin{bmatrix} 56\, g_{i-1}^3 \\ 8\,\kappa\, g_{i-1}^2\, g_i \end{bmatrix} & \text{for } i = 1 \wedge w = 2\,, \quad \begin{bmatrix} 56\, g_i^3 \\ -8\,\kappa\, g_i^3 \end{bmatrix} \quad \text{for } i = n-1 \wedge w = 5\,, \\ \begin{bmatrix} 8\, g_{i-1}^2 \\ \kappa\, g_{i-1}\, g_i \end{bmatrix} & \text{for } i = 1 \wedge w = 3\,, \quad \begin{bmatrix} -8\, g_i^2 \\ \kappa\, g_i^2 \end{bmatrix} \quad \text{for } i = n-1 \wedge w = 6\,, \\ \mathbf{0} & \text{else} \end{cases} \tag{A32}$$

for $i = 1, \ldots, n-1$, where we used (37), (38), and the definition of ${}_{\mathrm{Q}}\boldsymbol{\lambda}$ and ${}_{\mathrm{Q}}\boldsymbol{r}$.

Appendix C. Software Design in BROCCOLI

The basic structure of the source code related to the classes `CubicSplineCollocator` and `QuinticSplineCollocator` is illustrated in Figure A1 (left). Both classes are derived from the abstract base class `SplineCollocatorBase`, which declares the interface for data in-/output and the main processing operators. In order to trigger Algorithm 2 for a given input dataset, a simple call of `process()` is sufficient. For a more fine-grained control, `process()` is split up into publicly available subroutines (names starting with "`substep_`"). Note that providing public access to the subroutines not only allows detailed runtime measurements by the user, but also enables an efficient parallel solution for decoupled, multi-dimensional BVPs, see Figure A1 right. For this, we exploit that large parts of Algorithm 2 only depend on the spline segmentation, i.e., the collocation sites, (green box). In contrast, the remaining subroutines require explicit information about the investigated BVP (blue box). If all dimensions of the BVP, e.g., the x- and y-component of LOLA's CoM motion, share the same spline segmentation, subroutines covered by the green box in Figure A1 have to be called only once. Subsequently, the used instance of `CubicSplineCollocator` or `QuinticSplineCollocator` (holding intermediate results) is copied such that the remaining subroutines, covered by the blue box in Figure A1, can be run in parallel. Note that this includes solving $\boldsymbol{A}_{\mathrm{coll}}^{-1}\,\boldsymbol{B}_{\mathrm{coll}}$ (`substep_solveCollocationLSE()`), which is the bottleneck for large-scale systems. We plan to use this feature in our target application to efficiently generate the CoM motion of LOLA (the x- and y-component represent a decoupled, two-dimensional BVP).

