Title
Spline Deferred Correction Methods for Ordinary Differential Equations

Author
Cheng, Peter

Publication Date
2018

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Spline Deferred Correction Methods for Ordinary Differential Equations

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Mathematics

by

Peter Cheng

2018

ABSTRACT OF THE DISSERTATION

Spline Deferred Correction Methods

for Ordinary Differential Equations

by

Peter Cheng

Doctor of Philosophy in Mathematics

University of California, Los Angeles, 2018

Professor Christopher R. Anderson, Chair

We introduce a new variant of Picard integral deferred correction, a class of methods aimed at solving initial value problems with arbitrarily high order of accuracy, called spline deferred correction (SplineDC). Similar to spectral deferred correction (SpectralDC), SplineDC applies a series of correction sweeps, derived from the Picard integral form, that are driven by either first-order explicit or implicit Euler integration schemes to produce a highly accurate numerical solution. However, as the name implies, SplineDC makes use of spline interpolation as opposed to Lagrange interpolation in the case of SpectralDC. This idea is motivated by our interest in circumventing the inherent restrictions of SpectralDC methods which are enforced to avoid the instabilities associated with high-order and/or equispaced polynomial interpolation, i.e. Runge's phenomenon. In doing so, we develop a class of methods with favorable convergence behavior and excellent stability properties, all of which we demonstrate through various numerical experiments. Additionally, because of the freedom SplineDC has with respect to time step selection, we demonstrate the advantages of implementing adaptive time-stepping in SplineDC over SpectralDC. Finally, we present an alternate method of spline construction that facilitates the ease by which one can construct splines with arbitrary continuity and interpolation conditions.

The dissertation of Peter Cheng is approved.

Jeffrey D. Eldredge

Marcus Leigh Roper

Luminita Aura Vese

Christopher R. Anderson, Committee Chair

University of California, Los Angeles

2018

Table of Contents

List of Figures

List of Tables

Acknowledgments

First, I would like to thank my advisor, Chris Anderson, for all of his support and guidance over the past few years. The eloquence and ease in which you talk about math made me believe I could do it, too. You are the advisor/teacher/professor that I will always aspire to be. I would like to thank my family for their never-ending love and support from the Bay Area. I would like to thank my committee, Jeff Eldredge, Marcus Roper, and Luminita Vese, for their encouragements and wisdom. I would also like to thank the late William Klug, who was the one to initially present a research project I immediately found interesting. Though the direction of my thesis eventually changed, it was because of his research ideas that ultimately set me on this path. Thanks to Martha and Maida for their help with the administrative side of the PhD.

I would like to thank Lindsay for her constant support throughout the majority of my PhD career. We built a life that worked for us, filled with jalapeño cheese pretzels, silly antics, and game nights, and it was an essential part of my growth as a person. Grad school would have been very bleak without you and Jasper. Even though you're now in North Carolina, I will do whatever I can to return the favor in your journey for a PhD. I would like to thank Andre, Will, Bonsoon, Michael, Yacoub, and Denali for their time and support whenever I reached out to them about anything research or TA-related. I would like to thank Zane for all of the hard work he put in with me in order to pass the ADE qualifying exam. Finally, I would like to thank my friends from home for their camaraderie over the years ever since high school.

Vita

2016 M.A. (Mathematics), University of California, Los Angeles

2009 B.A. (Applied Mathematics), University of California, Berkeley

Chapter 1

Introduction

The subject of constructing stable and high-order numerical methods for solving ordinary differential equation initial value problems is one that has been explored quite thorougly [1, 4, 6, 13]. Generally speaking, existing methods fall under one of two categories:

1. those that are intrinsically high-order, and
2. those that aim to obtain high-order accuracy by combinging the result of lower order methods

Examples of numerical methods that fall under the first category include Runge-Kutta and multistep methods. These methods, however, suffer from certain drawbacks as the order of accuracy increases. For non-stiff problems where explicit methods are used, the time step restriction becomes increasingly severe for multistep methods, while the number of function evaluations grows drastically for Runge-Kutta methods. For stiff-problems, implicit Runge-Kutta is extremely expensive (hence the development of singly-implicit or diagonally-implicit Runge-Kutta methods [6]), while high-order implicit multi-step methods suffer from poor stability properties [4]. Examples of numerical methods that fall under the second category include extrapolation and deferred correction methods. However, these too also have drawbacks when high- order versions are required. Extrapolation methods require computation of solutions on finer and finer temporal meshes in order to obtain a solution of the required

order of accuracy, while classical deferred correction methods requires smoothness of the numerical solution since it relies upon repeated application of numerical differentiation.

More recently, a new type of deferred correction method has been proposed and explored: spectral deferred correction (SpectralDC). Introduced by Dutt, Greengard, and Rokhlin in 2000 [10], SpectralDC uses the same idea as classical deferred correction, where an *error equation* is repeatedly solved using a low-order method, and the result is added back to the numerical solution to correct it. Since SpectralDC begins by first recasting the initial value problem into its Picard integral form, this removes the need for numerical differentiation that introduces problems in the standard deferred correction framework. It has been shown in [7, 14, 15, 19] that SpectralDC methods also possess the same convergence properties as classical deferred correction, which is that each correction sweep results in an increase in the order of accuracy of the numerical solution but only at the cost of a low-order method.

Researchers have taken the basic idea of SpectralDC further and derived variants of SpectralDC for a wide variety of initial value problems. For example, Minion introduced the class of semi-implicit SpectralDC methods for initial value problems that contain terms of varying levels of stiffness [19]. Bourlioux et al. introduced the class of multi-implicit SpectralDC methods to address initial value problems that involve processes with widely differing characteristic time scales [3]. These methods are the SpectralDC version of splitting methods. Speck et al. introduced the class of multi-level SpectralDC methods, that aim to decrease the computational cost of the correction sweeps by employing a multigrid-like idea [24]. Huang et al. introduced Krylov spectral deferred correction, an accelerated version of the classic method [16]. More recently, researchers have been incorporating SpectralDC in parareal algorithms, which are parallel-in-time methods for solving initial value problems [12, 20–22].

Despite the success of SpectralDC, it still has its drawbacks. As we will see in more detail in the next chapter, SpectralDC relies implicitly on a Lagrange interpolant of the right-hand side of the initial value problem. This means that the method uses a numerical technique that

may suffer from the problems associated with the use of high-order or equispaced polynomial interpolation, e.g. Runge's phenomenon [5, 11]. Though SpectralDC has been developed with this in mind, there is no guarantee that these problems are fully avoided. Secondly, the quadrature step of SpectralDC can be done relatively quickly because of the ability to precompute the quadrature weights. However, this relies on the idea that the substep nodes of a time step remains in the same relative locations across all time steps. This makes adaptive time-stepping a more expensive to implement in the context of SpectralDC. We should point out, though, that it has been successfully used, see [8, 23].

This dissertation is motivated by addressing the drawbacks inherrent to SpectralDC mentioned above. To this end, we introduce a new class of deferred correction methods we call *spline deferred correction* (SplineDC). These methods replace the implicit Lagrange interpolation of SpectralDC with spline interpolation, and in doing so, frees us from the time-stepping restrictions inherent to SpectralDC. These methods also have diserable stability characteristics. In particular, as the order of accuracy is increased, the associated regions of absolute stability grow in size.

This dissertation is organized into five parts:

Chapter 2: We present, in detail, the derivation of deferred correction methods. This begins with a brief overview of classical deferred correction, and then transitions into a full exposition of Picard integral deferred correction (PIDC). Finally, we introduce SplineDC, a sub-class of PIDC, and discuss the advantages associated with our variant of the broad class of deferred correction methods.

Chapter 3: We provide a proof of convergence for SplineDC methods whose correction sweeps are driven by first-order explicit or implicit Euler integration schemes. The results are analogous to those of SpectralDC. However, we bring to light an integral player in the convergene of these methods, which is the existence of a linear bounded spline operator, a transformation that takes function data to a spline interpolant that satisfies user-specified

continuity and interpolation conditions.

Chapter 4: Stability and accuracy are two important properties of any numerical method, and in this chapter, we focus our efforts in better understanding SplineDC in this context. Through numerical experiements, we demonstrate that the accuracy of SplineDC methods are on par with that of SpectralDC methods of equivalent order. However, we discover that our methods exhibit much more favorable stability properties: given a target order of accuracy, we can construct both explicit and implicit SplineDC methods with increasingly large regions of absolute stability.

Chapter 5: In this chapter, adaptive time-stepping is explored in the context of Picard integral deferred correction methods, where it is used when computing the provisional solution.

Chapter 6: We introduce AltSplines, a new and alternate way of constructing splines. In practice, splines are constructed in one of two ways: the classical way where one finds the coefficients for each piecewise polynomial of the spline (like in the commonly used case of clamped/complete cubic splines), or the B-spline approach where one represents splines as a linear combination of basis splines. The method of AltSplines is motivated by the idea that the continuity and interpolating conditions of a spline can be decoupled. We then present two case studies where the goal is to develop a method for constructing splines without the need to specify extra osculatory conditions in order to close the system of equations for the spline coefficients.

Chapter 2

Deferred Correction

2.1 Introduction

Deferred correction is a broad class of methods for numerically solving initial value problems up to an arbitrary order of accuracy for non-stiff and stiff problems. The fundamental idea is to create and solve an error correction equation based on the residual associated with an approximate solution. Fixing notation, suppose we aim to solve the initial value problem

$$y'(t) = F(t, y(t)) \quad t \in [t_0, T] \tag{2.1.1}$$

$$y(t_0) = y_0. \tag{2.1.2}$$

While the following discussion holds for $y(t)$, $y_0 \in \mathbb{C}^N$ and $F : \mathbb{R} \times \mathbb{C}^N \to \mathbb{C}^N$, we'll assume $N = 1$ for ease of notation unless specified otherwise. Futhermore, though $F \in C^1(\mathbb{R} \times \mathbb{C}^N)$ is sufficient to guarantee local existence and uniqueness, we assume F is sufficiently smooth because the high-order accuracy obtainable through deferred correction methods requires the existence of higher-order derivatives.

2.2 Classical Deferred Correction

Before we discuss Picard integral deferred correction, let us first present the foundational ideas that were introduced with classical deferred correction. Recall that we aim to solve eqs. (2.1.1) and (2.1.2). To this end, suppose we discretize the time interval $[t_0, T]$ using N uniform time steps of size ΔT, i.e.

$$t_i = t_0 + i\Delta T, \quad i = 0, \ldots, N$$

where $\Delta T = (T - t_0)/N$, and we wish to find the values of the solution to eq. (2.1.1) and eq. (2.1.2) at these nodes. Suppose φ_i is a kth-order accurate approximation to $y(t)$, the true solution, at t_i for all i. This allows us to construct the unique Nth degree interpolating polynomial $L^N(t)$ through the points $\{(t_i, \varphi_i)\}_{i=0}^N$. We can then define the error function associated with $L^N(t)$ by

$$\delta(t) = y(t) - L^N(t). \tag{2.2.1}$$

Observe that eq. (2.2.1) satisfies the differential equation

$$\begin{aligned} \delta'(t) &= y'(t) - \frac{d}{dt}L^N(t) \\ &= F\left(t, \delta(t) + L^N(t)\right) - \frac{d}{dt}L^N(t) \\ \delta(0) &= 0. \end{aligned} \tag{2.2.2}$$

We can solve eq. (2.2.2) using the same kth-order method used on the original problem, which yields kth-order accurate approximations $\eta_i \approx \delta(t_i)$ for all i. It is well-known that the corrected approximation

$$\varphi_i + \eta_i \approx y(t_i)$$

is of $(2k)$th order accuracy [2, 10].

Deferred correction proceeds this way in an iterative manner: solve an appropriate initial value problem to obtain the error associated with the current approximation to the solution, and use it to correct the solution. This method can only be iterated if $L^N(t)$ and $\frac{d}{dt}L^N(t)$ are both sufficiently accurate for all $t \in [t_0, T]$. If so, then after S iterations, our approximate solution is of order $\mathcal{O}(h^{\min[(S+1)k, m+1]})$ [2, 10]. The first argument of the minimum function comes from the order we achieve after S kth-order correction iterations, while the second argument is from the order of accuracy of $L^N(t)$.

Though this might seem like a great way for developing high-order accurate methods, there are two things that we need to take note of. First, the use of equispaced nodes for interpolation is numerically ill-conditioned (i.e. Runge's phenomenon). Second, numerical differentiation requires smoothness of data, which we might not necessarily have due to random noise, thus introducing other sources of instability. The problems introduced by interpolation can be reduced by using non- equispaced nodes, such as the Chebyshev nodes [5]. The need for differentiation can be removed by using the Picard integral form of the differential equation, an observation that leads to Picard integral deferred correction methods.

2.3 Picard Integral Deferred Correction

Picard integral deferred correction (PIDC) methods are deferred correction methods that start by first rewriting the initial value problem into its Picard integral formulation. To this end, we rewrite eqs. (2.1.1) and (2.1.2) as

$$y(t) = y_0 + \int_{t_0}^{t} F(\tau, y(\tau))\, d\tau, \quad t \in [t_0, T]. \tag{2.3.1}$$

Furthermore, computing the solution on the entire time interval simply amounts to using the numerical method to advance the solution across time steps in sequential order. Thus

without loss of generality, our derivations are only concerned with obtaining a solution on an arbitrary time step $[t_n, t_{n+1}]$ of the entire time interval of interest. As a result, we will write eq. (2.3.1) as

$$y(t) = y_n + \int_{t_n}^{t} F(\tau, y(\tau))\, d\tau, \quad t \in [t_n, t_{n+1}]. \tag{2.3.2}$$

Let $\varphi^k(t)$ denote the kth corrected solution to eq. (2.3.2). Though our goal is to find the value of the solution at discrete temporal nodes, whether it would be at the endpoints of $[t_n, t_{n+1}]$ or also including substeps in between, we will derive the main steps of SpectralDC assuming a functional form of the numerical solution. First, we define the residual $\epsilon^k(t)$ associated with the kth corrected solution $\varphi^k(t)$ to be

$$\epsilon^k(t) := y_0 + \int_{t_n}^{t} F(\tau, \varphi^k(\tau))\, d\tau - \varphi^k(t). \tag{2.3.3}$$

The choice of method usd to approximate the integral term in eq. (2.3.3) is what will eventually distinguish our spline deferred correction methods from spectral deferred correction methods. The error $\delta^k(t)$ associated with $\varphi^k(t)$ is defined to be

$$\delta^k(t) := y(t) - \varphi^k(t). \tag{2.3.4}$$

Using eqs. (2.3.2) to (2.3.4), we can write

$$\begin{aligned}
\epsilon^k(t) &= y_0 + \int_{t_n}^{t} F(\tau, \varphi^k(\tau))\, d\tau - \varphi^k(t) \\
&= y(t) - \int_{t_n}^{t} F(\tau, y(\tau))\, d\tau + \int_{t_n}^{t} F(\tau, \varphi^k(\tau))\, d\tau - \varphi^k(\tau) \\
&= \delta^k(t) - \int_{t_n}^{t} F(\tau, \varphi^k(\tau) + \delta^k(\tau)) - F(\tau, \varphi^k(\tau))\, d\tau.
\end{aligned}$$

Rearranging terms yields

$$\delta^k(t) = \epsilon^k(t) + \int_{t_n}^{t} F(\tau, \varphi^k(\tau) + \delta^k(\tau)) - F(\tau, \varphi^k(\tau))\, d\tau \tag{2.3.5}$$

which is a Picard-like integral equation relating the error to the residual. Finally, we update $\varphi^k(t)$ by

$$\varphi^{k+1}(t) = \varphi^k(t) + \delta^k(t). \tag{2.3.6}$$

Hence, PIDC reduces to three main steps:

1. Compute the residual associated with an approximate solution.
2. Compute the error associated with this residual using the integral equation relationship between residual and error.
3. Correct the solution using the error computed above.

The methods we consider differ in the manner in which this general computational strategy is implemented.

2.4 Spectral Deferred Correction

Without loss of generality, suppose we introduce substeps on the timestep $[t_n, t_{n+1}]$, notated by $\hat{t}_j$ for $j = 0, \ldots, m$, such that $t_n = \hat{t}_0 < \hat{t}_1 < \cdots < \hat{t}_m = t_{n+1}$. These substeps are generally introduced in order to stay within the stability regime of the method, or to adequately resolve the solution. We do not require the substeps to be equispaced. Thus, define $\Delta\hat{t}_j := \hat{t}_{j+1} - \hat{t}_j$. Given a kth corrected solution φ_j^k for $j = 0, \ldots, m$, i.e. $\varphi_j^k \approx y(\hat{t}_j)$, we need to evaluate the integral that shows up in eq. (2.3.4) in order to find the residual ϵ_j^k for $j = 0, \ldots, m$. In the current literature, this process consists of precomputing the coefficients $s_{j,i}$ such that

$$\int_{t_n}^{\hat{t}_j} F(\tau, \varphi^k(\tau))\, d\tau \approx \sum_{i=0}^{m} s_{j,i} F(\hat{t}_i, \varphi_i^k) \tag{2.4.1}$$

for all $j = 0, \ldots, m$. Note that $s_{0,i} = 0$ for all i. In SpectralDC, these coefficients are obtained by integrating the Lagrange basis polynomials constructed on the temporal nodes. Specifically, considering $\{(\hat{t}_j, \varphi_j^k)\}_{j=0}^m$, the polynomial interpolant through the data is given by

$$F(\tau, \varphi^k(\tau)) \approx \sum_{i=0}^{m} F(\hat{t}_i, \varphi_i^k) L_i(\tau), \quad \text{where} \quad L_i(\tau) = \prod_{k \neq i} \frac{\tau - \hat{t}_k}{\hat{t}_i - \hat{t}_k}.$$

Thus,

$$\int_{t_n}^{\hat{t}_j} F(\tau, \varphi^k(\tau))\, d\tau \approx \sum_{i=0}^{m} \left(\int_{t_n}^{\hat{t}_j} L_i(\tau)\, d\tau \right) F(\hat{t}_i, \varphi_i^k) \quad \text{where} \quad s_{j,i} = \int_{t_n}^{\hat{t}_j} L_i(\tau)\, d\tau$$

Note that the precomputation of these coefficients is possible if the substeps for each timestep are chosen to be at the same relative positions across all timesteps. If the timesteps (*not* substeps) aren't uniform, then one only need to scale the coefficients by a suitable constant to compute the integrals correctly for each timestep.

Something worth noting at this point is the "spectral" part of the name. It is somewhat misleading because the integrals in eq. (2.4.1) are not computed using spectral quadrature methods. The "spectral" name actually comes from the idea that we can choose the nodes of spectral quadrature methods (e.g. Gauss-Lobatto, Chebyshev, etc.) as the substeps. However, this only makes it so that the integral over the *entire* timestep is the result of a spectral quadrature rule, but any of the smaller integrals are not. The reason for wanting to pick substeps according to spectral quadrature methods is to improve with the accuracy of the implicit interpolation that occurs during the precomputation step. For example, using Chebyshev nodes as the interpolation locations minimizes interpolation error, thus minimizing the effect of Runge's phenomenon. In [18], Layton et al. studies the effect of using the nodes of various quadrature methods on the efficacy of SpectralDC methods. However, it is not required to substep according to the nodes of any spectral quadrature method, and the coefficients of quadrature are always calculated from integrating the Lagrange basis

polynomials at the nodes.

Returning to the residual equation eq. (2.3.3), we can now compute

$$\epsilon_j^k = \varphi_0 + \sum_{i=0}^{m} s_{j,i} F(\hat{t}_i, \varphi_i^k) - \varphi_j^k \tag{2.4.2}$$

by a matrix-vector multiplication. Then to numerically solve eq. (2.3.5), it is common practice to use a left-hand or right-hand sum to approximate the integral when advancing δ^k, yielding either

$$\delta_{j+1}^k = \delta_j^k + \epsilon_{j+1}^k - \epsilon_j^k + \Delta\hat{t}_j \left[F(\hat{t}_j, \varphi_j^k + \delta_j^k) - F(\hat{t}_j, \varphi_j^k)\right] \tag{2.4.3}$$

or

$$\delta_{j+1}^k = \delta_j^k + \epsilon_{j+1}^k - \epsilon_j^k + \Delta\hat{t}_j \left[F(\hat{t}_{j+1}, \varphi_{j+1}^k + \delta_{j+1}^k) - F(\hat{t}_{j+1}, \varphi_{j+1}^k)\right] \tag{2.4.4}$$

which are reminiscent of forward or backward Euler-type integration, respectively. The choice between left or right is made based on the stiffness of the underlying problem. Note that it is not required to only use a left-hand or right-hand sum to approximate the integral. Higher-order quadrature approximation may also be used to solve for the error. However, for the purpose of our work, we will only be using first-order methods, i.e. forward or backward Euler, to solve the error- residual relationship because our goal is to develop high-order methods using low-order ones.

We correct our solution by using the update

$$\varphi_j^{k+1} = \varphi_j^k + \delta_j^k \tag{2.4.5}$$

for all $j = 0, \ldots, m$. Finally, this entire process is repeated on each of the time steps of the entire time interval of interest. Using either of the first-order methods in eqs. (2.4.3) and (2.4.4) to compute the error for correction k sweeps per time step, the global accuracy

of the solution is formally

$$\mathcal{O}(\Delta T^{\min(k+p_0,m+1)}),$$

where ΔT is the time step size, p_0 is the order of accuracy of the initial approximation, and m is the number of substeps within a timestep (or equivalently, $m+1$ is the number of substep nodes) [10, 19]. The reason why the minimum function occurs here is because the order of accuracy of SpectralDC is limited by the following:

1. the order of accuracy of the provisional solution,

2. the quadrature of the interpolation, and

3. the number of correction sweeps.

The first argument of the minimum function comes from (1) and (3), while the second argument comes from (2). There is a subtle point here that is worth mentioning regarding the order of quadrature. Recall, a polynomial that interpolates $m+1$ data points is $\mathcal{O}(h^{m+1})$ accurate, where h is the length of the interval over which the interpolation takes place. One can then prove that a quadrature method using such an interpolant will be $\mathcal{O}(h^{m+2})$ accurate. However, in the context of SpectralDC, this level of accuracy is only local. In other words, the local truncation error associated with the quadrature method on each time step is $\mathcal{O}(h^{m+2})$ where h is the time step size. Thus, the global error will be $\mathcal{O}(h^{m+1})$.

Note that since we are using a correction method, the initial approximation doesn't have to be especially accurate. In other words, we may let $\varphi_j^0 = y_n$ for all $j = 0, \ldots, m$, which is an $\mathcal{O}(1)$ approximation to the solution. In this particular case, the formal order of accuracy will be $\mathcal{O}(\Delta \hat{t}^{\min(k,m+1)})$. However, we must also keep in mind that, using an $\mathcal{O}(1)$ accurate initial approximations means we need to perform one more correction sweep in order to obtain the maximum order of accuracy of the SpectralDC method, which is more expensive than obtaining an initial approximation by using, for example, a first-order method.

We can also write a direct update for φ^{k+1}. Inserting $t = \hat{t}_j$ and $t = \hat{t}_{j+1}$ into eq. (2.3.3) and subtracting yields

$$\epsilon^k_{j+1} - \epsilon^k_j = \int_{\hat{t}_j}^{\hat{t}_{j+1}} F(\tau, \varphi^k(\tau))\, d\tau - (\varphi^k_{j+1} - \varphi^k_j). \tag{2.4.6}$$

The integral in the equation above can computed by using the precomputed coefficient as follows:

$$\int_{\hat{t}_j}^{\hat{t}_{j+1}} F(\tau, \varphi^k(\tau))\, d\tau \approx \sum_{i=0}^{m} (s_{j+1,i} - s_{j,i}) F(t_i, \varphi^k_i).$$

Then, inserting eq. (2.4.6) into eq. (2.4.3) yields the explicit full update

$$\varphi^{k+1}_{j+1} = \varphi^{k+1}_j + \Delta\hat{t}_j \left[F(\hat{t}_j, \varphi^{k+1}_j) - F(\hat{t}_j, \varphi^k_j)\right] + \sum_{i=0}^{m} (s_{j+1,i} - s_{j,i}) F(\hat{t}_i, \varphi^k_i). \tag{2.4.7}$$

An implicit full update can just as easily be derived from eq. (2.4.4),

$$\varphi^{k+1}_{j+1} = \varphi^{k+1}_j + \Delta\hat{t}_j \left[F(\hat{t}_{j+1}, \varphi^{k+1}_{j+1}) - F(\hat{t}_{j+1}, \varphi^k_{j+1})\right] + \sum_{i=0}^{m} (s_{j+1,i} - s_{j,i}) F(\hat{t}_i, \varphi^k_i). \tag{2.4.8}$$

In practice, however, these full updates are not used to avoid loss in precision [19].

2.4.1 The Pseudocode - SpectralDC

Algorithm 1: Spectral Deferred Correction

function SpectralDC $(F, t_I, t_F, y_0, S, N, m)$;

Input : F - right-hand side, t_I - initial time, t_F - final time, y_0 - initial condition,
S - number of correction sweeps,
N - number of (coarse) time steps,
m - number of (fine) substeps per time step

Output: An array of solution values at each of the coarse temporal nodes $\{t_n\}_{n=0}^{N}$

$\Delta T = (t_F - t_I)/N$;

Compute the coefficients $s_{j,i}$ with respect to the $m+1$ spectral quadrature nodes on the reference interval $[-1, 1]$;

for $n = 0, \ldots, N-1$ **do**

  Compute the provisional solution, $\{\varphi_j^0\}_{j=1}^{m}$, on the nth timestep using a low-order (e.g. first order) method;

  for $s = 0, \ldots, S-1$ **do**

    Compute the residual $\{\epsilon_j^s\}_{j=0}^{m}$ using eq. (2.4.2); `// Scale` $s_{j,i}$ `by` $\Delta T/2$

    Compute the associated error $\{\delta_j^s\}_{j=0}^{m}$ using either eq. (2.4.3) or eq. (2.4.4);

    Compute the corrected solution $\{\varphi_j^{s+1}\}_{j=0}^{m}$ using eq. (2.4.5);

  end

  Push φ_m^S into the output array of solution values;

  Set $\varphi_0^0 = \varphi_m^S$ to initialize the computation of the provisional solution for the next time step;

end

Numerical solution has global order of accuracy $\mathcal{O}(\Delta T^{\min(S+1, m+1)})$;

2.5 Spline Deferred Correction

Fixing notation, let $S(x)$ be a spline on the interval $[a, b]$ with respect to the *knots* $\{z_i\}_{i=0}^N$ so that $a = z_0 < z_1 < \cdots < z_N = b$. We will also refer to the knots as *break points*. We say $S(x)$ is a degree-D spline if each of the polynomial pieces of S are degree-D polynomials, i.e. for $j = 0, \ldots, N-1$, $S(x) = S_j(x)$ on the interval $[z_j, z_{j+1}]$, where

$$S_j(x) = \sum_{k=0}^{D} a_{j,k}(x - z_j)^k.$$

Recall that the construction of a spline requires some level continuity imposed at the break points and the interpolatory conditions (which need not be at the break points). We will discuss the construction of splines in great detail in Chapter 6.

It is also worth noting that there is the B-spline approach to splines, which consists of representing a spline as a linear combination of basis piecewise polynomials with compact support. One may wish to refer to [9] for a reference on B-splines. Throughout the rest of our discussion, we will not be using B-spline representation in spline deferred correction.

The primary difference between SpectralDC and SplineDC is implementation of the quadrature step in the computation of the residual. In SpectralDC, the quadrature approximation is obtained by integrating a polynomial interpolant approximation of F evaluated on the current approximation over the intervals $[t_n, \hat{t}_j]$. In SplineDC, we modify this quadrature step by, instead, constructing a spline approximation of F and analytically integrating the spline to compute the integrals. To be more specific, let $\{\varphi_j^k\}_{j=0}^{m+1}$ be the kth correction solution at the substeps; this allows us to evaluate $F(\hat{t}_j, \varphi_j^k)$ for all j. Then, we construct a spline $S_F(t)$ through the data $\{(\hat{t}_j, F(\hat{t}_j, \varphi_j^k)\}_{j=0}^{m+1}$, where the subscript F on S denotes that $S_F(t)$ is a spline approximation to $F(t, y(t))$ (evaluated on the solution). To keep things simple, we take the substeps $\{\hat{t}_j\}_{j=0}^{m+1}$ as the break point, i.e. each piece of the spline will

have the form

$$S_{Fj}(t) = \sum_{k=0}^{D} a_{j,k}(t - \hat{t}_j)^k$$

We then carry on with the integrations to compute the residual by analytically integrating the spline over the intervals $[t_n, \hat{t}_j]$ for $j = 0, \ldots, m+1$, i.e.

$$\begin{aligned} \epsilon_j^k &= \varphi_0 + \int_{t_n}^{\hat{t}_j} S_F(\tau)\, d\tau - \varphi_j^k \\ &= \varphi_0 + \sum_{i=0}^{j-1} \sum_{k=0}^{D} a_{i,k} \frac{(\Delta \hat{t}_i)^{k+1}}{k+1} - \varphi_j^k \end{aligned} \tag{2.5.1}$$

To implement SplineDC by using full updates (avoiding the need to explicitly compute the residual and the error), we can rewrite those updates as

$$\varphi_{j+1}^{k+1} = \varphi_j^{k+1} + \Delta \hat{t}_j \left[F(\hat{t}_j, \varphi_j^{k+1}) - F(\hat{t}_j, \varphi_j^k) \right] + \sum_{k=0}^{D} a_{j,k} \frac{(\Delta \hat{t}_j)^{k+1}}{k+1} \tag{2.5.2}$$

for the explicit version, and

$$\varphi_{j+1}^{k+1} = \varphi_j^{k+1} + \Delta \hat{t}_j \left[F(\hat{t}_{j+1}, \varphi_{j+1}^{k+1}) - F(\hat{t}_{j+1}, \varphi_{j+1}^k) \right] + \sum_{k=0}^{D} a_{j,k} \frac{(\Delta \hat{t}_j)^{k+1}}{k+1} \tag{2.5.3}$$

for the implicit version, where the final term in both expressions is simply the analytical evaluation of $\int_{\hat{t}_j}^{\hat{t}_{j+1}} S_{Fj}(\tau)\, d\tau$. Again, we are restricting ourselves to only using first-order methods to perform the correction sweeps.

