©Copyright 2018
Tommaso Buvoli

Polynomial-Based Methods for Time-Integration

Tommaso Buvoli

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2018

Reading Committee:

Randall J. LeVeque, Chair

Mayya Tokman

Anne Greenbaum

Lucien Brush

Program Authorized to Offer Degree:
Applied Mathematics

University of Washington

Abstract

Polynomial-Based Methods for Time-Integration

Tommaso Buvoli

Chair of the Supervisory Committee:
Professor Randall J. LeVeque
Applied Mathematics

This thesis is divided into two parts: The first introduces a new time integration framework that is based on interpolating polynomials, and the second extends exponential integration to the spectral deferred correction framework. Both parts discuss time integration methods that can be derived without solving nonlinear order conditions.

In part I, we introduce a time-integration framework for solving systems of first-order ordinary differential equations by using interpolating polynomials. Our approach is to combine ideas from complex analysis and approximation theory to construct new integrators. This strategy allows us to trivially satisfy order conditions and easily construct a range of implicit or explicit integrators with properties such as parallelism and high-order of accuracy. The breadth of our framework is made possible by combining ideas from complex analysis, approximation theory, and general linear methods. In this work, we present several example polynomial methods including generalizations of the backward differentiation formula and Adams-Moulton methods. We compare the stability regions of these generalized methods to their classical counterparts and find that the new methods offer improved stability especially at high order. Finally, we evaluate the performance of our polynomial integrators by running a variety of numerical experiments and find that polynomial integrators offer improved stability and accuracy in comparison to classical integrators.

In part II, we introduce a new class of arbitrary-order exponential time differencing methods based on spectral deferred correction and describe a simple procedure for initializing the requisite matrix functions. We compare the stability and accuracy properties of our new exponential methods to those of an existing implicit-explicit spectral deferred correction scheme. We find that exponential integrators have larger accuracy regions and comparable stability regions. We conduct numerical experiments to compare exponential and implicit-explicit spectral deferred correction schemes against a competing fourth-order exponential Runge-Kutta scheme. We find that high-order exponential spectral deferred correction schemes are the most efficient in terms of function evaluations and overall speed when solving partial differential equations to high accuracy. Our results suggest that high-order exponential spectral deferred correction schemes are well-suited to work in conjunction with spectral spatial methods or other high-order spatial discretizations.

TABLE OF CONTENTS

Page

List of Figures vi

Glossary xii

Chapter 1: Introduction 1

Part I: A Framework For Time-Integration Based On Interpolating Polynomials 4

Chapter 2: Introduction 5

2.1 The Model Problem 7

2.2 Time-Stepping in the Complex Plane 7

2.2.1 Ordinary Differential Equations in Complex Time 7

2.2.2 A Complex Time-Stepping Scheme Based on Backward Differentiation Formula 8

2.2.3 A Time-Stepping Scheme Based on the Cauchy Integral Formula . . . 11

2.2.4 An Alternative Derivation using Polynomial Interpolation 13

2.3 Developing a New Time Integration Framework using Polynomials 15

Chapter 3: ODE Polynomials and ODE datasets 16

3.1 The ODE Dataset 16

3.2 The ODE Solution Polynomial 18

3.2.1 Special Families of ODE Solution Polynomials 19

3.2.2 Computing Coefficients 21

3.3 The ODE Derivative Polynomial 23

3.3.1 Special Families of ODE Derivative Polynomial 24

3.3.2 Computing Coefficients 25

3.4 Properties of ODE Polynomials 25

3.4.1 Truncation Error and Order of Accuracy 26

3.4.1.1 Determining Truncation Error . . . 27
3.4.1.2 Constructing Polynomials With a Given Truncation Error . 28
3.4.1.3 Determining Order of Accuracy . . . 28
3.4.1.4 Constructing Polynomials With a Given Order of Accuracy 29
3.4.2 Symmetric Polynomials and Conjugate Polynomials . . . 29
3.4.3 Spectral Accuracy for Certain ODE Polynomials . . . 31
3.5 Expanding ODE Datasets . . . 36
3.5.1 Implicitly Defined Values . . . 37
3.5.2 Interpolated Values . . . 38
3.5.2.1 Order & Truncation Error for ODE Polynomials Constructed Using Interpolated Values . . . 39
3.5.2.2 Computing Coefficients for ODE Polynomials Constructed Using Interpolated Values . . . 39
3.6 Diagrams for Geometrically Interpreting ODE Polynomials . . . 41
3.6.1 The Node Stencil . . . 41
3.6.2 The ODE Polynomial Diagram . . . 41
3.6.3 The Expansion-Point Stencil . . . 43
3.7 Example ODE Datasets and ODE Polynomials . . . 44
3.8 Alterative Polynomial Formulations . . . 47
3.8.1 Weighted Linear Approximations . . . 48

Chapter 4: Polynomial Time-Integrators . . . 50
4.1 What is a Polynomial Time Integrator? . . . 50
4.1.1 Parameters and Notation . . . 50
4.1.2 Parametrizing the Stepsize . . . 52
4.1.3 Special Families of Methods . . . 52
4.1.3.1 Propagators and Iterators . . . 52
4.1.3.2 Coarsener and Refiner Methods . . . 52
4.2 Polynomial Block Methods . . . 53
4.2.1 General Form . . . 54
4.2.1.1 Coarseners and Refiners . . . 55
4.3 Polynomial General Linear Methods . . . 56
4.3.1 General Form . . . 57

4.3.1.1 Coarseners and Refiners 58
4.3.2 Polynomial Linear Multistep Methods 58
4.3.3 Polynomial Runge-Kutta Methods 59
4.4 Adams, BDF and GBDF Integrators 61
4.5 Conjugate Inputs, Outputs and Stages 61
4.6 Order of Accuracy 62
4.7 Linear Stability 62
4.7.1 Linear Stability for Parametrized Methods 63
4.7.2 Characterizing Linear Stability Regions 63
4.8 Diagrams for Geometrically Describing Polynomial Integrators 64
4.8.1 Example Diagrams 65

Chapter 5: Constructing Polynomial Block Methods 67
5.1 Selecting Nodes 68
5.1.1 Real-Valued Nodes 69
5.1.2 Real-Symmetric Imaginary Nodes 69
5.1.2.1 Order Mappings for Real-Symmatric Imaginary Nodes . . . 70
5.2 Choosing An Active Node Index Set 71
5.3 Choosing ODE solution polynomials 82
5.3.1 Endpoint Choices For Adams Polynomials 83
5.4 A Collection of Proposed Methods 93
5.4.1 The Methods List 95

Chapter 6: Constructing Polynomial General Linear Methods 101
6.1 The Composition Product 102
6.1.1 The Composition Product For Polynomial Integrators 102
6.1.1.1 Closure Under Composition for PGLMs 103
6.2 Iterators For Improving Accuracy 104
6.3 Example Polynomial GLMs 106
6.3.1 A New Fourth-Order Polynomial GLM 106

Chapter 7: Linear Stability 109
7.0.0.1 Selecting A Set of Methods To Compare 109
7.1 Linear Stability Results for BBDF, BAM, and BAB schemes 110

7.2 Linear Stability Results for APGLM4 112

Chapter 8: Numerical Experiments 116
8.1 Test Problems 118
8.2 Results for Diagonally Implicit Polynomial Block Methods 119
8.2.1 Results for Diagonally Implicit Polynomial GLM 120
8.2.2 Results for Explicit Polynomial Block Methods 120
8.2.3 Effects of Choosing Alpha on Computational Accuracy 120

Chapter 9: Special Families of Polynomial Integrators 127
9.1 Model Problems 127
9.1.1 Linearizing Unpartitioned Systems 128
9.1.2 Semilinear Systems 128
9.2 Additive Integrators 128
9.2.1 Overview 129
9.2.2 Extending ODE Datasets and ODE Polynomials 129
9.2.2.1 Families of ODE Solution Polynomials for Partitioned Systems 133
9.2.3 Additive Polynomial Block Methods 135
9.2.4 Implicit-Explicit Polynomial Block Methods 135
9.3 Exponential Integrators 137
9.3.1 Overview 137
9.3.1.1 Variation of Constants and $\varphi(x)$ Expansions 138
9.3.2 Extending ODE datasets and ODE polynomials 138
9.3.2.1 Types of ODE Solution Polynomial 141
9.3.3 Exponential Polynomial Block Methods 141
9.3.3.1 Constructing Exponential Methods 141

Chapter 10: Conclusions 143

Part II: Exponential Spectral Deferred Correction 144

Chapter 11: Exponential Integrators Based on Spectral Deferred Correction 145
11.1 Introduction 145
11.2 Spectral Deferred Correction Methods 146

11.2.1 Preliminaries . . . 146
11.2.2 Euler-Based Spectral Deferred Correction Methods . . . 148
11.2.3 ETD Spectral Deferred Correction Methods . . . 149
11.2.4 IMEX Spectral Deferred Correction . . . 151
11.3 Stability and Accuracy . . . 153
11.4 Calculating $W_i^{i+1}(\phi^k)$ . . . 155
11.4.1 Proposed Algorithm . . . 158
11.4.2 φ Functions . . . 160
11.4.2.1 Taylor/Padé Scaling and Squaring Algorithm . . . 160
11.4.2.2 Contour Integration Algorithm . . . 161
11.5 Numerical Experiments . . . 161
11.5.1 Discussion . . . 167
11.6 Conclusion . . . 168

Part III: Conclusions and Future Work . . . 169

Chapter 12: Summary, Conclusions, and Future Work . . . 170
12.1 Summary . . . 170
12.2 Conclusions . . . 171
12.3 Future Work . . . 171

Bibliography . . . 173

LIST OF FIGURES

Figure Number Page

2.1 Stencils for the first three BDF methods. Each stencil shows the real t-axis along with the temporal nodes of the data values used to form the polynomials $H(t)$. A temporal node that corresponds to an input value is marked with a black square, while a temporal node for an output derivative is marked with a circle. 9

2.2 Stencils for the second-order and third-order BBDF methods. Each stencil shows the imaginary t-plane along with the temporal nodes of the data values used to form the polynomials $H^{[j]}(t)$, $j = 1, \ldots, 1$. As before, a temporal node that corresponds to an input value is marked with a black square, while a temporal node for an output derivative is marked with a circle. 10

2.3 Linear stability regions for BBDF and BDF methods of orders 2 through eight. The stability contour for the 8th order BBDF method is labeled in red in both (a) and (b). The stability region for all methods is exterior to each curve. . . 11

2.4 The complex t-plane containing the input times $t_j^{[n]}$ (black circles) and the output times $t_j^{[n+1]}$ (white circles) for $q = 4$. We also plot the time-step centers t_n, $t_{n+1} = t_n + h$ (grey circles) and two circles of radius r centered around t_n and t_{n+1}. The stepsize h has been parametrized as $h = r\alpha$, where α denotes number of radii r per timestep. 13

2.5 Linear stability regions for Adams-Bashforth methods and for the method (2.10). The red half-circle in both subfigures marks the perimeter of the region $\left\{|z| \leq \frac{1}{e}\right\} \cap \{\text{Re}(z) \leq 0\}$ and is the conjectured limiting stability domain for method (2.10) as $q \to \infty$. 14

3.1 Diagrams for three ODE solution polynomials constructed from the ODE dataset $D(r, s) = \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{4}$ with temporal nodes $\tau_1 = i$, $\tau_2 = -i$, $\tau_3 = i + 1$, $\tau_4 = -i + 1$. 30

3.2 Example node stencils for the ODE polynomial $p(\tau; b) = 0$ constructed from **(a)** an ODE dataset with the three complex temporal nodes $\{\tau_j\} = \{-i, 0, i\}$ and one interpolated value at $\tau = 1/2$, and **(b)** an ODE dataset with the three real temporal nodes $\{\tau_j\} = \{-1, 0, 1\}$ and one interpolated value at $\tau = 3/2$. 42

3.3 A collection of four ODE solution polynomials constructed in examples 3.7.2, 3.7.3, and 3.7.4 from an ODE dataset with nodes $\tau_1 = -1$, $\tau_2 = 0$, and $\tau_3 = 1$. For each polynomial, the corresponding diagram, active value set, and active node stencil are shown. For the BDF polynomial with one interpolated derivative, the diagram and active value set of the ODE derivative polynomial $\dot{p}(\tau, b)$ used to compute the interpolated derivative are also shown. The ordering reflects the ordering of the examples in this section. 48

3.4 A collection of three ODE solution polynomials constructed in example 3.7.5 from an ODE dataset with nodes $\tau_1 = -i$, $\tau_2 = 0$, and $\tau_3 = i$. For each polynomial, the corresponding diagram, active value set, and active node stencil are shown. The ordering reflects the ordering in example (3.7.5). 49

4.1 We illustrate the effects of varying the node radius r while keeping the stepsize h constant. We show the complex t-plane containing the input times $t_j^{[n]}$ (black circles) and the output times $t_j^{[n+1]}$ (white circles) corresponding to the four roots of unity. 51

4.2 We distinguish between iterator and propagator methods. We show input times $t_j^{[n]}$ and output times $t_j^{[n+1]}$ on the real t line when z_j are three equispaced points. Iterators can be useful for correcting or updating an approximate solution, while propagators are traditional time-stepping schemes. 53

4.3 Refiners and coarseners are not classical time-stepping schemes since they cannot be applied in succession. However, they can be used to compute additional outputs times or to modify an input in preparation for a second method. . . 53

4.4 60

4.5 Four illustrations depicting different properties of stability regions. **(a)** The exact solution to the Dahlquist test problem remains bounded for all time if $\mathrm{Re}(z) \leq 0$. **(b)** $A(\theta)$ stability requires that a stability region contains a sector of the the left half plane with angle 2θ, centered along the real line. **(c)** The negative stability interval measures the width of the a stability region along the negative real line and relative to the origin. **(d)** The imaginary stability interval measures the width of the a stability region along the positive imaginary line and relative to the origin. 64

4.6 Example node diagrams for linear multistep methods and polynomial block methods with $q = 2$. For both types of methods, we show diagrams for a diagonally-implicit method and an explicit method. The node diagram for the diagonally-implicit LMM describes both the 3rd order Adams-Moulton method and the 2nd order backwards differentiation formula while the explicit node diagram describes 2nd order Adams-Bashforth. 66

4.7 Polynomial diagrams for the ODE polynomial used by the third order BDF method, fourth-order Adams-Moulton, and third-order Adams-Bashforth. . . 66

5.1 Real-symmetric imaginary node orderings shown for the node set $\{z_j\}_{j=1}^{4} = \{i,\ i/3,\ i/3,\ i\}$ and $\{z_j\}_{j=1}^{5} = \{-i,\ -i/2,\ 0,\ i/2,\ i\}$. 70

5.2 Empty active node stencils for an ODE solution polynomial for a block method with (**a**) nodes $\{z_j\}_{j=1}^{5} = \{-i,\ -i/2,\ 0,\ i/2,\ i\}$ and $\alpha > 0$, (**b**) nodes $\{z_j\}_{j=1}^{4} = \{i,\ i/3,\ i/3,\ i\}$ and $\alpha > 0$, (**c**) nodes $\{z_j\}_{j=1}^{3} = \{-1, 0, 1\}$ with $\alpha > 2$. For clarity, all inactive nodes have been enlarged, and inputs are labeled with the letter I while outputs are labeled with the letter O. 76

5.3 Node stencils for diagonally implicit block methods with three real equispaced points and $\alpha > 0$. 77

5.4 Node stencils for diagonally implicit block methods with three real equispaced points and $\alpha > 0$. 77

5.5 Node stencils for explicit block methods with four imaginary equispaced points and $\alpha > 0$. 78

5.6 Node stencils for diagonally implicit block methods with four imaginary equispaced points and $\alpha > 0$. 79

5.7 Node stencils for explicit block methods with five imaginary equispaced points and $\alpha > 0$. 80

5.8 Node stencils for diagonally implicit block methods with five imaginary equispaced points and $\alpha > 0$. 81

5.9 (**Fixed Input & Sliding Input, Real Nodes**) Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with three real equispaced nodes. 87

5.10 **(Sliding Output, Real Nodes)** Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with three real equispaced nodes. . 88

5.11 (**Fixed Input, Real-Symmetric Imaginary Nodes**) Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with four and five imaginary equispaced nodes. Conjugate expansion points are shown in red circles instead of blue circles. 89

5.12 **(Sliding Input, Real-Symmetric Imaginary Nodes)** Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with with four and five imaginary equispaced nodes. Conjugate expansion points are shown in red circles instead of blue circles. 90

5.13 **(Sliding Output, Real-Symmetric Imaginary Nodes)** Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with with four and five imaginary equispaced nodes. Conjugate expansion points are shown in red circles instead of blue circles. 91

5.14 ODE polynomial diagrams for $L_y^{[j]}$ constructed using (5.17) and either four and five imaginary equispaced nodes. This non-compact representation of can be used with all endpoint listed in Subsection 5.3.1 92

5.15 ODE polynomial diagrams for $L_y^{[j]}$ constructed using (5.17) and three real equispaced nodes. This non-compact representation of can be used with all endpoint listed in Subsection 5.3.1 92

5.16 A graph displaying the parameter choices listed in this chapter for constructing a polynomial block method. Each blue vertex leads to ten possible method families; in total there 180 families. Instead of trees representing order conditions, we now find ourselves with trees of methods. 94

7.1 Stability regions for BDF methods of order two through six and block BDF (BBDF) methods of orders two through eight with rotated equispaced nodes. The stability boundary for eighth order BBDF with $\alpha = 1/2$ and $\alpha = 1/4$ are respectively labeled in red and green. Decreasing the extrapolation parameter α, consistently improves $A(\theta)$ stability for BBDF schemes. 113

7.2 Magnification of Figure 7.1 showing stability regions near the imaginary axis. 113

7.3 Stability regions for Adams-Moulton (AM) and block Adams-Moulton (BAM) of orders three through nine. The stability boundary for third-order Adams-Moulton appears as a blue contour in all figures. Block methods have significantly larger stability regions but do not include any part of the imaginary axis. Smaller α leads to larger and more circular stability regions. 114

7.4 Left: stability regions for APGML4 for $\alpha = 1$, 2, and 4. Right: magnified stability regions around imaginary axis. The imaginary axis is never fully included in the stability region for any of these α values. 114

7.5 Stability regions for Adams-Bashforth (AB) and block Adams-Bashforth (BAB) schemes of orders two through eight. The stability regions for BAB methods monotonically decrease and appear to converge to a half circle, while the stability regions for AB converge to zero. For both $\alpha = 1$ and $\alpha = 1/2$ BAB methods of orders 4, 5, and 8 are stable along a non-empty interval of the imaginary axis. Changing α has significantly less effect on stability regions then was the case for implicit methods. 115

8.1 Error vs stepsize and error vs running time diagrams for the Viscous Burgers equation. We present results for BDF and BBDF in the top two plots, and results for BAM and Adams-Moulton (AM) in the bottom two plots. All BAM and BBDF methods are run using $\alpha = 1/2$. The seven thinly-dashed grey lines on the error vs stepsize plots show the slopes expected in these log-log plots for 2nd through 8th order convergence. 121

8.2 Error vs stepsize and error vs running time diagrams for the 2D ADR equation. We present results for BDF and BBDF in the top two plots, and results for BAM and Adams-Moulton (AM) in the bottom two plots. All BAM and BBDF methods are run using $\alpha = 1/2$. The seven thinly-dashed grey lines on the error vs stepsize plots show the slopes expected in these log-log plots for 2nd through 8th order convergence. 122

8.3 Error vs stepsize and error vs running time diagrams for the Burgers and 2D ADR equation. We present results for APGLM4 and the block methods BBDF2, BAM3 all with $\alpha = 4$. We also include second order BDF and third order Adams-Moulton in our plots. The three thinly-dashed grey lines on the error vs stepsize plots show the slopes expected in these log-log plots for 2nd, 3rd and 4th order convergence. 123

8.4 Error vs stepsize and error vs running time diagrams for the Van der Pol equation. We present results for AB and BAB. All BAB methods are run using $\alpha = 1$. The thin dashed black lines on Error vs Stepsize plots correspond to order $2, 3, \ldots, 8$ accuracy. The seven thinly-dashed grey lines on the error vs stepsize plots show the slopes expected in these log-log plots for 2nd through 8th order convergence. 124

8.5 Error vs stepsize and error vs running time diagrams for the viscous Burgers equation. We present results for BDF, BBDF, AM, and BAM methods. For BBDF and BAM methods we test methods with α values of 1/2, 1/4, and 1/8. The three thinly-dashed grey lines on the error vs stepsize plots correspond to 4th, 6th and 8th order convergence. 126

11.1 Stability regions for 8th order and 16th order methods with Chebyshev quadrature nodes. Colored contours correspond to different r values as described in the legend. We plot an additional black contour for the ETDSDC^{15}_{16} method on the dispersive model problem to show that stability regions eventually grow for sufficiently large imaginary r. For large $|r|$, increasing the order of the ETD and IMEX methods does not lead to significantly larger stability regions. 156

11.2 Accuracy regions corresponding to $\epsilon = 1 \times 10^{-8}$ for 8th order and 16th order methods with Chebyshev quadrature nodes. Colored contours correspond to different r values as described in legend. We choose R_0 in each figure so that the red contour marks a near vanishing accuracy region around $z = 0$. As expected, 16th order methods possess larger accuracy regions for a wider range of r than 8th order methods. 157

11.3 Performance results for the Kuramoto-Sivianshi and Nikolaevskiy equations. Gray dashed lines of increasing steepness in the accuracy vs stepsize plots correspond to $O(h^4)$, $O(h^8)$ and $O(h^{16})$, respectively. IMEXSDC schemes experience significant order reduction on both problems. 165

11.4 Performance results for the Quasigeostrophic and Korteweg-de Vries equations. Dashed lines of increasing steepness in the accuracy vs stepsize plots correspond to $O(h^4)$, $O(h^8)$ $O(h^{16})$ and $O(h^{32})$, respectively. Notice that high-order IMEXSDC schemes are unstable on the KDV equation. Order reduction does not occur for any method on the quasigeostrophic equation, but affects both IMEXSDC and ETDSDC schemes on the KDV equation. 166

GLOSSARY

Through this document we will consistently use a set of fixed variables to denote certain important quantities. We provide a list of these variables here.