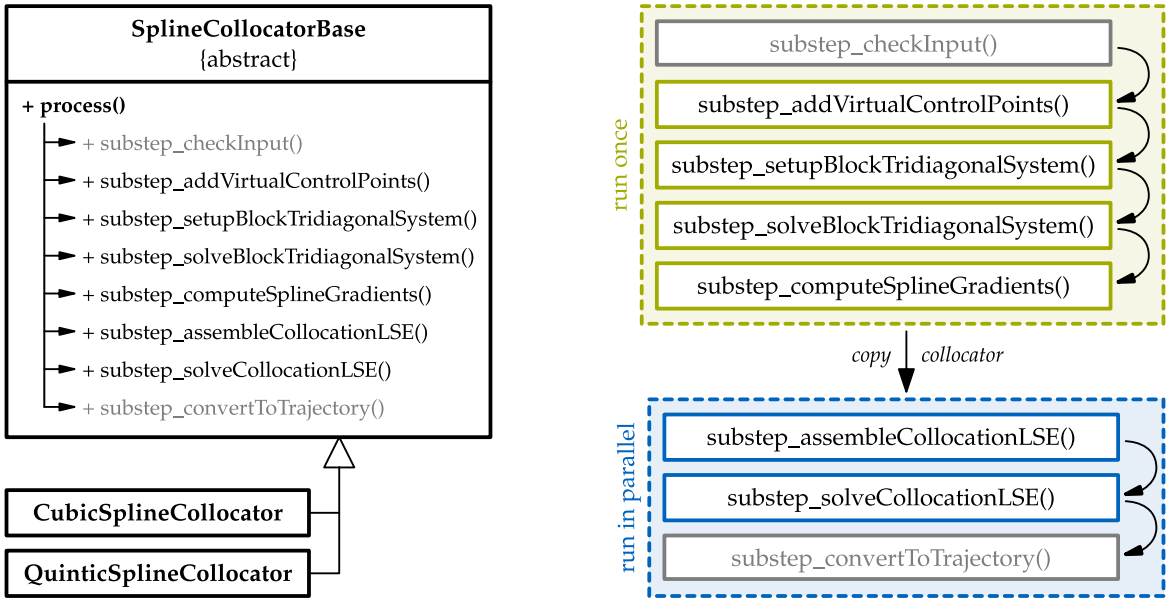

Figure A1. Design of the classes `CubicSplineCollocator` and `QuinticSplineCollocator` in

BROCCOLI. Left: class inheritance and segmentation of `process()` into subroutines. Right: proposed strategy for efficient parallelization in the case of a decoupled, multi-dimensional BVP. Note that the first and last subroutine (grayed out) are optional and perform a validity check of the given input parameters and convert the final result into a corresponding `broccoli::curve::Trajectory` data structure for convenient evaluation of the generated polynomial spline, respectively.

References

1. Lee, J.F.; Lee, R.; Cangellaris, A. Time-Domain Finite-Element Methods. *IEEE Trans. Antennas Propag.* **1997**, *45*, 430–442.
2. Ahlberg, J.H.; Nilson, E.N.; Walsh, J.L. *The Theory of Splines and Their Applications*; Mathematics in Science and Engineering, Academic Press: New York, NY, USA, 1967.
3. de Boor, C.; Swartz, B. Collocation at Gaussian Points. *SIAM J. Numer. Anal.* **1973**, *10*, 582–606. [CrossRef]
4. Ahlberg, J.; Ito, T. A Collocation Method for Two-Point Boundary Value Problems. *Math. Comput.* **1975**, *29*, 761–776. [CrossRef]
5. Christara, C.C.; Ng, K.S. Optimal Quadratic and Cubic Spline Collocation on Nonuniform Partitions. *Computing* **2005**, *76*, 227–257. [CrossRef]
6. Bialecki, B.; Fairweather, G. Orthogonal spline collocation methods for partial differential equations. *J. Comput. Appl. Math.* **2001**, *128*, 55–82. [CrossRef]
7. Houstis, E.N.; Christara, C.C.; Rice, J.R. Quadratic-Spline Collocation Methods for Two-Point Boundary Value Problems. *Int. J. Numer. Methods Eng.* **1988**, *26*, 935–952. [CrossRef]
8. Irodotou-Ellina, M.; Houstis, E.N. An $\mathcal{O}(h^6)$ Quintic Spline Collocation Method for Fourth Order Two-Point Boundary Value Problems. *BIT Numer. Math.* **1988**, *28*, 288–301. [CrossRef]
9. Ascher, U. Solving Boundary-Value Problems with a Spline-Collocation Code. *J. Comput. Phys.* **1980**, *34*, 401–413. [CrossRef]
10. Zhang, H.; Han, X.; Yang, X. Quintic B-spline collocation method for fourth order partial integro-differential equations with a weakly singular kernel. *Appl. Math. Comput.* **2013**, *219*, 6565–6575. [CrossRef]
11. Akram, G.; Tariq, H. Quintic spline collocation method for fractional boundary value problems. *J. Assoc. Arab Univ. Basic Appl. Sci.* **2017**, *23*, 57–65. [CrossRef]
12. Ascher, U.; Spiteri, R. Collocation Software for Boundary Value Differential-Algebraic Equations. *SIAM J. Sci. Comput.* **1995**, *15*. [CrossRef]
13. Seiwald, P.; Sygulla, F.; Staufenberg, N.S.; Rixen, D. Quintic Spline Collocation for Real-Time Biped Walking-Pattern Generation with variable Torso Height. In Proceedings of the IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids), Toronto, ON, Canada, 15–17 October, 2019; pp. 56–63.
14. Seiwald, P. Video: Smooth Real-Time Walking-Pattern Generation for Humanoid Robot LOLA. Available online: https://youtu.be/piQm_oTYXIc (accessed on 22 June 2020).
15. Wright, S.J. Stable Parallel Algorithms for Two-Point Boundary Value Problems. *SIAM J. Sci. Statistical Comput.* **1992**, *13*, 742–764. [CrossRef]
16. Magoulés, F.; Roux, F.X.; Houzeaux, G. *Parallel Scientific Computing*; Wiley & Sons: New York, NY, USA, 2015.
17. Usmani, R.A. Smooth spline approximations for the solution of a boundary value problem with engineering applications. *J. Comput. Appl. Math.* **1980**, *6*, 93–98. [CrossRef]
18. Mund, E.H.; Hallet, P.; Hennart, J.P. An algorithm for the interpolation of functions using quintic splines. *J. Comput. Appl. Math.* **1975**, *1*, 279–288. [CrossRef]
19. Buschmann, T.; Lohmeier, S.; Bachmayer, M.; Ulbrich, H.; Pfeiffer, F. A Collocation Method for Real-Time Walking Pattern Generation. In Proceedings of the IEEE-RAS 7th International Conference on Humanoid Robots, Pittsburgh, PA, USA, 29 November–1 December 2007; pp. 1–6.
20. Ascher, U.; Pruess, S.; Russell, R.D. On Spline Basis Selection for Solving Differential Equations. *SIAM J. Numer. Anal.* **1983**, *20*, 121–142. [CrossRef]
21. de Boor, C. *A Practical Guide to Splines*; Springer: New York, NY, 2001; ISBN:978-0387953663.
22. de Boor, C.; Weiss, R. SOLVEBLOK: A Package for Solving Almost Block Diagonal Linear Systems. *ACM Trans. Math. Software* **1980**, *6*, 80–87. [CrossRef]
23. Majaess, F.; Keast, P.; Fairweather, G. The Packages for solving almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions. In *Scientific Software Systems*; Springer: Dordt, The Netherlands, 1988; pp. 47–58.