The choice of which spline to use completely depends upon the user, and should be made according to the required overall order of accuracy of the method. In the following chapter, we will present a convergence proof for SplineDC methods that use first-order correction sweeps. However, for now, we will use their results without proof to complete our discussion. Assuming that one uses a spline that is qth-order accurate, i.e. $\mathcal{O}(\Delta \hat{t}^q)$, then the order of

accuracy of a SplineDC method using S first-order correction sweeps will be

$$\mathcal{O}\left(\Delta\hat{t}^{\min(S+p_0,q)}\right). \tag{2.5.4}$$

where p_0 is the order of accuracy of the method used to compute the provisional solution. Note that this result is analogous to that of the SpectralDC. However, since the order of accuracy of splines is an independent parameter, splines of various orders of accuracy may be used in each correction sweep without a reduction of order of the method itself. More specifically, suppose one would like to construct a pth-order SplineDC method. This then implies that in the kth correction sweep, only a spline of order of accuracy $\min(k+p_0, p)$ needs to be used. This results in a computationally cheaper variant of SplineDC that we will refer to as MultiSplineDC.

2.5.1 The Pseudocode - SplineDC

Athough the pseudocode for SplineDC is nearly identical to that of SpectralDC, we include it here in full for completeness.

Algorithm 2: Spline Deferred Correction

function SpectralDC $(F, t_I, t_F, y_0, S, N, m, \text{splineOp})$;

Input : F - right-hand side, t_I - initial time, t_F - final time, y_0 - initial condition,

S - number of correction sweeps,

N - number of (coarse) time steps,

m - number of (fine) substeps per time step

splineOp - operator for constructing splines

Output: An array of solution values at each of the coarse temporal nodes $\{t_n\}_{n=0}^{N}$

$\Delta T = (t_F - t_I)/N$;

for $n = 0, \ldots, N-1$ **do**

  Compute the provisional solution, $\{\varphi_j^0\}_{j=1}^m$, on the nth time step;

  for $s = 0, \ldots, S-1$ **do**

    Use splineOp to construct a qth-order accurate spline through $\{(\hat{t}_j, \varphi_j^0\}_{j=0}^m$;

    Compute the residual $\{\epsilon_j^s\}_{j=0}^m$ using eq. (2.5.1);

    Compute the associated error $\{\delta_j^s\}_{j=0}^m$ using either eq. (2.4.3) or eq. (2.4.4);

    Compute the corrected solution $\{\varphi_j^{s+1}\}_{j=0}^m$ using eq. (2.4.5);

  end

  Push φ_m^S into the output array of solution values, and reset $\varphi_0^0 = \varphi_m^S$;

end

Numerical solution has order of accuracy $\mathcal{O}(\Delta \hat{t}^{\min(S+1,q)})$;

2.5.2 Additional details and advantages of SplineDC

Constructing higher-order SplineDC methods

In order to construct SpectralDC methods of increasing order of accuracy, the number of substeps per time steps must be increased. This is because the number of interpolation locations directly affects the degree of the Lagrange interpolant, which affects the order of accuracy of the polynomial. However, increasing the number of interpolation locations without limit is not recommended because of the instabilites associated with high-degree polynomial interpolation, e.g. Runge's phenomenon. In the case of SpineDC, constructing methods of increasing order amounts to using splines of sufficiently high order of accuracy which does not impose any requirements on the number of substeps. Furthermore, uniform substeps may now also be used.

In various SplineDC methods that we use later in numerical experiments, there is a notion of the minimum number of substeps required because of how we are specifying the extra conditions needed to close the system of equations for the coefficients. However, even in this case, we do not need to increase the number of substeps in order to increase the order of accuracy of the spline.

Time complexity

Let N be the number of time steps, m be the number of substeps per timestep, S be the number of correction sweeps, and K be the number of operations needed for one function evaluation of the right-hand side of the differential equation. Time complexity-wise, explicit SpectralDC is an $\mathcal{O}(SN(m^2 + Km))$ method, where the m^2 term comes from the dense matrix-vector multiplication in the quadrature step, and the Km term comes from the extra work incurred from function evaluations. K is scaled by m because computing the residual and integrating the error differential equation requires a function evaluation per substep. For implicit SpectralDC, there will be an extra term dependent on the dimensionality of the problem in the sum $m^2 + Km$ that accounts for the extra work required in an implicit solve.

With all of this in mind, we first note that the time complexity of explicit SplineDC is at worst $\mathcal{O}(SN(m^2 + Km))$. If one chooses to use B-spline representations, then, as de Boor points out in [9], the compact support of the basis splines yields a banded system to solve for the linear combination of the basis splines. Furthermore, the coefficient matrix is diagonally dominant, which means it can be stably solved using Gaussian elimination without pivoting with $\mathcal{O}(m^2)$ work. Then, de Boor also points out that integrating a spline in its B-spline form can be done with $\mathcal{O}(m^2)$ work. This holds for all splines with break points at the substep nodes, so at worst, the time complexity of SplineDC is the same as SpectralDC.

However, the time complexity of SplineDC can be as low as

$$\mathcal{O}(SN(m + Km)) = \mathcal{O}(SNKm),$$

and it is exactly when the spline can be constructed and integrated with $\mathcal{O}(m)$ work. This is the case for at least linear, quadratic, and cubic splines if they are constructed in the classical way, which is to find the coefficients such that each piece of the spline can be written as

$$S_j(x) = \sum_{k=0}^{D} a_{j,k}(x - z_j)^k.$$

In all of the cases just listed, the equations for the unknowns (splines coefficients) can be manipulated so that it is a tridiagonal solve to find the coefficients of the second-highest-degree term in each polynomial piece, and the remaining coefficients can be computed via backward/forward substitutions. Hence, the entire solve has complexity $\mathcal{O}(m)$. Then, integration of the entire spline can be done analytically, which is also linear in time.

A consequence of SplineDC having time complexity $\mathcal{O}(SNKm)$ is there is more freedom in choosing the number of time steps and substeps. Because the amount of work SpectralDC does per time step per correction sweep is quadratic in m, it's practical to choose time steps and substeps accordingly so m is not too large separate from the instabilities associated with

polynomial interpolation. However, in the case of SplineDC, this is no longer a concern since the time complexity is linear in both number of time steps and substeps. Furthermore, if we let M be the total number of steps a numerical scheme needs to obtain a solution of a certain accuracy, then M must be equal to the product of the number of (coarse) time steps and (fine) substeps. Thus, the Nm term in the complexity of SplineDC is just constant, so the complexity simplifies to

$$\mathcal{O}(SMK) \tag{2.5.5}$$

Given eq. (2.5.5) for the cases where the work in constructing and integrating the spline is linear in the number of substeps, M can be minimized with respect to some threshold tolerance by employing adaptive time-stepping in the computation of the provisional solution. Note that the implementation of adaptive time-stepping unavoidably increases the number of function evaluations. However, this increase should only be linear in the quantity KM.

Using a time complexity argument, it seems favorable to implement adaptive time-stepping in the context of SplineDC. However, separate from the discussion above, adaptive time-stepping is inherently easier to efficiently implement in SpectralDC. In order for adaptive time-stepping to work in the context of SpectralDC methods (seen in [8, 10, 23]), one must adaptively choose the time steps as opposed to the substeps. The reason is because the ability to precompute the quadrature coefficients relies on the assumption that the substep nodes are always in the same relative locations per time step. Of course, one can recompute these quadrature coefficients each time step if the substeps were to change, but this is expensive. However, this implies that in order for the adaptive time-stepping scheme to know to reject a time step, it must have already computed the solution across at least some of the substeps. In SplineDC methods, we can instead carry out a more classical adaptive time-stepping scheme by keeping the time step fixed while adaptively choosing the substep sizes. In this approach, we can immediately reject a substep when needed.

Computationally more optimal correction sweeps

The idea of MultiSplineDC was discussed in the previous section, but we mention it again here for completeness of this section regarding advantages of SplineDC. MultiSplineDC, a variant of SplineDC where splines of sufficient order are used in each correction sweep, is computationally more optimal. In classic SplineDC methods and SpectralDC methods, the order of accuracy of the interpolant that's used only contributes to the limiting factor of the accuracy of the numerical solution after sufficiently many correction sweeps. This implies that initially, the interpolant is "doing too much work." Unfortunately, this is unavoidable for SpectralDC methods because the degree, and hence the order, of the Lagrange interpolant is dependent on the number of substeps. However, it's straightforward to use different splines in each correction sweep; hence, the development of MultiSplineDC.

2.6 Collocation and Picard Integral Deferred Correction

So far, our only motivation for the derivation of Picard integral deferred correction (PIDC) methods is from an error-correction approach. However, there is an alternative approach to the derivation of these methods, i.e. as an iterative solver for a collocation solution. This can be found in [16, 21] but we include it here for completeness.

2.6.1 The Collocation Formulation

Recall that we are concerned with solving the initial value problem

$$\begin{aligned} y'(t) &= F(t, y(t)) \quad t \in [t_0, T] \\ y(t_0) &= y_0. \end{aligned}$$

where $y(t)$, $y_0 \in \mathbb{C}^N$ and $F : \mathbb{R} \times \mathbb{C}^N \to \mathbb{C}^N$. For simplicity, we let $N = 1$, even though the following derivation works for any N. Then for our methods, we rewrite the above into its

Picard integral formulation

$$y(t) = y_0 + \int_{t_0}^{t} F(\tau, y(\tau))\, d\tau, \quad t \in [t_0, T].$$

Like before, we'll only focus our derivation on an arbitrary time step $[t_n, t_{n+1}]$ of the entire time interval of interest, and without loss of generality, we introduce m substeps, $t_n = \hat{t}_0 < \cdots < \hat{t}_m = t_{n+1}$, on the time step. Hence, we rewrite the Picard integral formulation above as

$$y(t) = y_0 + \int_{t_n}^{t} F(\tau, y(\tau))\, d\tau, \quad t \in [t_n, t_{n+1}]. \tag{2.6.1}$$

Let $q_{j,k}$ be coefficients that allow us to approximate the integral of some real-valued function g on the subintervals $[t_n, \hat{t}_j]$ using the following quadrature

$$\int_{t_n}^{\hat{t}_j} g(\tau)\, d\tau \approx \Delta T \sum_{k=0}^{m} q_{j,k} g(t_k),$$

where ΔT is the length of the timetep. Note that $q_{0,k} = 0$ for all k. Let $\boldsymbol{Q}$ be an $(m+1) \times (m+1)$ matrix of the coefficients $q_{j,k}$, and hereafter, $\boldsymbol{Q}$ will be referred to as the integration matrix.

Note that here, we are not assuming we know anything about the coefficients $q_{j,k}$. This generality allows us to continue with the following derivations of this chapter without assuming which class of PIDC methods we are dealing with: spectral or spline deferred correction. In either case, the quadrature is ultimately computed by using an appropriate linear combination of the function data at the substeps.

Using the integration matrix $\boldsymbol{Q}$, we can discretize eq. (2.6.1) over the time step $[t_n, t_{n+1}]$ with respect to the substeps $\{\hat{t}_j\}_{j=0}^{m}$. To do so, we first introduce some notation. Let $\vec{\varphi} = (\varphi_0, \ldots, \varphi_m)^T$ be the vector of solution values we are solving for, so $[\vec{\varphi}]_j = \varphi_j \approx y(\hat{t}_j)$, and $\vec{F}(\vec{\varphi}) = (F(t_0, \varphi_0), \ldots, F(t_m, \varphi_m))$. Finally, after relabeling $y(t_n)$ as y_0 (since t_n is technically our initial time), let $\vec{y}_0$ be the $(m+1) \times 1$ column vector with each entry equal

to y_0. Using $\boldsymbol{Q}$, we may now rewrite eq. (2.6.1) as

$$\vec{\varphi} = \vec{y}_0 + \Delta T \boldsymbol{Q} \vec{F}(\vec{\varphi}) \tag{2.6.2}$$

which is the collocation form of the initial value problem. Solving this, which is what PIDC aims to do, results in a collocation scheme for the ODE.

For completeness, we briefly discuss the derivation of the collocation formulation for the multi-dimensional case, i.e. $N > 1$. The solution vector is now a column vector of length $N(m+1)$ of the form

$$\vec{\Phi} = \left(\varphi_0^1, \ldots, \varphi_0^N, \ldots, \varphi_m^1, \ldots, \varphi_m^N\right)^T$$

The vector of right-hand side values evaluated at the solution is now

$$\vec{F}(\vec{\Phi}) = \left(F^1(t_0, \vec{\Phi}_0), \ldots, F^N(t_0, \vec{\Phi}_0), \ldots, F^1(t_m, \vec{\Phi}_m), \ldots, F^N(t_m, \vec{\Phi}_m)\right)^T$$

Similarly, the initial vector is now $\vec{Y}_0$, and it contains $N(m+1)$ copies of y_0. Finally, let $\tilde{\boldsymbol{Q}} = \boldsymbol{Q} \otimes \boldsymbol{I}_N$ where $\boldsymbol{I}_N$ is the $N \times N$ identity matrix. Then, the analogous form of eq. (2.6.2) is

$$\vec{\Phi} = \vec{Y}_0 + \Delta T \tilde{\boldsymbol{Q}} \vec{F}(\vec{\Phi})$$

2.6.2 PIDC in Compact Notation

Given the collocation formulation eq. (2.6.2), we can now rewrite the steps of PIDC using compact notation.

Recall the definition $\Delta \hat{t}_j = \hat{t}_{j+1} - \hat{t}_j$ and let $\xi_j = \Delta \hat{t}_j / \Delta T$. Now, define $\boldsymbol{Q}_E$ to be the

following $(m+1) \times (m+1)$ matrix

$$\boldsymbol{Q}_E = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ \xi_0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \xi_0 & \xi_1 & \cdots & 0 & 0 \\ \xi_0 & \xi_1 & \cdots & \xi_{m-1} & 0 \end{bmatrix}.$$

This matrix is referred to as a low-order integration matrix by Minion et al. in [21]. The subscript E refers to the explicit nature of the integration because of its strictly lower-triangular structure. We also define an analogous implicit version:

$$\boldsymbol{Q}_I = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & \xi_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \xi_0 & \cdots & \xi_{m-2} & 0 \\ 0 & \xi_0 & \cdots & \xi_{m-2} & \xi_{m-1} \end{bmatrix}$$

Using the same matrix-vector/collocation notation as in the previous section, taking m steps of forward Euler to advance the solution of an IVP can be written as

$$\vec{\varphi} = \vec{\varphi}_0 + \Delta T \boldsymbol{Q}_E \vec{F}(\vec{\varphi})$$

Note that taking m steps of backward Euler instead amounts to replacing $\boldsymbol{Q}_E$ with $\boldsymbol{Q}_I$.

Recall the two automatic update equations for PIDC:

$$\varphi_{j+1}^{k+1} = \varphi_j^{k+1} + \Delta \hat{t}_j \left[F(\hat{t}_j, \varphi_j^{k+1}) - F(\hat{t}_j, \varphi_j^k) \right] + \Delta T \sum_{i=0}^{m} (q_{j+1,i} - q_{j,i}) F(\hat{t}_i, \varphi_i^k). \qquad (2.6.3)$$

and

$$\varphi_{j+1}^{k+1} = \varphi_j^{k+1} + \Delta\hat{t}_j \left[F(\hat{t}_{j+1}, \varphi_{j+1}^{k+1}) - F(\hat{t}_{j+1}, \varphi_{j+1}^{k})\right] + \Delta T \sum_{i=0}^{m} (q_{j+1,i} - q_{j,i}) F(\hat{t}_i, \varphi_i^k). \quad (2.6.4)$$

Notice that eqs. (2.6.3) and (2.6.4) slightly differ from when we first introduced them in Chapter 1. In Chapter 1, we assumed that the integrals were computed by using a Lagrange interpolation of the integrand. Here, we only assume that the quadrature values are simply linear combinations of the data, where the coefficients need not be computed by integrating the Lagrange basis polynomials. Using the explicit low-order integration matrix, all of the iteratons implied in eq. (2.6.3) to compute the $(k+1)$th corrected solution can be written in the following form:

$$\begin{aligned} \vec{\varphi}^{k+1} &= \vec{\varphi}_0 + \Delta T \boldsymbol{Q}_E \left(\vec{F}(\vec{\varphi}^{k+1}) - \vec{F}(\vec{\varphi}^{k})\right) + \Delta T \boldsymbol{Q} \vec{F}(\vec{\varphi}^{k}) \\ &= \vec{\varphi}_0 + \Delta T \boldsymbol{Q}_E \vec{F}(\vec{\varphi}^{k+1}) + \Delta T (\boldsymbol{Q} - \boldsymbol{Q}_E) \vec{F}(\vec{\varphi}^{k}). \end{aligned} \quad (2.6.5)$$

We may also do the same for eq. (2.6.4):

$$\vec{\varphi}^{k+1} = \vec{\varphi}_0 + \Delta T \boldsymbol{Q}_I \vec{F}(\vec{\varphi}^{k+1}) + \Delta T (\boldsymbol{Q} - \boldsymbol{Q}_I) \vec{F}(\vec{\varphi}^{k}), \quad (2.6.6)$$

where in both cases, $\vec{\varphi}_0 = \vec{y}_0$. These two expressions are PIDC in its most compact form. However, it should be noted that this is only the case because we assume the correction equations are advanced using forward or backward Euler. If higher-order methods are used to solve the correction equations, then one must replace $\boldsymbol{Q}_E$ and $\boldsymbol{Q}_I$ above appropriately.

2.6.3 Special Property for Linear IVPs

If we suppose that $\vec{F}$ is linear, i.e. $\vec{F}(\vec{y}) = \boldsymbol{A}\vec{y}$ for some matrix $\boldsymbol{A}$, then the collocation formulation eq. (2.6.2) can be written as

$$(\boldsymbol{I} - \Delta T \boldsymbol{Q}\boldsymbol{A})\vec{\varphi} = \vec{\varphi}_0 \tag{2.6.7}$$

Furthermore, eq. (2.6.5), which is an explicit PIDC scheme used to solve eq. (2.6.7), is now

$$(\boldsymbol{I} - \Delta T \boldsymbol{Q}_E\boldsymbol{A})\vec{\varphi}^{k+1} = \vec{\varphi}_0 + \Delta T(\boldsymbol{Q} - \boldsymbol{Q}_E)\boldsymbol{A}\vec{\varphi}^k \tag{2.6.8}$$

At this point, we embark on a slight tangent to make an interesting observation, that in the linear case, PIDC can be viewed as a relaxation method for solving the collocation formulation of an IVP. To do so, let us quickly review the idea of relaxation methods.

Recall that relaxation methods are a class of methods for solving systems of equations. Though the methods are also used in the case of non-linear systems, our discussion will only be concerned with linear systems. That being said, let us consider solving the linear problem $\boldsymbol{A}\vec{x} = \vec{b}$. The basic idea behind relaxation methods is to find a splitting of $\boldsymbol{A}$ into $\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$ where $\boldsymbol{M}$ is easily invertible. Then, the iterative method is derived as follows:

$$\begin{aligned} \boldsymbol{A}\vec{x} = \vec{b} \quad &\implies \quad \boldsymbol{M}\vec{x} = \boldsymbol{N}\vec{x} + \vec{b} \\ &\implies \quad \boldsymbol{M}\vec{x}^{n+1} = \boldsymbol{N}\vec{x}^n + \vec{b} \end{aligned} \tag{2.6.9}$$

It is well known that an iteration of this form converges if and only if $\rho(\boldsymbol{M}^{-1}\boldsymbol{N}) < 1$, where $\rho(\cdot)$ is the spectral radius.

Something else worth noting about relaxation methods in this brief discussion is the nature of how these methods work. Let $\vec{x}^*$ be the solution of $\boldsymbol{A}\vec{x} = \vec{b}$, and let $\vec{x}^n$ be the nth iterate of some relaxation method. Define $\vec{r}^n = \vec{b} - \boldsymbol{A}\vec{x}^n$ to be the residual and $\vec{e}^n = \vec{x}^* - \vec{x}^n$

be the error associated with the nth iterate. Combining these two, we obtain the relationship between the error and the residual, $\boldsymbol{A}\vec{e}^n = \vec{r}^n$. With this in mind, let's subtract $\boldsymbol{M}x^n$ from both sides of eq. (2.6.9) to obtain

$$\begin{aligned} \boldsymbol{M}\vec{x}^{n+1} = \boldsymbol{N}\vec{x}^n + \vec{b} \quad &\implies \quad \boldsymbol{M}\vec{x}^{n+1} - \boldsymbol{M}\vec{x}^n = \vec{b} - \boldsymbol{A}\vec{x}^n \\ &\implies \quad \vec{x}^{n+1} = \vec{x}^n + \boldsymbol{M}^{-1}\vec{r}^n \end{aligned} \tag{2.6.10}$$

Finally, if we view $\boldsymbol{M}$ as an approximation to $\boldsymbol{A}$, observe that

$$\boldsymbol{A}\vec{e}^n = \vec{r}^n \quad \implies \quad \boldsymbol{M}\vec{e}^n \approx \vec{r}^n \quad \implies \quad \vec{e}^n \approx \boldsymbol{M}\vec{r}^n \tag{2.6.11}$$

The idea that $\boldsymbol{M}$ could be an approximation of $\boldsymbol{A}$ isn't too weird, even if we are using the word "approximation" very loosely here. Note that if $\boldsymbol{M} = \boldsymbol{A}$, then the iterative method is actually a one-step method for solving the linear system, implying that the better $\boldsymbol{M}$ approximates $\boldsymbol{A}$, the more closely the iterative method boils down to just directly solving the system. Combining eqs. (2.6.10) and (2.6.11) illuminates the underlying nature of relaxation methods, which is that these methods use the residual to compute an approximation to the error, and uses this approximation to correct the current iterate. This idea is exactly what PIDC uses in its correction step.

Returning back to PIDC, we can now show that eq. (2.6.8) is a relaxation method for solving eq. (2.6.7). To do so, we look for a decomposition of $\boldsymbol{I} - \Delta T\boldsymbol{Q}\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$ such that $\boldsymbol{M}$ is less expensive to invert than $\boldsymbol{I} - \Delta T\boldsymbol{Q}\boldsymbol{A}$. The cooresponding relaxation scheme would then be written as

$$\vec{\varphi}^{k+1} = \boldsymbol{M}^{-1}(\boldsymbol{N}\vec{\varphi}^k + \vec{\varphi}_0)$$

Choosing

$$\boldsymbol{M} = \boldsymbol{I} - \Delta T\boldsymbol{Q}_E\boldsymbol{A} \quad \text{and} \quad \boldsymbol{N} = \Delta T(\boldsymbol{Q} - \boldsymbol{Q}_E)\boldsymbol{A} \tag{2.6.12}$$

produces the PIDC automatic update in eq. (2.6.8). Note that this choice of $\boldsymbol{M}$ is indeed relatively easy to invert — its inversion is exactly what PIDC is doing! Obviously, the same type of splitting exists if we were to replace $\boldsymbol{Q}_E$ with $\boldsymbol{Q}_I$.

Thus, we have observed that in the case of a linear initial value problem, the use of the error- residual relationship in the correction step of PIDC methods is very much related to the same relationship we see in relaxation methods. This observation hints that the analysis used to understand relaxation methods for linear systems might be able to be used to understand PIDC in a different light.

Chapter 3

Convergence of Spline Deferred Correction

In this chapter, we present a proof of convergence of spline deferred correction methods whose correction sweeps are advanced using either forward or backward Euler. In essence, it will follow the proof of convergence for spectal deferred correction methods by Causley et al. [7]. However, there are extra details to account for because of the use of splines.

3.1 Mathematical Preliminaries

3.1.1 Notation

Borrowing notation from [9], let $\mathbb{S}_{D,\vec{\xi}}$ where $\vec{\xi} \in \mathbb{R}^{m+1}$ denote the set of all piecewise polynomial functions (i.e. splines) of degree at most D with respect to the break points $\xi_0 < \xi_1 < \cdots < \xi_m$. In other words, if $S \in \mathbb{S}_{D,\vec{\xi}}$, then S restricted to any of the intervals $[\xi_j, \xi_{j+1}]$ for $j = 0, \ldots, m-1$ is a polynomial of degree at most D. Now, fixing $\vec{\nu} \in \mathbb{N}^{m-1}$, let

$$\mathbb{S}_{D,\vec{\xi},\vec{\nu}} = \left\{ S \in \mathbb{S}_{D,\vec{\xi}} : S \text{ is } C^{\nu_j}\text{-continuous at the interior breaks } \xi_j \text{ for } j = 1, \ldots, m-1 \right\}$$

Note that we choose to index the components of $\vec{\nu}$ starting from 1 to imply that ν_j applies to ξ_j. For example, the classic clamped cubic spline is an element of $\mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ where $D = 3$, $\vec{\xi}$ is any arbitrary vector of break points in $\mathbb{R}^{m+1}$, and $\vec{\nu} = (2, 2, \ldots, 2)^T \in \mathbb{N}^{m-1}$. Finally, if we

always let $m+1$ denote the number of break points (including the left and right endpoints), then

$$\dim(\mathbb{S}_{D,\vec{\xi}}) = m(D+1)$$

and

$$\dim(\mathbb{S}_{D,\vec{\xi},\vec{\nu}}) = m(D+1) - \sum_{j=1}^{m-1}(\nu_j + 1)$$

For the purpose of the convergence proof and in most of our numerical experiments, we will always construct splines such that the break points and the data points coincide. Since the data points are located at the substep nodes of each time step, i.e. $\{\hat{t}_j\}_{j=0}^{m}$ from previous notation, $\hat{t}_j$ and ξ_j will be interchangeable. Furthermore, for all splines other than linear splines that we have constructed and tested with SplineDC, extra conditions must be specified to uniquely determine the spline. There are two approaches to constructing these conditions.

1. The classic approach is to specify osculatory (derivative interpolation) conditions at the end points. The derivatives can either be computed by analytically differentiating the function whose data to which we are fitting a spline, or by using a finite difference approximation involving the data points. If the latter option is chosen, the finite difference approximations need to be of sufficient order in order for the spline to maintain its order of accuracy.

2. A second approach that we have experimented with success is specifying enough extra conditions on the spline itself so the interpolation conditions from the data points are sufficient. Note that this would decrease the dimension of $\mathbb{S}_{D,\vec{\xi},\vec{\nu}}$, so the expression derived above would no longer hold. This approach will be explained in great detail in Chapter 6.

In all of our experiments, if we opt for supplying osculatory conditions, we always use finite difference approximations because of its flexibility; it doesn't require taking derivatives of

potentially complicated functions. Also, the finite difference approximations will simply be linear combinations of the data at the break points. This means that, no matter what approach we choose use to specify extra conditions to close the system of equations that determine the spline coefficients, the data at the $m+1$ substep nodes (or break points) are sufficient.

Thus, we let $\mathcal{S} : \mathbb{R}^{m+1} \to \mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ be the linear operator that takes data at the $m+1$ substep nodes to the unique interpolatory spline of $\mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ through the data. Observe that $\mathcal{S}$ can be understood as the result of two separate operations:

1. A linear system solve that yields a vector of spline coefficients from a vector of substep data. Equivalently, this is a matrix-vector multiplication of an inverse matrix with a vector of substep data to yield the spline coefficients.

2. A map that sends the vector of spline coefficients to the spline function itself.

Putting everything together, given data, $\vec{v}$, at the break points, $\mathcal{S}[\vec{v}](t) \in \mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ is the unique spline of at most degree D with break points $\vec{\xi}$ and is C^{ν_j}-continuous at ξ_j such that

$$\mathcal{S}[\vec{v}](\hat{t}_j) = v_j$$

for all $j = 0, \ldots, m-1$. The error bounds that we present below assume that the operator $\mathcal{S}$ is bounded. We will introduce a spline construction framework in Chapter 6 that will bring to light how boundedness of this operator can be measured and achieved.

3.1.2 Quadrature error bound

For the following proofs, let $\Delta T = t_{n+1} - t_n$, i.e. the time step size. Furthermore, we will use the same notation as before for the substeps: $t_n = \hat{t}_0 < \hat{t}_1 < \cdots < \hat{t}_m = t_{n+1}$ where the substeps need not by uniform. Finally, let $\Delta\hat{t}_j = \hat{t}_{j+1} - \hat{t}_j$ for $j = 0, \ldots, m-1$, and $h = \max_j \Delta\hat{t}_j$.