Variables used for ODE polynomials

$p(\tau; b)$ an ODE polynomial

$a_j(b)$ approximate derivatives of an ODE polynomial

b expansion point for an ODE polynomial

g degree of an ODE polynomial

Variables used for ODE datasets and interpolated value set

$D(r, s)$ an ODE datasets

w size of an ODE dataset

l number of interpolated values

s dataset translation factor

Variables used for polynomial time-integrators

n timestep index

q number of inputs and outputs

m number of outputs for a coarsener or refiner method

r node radius

α extrapolation factor

$p_j(\tau; b_j)$ The ODE solution polynomial for computing output or stage j

$y_j^{[n]}$ input

$f_j^{[n]}$ input derivatives

Y_j stage

F_j stage derivatives

$I(j)$ active input index set for the ODE solution polynomial for jth output

$O(j)$ active output index set for the ODE solution polynomial for jth output

Miscellanious

$y(t)$ solution to the initial value problem

$F(t, y(t))$ the initial value problem derivative

t global time for initial value problem

τ local time for initial value problem

j, k index variables

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to all the people who made this thesis possible. First I would like to thank my advisor Randall LeVeque for his support, for his guidance, and for the many useful discussions we had over the entire course of this work. I will always appreciate the freedom that Randy gave me to pursue my interest in time-integration and approximation theory. Next, I would like to thank Mayya Tokman whom very kindly hosted me at UC Merced for one year, taught me about time-integrators, and helped me improve both the clarity and organization of my work. I would also like to thank John Butcher for kindly hosting me in New Zealand and for his illuminating discussions about general linear methods and B-Series. Finally, I would also like to thank Lucien Brush and Anne Greenbaum for their feedback during the course of this work, and for taking the time to serve on my committee.

This work was supported in part by funding from the Applied Mathematics Department at the University of Washington and the National Science Foundation, Computational Mathematics Program, under DMS-1115978, DMS-1216732, and the EAPSI grant #614232.

Chapter 1

INTRODUCTION

Partial differential equations provide accurate mathematical descriptions of a wide variety of physical phenomena. In the last century, the study of partial differential equations has grown into an interdisciplinary venture spanning across the fields of mathematics, computation, physics, biology, economics and more. As the scale and complexity of engineering problems increases, so too does the need for efficient computational methods. Such developments enable scientists and engineers to simulate costly experiments and obtain results that are difficult to measure in the laboratory setting.

This work focuses on developing novel time-integration methods for efficiently solving stiff initial value problems arising from the discretization of time-dependent partial differential equations. The broad aim of the work is to simplify the construction of new high-order methods and new parallel methods that can leverage current advances in computational hardware.

The central theme of this thesis is the application of polynomial approximations to solve systems of first order ordinary differential equations. We explore this idea in two separate contexts; in part I we present a new methodology for constructing time-integrators using interpolating polynomials, and in part II we discuss exponential integration for spectral deferred correction methods.

The first part of this work is heavily influenced by approximation theory, complex analysis, and a desire to apply techniques from these domains to derive new time-integrators. We discuss ordinary differential equations in the complex plane, present a class of polynomial approximations for the initial value problem, and introduce a new time-integration framework that encompasses a range of polynomial-based methods. Through numerical experiments, we show that these new integrators offer improved stability and performance in comparison to classical polynomial based methods. Several sections in part I have been adapted from the manuscript [20].

In the second part of the this work, we extend a previously introduced time-integration framework known as spectral deferred correction (SDC) [28] to allow for exponential integration. This simplicity and extendibility of the SDC framework allows us to construct extremely high-order exponential integrators with good stability properties and excellent computational performance. All the content from this part is taken directly from [19].

1. Applying Polynomials to the Time Domain

Polynomials are widely used in approximation theory to estimate functions, derivatives, and

integrals [95]. They are also frequently used for solving partial differential equations and form the basis of finite difference methods, spectral methods, and linear multistep methods. In comparison to other discretizations, polynomial-based methods have a clear advantage: they are simple to derive and implement at both low and high order. Though polynomial methods have proven extremely effective in spatial discritizations, linear multistep methods suffer from unfavorable stability properties and complications with variable time-stepping. More recent developments in time integration have focused on Runge-Kutta methods and general linear methods. Unfortunately, this move away from polynomials introduces large numbers of nonlinear order conditions that make it more challenging to derive high order schemes.

The approaches presented in this thesis eliminate the complexity of order conditions, enabling simple construction of methods with a specific architecture (parallel or serial), degree of implicitness (explicit, diagonally implicit, fully implicit) and desired order of accuracy. Moreover, by extending the ODE solution into complex time, we are able to ameliorate many of the shortcomings of existing real-valued polynomial-based methods. In short, our work attempts to bring together the flexibility of general linear methods [16], the derivational simplicity of linear multistep methods, and the iterative correction capabilities of spectral deferred correction schemes [28], in order to derive a powerful framework for constructing a wide range of polynomial-based schemes.

2. Time-Integration of Large, Stiff Systems

The second aim of this work is to advance the development of new high-order integrators for solving the large, stiff systems arising from the discretization of time-dependant partial differential equations. We approach this topic in three separate ways.

1. *Implicit Methods*

   Our polynomial framework includes a range of implicit methods including those possessing parallelism. We will discuss their utility and compare their performance on several stiff partial differential equations.

2. *Implicit-Explicit and Exponential Polynomial Methods*

   The concepts presented in this thesis for a basis for developing new high-order exponential and implicit-explicit integrators. The simplicity of the polynomial approach extends even within the context of exponential integration and implicit-explicit integration. This allows one to derive specialized integrators that satisfy the stability and accuracy requirements of a specific application problem, and appropriately leverage the computer architecture used for the computation. In contrast, exponential or implicit-explicit Runge-Kutta methods do not allow for this flexibility and require careful derivation due to a large number of order-conditions.

3. *Exponential Integrators based on Spectral Deferred Correction*

   Spectral deferred correction schemes are a relatively new class of time-integrators that iteratively improve a candidate solution by repeatedly solving an integral equation that describes the error. The spectral deferred correction framework provides improved stability in comparison to linear multistep methods and simplified construction of high-order schemes in comparison to Runge-Kutta methods. Our push to develop exponential SDC integrators enables the construction of new integrators for stiff systems with orders of accuracy as high as thirty. These new methods outperform existing Runge-Kutta exponential integrators when solving partial differential equations to high accuracy, and are significantly less prone to order reduction than competing implicit-explicit SDC schemes.

Part I

A FRAMEWORK FOR TIME-INTEGRATION BASED ON INTERPOLATING POLYNOMIALS

Chapter 2

INTRODUCTION

Interpolating polynomials are widely used in approximation theory to estimate functions, derivatives, and integrals. Choosing a stable set of interpolation nodes enables the construction of convergent approximations of high degree with applicability to many computational domains [95, 12]. Commonly used node sets include the real-valued Chebyshev points and the roots of the Legendre polynomials; however, it is also possible to use complex-valued nodes. For example, by sampling a function at the roots of unity, one can overcome the ill-conditioning inherent in classical finite difference formulas for approximating high-order derivatives [31, 70].

Interpolating polynomials have proven particularly effective for solving partial differential equations and have led to the development of finite difference methods [90] and spectral methods [34, 93]. In each case a polynomial approximation, either local or global, is formed using spatial information and then differentiated. Interpolating polynomials have also been applied in the time dimension to derive classical linear multistep methods such as backward differentiation formula, Adams-Moulton methods, and Adams-Bashforth methods. Unfortunately many polynomial-based linear multistep methods suffer from unfavorable stability properties and complications with variable time-stepping.

More recent developments in time integration have focused on Runge-Kutta [42, 41] and general linear methods [16]. Instead of using polynomials, one proposes a method ansatz in terms of unknown coefficients and then solves a resulting set of nonlinear order conditions. Though the theory for obtaining order conditions is well-understood, the large number of conditions makes it challenging to derive high-order methods. To illustrate this, we list the number of order conditions for Runge-Kutta methods of orders one through ten.

Order:	1	2	3	4	5	6	7	8	9	10
Number of conditions:	1	2	4	8	17	37	85	200	486	1205

The number of total order conditions is larger for general linear methods in which one must additionally specify and derive a starting method. Moreover, order conditions become even more numerous for specialized integrators such as implicit-explicit schemes [79], exponential integrators [6, 69], Rosenbrock integrators [77], and additive integrators [56, 57].

In this work we present a general approach for applying interpolating polynomials to solve initial value problems. Our broad aim is to simplify the derivation of high-order methods with good stability properties. Our polynomial-based framework for time integration has four key components:

1. *A New Method Formulation*: We describe methods in terms of the interpolating polynomials used to compute stage and output values. This formulation is complementary to the traditional coefficient representation.

2. *Complex Time Integration*: Our framework allows for methods with complex coefficients and methods with inputs and stages at complex times. This generalization allows us to construct schemes that more broadly incorporate ideas from approximation theory and stable numerical differentiation. To the best of our knowledge, this is the first attempt to broadly couple complex time with classical integrator design. However, previous work explores Taylor methods in the complex plane [25], a Taylor/Padé-based ODE solver for the Painlevé equations [35], and complex splitting schemes [44].

3. *A Wide Variety of Methods*: Our framework includes multivalue and one-step methods. We incorporate the flexibility of general linear methods [16] and the iterative correction capabilities of spectral deferred correction schemes [28].

4. *A New Parameter*: In addition to stepsize, polynomial integrators have an additional parameter that scales the input nodes. This new parameter allows methods to incorporate polynomial approximations that adapt to the local smoothness of the solution, and is essential for obtaining methods with improved stability regions.

This work is broadly organized as follows. Chapters 2 to 4 motivate and introduce our polynomial framework and define terms. Chapters 5 and 6 are dedicated to method construction and the final three chapters present linear stability comparisons, numerical experiments, future work, and conclusions. The content of each individual chapter is as follows. In this chapter, we discuss the initial value problem in the complex plane and derive two simple polynomial-based integrators. In Chapter 3, we introduce a generalized interpolating polynomial for approximating initial value problems, discuss its properties, and describe diagrams for visualizing these polynomials. In Chapter 4, we introduce our new framework for time integration and discuss block methods, general linear methods, linear multistep methods, and Runge-Kutta methods. In Chapter 5, we present a detailed discussion on constructing polynomial block methods and derive a wide range of schemes. In Chapter 6, we introduce method composition and discuss the construction of an example polynomial general linear method. In Chapter 7 we compare linear stability properties for several polynomial schemes. Next, in Chapter 8 we present several numerical experiments for these generalized methods and compare them to their classical counterparts. In Chapter 9 we present additive and exponential integrators and finally, in Chapter 10, we present some concluding remarks and discuss future work.

2.1 The Model Problem

We are interested in developing time-integrators for solving systems of ordinary differential equations of the form

$$\begin{aligned} \frac{dy_j}{dt} &= F_j(t, y_1, \ldots, y_M) \\ y_j(t_n) &= y_{j,n} < \infty \end{aligned} \qquad j = 1, \ldots, M. \tag{2.1}$$

However, to avoid notation with Kronecker products in our derivations, we take $M = 1$ and introduce our framework using the scalar differential equation

$$y'(t) = F(t, y(t)), \qquad y(t_n) = y_n. \tag{2.2}$$

2.2 Time-Stepping in the Complex Plane

In this section we briefly discuss how initial-value problems posed on a real time interval can be extended into the complex time plane. We then use this observation to construct two time-stepping methods that compute solutions at complex time points. We first derive this method using the Cauchy integral formula and then present an alternative derivation using Lagrange interpolating polynomials in the complex plane. Interpreting this method from a polynomial perspective provides a clear direction for further generalization and motivates the construction of a wide class of time-integrators based on interpolating polynomials.

2.2.1 Ordinary Differential Equations in Complex Time

It is normally assumed that the initial value problem (2.2) is valid only for real time. However, if the solution $y(t)$ is locally analytic around the initial condition, then we may extend the initial value problem to complex time via analytic continuation. Local analyticity is guaranteed by the following theorem.

Theorem 2.2.1 (Cauchy-Kowalevski). *Consider the system of ordinary differential equations (2.1). If each $F_j(t, y_1, \ldots, y_M)$, $j = 1, \ldots M$, is an analytic function of each of its arguments in a connected open region of the complex plane containing $t = t_n$, then (2.1) has a unique, analytic solution in a neighborhood of t_n [1, Section 3.7].*

Many common ordinary differential equations and spatial discretizations of partial differential equations satisfy the conditions required by the Cauchy-Kowalevski theorem. If we restrict ourselves to these problems, it becomes possible to explore time integration methods in the complex time plane. By simple application of Taylor series we can locally approximate the solution in complex time via

$$y(t_n + z) = \sum_{\nu=0}^{q} \frac{y^{(\nu)}(t_n) z^{\nu}}{\nu!} + \mathcal{O}(z^{q+1}). \tag{2.3}$$

Unfortunately, despite the simplicity of this Taylor series method it is generally impractical to differentiate the right-hand-side explicitly. Moreover, this scheme is explicit and extremely small time-steps must be taken to maintain stability when solving stiff initial value problems.

In the following subsections we present two new time integrators that approximate the Taylor series (2.3) using polynomial approximations. The first is an implicit method that generalizes the classical BDF scheme by moving into the complex plane. The second is an explicit method that approximates high-order derivatives using a discrete Cauchy integral formula.

2.2.2 A Complex Time-Stepping Scheme Based on Backward Differentiation Formula

One of the simplest ways to approximate the truncated Taylor series (2.3) is to replace it with an interpolating polynomial of degree q that approximates the solution. We can use this strategy to derive the backward differentiation formula (BDF) methods. For example, suppose that we are provided with the solution and derivative values

$$y_{n-1} = y(t_n - h), \qquad y_n = y(t_n), \qquad \text{and} \qquad f_{n+1} = f(t_n + h, y_{n+1}).$$

We can use this data to construct a polynomial $H(t)$ that matches the two solution values at t_n, $t_n - h$ and the derivative at $t_n + h$. If we define y_{n+1} as

$$y_{n+1} = H(t_n + h),$$

then we obtain the second-order backward differentiation formula

$$y_{n+1} = -\frac{1}{3}y_{n-1} + \frac{4}{3}y_n + \frac{2h}{3}f_{n+1}.$$

More generally, we can derive the formula for the qth order backward differentiation formula by taking $H(t)$ to be an interpolating polynomial that matches the following solution and derivative values:

inputs	$y_{n-q}, \ \ldots, \ y_n$
output derivative	f_{n+1}.

Instead of writing the corresponding formulas, we can graphically illustrate each BDF method by using a stencil. Each stencil shows the temporal locations of the solution and derivative values used to construct the polynomials for computing the method's outputs. In Figure 2.1 we show the stencils for the BDF methods of orders one through three. These stencils are generalizations of the stencils used for finite difference formulas for approximating derivatives.

The second-order BDF method is A-stable and can be used to solve both dispersive and dissipative partial differential equations. However, the stability regions for higher-order BDF methods do not encompass the imaginary axis, and become increasingly restrictive as the order of the methods increases. Morover, beyond order six, there are no stable BDF methods.

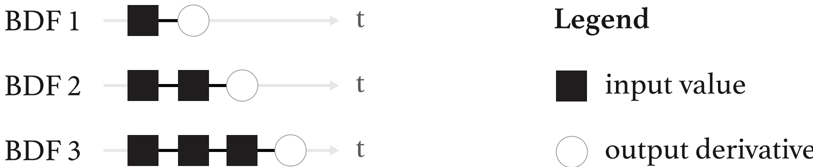
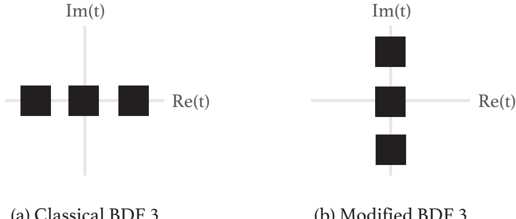


Figure 2.1: Stencils for the first three BDF methods. Each stencil shows the real t-axis along with the temporal nodes of the data values used to form the polynomials $H(t)$. A temporal node that corresponds to an input value is marked with a black square, while a temporal node for an output derivative is marked with a circle.

We can hope to alleviate these shortcomings by generalizing the BDF method to allow for solution values at complex times. We begin by rotating our input times into the complex plane so that at each timestep we are provided with q solution values with temporal nodes that run parallel to the imaginary axis.



(a) Classical BDF 3 (b) Modified BDF 3

During the nth timestep the modified method requires the q inputs

$$y_j^{[n]} \approx y(t_n + h\tau_j) \qquad \text{for} \qquad \tau_j = -i + \frac{2i(j-1)}{q-1} \qquad \text{and} \qquad j = 1, \ldots, q.$$

The formula for computing the outputs is given by

$$y_j^{[n+1]} = H^{[j]}(t_n + h), \qquad j = 1, \ldots q \tag{2.4}$$

where $H^{[j]}(t) \approx y(t)$ is an interpolating polynomial matching the following solution and derivative values:

inputs	$y_1^{[n]}, \ \ldots, \ y_q^{[n]}$
output derivative	$f_j^{[n+1]}$.

We will call this new implicit method block BDF (BBDF) since it advances a "block" of solution values at each timestep. The stencils for BBDF2 and BBDF3 are shown in Figure 2.2 and the formula are listed in Table 2.1.