24. de Boor, C. On calculating with B-splines. *J. Approximation Theor.* **1972**, *6*, *50–62*. [CrossRef]
25. Böhm, W. Efficient Evaluation of Splines. *Computing* **1984**, *33*, 171–177. [CrossRef]
26. Lee, E.T.Y. A Simplified B-Spline Computation Routine. *Computing* **1982**, *29*, 365–371. [CrossRef]
27. Lee, E.T.Y. Comments on Some B-Spline Algorithms. *Computing* **1986**, *36*, 229–238. [CrossRef]
28. Horner, W.G.; Gilbert, D. *A New Method of Solving Numerical Equations of All Orders, by Continuous Approximation*; Philosophical Transactions of the Royal Society of London: London, UK, 1819; pp. 308–335.
29. Higham, N.J. *Accuracy and Stability of Numerical Algorithms*, 2nd ed.; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2002.
30. Quarteroni, A.; Sacco, R.; Saler, F. *Numerical Mathematics*, 2nd ed.; Springer: Berlin, Germany, 2007.
31. Isaacson, E.; Keller, H.B. *Analysis of Numerical Methods*; Dover Publications, Inc.: New York, NY, USA, 1966.
32. Meurant, G. A Review on the Inverse of Symmetric Tridiagonal and Block Tridiagonal Matrices. *SIAM J. Matrix Anal. Appl.* **1992**, *13*, 707–728. [CrossRef]
33. Buschmann, T. Simulation and Control of Biped Walking Robots. Ph.D. Thesis, Technical University of Munich, Munich, Germany, November 2010. Available online: https://mediatum.ub.tum.de/997204 (accessed on 22 June 2020).
34. Seiwald, P.; Sygulla, F. Broccoli: Beautiful Robot C++ Code Library. Available online: https://gitlab.lrz.de/AM/broccoli (accessed on 22 June 2020).
35. Guennebaud, G.; Jacob, B. Eigen. Available online: http://eigen.tuxfamily.org (accessed on 22 June 2020).
36. Christara, C.C.; Ng, K.S. Adaptive Techniques for Spline Collocation. *Computing* **2006**, *76*, 259–277. [CrossRef]
37. Varah, J.M. On the Solution of Block-Tridiagonal Systems Arising from Certain Finite-Difference Equations. *Math. Comput.* **1972**, *26*, 859–868. [CrossRef]