Lemma 1. *Fix $\mathbb{S}_{D,\vec{\xi},\vec{\nu}}$, and let $\mathcal{S}$ and λ be the linear operators as described above. Suppose*

1. *$F(t, y(t)) \in C^{D+1}([t_n, t_{n+1}])$,*

2. *F is Lipschitz continuous in its second argument with Lipschitz constant L,*

3. *$\mathcal{S} : \mathbb{R}^{m+1} \to \mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ is bounded, i.e.,*

$$\|\mathcal{S}\| := \sup\left\{\frac{\|\mathcal{S}(\vec{v})\|}{\|\vec{v}\|} \ : \ \vec{v} \in \mathbb{R}^{m+1}, \vec{v} \neq 0\right\} \leq B, \text{ and}$$

4. *for any $g \in C^{D+1}([t_n, t_{n+1}])$,*

$$\|\mathcal{S}[g](t) - g(t)\|_\infty \leq Mh^q$$

where M is a constant, i.e. $\mathcal{S}$ produces qth-order accurate splines.

Then, one can obtain the following error bound

$$\left|\int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi})\right](t) - F(y(t))\,dt\right| \leq BL\Delta T||\vec{\varphi} - \vec{y}||_\infty + M(\Delta T)^{q+1} \tag{3.1.1}$$

where $\vec{\varphi}$ and $\vec{y}$ are vectors of values at the substep nodes of a numerical solution and the exact solution, respectively, and $\vec{F}(\vec{\varphi})$ is the vector of values of the right-hand side of the IVP evaluated on the numerical solution.

Proof. We begin by adding and subtracting $\mathcal{S}[\vec{F}(\vec{y})](t)$, the spline interpolant through the data $\{\hat{t}_j, F(\hat{t}_j, y(\hat{t}_j))\}_{j=0}^m$, inside the integral of eq. (3.1.1) and applying the triangle inequality to obtain

$$\begin{aligned}\left|\int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi})\right](t) - F(y(t))\,dt\right| \leq &\left|\int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi})\right](t) - \mathcal{S}\left[\vec{F}(\vec{y})\right](t)\,dt\right| \\ &+ \left|\int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{y})\right](t) - F(y(t))\,dt\right|\end{aligned} \tag{3.1.2}$$

The first term of the right-hand side of eq. (3.1.2) can be bounded by using linearity of the operator S_λ:

$$\begin{aligned}\left|\int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi})\right](t) - \mathcal{S}\left[\vec{F}(\vec{y})\right](t)\,dt\right| &\leq \int_{\hat{t}_j}^{\hat{t}_{j+1}} \left|\mathcal{S}\left[\vec{F}(\vec{\varphi}) - \vec{F}(\vec{y})\right](t)\right| dt \\ &\leq \int_{\hat{t}_j}^{\hat{t}_{j+1}} \|\mathcal{S}\| \left\|\vec{F}(\vec{\varphi}) - \vec{F}(\vec{y})\right\|_\infty dt \\ &\leq BL\Delta T \left\|\vec{\varphi} - \vec{y}\right\|_\infty \end{aligned} \tag{3.1.3}$$

where the final inequality holds because $\Delta\hat{t}_j \leq \Delta T$ for all j, $\|\mathcal{S}\| \leq B$, and F is Lipschitz. The second term on the right-hand side of eq. (3.1.2) is bounded as follows:

$$\begin{aligned}\left|\int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{y})\right](t) - F(y(t))\,dt\right| &\leq \left|\int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{y})\right](t) - F(y(t))\,dt\right| \\ &\leq \int_{\hat{t}_j}^{\hat{t}_{j+1}} \left\|\mathcal{S}\left[\vec{F}(\vec{y})\right](t) - F(y(t))\right\|_\infty dt \\ &\leq \int_{\hat{t}_j}^{\hat{t}_{j+1}} Mh^q\,dt \\ &\leq M(\Delta T)^{q+1} \end{aligned} \tag{3.1.4}$$

where the second-to-last inequality is due to the fourth assumption, and the final inequality uses the fact that $h \leq \Delta T$ for all j. Combining eqs. (3.1.2) to (3.1.4) finishes the proof. □

A key aspect of this lemma is the assumption that the spline construction operator $\mathcal{S}$ is a linear bounded operator, implying that the types of splines to be used and how they are to be constructed need to be carefully considered. We will explore this idea more in Chapter 6. We also note that, regarding the fourth assumption of this lemma, Theorem 6 in Chapter 12 of [9] claims the order of accuracy, q, attainable by splines in $\mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ for functions $g \in C^{D+1}([t_n, t_{n+1}])$ is $D+1$. Obviously, this relies on $\mathcal{S}$, which is further evidence for the importance of understanding the spline operator.

3.2 Explicit SplineDC Convergence Theorem

We are now ready to show the convergence/order of accuracy of explicit SplineDC methods. There are three sources of error in SplineDC methods:

1. The error associated with the initial condition of the timestep.
2. The error from the previous correction sweep.
3. The error associated with the quadrature step.

Furthermore, the error term of the previous correction sweep has a factor of the step size attached to it, which is the reason why these methods increase the order of accuracy of its numerical solutions by an order of one each sweep. Of course, this only happens as long as the order of the numerical solution is strictly less than the order of accuracy of the quadrature step. After sufficiently many correction sweeps, the order of accuracy of the numerical solution can no longer keep increasing.

Theorem 1. *Suppose all of the assumptions of Lemma 1 are satisfied. Fixing $\mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ and $\mathcal{S}$, the error of the $(k+1)$th sweep of explicit SplineDC on one time step satisfies*

$$\left\|\vec{e}^{k+1}\right\|_\infty \leq C_1|e_0| + C_2\Delta T \left\|\vec{e}^{k}\right\|_\infty + C_3\Delta T^{q+1} \tag{3.2.1}$$

for all ΔT, where the constants C_1, C_2, and C_3 only depend on F, the spline operator $\mathcal{S}$, the number of substeps m, and the length of the entire time interval, $[t_I, t_F]$, of interest.

Proof. First, recall the automatic update of explicit SplineDC methods for $j = 0, \ldots, m-1$:

$$\varphi_{j+1}^{k+1} = \varphi_j^{k+1} + \Delta\hat{t}_j\left[F(\hat{t}_j, \varphi_j^{k+1}) - F(\hat{t}_j, \varphi_j^{k})\right] + \int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi}^{\,k})\right](t)\,dt \tag{3.2.2}$$

Also, recall, for $j = 0, \ldots, m-1$, the true solution satisfies

$$y(\hat{t}_{j+1}) = y(\hat{t}_j) + \int_{\hat{t}_j}^{\hat{t}_{j+1}} F(t, y(t))\, dt \tag{3.2.3}$$

Subtracting eq. (3.2.3) from eq. (3.2.2) and letting e_j^k be the error at $\hat{t}_j$ of the kth corrections sweep yields

$$e_{j+1}^{k+1} = e_j^{k+1} + \Delta\hat{t}_j \left[F(\hat{t}_j, \varphi_j^{k+1}) - F(\hat{t}_j, \varphi_j^k)\right] + \int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi}^{\,k})\right](t) - F(t, y(t))\, dt$$

Taking an absolute value of both sides and applying the triangle inequality gives us

$$\begin{aligned} \left|e_{j+1}^{k+1}\right| &\leq \left|e_j^{k+1}\right| + \Delta\hat{t}_j \left|F(\hat{t}_j, \varphi_j^{k+1}) - F(\hat{t}_j, \varphi_j^k)\right| + \left|\int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi}^{\,k})\right](t) - F(t, y(t))\, dt\right| \\ &\leq \left|e_j^{k+1}\right| + \Delta\hat{t}_j L \left|\varphi_j^{k+1} - \varphi_j^k\right| + BL\Delta T||\vec{e}^{\,k}||_\infty + M(\Delta T)^{q+1} \end{aligned} \tag{3.2.4}$$

where we use the Lipschitz assumption of F and Corollary 1 to obtain the second inequality above. We may expand and bound the second term above by

$$\begin{aligned} \left|\varphi_j^{k+1} - \varphi_j^k\right| &\leq \left|\varphi_j^{k+1} - y(\hat{t}_j)\right| + \left|y(\hat{t}_j) - \varphi_j^k\right| \\ &\leq \left|e_j^{k+1}\right| + \left|e_j^k\right| \\ &\leq \left|e_j^{k+1}\right| + \left\|\vec{e}^{\,k}\right\|_\infty, \end{aligned} \tag{3.2.5}$$

so combining eq. (3.2.5) with eq. (3.2.4) yields

$$\begin{aligned} \left|e_{j+1}^{k+1}\right| &\leq (1 + \Delta\hat{t}_j L)\left|e_j^{k+1}\right| + \Delta\hat{t}_j L \left\|\vec{e}^{\,k}\right\|_\infty + BL\Delta T||\vec{e}^{\,k}||_\infty + M(\Delta T)^{q+1} \\ &\leq (1 + \Delta\hat{t}_j L)\left|e_j^{k+1}\right| + \tilde{B}L\Delta T \left\|\vec{e}^{\,k}\right\|_\infty + M(\Delta T)^{q+1} \end{aligned}$$

where we let $\tilde{B} := 1 + B$. Note, we also used the fact that $\Delta\hat{t}_j \leq \Delta T$ for all j. Temporarily,

let $A = \tilde{B}L\Delta T \left\|\vec{e}^{\,k}\right\|_\infty + M(\Delta T)^{q+1}$ for the purpose of suppressing notation. We can iterate this inequality down to e_0^{k+1} to obtain

$$\left|e_{j+1}^{k+1}\right| \leq \left|e_0^{k+1}\right| \prod_{n=0}^{j}(1+\Delta\hat{t}_n L) + A\sum_{n=0}^{j}\prod_{l=0}^{n-1}(1+\Delta\hat{t}_{j-l}L) \tag{3.2.6}$$

Since $\Delta\hat{t}_j L$ is positive, we can write $1+\Delta\hat{t}_j L \leq e^{\hat{t}_j L}$ for all $j = 0, \ldots, m-1$. It then follows that we may obtain the following upper bound:

$$\prod_{n=0}^{j}(1+\Delta\hat{t}_n L) \leq \prod_{n=0}^{j} e^{\Delta\hat{t}_n L} = e^{\sum_{n=0}^{j}\Delta\hat{t}_n L} \leq e^{\Delta T L} \tag{3.2.7}$$

Furthermore,

$$\begin{aligned}
\sum_{n=0}^{j}\prod_{l=0}^{n-1}(1+\Delta\hat{t}_{j-l}L) &\leq 1 + \sum_{n=1}^{j}\prod_{l=0}^{n-1} e^{\Delta\hat{t}_{j-l}L} \\
&\leq 1 + \sum_{n=1}^{j} e^{\sum_{l=0}^{n-1}\Delta\hat{t}_{j-l}L} \\
&\leq 1 + \sum_{n=1}^{j} e^{\Delta T L} \\
&\leq (j+1)e^{\Delta T L} \\
&\leq m e^{\Delta T L}
\end{aligned} \tag{3.2.8}$$

where the second to last inequality uses the fact that $1 \leq e^{\Delta T L}$, and the final inequality follows from the fact that j indexes the substeps, implying that $j+1 \leq m$. Applying eqs. (3.2.7) and (3.2.8) to eq. (3.2.6) yields

$$\begin{aligned}
\left|e_{j+1}^{k+1}\right| &\leq e^{\Delta T L}\left|e_0^{k+1}\right| + \left[\tilde{B}L\Delta T\left\|\vec{e}^{\,k}\right\|_\infty + M(\Delta T)^{q+1}\right] m e^{\Delta T L} \\
&\leq e^{L(t_F - t_I)}\left|e_0\right| + m e^{L(t_F-t_I)}\left[\tilde{B}L\Delta T\left\|\vec{e}^{\,k}\right\|_\infty + M(\Delta T)^{q+1}\right]
\end{aligned}$$

Note we can drop the superscript on the term e_0^{k+1} and simply just write e_0 because the initial condition is never corrected. This inequality holds for all j, so we can rewrite the left-hand side of the inequality as

$$\left\|\vec{e}^{k+1}\right\| \leq e^{L(t_F-t_I)}\left|e_0\right| + me^{L(t_F-t_I)}\left[\tilde{B}L\Delta T\left\|\vec{e}^{k}\right\|_\infty + M(\Delta T)^{q+1}\right]$$

Therefore, letting

$$C_1 = e^{L(t_F-t_I)}, \quad C_2 = m\tilde{B}Le^{L(t_F-t_I)}, \quad C_3 = me^{L(t_F-t_I)}M$$

concludes the proof. □

An immediate consequence of Theorem 1 is the following corollary, which explicitly demonstrates the order of accuracy of explicit SplineDC methods.

Corollary 1. *Suppose all of the assumptions of Theorem 1 are satisfied, which also assumes that the assumptions of Lemma 1 are satisfied. Furthermore, suppose S correction sweeps are used on each time step. The error associated with the numerical solution at t_F, $\varphi(t_F)$, satisfies*

$$\|y(t_F) - \varphi(t_F)\| = \mathcal{O}(\Delta T^{\min(p_0+S,q)})$$

where p_0 is the order of accuracy of the method used to compute the provisional solution.

Proof. Suppose we discretize the time interval $[t_I, t_F]$ into N time steps of size ΔT. On an arbitrary time step, the error after S correction sweeps satisfies

$$\left\|\vec{e}^{S}\right\| \leq C_1|e_0| + C_2\Delta T\left\|\vec{e}^{S-1}\right\| + C_3(\Delta T)^{q+1}$$

by Theorem 1. Iterating this recursive relationship down to $\vec{e}^{0}$, the error associated with the

provisional solution, yields

$$\begin{aligned}\left\|\vec{e}^{S}\right\| &\leq (C_2\Delta T)^S \left\|\vec{e}^{0}\right\| + \left[C_1|e_0| + C_3(\Delta T)^{q+1}\right]\sum_{k=0}^{S-1}(C_2\Delta T)^k \\ &\leq C_2^S(\Delta T)^{S+p_0} + \left[C_1|e_0| + C_3(\Delta T)^{q+1}\right]\sum_{k=0}^{S-1}(C_2\Delta T)^k\end{aligned}$$

where the second inequality uses the fact that $\|\vec{e}^{0}\|$ is the error associated with an order-p_0 method. Note the subtle difference in notation: e_0 is the error associated with the initial condition on the current time step while $\vec{e}^{0}$ is the error asociated with the provisional solution on the current time step. For ΔT sufficiently small, we may omit the higher-order-terms, yielding

$$\left\|\vec{e}^{S}\right\| \leq C_2^S(\Delta T)^{S+p_0} + C_1|e_0| + C_3(\Delta T)^{q+1}$$

Omitting the higher-order-terms here amounts to rewriting the right-hand side expression as

$$\left\|\vec{e}^{S}\right\| \leq C_1|e_0| + \tilde{C}(\Delta T)^{\min(p_0+S,q+1)}$$

where $\tilde{C} \geq \max(C_2^S, C_3)$. This is the amount of error that is made on each time step of size ΔT after S sweeps. Thus, the error associated with the numerical solution after N time steps of size ΔT, i.e. the error at t_F, will be

$$\mathcal{O}(\Delta T^{\min(p_0+S,q)})$$

Note that we only need to divide a factor of ΔT into the second argument of the minimum function because we can easily perform one extra correction sweep. □

3.3 Implicit SplineDC Convergence Theorem

Here, we present the proof for the convergence of implicit SplineDC methods backward Euler to advance the correction sweeps. It is essentially an analogous proof to the explicit case where some of the bounds we establish above need to be handled differently.

Theorem 2. *Suppose all of the assumptions of Lemma 1 are satisfied. Fixing* $\mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ *and* $\mathcal{S}$*, the error of the* $(k+1)$*th sweep of implicit SplineDC on one time step satisfies*

$$\left\|\vec{e}^{k+1}\right\|_\infty \leq C_1|e_0| + C_2\Delta T\left\|\vec{e}^{k}\right\|_\infty + C_3\Delta T^{q+1} \tag{3.3.1}$$

for all ΔT *such that* $\Delta\hat{t}_j \leq 1/2L$ *for all* $j = 0,\ldots,m-1$*, where* L *is the Lipschitz constant of* F*. The constants* C_1, C_2*, and* C_3 *only depend on* F*, the spline operator* $\mathcal{S}$*, the number of substeps* m*, and the length of the entire time interval,* $[t_I, t_F]$*, of interest.*

Proof. We proceed in an analogous way to the previous proof. First, recall the automatic update of implicit SplineDC methods for $j = 0,\ldots,m-1$:

$$\varphi_{j+1}^{k+1} = \varphi_j^{k+1} + \Delta\hat{t}_j\left[F(\hat{t}_{j+1},\varphi_{j+1}^{k+1}) - F(\hat{t}_{j+1},\varphi_{j+1}^{k})\right] + \int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi}^{\,k})\right](t)\,dt \tag{3.3.2}$$

Subtracting eq. (3.2.3) from eq. (3.3.2) and letting e_j^k be the error at $\hat{t}_j$ of the kth correction sweep yields

$$e_{j+1}^{k+1} = e_j^{k+1} + \Delta\hat{t}_j\left[F(\hat{t}_{j+1},\varphi_{j+1}^{k+1}) - F(\hat{t}_{j+1},\varphi_{j+1}^{k})\right] + \int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi}^{\,k})\right](t) - F(t,y(t))\,dt$$

Taking an absolute value of both sides, applying the triangle inequality, and using Lemma

1 gives us

$$\begin{aligned}\left|e_{j+1}^{k+1}\right| &\leq \left|e_{j}^{k+1}\right| + \Delta\hat{t}_j \left|F(\hat{t}_j, \varphi_{j+1}^{k+1}) - F(\hat{t}_{j+1}, \varphi_{j+1}^{k})\right| + \left|\int_{\hat{t}_j}^{\hat{t}_{j+1}} \mathcal{S}\left[\vec{F}(\vec{\varphi}^{\,k})\right](t) - F(t, y(t))\, dt\right| \\ &\leq \left|e_{j}^{k+1}\right| + \Delta\hat{t}_j L \left|\varphi_{j+1}^{k+1} - \varphi_{j+1}^{k}\right| + BL\Delta T||\vec{e}^{\,k}||_\infty + M(\Delta T)^{q+1} \qquad (3.3.3)\end{aligned}$$

Expanding the second term above allows us to bound it by

$$\begin{aligned}\left|\varphi_{j+1}^{k+1} - \varphi_{j+1}^{k}\right| &\leq \left|\varphi_{j+1}^{k+1} - y(\hat{t}_{j+1})\right| + \left|y(\hat{t}_{j+1}) - \varphi_{j+1}^{k}\right| \\ &\leq \left|e_{j+1}^{k+1}\right| + \left|e_{j+1}^{k}\right| \\ &\leq \left|e_{j+1}^{k+1}\right| + \left\|\vec{e}^{\,k}\right\|_\infty\end{aligned}$$

Combining the above with eq. (3.3.3) yields

$$\begin{aligned}\left|e_{j+1}^{k+1}\right| &\leq \left|e_{j}^{k+1}\right| + \Delta\hat{t}_j L \left|e_{j+1}^{k+1}\right| + \Delta\hat{t}_j L \left\|\vec{e}^{\,k}\right\|_\infty + BL\Delta T||\vec{e}^{\,k}||_\infty + M(\Delta T)^{q+1} \\ &\leq \left|e_{j}^{k+1}\right| + \Delta\hat{t}_j L \left|e_{j+1}^{k+1}\right| + \tilde{B}L\Delta T \left\|\vec{e}^{\,k}\right\|_\infty + M(\Delta T)^{q+1}\end{aligned}$$

where we used $\Delta\hat{t}_j \leq \Delta T$ and let $\tilde{B} = 1 + B$ to obtain the second inequality. Since it is assumed that $\Delta\hat{t}_j \leq 1/2L$ for all $j = 0, \ldots, m-1$, we must have $1 - \Delta\hat{t}_j L \geq 1/2$. Thus, we can subtract $\Delta\hat{t}_j L \left|e_{j+1}^{k+1}\right|$ from both sides of the above and then divide by $1 - \Delta\hat{t}_j L$ to obtain

$$\left|e_{j+1}^{k+1}\right| \leq \frac{1}{1 - \Delta\hat{t}_j L}\left(\left|e_{j}^{k+1}\right| + BL\Delta T||\vec{e}^{\,k}||_\infty + M(\Delta T)^{q+1}\right) \qquad (3.3.4)$$

Since $\Delta\hat{t}_j L \leq 1/2$, it follows that

$$\begin{aligned}\frac{1}{1-\Delta\hat{t}_j L} &= 1 + \Delta\hat{t}_j L + \left(\Delta\hat{t}_j L\right)^2 + \cdots \\ &= 1 + \Delta\hat{t}_j L\left(1 + \Delta\hat{t}_j L + \left(\Delta\hat{t}_j L\right)^2 + \cdots\right) \\ &= 1 + \frac{\Delta\hat{t}_j L}{1-\Delta\hat{t}_j L} \\ &\leq 1 + 2\Delta\hat{t}_j L\end{aligned}$$

Combining this with eq. (3.3.4) and using the same suppression of notation as the previous proof, where we let $A = \tilde{B}L\Delta T\left\|\vec{e}^{\,k}\right\|_\infty + M(\Delta T)^{q+1}$ yields

$$\left|e_{j+1}^{k+1}\right| \leq (1 + 2\Delta\hat{t}_j L)\left(\left|e_j^{k+1}\right| + A\right) \leq (1 + 2\Delta\hat{t}_j L)\left|e_j^{k+1}\right| + 2A$$

Iterating this down to $e_0^{k+1} = e_0$ gives us

$$\left|e_{j+1}^{k+1}\right| \leq |e_0|\prod_{n=0}^{j}(1 + 2\Delta\hat{t}_n L) + 2A\sum_{n=0}^{j}\prod_{l=0}^{n-1}(1 + 2\Delta\hat{t}_{j-l} L)$$

The work done in the previous proof in an effort to upper bound the product and sum that are introduced into this error expression still hold, so we can write

$$\begin{aligned}\left|e_{j+1}^{k+1}\right| &\leq e^{2\Delta TL}\left|e_0^{k+1}\right| + 2\left[\tilde{B}L\Delta T\left\|\vec{e}^{\,k}\right\|_\infty + M(\Delta T)^{q+1}\right] me^{2\Delta TL} \\ &\leq e^{2L(t_F - t_I)}|e_0| + 2me^{2L(t_F-t_I)}\left[\tilde{B}L\Delta T\left\|\vec{e}^{\,k}\right\|_\infty + M(\Delta T)^{q+1}\right]\end{aligned}$$

This holds for all j, so we can rewrite the inequality as

$$\left\|\vec{e}^{\,k+1}\right\| \leq e^{2L(t_F-t_I)}|e_0| + 2me^{2L(t_F-t_I)}\left[\tilde{B}L\Delta T\left\|\vec{e}^{\,k}\right\|_\infty + M(\Delta T)^{q+1}\right]$$

Therefore, letting

$$C_1 = e^{2L(t_F - t_I)}, \quad C_2 = 2m\tilde{B}Le^{2L(t_F - t_I)}, \quad C_3 = 2me^{2L(t_F - t_I)}M$$

concludes the proof. □

It is worth explicitly pointing out that the result of this theorem relies on the assumption that $\Delta\hat{t}_j \leq 1/2L$ for all $j = 0, \ldots, m-1$. In practice, large time steps are used in implicit methods because of their stability properties. Hence, it is very possible to use implicit SplineDC methods without satisfying the assumptions of this theorem. Further work would have to be done to account for the case when $\Delta\hat{t}_j > 1/2L$ for any $j = 0, \ldots, m-1$.

Like before, an immediate consequence of Theorem 2 is the following corollary, which explicitly demonstrates the order of accuracy of implicit SplineDC methods.

Corollary 2. *Suppose all of the assumptions of Theorem 2 are satisfied, which also assumes that the assumptions of Lemma 1 are satisfied. Furthermore, suppose S correction sweeps are used on each time step. The error associated with the numerical solution at t_F, $\varphi(t_F)$, satisfies*

$$\|y(t_F) - \varphi(t_F)\| = \mathcal{O}(\Delta T^{\min(p_0+S,q)})$$

where p_0 is the order of accuracy of the method used to compute the provisional solution.

Proof. See the proof of Corollary 1. □

Chapter 4

Accuracy and Absolute Stability of Spline Deferred Correction

4.1 Introduction

In this chapter, we explore the differences between SplineDC and SpectralDC with respect to accuracy and absolute stability of the two classes of methods. We first summarize some well-known facts regarding both properties.

Accuracy:

Suppose we are concerned with solving the initial value problem

$$y'(t) = F(t, y(t)), \quad t \in [t_0, T]$$
$$y(t_0) = y_0$$

where $y(t)$, $y_0 \in \mathbb{C}^N$ and $F : \mathbb{R} \times \mathbb{C}^N \to \mathbb{C}^N$, and $F \in C^q(\mathbb{R} \times \mathbb{C}^N)$, where $q \geq 1$. A numerical method is said to be of *order of accuracy* p if, for any sufficiently smooth right-hand side F,

$$\max_{i=0,\ldots,n} |\varphi_i - y(t_i)| = \mathcal{O}(h^p)$$

where φ_i is the numerical solution evaluated at t_i, $y(t_i)$ is the true solution evaluated at

t_i, and h is the time step size. In the case where the time step isn't uniform, we let $h = \max_{0 \leq i < n} \Delta t_i := t_{i+1} - t_i$.

Absolute stability:

We will refer to the following initial value problem as the *model problem*:

$$\begin{aligned} y'(t) &= \lambda y(t) \\ y(0) &= 1 \end{aligned} \tag{4.1.1}$$

where $\lambda \in \mathbb{C}$. Absolute stability of numerical methods for ODEs are studied by considering the use of them on the model problem. The *region of absolute stability* is defined as

$$\{\lambda \Delta t \in \mathbb{C} : |\varphi_n| \to 0 \text{ as } n \to \infty\} \tag{4.1.2}$$

assuming $y(0) \neq 0$. For one-step methods, the numerical solution obtained satisfies the following recursive relationship,

$$\varphi_n = \rho(\lambda \Delta t) \varphi_{n-1},$$

where $\rho(\lambda \Delta t)$ is referred to as the *amplification factor* of the numerical method. Hence, the region of absolute stability can be rewritten as

$$\{\lambda \Delta t \in \mathbb{C} : |\rho(\lambda \Delta t)| < 1\}$$

We note SplineDC methods can be considered as a one-step method where the substep calculations are analogous to the stage computations of a Runge-Kutta method.

In order to numerically determine the region of absolute stability for SplineDC methods, we sample the amplification factor at at a number of points in the complex plane. To this end, we note that without loss of generality, we may fix $\Delta t = 1$. Then, let $\tilde{\varphi}(\lambda)$ denote the solution, for fixed λ, obtained by using one step of the method with a time step of size 1, i.e.

$\Delta t = 1$. Since the initial condition for the model problem is $y(0) = 1$, the region of absolute stability can be viewed as

$$\{\lambda \in \mathbb{C} : |\tilde{\varphi}(\lambda)| < 1\} \tag{4.1.3}$$

Finally, assuming the amplification factor is an analytic function of λ, finding the contour of $|\tilde{\varphi}(\lambda)| = 1$ and knowing the value of $|\tilde{\varphi}(\lambda)|$ for any λ (assuming we know if λ lies inside or outside the contour) allows us determine the entire region of absolute stability.

If a method is stable for all λ such that $\mathrm{Re}(\lambda) \leq 0$, then the method is said to be *A-stable*. If a method is stable for all λ such that $180^\circ - \alpha \leq \arg(\lambda) \leq 180^\circ + \alpha$, then the method is said to be *A(α)-stable*. If in addition to the method being A-stable/$A(\alpha)$-stable we also have

$$L := \lim_{\mathrm{Re}(\lambda)\to-\infty} \mathrm{Am}(\lambda) = 0, \tag{4.1.4}$$

then the method is said to be *L-stable*/$L(\alpha)$*-stable*. L-stable and $L(\alpha)$-stable methods are known to be very good at integrating stiff equations. More technically, these numerical methods have the correct asymptotic behavior for linear problems in the stiff limit [10]. However, Layton et al. also states in [18] that if a numerical method satisfies $|L| < 1/2$, then it is sufficient for most stiff problems. We note that if L satisfies instead $0 \leq L < 1/2$, then the numerical method will yield better behavior in the stiff limit, i.e. the numerical solution won't exhibit oscillations.

In the following discussion, we will present numerical estimations of the order of accuracy of select SplineDC methods, along with plots of regions of absolute stability. All of the results will be compared to those of SpectralDC methods of equivalent order of accuracy.