Figure 2.2: Stencils for the second-order and third-order BBDF methods. Each stencil shows the imaginary t-plane along with the temporal nodes of the data values used to form the polynomials $H^{[j]}(t)$, $j = 1, \ldots, 1$. As before, a temporal node that corresponds to an input value is marked with a black square, while a temporal node for an output derivative is marked with a circle.

BBDF2

$$\begin{aligned} y_1^{[n+1]} &= \left(\tfrac{4}{5} + \tfrac{2i}{5}\right) y_1^{[n]} + \left(\tfrac{1}{5} - \tfrac{2i}{5}\right) y_2^{[n]} + \left(\tfrac{3}{5} - \tfrac{i}{5}\right) h f_1^{[n+1]} \\ y_2^{[n+1]} &= \left(\tfrac{1}{5} + \tfrac{2i}{5}\right) y_1^{[n]} + \left(\tfrac{4}{5} - \tfrac{2i}{5}\right) y_2^{[n]} + \left(\tfrac{3}{5} + \tfrac{i}{5}\right) h f_2^{[n+1]} \end{aligned} \tag{2.5}$$

BBDF3

$$\begin{aligned} y_1^{[n+1]} &= \left(\tfrac{22}{37} + \tfrac{21i}{37}\right) y_1^{[n]} + \left(\tfrac{21}{37} - \tfrac{22i}{37}\right) y_2^{[n]} + \left(-\tfrac{6}{37} + \tfrac{i}{37}\right) y_3^{[n]} + \left(\tfrac{17}{37} - \tfrac{9i}{37}\right) h f_1^{[n+1]} \\ y_2^{[n+1]} &= \left(\tfrac{i}{4}\right) y_1^{[n]} + (1)\, y_2^{[n]} + \left(-\tfrac{i}{4}\right) y_3^{[n]} + \left(\tfrac{1}{2}\right) h f_2^{[n+1]} \\ y_3^{[n+1]} &= \left(-\tfrac{6}{37} - \tfrac{i}{37}\right) y_1^{[n]} + \left(\tfrac{21}{37} + \tfrac{22i}{37}\right) y_2^{[n]} + \left(\tfrac{22}{37} - \tfrac{21i}{37}\right) y_3^{[n]} + \left(\tfrac{17}{37} + \tfrac{9i}{37}\right) h f_3^{[n+1]} \end{aligned} \tag{2.6}$$

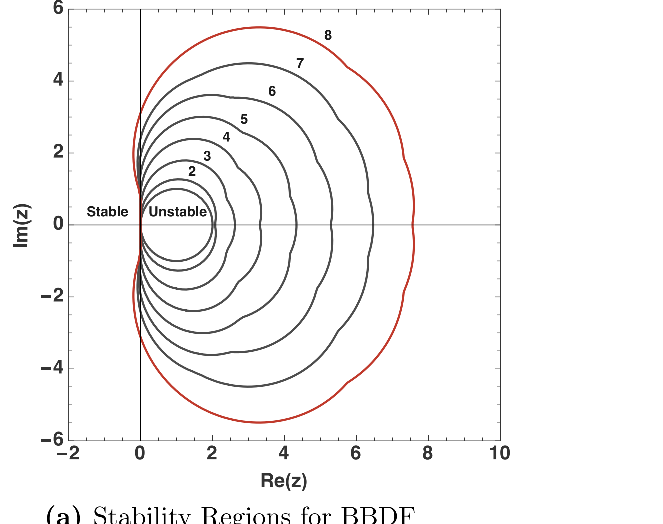
Table 2.1: Formula for BBDF2 and BBDF3 methods

Linear stability for time-integrators is determined using the Dahlquist test problem

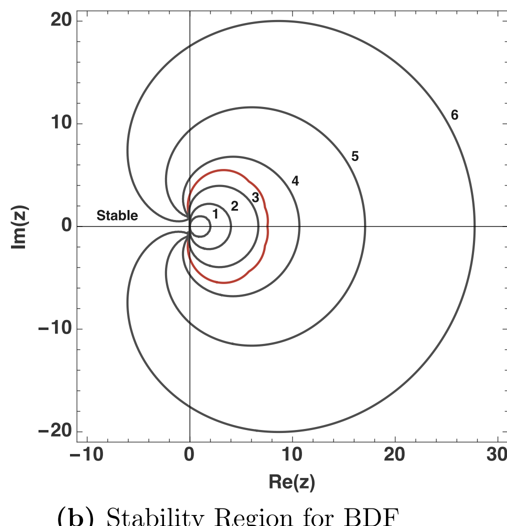
$$y' = \lambda y.$$

A linear stability diagram shows the values of $z = h\lambda$ for which a method is stable. The linear stability regions for the BBDF scheme (2.4) and the BDF schemes are shown in Figure 2.3. We can clearly see that moving the inputs into the complex plane provides significantly improved stability and allows for methods with order up to eight. In contrast, classical BDF methods are no longer root stable past order six.

How about cost? A single time-step with the BBDF scheme requires q nonlinear solves. At first glance this can seem prohibitively expensive in comparison to the single nonlinear solve required by BDF, However, the q nonlinear systems are independent and can be solved in parallel. We can therefore expect that classical BDF and parallelized BBDF will run in comparable times.



(a) Stability Regions for BBDF



(b) Stability Region for BDF

Figure 2.3: Linear stability regions for BBDF and BDF methods of orders 2 through eight. The stability contour for the 8th order BBDF method is labeled in red in both (a) and (b). The stability region for all methods is exterior to each curve.

2.2.3 A Time-Stepping Scheme Based on the Cauchy Integral Formula

We now propose to approximate the truncated Taylor series (2.3) by utilizing the Cauchy integral formula coupled with trapezoidal quadrature to approximate derivatives of $y(t)$. This approximation is well-conditioned, spectrally accurate, and allows for simple calculation of high-order derivatives [70, 31, 10, 11, 5]. This approach will lead us naturally to an explicit multivalue time-integrator that joins a host of other algorithms that use analytic functions at the the roots of unity [5].

By restricting ourselves to the class of initial value problems that satisfy the conditions of the Cauchy-Kowalevski theorem, we are guaranteed that there exists some $R > 0$ such that the solution $y(t)$ is analytic inside the circular domain of radius R centered around t_n. This allows us to express the local derivatives of the solution at $t = t_n$ using the Cauchy integral formula:

$$\begin{aligned} y^{(0)}(t_n) &= \frac{1}{2\pi i}\oint_\Gamma \frac{y(z)}{(z-t_n)}dz, \\ y^{(\nu\geq 1)}(t_n) &= \frac{(\nu-1)!}{2\pi i}\oint_\Gamma \frac{y'(z)}{(z-t_n)^\nu}dz \underset{\substack{\uparrow \\ \textbf{Apply ODE}}}{=} \frac{(\nu-1)!}{2\pi i}\oint_\Gamma \frac{F(z,y(z))}{(z-t_n)^\nu}dz, \end{aligned} \tag{2.7}$$

where Γ is a simple closed contour inside the disk of radius R enclosing t_n. If we take Γ to be a circular contour of radius $r < R$ centered at t_n, and apply the change of variables

$z = re^{i\theta} + t_n$, the contour integrals reduce to

$$
\begin{aligned}
y^{(0)}(t_n) &= \frac{1}{2\pi}\int_0^{2\pi} y(re^{i\theta}+t_n)d\theta, \\
y^{(\nu\geq 1)}(t_n) &= \frac{(\nu-1)!}{2\pi r^{\nu-1}}\int_0^{2\pi} \frac{F(re^{i\theta}+t_n, y(re^{i\theta}+t_n))}{e^{i\nu\theta}}d\theta,
\end{aligned}
\tag{2.8}
$$

The change of variables leads to locally analytic, periodic integrands. By discretizing the interval $[0, 2\pi)$ into the q points $\theta_j = \frac{2\pi(j-1)}{q}$, $j = 1, \ldots q$, we may approximate the integrals to exponential accuracy using the trapezoidal rule [96]. This leads us to the approximations

$$
y^{(0)}(t_n) \approx \sum_{j=1}^{q} \frac{y_j^{[n]}}{q}, \qquad y^{(\nu\geq 1)}(t_n) \approx \frac{(\nu-1)!}{qr^{\nu-1}} \sum_{j=1}^{q} f_j^{[n]} e^{-i(\nu-1)\theta_j}. \tag{2.9}
$$

where $y_j^{[n]}$ and $f_j^{[n]}$ are the solution and derivative values at the q roots of unity enclosing $t = t_n$:

$$
\left.\begin{array}{lll}
\text{inputs} & y_j^{[n]} &= y(t_j^{[n]}) \\
\text{input derivatives} & f_j^{[n]} &= F(t_j^{[n]}, y_j^{[n]}) \\
\text{input times} & t_j^{[n]} &= re^{i\theta_j} + t_n
\end{array}\right\} \qquad j = 1, \ldots, q.
$$

We may now substitute the derivative approximations (2.9) into the Taylor polynomial (2.3). In order to develop an iterative time-stepping method, we must evaluate the polynomial at the points $t_j^{[n+1]} = t_j^{[n]} + h$ for $j = 1, \ldots q$. The time-step iteration can be written as

$$
y_j^{[n+1]} = \frac{1}{q}\sum_{k=1}^{q} y_k^{[n]} + \sum_{\nu=1}^{q}\left[\frac{\left(h + re^{i\theta_j}\right)^{\nu}}{\nu q r^{\nu-1}} \sum_{k=1}^{q} f_k^{[n]} e^{-i(\nu-1)\theta_k}\right], \qquad j = 1, \ \ldots, \ q.
$$

It is convenient to introduce the *extrapolation factor* α and parametrize the stepsize h as the product of α and the radius r. Letting $h = r\alpha$ the time-step simplifies to

$$
y_j^{[n+1]} = \frac{1}{q}\sum_{k=1}^{q} y_k^{[n]} + r\sum_{k=1}^{q}\sum_{\nu=1}^{q}\left[\frac{\left(\alpha + e^{i\theta_j}\right)^{\nu} e^{-i(\nu-1)\theta_k}}{\nu q}\right] f_k^{[n]}, \qquad j = 1, \ \ldots, \ q. \tag{2.10}
$$

In Figure 2.4 we present a graphical representation of the input and output times $t_j^{[n]}$, $t_j^{[n+1]}$ and the parameters r, h and α. At the first time-step the method requires the solution at each of the q roots of unity.

The time-stepping scheme (2.10) is a Taylor method where the derivatives have been approximated by discrete Cauchy integrals. This explicit time-integrator has several interesting properties. First the derivation is simple to understand and the method is easy to implement at any order of accuracy by varying q. Second, unlike Adams-Bashforth methods, the linear stability regions for methods of orders two to seven do not contract to zero as order increases (See Figure 2.5). Finally, the method is parallel in the sense that each of the q right-hand-side evaluations may be computed simultaneously at each time-step.

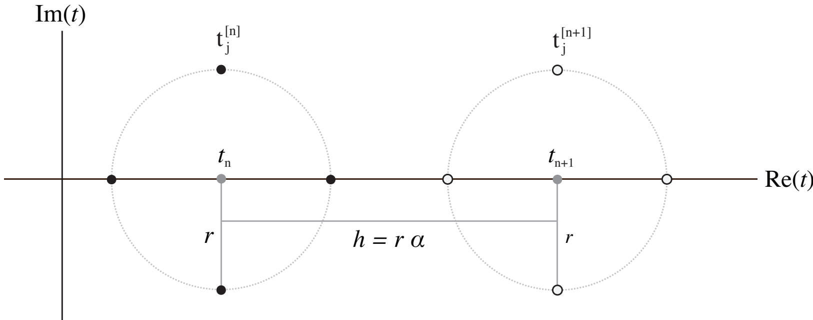


Figure 2.4: The complex t-plane containing the input times $t_j^{[n]}$ (black circles) and the output times $t_j^{[n+1]}$ (white circles) for $q = 4$. We also plot the time-step centers t_n, $t_{n+1} = t_n + h$ (grey circles) and two circles of radius r centered around t_n and t_{n+1}. The stepsize h has been parametrized as $h = r\alpha$, where α denotes number of radii r per timestep.

2.2.4 An Alternative Derivation using Polynomial Interpolation

We derived the integrator (2.10) by considering a truncated Taylor series where the exact derivatives were replaced with approximations computed using a discrete Cauchy integral formula. It is natural to ask if we may generalize this idea by considering other derivative approximations. We are motivated by the observation that the discrete Cauchy integral formula we used to derive (2.10) produces the same approximations as a polynomial-based finite difference formula with nodes at the roots of unity. We will briefly explore this connection and then re-derive the integrator (2.10) using polynomials.

Suppose we seek to approximate the derivatives of a function $g(z)$ at $z = 0$ using the values $g_k = g(z_k)$ where z_k are scaled roots of unity given by $z_k = r\exp(2\pi i(k-1)/q)$ for $k = 1, \ldots q$. Then, the following two algorithms are equivalent:

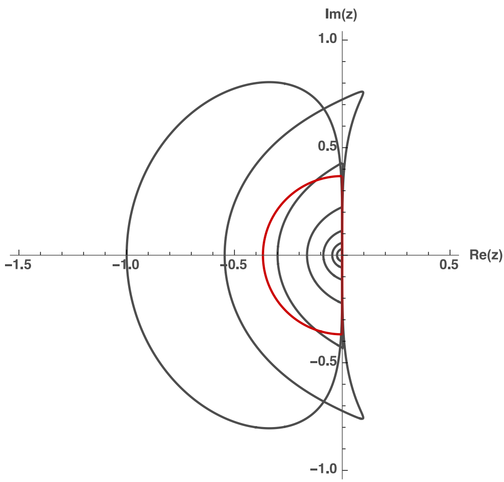
Algorithm 1. *Polynomial Differentiation*: Differentiate a Lagrange interpolating polynomial $p(z)$ that passes through each g_k, $k = 1, \ldots, q$, then $g^{(\nu)}(0) \approx p^{(\nu)}(0)$.

Algorithm 2. *Discrete Cauchy Integral Formula*: Approximate the Cauchy integral formula using a circular contour of radius r centered at zero and trapezoidal quadrature at each of the points z_k, then

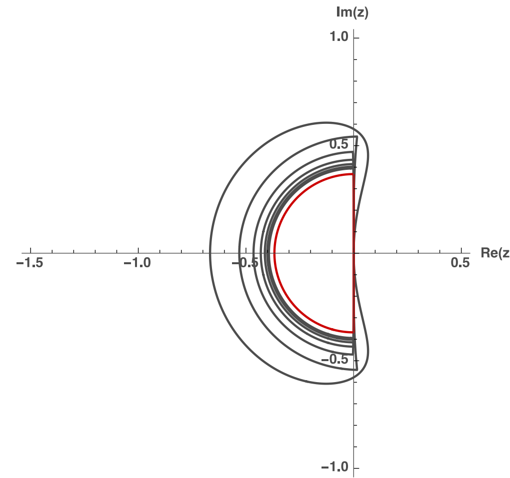
$$g^{(\nu)}(0) = \frac{\nu!}{2\pi i}\oint_\Gamma \frac{g(\zeta)}{\zeta^{\nu+1}} \approx \frac{\nu!}{qr^\nu}\sum_{k=1}^{q} g_k e^{-i\nu\theta_k} \qquad \text{for} \qquad \theta_k = \frac{2\pi(k-1)}{q}. \tag{2.11}$$

To prove equivalence, we express the Lagrange interpolating polynomial as

$$p(z) = \textstyle\sum_{k=0}^{q-1} a_k z^k$$



(a) Stability regions for the Adams-Bashforth methods of orders 2 through 7.



(b) Stability region for the method (2.10) of orders 2 through 7.

Figure 2.5: Linear stability regions for Adams-Bashforth methods and for the method (2.10). The red half-circle in both subfigures marks the perimeter of the region $\left\{|z| \leq \frac{1}{e}\right\} \cap \{\mathrm{Re}(z) \leq 0\}$ and is the conjectured limiting stability domain for method (2.10) as $q \to \infty$.

where the coefficients a_k are obtained by solving the $q \times q$ linear system

$$\mathbf{W}\mathbf{D}\,\mathbf{a} = \mathbf{g} \qquad \text{for} \qquad \begin{aligned} \mathbf{W}_{ij} &= (z_j/r)^{i-1} \\ \mathbf{D} &= \mathrm{diag}(1, r, \ldots r^{q-1}) \end{aligned}, \qquad \text{and} \qquad \begin{aligned} \mathbf{a} &= [a_0, \ldots, a_{q-1}]^T \\ \mathbf{g} &= [g_1, \ldots, g_q]^T \end{aligned}.$$

The Vandermonde matrix $\mathbf{W}$ is the inverse discrete Fourier transform matrix. We can obtain explicit formulas for the coefficients by inverting the system, yielding

$$\mathbf{a} = \mathbf{D}^{-1}\mathbf{W}^{-1}\mathbf{g} \qquad \text{for} \qquad \mathbf{W}^{-1}_{ij} = q^{-1}(z_j/r)^{1-i}.$$

Finally, using $p^{(\nu)}(0) = \nu! a_\nu$, we obtain the approximation (2.11). Had we approximated the derivatives at $z \neq 0$, then the trapezoidal approximation of the Cauchy integral formula amounts to rational approximation of the underlying function and the two algorithms would no longer be identical [5].

We can now present a second derivation for the method (2.10) using interpolating polynomials in the complex plane. At each timestep, we can use the inputs and input derivatives to form local Lagrange interpolating polynomials approximating $y(t)$ and $F(t, y(t))$. We may express these polynomials as

$$L_y^{[n]}(t) = \sum_{j=1}^{q} \ell_j(t) y_j^{[n]} \quad \text{and} \quad L_F^{[n]}(t) = \sum_{j=1}^{q} \ell_j(t) f_j^{[n]}, \quad \text{where} \quad \ell_j(z) = \prod_{\substack{l=1 \\ l \neq j}}^{q} \frac{z - z_j}{z_j - z_l}.$$

Next, we can approximate the derivatives of $y(t)$ at $t = t_n$ as

$$y^{(0)}(t_n) = L_y^{[n]}(0) \qquad \text{and} \qquad y^{(\nu\geq 1)}(t_n) = \frac{d^{\nu-1}}{dt^{\nu-1}} \, L_F^{[n]}(t)\Big|_{t=0}.$$

By substituting these derivative approximations into the Taylor series (2.3), and evaluating at each $t_j^{[n+1]}$ we obtain the method (2.10).

2.3 Developing a New Time Integration Framework using Polynomials

In this chapter we derived the diagonally-implicit integrator (2.4) by modifying BDF and the explicit integrator (2.10) by applying Lagrange interpolation at the roots of unity. In both cases, extending into the complex plane leads to new schemes with improved linear stability in comparison to classical real-valued linear multistep methods. In the subsequent chapters, we discuss polynomial approximations for the initial value problem and introduce a new framework for constructing a wide range of new polynomial-based integrators based on the following three generalizations:

1. Consider polynomials that pass through both function and derivative values at an arbitrary set of nodes

2. Expand the Taylor series (2.3) at an arbitrary point and approximate each derivative using a different polynomial approximation.

3. Construct methods that compute stage values.

Chapter 3

ODE POLYNOMIALS AND ODE DATASETS

In this chapter we discuss polynomial approximations for the initial value problem

$$y'(t) = F(t, y(t)), \qquad\qquad y(t_0) = y_0. \tag{3.1}$$

As is the case for classical function interpolation, we assume that a collection of solution values y_1, y_2, $\ldots$, y_w are provided for constructing these polynomials. By evaluating $F(t, y(t))$, we can also obtain derivative information at each point, thus allowing for a range of possible approximations. For example, we can construct a Lagrange interpolating polynomial that passes through known solution values or a Hermite interpolating polynomial that matches both solution and derivative values.