4.2 Order of Accuracy

4.2.1 Numerically determining order of accuracy

Assuming that the error of SplineDC methods have an asymptotic error expansion with respect to ΔT, the time step size, we may use the method of step-doubling to obtain an estimate of the order of accuracy of the method. That is, if we compute the solution to an IVP at some time T using time steps of size ΔT, $\Delta T/2$, and $\Delta T/4$, then

$$p \approx \log_2 \frac{||\varphi_{\Delta T/2}(T) - \varphi_{\Delta T}(T)||}{||\varphi_{\Delta T/4}(T) - \varphi_{\Delta T/2}(T)||} \tag{4.2.1}$$

Note that if we happen to know $y(T)$, the true solution at time T, then we only need to compute solutions at time T using time steps of size ΔT and $\Delta T/2$ to obtain the following approximation

$$p \approx \log_2 \frac{||\varphi_{\Delta T}(T) - y(T)||}{||\varphi_{\Delta T/2}(T) - y(T)||} \tag{4.2.2}$$

4.2.2 Numerical experiments

We illustrate the order of accuracy of various SplineDC methods with two examples. The first one will be the system of three ordinary differential equations satisfied by the Jacobi elliptic functions sn, cn, and dn:

$$\begin{aligned} sn'(t) &= cn(t) \cdot dn(t) \\ cn'(t) &= -sn(t) \cdot dn(t) \\ dn'(t) &= -\mu \cdot sn(t) \cdot cn(t) \end{aligned} \tag{4.2.3}$$

where $\mu = 0.5$ on the interval $[0, 1]$ with initial condition $sn(0) = 0$, $cn(0) = 1$, and $dn(0) = 1$. According to Dutt et al. in [10], this is a common model for non-stiff problems, so we use it to demonstrate the order of accuracy of explicit SplineDC methods using quadratic, clamped

	Order estimate								
	1 sweep			2 sweeps*			3 sweeps		
Time steps	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*
1	-	-	-	-	-	-	-	-	-
2	-1.31	2.25	2.24	3.18	4.69	4.28	4.28	4.00	4.39
4	1.49	2.07	2.08	3.06	5.79	3.96	3.42	6.81	3.24
8	1.80	2.02	2.03	3.02	2.20	3.75	3.02	0.38	3.46
16	1.91	2.01	2.01	3.01	2.18	3.55	2.96	2.30	3.39
32	1.96	2.00	2.01	3.00	2.72	3.37	2.97	2.74	3.27
64	1.98	2.00	2.00	3.00	2.88	3.22	2.98	2.88	3.16

Table 4.1: Performance of explicit SplineDC with quadratic splines on eq. (4.2.3). This table displays the results of three separate runs with 1, 2, and 3 correction sweeps, respectively, each run using 5 uniform substeps per time step. The asterick denotes the optimal number of sweeps before further sweeping no longer increases the order of accuracy.

cubic splines, and quintic splines (see tables 4.1 to 4.3). To demonstrate the limiting factor of the order of accuracy of each method, we perform multiple runs of each method using a different number of correction sweeps each time. The optimal number of sweeps for each method is marked with an asterick in the table. We appeal to eq. (4.2.2) to estimate each method's order of accuracy, where the "exact" solution is computed using Matlab's built-in ode45 function, with relative tolerance set to 10^{-13} and absolute tolerance set to 10^{-15}. The number of (uniform) substeps per time step is chosen arbitrarily, and the knots (break points) of the splines are the substep nodes themselves.

Observe that each of these methods attains the order of accuracy that is equal to the order of the error bound given in Chapter 3. Furthermore, we observe that the number of sweeps is initially the limiting factor in the order of the overall method. When the number of sweeps is sufficiently large, the order becomes limited by the quadrature step, which is exactly what we expect. There are a few other points that are worth mentioning regarding these results (tables 4.1 to 4.3):

	Order estimate								
	2 sweeps			3 sweeps*			4 sweeps		
Time steps	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*
1	-	-	-	-	-	-	-	-	-
2	3.13	4.60	4.30	5.33	3.68	3.00	4.58	5.40	2.86
4	3.05	5.16	3.98	5.49	3.95	3.89	4.53	3.03	3.67
8	3.01	3.75	3.78	6.78	3.97	3.96	4.33	3.83	3.91
16	3.00	1.80	3.59	3.10	3.98	3.98	4.19	3.96	3.97
32	3.00	2.65	3.41	3.18	3.99	3.99	4.10	3.99	3.99
64	3.00	2.85	3.25	3.73	4.00	3.99	4.02	4.00	4.01

Table 4.2: Performance of explicit SplineDC with clamped cubic splines on eq. (4.2.3). This table displays the results of three separate runs with 2, 3, and 4 correction sweeps, respectively, each run using 5 uniform substeps per time step. The asterick denotes the optimal number of sweeps before further sweeping no longer increases the order of accuracy.

	Order estimate								
	4 sweeps			5 sweeps*			6 sweeps		
Time steps	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*
1	-	-	-	-	-	-	-	-	-
2	5.32	6.17	5.24	6.34	6.21	6.56	6.44	5.98	6.38
4	5.21	5.96	5.47	6.01	6.02	6.11	6.09	5.99	6.08
8	5.11	5.79	5.25	5.98	6.01	6.03	6.02	6.00	6.02
16	5.05	5.64	5.13	5.99	5.95	5.99	6.01	5.95	5.98
32	5.03	7.13	5.27	6.44	4.18	5.02	6.33	4.38	5.03

Table 4.3: Performance of explicit SplineDC with quintic splines on eq. (4.2.3). This table displays the results of three separate runs with 4, 5, and 6 correction sweeps, respectively, each run using 5 uniform substeps per time step. We omit the results from using 64 time steps because the numerical solutions are nearly accurate to machine precision, so the order estimations at that level don't make sense. The asterick denotes the optimal number of sweeps before further sweeping no longer increases the order of accuracy.

1. We do not include the results of SplineDC with natural cubic splines, even though this type of spline is quite popular. As pointed out in [9], the arbitrary choice of enforcing

zero curvature at the endpoints results in a decay in the order of accuracy of the spline, reducing it to only $\mathcal{O}(h^2)$, whereas clamped/complete cubic splines are $\mathcal{O}(h^4)$. In our experiments, we do observe this decrease in spline accuracy in the overall order of the SplineDC method.

2. In the case of SplineDC with quadratic and clamped cubic splines (in tables 4.1 and 4.2), some number of end derivatives must be satsisfied in order to uniquely determine the spline. Since the spline process constructs a spline for the right-hand side of the IVP (and not of the solution itself), one may not have derivative information readily available. There are two immediate options here: either one can analytically differentiate the right-hand side and evaluate it on the numerical solution, or one can use finite-difference schemes to approximate the endpoint derivatives from the data. In our tests, we opt for using finite-difference schemes since it is more applicable in cases where the right-hand side is difficult or expensive to differentiate. In doing so, we discovered that the approximations need to be of the same order of accuracy as the spline. Using lower order derivative approximations resulted in a reduced order of accuracy of the overall method. For example, clamped cubic splines can attain $\mathcal{O}(h^4)$ accuracy, but in order to maintain that level of accuracy, we must use a five-point one-sided finite difference scheme to approximate the endpoint derivatives. Analogously, a four-point one-sided finite difference scheme is used to approximate the left endpoint deriative for the construction of quadratic splines. This implies that, for methods where one must rely on finite-difference schemes to construct the spline, there is a minimum number of substeps that is required per time step.

3. The results of table 4.3 were generated using quintic splines. In the current literature, it's not made clear what extra conditions should be included to uniquely determine the spline in a "good" way. Hence, we have developed an alternative method of constructing splines, which is what we used to construct our quintic splines. Details of

	Order estimate								
	1 sweep			2 sweeps			3 sweeps		
Time steps	sn	cn	dn	sn	cn	dn	sn	cn	dn
1	-	-	-	-	-	-	-	-	-
2	2.61	2.30	2.29	3.13	5.53	4.49	5.65	3.71	3.06
4	0.88	2.08	2.13	3.10	3.18	4.60	7.63	4.00	3.95
8	1.66	2.02	2.05	3.05	2.00	6.20	1.88	4.01	4.00
16	1.86	2.00	2.02	3.02	2.69	1.49	3.32	4.01	4.00
32	1.93	2.00	2.01	3.01	2.87	2.27	3.75	4.00	4.00
64	1.97	2.00	2.00	3.01	2.94	2.74	3.90	4.00	4.00

Table 4.4: Performance of explicit MultiSplineDC on eq. (4.2.3). This table displays the results of three separate runs with 1, 2, and 3 correction sweeps, where each sweep uses a different spline. On the sth correction sweep, a spline of order $s+1$ is used.

this method can be found in Chapter 5.

In table 4.4, we also demonstrate the order of accuracy of a MultiSplineDC method using up to three sweeps. Recall from the previous chapter that the idea of MultiSplineDC is that, on the sth correction sweep, one should only need to use a spline that is at least $(s+1)$-order accurate. Hence, in the first, second and third sweeps, we use a linear, quadratic, and clamped cubic spline, respectively. We achieve the order of accuracy as predicted, but at a lower computational cost.

Our second example to demonstrate the order of accuracy of SplineDC methods is the Van der Pol oscillator, the well-known stiff system of two equations:

$$\begin{aligned} y_1'(t) &= y_2(t) \\ y_2'(t) &= \frac{1}{\epsilon}(1 - y_1^2(t))y_2(t) - y_1(t) \end{aligned} \tag{4.2.4}$$

where $y_1(0) = 2$, and $y_2(0) = -0.6666654321121172$. The value of ϵ dictates the stiffness of this system, and we choose two values arbitrarily to demonstrate the order of accuracy of

	Order estimate					
	1 sweep		2 sweeps*		3 sweeps	
Time steps	y_1	y_2	y_1	y_2	y_1	y_2
40	-	-	-	-	-	-
80	2.31	1.67	4.94	5.74	6.88	6.46
160	1.27	1.35	2.75	2.77	8.52	8.53
320	1.94	1.97	5.47	5.48	-2.00	-2.00
640	2.15	2.16	0.61	0.60	3.50	3.50
1280	2.13	2.14	2.57	2.57	3.56	3.56
2560	2.08	2.08	2.86	2.86	3.33	3.33

Table 4.5: Performance of implicit SplineDC with quadratic splines on eq. (4.2.4) with $\epsilon = 1/5$ on $[0, 20]$. This table displays the results of three separate runs with 1, 2, and 3 correction sweeps, respectively, with 5 uniform substeps per time step. The asterick denotes the optimal number of sweeps.

implicit SplineDC methods. The results of tables 4.5 to 4.7 are gathered from tests analogous to the above with $\epsilon = 1/5$ and time interval $[0, 20]$. For the quadratic and clamped cubic spline cases, the exact solution is computed using Matlab's built-in ode45 function with the same relative and absolute tolerances as before. Even though the IVP is mildly stiff, ode45 still does a good job (by using a lot of timesteps) in finding a well-converged solution.

For all three cases, the estimation of the order varies likely due to effects of finite precision in the order estimation procedure. However, we still have satisfactory convergence of each of the methods, i.e. the methods that use higher-order splines achieve higher overall order of accuracy on average.

In table 4.8, we present the results of implicit MultiSplineDC with three correction sweeps on a stiff case of the Van der Pol oscillator where $\epsilon = 1/1000$ and on the interval $[0, 1]$. We observe that the array of estimated orders starts off lower than what the method is supposed to attain, and then it slowly picks back up as the number of time steps is sufficiently large. This phenomenon can be considered as *order reduction*, and it occurs in Runge-Kutta methods as well [17]. Following the discussion in [18], for singly-diagonally- implicit Runge-

	Order estimate					
	2 sweep		3 sweeps*		4 sweeps	
Time steps	y_1	y_2	y_1	y_2	y_1	y_2
40	-	-	-	-	-	-
80	4.11	4.54	5.44	5.03	5.73	6.48
160	2.77	2.79	6.74	6.73	7.02	7.01
320	5.69	5.70	1.51	1.51	5.25	5.25
640	0.48	0.48	3.83	3.83	5.29	5.29
1280	2.56	2.56	4.19	4.19	4.63	4.63
2560	2.86	2.86	4.19	4.19	4.00	4.00

Table 4.6: Performance of implicit SplineDC with clamped cubic splines on eq. (4.2.4) with $\epsilon = 1/5$ on $[0, 20]$. This table displays the results of three separate runs with 2, 3, and 4 correction sweeps, respectively, with 5 uniform substeps per time step. The asterick denotes the optimal number of sweeps.

	Order estimate					
	4 sweep		5 sweeps*		6 sweeps	
Time steps	y_1	y_2	y_1	y_2	y_1	y_2
40	-	-	-	-	-	-
80	2.96	9.59	3.26	7.16	3.14	9.07
160	11.14	6.46	7.83	3.49	7.60	4.51
320	1.75	0.09	9.34	9.34	9.28	9.30
640	9.16	6.72	7.77	6.51	5.85	5.86
1280	1.70	6.66	7.99	5.82	5.89	5.87
2560	4.33	3.40	3.30	13.35	7.15	9.88

Table 4.7: Performance of implicit SplineDC with quintic splines on eq. (4.2.4) with $\epsilon = 1/5$ on $[0, 20]$. This table displays the results of three separate runs with 4, 5, and 6 correction sweeps, respectively, with 5 uniform substeps per time step. The asterick denotes the optimal number of sweeps.

Kutta (SDIRK) and additive Runge-Kutta (ARK) methods, it is assumed that the error behaves like

$$c_1(\Delta t)^\alpha + c_2\epsilon(\Delta t)^\beta$$

	Absolute error		Order estimate	
Time steps	$y_1(1)$	$y_2(1)$	y_1	y_2
1	4.160667e-05	1.750797e-01	-	-
2	2.189023e-05	4.285921e-02	0.93	2.03
4	4.171662e-06	2.084534e-03	2.39	4.36
8	1.533553e-06	2.193591e-06	1.44	9.89
16	6.343806e-07	2.248723e-08	1.27	6.61
32	2.382379e-07	1.470191e-08	1.41	0.61
64	7.678020e-08	6.911310e-09	1.63	1.09
128	1.813600e-08	1.927995e-09	2.08	1.84
256	2.347712e-09	2.232507e-10	2.95	3.11
512	1.212050e-10	2.778043e-10	4.28	-0.32
1024	1.660005e-12	1.548162e-11	6.19	4.17
2048	1.383338e-13	1.558166e-11	3.58	-0.01

Table 4.8: Performance of implicit SplineDC with clamped cubic splines on eq. (4.2.4) with $\epsilon = 1/1000$ on $[0, 0.5]$. Every time step has 5 substeps uses 3 correction sweep.

when $\epsilon < C\Delta t$, where c_1, c_2, C are constants. ϵ is the measure of stiffness for the problem, where as $\epsilon \to 0$, the problem becomes infinitely stiff. α represents the order of the method while $\beta < \alpha$ represents order reduction. If this same idea applies to implicit SplineDC methods, then the order reduction that we observe in our numerical experiment can be explained via the above. Suppose ϵ is sufficiently small for the problem to be considered stiff. For large enough Δt, the first term would dominate since ϵ is small. As Δt decreases (but not so small that $\epsilon > C\Delta t$), the second term will begin to dominate, which is when we experience order reduction. Finally, when Δt is sufficiently small (where $C\Delta < \epsilon$), we return to our classic classication of order where there is no order reduction. This is what we observe in table 4.8, and what [18] observes with implicit SpectralDC methods on stiff IVPs.

Note that in all of these cases, we advance the solution of the IVP using coarse time steps

with some number of substeps on each. In practice, this approach is generally used when one only needs to save the solution values at certain intervals, but resolving the solution requires a finer temporal mesh. This is also how one would advance the solution when using SpectralDC methods because the overall order of the method and the stability of the interpolation step is dependent upon the number of substeps. Recall that in SpectralDC, it is not advised to use uniform substeps since interoplating on uniformly placed nodes is not stable. However, as pointed out in the previous chapter, when the spline can be constructed in $\mathcal{O}(m)$ complexity where m is the number of substeps (as is the case for all of the splines we use here), there is really no need to make the value of m small. Hence, all of these results can be generated by using one time step and repeatedly doubling the number of substeps while maintaining similar computational time.

In the non-stiff and mildly stiff cases, we observe that the SplineDC methods used in this section all attain the order of accuracy as expected based upon the error bound result. In the stiff case, the method that we tested experienced what is likely to be order reduction, which can be explained if the error of SplineDC methods does indeed behave like the error of SDIRK and ARK methods, and agrees with the observations made by those using implicit SpectralDC methods.

4.2.3 Revisiting SpectralDC methods

As mentioned in the previous chapter, it has been claimed and shown in various sources [7, 10, 14–16, 23, 25] that SpectralDC methods using S correction sweeps and m substeps per time step are globally $\mathcal{O}(\Delta T^{\min(S+k_0,m+1)})$ accurate, where ΔT is the time step size and k_0 is the global order of the method used to compute the provisional solution. When others demonstrate the order of accuracy of SpectralDC methods, it is common to decide on m substeps for an order-$(m+1)$ method, and then choose the number of sweeps to exactly match that order. However, we have observed that the order of SpectralDC methods

	Order estimate								
	1 sweep*			2 sweeps			5 sweeps		
Time steps	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*
1	-	-	-	-	-	-	-	-	-
2	2.50	1.85	2.47	2.70	1.07	2.05	3.12	1.58	1.80
4	2.26	1.89	2.14	2.82	1.70	1.95	2.51	1.93	1.97
8	2.16	1.95	2.06	2.79	1.91	1.99	2.17	1.99	2.00
16	2.09	1.98	2.03	2.69	1.97	2.00	2.05	2.00	2.00
32	2.05	1.99	2.01	2.53	1.99	2.00	2.01	2.00	2.00

Table 4.9: Performance of explicit SpectralDC with 1 substep on eq. (4.2.3). This table displays the results of three separate runs with 1, 2, and 5 correction sweeps, respectively. The asterick denotes the optimal number of sweeps according to past results.

is a little bit more subtle than this, specifically with respect to the order of accuracy of the quadrature. In this section, we will bring these subtleties to light through numerical experiements, and attempt to provide an explanation

Similar to the previous section, we will consider the system of differential equations that governs the Jacobi elliptic functions eq. (4.2.3) with the same initial conditions and time interval. Tables 4.9 to 4.11 contain the estimates of the order of accuracy of various SpectralDC methods. Recall that the number of substeps on a time step dictates the type of method that is being used, i.e. it determines the degree of the interpolant being used, which effects the order. In all cases, the provisional solutions and correction sweeps are advanced with forward Euler, implying that each iteration of the method should increase the overall order of the method by one. Furthermore, our substep nodes are chosen to be the Gauss-Lobatto nodes because this family of nodes contains the endpoints.

The columns whose headings are marked with an asterick indicate the number of sweeps that papers in the past have claimed to be the optimal number of corrections sweeps needed for each method. That is, we won't see an increase in the order of the method if more sweeps are used. Recall, for a SpectralDC method that uses m substeps where the provisional

	Order estimate								
	2 sweeps*			3 sweeps			6 sweeps		
Time steps	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*
1	-	-	-	-	-	-	-	-	-
2	3.38	3.19	4.38	4.48	4.39	4.62	4.57	3.92	4.02
4	3.11	3.05	3.59	4.27	4.11	4.19	4.04	3.98	3.96
8	3.03	3.02	3.25	4.14	4.03	4.06	4.00	4.00	3.99
16	3.01	3.02	3.08	4.08	4.01	4.02	4.00	4.00	4.00
32	3.00	3.01	3.03	4.04	4.00	4.01	4.00	4.00	4.00

Table 4.10: Performance of explicit SpectralDC with 2 substeps on eq. (4.2.3). This table displays the results of three separate runs with 2, 3, and 6 correction sweeps, respectively. The asterick denotes the optimal number of sweeps according to past results.

	Order estimate								
	3 sweeps*			4 sweeps			7 sweeps		
Time steps	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*	*sn*	*cn*	*dn*
1	-	-	-	-	-	-	-	-	-
2	3.55	4.37	4.43	5.37	3.83	6.22	5.87	6.12	6.09
4	3.83	4.08	4.11	5.07	7.17	5.22	6.05	6.00	5.98
8	3.94	4.01	4.03	5.00	4.52	5.03	6.03	6.00	5.99
16	3.98	4.00	4.01	5.01	4.72	5.05	2.85	4.26	4.50
32	3.99	4.00	4.00	6.16	3.17	3.38	2.44	-0.50	0.84

Table 4.11: Performance of explicit SpectralDC with 3 substeps on eq. (4.2.3). This table displays the results of three separate runs with 3, 4, and 7 correction sweeps, respectively. The asterick denotes the optimal number of sweeps according to past results.

solution and correction sweeps are advanced using first-order methods, the optimal number of sweeps is intuitively m. The reader may want to refer to Chapter 1 for an explanation of this. However, based on the results in tables 4.9 to 4.11, this is not always the case, the exception being SpectralDC with 1 substep. For the other two cases, we observe that we can obtain more accuracy from the methods. For the case of 2 substeps (table 4.10), the

optimal number of sweeps is 3 since the estimation of the order seems to plateau at 4. For the case 3 substeps (table 4.11), the optimal number of sweeps is 5 (not explicitly shown) since the estimation of the order seems to plateau at 6. The last two rows of the the final three columns of table 4.11 show bad estimates because the numerical solution at that point is already as accurate as it can be.

These results suggest that we need to be more careful when talking about the global order of accuracy of SpectralDC methods. In the worst case, the order will be $\mathcal{O}(\Delta T^{\min(S+k_0,m+1)})$, which one can show if there is no assumption of where the substep nodes (equivalently, the interpolation locations) are. However, in every presentation of SpectralDC, the location of the substep nodes are always known since they are commonly chosen to be the nodes spectral quadrature methods, e.g. Gauss, Gauss-Lobatto, Gauss-Radau, Chebyshev, etc. As mentioned before, this doesn't guarantee that the quadrature step can be computed to spectral accuracy, but since the nodes aren't uniformly spaced, the implict Lagrange interpolation is less likely to suffer from Runge's phenomenon.

We are not able to come up with a satisfactory explanation for why the choice of Gauss-Lobatto nodes as the substep nodes in our SpectralDC methods results in only even-order accurate methods. However, the theory of Newton-Cotes quadrature might be a good starting point. This family of quadrature methods is based on exact integration of Lagrange interpolants through *uniformly spaced* points. Quoting directly from [1], part of the convergence theorem for Newton-Cotes quadrature says:

> For m even, assume $f(x)$ is $m+2$ times continuously differentiable on $[a,b]$. Then,
>
> $$\int_a^b f(x)\,dx - I_m(f) = C_m h^{m+3} f^{(m+2)}(\eta), \quad \eta \in [a,b]$$
>
> with
>
> $$C_m = \frac{1}{(m+2)!}\int_0^m \mu^2(\mu-1)\cdots(\mu-m)\,d\mu$$

$I_m(f)$ is the exact integration of the Lagrante interpolant through $f(x_i)$, where $x_i = a+ih$ for $i = 0, \ldots, m$. Note that $x_m = b$. What this part of the theorem says is that, the order of accuracy of integrating a Lagrange interpolant through an odd-number of equispaced points is 2 higher than the order of the interpolant itself, one more than the general case of integrating an arbitrary Lagrange interpolant. This suggests that the interpolation locations might be a more significant factor in SpectralDC methods that we first imagined.

We hypothesize that a similar result holds for when the interpolation nodes are chosen to be the Gauss-Lobatto quadrature nodes. This would potentially explain why we observe only even-numbered order accurate SpectralDC methods. In any case, these numerical experiments have shown us that it might be worthwhile to always run PIDC methods for a few more sweeps than expected to see if it is possible to squeeze a little bit more out of the method.

4.3 Regions of Absolute Stability

In this section, we present the stability regions of various SplineDC methods that we have previously seen, and compare them to those of SpectralDC methods of equal order. Since the development of PIDC methods are motivated by the goal of constructing high-order methods, accuracy is something we must think about as well. With this in mind, our stability region plots will also include *accuracy region* plots, regions that, for fixed threshold ϵ, we define to be

$$\{\lambda \in \mathbb{C} : |\varphi(1) - y(1)| < \epsilon\}$$

i.e. the accuracy with a time step of size $\Delta t = 1$. To remind the reader, all of the plots are generated by applying the method in question on the model problem, eq. (4.1.1).

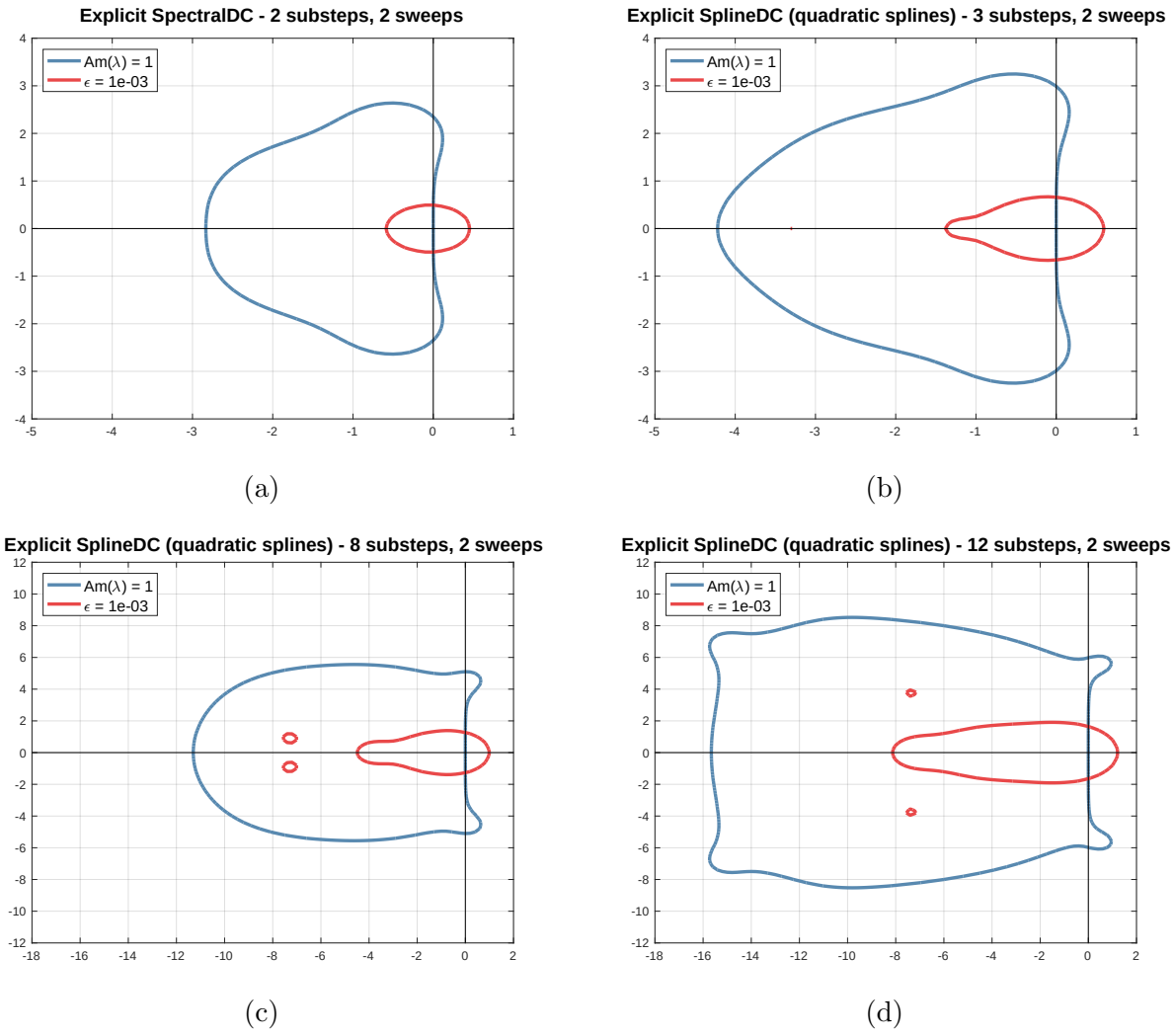


Figure 4.1: Absolute stability and accuracy regions of our 3rd-order explicit SpectralDC and SplineDC (quadratic splines) methods. The substep nodes of the SpectralDC method are the nodes of Gauss-Lobatto quadrature, while the substeps for the SplineDC method are uniform. Figures 4.1a and 4.1b compare SpectralDC with SplineDC that uses the fewest number of substeps possible, while figs. 4.1c and 4.1d demonstrate the growth in the regions as the number of substeps is increased. Note the difference in axis scaling and limits.

4.3.1 Explicit SplineDC

First, we report the absolute stability and accuracy regions of the explicit SplineDC methods from the previous section, and compare them to those of SpectralDC methods of equal order (figs. 4.1, 4.2 and 4.4). We also include the result of 4th-order MultiSplineDC in fig. 4.3. Note that all of the regions are compact, i.e. the methods are stable or accurate within the

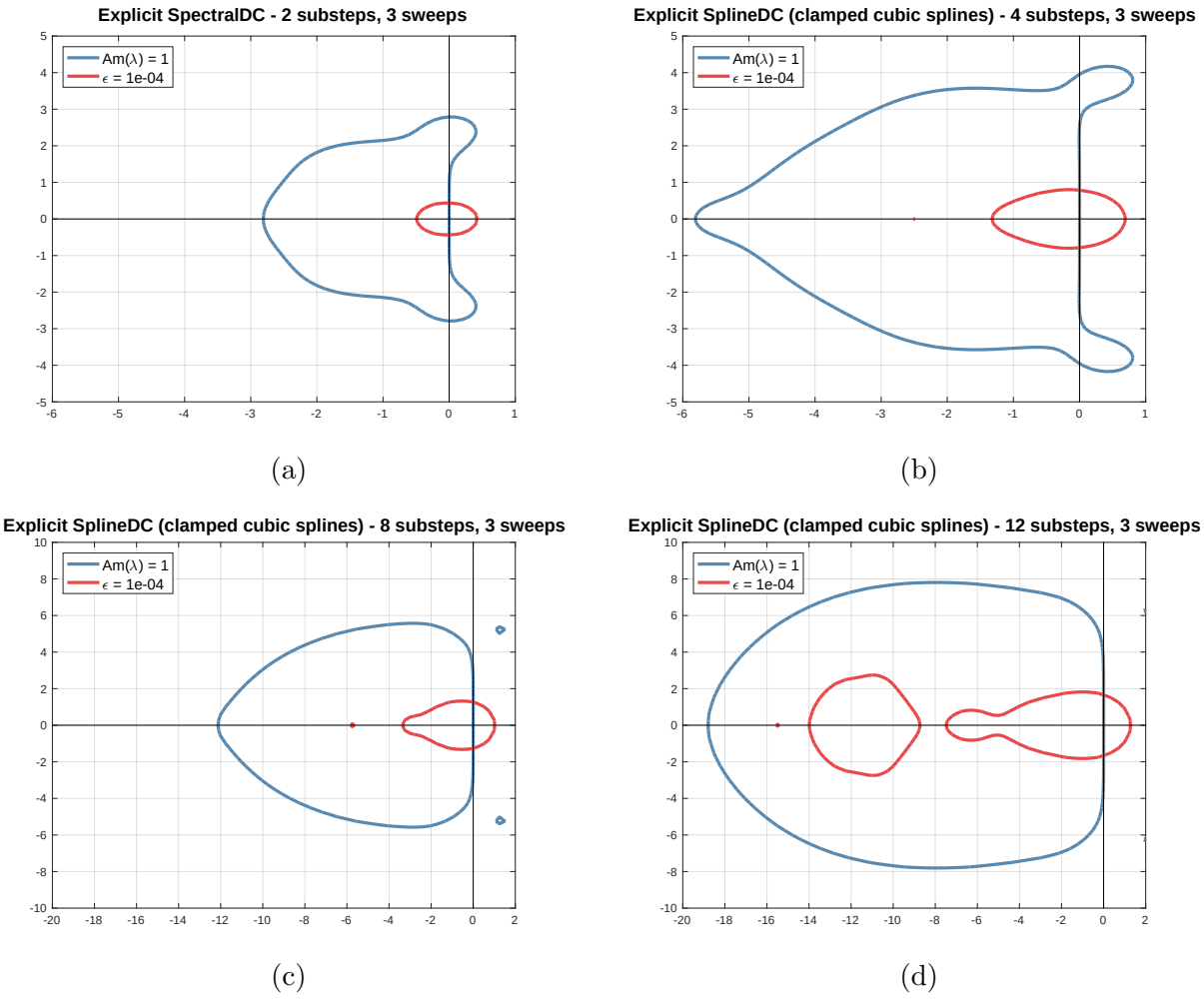


Figure 4.2: Absolute stability and accuracy regions of our 4th-order explicit SpectralDC and SplineDC (clamped cubic splines) methods. The substep nodes of the SpectralDC method are the nodes of Gauss-Lobatto quadrature, while the substeps for the SplineDC method are uniform. Figures 4.2a and 4.2b compare SpectralDC with SplineDC that uses the fewest number of substeps possible, while figs. 4.2c and 4.2d demonstrate the growth in the regions as the number of substeps is increased. Note the difference in axis scaling and limits.

appropriately marked boundaries. Though a minor point, something else to keep in mind is that the substeps of the SpectralDC methods are the (scaled and shifted) nodes of Gauss-Lobatto quadrature rules, while the substeps of the SplineDC methods are simply uniformly spaced.

From the first rows of figs. 4.1 to 4.4, we observe that SplineDC methods that use the fewest number of substeps possible has larger stability and accuracy regions than those of

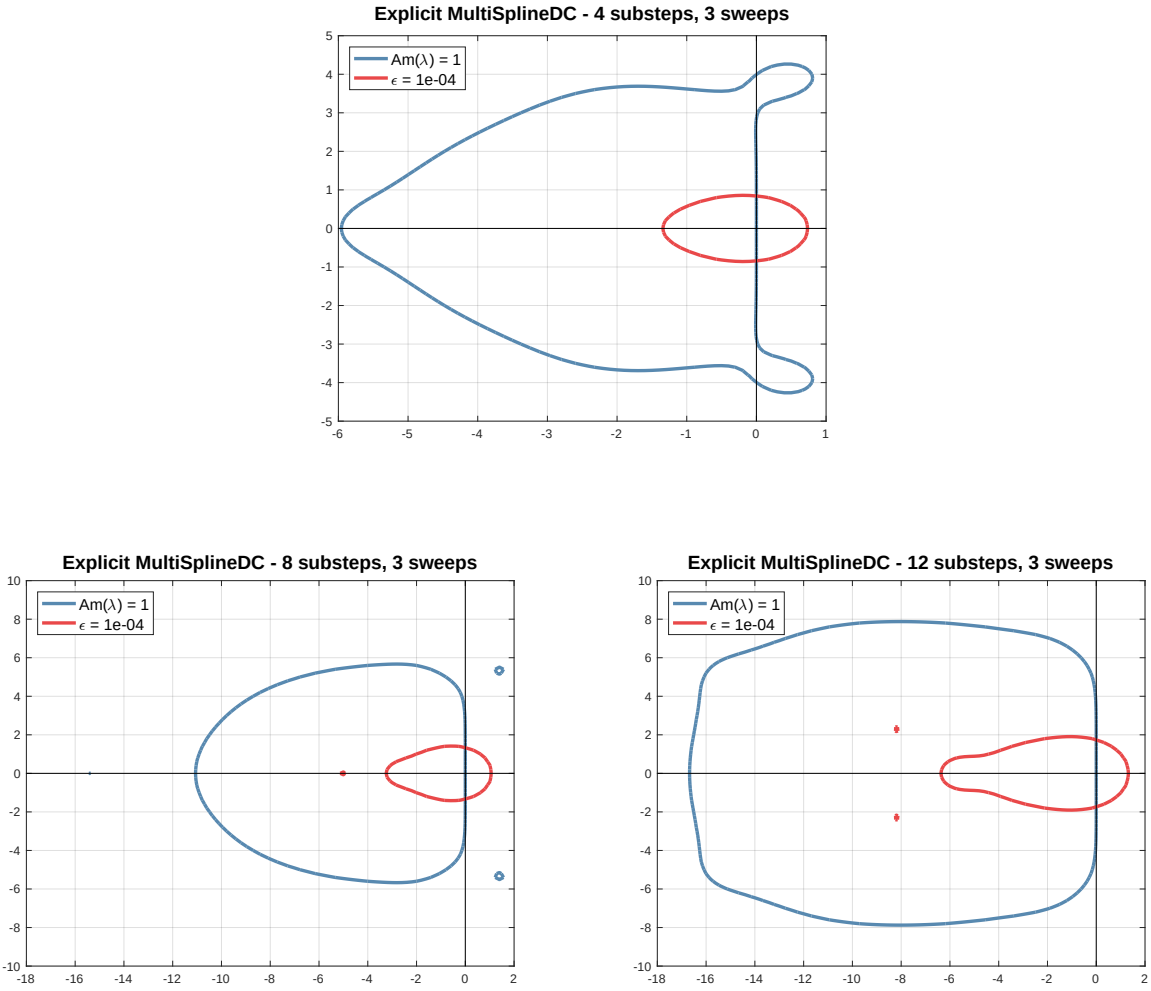


Figure 4.3: Stability and accuracy regions of our 4th-order MultiSplineDC method with different number of substeps. Recall that MultiSplineDC is SplineDC that uses different splines on each correction sweep. The first plot, which uses the minimum number of substeps, is for comparison with SpectralDC from fig. 4.2a, while the latter two demonstrate the growth in the regions as the number of substeps is increased.

SpectralDC methods of the equivalent order. Recall that, in our construction of quadratic and clamped cubic splines, one-sided finite difference schemes of the appropriate order are used to approximate the derivatives at the endpoints, implying that there is a minimum number of substeps required per timestep. Though we haven't yet discussed the specifics of how our quintic splines are constructed, there is also a minimum number of substeps required.

The last two rows of these figures demonstrate the growth in the absolute stability and

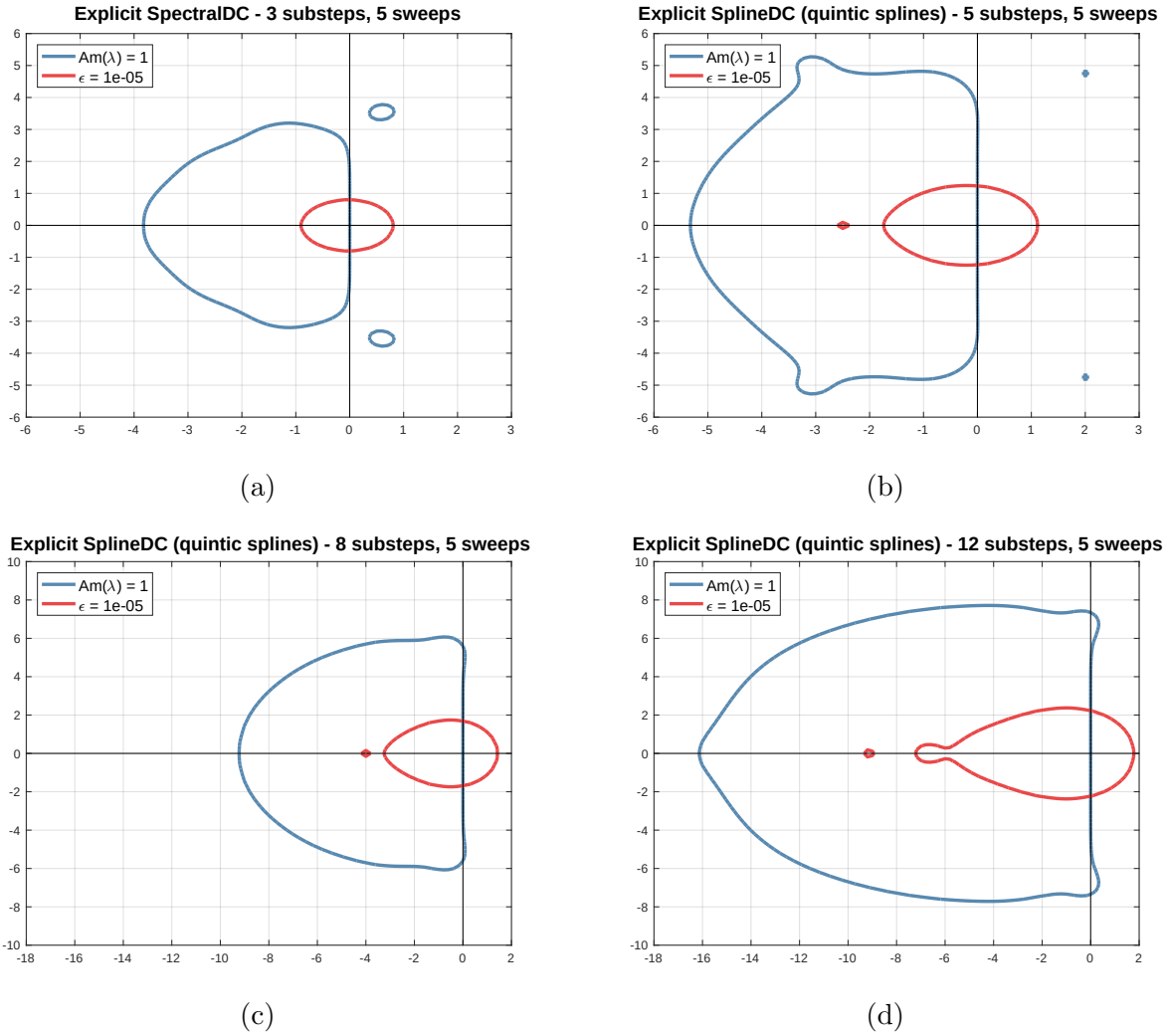


Figure 4.4: Absolute stability and accuracy regions of our 6th-order explicit SpectralDC and SplineDC (quintic splines) methods. The substep nodes of the SpectralDC method are the nodes of Gauss-Lobatto quadrature, while the substeps for the SplineDC method are uniform. Figures 4.4a and 4.4b compare SpectralDC with SplineDC that uses the fewest number of substeps possible, while figs. 4.4c and 4.4d demonstrate the growth in the regions as the number of substeps is increased. Note the difference in axis scaling and limits.

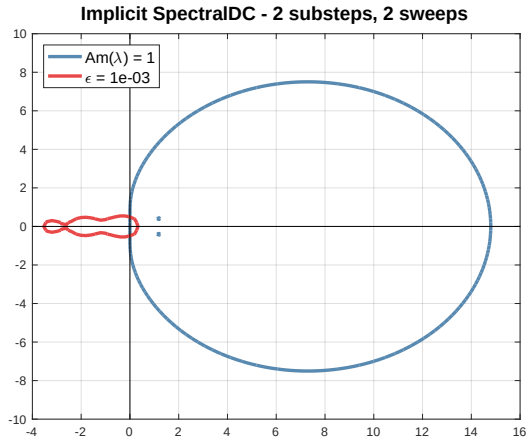
accuracy regions of the SplineDC methods as the number of substeps is increased. This is analogous to what we observe with Runge-Kutta and SpectralDC methods. However, what is fundamentally different about SplineDC methods with respect to this point is that, changes in the number of substeps do not change the overall order of accuracy of the method. Recall that in the case of SpectralDC methods, increasing the number of substeps per timestep inherently increases the order of the method. Furthermore, it's not recommended to use

SpectralDC methods with a large number of substeps because of the problems associated with high-order polynomial interpolation. How this relates to Runge-Kutta methods is that the substep computations of PIDC can be thought of as stage computations. Increasing the number of stage computations changes the Runge-Kutta method because it also increases the order of accuracy. These observations imply that we may construct explicit SplineDC methods of any fixed order with an arbitrarily large absolute stability region.

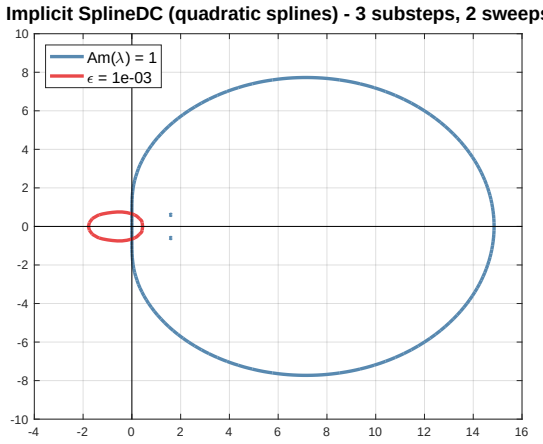
Figure 4.3 displays the stability and accuracy regions of our 4th- order explict Multi-SplineDC method. While it is also implied that the absolute stability region can be made arbitrarily large by picking more substeps, the regions are smaller than those of explicit SplineDC using clamped cubic splines (fig. 4.2), which is of the same order. This makes sense since MultiSplineDC methods incorporates the use of a variety of splines of different orders, and the results shown in fig. 4.1 have indicated that the regions of stability of explict SplineDC with quadratic splines are smaller than those of explicit SplineDC with clamped cubic splines while holding the number of substeps constant. However, we do note that absolute stability region of the MultiSplineDC method of fewest number of substeps is larger than that of the SpectralDC method of equivalent order.

4.3.2 Implicit SplineDC

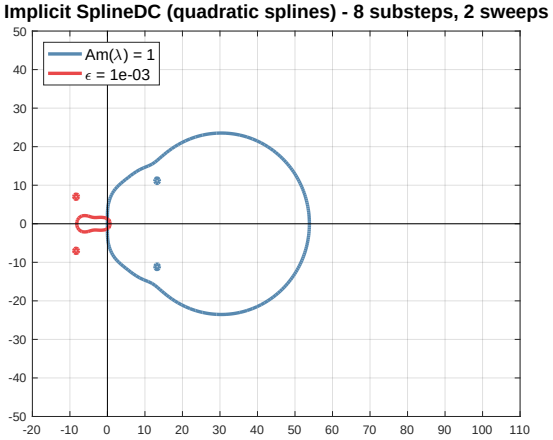
We perform analogous experiments to demonstrate the absolute stability and accuracy regions of implicit SplineDC methods from the previous section, and compare them to those of implicit SpectralDC methods of equivalent order. It is important to note that the stability regions are the unbounded regions in each of the figures, or in other words, the compliment of the compact region enclosed by the appropriate contour. The accuracy regions are still the compact regions inside the appropriate contour. Along with the plots, we also include the angle α associated with $A(\alpha)$-stability and the limit, L, of eq. (4.1.4), which indicates a measure of $L(\alpha)$-stability. As a reminder, if $\alpha = 90°$, then we can simply refer to both types



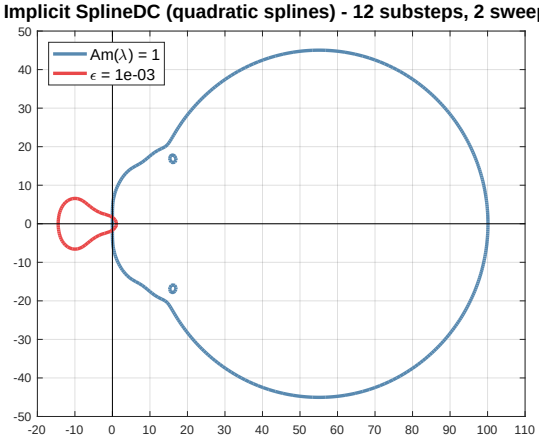
(a) $\alpha = 89.9445°$, $L = 0.4097$



(b) $\alpha = 89.9646°$, $L = -0.1258$



(c) $\alpha = 89.9933°$, $L = 0.2184$

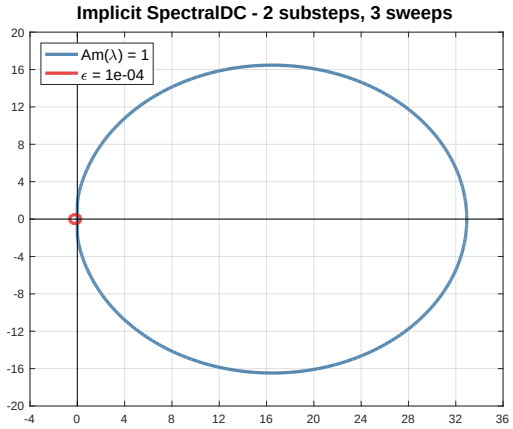


(d) $\alpha = 89.9970°$, $L = 0.2924$

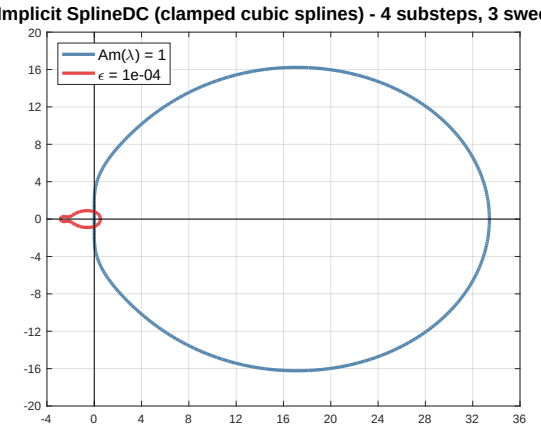
Figure 4.5: Stability and accuracy regions of 3rd-order implicit SpectralDC and SplineDC (quadratic splines) methods with different numbers of substeps. The substep nodes of the SpectralDC method are affine transformations of the nodes of Gauss-Lobatto quadrature, while the substeps for the SplineDC methods are uniform. Note the difference in axis limits and scaling of each plot.

of stability as A-stability and L-stability (dropping the α dependence).

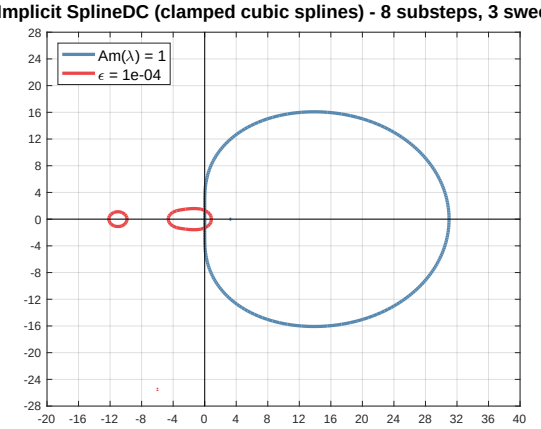
We first consider just figs. 4.5 and 4.6. When comparing the results of implicit SplineDC methods with minimum number of substeps against those of the implicit SpectralDC methods of the same order (the first rows of the figures), we notice that the absolute stability regions are approximately the same size. However, the angle α associated with the SplineDC methods are a little larger. Furthermore, the limit value L of the SplineDC methods are smaller. This implies that SplineDC methods with quadratic and clamped cubic splines are slightly



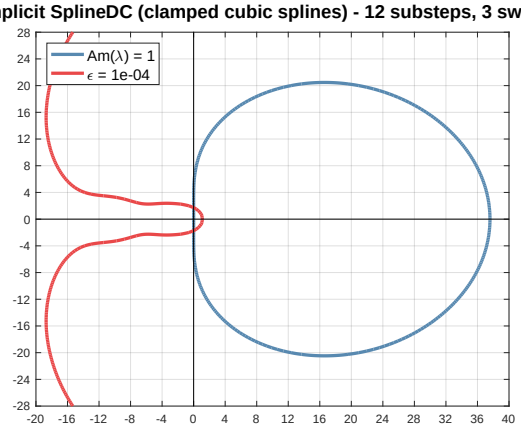
(a) $\alpha = 89.9115°$, $L = 0.4097$



(b) $\alpha = 89.9678°$, $L = 0.3049$



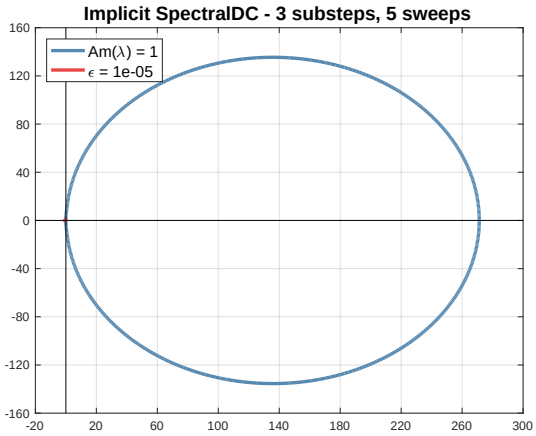
(c) $\alpha = 89.9886°$, $L = 2.363 \times 10^{-3}$



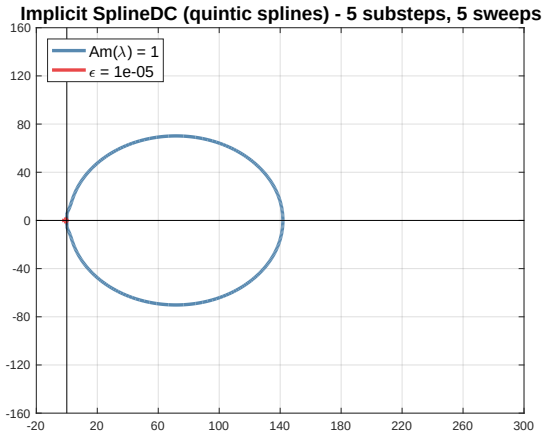
(d) $\alpha = 89.9946°$, $L = 1.924 \times 10^{-5}$

Figure 4.6: Stability and accuracy regions of 4th-order implicit SpectralDC and SplineDC (clamped cubic splines) methods with different numbers of substeps. The substep nodes of the SpectralDC method are affine transformations of the nodes of Gauss-Lobatto quadrature, while the substeps for the SplineDC methods are uniform. Note the difference in axis limits and scaling of each plot.

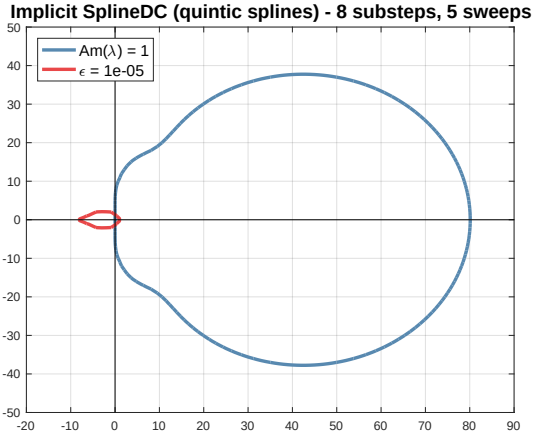
better suited for integrating stiff ODEs than the SpectralDC methods of equivalent order. According to Dutt et al. in [10], methods whose limit L are less than $1/2$ are, in practice, sufficient for most stiff problems. As the number of substeps is increased for the SplineDC methods (second rows of the figures), while the absolute stability region shrinks in the half-plane $\mathrm{Re}(\lambda) > 0$, the angle α increases toward $90°$. However, the limit value L for the quadratic case doesn't decrease while it does for the case of clamped cubic splines. On top of this, the accuracy regions in fig. 4.6 grow much faster than those in fig. 4.5. This



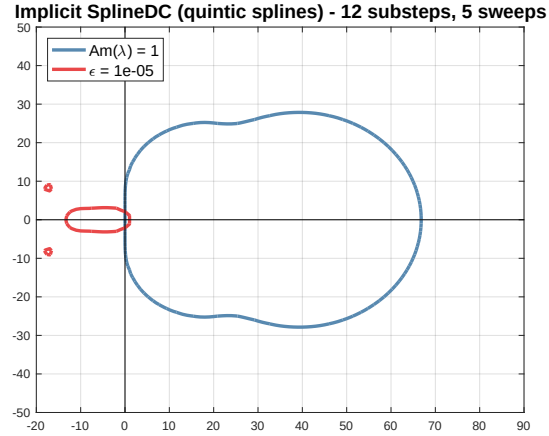
(a) $\alpha = 89.9424°$, $L = -0.902$



(b) $\alpha = 89.9703°$, $L = -0.6917$



(c) $\alpha = 89.9837°$, $L = 0.2149$



(d) $\alpha = 89.9913°$, $L = 2.323 \times 10^{-2}$

Figure 4.7: Stability and accuracy regions of 6th-order implicit SpectralDC and SplineDC (quintic splines) methods with different numbers of substeps. The substep nodes of the SpectralDC method are affine transformations of the nodes of Gauss-Lobatto quadrature, while the substeps for the SplineDC methods are uniform. Note the difference in axis limits and scaling of each plot.

suggests that it is favorable to use clamped cubic splines over quadratic splines in implicit SplineDC, and there might be a reason why. In [9], de Boor proves, if $S(x)$ is a clamped cubic spline that agrees with a twice- differentiable function f at some set of nodes, then S actually minimizes $\int_a^b g''(x)^2\,dx$ over all twice-differentiable functions g that agree with f at the aforementioned set of nodes. He refers to this as the *smoothest interpolation* property of clamped cubic splines. This property might manifest itself in the context of SplineDC by introducing extra damping (caused by the inherent smoothing) into the method itself,

allowing for a SplineDC method that is more L-stable.

Now, we consider fig. 4.7. We discuss these results separately from the previous two because our construction of quintic splines isn't what one might expect. This will be explained in detail in Chapter 6, but the main point is that extra osculatory constraints at the endpoints are not specified to close the system of equations for the coefficients. Instead, we add extra conditions on the piecewise polynomials themselves so just the interpolation conditions at the substeps are sufficient. This alternate spline construction yields very interesting results for our implicit SplineDC method. First, the absolute stability and accuracy regions of the SplineDC method of minimal number of substeps (first row) is much larger than that of the SpectralDC method of equivalent order. (As a reminder, the stability region is the compliment of the region that is enclosed by the blue contour.) Moreover, the SplineDC method yields better α and L values. Then, as the number of substeps is increased, α increases, L decreases, and the accuracy regions get larger, which implies that these methods are increasingly better for stiff problems. This is analogous to what we've been observing for implicit SplineDC methods. However, what's extremely surprising is that the stability regions grow in size in the half-plane $\mathrm{Re}(\lambda) > 0$, whereas the results for the lower-order implicit SplineDC methods shown earlier show the opposite phenomenon. This implies that our choice of construction for quintic splines yields a SplineDC method that damps. Whether or not this is favorable is up to discussion, but what can be gathered from this experiment is that spline construction makes a large difference on the characteristics of SplineDC method.

We are unable to explain the cause of growth in the absolute stability region in the half-plane $\mathrm{Re}(\lambda) > 0$ as the number of substep is increased, but we have evidence that this property comes from the alternate method of spline construction. In other experiments, we have observed the same phenomenon from implicit SplineDC that uses cubic splines constructed in a similar fashion.

Our computational experiments indicate that SplineDC and SpectralDC methods of

equivalent order have qualitatively equivalent behavior with respect to accuracy and stability when the number of substeps for the SplineDC method is at or near its minimum. Because the order of SplineDC methods is not determined by the number of substeps per time step, as is the case for SpectralDC methods, our results indicate that, for fixed order of accuracy, we can construct explicit SplineDC methods with arbitrarily large absolutey stability and accuracy regions. For the implicit cases, all of the methods that we tested were nearly A-stable, with α being very close to $90°$. With sufficiently many substeps per time step, the limiting value for checking for L-stability can be made quite small, implying that implicit SplineDC methods is better-suited for integrating stiff initial-value problems when compared to SpectralDC methods.