In the following sections we introduce the *ODE solution polynomial* and the *ODE derivative polynomial*. These two families of *ODE polynomials* compactly describe a range of approximations that include classical Lagrange and Hermite interpolants. Every ODE polynomial approximates a local Taylor series for the solution $y(t)$ or its derivative and is constructed from a corresponding *ODE dataset* that contains solution data. In the next chapter, we will show how the ODE polynomial and the ODE dataset form the foundation of our polynomial framework for time integration. For the moment, these two families of polynomials should be regarded more broadly as tools for locally approximating the solution to initial value problems. This chapter is dedicated to defining both objects and characterizing their properties.

Notation

We will frequently use the phrase let $p(t)$ be a polynomial "passing through" the solution values $y_1, \ldots, y_\omega$ and the derivatives values $f_1, \ldots, f_\nu$. By this we mean that the values and derivatives of $p(t)$ match with the prescribed solution or derivative values at their corresponding temporal nodes. In other words,

$$\begin{aligned} p(t_i) &= y_i & \qquad\qquad i &= 1, \ldots, \omega \\ p'(\hat{t}_j) &= f_j & \qquad\qquad j &= 1, \ldots, \omega \end{aligned}$$

where t_j and $\hat{t}_j$ are the temporal nodes of the solution and derivative values.

3.1 The ODE Dataset

An *ODE dataset* contains approximate solution and derivative values, along with their corresponding temporal nodes. The nodes are expressed in the local coordinates τ, where the

global time t is given by

$$t(\tau) = r\tau + s. \tag{3.2}$$

All ODE datasets are parametrized by the scaling factor r and the translation factor s, which together implicitly determine the temporal location of each data element.

Definition 3.1.1 (ODE Dataset)**.** *An ODE dataset $D(r, s)$ of size w is an ordered set of tuples of the form*

$$\{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{w} \tag{3.3}$$

where $t(\tau) = r\tau + s$, $y_j \approx y(t(\tau_j))$, and $f_j = F(t(\tau_j), y_j)$. Note that each derivative term in the dataset is multiplied by a factor of r to reflect the transformation into local coordinates τ where

$$\frac{d}{dt} = \frac{1}{r}\frac{d}{d\tau}.$$

The elements of any dataset $D(r, s)$ may labeled into different categories using the following notation:

$$\begin{cases} \text{data label 1} : \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{w_1} \\ \text{data label 2} : \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{w_2} \\ \qquad \vdots \\ \text{data label n} : \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{w_n} \end{cases}$$

where $\sum_{j=1}^{n} w_j = w$. When we introduce polynomial-based time-stepping methods, datasets will categorized using into inputs, outputs, and stage values.

Since ODE datasets may contain exact or approximate solution values, we introduce notation for describing their accuracy. We have not specified the manner in which the solution values are generated, therefore, it is possible that they depend on a number of different variables. To address this, we will introduce the function *order*, which accepts an ODE dataset and a variable name, and returns the order of accuracy of the solution values with respect to this variable.

Definition 3.1.2 (Dataset Order Function)**.** *The function* order *maps an ODE dataset $D(r, s) = \{(\tau_j, y_j, f_j)\}_{j=1}^{w}$ and a variable name x to an ordered set of w positive integers corresponding to the order of accuracy of each y_j with respect to the variable x. If a data value is exact,* order *returns ∞ for this element. We demonstrate the usage of* order *with the following equivalence:*

$$y_j = y(t(\tau_j)) + \mathcal{O}(x^{\rho_j+1}), \qquad \Longleftrightarrow \qquad \{\rho_1,\ \ldots,\ \rho_w\} = \text{order}(D(r, s); x)$$

If the dataset is labeled, then we will use the notation

$$\text{order}(D(r,s); x, \textit{datalabel})$$

to denote the order of accuracy of the solution values with the label "datalabel".

Remark 3.1.1. *If each of the solution values in an ODE dataset is at least $\mathcal{O}(r)$ accurate, then we can use a Taylor expansion to show that the order of accuracy of the local derivative terms rf_j with respect to r is one more than that of the solution values y_j:*

$$\begin{aligned} rf_j &= rF(t_j, y_j) \\ &= rF(t_j, y(t_j)) + r\sum_{j=1}^{\infty} F^{(j)}(t_j, y(t_j)) \underbrace{(y_j - y(t_j))}_{\mathcal{O}(r^{\rho_j+1})} \\ &= rF(t_j, y(t_j)) + \mathcal{O}(r^{\rho_j+2}) \end{aligned}$$

where $t_j = t(\tau_j) = r\tau_j + s$.

3.2 The ODE Solution Polynomial

Every *ODE solution polynomial* approximates the solution of the initial value problem (3.1) and is constructed using data from an ODE dataset. All ODE solution polynomials can be written as a truncated Taylor series of $y(t)$ in which every exact derivative has been replaced with an approximation obtained by differentiating an interpolating polynomial. If an ODE solution polynomial is built using an ODE dataset $D(r,s)$, then it will be expressed in terms of the local coordinates τ where $t(\tau) = r\tau + s$.

Definition 3.2.1 (ODE Solution Polynomial)**.** *An ODE solution polynomial $p(\tau; b)$ approximates the truncated Taylor series for the local solution $y(t(\tau))$, where $t(\tau) = r\tau + s$, expanded around the point $\tau = b$. An ODE solution polynomial of degree g can be expressed as*

$$p(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!} \tag{3.4}$$

where each approximate derivative $a_j(b)$ is computed using values from an ODE dataset $D(r,s)$ in one of the following ways:

1. Differentiate a polynomial approximation to $y(t(\tau))$: *Construct a local interpolating polynomial $h_j(\tau)$ passing through any subset of the ODE dataset $D(r,s)$ that includes at least one solution value. Then,*

$$h_j(\tau) \approx y(t(\tau)) \qquad \textit{and} \qquad a_j(b) = \frac{d^j}{d\tau^j} h_j(\tau)\bigg|_{\tau=b}.$$

2. Differentiate a polynomial approximation to $rF(t(\tau), y(t(\tau)))$: *Construct a local interpolating polynomial $l_j(\tau)$ passing through any subset of the derivative values in the ODE dataset $D(r, s)$. Then,*

$$l_j(\tau) \approx F(t(\tau), y(t(\tau))) \quad \text{and} \quad a_j(b) = \frac{d^{j-1}}{d\tau^{j-1}} l_j(\tau) \bigg|_{\tau=b} \quad \text{(only valid for } j \geq 1\text{).}$$

Every ODE solution polynomial approximates the local solution $y(t(\tau))$ such that

$$y(t(\tau)) \approx p(\tau; b), \quad \forall b.$$

It follows that ODE solution polynomials depend implicitly on the scaling factor r. However, we will avoid writing $p(\tau, r; b)$ since the dependance on r does not arise from the structure of the polynomial, but instead comes from the values in the ODE dataset $D(r, s)$ used to compute the derivative terms $a_j(b)$.

Remark 3.2.1. *If an ODE dataset contains non-distinct nodes τ_i (i.e. more than one approximation of $y(t)$ is provided at the same temporal node), then any interpolating polynomial $h_j(\tau)$ or $l_j(\tau)$ that match solution or derivative information at these nodes must pass through a convex combination of the corresponding solution or derivative values.*

3.2.1 Special Families of ODE Solution Polynomials

The family of all ODE solution polynomials is large and the Taylor formulation (3.4) provides little guidance for choosing derivative approximations. To aid in the exploration of these polynomials, we introduce several special families that are inspired by classical linear multistep methods. Let $p(\tau; b)$ be an *ODE polynomial* of degree g

$$p(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!}.$$

constructed from a dataset $D(r, s)$ of size w. Then, we say that:

1. The polynomial $p(\tau; b)$ is an **Adams** ODE solution polynomial if

$$a_j(b) = \begin{cases} L_y(b) & j = 0 \\ \frac{d^{j-1}}{dx^{j-1}} L_F(\tau) \Big|_{\tau=b} & j > 0 \end{cases} \tag{3.5}$$

where $L_y(\tau)$ is a Lagrange interpolating polynomial passing through any subset of the solution data and $L_F(\tau)$ is a Lagrange interpolating polynomial of degree g that passes through a subset of the derivative data. We may write any Adams ODE solution polynomial in integral form as

$$p(\tau; b) = L_y(b) + \int_b^\tau L_F(\xi) d\xi, \tag{3.6}$$

where the expansion point b serves as a left integration endpoint. Adams ODE solution polynomials are characterized by the data used to construct the Lagrange interpolants $L_y(\tau)$ and $L_F(\tau)$. If $L_y(\tau)$ passes through the subset of solution data y_{c_i} for $i = 1, \ldots, \gamma$ and $L_F(\tau)$ passes through the subset of derivative data f_{d_i} for $i = 1, \ldots, \kappa$, then we will use the following notation to describe the data values used to form $p(\tau; b)$:

$$L_y : \{y_{c_1}, \; \ldots \; , \; y_{c_\gamma}\} \qquad\qquad L_F : \{f_{d_1}, \; \ldots \; , \; f_{d_\kappa}\}. \tag{3.7}$$

We call both (3.5, 3.6) an Adams ODE solution polynomial since a single time-step of the Adams-Bashforth and Adams-Moulton methods may be expressed in terms of an Adams ODE solution polynomial passing through equispaced data.

2. The polynomial $p(\tau; b)$ is a **BDF** ODE solution polynomial if

$$a_j(b) = \frac{d^j}{dx^j} H_y(\tau)\bigg|_{\tau=b} \qquad\qquad i = 1, \ldots, g \tag{3.8}$$

where $H_y(\tau)$ is an interpolating polynomial of degree g passing through one derivative value and at least one solution value. Note that a BDF ODE solution polynomial does not depend on b since

$$p(\tau; b) = H_y(\tau), \quad \forall b.$$

BDF ODE solution polynomials are characterized by the solution and derivative data used to construct $H_y(\tau)$. If $H_y(\tau)$ passes through the derivative value f_d, for any $d \in \{1, \ldots, w\}$ and the subset of solution data y_{c_i} for $i = 1, \ldots, \gamma$, then we will use the following notation to describe the data values used to form $p(\tau; b)$:

$$H_y : \{y_{c_1}, \; \ldots \; , \; y_{c_\gamma}, \; f_d\}. \tag{3.9}$$

We call (3.8) a BDF ODE solution polynomial since a single time-step of a classical BDF method may be expressed in terms of a BDF ODE solution polynomial passing through equispaced data.

3. The polynomial $p(\tau; b)$ is a **Generalized-BDF** (GBDF) ODE solution polynomial if

$$a_j(b) = \frac{d^j}{dx^j} H_y(\tau)\bigg|_{\tau=b} \qquad\qquad i = 1, \ldots, g \tag{3.10}$$

where $H_y(\tau)$ is an interpolating polynomial of degree g passing through at least one derivative value and at least one solution value. Like BDF ODE solution polynomials, a GBDF ODE solution polynomial also does not depend on b and is characterized by the solution and derivative data used to construct $H_y(\tau)$. If $H_y(\tau)$ passes through the subset of solution data y_{c_i} for $i = 1, \ldots, \gamma$, and the subset of the derivative data f_{d_i} for $i = 1, \ldots, \kappa$ then we will use the following notation to describe the GBDF ODE solution polynomial:

$$H_y : \{y_{c_1}, \; \ldots \; , \; y_{c_\gamma}, \; f_{d_1}, \; \ldots \; , f_{d_\kappa}\}. \tag{3.11}$$

3.2.2 Computing Coefficients

Evaluating an ODE solution polynomial amounts to taking a weighted linear combination of solution and derivative data. Each weight depends on the evaluation location and can be computed by solving linear systems. The number of linear systems is dependent on the number of unique polynomials used to compute the approximate derivatives of $a_j(b)$ and the size of each linear system is given by the order of these polynomials. Here we describe a procedure for generating the weights for evaluating an ODE solution polynomial

$$p(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!}$$

built from an ODE dataset $\mathcal{D}(r,s) = \{(\tau_j, y_j, rf_j)\}_{j=1}^{w}$. We begin by introducing the data vector

$$\mathbf{d} = [y_1, \ \ldots, \ y_w, \ rf_1, \ \ldots, \ rf_w]^{\mathrm{T}}$$

that contains all the solution and derivative data in $D(r,s)$. Next, we rewrite $p(\tau; b)$ as a weighted linear combination of the data elements, where each weight depends on τ and b, via

$$p(\tau; b) = \mathbf{w}(\tau; b) \cdot \mathbf{d}$$

where $\mathbf{w}(\tau; b) \in \mathbb{C}^{2w}$. The weight vector $\mathbf{w}(\tau; b)$ can be separated as

$$\mathbf{w}(\tau; b) = \sum_{j=0}^{g} (\tau - b)^j \mathbf{a}(j; b)$$

where the vectors $\mathbf{a}(j; b)$ no longer depend on τ and satisfy

$$\mathbf{a}(j; b) \cdot \mathbf{d} = \frac{a_j(b)}{j!}, \qquad j = 0, \ldots, g.$$

To simplify our discussion for computing the vectors $\mathbf{a}(j; b)$, we introduce the following variables:

- $\mathbb{A}$ set of size γ containing indices of the solution data used to compute $a_j(b)$.
- $\mathbb{B}$ set of size ϕ containing indices of the derivative data used to compute $a_j(b)$.
- ν the order of the underlying polynomial, $\nu = \gamma + \phi - 1$.

The approximate derivatives $a_j(b)$ are computed by differentiating interpolating polynomials that pass through solution and derivative values. For notational simplicity, we assume that the temporal nodes of these data values are unique such that $\tau_k \neq \tau_l$ for all $k, \ l \in \mathbb{A} \cup \mathbb{B}$. Though notationally tedious, this algorithm can be easily generalized for polynomials that pass through convex combinations of values at the same time point. Assuming unique temporal nodes, we may compute $\mathbf{a}(j; b)$ as follows:

1. *If $a_j(b)$ is computed by differentiating a polynomial approximation to $y(\tau)$*

   The polynomial approximation to $y(\tau)$ can expanded around $\tau = b$ as

   $$h_j(\tau) = \sum_{k=0}^{\nu} c_k(\tau - b)^k \qquad \text{where the coefficients } c_k \text{ must satisfy the conditions}$$

   $$\sum_{k=0}^{\nu} c_k(\tau_l - b)^k = y_l \text{ for all } l \in \mathbb{A} \qquad \text{and} \qquad \sum_{k=1}^{\nu} k c_k(\tau_l - b)^{k-1} = r f_l \text{ for all } l \in \mathbb{B}.$$

   These conditions will lead to a $(\nu+1) \times (\nu+1)$ linear system $\mathbf{Hc} = \mathbf{y}$ where

   $$\left[\begin{array}{ccccc} 1 & \tau_{\mathbb{A}_1} & \tau_{\mathbb{A}_1}^2 & \dots & \tau_{\mathbb{A}_1}^{\nu} \\ \vdots & \vdots & & & \vdots \\ 1 & \tau_{\mathbb{A}_\gamma} & \tau_{\mathbb{A}_\gamma}^2 & \dots & \tau_{\mathbb{A}_\gamma}^{\nu} \\ \hline 0 & 1 & 2\tau_{\mathbb{B}_1} & \dots & \nu\tau_{\mathbb{B}_1}^{\nu-1} \\ \vdots & \vdots & & \vdots & \\ 0 & 1 & 2\tau_{\mathbb{B}_\phi} & \dots & \nu\tau_{\mathbb{B}_\phi}^{\nu-1} \end{array}\right] \left[\begin{array}{c} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_\nu \end{array}\right] = \left[\begin{array}{c} y_{\mathbb{A}_1} \\ \vdots \\ y_{\mathbb{A}_\gamma} \\ \hline r f_{\mathbb{B}_1} \\ \vdots \\ r f_{\mathbb{B}_\phi} \end{array}\right].$$

   If $\mathbf{H}$ is invertible, then the non-zero entries of the vector $\mathbf{a}(j;b)$ are given by

   $$[\mathbf{a}(j;b)]_{\mathbb{A}_k} = \mathbf{H}^{-1}_{j+1,k}, \qquad k = 1, \dots, \gamma \quad \text{and} \quad [\mathbf{a}(j;b)]_{\mathbb{B}_k} = \mathbf{H}^{-1}_{j+1,k+\gamma} \qquad k = 1, \dots, \phi.$$

   Note that $\mathbf{a}(j;b)$ is computed from the $(j+1)$th row of $\mathbf{H}^{-1}$ since our index j runs from zero to ν.

   If $\mathbf{H}$ is non-invertible, than the $p(\tau;b)$ is either non unique or does not exist. In either case, we will consider such polynomials not suitable for time-stepping.

2. *If $a_j(b)$ is computed by differentiating a polynomial approximation to $f(\tau, y(\tau))$*

   In this case, $\gamma = 0$, and the polynomial approximation to $f(\tau, y(\tau))$ can expanded around $\tau = b$ as

   $$l_j(\tau) = \sum_{k=0}^{\nu} c_k(\tau - b)^k \qquad \text{where} \qquad \sum_{k=0}^{\nu} c_k(\tau_l - b)^k = r f_l \text{ for all } l \in \mathbb{B}.$$

   These conditions will lead to a $(\nu+1) \times (\nu+1)$ Vandermonde system $\mathbf{Lc} = \mathbf{f}$ where

   $$\left[\begin{array}{cccc} 1 & \tau_{\mathbb{B}_1} & \dots & \tau_{\mathbb{B}_1}^{v} \\ \vdots & \vdots & & \vdots \\ 1 & \tau_{\mathbb{B}_\gamma} & \dots & \tau_{\mathbb{B}_\gamma}^{v} \end{array}\right] \left[\begin{array}{c} c_0 \\ \vdots \\ c_\nu \end{array}\right] = \left[\begin{array}{c} r f_{\mathbb{B}_1} \\ \vdots \\ r f_{\mathbb{B}_\phi} \end{array}\right].$$

The non-zero entries of the vector $\mathbf{a}(j,b)$ are given by

$$[\mathbf{a}(j;b)]_{\mathbb{B}_k} = \mathbf{L}^{-1}_{j,k}/j \qquad k = 1, \dots, \phi.$$

3.3 The ODE Derivative Polynomial

An *ODE derivative polynomial* approximates the solution derivative $F(t, y(t))$ of the initial value problem (3.1). Like all ODE polynomials, ODE derivative polynomials are built from an ODE dataset and are expressible as local Taylor series where the derivatives have been replaced with approximations obtained by differentiating interpolating polynomials. If an ODE derivative polynomial is built using an ODE dataset $D(r,s)$, then it will be expressed in terms of the local coordinates τ where $t(\tau) = r\tau + s$.

Definition 3.3.1 (ODE Derivative Polynomial)**.** *An ODE derivative polynomial approximates the Taylor series for the local solution derivative $rF(t(\tau), y(t(\tau)))$, where $t(\tau) = r\tau + s$, expanded around the point $\tau = b$. Every ODE derivative polynomial can be expressed as*

$$\dot{p}(\tau;b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!} \tag{3.12}$$

where each approximate derivative $a_j(b)$ is computed using values from an ODE dataset $D(r,s)$ in either of the following ways:

1. Differentiate a polynomial approximation to $y(t(\tau))$: *Construct a local interpolating polynomial $h_j(\tau)$ passing through any subset of an ODE Dataset that includes* <u>*at least one solution value*</u>*. Then,*

$$h_j(\tau) \approx y(t(\tau)) \qquad \text{and} \qquad a_j(b) = \left. \frac{d^{j+1}}{d\tau^{j+1}} h_j(\tau) \right|_{\tau=b}.$$

2. Differentiate a polynomial approximation to $rF(t(\tau), y(t(\tau)))$: *Construct a local interpolating polynomial $l_j(\tau)$ passing through any subset of the* <u>*derivative values*</u> *in the dataset $\mathcal{D}$. Then,*

$$l_j(\tau) \approx F(t(\tau), y(t(\tau))) \qquad \text{and} \qquad a_j(b) = \left. \frac{d^j}{d\tau^j} l_j(\tau) \right|_{\tau=b}.$$

Every ODE derivative polynomial approximates the local derivative such that

$$\dot{p}(\tau;b) \approx rF(t(\tau), y(t(\tau))) \quad \forall b.$$

3.3.1 Special Families of ODE Derivative Polynomial

As was the case for ODE solution polynomials, we can construct Adams, BDF, and GBDF ODE derivative polynomials. Let $\dot{p}(\tau; b)$ be an ODE derivative polynomial, constructed from a dataset $\mathcal{D}(r, s)$, where

$$\dot{p}(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!}, \qquad \text{and} \qquad \mathcal{D}(r, s) = \{(\tau_j, y_j, f_j)\}_{j=1}^{w}.$$

Then, we say that:

1. The polynomial $\dot{p}(\tau; b)$ is an **Adams** ODE derivative polynomial if

$$a_j(b) = \frac{d^j}{dx^j} L_F(\tau)\bigg|_{\tau=b} \qquad j = 1, \ldots, g, \tag{3.13}$$

where $L_F(x)$ is a Lagrange interpolating polynomial of degree g that passes through a subset of the derivative data. An Adams ODE derivative polynomial does not depend on the expansion point b since

$$\dot{p}(\tau; b) = L_F(\tau), \quad \forall b \tag{3.14}$$

If $L_F(x)$ passes through the subset of derivative data f_{b_i} for $i = 1, \ldots, \kappa$, then we will again use the notation (3.7) to describe the active data values of the Adams derivative polynomial.