4.4 Comparing SplineDC to SpectralDC

We have observed that SplineDC methods achieve the expected order of accuracy on various test problems. However, having comparable rates of convergence to SpectralDC methods doesn't mean too much; one method could have much larger errors than the other while achieving the same rate of convergence. Hence, in this final section, we examine the magnitude of the errors associated with the various SplineDC methods we have already seen in comparison to those of SpectralDC methods of the same order. The figures in this section are essentially the errors associated with the step-doubling numerical experiements from the section on numerically estimating the order of accuracy. However for the SplineDC methods, we also include the case where we fix the number of time steps and repeatedly double the number of substeps. As mentioned before, this approach to time-stepping is not available to SpectralDC methods because increasing the number of substeps directly increases the degree of the implicit Lagrange interpolant, which increases the risk of experiencing Runge's phenomenon. This substep technique is worth exploring because, as results seem to suggest from solving the model problem, increasing the number of substeps causes the accuracy region of

the method to grow. Hence, we might be able to obtain more accurate solutions.

We first define some notation. Each of the figures includes five plots, and they are labeled as SpectralDC$_{\Delta T}$, SplineDC$_{\Delta T}$, MultiSplineDC$_{\Delta T}$, SplineDC$_{\Delta t}$, and MultiSplineDC$_{\Delta t}$. Note that the first three have ΔT as the subscript while the last two have Δt. A method with the ΔT subscript implies that the time-stepping refinement is done on the time steps, while the Δt subscript implies that refinement is done on the substeps (while fixing the number of time steps at 1). For the three ΔT methods, the substeps on each time step are chosen to be the nodes of a fixed Gauss-Lobatto quadrature rule since this is what is classically used for SpectralDC methods. For the two Δt methods, the substeps are uniformly spaced. Finally, as a reminder, MultiSplineDC is SplineDC where the jth correction sweep uses a spline that is of order $j + 1$. These methods achieve the same order of accuracy as classical SplineDC but at a cheaper cost.

4.4.1 Explicit methods

In this section, we present the results obtained from explicit SplineDC and SpectralDC methods. Figures 4.8 and 4.9 show the errors of these two classes of methods on the Jacobi elliptic problem eq. (4.2.3). In both figures, the errors of all the methods are more-or-less the same. This is great for the MultiSplineDC methods, indicating that even though lower-order splines are used in the earlier correction sweeps, these methods still achieve the same level of accuracy as the others. These results also seem to suggest that keeping the time step fixed and refining the substeps doesn't increase the accuracy of the solution, even though we observed earlier that an increase in the number of substeps increased the accuracy region when applying these methods to the model problem. However, we did note that across multiple runs, the Δt-methods were slightly faster than their ΔT counterparts. This might be because there is more overhead in setting up a larger number of small spline coefficient solves, while it is more efficient to set up fewer slightly larger system solves.

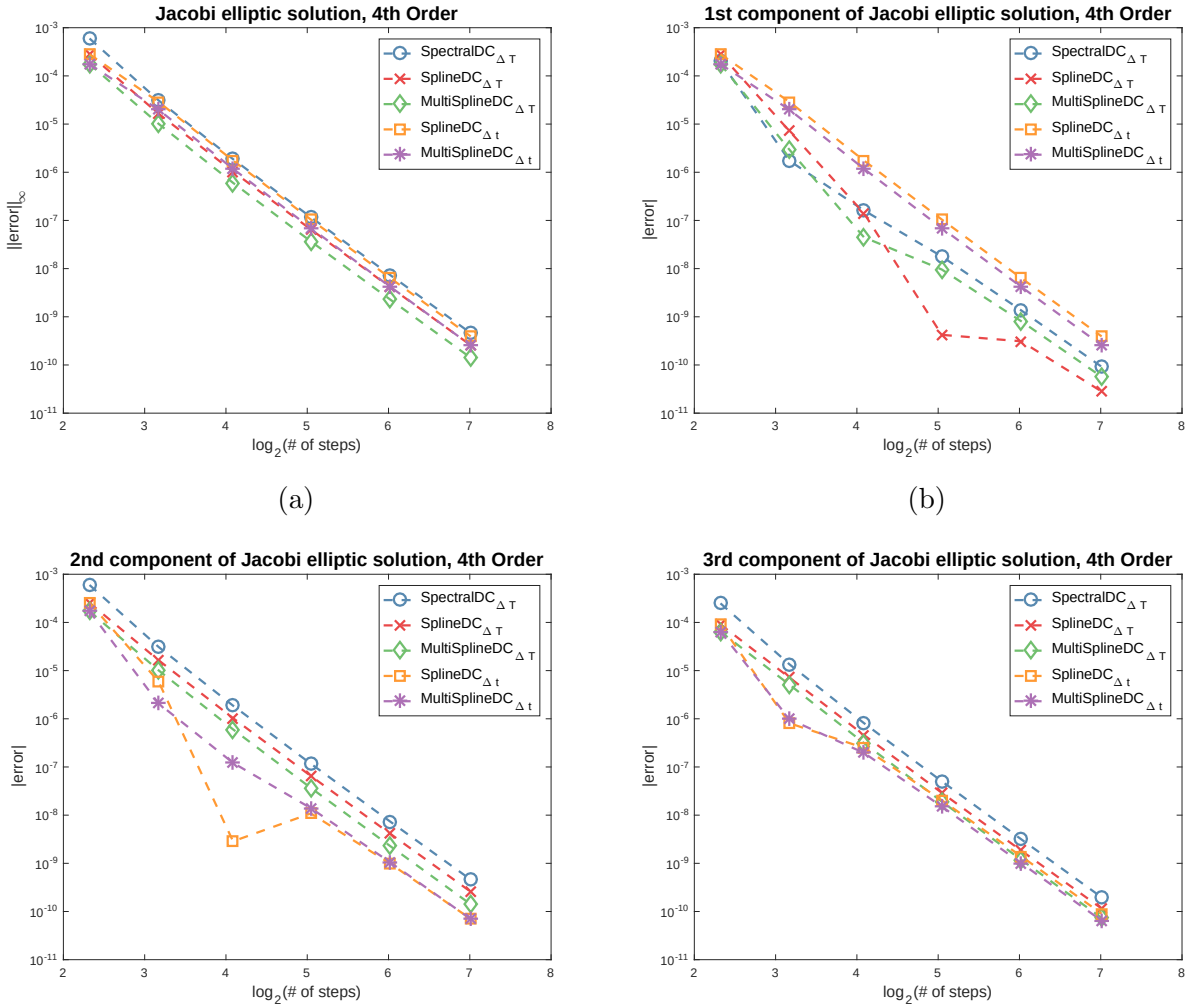


Figure 4.8: Comparison of errors from the various 4th-order methods listed in the legend on the Jacobi elliptic test problem from previous sections. Errors are computed at $t = 1$ against the solution obtained from Matlab's built-in ode45 function. Figure 4.8a shows the L_∞-norm of the error while figs. 4.8b to 4.8d shows the error associated with each component.

In fig. 4.10, we present results obtained from explicit 4th-order SplineDC and SpectralDC methods on FitzHugh-Nagumo model:

$$\begin{aligned} u'(t) &= -u(u-\theta)(u-1) - v + \delta \\ v'(t) &= \epsilon(u - \gamma v) \end{aligned} \tag{4.4.1}$$

where $\theta = 0.2$, $\delta = 0.112$, $\epsilon = 0.01$, and $\gamma = 2.5$ with $u(0) = v(0) = 0.1$ for $t \in [0, 250]$.

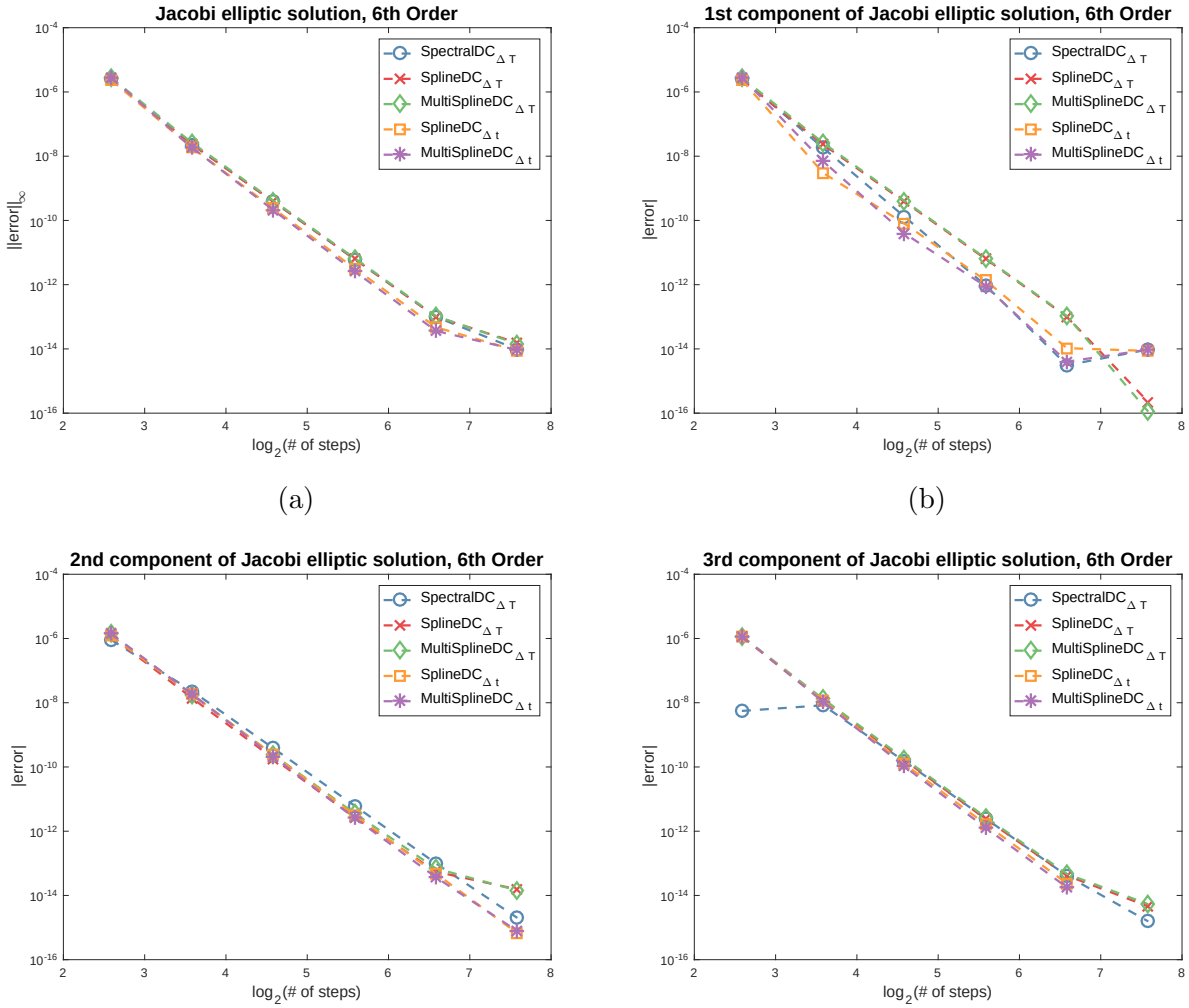


Figure 4.9: Comparison of errors from the various 6th-order methods listed in the legend on the Jacobi elliptic test problem from previous sections. Errors are computed at $t = 1$ against the solution obtained from Matlab's built-in ode45 function. Figure 4.9a shows the L_∞-norm of the error while figs. 4.9b to 4.9d shows the error associated with each component.

Similar to before, errors are computed against those obtained from Matlab's built-in ode45 function at $t = 250$. The results obtained here are more-or-less in line with what we have already seen when testing these methods on the Jacobi elliptic problem, that there's not too much of a difference between the errors of SplineDC and SpectralDC methods. Note that because the errors associated with the first component are uniformly larger than those of the second component for all of the methods as the number of steps increases, figs. 4.10a

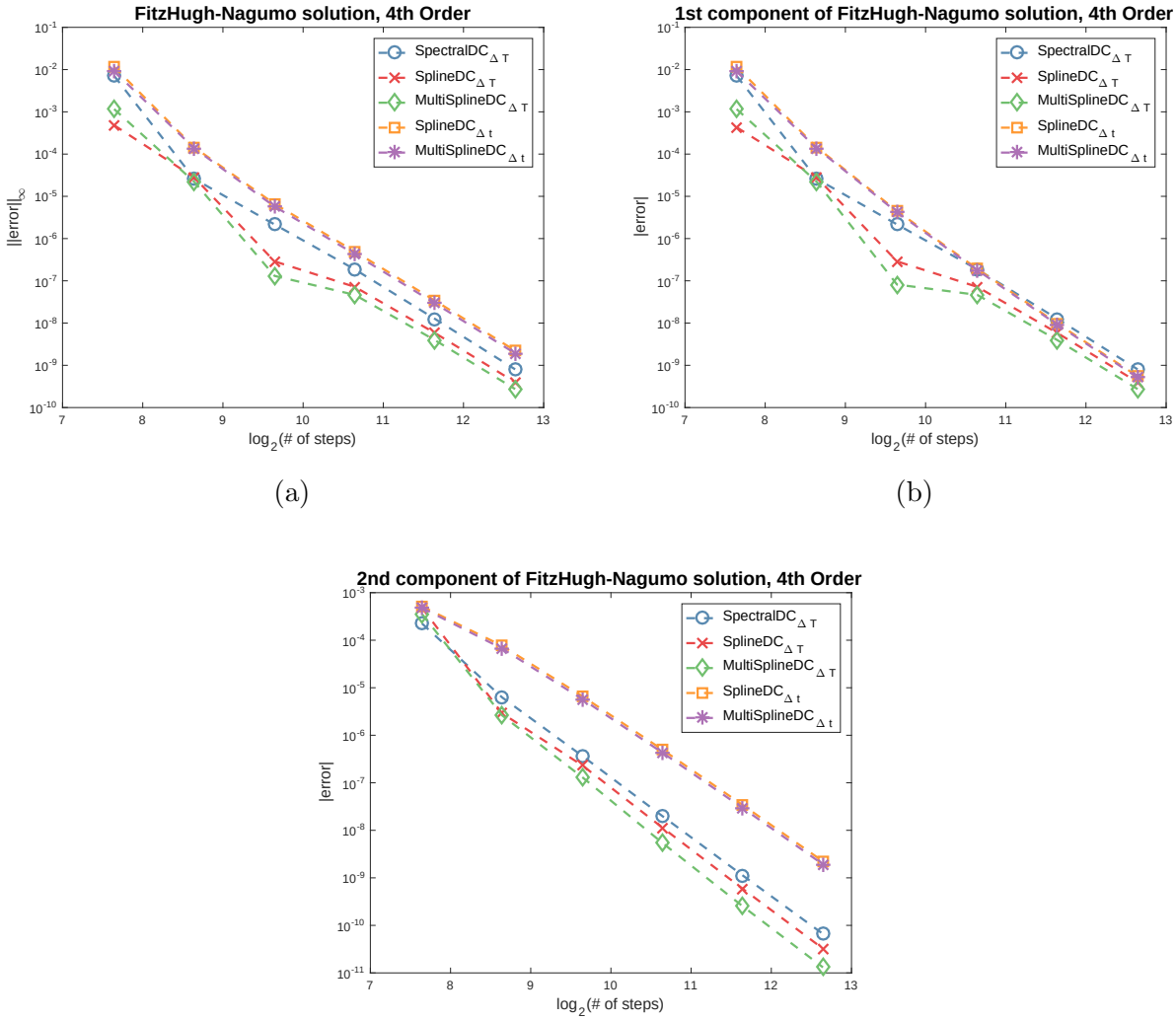


Figure 4.10: Comparison of errors from the various 4th-order methods listed in the legend on the FitzHugh-Nagumo test problem. Errors are computed at $t = 250$ against the solution obtained from Matlab's built-in ode45 function. Figure 4.10a shows the L_∞-norm of the error while figs. 4.10b and 4.10c shows the error associated with each component.

and 4.10b are actually identical.

4.4.2 Implicit methods

To finish off our comparison between the accuracy of SpectralDC and SplineDC methods, we present the results of implicit 4th-order methods on the stiff Van der Pol problem eq. (4.2.4)

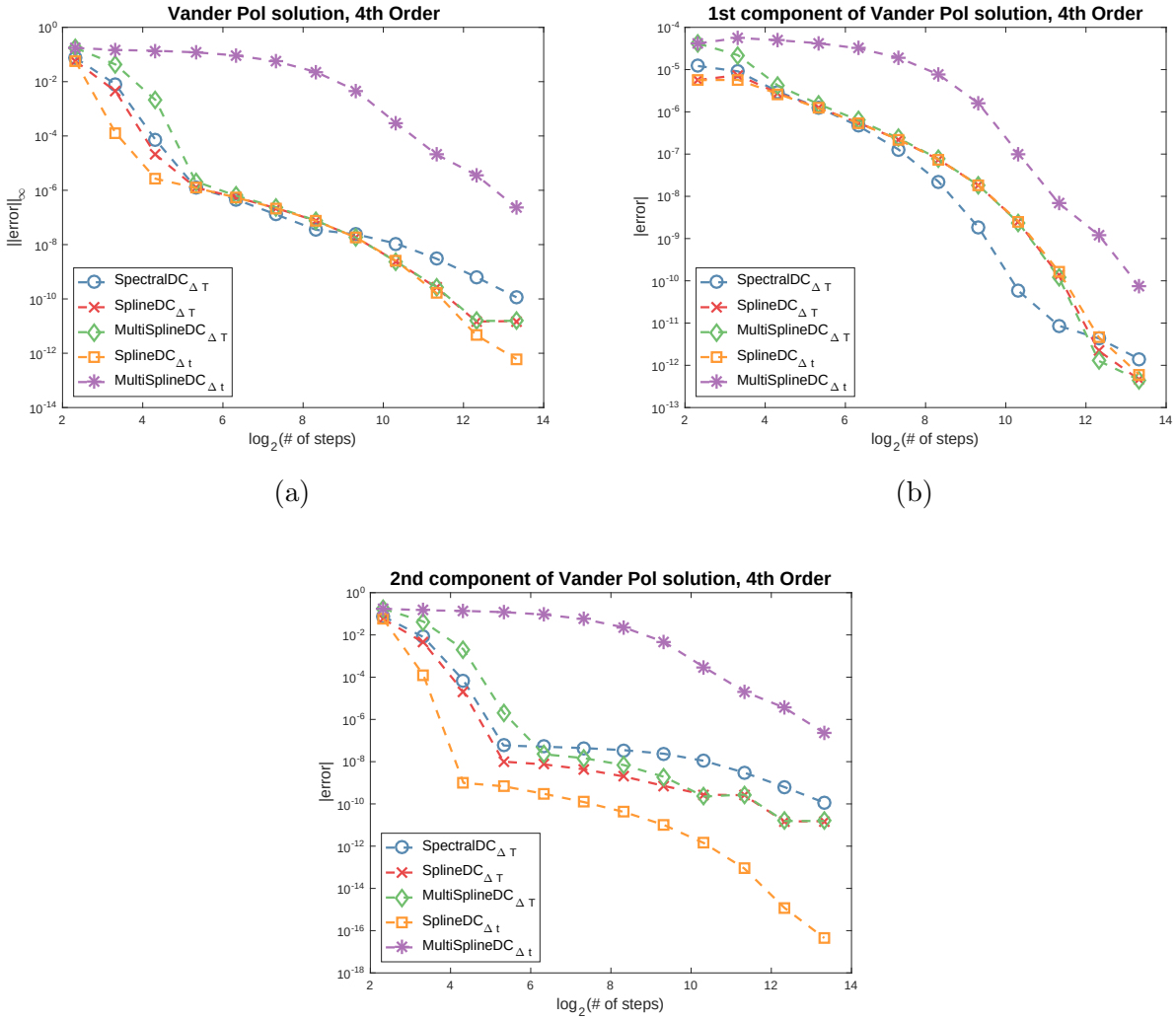


Figure 4.11: Comparison of errors from the various 4th-order methods listed in the legend on the Van der Pol test problem with $\epsilon = 1/1000$. Errors are computed at $t = 0.5$ against the solution obtained from Matlab's built-in ode45 function. Figure 4.11a shows the L_∞-norm of the error while figs. 4.11b and 4.11c shows the error associated with each component.

on the time interval $[0, 0.5]$ with $\epsilon = 1/1000$ and the same initial conditions. The results can be seen in fig. 4.11. As expected, the errors do not decrease in a well-behaved manner as the number of steps increases. This is probably the result of order reduction that we discussed earlier when numerically analyzing the rate of convergence of our 4th-order implicit SplineDC method on a stiff problem. Furthermore, we do observe that MultiSplineDC$_{\Delta t}$ does not

perform very well when compared to the other methods, especially in the 2nd component of the solution. Unfortunately, we are not sure how to explain this. However, we do see that all of the other methods perform on par with each other.

Our experiments indicate that, while SplineDC methods can easily be constructed to have better stability regions than their SpectralDC counterpart of equivalent order, their solutions are as accurate as those of SpectralDC methods. Furthermore, it's not entirely clear whether or not fixing the time step and increasing the number of substeps per time step is an advantage for SplineDC methods. Across the various numerical experiments conducted, we did observe a sweet spot in number of substeps allows the SplineDC methods to be computationally faster than methods where the number of substeps was fixed at a small number (as is the case for SpectralDC methods). Further larger- scale experiments would have to be done in a more controlled setting to make better conclusions.

Chapter 5

SplineDC and Adaptive Time-Stepping

5.1 Introduction

Adaptive time-stepping allows numerical methods for initial value problems to improve both reliability and efficiency. Methods that use this technique are able to achieve a user-specified error tolerance while picking suitable time step sizes to resolve solutions that experience varying time scales [4]. In Chapter 2, we discussed why SplineDC might be advantageous over SpectralDC methods, one of which was with respect to adaptive time-stepping. In particular, there are two ways in which SplineDC is better able to incorporate adaptive time- stepping compared to SplineDC:

1. For cases where the construction of the spline can be done in $\mathcal{O}(m)$ complexity where m is the number of spline panels, adaptive time-stepping would directly reduce the method's overall time-complexity.

2. In SpectralDC methods, only the composite time steps, ΔT_n, can be chosen adaptively, and the calculation using a candidate timestep $\tilde{\Delta T}$ may require multiple substeps to be computed before a decision to accept or reject the time step. Since SplineDC does not require substeps to be fixed in the same relative locations per composite time step, adaptive time-stepping can be used to pick the substeps. Thus, implementing adpative time-stepping in SplineDC may result in a much more efficient procedure.

In this section, we will review the major concepts of adaptive time-stepping, discuss its use in SplineDC, and report the results of several computational experiments that demonstrate its efficacy.

5.2 Adaptive Time-Stepping

Being able to employ adaptive time-stepping relies on the ability to obtain local error estimates which the method uses to modify the each time step. Here, we discuss the two common techniques for estimating local error. Fixing notation, suppose we are advancing a solution from t_n to $t_{n+1} = t_n + \Delta t$, and we would like to know the local error at t_{n+1}, which will eventually be used to decided whether to accept or reject the time step.

5.2.1 Local error estimation

The first method of local error estimation uses step-doubling. First, we estimate our solution twice at t_{n+1}: once with a full time step of Δt, and once with two half steps of $\Delta t/2$. For example, if we are using forward Euler as our numerical method (the method typically used to construct a provisional solution for PIDC), the estimation of the solution with a full time step would be

$$\varphi_{n+1} = \varphi_n + \Delta t F(t_n, \varphi_n)$$

while the estimation of the solution with two half steps would be

$$\varphi^*_{n+1} = \varphi_n + \frac{\Delta t}{2} F(t_n, \varphi_n) + \frac{\Delta t}{2} F\left(t_n + \frac{\Delta t}{2}, \varphi_n + \frac{\Delta t}{2} F(t_n, \varphi_n)\right)$$

Then, the difference between these two solutions, $\left|\varphi_{n+1} - \varphi^*_{n+1}\right|$, yields an estimate of the local error at t_{n+1} associated with the solution obtained via two half steps, φ^*_{n+1}. The derivation of this fact relies on the asymptotic expansion of the local truncation error of the numerical method being used.

Another common approach to estimating local errors uses embedded Runge-Kutta pairs [4], the most famous of which is the Runge-Kutta-Fehlberg method, a version of which is found in Matlab and GNU Octave's built-in ode45 function. This method has the same underlying idea as step- doubling: use two approximations to the solution at t_{n+1} to estimate the local error at that time. Instead of using two step sizes, this method uses two Runge-Kutta methods of orders p and $p+1$ that share stage computations, i.e. the order-p method is embedded inside the order-$(p+1)$ method. We obtain two estimates of the solution at t_{n+1} using the order p and $p+1$ methods, and the difference between the two solutions yields an estimate of the local error associated with the solution obtained via the order p method.

In our numerical experiments, we choose to use forward Euler and modified forward Euler as our embedded Runge-Kutta pair to estimate local errors. These two methods are

$$\varphi_{n+1} = \varphi_n + \Delta t F(t_n, \varphi_n) \tag{5.2.1}$$

and

$$\varphi^*_{n+1} = \varphi_n + \frac{\Delta t}{2} F(t_n, \varphi_n) + \frac{\Delta t}{2} F\left(t_n + \Delta t, \varphi_n + \Delta t F(t_n, \varphi_n)\right) \tag{5.2.2}$$

respectively. Notice the embedding of eq. (5.2.1) within eq. (5.2.2). Furthermore, once a time step is accepted, we take the estimated solution from the method of higher order, i.e. modified forward Euler, which of 2nd-order accuracy. The practice of using the high-order method is known as "local extrapolation." This technique is justified by the fact that, because of unknown instability properties of the underlying system, the local errors, in general, have little do with the global error [13]. Using local extrapolation, we can avoid one correction sweep to formally attain the maximum order of accuracy of the SplineDC method. There is no ned to use the high-order Runge-Kutta-Fehlberg pair, which combines a 4th-order method with a 5th-order method for error estimation, because that is contrary to the central idea of PIDC, which is to build high-order methods from low-order ones. Also,

though we choose not to use step-doubling in our numerical experiments, that technique is also a perfectly valid way of local error estimation within the context of SplineDC.

5.2.2 Choosing time steps

Once local error is estimated, we then need a procedure to decide to accept/reject time steps, and in the case of rejecting a time step, to pick a new one. The process we use is the one presented [13], but we include it here for completeness.

Let RTOL and ATOL be vectors of the dimension of the system of relative and absolute tolerances the user can specify, respectively. The reason for vectors of tolerances is because in the context of a system of differential equations, one may wish to use a different tolerance for different components. Using the notation from above, let φ_n be the numerical solution at t_n that has already been accepted, and let φ_{n+1} and φ^*_{n+1} be estimations of the solution at t_{n+1}, where φ^*_{n+1} was obtained by either step-doubling or an embedded Runge-Kutta pair. Then, we would like the error estimation for the ith component of the less-accurate (pth-order) method to satisfy $|\varphi_{n+1,i} - \varphi^*_{n+1,i}| < \tau_i$, where

$$\tau_i = \text{ATOL}_i + \max(|\varphi_{n,i}|, |\varphi_{n+1,i}|) \cdot \text{RTOL}_i \tag{5.2.3}$$

A measurement of the error is computed as follows:

$$\epsilon = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\varphi_{n+1,i} - \varphi^*_{n+1,i}}{\tau_i}\right)^2}, \tag{5.2.4}$$

where ϵ is used to calculate the optimal step size:

$$\Delta t_{opt} = \Delta t \left(\frac{1}{\epsilon}\right)^{1/p+1} \tag{5.2.5}$$

However, to actually determine the next time step size to be used, we introduce some safety

factors to ensure that the time step size doesn't grow or shrink too quickly. Time step sizes that grow too quickly could easily result in a large number of rejected time steps, which ultimately leads to an inefficient method. Time step sizes that shrink too quickly won't necessarily result in rejected time steps, but the numerical method is doing more work than needed, which is also inefficient. To this end, let α, $\beta_{\min}$, and $\beta_{\max}$ be the safety factors that the user can specify. The next time step can then be calculated by

$$\Delta t_{new} = \Delta t \cdot \min\left(\beta_{\max}, \max\left(\beta_{min}, \alpha\left(\frac{1}{\epsilon}\right)^{1/p+1}\right)\right) \tag{5.2.6}$$

Possible choices for α are 0.8 or 0.9 to shrink the optimal step size. $\beta_{\max} \in (1, 10]$ and $\beta_{\min} \in [0.5, 1)$ are good ranges for the other two saftey factors. These choices of safety factors in the context of eq. (5.2.6) safeguards the method from choosing time steps that grow or shrink too quickly.

Finally, note that eq. (5.2.6) covers the choice of the next time step in either case of accepting or rejecting the current one. If $\epsilon \leq 1$, then we accept the estimated solution at t_{n+1} and Δt_{new} is the new step size for the next time step. If $\epsilon > 1$, then we reject the estimated solution and retry with Δt_{new}.

5.2.3 Adaptive time-stepping in SplineDC

There are different ways in which adaptive time-stepping can be incorprated into SplineDC. We have only implemented the simplest case, which is using adpative time-stepping only in the computation of the provisional solution and keeping these substeps fixed during the correction phase. An extension of this would be to also adaptively choose the time steps used to advance the solution to the error equation. Note that in this case, interpolation would be required to incorporate the correction back into the approximate solution.

5.3 Numerical Experiments

5.3.1 Correction properties of SplineDC

For our numerical experiments, we consider the following classic example from astronomy which we will refer to as the "orbit problem" [4].

$$\begin{aligned} y_1'' &= y_1 + 2y_2' - \hat{\mu}\frac{y_1+\mu}{D_1} - \mu\frac{y_1-\hat{\mu}}{D_2} \\ y_2'' &= y_2 - 2y_1' - \hat{\mu}\frac{y_2}{D_1} - \mu\frac{y_2}{D_2} \\ D_1 &= \left((y_1+\mu)^2 + y_2^2\right)^{3/2} \\ D_2 &= \left((y_1-\hat{\mu})^2 + y_2^2\right)^{3/2} \end{aligned} \tag{5.3.1}$$

where $\mu = 0.12277471$ and $\hat{\mu} = 1 - \mu$ with initial conditions

$$\begin{aligned} y_1(0) &= 0.994 \\ y_2(0) &= 0 \\ y_1'(0) &= 0 \\ y_2'(0) &= -2.00158510637908252240537862224. \end{aligned}$$

on the interval $t \in [0, 17.065216560159625588917206 2]$. Note that at $(y_1, y_2) = (-\mu, 0)$, $D_1 = 0$ and at $(y_1, y_2) = (\hat{\mu}, 0)$, $D_2 = 0$, so when the solution is near these points, terms in eq. (5.3.1) become very large and high precision is needed; thus, the need for adaptive time-stepping.

Our first numerical experiment is done to reveal the effects of the correction sweeps. To this end, eq. (5.3.1) is solved on a single composite time step, i.e. the entire time interval of interest, and adaptive time-stepping is used to pick the substeps. This is only possible because we are using SplineDC. Furthermore, since the aim is to visualize what is happening during correction, we choose to not use local extrapolation for this experiment. The results

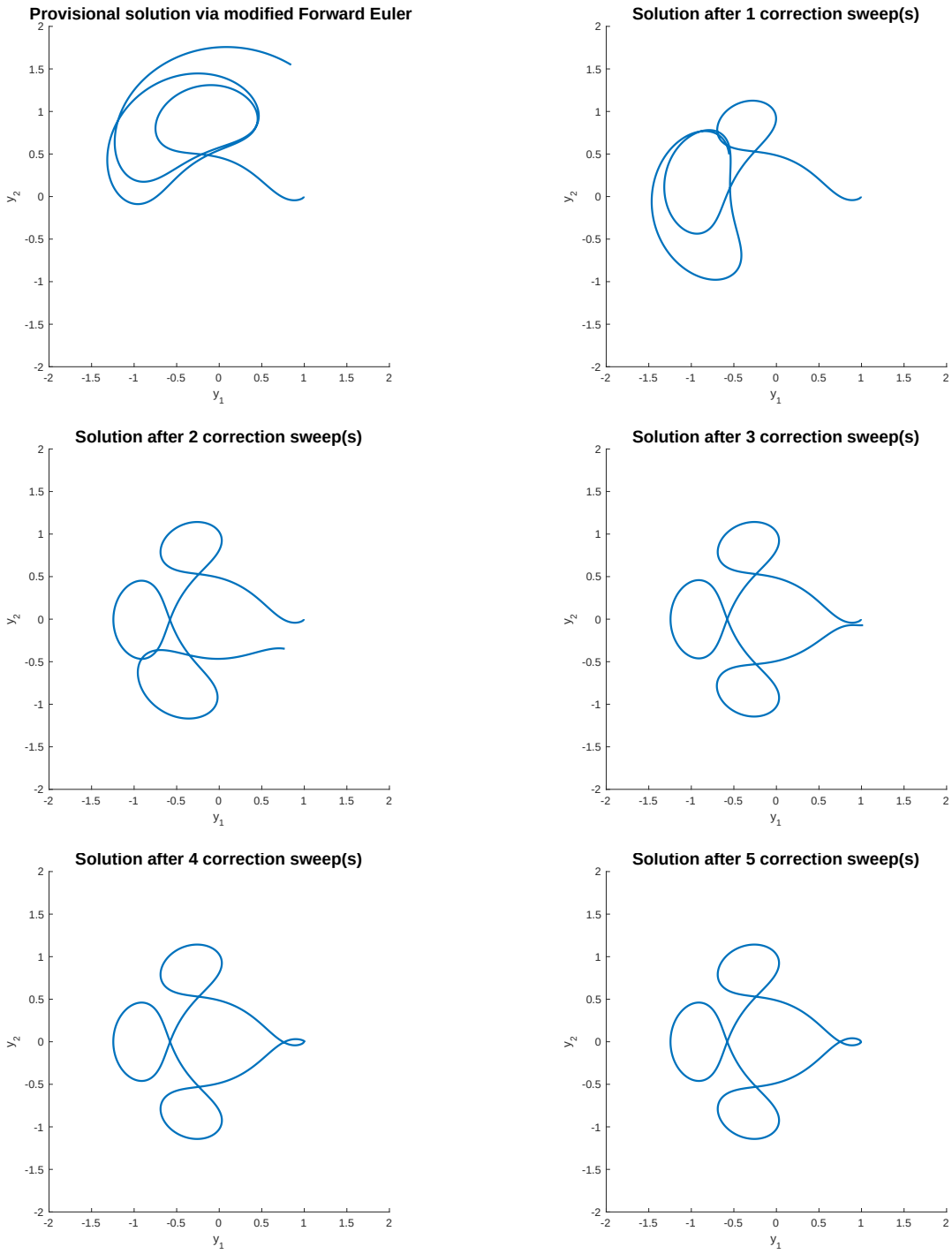


Figure 5.1: A demonstration of the correction properties of SplineDC. The entire time interval was treated as one composite time step. Although the provisional solution obtained from forward Euler with adaptive time-stepping yields an inaccurate solution, the successive correction sweeps are able to reduce the error, producing a much better approximation of the solution. RTOL $= 10^{-4}$ and ATOL $= 10^{-5}$ were used. Note the difference axis limits in some of the plots.

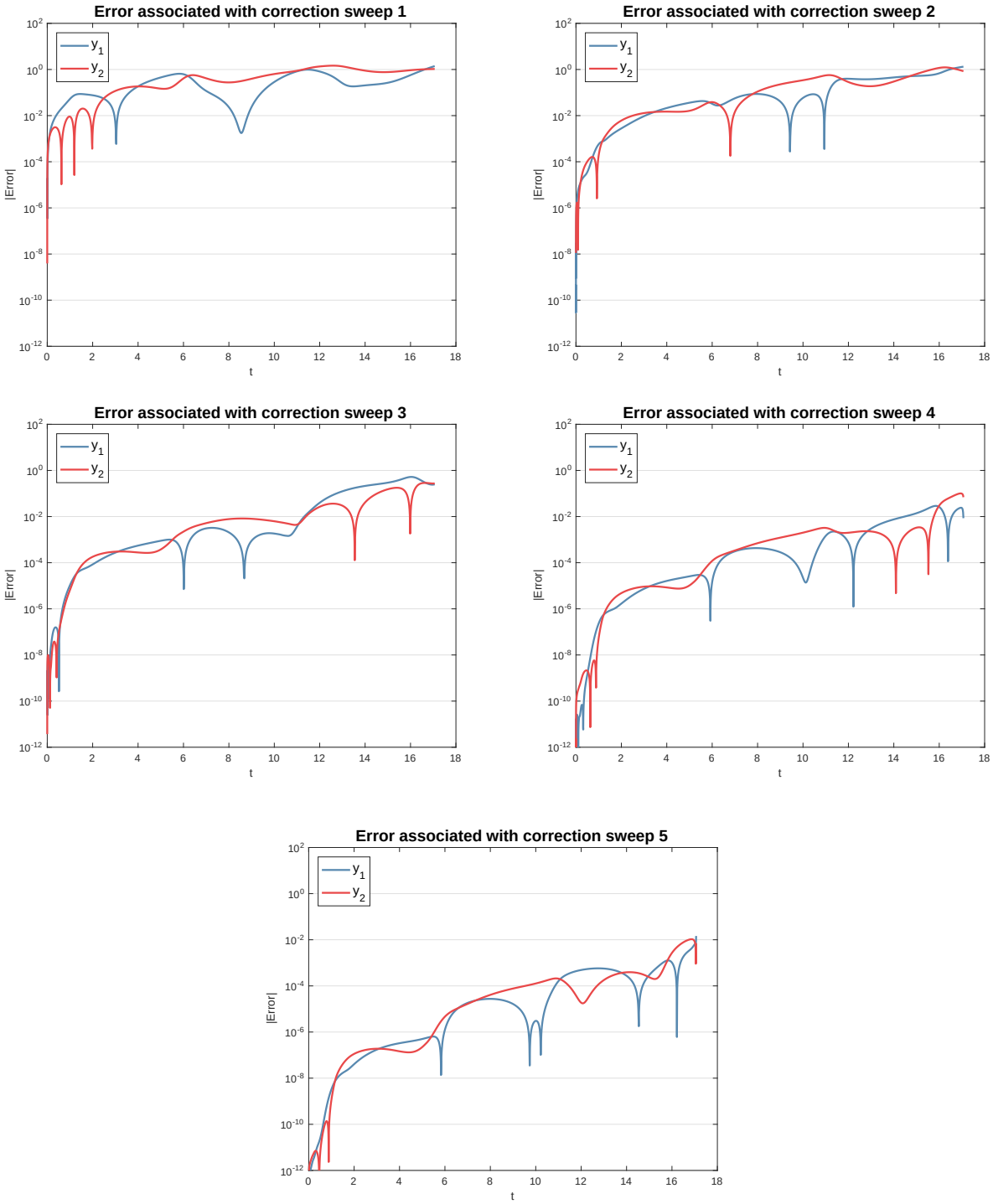


Figure 5.2: Plots of the errors associated with each of the correction sweeps of the numerical experiment seen in fig. 5.1. Errors are calculated from within the deferred correction setting.

of each of the correction sweeps can be seen in fig. 5.1, while the associated errors are in shown in fig. 5.2.

The provisional solution is computed by using forward Euler with adaptive time-stepping, where local errors were estimated using the embedded pair forward Euler and modified forward Euler. We set RTOL = 10^{-4} and ATOL = 10^{-5} to capture intended result. While the provisional solution is computed stably — a sign that adaptive time-stepping is successful — it is highly innacurate. However, successive correction sweeps using clamped cubic splines are able to reduce the initial errors, and eventually produce a better approximation of the solution.

The error plots suggest that using a large composite time step might not be the most ideal technique when using SplineDC. As expected, the larger the composite time step, the more accumulated error there will be in the solutions. This implies the need for more correction sweeps, assuming the goal of the procedure is to obtain a solution that satisfies a user-specified error tolerance. However, this concept of more accumulated errors over larger composite time steps also applies to the computation of the error (or correction) during each correction sweep as well, i.e. the corrections also suffer from more accumulated errors. We are able to observe this phenomenon in fig. 5.2. As the solution is corrected, notice how the errors associated with the beginning part of the solution decrease faster than the errors associated with the latter part of the solution. These results indicate that, while it is possible to implement SplineDC by using large composite time steps, using smaller composite time steps might lead to a more efficient method. This would bring us a little closer to the ideas of SpectralDC, but not entirely; we still wouldn't have to pick composite time step sizes small enough to keep the number of substeps on each time step relatively small. Of course, picking the composite time step size so that the error on each gets sufficientlly eliminated within some fixed number of correction sweeps would be much harder task.

The second experiment is on the effects of using different splines in the context of adaptive time-stepping. To demonstrate this, we solve the same orbit problem from eq. (5.3.1) but

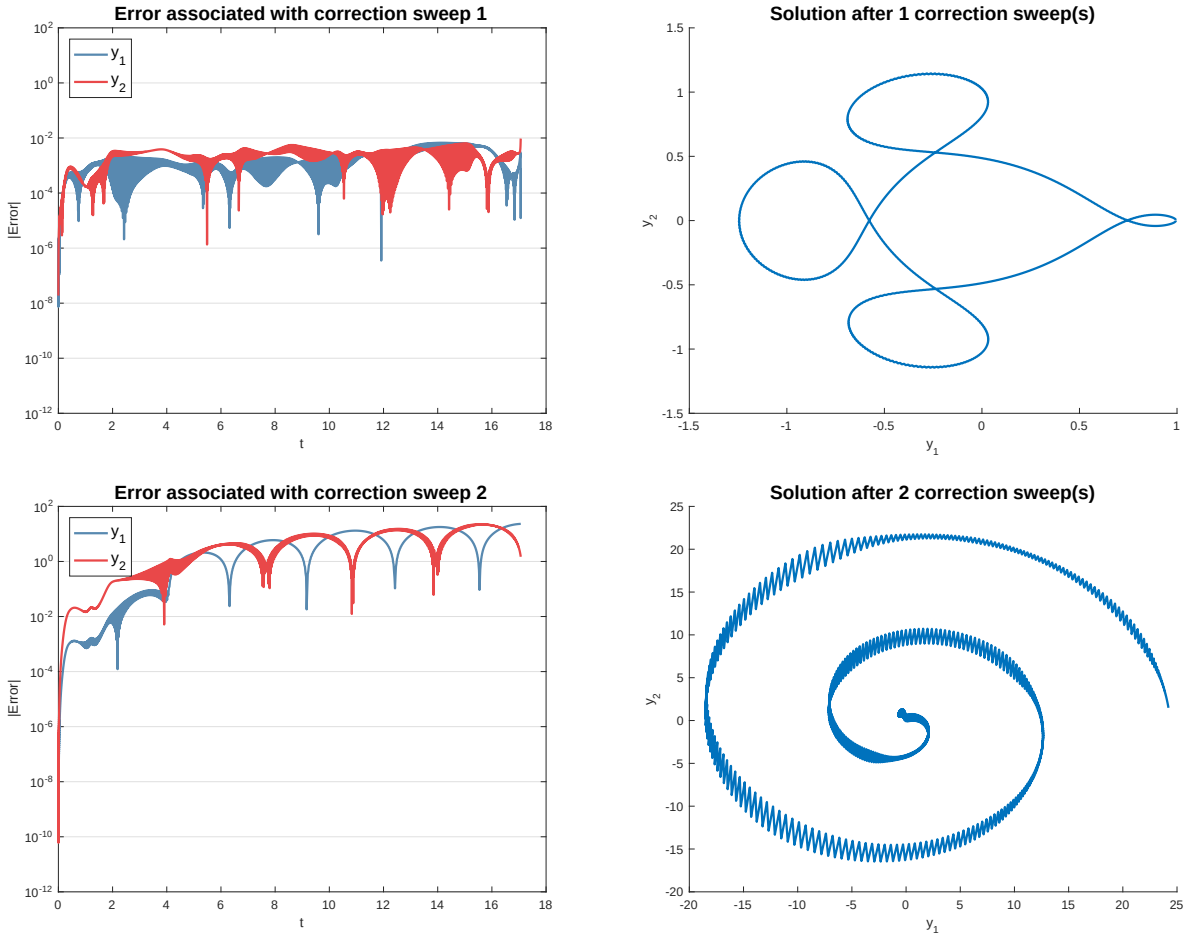


Figure 5.3: Plots of the error and solution associated with each correction sweep of SplineDC with quadratic splines. Provisional solution is (stably) computed using RTOL $= 10^{-4}$ and ATOL $= 10^{-4}$ with local extrapolation. However, instability is introduced into the computation of the error, which eventually causes the correction procedure to not converge. The thickness of the error plots are a result high frequency oscillations.

this time using local extrapolation with RTOL $= 10^{-4}$ and ATOL $= 10^{-4}$. As reference, we first use SplineDC with clamped cubic splines to solve the problem, observing that we gain two digits of accuracy between the provisional solution and the solution after 4 corrections sweeps, which is one more than needed to achieve the full order of the method. Unfortunately, anything less than 4 correction sweeps does not result in any appreciable amount of increase in accuracy. For comparison, we substitute clamped cubic splines with quadratic splines, see fig. 5.3. The quadratic splines are constructed by specifying one extra derivative condition at the left endpoint, where the derivative is computed by using a 3rd-order one-sided finite

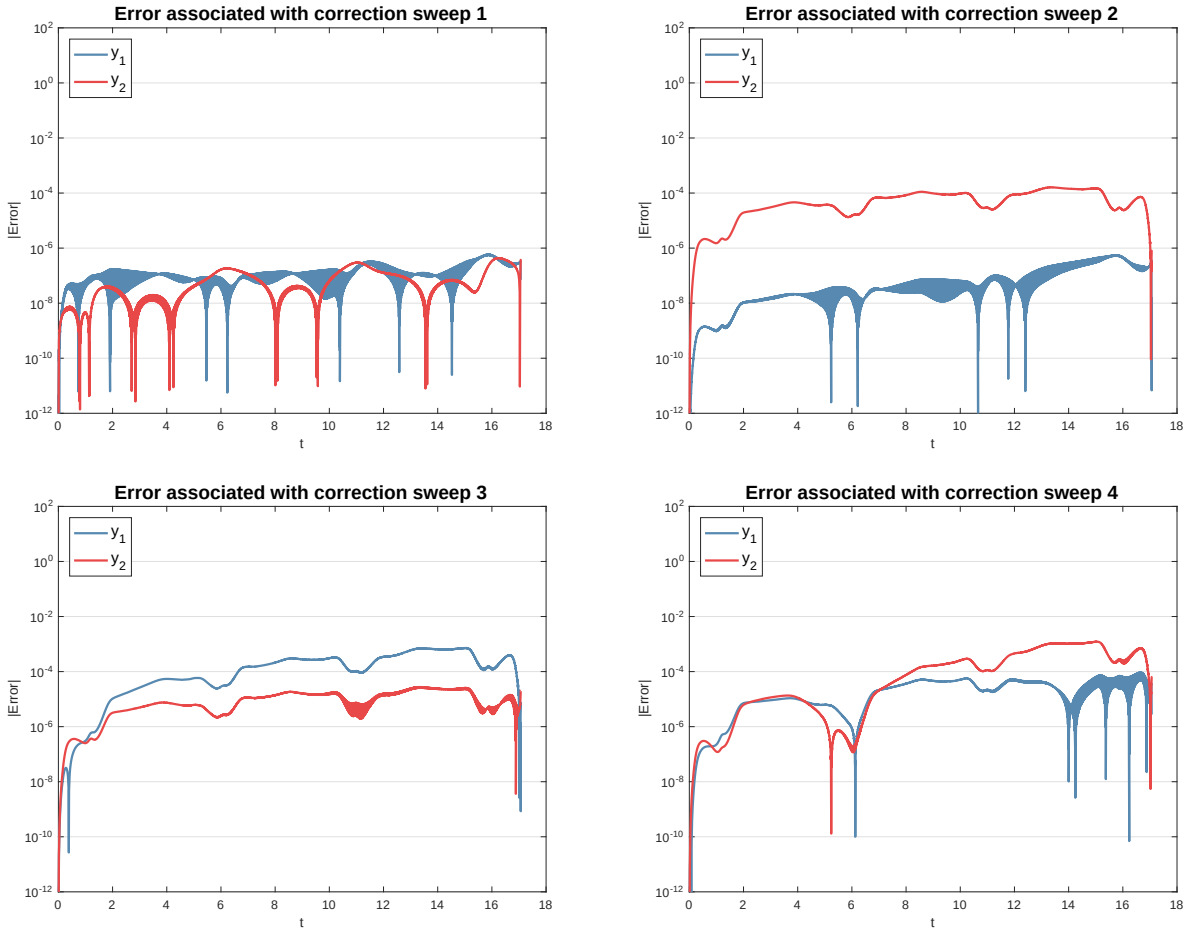


Figure 5.4: Plots of the error associated with each correction sweep. Provisional solution is (stably) computed using RTOL $= 10^{-8}$ and ATOL $= 10^{-8}$ with local extrapolation. However, instability is still introduced into the computation of the error. The thickness of the error plots are a result of high frequency oscillations.

difference approximation from the data. The provisional solution is stably computed with at least two digits of accuracy everywhere (not pictured), but the computation of the error immediately introduces instablities into procedure. The thickness of the error plots are a result of high-frequency oscillations in the error. An inspection of the solution readily explains the oscillations of the error.

We repeat the same experiment with RTOL $= 10^{-8}$ and ATOL $= 10^{-8}$ to obtain a more accurate provisional solution with a globally finer time step size, with the hope that the finer step size will prevent instablility. The results are shown in fig. 5.4. The provisional solution is computed with $146,257$ time steps on an interval of length approximately 17.1.

Upon close inspection, high-frequency regions can be seen in all of the error plots regardless of how fine the time steps are in this case.

We hypothesize that this is a result of the choice of spline. It is well-known (mentioned in the previous chapter) that clamped cubic splines have the nice property that they minimize the functional $\int_a^b f''(x)^2\,dx$ over all twice-differentiable functions that agree on some fixed set of data. In other words, they are the smoothest twice-differentiable interpolant we can use. While this smoothing property of clamped cubic splines isn't able to explain the unstable behavior we are observing in the correction sweeps associated with quadratic splines, it suggests why we don't see the same results when using clamped cubic splines.

5.3.2 Comparison between SpectralDC and SplineDC

We now compare the results of using adaptive time-stepping in both SpectralDC and SplineDC methods. In both methods, local extrapolation is used. For SplineDC, the entire time interval is adaptively partitioned into time steps because of the observation made earlier, that larger composite time steps accumulate more error. To do so, a restriction is imposed on the maximum number of substeps per composite timestep. In SpectralDC, since each time step automatically has a constant number of substeps, this restriction facilitates comparisions between SpectralDC and SplineDC.

As mentioned before, adaptive time-stepping is not as straightforward to implement for SpectralDC because of the requirement that the substeps of each composite time step must be in the same relative locations. Hence, we can only adaptive choose the composite time steps, but in order to do so, the method must evaluate the solution at the substeps of fixed relative locations. If at any substep the difference between two solution estimates of the embedded Runge-Kutta pair is too large and the time step is rejected, the method returns to the beginning of the composite time step to begin the substepping process again, this time on a smaller composite time step. Thus, the process of adaptive time-stepping is a more

RTOL \ ATOL	10^{-2}	10^{-3}	10^{-4}
10^{-2}	135 (47)	260 (101)	342 (143)
10^{-3}	166 (49)	458 (142)	819 (228)
10^{-4}	178 (70)	550 (159)	1452 (403)

Table 5.1: Number of temporal nodes picked and rejected (in parentheses) by our adaptive time-stepping technique for SplineDC with the specified RTOL and ATOL values. This is the standard adaptive time-stepping technique where an embedded Runge-Kutta pair is used to pick the substeps themselves.

RTOL \ ATOL	10^{-2}	10^{-3}	10^{-4}
10^{-2}	187 (60)	303 (80)	379 (114)
10^{-3}	203 (65)	527 (78)	1163 (120)
10^{-4}	215 (66)	727 (81)	1407 (47)

Table 5.2: Number of temporal nodes picked and rejected (in parentheses) by our adaptive time-stepping technique for SpectralDC with the specified RTOL and ATOL values. Every time step contains 2 substeps (3 substep nodes) determined from the nodes of Gauss-Lobatto quadrature.

RTOL \ ATOL	10^{-2}	10^{-3}	10^{-4}
10^{-2}	235 (100)	352 (140)	457 (180)
10^{-3}	271 (110)	547 (210)	1042 (387)
10^{-4}	289 (115)	727 (272)	1924 (681)

Table 5.3: Number of temporal nodes picked and rejected (in parentheses) by our adaptive time-stepping technique for SpectralDC with the specified RTOL and ATOL values. Every time step contains 3 substeps (4 substep nodes) determined from the nodes of Gauss-Lobatto quadrature.

expensive procedure to implement in the context of SpectralDC.

For our numerical experiments, we use SplineDC with clamped cubic splines because it seems to perform well in the context of adaptive time-stepping (from the previous section). For the SpectralDC method, each time step will contain 2 substeps, or equivalently, 3 substep nodes, which are determined by the the nodes of Gauss-Lobatto quadrature. As we have

shown in the previous chapter, this SpectralDC method actually has a maximum order of 4, which matches the order of our chosen SplineDC method.

Table 5.1 shows the time-stepping results of the computational of the provisional solution of eq. (5.3.1) for SplineDC using various values for RTOL and ATOL while tables 5.2 and 5.3 show analogous results for SpectralDC. Each element in the table displays two numbers: the first number is the final count of the total number cumulative substeps taken, while the second number in the parentheses indicates how many time steps were rejected. At least two tables (if not more) need to be shown for SpectralDC because of the fact that we need to fix the substeps per time step, which greatly affects the efficiency of the adaptive time-stepping procedure. The results strongly indicate adaptive time- stepping is better suited for SplineDC. In all of the cases, SplineDC ends up with fewer temporal nodes with fewer rejected time steps.

The three tables show results for different combinations of RTOL and ATOL chosen only from the set $\{10^{-2}, 10^{-3}, 10^{-4}\}$ for a reason. Since the provisional solution is being fed into a correction process, it doesn't need to be very accurate. Furthermore, the only way for the adaptive time-stepping procedure to yield a more accurate provisional solution is by using more steps/substeps. This automatically implies more expensive correction sweeps. Hence, it is in our interest to keep RTOL and ATOL relatively large.

Table 5.4 shows the results of both SplineDC and SpectralDC on a provisional solution constructed with RTOL = ATOL = 10^{-2}. For the SplineDC method, every 9 substeps marks a time step, i.e. all of the splines that are constructed only span a total of at most 9 substeps. The reason for choosing 9 substeps (or 10 substep nodes) is because order-wise, it is consistent with approximately how much work is needed to compute the quadratures of an implicit Lagrange interpolant over 2 substeps (or 3 substep nodes). More specifically, the amount of work it takes to construct a clamped cubic spline and to analytically integrate it is $\mathcal{O}(m)$, where $m = 9$, and the amount of work to compute the integrals of SpectralDC is $\mathcal{O}(m^2)$, where $m = 3$. During the correction phases, each method is allowed to perform

	SplineDC		SpectralDC	
Stop tolerance	RHS evals	Sweeps (avg)	RHS evals	Sweeps (avg)
10^{-4}	1495	3.80	1921	3.00
10^{-5}	1875	5.13	3151	3.71
10^{-6}	2217	6.33	3571	4.40
10^{-7}	2635	7.80	4005	5.18
10^{-8}	3034	9.20	4576	6.14
10^{-9}	3357	10.33	5096	6.98
10^{-10}	3775	11.80	5606	7.78
10^{-11}	4155	13.13	6091	8.57
10^{-12}	4554	14.53	6646	9.46

Table 5.4: Comparison between SplineDC and SpectralDC with adaptive time-stepping. RTOL = ATOL = 10^{-2} for all experiments. The column titled "RHS evals" counts the total number of function evaluations, where the function of interest is the right-hand side of the system, while the column titled "Sweeps (avg)" displays the average number of correction sweeps per time step. Stop tolerance is the threshold for the discrete L_2-norm of the error for determining when to stop the correction phase.

up to 24 correction sweeps to achieve a specific level of tolerance, which is specified in the left column. From these results, the benefit of a provisional soluion determined by adaptive time-stepping can be clearly seen. In all of the experiments, SplineDC method uses fewer function evaluations to achieve the required tolerance even though it uses, on average, more correction sweeps per time step. In the final case where stopping tolerance is set to 10^{-12}, SplineDC uses approximately 32% fewer function evaluations than SpectralDC.

The results suggest that adaptive time-stepping is something that is much more easily implemented, with a clear advantage, in the context of SplineDC. As mentioned before, note that we have only implemented adaptive time-stepping in the computation of the provisional solution. One can imagine using adaptive time-stepping as well during each correction phase. This would be warranted if the computation of the error requires a different time step stencil than what was used to construct the provisional solution.

Implementing adaptive time-stepping in the correction phases would also be easier to

do within SplineDC. The computation of the error at each time step node requires the computation of integrals of the right-hand side evaluated at the current solution. Recall that in SpectralDC, these integrals are computed by using precomputed quadrature weights, which assumes we already know the substeps the method is taking on each time step, even in the correction phase. This is no longer viable if adaptive time-stepping is used in the corection sweeps. However in SplineDC, the integrals are computed by performing exact integration of the constructed spline, implying that it would be trivial to evaluate the integrals even if the substeps are picked adaptively.

Future work in this direction would be to implement adaptive time-stepping in the correction phase to see if it is beneficial. However, from the numerical experiements that we have performed, we have not come across a case where the time-stepping chosen for the provisional solution is not sufficiently fine for the correction phase.

5.4 Generalizing SplineDC

Thus far, we have only ever referred to SplineDC as a method that begins with computing a provisional solution using forward or backward Euler, and then uses corrections sweeps to increase the order of accuracy of the solution. However, having observed the ease by which SplineDC can handle variable time step sizes when compared to SpectralDC, we can imagine applying SplineDC as a method for correcting the solution of other numerical methods. Obviously, one can also view SpectralDC similarly. However, because of the substep restrictions that are inherent to SpectralDC, it would not be as efficient.