2. The polynomial $\dot{p}(\tau; b)$ is a **GBDF** ODE derivative polynomial if

$$a_j(b) = \frac{d^{j+1}}{dx^{j+1}} H(\tau)\bigg|_{\tau=b} \qquad j = 1, \ldots, g, \tag{3.15}$$

where $H(x)$ is an interpolating polynomial of degree $g + 1$ passing through at least one derivative value and at least one solution value. A GDF polynomial does not depend on the expansion point b since

$$p(\tau; b) = H'(\tau), \quad \forall b.$$

If $H(\tau)$ passes through the subset of solution data y_{a_i} for $i = 1, \ldots, \gamma$, and the subset of the derivative data f_{b_i} for $i = 1, \ldots, \kappa$ then we will again use the notation (3.11) to describe the active data values of the GBDF derivative polynomial.

3. The polynomial $\dot{p}(\tau; b)$ is a **BDF** ODE derivative polynomial if it is a GBDF derivative polynomial with $\kappa = 1$. We will denote active data values for a BDF ODE derivative polynomial using the notation (3.9).

3.3.2 Computing Coefficients

Here we describe a procedure for calculating the weights for evaluating an ODE derivative polynomial $\dot{p}(\tau; b)$ built from the dataset $\mathcal{D}(r, s)$ where

$$\dot{p}(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!}, \qquad \text{and} \qquad \mathcal{D}(r, t_n) = \{\tau_j, y_j, rf_j\}_{j=1}^{w}.$$

As was the case for ODE solution polynomials, evaluating the ODE derivative polynomial is equivalent to taking a weighted linear combination of the elements of the data vector

$$\mathbf{d} = [y_1, \ldots, y_w, rf_1, \ldots, rf_w]^{\mathrm{T}}.$$

We may express the ODE derivative polynomial via $\dot{p}(\tau; b) = \dot{\mathbf{w}}(\tau; b) \cdot \mathbf{d}$, where

$$\mathbf{w}(\tau; b) = \sum_{j=0}^{g} (\tau - b)^j \mathbf{a}(j; b) \qquad \text{and the vectors } \mathbf{a}(j; b) \text{ satisfy} \qquad \mathbf{a}(j; b) \cdot \mathbf{d} = \frac{a_j(b)}{j!}$$

Using the notation and assumptions from subsection 3.2.2, we may compute $\mathbf{a}(j; b)$ as follows:

1. *If $a_j(b)$ is computed by differentiating a polynomial approximation to $y(\tau)$*

   Form the linear system $\mathbf{Hc} = \mathbf{y}$ from subsection 3.2.2. If $\mathbf{H}$ is invertible then the non-zero entries of the vector $\mathbf{a}(j; b)$ are

$$\begin{aligned} [\mathbf{a}(j; b)]_{\mathbb{A}_k} &= (j + 1)\mathbf{H}^{-1}_{j+2,k}, & k &= 1, \ \ldots, \ \gamma \\ [\mathbf{a}(j; b)]_{\mathbb{B}_k} &= (j + 1)\mathbf{H}^{-1}_{j+2,k+\gamma} & k &= 1, \ \ldots, \ \phi. \end{aligned}$$

   If $\mathbf{H}$ is non-invertible, than the ODE polynomial is not valid and cannot be evaluated.

2. *If $a_j(b)$ is computed by differentiating a polynomial approximation to $f(\tau, y(\tau))$*

   Form the linear system $\mathbf{Lc} = \mathbf{f}$ from subsection 3.2.2. The non-zero entries of the vector $\mathbf{a}(j, b)$ are

$$[\mathbf{a}(j; b)]_{\mathbb{B}_k} = \mathbf{L}^{-1}_{j+1,k} \qquad k = 1, \ldots, \phi.$$

3.4 Properties of ODE Polynomials

We endow every ODE polynomial with a set of properties that are useful for characterization, construction, and comparison with other polynomials. In this section we introduce these properties and discuss several strategies for constructing ODE polynomial with certain characteristics. As usual, let $p(\tau; b)$ be an ODE polynomial formed from the ODE dataset

$$D(r, s) = \{(\tau_j, y_j, f_j)\}_{j=1}^{w}.$$

When we discuss order and truncation error it is convenient to denote the function that $p(\tau;b)$ approximates as $v(\tau)$. If $p(\tau;b)$ is an ODE solution polynomial, then

$$v(\tau) = y(t(\tau)), \text{ for } t(\tau) = r\tau + s.$$

Similarly, if $p(\tau;b)$ is an ODE derivative polynomial, then

$$v(\tau) = F(t(\tau), y(t(\tau))) \text{ for } t(\tau) = r\tau + s.$$

Every ODE polynomial $p(\tau;b)$ has the following properties:

1. **Degree.** The degree of $p(\tau;b)$ is the degree of the highest non-zero monomial in the variable τ.

2. **Order of Accuracy.** The ODE polynomial $p(\tau;b)$ is order ρ accurate with respect to the node radius r if ρ is a positive integer satisfying

   $$|p(\tau;b) - v(\tau)| = \mathcal{O}\left(r^{\rho+1}\right) \qquad \text{for any fixed } \tau.$$

   Recall that the temporal locations of solution and derivative values in an ODE dataset depend on the node radius r, and thus the polynomial $p(\tau;b)$ is implicitly dependent on the node radius r.

3. **Truncation Error.** The truncation error for $p(\tau;b)$ is given by

   $$\text{TE}(r;\tau) = |\tilde{p}(\tau;b) - v(\tau)|$$

   where $\tilde{p}(\tau;b)$ is the ODE polynomial we would obtain, had we replaced the dataset $D(r,s)$ with one containing the exact solution values $y_j = y(t(\tau_j))$. Moreover, we say that the truncation error is of order ρ if $\text{TE}(r;\tau) = \mathcal{O}(r^\rho)$ for any fixed τ.

4. **The Active Value Set.** The active value set for $p(\tau;b)$ is the set of all solution data y_j and derivative data f_j used to form $p(\tau;b)$. An individual piece of solution or derivative data is *active* if it is a member of the active value set.

5. **The Active Node Set.** The active node set for an ODE polynomial $p(\tau;b)$ is the set of unique temporal locations τ_i of all the active data. An individual temporal node is *active* if it is a member of the active node set.

3.4.1 Truncation Error and Order of Accuracy

When constructing and analyzing ODE polynomials, we will be interested in determining their truncation error and order of accuracy. The truncation error is a theoretical property that allows us to assess the quality of an ODE polynomial independently of its underlying dataset. Conversely, the order of accuracy for an ODE polynomial depends on the order of accuracy of the ODE dataset used for construction and reflects the error we would expect to achieve on a real problem. In certain applications however, these two quantities will be equal and it is sufficient to consider truncation error.

3.4.1.1 Determining Truncation Error

The truncation error for any ODE polynomial

$$p(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!}$$

can be formally determined by Taylor expanding all the exact data values used to form $\tilde{p}(\tau; b)$ (recall that $\tilde{p}(\tau; b)$ is the ODE polynomial we would obtain, had we replaced the dataset for constructing $p(\tau; b)$ with one containing the exact solution values) and collecting powers of r in the expression

$$\tilde{p}(\tau; b) - v(\tau).$$

However, if the construction strategy for an ODE polynomial is known, then the order of the truncation is given by

$$\rho = \min\left(g + 1,\ \lambda_1,\ \ldots,\ \lambda_g\right)$$

where the constants λ_j are computed as follows:

1. *If $a_j(b)$ is computed by differentiating a polynomial approximation to $y(\tau)$ of degree d*

$$\lambda_j = \begin{cases} \infty & j = 0 \text{ and the polynomial approximation passes through } y(b) \\ \infty & j = 1 \text{ and the polynomial approximation passes through } f(b, y(b)) \\ d + 1 & \text{otherwise} \end{cases}$$

2. *If $a_j(b)$ is computed by differentiating a polynomial approximation to $f(\tau, y(\tau))$ of degree d*

$$\lambda_j = \begin{cases} \infty & j = 1 \text{ and the polynomial approximation passes through } f(b, y(b)) \\ d + 2 & \text{otherwise} \end{cases}$$

The constants λ_j are lower bounds on the order of accuracy of the approximate derivatives $a_j(b)$ which satisfy the equality

$$\left| a_j(b) - r^j g^{(j)}(rb + s) \right| = \mathcal{O}\left(r^{\lambda_j}\right).$$

The formula for λ_j is obtained as follows. If the polynomial approximation for computing $a_j(b)$ passes through the jth derivative at $\tau = b_0$, then the approximation is exact and λ_j is infinite. Otherwise, if $a_j(b)$ is computed by differentiating an approximation to $y(\tau)$, then λ_j is equal to the total number of data values that the polynomial passes through. If $a_j(b)$ is computed by differentiating an approximation to $f(\tau, y(\tau))$, then we gain an additional order of accuracy since the polynomial passes exclusively though derivative data, which is multiplied by a factor of r in local coordinates.

3.4.1.2 Constructing Polynomials With a Given Truncation Error

To construct an ODE Polynomial with a truncation error of order ρ we must choose derivative approximations of sufficiently high degree. Here we summarize construction strategies for several families of ODE polynomials.

1. *Family:* All ODE polynomials.
   Construction: The ODE polynomial must be of degree $\rho - 1$ and the polynomials for computing the terms $a_j(b)$ must be of degree greater then or equal to $\rho - 1$ if they approximate $y(t)$ and of degree greater then or equal to $\rho - 2$ if they approximate $F(t, y(t))$.

2. *Family:* BDF or GBDF.
   Construction: The ODE polynomial and the polynomial $H(\tau)$ must both be of degree $\rho - 1$.

3. *Family:* Adams.
   Construction: The ODE polynomials $L_y(\tau)$ and $L_F(\tau)$ must both be of degree $\rho - 1$. Either choose $L_y(\tau)$ of degree ρ and $L_F(\tau)$ of degree $\rho - 1$, **or** choose $L_F(\tau)$ of degree $\rho - 2$ and choose the expansion points b_j that are each equal to any interpolation node of $L_y(\tau)$.

3.4.1.3 Determining Order of Accuracy

An ODE polynomial $p(\tau; b)$ is order ν accurate if

$$|p(\tau; b) - y(t(\tau))| = \mathcal{O}\left(r^{\nu+1}\right) \qquad \text{for any fixed } \tau.$$

The order of accuracy depends on the truncation error of the polynomial and on the accuracy of the ODE dataset $D(r, s)$ used to form $p(\tau; b)$. The order of accuracy is equal to the smallest of the following quantities: 1) the truncation error order, and 2) the lowest order of accuracy of the active data values. Let $D(r, s)$ be an ODE dataset such that

$$D(r, s) = \{(\tau_j, y_j, f_j)\}_{j=1}^{w} \qquad \text{where} \qquad \{\mu_1, \ \ldots, \ \mu_w\} = \text{order}(D(r, s); r)$$

and let $p(\tau; b)$ be constructed using the solution values y_j for $j \in \mathbb{A}$ and the derivative values f_j for $j \in \mathbb{B}$ where the sets $\mathbb{A}$ and $\mathbb{B}$ are of dimension $|\mathbb{A}|$, $|\mathbb{B}|$. The order of accuracy ν is given by

$$\nu = \min\left(\rho - 1, \ \mu_{\mathbb{A}_1}, \ \ldots, \ \mu_{\mathbb{A}_{|\mathbb{A}|}}, \ 1 + \mu_{\mathbb{B}_1}, \ \ldots, \ 1 + \mu_{\mathbb{B}_{|\mathbb{B}|}}\right)$$

where the truncation error is $\rho = \min(g + 1, \ \lambda_1, \ \ldots, \ \lambda_g)$.

3.4.1.4 Constructing Polynomials With a Given Order of Accuracy

To construct an ODE polynomial with an order of accuracy ρ, we require an ODE dataset with values that are at least order ν accurate and a construction strategy that produces an ODE polynomial with truncation error $\rho = \nu + 1$.

3.4.2 Symmetric Polynomials and Conjugate Polynomials

If the initial value problem (3.1) is real-valued, then it follows from the Swartz reflection principle that the solution $y(t)$ and its derivative $F(t, y(t))$ satisfy

$$y(z^*) = y(z)^* \qquad \text{and} \qquad F(z^*, y(z^*)) = F(z, y(z))^*.$$

where * denotes the complex conjugate. An ODE polynomial will preserve this property if: (1) it was constructed using data that satisfies the Swartz reflection principle and (2) the approximate derivatives were computed using polynomials passing through solution and derivative values symmetrically with respect to the real-axis. We will call these polynomials *symmetric*.

It may not always be desirable to construct symmetric polynomials. If we must approximate the solution (or its derivative) at the points $t = z$ and $t = z^*$ using two ODE polynomials $p_1(\tau; b_1)$ and $p_2(\tau; b)$, then it is generally desirable that our approximations satisfy the Swartz reflection principle for real initial value problems. ODE polynomial pairs that satisfy this relation will be called *conjugate*.

Definition 3.4.1 (Conjugate ODE Polynomials)**.** *We say that two ODE polynomials $p_1(\tau; b_1)$ and $p_2(\tau; b_2)$ are conjugate if*

$$p_1(z; b_1)^* = p_2(z^*; b_2) \quad \forall z$$

where * denotes complex conjugate. Notice that if $g_1 = p_1(z; b_1)$ and $g_2 = p_2(z^*; b_1)$ then $g_1 = g_2^*$. In Figure 3.1 we show ODE polynomial diagrams for a symmetric GBDF polynomial, and two conjugate BDF polynomials.

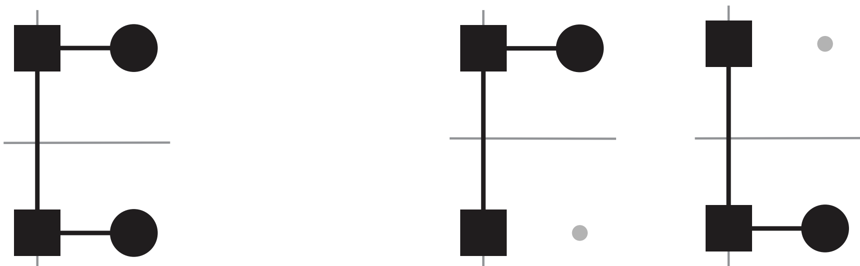
We can determine whether two simple ODE polynomial are conjugate by inspection. For more complex polynomials we close this section with a simple lemma and Theorem.

Lemma 3.4.1 (Reflected Interpolation Conditions)**.** *Let $\{z_j\}_{j=1}^{w}$ be a set of distinct nodes and let $p(z)$ and $q(z)$ be polynomials of degree $\gamma - 1 < 2w$. Next, suppose that each polynomial satisfies γ total conditions. The polynomial $p(z)$ must satisfy at least one of the conditions*

$$p(z_j) = a_j, \qquad j = 1, \ldots, w$$

and at most $\gamma - 1$ of the conditions

$$p'(z_j) = b_j \qquad j = 1, \ldots, w$$



(a) Symmetric Polynomial **(b)** Pair of Conjugate Polynomials

Figure 3.1: Diagrams for three ODE solution polynomials constructed from the ODE dataset $D(r, s) = \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{4}$ with temporal nodes $\tau_1 = i$, $\tau_2 = -i$, $\tau_3 = i + 1$, $\tau_4 = -i + 1$.

The polynomial $q(z)$ must satisfy at least one of the conditions

$$q(z_j^*) = a_j^*, \qquad\qquad j = 1, \ldots, w$$

and at most $\gamma - 1$ of the conditions

$$q'(z_j^*) = b_j^* \qquad\qquad j = 1, \ldots, w$$

If $p(z)$ and $q(z)$ satisfy the same subset of conditions such that

$$p(z_k) = a_k \iff q(z_k^*) = a_k^* \qquad \text{and} \qquad p'(z_k) = b_k \iff q'(z_k^*) = b_k^*$$

for all $k = 1, \ldots, q$, then $p^\nu(z)^* = q^\nu(z^*)$ for $\nu = 1, 2, \ldots$.

Proof. The polynomials $p(z)$ and $q(z)$ can be expanded as

$$p(z) = \sum_{j=0}^{\gamma-1} c_j z^j \qquad \text{and} \qquad g(z) = \sum_{j=0}^{\gamma-1} d_j z^j$$

From the Swartz reflection principle, it follows that the coefficient vectors $c = [c_0, \ldots, c_{\gamma-1}]$ and $d = [d_0, \ldots, d_{\gamma-1}]$ satisfy linear systems of the form

$$\mathbf{Ac} = \mathbf{x} \qquad \text{and} \qquad \mathbf{A}^*\mathbf{d} = \mathbf{x}^*$$

where the operation $*$ replaces each element of a matrix or vector with its complex conjugate. Therefore $c_j = d_j^*$, and

$$p(z)^* = \left(\sum_{j=0}^{\gamma-1} c_j z^j \right)^* = \sum_{j=0}^{\gamma-1} c_j^* (z^*)^j = g(z^*) \tag{3.16}$$

By differentiating the series we see this result holds true for derivatives as well. □

Theorem 3.4.2. *Let $p(\tau;b_1)$ and $q(\tau;b_2)$ be three ODE polynomial of degree γ. If the following three conditions are satisfied:*

Condition 1: *$p(\tau;b_1)$ and $q(\tau;b_2)$ are both ODE derivative polynomials, or both ODE solution polynomials.*

Condition 2: *the jth approximate derivatives of $p(\tau;b_1)$ and $q(\tau;b_2)$ are respectively computed using polynomials $\nu(\tau)$ and $\upsilon(\tau)$ that satisfy the conditions of Lemma 3.4.1 for $j = 0,\dots,\gamma-1$.*

Condition 3: *$b_1^* = b_2$ or both $p(\tau;b_1)$ and $q(\tau;b_2)$ do not depend on the expansion point.*

Then the polynomials $p(\tau;b_1)$ and $q(\tau;b_2)$ are conjugate.