To be more specific, suppose we obtain a solution to a differential equation from a solver that isn't as accurate as we would like it to be. This could be because we are only in the development stage of the project, or because the current solver is just a low-order prototype. We can very easily apply the correction sweeps of SplineDC on this solution to increase its accuracy while using the time step stencil provided. Partitioning the solution into multiple

composite time steps for SplineDC would be trivial. Note that using SpectralDC would be much more inefficient because we would be forced to substep on each of the time steps of the provided solution. This immediately increases the number of function evaluations of SpectralDC.

Chapter 6

Alternative Splines (AltSplines)

6.1 Introduction

SplineDC methods depend on the stable construction of splines. Given a target order of accuracy for the method, and hence for the spline, choices must be made regarding the closing of the system of equations for the spline coefficients. In this chapter, we present a different approach to constructing splines, one that is easier to implement and facilitates the exploration of the choices to be made. This is then followed by discussions and experiments that demonstrate the consequences of spline choices on SplineDC.

Fixing notation for this chapter, if $S(x)$ is a spline of degree D on the interval $[a, b]$ with respect to the break points $a = z_0 < z_1 < \cdots < z_N = b$, then, for $j = 0, \ldots, N-1$, $S_j(x)$ is the jth polynomial piece on the interval $[z_j, z_{j+1}]$ (we will also refer to these intervals as spline panels) is given by

$$S_j(x) = \sum_{k=0}^{D} a_{j,k} \left(\frac{x - z_j}{\Delta z_j} \right)^k \tag{6.1.1}$$

where $\Delta z_j := z_{j+1} - z_j$.

6.2 Spline Construction

When constructing a spline, it is commonly assumed that the data being interpolated coincide with the breaks of the spline. Specifically, if we have the data $\{(z_j, f_j)\}_{j=0}^{N}$, then we seek a spline $S(x)$ that satisfies the following:

1. $S_j^{(m)}(z_{j+1}) = S_{j+1}^{(m)}(z_{j+1})$ for $m = 0, \ldots, P$ and $j = 1, \ldots, N-1$, where P is the order of continuity.

2. $S_j(z_j) = f_j$ for $j = 0, \ldots, N-1$ and $S_{N-1}(z_N) = f_N$.

We note that in almost all cases, the conditions specified above are insufficient for constructing a unique spline of a fixed order through the data. The construction of cubic splines requires two extra conditions, which can be specified in a number of different ways. For example, requiring interpolation of endpoint derivatives is just one way, leading to the well-known clamped/complete cubic spline.

However, a spline need not always satisfy the conditions listed above. For example, the breaks of a spline need not be coincident with the data we wish to interpolate, or when a large amount of data requires fitting, a spline that fits the data in a least squares sense is desired. To develop equations that determine the spline coefficients in these cases, we distinguish two types of conditions the spline must satisfy: (I) continuity conditions and (II) fit conditions. Hereafter, we consider the construction of splines while assuming only the most general Type (I) and (II) conditions.

Suppose we aim to construct a spline $S(x)$ over the interval $[a, b]$ that satisfies the following conditions:

(I) $S(x) \in C^P([a, b])$

(II) S satisfies, either exactly or in a least-squares sense, K interpolation and/or osculatory conditions.

To begin the derivation of this construction process, let $\vec{\alpha}$ be the vector of coefficients $a_{i,k}$ of the splines ordered in the order of the splines and from lowest to highest degree. Hence, $\vec{\alpha} \in \mathbb{R}^{N(D+1)}$. Note that Type (I) constraints for derivatives up to order P at all the interior break points determine $(N-1)(P+1)$ equations of the form

$$S_j^{(m)}(z_{j+1}) = S_{j+1}^{(m)}(z_{j+1}) \quad \Longrightarrow \quad \sum_{k=m}^{D} \frac{k!}{(k-m)!} \frac{a_{j,k}}{(\Delta z_j)^m} - m! \, \frac{a_{j+1,m}}{(\Delta z_{j+1})^m} = 0 \tag{6.2.1}$$

for the coefficients, where $j = 0, \ldots, N-2$ and $m = 0, \ldots, P$. As usual, $\Delta z_j := z_{j+1} - z_j$. These equations can be represented as a linear homogeneous system of the form

$$\boldsymbol{C}\vec{\alpha} = \vec{0} \tag{6.2.2}$$

where $\boldsymbol{C}$ is an $(N-1)(P+1) \times N(D+1)$ matrix. We also note that $\boldsymbol{C}$ is a $(N-1) \times N$ block-diagonal matrix of the form

$$\boldsymbol{C} = \begin{pmatrix} \boldsymbol{C}_L & \boldsymbol{C}_R & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{C}_L & \boldsymbol{C}_R & \cdots & \mathbf{0} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{C}_L & \boldsymbol{C}_R \end{pmatrix}$$

where $\boldsymbol{C}_L$ is "upper-triangular" and $\boldsymbol{C}_R$ is diagonal, both of which are $(P+1) \times D$ matrices. More specifically, $\boldsymbol{C}_{Li,j} = 0$ if $i > j$, and $\boldsymbol{C}_{Ri,j} = 0$ if $i \neq j$. $\boldsymbol{C}$ has this form because the continuity of splines at the break points only requires the interaction of each spline with its right-neighbor, as can be seen from eq. (6.2.1).

Assuming there are K Type (II) interpolation/osculatory conditions, we can represent the equations associated with these in the form

$$\boldsymbol{T}\vec{\alpha} = \vec{g} \tag{6.2.3}$$

where $\boldsymbol{T}$ is an $K \times N(D+1)$ block-diagonal matrix and $\vec{g} \in \mathbb{R}^K$ contains all of the fit conditions. Note that the left-hand multiplication of $\boldsymbol{T}$ can be thought of as an evaluation operator for a spline with coefficients $\vec{\alpha}$. In order to obtain a square system, we must have $K = N(D+1) - (N-1)(P+1)$. If K is larger than this value, then this will lead to an over-determined system of equations.

Observe eq. (6.2.2) implies that the coefficients $\vec{\alpha}$ lie within the orthogonal complement of the column-space of $\boldsymbol{C}^T$. Using standard techniques for orthonormalizing a set of vectors (modified Gram-Schmidt, for example), we can form an orthogonal basis of this space by first orthonormalizing the columns of $\boldsymbol{C}^T$, and then further orthonormalizing randomly generated vectors to form the basis of the orthogonal complement. We therefore construct a matrix $\boldsymbol{B}$ such that

$$\boldsymbol{C}\boldsymbol{B} = \boldsymbol{0} \quad \text{and} \quad \boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}$$

These two conditions imply that the columns of $\boldsymbol{B}$ form an orthonormal basis for the coefficients of splines of degree D in $C^P([a,b])$. Hence, instead of solving eqs. (6.2.2) and (6.2.3) simultaneously, we simply look for solutions of eq. (6.2.3) of the form $\vec{\alpha} = \boldsymbol{B}\vec{w}$. Therefore, construction of a spline that satisfies the Type (I) and Type (II) constraints only involves the following steps:

1. Compute the orthogonal compliment to $\boldsymbol{C}^T$, $\boldsymbol{B}$, such that $\boldsymbol{C}\boldsymbol{B} = \boldsymbol{0}$ and $\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}$
2. Solve $\boldsymbol{A}\vec{w} = \vec{g}$ where $\boldsymbol{A} = \boldsymbol{T}\boldsymbol{B}$, and set $\vec{\alpha} = \boldsymbol{B}\vec{w}$.

In cases when $\boldsymbol{A}$ is square, the system $\boldsymbol{A}\vec{w} = \vec{g}$ can be solved using an LU factorization with partial pivoting. If $\boldsymbol{A}$ is not square, the system can solved in the least- squares sense, using a QR factorization or SVD.

When constructing splines of fixed continuity conditions and a fixed number of equispaced break points, the matrix $\boldsymbol{C}$, and thus $\boldsymbol{B}$, can be computed once and saved for *all* future uses. This follows because, with equispaced break points, we have $\Delta z_j = \Delta z_{j+1}$ for all

$j = 0, \ldots, m-1$, and thus, the continuity conditions of eq. (6.2.1) simplify to

$$\sum_{k=m}^{D} \frac{k!}{(k-m)!} a_{j,k} - m!\, a_{j+1,m} = 0$$

Therefore, there is no longer any dependence on the interval lengths. In addition, for all subsequent runs of a particular procedure, we can also save the LU decomposition (or QR factorization) of the $\boldsymbol{A}$ matrix to reduce the computational cost of the spline construction procedure.

6.2.1 Revisiting the spline operator

The convergence proofs for explicit and implicit SplineDC methods presented earlier make use of the existence of a linear bounded spline operator, $\mathcal{S} : \mathbb{R}^K \to \mathbb{S}_{D,\vec{\xi},\vec{\nu}}$, and we now explore the relationship between the boundedness of this operator and its use for spline construction. Recall that $\mathbb{R}^K$ is the space of all vectors that contain the K interpolatory/osculatory conditions, while $\mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ is the space of all splines of at most degree D, with break points described in $\vec{\xi}$, and order of continuity at the interior break points described by the components of $\vec{\nu}$. Taking the AltSpline approach to constructing splines, we now have a concrete way to describe this operator.

Fixing $\mathbb{S}_{D,\vec{\xi},\vec{\nu}}$, let $\mathcal{M} : \mathbb{R}^{\dim(\mathbb{S}_{D,\vec{\xi},\vec{\nu}})} \to \mathbb{S}_{D,\vec{\xi},\vec{\nu}}$ be a map that sends a vector of coefficients to its spline representation of the form eq. (6.1.1). Then, the spline operator can be expressed as

$$\mathcal{S} = \mathcal{M}\boldsymbol{B}\boldsymbol{A}^{-1} \tag{6.2.4}$$

Note that all of the terms on the right-hand side of eq. (6.2.4) depend on ξ; we choose to notationally drop the dependence. Hence from an AltSpline standpoint, the boundedness of the spline operator means the operator defined in eq. (6.2.4) needs to be bounded. In particular, we note that boundedness will follow if $\boldsymbol{A}^{-1}$ is bounded, e.g. is non-singular,

since it is clear that $\mathcal{M}$ and $\boldsymbol{B}$ are both bounded. In the derivation of the error bound in Chapter 3, the bound on $\mathcal{S}$ contributes to the prefactor, and since its size depends upon $\|\boldsymbol{A}^{-1}\|$, the utility of choices made in spline construction can be judged by the size of $\|\boldsymbol{A}^{-1}\|$ (or the condition number of $\boldsymbol{A}$).

6.3 Effects of Choices of Spline Construction Parameters

In this section, we will describe and examine two ways in which the extra interpolation/osculatory conditions can be omitted while keeping track of the condition number of $\boldsymbol{A}$. Throughout the following developments, let $N+1$ denote the number of data points we wish to interpolate. We use $N+1$ so the data points can be indexed by $i = 0, \ldots, N$. Furthermore, we will always assume that the highest order of continuity possible is required everywhere, i.e. if our spline is of degree D, then we impose the condition that the spline must be C^{D-1}-continuous.

6.3.1 Case study #1: Adjusting the break points

Our first approach is motivated by the following question: if the break points of a spline need not be coincident with the interpolation locations, then is it possible to decrease the number of spline panels (thus decreasing the number of degrees of freedom) so that just the interpolation conditions are sufficient for creating a square system of equations for the coefficients? This is a like the "not-a-knot" condition where extra interpolation points are inserted in some of the spline panels to add extra interpolation conditions without increasing the number of break points. However, we are not requiring *any* of the break points to be coincident with the interpolation locations.

To this end, let M be the number of spline panels where M need not be N. This implies that there are $(D+1)M$ coefficients to find, or $(D+1)M$ degrees of freedom. Furthermore, M spline panels implies $D(M-1)$ continuity conditions to specify at the interior break

points. Hence, we need

$$(D+1)M - D(M-1) = D + M$$

more conditions to close the system of equations. If we would like just the interpolation conditions to be sufficient, then we must have

$$\begin{aligned} D + M = N + 1 \quad &\Longrightarrow \quad M = (N+1) - D \\ &\Longrightarrow \quad \#\,\text{of interpolation conditions} = M + D \end{aligned}$$

When using such a procedure, care must be taken with point placement. The break points for the spline panels need to be determined with the interpolation locations in mind. For example, if all of the interpolation locations are clusterd on one side of the interval over which we want to construct a spline, then equispaced break points might not be the best choice simply because there will then be polynomial pieces of the spline without any interpolation conditions while other polynomial pieces might have too many interpolation conditions to satisfy. However, if we have equispaced interpolation locations, then having equispaced break points would guarantee (by the pigeonhole principle) that each of the spline panels will have at least one interpolation condition, which is more ideal than the previous case. Hence in our following numerical experiments, we will always use equispaced interpolation locations and spline break points.

To test the efficacy of this approach using the AltSplines method detailed above, we construct cubic and quintic spline approximations to the function $f(x) = e^{\cos(x)}$ on the interval $x \in [-5, 5]$ under panel refinement. Table 6.1 displays the pertinent results, where for both splines, we keep track of $\kappa(\boldsymbol{A})$, the 1-norm condition number of $\boldsymbol{A}$, as a way to monitor the boundedness of the spline operator, and the L_∞ norm of the error between the spline and the function itself. It's immediately obvious that this approach doesn't provide a stable method of constructing splines, but for two reasons. The condition number of

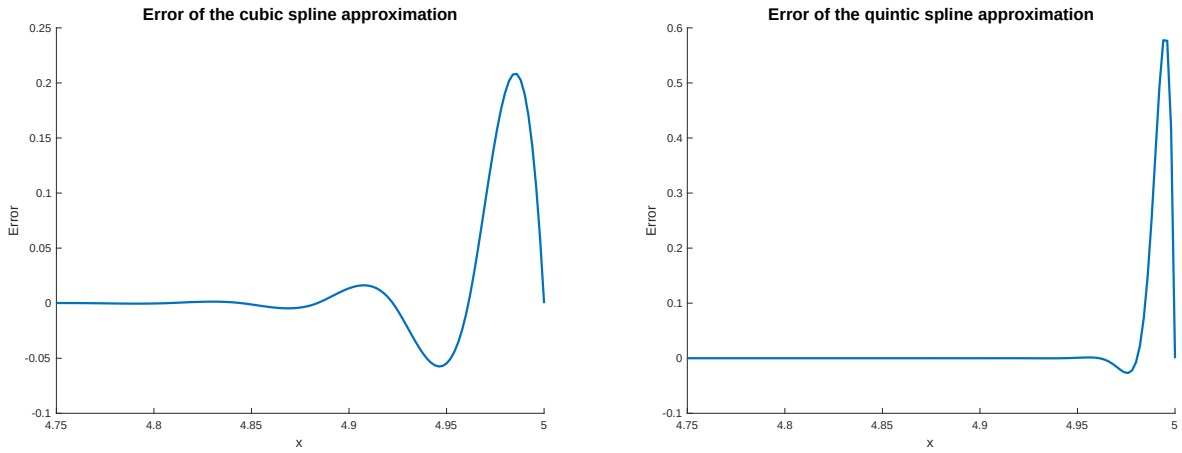


Figure 6.1: Plot of the errors associated with the cubic and quintic splines with non-coincident break points and interpolation locations constructed for the 256-panel case above of table 6.1. In both plots, note that the x-axis only shows the right-end of the interval. Similar behavior can be seen on the left-end of the interval, while errors are sufficiently small in the interior.

$\boldsymbol{A}$ does not stay bounded, implying that the matrix is becoming more and more singular. Furthermore, the errors under panel refinement don't behave well. In the cubic spline case, the errors decay like h^4 (as expected), where h is panel size, when a moderate number of spline panels is used. However, as this number gets sufficiently large, the approximating power of the spline deteriorates. This is worse in the quintic spline case, where the errors don't even decrease under panel refinement.

In fig. 6.1, we present the errors associated with the cubic and quintic splines in the case of 256 spline panels. In both plots, we choose to only show the error near the (right) end of the interval since this is exactly where the accuracy of the spline deterioriates. Similar behavior happens on the left end of the interval as well, while errors are sufficiently small away from the endpoints. What we can gather from these results is that extra care needs to be taken near the endpoints when constructing splines. In this experiment, the first and last panels of the splines we construct only contain one interpolation condition, which is the one at the appropriate endpoint. Note that in all of the well-known methods for constructing cubic splines, the first and last panels contain two Type (II) (interpolation/osculatory) conditions. The lack of a second condition is clearly the cause of the errors observed.

	Cubic spline		Quintic spline	
Panel count	$\kappa(\boldsymbol{A})$	$\|\|e(x)\|\|_\infty$	$\kappa(\boldsymbol{A})$	$\|\|e(x)\|\|_\infty$
8	2.59×10^2	4.72×10^{-2}	1.71×10^4	7.49×10^{-1}
16	6.17×10^2	4.54×10^{-3}	1.04×10^6	2.23×10^0
32	1.20×10^4	2.54×10^{-4}	6.56×10^9	1.09×10^0
64	6.98×10^6	1.47×10^{-5}	1.23×10^{17}	1.13×10^0
128	3.39×10^{11}	8.63×10^{-7}	6.79×10^{17}	3.13×10^0
256	1.26×10^{18}	4.90×10^{-1}	3.63×10^{18}	5.76×10^{-1}

Table 6.1: Results of constructing cubic and quintic spline approximations with non-coincident break points and interpolation points to $f(x) = e^{\cos(x)}$ on the interval $x \in [-5, 5]$ under panel refinement. The first column under each section displays the 1-norm condition number of the $\boldsymbol{A}$ matrix, while the second under each section displays the L_∞ norm of the error between the spline and the function. This error is calculated by sampling both at 5000 equispaced points on the entire interval.

6.3.2 Case study #2: Spline constraints at the endpoints

In our second approach, we still seek to avoid the need to specify any extra osculatory conditions to make the system of equations square. We observe that since the continuity and interpolation conditions for constructing splines can be decoupled, we can add extra differential conditions on the spline so that just the $N + 1$ interpolation conditions are sufficient to close the system of equations. By differential conditions, we mean conditions that the spline derivative(s) must satisfy, *separate* from the spline continuity constraints.

We assume that the interpolation locations are coincident with the break points for the spline. Thus, given $N+1$ interpolation locations, there will be N spline panels, and thus, we will have $(D+1)N$ coefficients to solve for, or $(D+1)N$ degrees of freedom. Furthermore, N spline panels implies $D(N-1)$ continuity conditions to specify at the interior break points. Hence, after including the $N + 1$ interpolation conditions, we still need

$$(D+1)N - D(N-1) - (N+1) = D - 1$$

	Cubic spline		Quintic spline	
Panel count	$\kappa(\boldsymbol{A})$	$\|\|e(x)\|\|_\infty$	$\kappa(\boldsymbol{A})$	$\|\|e(x)\|\|_\infty$
8	3.97×10^1	4.46×10^{-2}	1.41×10^2	2.72×10^{-1}
16	5.96×10^1	4.93×10^{-3}	2.23×10^2	3.28×10^{-3}
32	9.50×10^1	7.34×10^{-4}	4.26×10^2	1.04×10^{-4}
64	1.67×10^2	6.76×10^{-5}	7.63×10^2	3.78×10^{-7}
128	3.35×10^2	4.86×10^{-6}	1.56×10^3	5.15×10^{-9}
256	6.91×10^2	3.22×10^{-7}	3.00×10^3	7.76×10^{-11}

Table 6.2: Results of constructing cubic and quintic spline approximations with differential constraints to $f(x) = e^{\cos(x)}$ on the interval $x \in [-5, 5]$ under panel refinement. The first column under each section displays the 1-norm condition number of the $\boldsymbol{A}$ matrix, while the second under each section displays the L_∞ norm of the error between the spline and the function. This error is calculated by sampling both at 5000 equispaced points on the entire interval.

more conditions.

We require the spline to satisfy a finite-difference approximation to its derivatives at the break points at or near the endpoints. For example, at the left endpoint, the spline $S(x)$ would have to satisfy a condition of the following form:

$$S'(z_0) = \sum_{i=0}^{q} a_i S(x_i)$$

where $\{x_i\}_{i=0}^{q}$ for some fixed q is just a sequence of points in the interval. Given appropriately chosen coefficients a_i, q is determiend by the order of accuracy of the finite difference approximation used. The motivation behind this is to force the spline to behave like a single polynomial over consecutive spline panels at the ends of the interval. Thus, the value of q is chosen according to the degree of polynomial that we would like the spline to behave like, which will simply be the degree of the spline itself.

In table 6.2, we present the pretinent results of the construction of cubic and quintic splines with differential constraints for the same test case as before. We immediately notice that this approach is much better than our previous more naive approach. The L_∞-norm of

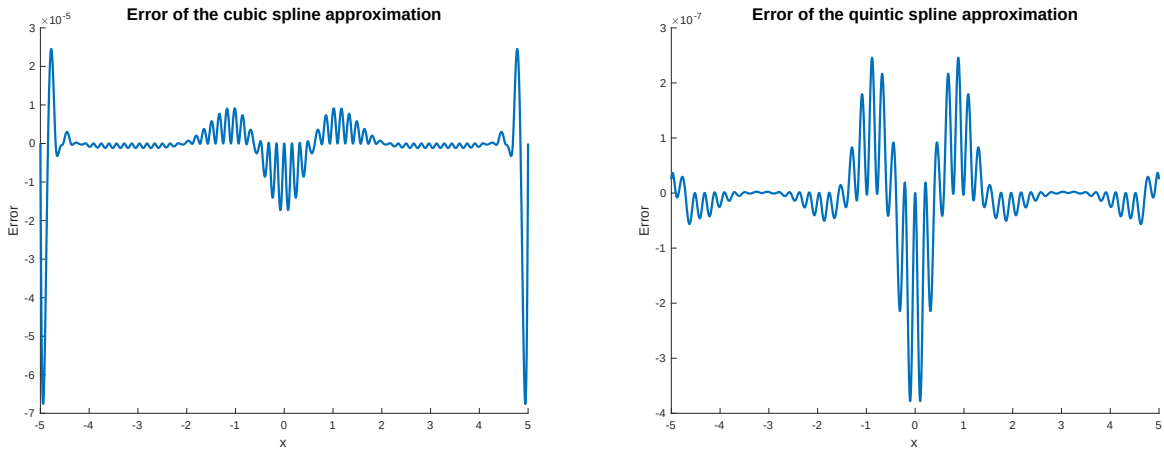


Figure 6.2: Plot of the errors associated with the cubic and quintic splines with differential constraints constructed for the 64-panel case above of table 6.1.

the errors decay like h^4 for the cubic spline case and h^6 for the quintic spline case, which is conformation that we are able to achieve the order of accuracy that these splines should attain. The condition number of $\boldsymbol{A}$ does grow under panel refinement, but very slowly.

Figure 6.2 depicts the error associated with the cubic and quintic splines with differential constraints of the 64-panel case. The errors of the 128- and 256-panel cases look similar but with errors of smaller magnitude and more oscillations. In the cubic case, even though we still observe high-frequency error components near the endpoints, the global error decreases upon panel refinement. However in the quintic case, the error is well-behaved near the endpoints even as the panel size is decreased.

This procedure for specifying differential constraints to construct the quintic splines is used in our numerical experiments of Chapter 4 for demonstrating accuracy and stability of SplineDC methods. Though we could have used the same technique for the cubic splines, we opted for clamped cubic splines because of their familiarity. Recall that in the quintic spline case, we were able to achieve the theoretical order of accuracy that we showed in Chapter 2. However, we did observe a phenomenon we didn't expect, which is that the stability region of implicit SplineDC using quintic splines with differential constraints grows in the region $\mathrm{Re}(\lambda) > 0$ as the number of substeps increases.

Chapter 7

Conclusion

In this thesis, we introduced spline deferred correction, a variant of the existing Picard integral deferred correction methods. However, as the name implies, spline deferred correction uses spline interpolation as opposed to Lagrange interpolation of the well-known spectral deferred correction methods. An immediate result of this difference in interpolation technique is the increase in freedom in picking the substeps of each composite time step, which leads to having interesting effects on the regions of absolute stability of these methods. Through numerical experiments, we demonstrated that the regions of absolute stability of explicit methods of any fixed order of accuracy grow in size as the number of substeps is increased. This is very different from what has been observed for Runge-Kutta and SpectralDC methods, where only one region of absolute stability exists per order of accuracy. Furthermore, we also showed that one can construct implicit methods of any fixed order of accuracy that is nearly A-stable and sufficiently L-stable. This property is much more favorable than the stability characteristics of the well-known backward difference methods whose $A(\alpha)$-stability deteriorates as the order of accuracy is increased. In terms of accuracy, SplineDC exhibits favorable convergence behavior, on par with that of SpectralDC.

Another consequence of the freedom in choosing substep locations for SplineDC is the ease by which one is able to implement adaptive time-stepping, and its obvious advantage over SpectralDC with respect to computational efficiency. We demonstrated that adaptive time-

stepping in the context of SplineDC yields a provisional solution with fewer total number of cumulative substeps. This amounts to SplineDC methods using many fewer function evalutions than SpectralDC methods in order to achieve a user-specified error tolerance.

Finally, we presented AltSplines, an alternate way of spline construction that leverages the decoupling of the continuity conditions of the spline from the interpolation/osculatory conditions. This results in not only a simpler method of spline construction, but allows one to easily incorporate various different conditions on the spline. Furthermore, it turns out that this framework allows one to monitor the efficacy of the decisions made in specifying conditions that close the system of equations for the spline coefficients. This allows one to determine the utility of the spline separate from its order of accuracy. In the context of SplineDC, we discovered a type of spline boundary condition that leads to implicit SplineDC methods with regions of absolute stability that uniformly grow as the number of substeps is increased.

Bibliography

[1] K. Atkinson. *An Introduction to Numerical Analysis*. Wiley, 1989.

[2] K. Bohmer and H. J. Stetter. *Defect Correction Methods, Theory and Applications*. Springer-Verlag, 1984.

[3] A. Bourlioux, A. Layton, and M. Minion. High-order multi-implicit spectral deferred correction methods for problems of reactive flow. *Journal of Computational Physics*, 189(2):651–675, 2003.

[4] K. Brenan, S. Campbell, and L. Petzold. *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1995.

[5] R. Burden and J. D. Faires. *Numerical Analysis*. Brooks/Cole, 2011.

[6] J. Butcher. *The Numerical Analysis of Ordinary Differential Equations: RungeKutta and General Linear Methods*. Wiley, 1987.

[7] M. Causley and D. Seal. On the convergence of spectral deferred correction methods. http://arxiv.org/abs/1706.06245, 2017.

[8] A. Christlieb, C. Macdonald, B. Ong, and R. Spiteri. Revisionist integral deferred correction with adaptive step-size control. *Communications in Applied Mathematics and Computational Science*, 10(1):1–25, 2015.

[9] C. de Boor. *A Practical Guide to Splines*. Springer, 2001.

[10] A. Dutt, L. Greengard, and V. Rokhlin. Spectral deferred correction methods for ordinary differential equations. *BIT Numerical Mathematics*, 40(2):241–266, 2000.

[11] A. Dutt, M. Gu, and V. Rokhlin. Fast algorithms for polynomial interpolation, integration, and differentiation. *SIAM Journal on Numerical Analysis*, 33(5):1689–1711, 1996.

[12] M. Emmett and M. Minion. Toward an efficient parallel in time method for partial differential equations. *Communications in Applied Mathematics and Computational Science*, 7(1):105–132, 2012.

[13] E. Hairer, G. Wanner, and S. Nørsett. *Solving Ordinary Differential Equations I*. Springer, 1987.

[14] A. Hansen and J. Strain. Convergence theory for spectral deferred correction, 2005.

[15] A. Hansen and J. Strain. On the order of deferred correction. *Applied Numerical Mathematics*, 61:961–973, 2011.

[16] J. Huang, J. Jia, and M. Minion. Accelerating the convergence of spectral deferred correction methods. *Journal of Computational Physics*, 214(2):633–656, 2006.

[17] C. Kennedy and M. Carpenter. Additive runge-kutta schemes for convection-diffusion-reaction equations. *Applied Numerical Mathematics*, 44:139–181, 2003.

[18] A. Layton and M. Minion. Implications of the choice of quadrature notes for picard integral deferred corrections methods for ordinary differential equations. *BIT Numerical Mathematics*, 45(2):341–373, 2005.

[19] M. Minion. Semi-implicit spectral deferred correction methods for ordinary differential equations. *BIT Numerical Mathematics*, 1(2):471–500, 2003.

[20] M. Minion. A hybrid parareal spectral deferred corrections method. *Communications in Applied Mathematics and Computational Science*, 5(2):265–301, 2010.

[21] M. Minion, R. Speck, M. Bolten, M. Emmett, and D. Ruprecht. Interweaving pfasst and parallel multigrid. *SIAM Journal on Scientific Computing*, 37(5):S244–S263, 2015.

[22] M. Minion and S. Williams. Parareal and spectral deferred corrections. *AIP Conference Proceedings*, 1048:388–391, 2008.

[23] B. Quaife and G. Biros. High-order adaptive time stepping for vesicle suspensions with viscosity contrast. *Procedia IUTAM*, 16:89–98, 2015.

[24] R. Speck, D. Ruprecht, M. Emmett, M. Minion, M. Bolton, and R. Krause. A multi-level spectral deferred correction method. *BIT Numerical Mathematics*, 55(3):843–867, 2015.

[25] T. Tang, H. Xie, and X. Yin. High-order convergence of spectral deferred correction methods on general quadrature nodes. *Journal of Scientific Computing*, 56:1–13, 2013.