Proof. If $p(\tau;b_1)$ and $q(\tau;b_2)$ do not depend on the expansion point, then we take $b_1 = b_2 = 0$ through this proof. Using $b_2 = b_1^*$, we may write

$$p(\tau_j;b_1) = \sum_{j=0}^{\gamma} \frac{a_j(b_1)(t-b_1)^j}{j!} \tag{3.17}$$

$$q(\tau_j;b_2) = \sum_{j=0}^{\gamma} \frac{d_j(b_1^*)(t-b_1^*)^j}{j!} \tag{3.18}$$

Combining the result from 3.4.1 and using all three conditions it follows that the approximate derivatives $a_j(b_1)$ and $b_j(b_1^*)$ satisfy

$$a_j(b_1)^* = d_j(b_1^*)$$

Therefore,

$$p_2(\tau^*;b_2) = \sum_{j=0}^{\gamma} \frac{a_j(b_1)^*(\tau^*-b_1^*)^j}{j!} = \left(\sum_{j=0}^{\gamma} \frac{a_j(b_1)(t-b_1)^j}{j!}\right)^* = p_1(\tau;b_1)^*.$$

□

3.4.3 Spectral Accuracy for Certain ODE Polynomials

Let $p(\tau;b)$ be an ODE polynomial constructed from an ODE dataset $D(r,s)$. For sufficiently small r, the truncation error of $p(\tau,b)$ may decay exponential with respect to the polynomial's degree. In this subsection, we prove this result for any ODE polynomial

$$p(\tau;b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau-b)^j}{j!},$$

with derivative approximations $a_j(b)$ that are formed by:

1. Differentiating Lagrange interpolating polynomials that approximate $F(t, y(t))$.

2. Differentiating interpolating polynomials of degree g that approximate $y(t)$ and pass through solution values, or through both solution and derivative values at each node (i.e. the polynomial cannot pass through a derivative value f_j without also passing through the corresponding function value y_j).

Under these assumptions these exists a sufficiently small r, such that, for any fixed τ

$$|p(\tau; b) - y(t(\tau))| = \mathcal{O}(\gamma^g) \qquad \text{where} \qquad \gamma < 1.$$

This result holds irregardless of the locations for the temporal nodes used to form the polynomials for computing $a_j(b)$.

Our proof proceeds as follows. First we introduce *stadium regions* in the complex plane along with the Hermite integral formula for expressing the error of an interpolating polynomial as a contour integral in the complex plane. We then use the Hermite integral formala to prove spectral convergence for the polynomial derivative approximations from computing the terms $a_j(b)$, and for the the truncated Taylor polynomial approximating initial value solution $y(t)$. Finally we combine these two results to show exponential convergence for an ODE polynomial. Our proof is similar to existing proofs demonstrating spectral convergence of polynomials approximations (See for example [95]).

Definition 3.4.2. *A* closed stadium *of width h, radius r, and left endpoint s is the set of points whose distance from the real interval $[s,\ s+h]$ is less then or equal to r. We can formally define it as*

$$S_r(s, h) = \{z \in \mathbb{C} : |z - s + t| \leq r \text{ for } t \in [0, h]\} \tag{3.19}$$

This region is the union of two circles of radius r located at s and $s+h$, and a rectangle with vertices at $s \pm ir$ and $(s+h) \pm ir$.

Definition 3.4.3. *A* closed stadium contour *of width h, radius r, and left endpoint s is the curve running counter-clockwise along the boundary of a closed stadium $S_r(s, h)$. We will use the symbol $\gamma_r(s, h)$ to represent curve.*

Theorem 3.4.3 (Hermite Integral Formula)**.** *Let $z_1, \ldots, z_q$ be q distinct points in the complex plane, and suppose that $g(z)$ is an analytic function in a region Ω that contains these points. Next, let $\{\alpha_j\}_{j=1}^{q}$ be a set of positive integers and let $p(z)$ be an interpolating polynomial of degree $d = -1 + \sum_{j=1}^{q} \alpha_j$ that approximates $g(z)$ by satisfying the conditions*

$$p^{(k)}(z_j) = g^{(k)}(z_j) \qquad \text{for} \qquad \begin{array}{l} j = 1, \ldots, q, \\ k = 1, \ldots, \alpha_j. \end{array}$$

If Γ is a contour inside Ω that also encloses the points $z_0, \ldots z_q$ in the positive direction, then the error of the interpolating polynomial at any point inside Γ can be expressed as

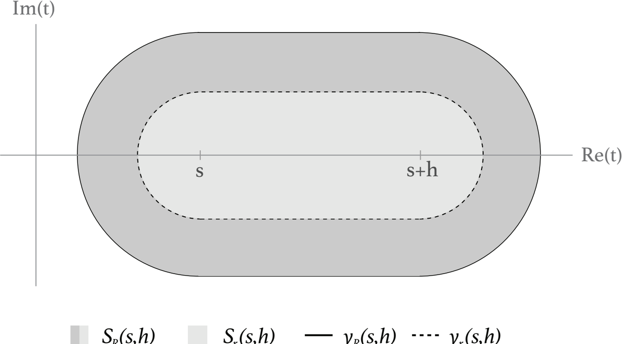
$$e(z) = g(z) - p(z) = \frac{1}{2\pi i}\oint_{\Gamma} \frac{w(z)g(t)}{w(t)(t-z)}dt \qquad \textit{for} \qquad w(z) = \prod_{j=1}^{q}(z-z_j)^{\alpha_j}. \tag{3.20}$$

Proof. This is a well-known result originally derived by Hermite in 1877 [45]. For current references see [95, 27, 86]. □

Corollary 3.4.3.1 (Spectral Accuracy of Polynomial Approximations)**.** *Let $g(z)$ be an analytic function in the closed stadium $S_R(s,h)$ and let $p(z)$ be an interpolating polynomial of degree $d = -1 + \sum_{j=1}^{q} \alpha_j$ that approximates $g(z)$ by satisfying the conditions*

$$p^{(k)}(z_j) = g^{(k)}(z_j) \qquad \textit{for} \qquad \begin{array}{l} j = 1, \ldots, q, \\ k = 1, \ldots, \alpha_j. \end{array}$$

Moreover, suppose that each node z_j lies inside the closed stadium $S_r(s,h)$ for $r < R$.



If we parametrize h as $h = r\alpha$ (where α is a positive constant) then, it follows that for any $z \in S_r(s,h)$ the error of the polynomial approximation satisfies

$$|e^{(\nu)}(z)| = |g^{(\nu)}(z) - p^{(\nu)}(z)| = \mathcal{O}\left(\frac{c(\alpha)^d r^{d+1-\nu}}{R^{d+1}}\right) \quad \textit{as} \quad r \to 0,\ R \to \infty,\ p \to \infty, \tag{3.21}$$

where $c(\alpha) < (1+\epsilon)(2+\alpha)$ for any $\epsilon > 0$. Finally, by fixing $\epsilon = \epsilon_0 > 0$, and choosing r so that

$$r < \frac{(1+\epsilon_0)(2+\alpha)}{R}$$

It follows that $|e^{(\nu)}(z)| = \mathcal{O}\left(\gamma^d\right)$ for $\gamma < 1$.

Proof. Since $g(z)$ is analytic inside $S_R(s,h)$ there exists a positive constant M_1 such that $|g(z)| < M_1$ for all $z \in S_R(s,h)$. Furthermore, $|t-z| > R-r$ for all $z \in S_r(s,h)$, $t \in \gamma_R(s,h)$. We may therefore bound the contour integral (3.20) as

$$|e(z)| \leq \frac{M_1}{R-r} \oint_{\Gamma_1} \frac{|w(z)|}{|w(t)|} dt \qquad \text{where} \qquad \Gamma_1 = \gamma_R(s,h).$$

We may bound the integrand using the inequalities

$$|(z-z_j)| \leq 2r + h \text{ for any } z \in S_R(p,h) \quad \text{and} \quad |(t-z_j)| \geq R - r \text{ for any } t \in \gamma_R(p,h).$$

Finally, letting $h = r\alpha$ we obtain

$$|e(z)| \leq \frac{M_1(2r + r\alpha)^d}{(R-r)^{d+1}} \oint_{\Gamma_1} dt = \frac{M_1(2\pi r + 2r\alpha)(2r + r\alpha)^d}{(R-r)^{d+1}} = \mathcal{O}\left(\frac{(2+\alpha)^d r^{d+1}}{R^{d+1}}\right) \tag{3.22}$$

as $r \to 0$, $R \to \infty$, $p \to \infty$. Next, to prove spectral convergence for polynomial derivative approximations we express the derivatives of $e(z)$ for $z \in S_r(p,h)$ using the Cauchy integral formula:

$$e^{(\nu)}(z) = \frac{\nu!}{2\pi i} \oint_{\Gamma_2} \frac{e(t)}{(t-z)^{\nu+1}} dt \quad \text{where} \quad \Gamma_2 = \gamma_{\tilde{r}}(p,h) \text{ for any } \tilde{r} \text{ such that } r < \tilde{r} < R.$$

For any t along the contour Γ_2, it follows from (3.22) that

$$|e(t)| \leq M_2 \qquad \text{where} \qquad M_2 = \frac{M_1(2\pi\tilde{r} + 2\tilde{r}\alpha)(2\tilde{r} + \tilde{r}\alpha)^d}{(R-\tilde{r})^{d+1}}.$$

Using the bound for $e(z)$ with the inequality $(t-z) \leq (\tilde{r} - r)$ for $t \in \Gamma_2$ and $z \in S_r(p,h)$, we obtain

$$|e^{(\nu)}(z)| \leq \frac{\nu! M_2}{2\pi(\tilde{r}-r)^{\nu+1}} \oint_{\Gamma_2} dt = \frac{\nu! M_2(2\pi\tilde{r} + 2\tilde{r}\alpha)}{2\pi(\tilde{r}-r)^{\nu+1}}$$

We may pick $\tilde{r} = (1+\epsilon)r$ for any $\epsilon > 0$ that satisfies $(1+\epsilon)r < R$. We are interested in the error as $r \to 0$, therefore we may pick $\epsilon = \epsilon_0$ where ϵ_0 is any positive constant, since for sufficiently small r it will hold that $\tilde{r} < R$. This choice leads to

$$|e^{(\nu)}(z)| = \mathcal{O}\left(\frac{\nu! M_2(1+\epsilon_0)(2\pi r + 2r\alpha)}{2\pi(\epsilon r)^{\nu+1}}\right) \qquad \text{as} \qquad r \to 0$$
$$M_2 = \frac{M_1(1+\epsilon_0)^d(2\pi r + 2r\alpha)(2r + r\alpha)^d}{(R-r)^{d+1}}.$$

By collecting terms that do not depend on d, we may write this more compactly as

$$|e^{(\nu)}(z)| = \mathcal{O}\left(\frac{c(\alpha)^d r^{d+1-\nu}}{R^{d+1}}\right) \qquad r \to 0,\ R \to \infty,\ p \to \infty, \tag{3.23}$$

where $c(\alpha) = (1+\epsilon_0)(2+\alpha)$ and ϵ_0 is an arbitrary positive constant. Finally if we choose $r = r_0$ where

$$r_0 < \frac{R}{(1+\epsilon_0)(1+\alpha)}$$

then, it follows that for any $z \in S_r(p,h)$ the error of the polynomial approximation converges spectrally in d such that

$$|e^{(\nu)}(z)| = \mathcal{O}\left(\gamma(r_0)^d\right) \qquad \text{for} \qquad \gamma(r_0) = \frac{c(\alpha)^d r_0^{d+1-\nu}}{R^{d+1}} < 1$$

□

To simplify our proof for spectral convergence of ODE polynomials, we impose additional constraints on the polynomial's expansion point b, the temporal nodes of the ODE dataset, and the region where we prove exponential convergence. Specifically, we require that:

1. The expansion point b and the ODE dataset temporal nodes τ_j lie within the unit circle, such that $|b| \leq 1$ and $|\tau_j| \leq 1$

2. We only prove exponential convergence for $p(\tau; b)$ in the stadium region $\tau \in S_1(0, \alpha)$.

The first restriction, is merely a normalization. Given any ODE dataset we can rescale the initial value problem via $t \to vt$, so that (1) is satisfied. The second requirement can also be relaxed by modifying Corollary (3.4.3.1) to show exponential convergence in more general regions. However, this restriction is suitable for the purposes of time-stepping.

Theorem 3.4.4 (Spectral Convergence for Certain ODE Polynomials)**.** *Suppose that the initial value problem (3.1) has a right-hand-side that is analytic in each of its arguments around the point $t = s$. Next, let*

$$p(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!}, \qquad |b| \leq 1$$

be an ODE polynomial of degree g constructed from the ODE dataset $D(r,s) = \{\tau_j, y_j, rf_j\}_{j=1}^{w}$ where $|\tau_j| \leq 1$. Additionally, suppose that every derivative approximation $a_j(b)$ is computed by differentiating a polynomial $q_j(\tau)$ of degree g that is constructed in either of the following ways:

1. *The polynomial $q_j(\tau)$ approximates $y(t(\tau))$ by satisfying any subset of the constraints*

$$q_j(\tau_k) = y_k \qquad \text{or} \qquad \begin{cases} q_j(\tau_k) = y_k \\ q_j'(\tau_k) = rf_k \end{cases} \qquad k = 1, \ldots, w,$$

that lead to a degree g polynomial.

2. *The polynomial $q_j(\tau)$ approximates $rF(t(\tau), y(t(\tau)))$ by satisfying subset of the constraints:*

$$q_j(\tau_k) = rf_k \qquad\qquad k = 1, \ldots, w,$$

that lead to a degree g polynomial.

Then, for all α these exists a sufficiently small r such that

$$|p(\tau; b) - y(t(\tau))| = \mathcal{O}\left(\gamma^{g}\right) \qquad \textit{where} \qquad \gamma < 1.$$

for any $\tau \in S_1(0, \alpha)$

Proof. Since the initial value problem satisfies the conditions of the Cauchy-Kovalevsky theorem (2.2.1), we know that there exists a sufficiently small R and h so that the solution $y(t)$ is analytic in the region $S_R(s, h)$. From corollary (3.4.3.1) it follows that there exists a sufficiently small $r < R$, where each of the interpolating polynomials $q_j(\tau)$ satisfy

$$|y^{(\nu)}(t(\tau)) - q_j^{(\nu)}(\tau)| = \mathcal{O}\left(\gamma^{g+1}\right) \qquad \Longrightarrow \qquad |y^{(\nu)}(t(b)) - a_j(b)| = \mathcal{O}\left(\gamma^{g+1}\right)$$

for $\tau \in S_r(s, h)$. From Corollary (3.4.3.1) it follows that for the same r, the Taylor polynomial of degree g expanded at $t = s + rb$ satisfies

$$\left| y(r\tau + s) - \sum_{j=1}^{g} \frac{y^{(j)}(rb + s)(r\tau)^j}{j!} \right| = \mathcal{O}\left(\gamma^{g+1}\right)$$

for $\tau \in S_r(s, h)$. Together these two results imply that

$$|p(\tau; b) - y(t(\tau))| = \mathcal{O}\left(\gamma^{g+1}\right) \qquad \text{where} \qquad \gamma < 1.$$

for any $\tau \in S_1(0, \alpha)$. □

Remark: If the interpolating polynomial $q_j(\tau)$ are chosen of degree $d_j < g$, then it follows that

$$|p(\tau; b) - y(t(\tau))| = \mathcal{O}\left(\gamma^{\rho}\right),$$

where $\gamma < 1$ and $\rho = \min\left(g + 1, d_1, \ldots, d_g\right)$.

3.5 Expanding ODE Datasets

Suppose that we are provided with a nonempty ODE dataset $D(r, s)$. We can expand the dataset by appending new approximations obtained by forming ODE polynomials and

evaluating them at new time points. For example, suppose that $D(s,r)$ contains only one element such that

$$D(s,r) = \{(\tau_1,\ y_1,\ rf_1)\} \qquad \text{where} \qquad \tau_1 = 0,\ y_1 = y(s),\ f_1 = F(s, y_1). \tag{3.24}$$

We may form the ODE polynomial

$$p(\tau; b) = y_1 + \tau r f_1$$

and evaluate it at $\tau = 1$. If we append this new value to $D(r,s)$, we can now write

$$D(s,r) = \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{2} \quad \text{where} \quad \begin{cases} \tau_1 = 0, & y_1 = y(s), \qquad f_1 = f(s, y_1) \\ \tau_2 = 1, & y_2 = p(\tau_2; b), \quad f_2 = F(s + r\tau_2, y_2) \end{cases}.$$

Notice, that this particular operation is equivalent to taking a single step with the forward Euler method using the inital data $(\tau_1,\ y_1,\ rf_1)$. Even for a dataset with a single element, there are an unlimited number of ways to expand every dataset. In this section we will discuss two important ways to expand the dataset.

3.5.1 Implicitly Defined Values

Elements of an ODE dataset can be defined implicitly in terms of an ODE polynomial formed from the values. For example, consider the following dataset containing two elements:

$$D(s,r) = \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{2} \quad \text{where} \quad \begin{cases} \tau_1 = 0, & y_1 = y(s + r\tau_1), \quad f_1 = f(s + r\tau_1, y_1) \\ \tau_2 = 1, & y_2 = p(\tau_2; b), \qquad f_2 = f(s + r\tau_2, y_2) \end{cases},$$

and $p(\tau; b)$ is an ODE polynomial that is the implicitly defined in terms of the the dataset elements via

$$p(\tau; b) = y_1 + \tau r f_2.$$

To explicitly obtain the value y_2 we would need to solve the nonlinear system

$$y_2 = y_1 + \tau r f_2.$$

This operation is equivalent to taking a single step with the backwards-Euler method using the initial condition $(\tau_1,\ y_1,\ rf_1)$. Implicitly defined values can depend on other implicit values and on more than one known value. In each of these cases, we can always interpret implicitly defined values as a technique for expanding a smaller dataset.

3.5.2 Interpolated Values

Suppose that we are given a nonempty ODE dataset $D(r,s) = \{(\tau_j, y_j, f_j)\}_{j=1}^{w}$ and that we want to expand the dataset by taking weighted linear combinations of it original values. We can construct ODE solution polynomials to approximate $y(t)$ at new time points. Since each of the new solution values can be written as a linear combination of the original data elements, we may use them to form additional ODE solution polynomials. However, we cannot evaluate the derivative at these new points since the new derivative approximations cannot be written as linear combinations of the values originally in $D(r,s)$. If we want to approximate the derivative at new time points, we will need to construct and evaluate ODE derivative polynomials. These two special types of solution and derivative approximations will be called *interpolated values*; together they form *interpolated value sets.*

Definition 3.5.1. *An* interpolated value set $I(r,s)$ *is an ordered set of tuples* $I(r,s) = \{d_j\}_{j=1}^{w}$ *generated from an ODE dataset* $D(r,s)$*, where each tuple* d_j *contains a temporal node* $\tilde{\tau}$ *and either an interpolated solution* $\tilde{y}_j$ *or an interpolated derivative* $\tilde{f}_j$*. If the tuple* d_j *contains an interpolated solution value, it has the form*

$$d_j = (\tilde{\tau}_j,\ \tilde{y}_j,\ \emptyset) \qquad \textit{where} \qquad \tilde{y}_j \approx y(t(\tilde{\tau}_j))$$

and $t(\tau) = r\tau + s$*. If the tuple* d_j *contains an interpolated derivative value it has the form*

$$d_j = (\tilde{\tau}_j,\ \emptyset,\ r\tilde{f}_j) \qquad \textit{where} \qquad \tilde{f}_j \approx F(t(\tilde{\tau}_j), y(t(\tilde{\tau}_j)))$$

Each of the interpolated values must be computed via $\tilde{y}_j = p_j(\tau_j; b_j)$ *and* $\tilde{f}_j = \dot{p}_j(\tau_j; b_j)$ *where* $p_j(\tau; b)$ *are ODE polynomials and* $\dot{p}_j(\tau; b)$ *are ODE derivative polynomials formed using the data in* $D(r,s)$ *or* $I(r,s)$*.*

Definition 3.5.2 (Extending The Dataset Order Function)**.** *The function* order *can also map an interpolated value set* $I(r,s)$ *of size* w*, constructed from an ODE dataset* $D(r,s)$*, to an ordered set of positive integers such that*

$$\text{order}(I(r,s); x) = \{\rho_1,\ \ldots,\ \rho_w\}$$

where each ρ_j *corresponds to the order of accuracy of the solution approximations* $\tilde{y}_j$ *or the derivative approximations* $\tilde{f}_j$ *with respect to variable* x*. Note that the* order *function depends implicitly on the dataset* $D(r,s)$ *and on the ODE polynomials used to form the interpolated values.*

In summary, every interpolated value $\tilde{y}_j$ or $\tilde{f}_j$ can be written linear combination of the y_j and f_j values in generating the dataset. Therefore, the operation of computing interpolated values does not introduce new information, but it does provide a convenient way to expand the number of possible polynomial approximations that can be constructed from a fixed amount of data.

3.5.2.1 Order & Truncation Error for ODE Polynomials Constructed Using Interpolated Values

Let $I(r,s)$ be an interpolated value set constructed from an ODE dataset $D(r,s)$ and let $p(\tau;b)$ be an ODE polynomial of degree g, constructed from the values in $D(r,s)$ and $I(r,s)$, that approximates the function $v(\tau)$. The order of accuracy for $p(\tau;b)$, is defined identically to the order of accuracy for an ODE Polynomial constructed using only dataset elements. To determine the truncation error of $p(\tau;b)$, we will need to extend our definition to include interpolated values. The truncation error for $p(\tau;b)$, is given by

$$\mathrm{TE}(r;\tau) = \left|\tilde{p}\,(\tau;b) - v(\tau)\right|,$$

where $\tilde{p}(\tau;b)$ is the ODE polynomial we would obtain, had we replaced $D(r,s)$ with one containing the exact solution values and recomputed each of the interpolated value set using the exact dataset. Note that we do not replace the elements of $I(r,s)$ with exact solution and derivative values.

We can determine the order of the truncation error of $p(\tau;b)$ explicitly, if: 1) the construction strategy for forming $p(\tau;b)$ is known, and 2) the order of the truncation errors for the polynomials used to compute the interpolated value set $I(r,s)$ are known. If $p(\tau;b)$ is constructed using l interpolated values, then the order of the truncation error of $p(\tau;b)$ is given by

$$\rho = \min(g+1,\ \lambda_1,\ \ldots,\ \lambda_g,\ \omega_1,\ \ldots,\ \omega_l)$$

where λ_j are defined in subsection 3.4.1 and ω_j is the order of the truncation error for the ODE polynomial used to compute the jth interpolated value.

3.5.2.2 Computing Coefficients for ODE Polynomials Constructed Using Interpolated Values

To evaluate an ODE polynomial that was constructed using both dataset elements and interpolated values, we will require a modified procedure for obtaining the coefficients. First we must obtain coefficients for expressing the interpolated values in terms of the dataset elements and then we can use this result to obtain the coefficients of the ODE polynomial.

Let $D(r,s)$ be an ODE dataset of size w and let $I(r,s)$ be an interpolated value set of size l constructed from $D(r,s)$, where

$$D(r,s) = \{(\tau_j, y_j, rf_j)\}_{j=1}^{w} \qquad \text{and} \qquad I(r,s) = \{d_j\}_{j=1}^{l}.$$

Next, suppose that the ODE polynomial $p(\tau;b)$ is built using both values in $D(r,s)$ and $I(r,s)$. The procedure for evaluating $p(\tau;b)$ is described below.

1. **Expressing Interpolated Values in Terms of Dataset Values**

Let $\tilde{\mathbf{v}}$ be the interpolated value vector defined by

$$\tilde{\mathbf{v}}_j = \begin{cases} \tilde{y_j} = p_j(\tau_j; b) & \text{if the tuple } d_j \text{ contains an interpolated solution } \tilde{y}_j \\ r\tilde{f}_j = \dot{p}_j(\tau_j; b) & \text{if the tuple } d_j \text{ contains an interpolated derivative } r\tilde{f}_j \end{cases} \tag{3.25}$$

To compute $\tilde{\mathbf{v}}$, consider the data vector $\mathbf{d}$ and the augmented data vector $\widehat{\mathbf{d}}$ where

$$\mathbf{d} = [y_1,\ \ldots,\ y_w,\ rf_1,\ \ldots,\ rf_w]^T \tag{3.26}$$

$$\widehat{\mathbf{d}} = [y_1,\ \ldots,\ y_w,\ rf_1,\ \ldots,\ rf_w,\ \tilde{\mathbf{v}}_1,\ \ldots,\ \tilde{\mathbf{v}}_l]^T. \tag{3.27}$$

Then, each element of the interpolated value vector can be computed as

$$\tilde{\mathbf{v}}_j = \widehat{\mathbf{w}}(j) \cdot \widehat{\mathbf{d}}, \qquad \widehat{\mathbf{w}}(j) \in \mathbb{C}^{2w+l}.$$

If $\tilde{\mathbf{v}}_j$ is an interpolated solution value then, the weight vector $\mathbf{w}(j)$ can be determined using the procedure describe in Section 3.2.2. If $\tilde{\mathbf{v}}_j$ an interpolated derivative value, then the weight vector $\mathbf{w}(j)$ can be determined using the procedure describe in 3.3.2. The interpolated values vector can be expressed as

$$\tilde{\mathbf{v}} = \widehat{\mathbf{W}}\widehat{\mathbf{d}} \qquad \text{where} \qquad \widehat{\mathbf{W}} = [w(1),\ w(2),\ \ldots,\ w(l)]^T$$

To obtain an explicit formula for $\tilde{\mathbf{v}}$, we can rewrite the formula for $\tilde{\mathbf{v}}$ as

$$\tilde{\mathbf{v}} = \mathbf{A}\mathbf{d} + \mathbf{B}\tilde{\mathbf{v}} \qquad \mathbf{A} \in \mathbb{C}^{l\times 2w},\ \mathbf{B} \in \mathbb{C}^{l\times l}$$

where $\mathbf{A}$ is formed from the first $2w$ columns of $\widehat{\mathbf{W}}$, and $\widehat{\mathbf{B}}$ is formed from the the last l columns of $\widehat{\mathbf{W}}$. Then the interpolated values can be explicitly obtained from the dataset elements via

$$\tilde{\mathbf{v}} = \mathbf{W}\mathbf{d} \qquad \text{for} \qquad \mathbf{W} = (\mathbf{I} - \mathbf{B})^{-1}\mathbf{A}. \tag{3.28}$$

2. **Evaluating The ODE polynomial**

   We may express the ODE derivative polynomial $p(\tau; b)$ as

$$p(\tau; b) = \widehat{\mathbf{w}}(\tau; b) \cdot \widehat{\mathbf{d}}$$

   where $\widehat{\mathbf{d}}$ is the augmented data vector (3.27) and the weight vector $\widehat{\mathbf{w}}(\tau; b)$ can be computed using the procedure in Subsection 3.2.2 or 3.3.2 depending on whether $p(\tau; b)$ is an ODE solution polynomial or an ODE derivative polynomial. We can expand this formula as

$$p(\tau; b) = \mathbf{a}(\tau; b) \cdot \mathbf{d} + \mathbf{b}(\tau; b) \cdot \tilde{\mathbf{v}}$$

   where $\mathbf{a}$ contains the first $2w$ elements of $\widehat{\mathbf{w}}$, $\mathbf{b}$ contains the first last l elements of $\widehat{\mathbf{w}}$, $\mathbf{d}$ is the data vector (3.26), and $\tilde{\mathbf{v}}$ is the interpolated value vector (3.25). Finally, using (3.28) we can expression $p(\tau; b)$ in terms of the dataset elements

$$p(\tau; b) = \mathbf{w}(\tau; b) \cdot \mathbf{d} \qquad \text{where} \qquad \mathbf{w}(\tau; b) = \mathbf{a}(\tau; b) + \left(\mathbf{b}(\tau; b)^T (\mathbf{I} - \mathbf{B})^{-1}\mathbf{A}\right)^T.$$

3.6 Diagrams for Geometrically Interpreting ODE Polynomials

The formulaic descriptions for ODE polynomials do not convey any underlying geometric structures such as the active nodes locations and data types. To allow for geometric presentation of ODE polynomials, we introduce the two stencils and one diagram. We briefly describe each illustration, before providing a detailed discussion in the following sections.

1. A **node stencil** illustrates active nodes for an ODE Polynomial in the complex plane.

2. A **polynomial diagram** can be used to visualize the expansion point b along with the interpolating polynomials used to compute the ODE Polynomial derivatives $a_j(b)$.

3. An **expansion point stencil** shows the expansion point b. For Adams ODE polynomials it can modified to show the corresponding integration paths.

3.6.1 The Node Stencil

The node stencil for an ODE polynomial $p(\tau; b)$ shows the active and inactive temporal nodes of the ODE dataset $D(r, s)$ used to construct $p(\tau; b)$. A node stencil cannot be used to determine whether solution or derivative data at an active node was used to form $p(\tau; b)$. Nevertheless, these stencils are useful for comparing and classifying ODE polynomials that are built from different subsets of the same ODE dataset.

If the dataset $D(r, s)$ contains complex-valued nodes, then each stencil shows the complex τ-plane where the global time $t(\tau) = r\tau + s$. If the temporal nodes are all real-valued, then only the real line is drawn. In both cases, the origin marks the point $\tau = 0$. We color the temporal nodes depending on whether the node is active or inactive and use different markers depending on the type of data that is being used at each time point. For the moment, we only need to distinguish between values in a dataset and interpolated values using the following markers.

Underlying Data	Active	Inactive
Dataset Value	● (black)	● (gray)
Interpolated Value	⊕ (black)	⊕ (gray)

In Figure 5.2 we show an example node stencils for the empty ODE polynomial $p(t; b) = 0$ constructed from an ODE dataset with complex nodes and one with real nodes.

3.6.2 The ODE Polynomial Diagram

An ODE polynomial diagram contains geometrical illustrations of the the interpolating polynomials used to form an ODE polynomial. The diagram for an ODE polynomial of degree

(a) Example Complex Node Stencil **(b) Example Real Node Stencil**

Figure 3.2: Example node stencils for the ODE polynomial $p(\tau; b) = 0$ constructed from **(a)** an ODE dataset with the three complex temporal nodes $\{\tau_j\} = \{-i, 0, i\}$ and one interpolated value at $\tau = 1/2$, and **(b)** an ODE dataset with the three real temporal nodes $\{\tau_j\} = \{-1, 0, 1\}$ and one interpolated value at $\tau = 3/2$.

g,

$$p(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!},$$

consists of $g + 1$ stencils, where the jth stencil describes the interpolating polynomials used to compute the approximate derivative $a_j(b)$. Optionally, each stencil may also show the location of the expansion point b.

Each stencil in an ODE polynomial diagram shows all the temporal nodes of the underlying ODE dataset in the complex τ-plane. The nodes in the jth stencil are labeled differently, depending on whether the interpolating polynomial for computing $a_j(b)$ passes through solution, derivative, or both solution and derivative data at each node. The markers used to designate each of these cases are illustrated below; an empty marker indicates that the particular node type will not be shown.

Data Type	Dateset	Interpolated
Active Solution Value	■	⊞
Active Derivative Value	●	⊕
Active Solution & Derivative	◖■	⊕⊞
Inactive Data	•	

Optional Markers	
Expansion Point	•

It is possible for solution values, derivative values, and interpolated derivative values to overlap. However, we avoid introducing additional makers to denote each of these cases since we will not be requiring them.

For certain families of ODE polynomials, including the BDF, GBDF and Adams families, it is possible to obtain a more concise diagram. We will describe each of these cases below.

1. **BDF and GBDF**: This family of ODE polynomials can be expressed as

$$\begin{aligned} &\text{ODE solution polynomial} && p(\tau;b) = H(\tau)\ \forall b \\ &\text{ODE derivative polynomial} && \dot{p}(\tau;b) = H'(\tau)\ \forall b \end{aligned}$$

   where $H(\tau)$ is an interpolating polynomial passing through solution and derivative values. The diagram for $p(\tau;b)$ contains a single stencil that describes the interpolating polynomial $H(\tau)$, and should never show the expansion point.

2. **Adams**: This family of ODE polynomials can be expressed as

$$\begin{aligned} &\text{ODE solution polynomial} && p(\tau;b) = L_y(b) + \int_b^\tau L_F(s)ds \\ &\text{ODE derivative polynomial} && \dot{p}(\tau;b) = L_F(\tau)\ \forall b \end{aligned}$$

   where $L_y(\tau)$ and $L_F(\tau)$ are Lagrange interpolating polynomials that respectively pass through at least one solution value and at least one derivative value. The diagram for $p(\tau;b)$ contains two stencils; one that describes the polynomial $L_y(\tau)$ and one that describes the polynomial $L_F(\tau)$. Similarly, the diagram for $\dot{p}(\tau;b)$ contains one stencil that describes the polynomial $L_F(\tau)$.

3.6.3 The Expansion-Point Stencil

An expansion-point stencil for an ODE polynomial $p(\tau;b)$ shows the temporal nodes of the underlying ODE dataset, along with the expansion point b. The markers for this diagram are described as follows:

Temporal Node ● Expansion Point ●

Expansion-point stencils do not reveal the derivative approximations used to form $p(\tau;b)$, and will be used to illustrate different endpoint choices for polynomials constructed from the same derivative approximations.

If $p(\tau;b)$ is an Adams ODE solution polynomial, and the evaluation point is fixed, then the expansion point b can be interpreted as a left integration bound. The corresponding integration path is the straight line connecting the expansion point to the evaluation point. For these polynomials, it can be instructive to show the show integration path and endpoints in the expansion-point stencil. The markers for this modified diagram are described as follows:

Temporal Node ● Integration Endpoint ● Integration Path ⟶

3.7 Example ODE Datasets and ODE Polynomials

Now that we have introduced ODE datasets and ODE polynomials, we present a collection of examples to familiarize the reader with our notation and diagrams. The examples discuss the initialization of an ODE dataset and the construction of ODE polynomials from datasets with real nodes and datasets with imaginary nodes.

Example 3.7.1 (Populating an ODE dataset)**.** *If the solution to the initial value problem (3.1) cannot be determined explicitly, then we require a numerical method to initially populate an ODE dataset. Consider the generic ODE dataset*

$$D(r, t_0) = \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{w}$$

and suppose that we use one step of forward Euler to obtain each solution value y_j. This choice will lead to the approximations

$$y_j = y_0 + r\tau_j f(t_0, y_0)$$

where each element is order one accurate with respect to the scaling factor r. In general, we could use a Runge-Kutta method of order ρ_j to compute the solution value y_j. If the stepsize of each Runge-Kutta method scales proportionally to r, then the order of accuracy of each solution value with respect to r is given by

$$\{\rho_1, \ldots, \rho_w\} = \text{order}(D; r)$$

In the subsequent three examples, we assume that we are provided with exact solution data for the initial value problem (3.1) at the three real time-points $t = t_0$, $t = t_0 - r$, and $t = t_0 + r$. The corresponding ODE dataset centered at $t = t_0$ is

$$D(r, t_0) = \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{3}, \quad \text{where} \quad \tau_j = j - 2, \quad y_j = y(r\tau_j + t_n). \tag{3.29}$$

We will use this data to construct ODE solution polynomials

$$p(\tau; b) = \sum_{j=1}^{g} \frac{a_j(b)(\tau - b)^j}{j!}.$$

In Figure 3.3 we present the ODE polynomial diagrams and active node stencils for the ODE polynomial discussed in the examples.

Example 3.7.2 (Expressing a Lagrange Polynomial as an ODE Polynomial)**.** *Suppose that we are provided with the dataset (3.29). To construct an ODE solution polynomial $p(\tau; b)$ we must choose a degree g, an expansion point b, and a set of polynomials to compute the approximate derivatives $a_j(b)$. Suppose we select $g = 2$, $b = 0$ and choose to compute the approximate derivatives by differentiating the Lagrange polynomial*

$$L(\tau) = \left[\frac{\tau(\tau - 1)}{2}\right] y_1 - \left[\frac{(\tau + 1)(\tau - 1)}{1}\right] y_2 + \left[\frac{\tau(\tau + 1)}{2}\right] y_3$$

that passes through each solution value. This leads to the well-known finite difference approximations

$$a_0(0) = y_2 \qquad a_1(0) = \frac{y_3 - y_1}{2} \qquad a_2(0) = y_1 - 2y_2 + y_3$$

The cooresponding ODE solution polynomial is

$$p(\tau; 0) = y_2 + \left[\frac{y_3 - y_1}{2}\right]\tau + [y_1 - 2y_2 + y_3]\frac{\tau^2}{2}$$

Note that $p(\tau; 0) = L(\tau)$, hence we have simply expanded the Lagrange polynomial $L(\tau)$ at $\tau = 0$.

Example 3.7.3 (A Example BDF and Adams ODE solution polynomial)**.** *Suppose that we are provided with the ODE dataset (3.29). To build a BDF ODE solution polynomial $p(\tau; b) = H(\tau)$ we will choose $H(\tau)$ to be the interpolating polynomial that passes through solution values at $\tau = \tau_1$, τ_2 and derivative values at $\tau = \tau_3$. This leads to*

$$p(\tau; b) = \left[\frac{\tau^2 - 2\tau}{3}\right] y_1 + \left[\frac{-\tau^2 + 2\tau + 3}{3}\right] y_2 + \left[\frac{\tau^2}{3} + \frac{\tau}{3}\right] f_3. \tag{3.30}$$

To build an Adams ODE solution polynomial $p(\tau; b) = L_y(b) + \int_b^\tau L_F(s)ds$ we choose the Lagrange polynomials $L_y(\tau)$ and $L_F(\tau)$ so that they respectively passes through each solution and derivative value:

$$L_y(\tau) = \left[\frac{\tau(\tau-1)}{2}\right] y_1 - \left[\frac{(\tau+1)(\tau-1)}{1}\right] y_2 + \left[\frac{\tau(\tau+1)}{2}\right] y_3$$
$$L_F(\tau) = \left[\frac{\tau(\tau-1)}{2}\right] rf_1 - \left[\frac{(\tau+1)(\tau-1)}{1}\right] rf_2 + \left[\frac{\tau(\tau+1)}{2}\right] rf_3$$

The parameter b is free, however if $b \notin \{-1, 0, 1\}$ then the order of $p(\tau; b)$ will be reduce by one since $L_y(b)$ is no longer exact. We may illustrate these "optimal" choices for b using the following three endpoint diagrams

b = -1

b = 0

b = 1

If we pick pick $b = 0$, then the ODE solution polynomial can be written as

$$p(\tau; b) = y_2 + r\int_0^\tau \left[\frac{s(s-1)}{2}\right] f_1 - \left[\frac{(s+1)(s-1)}{1}\right] f_2 + \left[\frac{s(s+1)}{2}\right] f_3\, ds$$

Example 3.7.4 (A BDF ODE Solution Polynomial with Interpolated Values)**.** *Suppose that we are provided with the ODE dataset (3.29) and suppose we want to build a BDF ODE solution polynomial that is formed using both data values and interpolated values. First we need to choose a set of temporal nodes for the interpolated values along with a set of ODE polynomials to populate the interpolated value set* $I(r, s)$*. For simplicity we will only compute a single interpolated derivative using an ODE derivative polynomial* $\dot{p}(\tau; b)$*. The temporal location of this interpolated value will be at* $\tau = 1$ *(in global time this is* $t = t_0 + r$*).*

In the same spirit as example (3.7.2) we compute the approximate derivatives of $\dot{p}(\tau; b)$ *using a Lagrange polynomial that passes through the dataset derivate values* rf_1 *and* rf_2*. This leads to the ODE derivative polynomial*

$$\dot{p}(\tau; 0) = rf_2 + [rf_2 - rf_1]\,\tau.$$

The interpolated value set is given by

$$I(r, s) = \{(\tilde{\tau}_1, \emptyset, \tilde{f}_1)\}$$

where $\tilde{\tau}_1 = 1$*,* $\tilde{f}_1 = \dot{p}(\tau_1; 0)$*. To build a BDF ODE solution polynomial* $p(\tau; b) = H(\tau)$ *we will choose* $H(\tau)$ *to be the interpolating polynomial that passes through solution values at* $\tau = \tau_1$*,* τ_2 *and interpolated derivative value at* $\tau = \tilde{\tau}_1 = \tau_3$*. This leads to the ODE solution polynomial*

$$p(\tau; b) = \left[\frac{x^2 - 2x}{3}\right] y_1 + \left[\frac{-x^2 + 2x + 3}{3}\right] y_2 + \left[\frac{x^2}{3} + \frac{x}{3}\right] \tilde{f}_1 \quad \forall\ b.$$

We can easily express this ODE Polynomial using only the original data elements as

$$p(\tau; b) = \left[\frac{x^2 - 2x}{3}\right] y_1 + \left[\frac{-x^2 + 2x + 3}{3}\right] y_2 + \left[\frac{x^2}{3} + \frac{x}{3}\right] (3rf_2 - 2rf_1)\,.$$

In our final example, we assume that we are provided with exact solution data for the initial value problem (3.1) at the three imaginary time-points $t = t_0$, $t = t_0 - ir$, and $t = t_0 + ir$. The corresponding ODE dataset centered at $t = t_0$ is

$$D(r, t_0) = \{(\tau_j,\ y_j,\ rf_j)\}_{j=1}^{3}\,, \quad \text{where} \quad \tau_j = i(j-2), \quad y_j = y(r\tau_j + t_0). \tag{3.31}$$

In Figure 3.4 we present the ODE polynomial diagrams and active node stencils for the corresponding ODE polynomial.

Example 3.7.5 (Three ODE Polynomial with imaginary nodes)**.** *Suppose that we are provided with the ODE dataset (3.31). Similar to example (3.7.2) we choose an ODE solution polynomial with degree* $g = 2$*,* $b = 0$*, and with approximate derivatives computed by differentiating the Lagrange polynomial*

$$L(\tau) = \left[\frac{-\tau(\tau - i)}{2}\right] y_1 - \left[\frac{(\tau + i)(\tau - i)}{1}\right] y_2 + \left[\frac{-\tau(\tau + i)}{2}\right] y_3$$

that passes through each solution value. This leads to the complex finite difference approximations

$$a_0(0) = y_2 \qquad a_1(0) = \frac{iy_1 - iy_3}{2} \qquad a_2(0) = -y_1 + 2y_2 - y_3$$

The corresponding ODE solution polynomial is

$$p(\tau; 0) = y_2 + \left[\frac{iy_1 - iy_3}{2}\right]\tau + [y_1 - 2y_2 - y_3]\frac{\tau^2}{2}$$

Again it follows that $p(\tau; 0) = L(\tau)$.

To build an Adams ODE solution polynomial $p(\tau; b) = L_y(b) + \int_b^\tau L_F(s)ds$ *we choose the Lagrange polynomials* $L_y(\tau)$ *and* $L_F(\tau)$ *so that they respectively passes through each solution and derivative value:*

$$L_y(\tau) = \left[\frac{-\tau(\tau - i)}{2}\right] y_1 - \left[\frac{(\tau + i)(\tau - i)}{1}\right] y_2 + \left[\frac{-\tau(\tau + i)}{2}\right] y_3$$
$$L_F(\tau) = \left[\frac{-\tau(\tau - i)}{2}\right] rf_1 - \left[\frac{(\tau + i)(\tau - i)}{1}\right] rf_2 + \left[\frac{-\tau(\tau + i)}{2}\right] rf_3$$

The parameter b *is free, however if* $b \notin \{-i, 0, i\}$ *then the order of* $p(\tau; b)$ *will be reduce by one since* $L_y(b)$ *is no longer exact. If we pick pick* $b = 0$, *then the ODE solution polynomial can be written as*

$$p(\tau; b) = y_2 + r\int_0^\tau \left[\frac{-s(s - i)}{2}\right] f_1 - \left[\frac{(s + i)(s - i)}{1}\right] f_2 + \left[\frac{-s(s + i)}{2}\right] f_3\, ds$$

To build a GBDF ODE solution polynomial $p(\tau; b) = H(\tau)$ *we will choose* $H(\tau)$ *to be the interpolating polynomial that passes through solution values at* $\tau = \tau_2$, *and derivative values at* $\tau = \tau_1$ *and* $\tau = \tau_3$. *This leads to the ODE solution polynomial*

$$p(\tau; b) = \left[\frac{i\tau^2 + 2\tau}{4}\right] f_1 + y_2 + \left[\frac{-i\tau^2 + 2\tau}{4}\right] f_3. \tag{3.32}$$

3.8 Alterative Polynomial Formulations

The ODE polynomials presented in this chapter do not not encompass every possible approximation that can be constructed from an ODE dataset. Instead, our definition is an attempt to balance generality with practicality. Allowing for more generality decreases our ability to construct polynomials using intuition, and decreases our understanding of their underlying geometric properties. Nevertheless, this chapter could be rewritten using a different definition for the ODE polynomial. So long as the new approximations possess the properties of *order of accuracy* and *truncation error*, then many parts of the following Chapters could remain identical. We close this chapter by providing an additional “polynomial free” definition.

Family	Polynomial Diagram	Active Value Sets	Active Node Stencils
Lagrange	p	p $\{y_1, y_2, f_3\}$	
BDF	H_y	H_y $\{y_1, y_2, f_3\}$	
Adams	L_y	L_y $\{y_1, y_2, y_3\}$	
	L_F	L_F $\{f_1, f_2, f_3\}$	
BDF	H_y	H_y $\{y_1, y_2, \tilde{f}_3\}$	
	$\dot{p}$	$\dot{p}$ $\{f_1, f_2\}$	

Figure 3.3: A collection of four ODE solution polynomials constructed in examples 3.7.2, 3.7.3, and 3.7.4 from an ODE dataset with nodes $\tau_1 = -1$, $\tau_2 = 0$, and $\tau_3 = 1$. For each polynomial, the corresponding diagram, active value set, and active node stencil are shown. For the BDF polynomial with one interpolated derivative, the diagram and active value set of the ODE derivative polynomial $\dot{p}(\tau, b)$ used to compute the interpolated derivative are also shown. The ordering reflects the ordering of the examples in this section.

3.8.1 Weighted Linear Approximations

A more general way to form approximations from an ODE dataset is to avoid polynomials altogether, in favor of weighted linear combinations of the dataset elements. We call such values *weighted linear approximations* (WLA).

Definition 3.8.1 (Weighted Linear Approximations)**.** *Let $D(r,s)$ be an ODE dataset of the form $\{(\tau_j, y_j, rf_j)\}_{j=1}^{q}$. A weighted linear approximation $v(\xi)$ is a weighted linear combination of the elements of $D(r,s)$ such that*

$$v(\xi) = \sum_{j=1}^{w} c_j y_j + d_j r f_j \qquad \text{where} \qquad v(\xi) \approx y(t(\xi)) \text{ or } v(\xi) \approx y'(t(\xi))$$

where $t(\tau) = r\tau + s$. The WLA $v(\xi)$ is order ρ accurate if

$$|v(\xi) - y(t(\xi))| = \mathcal{O}(r^{\rho+1}) \qquad \text{or} \qquad |v(\xi) - F(t(\xi),\ t(\xi)))| = \mathcal{O}(r^{\rho}).$$

Similarly, the truncation error of $v(\xi)$ is given by

$$TE(r) = |\tilde{v}(\xi) - y(t(\xi))| \qquad \text{where} \qquad \tilde{v}(\xi) = \sum_{j=1}^{w} c_j y(t(\xi)) + d_j r F(t(\xi), y(t(\xi))).$$

Family	Polynomial Diagram	Active Value Sets	Active Node Stencils
Lagrange	p	p $\{y_1, y_2, y_3\}$	
Adams	L_y L_F	L_y $\{y_1, y_2, y_3\}$ L_F $\{f_1, f_2, f_3\}$	
GBDF	H_y	H_y $\{f_1, y_2, f_3\}$	

Figure 3.4: A collection of three ODE solution polynomials constructed in example 3.7.5 from an ODE dataset with nodes $\tau_1 = -i$, $\tau_2 = 0$, and $\tau_3 = i$. For each polynomial, the corresponding diagram, active value set, and active node stencil are shown. The ordering reflects the ordering in example (3.7.5).

In the subsequent chapter, we could replace ODE polynomials with weighted linear approximations. The primary drawback of this approach is that we could no longer use our geometric initiation to select approximations. Instead we will have simply traded the large number of nonlinear order-conditions found in GLMs and Runge-Kutta methods for a large number of linear order-conditions. Though this could be interesting topic for future exploration, we will presently avoid any further discussion of weighted linear approximations.

Chapter 4

POLYNOMIAL TIME-INTEGRATORS

In this chapter we introduce a new framework for deriving polynomial-based time-integrators for solving systems of first-order ordinary differential equations. Our framework is founded on the ODE datasets and ODE polynomials that we defined in the previous chapter. These two objects allow us to develop a simple, yet flexible approach for constructing a variety of integrators including those with parallelism and high-orders of accuracy.

We begin this chapter by formally defining a polynomial time-integrator and by introducing notation and parameters. We then use this notation to introduce polynomial block methods and polynomial general linear methods. Finally, we close the chapter by discussing order of accuracy and linear stability.

4.1 What is a Polynomial Time Integrator?

In short, a *polynomial time-integrator* is any time-integration method where each output and stage is computed by evaluating an ODE solution polynomial. Polynomial methods can be found within many classes of existing time integrators, including linear multistep methods, Runge-Kutta methods, and general linear methods. Adams-Bashforth and Adams-Moulton are polynomial methods, as are the Runge-Kutta midpoint and Heun methods.

Definition 4.1.1 (Polynomial Time Integrator)**.** *A* polynomial time-integrator *is any method where each stage and output is computed by evaluating an ODE solution polynomial* $p(\tau; b)$ *formed from an* ODE dataset $D(r, s)$ *or any interpolated value set* $I(r, s)$*, where* $D(r, s)$ *contains the method's inputs, outputs, stages and* $I(r, s)$ *is formed from* $D(r, s)$*.*

4.1.1 Parameters and Notation

During the time-step from t_n to $t_{n+1} = t_n + h$, a polynomial method accepts q inputs, computes s stages, and produces q outputs. We will denote these quantities as follows:

$$\left.\begin{array}{ll} \text{inputs} & y_j^{[n]} \\ \text{outputs} & y_j^{[n+1]} \end{array}\right\} \quad j = 1, \ldots, q,$$

$$\begin{array}{lll} \text{stages} & Y_j & \qquad j = 1, \ldots, s. \end{array}$$

Each input, output, and stage approximates the solution $y(t)$ at a specific time point. We will express the input times using the variables $t_j^{[n]}$ and the stage times using the variables

$T_j^{[n]}$ such that

$$y_j^{[n]} \approx y\left(t_j^{[n]}\right), \quad y_j^{[n+1]} \approx y\left(t_j^{[n]} + h\right), \quad \text{and} \quad Y_j \approx y\left(T_j^{[n]}\right).$$

For a classical time-integration method, the input times scale with the stepsize h. For a polynomial method the input times scale with the *radius* r, which is independent of the stepsize. This additional parameter allows a polynomial method to maintain a specific stepsize while scaling its input times relative to the local smoothness of the solution (See Figure 4.1).

We express the input times in terms of a node set $\{z_j\}_{j=1}^q$ where $|z_j| \leq 1$. The input times are obtained by scaling the node set by the radius r and translating by the current timestep center:

$$\text{input times} \quad t_j^{[n]} = rz_j + t_n, \quad j = 1, \ldots, q.$$

Similarly, we express the stage times in terms of a node set $\{c_j\}_{j=1}^q$, where the constants c_j may implicitly depend on the parameters r and h:

$$\text{stage times} \quad T_j^{[n]} = rc_j + t_n, \quad j = 1, \ldots, s.$$

Finally, the derivatives for each input, output, and stage will be respectively denoted as

$$f_j^{[n]} = f(t_j^{[n]}, y_j^{[n]}), \qquad f_j^{[n+1]} = f(t_j^{[n+1]}, y_j^{[n+1]}), \qquad \text{and} \qquad F_j = f(T_j, Y_j).$$

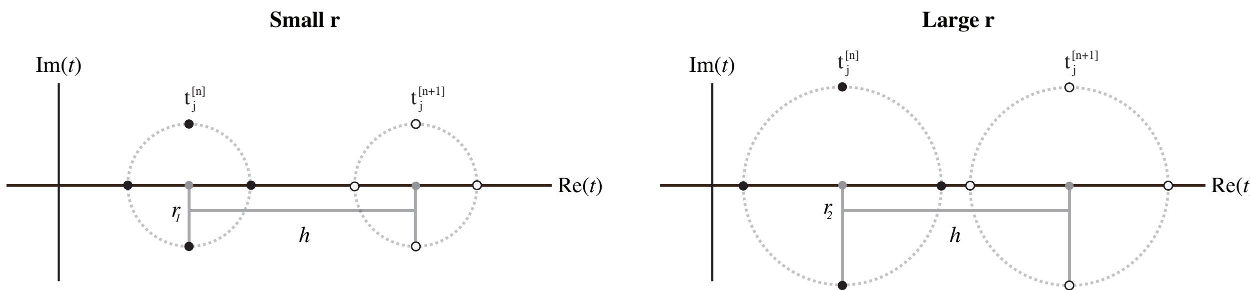


Figure 4.1: We illustrate the effects of varying the node radius r while keeping the stepsize h constant. We show the complex t-plane containing the input times $t_j^{[n]}$ (black circles) and the output times $t_j^{[n+1]}$ (white circles) corresponding to the four roots of unity.

At times, it is notationally convenient to use vector notation to express the inputs, outputs and stages. We therefore introduce the input vector $\mathbf{y}^{[n]}$ and the input derivative vector $\mathbf{f}^{[n]}$ where

$$\mathbf{y}^{[n]} = \left[y_1^{[n]}, \ \ldots, \ y_q^{[n]}\right]^{\mathrm{T}}, \qquad \mathbf{f}^{[n]} = \left[f_1^{[n]}, \ \ldots, \ f_q^{[n]}\right]^{\mathrm{T}},$$

as well as the similarly defined output vector $\mathbf{y}^{[n+1]}$, output derivative vector $\mathbf{f}^{[n+1]}$, stage vector $\mathbf{Y}$ and stage derivative vector $\mathbf{F}$.

4.1.2 Parametrizing the Stepsize

We will usually describe polynomial time-integrators in terms of their ODE polynomials rather than their coefficients. By parametrizing the stepsize h in terms of the node radius r, we obtain natural variables for working with polynomials in local coordinates. We introduce the extrapolation factor α and parametrize the stepsize as

$$h = r\alpha$$

where α represents the number of radii r per timestep h. If a polynomial time integrator has complex nodes, then smaller α will require more analyticity in the solution per timestep h. To minimize analyticity requirements, we generally seek methods with large α, though we will see in future sections that this will not always be possible for reasons of stability and round-off error.

In summary, using the new parametrization, the input times, output times, and stage times for every polynomial method are determined from the following parameters:

$\{z_j\}_{j=1}^{q}$	nodes	r	node radius
$\{c_j(\alpha)\}_{j=1}^{s}$	stage nodes	α	extrapolation factor

4.1.3 Special Families of Methods

Here we discuss several special families of polynomial methods. We first distinguish between propagator methods that advance the solution, and iterator methods that update or improve a candidate solution. We then introduce coarsener and refiner methods whose input and output times are generated from different node sets. It should be noted that these categorizations are not mutually exclusive and that one may construct integrators that simultaneously coarsen or refine and advance or correct the solution.

4.1.3.1 Propagators and Iterators

Propagators advance the solution forwards in time and are characterized by an extrapolation factor α that is greater than zero. When α is zero, the stepsize $h = r\alpha = 0$, and the method reduces to one that recomputes or improves the accuracy of the solution at the current time-step. We will refer to such methods as iterators. See Figure 5.2 for a visualization.

4.1.3.2 Coarsener and Refiner Methods

Coarseners and refiners are polynomial methods that accept q inputs and produce m outputs where $m \neq q$. If there are fewer outputs $(m < q)$ then the method is a coarsener, and if there are fewer inputs $(m > q)$ then the method is a refiner (See Figure 4.3). A single coarsener or refiner is not a time-stepping method as it can only be applied once. However these methods

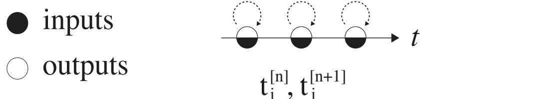
can be used to compute additional outputs and may be combined with standard polynomial time-integrators to construct more sophisticated composite methods.

We will express the input and output times for coarseners and refiners in terms of the node sets $\{z_j^{\text{in}}\}_{j=1}^{q}$ and $\{z_j^{\text{out}}\}_{j=1}^{m}$ such that

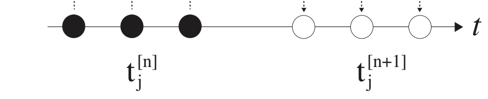
$$\begin{aligned} &\text{input times:} && t_j^{\text{in}} = r z_j^{\text{in}} + t_n, && j = 1, \ldots, q \\ &\text{output times:} && t_j^{\text{out}} = r z_j^{\text{out}} + t_n + r\alpha, && j = 1, \ldots, m \end{aligned}$$

The inputs, input derivatives, outputs and output derivatives will be respectively denoted as

$$y_j^{\text{in}},\ f_j^{\text{in}},\ j = 1, \ldots, q \qquad \text{and} \qquad y_j^{\text{out}},\ f_j^{\text{out}},\ j = 1, \ldots, m.$$

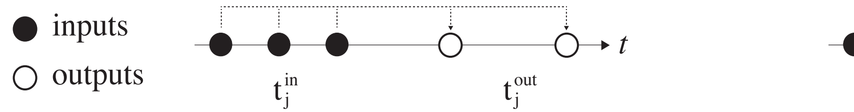


(a) An iterator recomputes the solution at the same time points, so that $t_j^{[n+1]} = t_j^{[n]}$

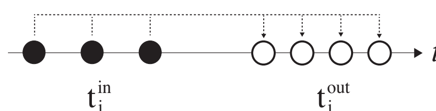


(b) A propagator advances the solution forwards in time so that $t_j^{[n+1]} = t_j^{[n]} + r\alpha$.

Figure 4.2: We distinguish between iterator and propagator methods. We show input times $t_j^{[n]}$ and output times $t_j^{[n+1]}$ on the real t line when z_j are three equispaced points. Iterators can be useful for correcting or updating an approximate solution, while propagators are traditional time-stepping schemes.



(a) An coarsener produces fewer outputs than inputs.



(b) A refiner produces more outputs than inputs.

Figure 4.3: Refiners and coarseners are not classical time-stepping schemes since they cannot be applied in succession. However, they can be used to compute additional outputs times or to modify an input in preparation for a second method.

4.2 Polynomial Block Methods

Block methods [36, 37] generate a set, or block, of q new values at each timestep. They are natural candidates for exploring polynomial time integration since we may use the multivalue

input to form high-order polynomial approximations of the differential equation solution. Here we restrict ourselves to block methods of the form

$$\mathbf{y}^{[n+1]} = \mathbf{A}\mathbf{y}^{[n]} + h\mathbf{B}\mathbf{f}^{[n]} + \mathbf{C}\mathbf{y}^{[n+1]} + h\mathbf{D}\mathbf{f}^{[n+1]} \tag{4.1}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are $q \times q$ coefficient matrices. These block methods do not compute any stages or off-step points, and all outputs are computed by taking linear combinations of the inputs, outputs and the corresponding derivatives. We distinguish between six types of block methods based on their *architecture* (parallel or serial) and their degree of *implicitness* (explicit, diagonally implicit, and fully implicit). For parallel diagonally implicit schemes, the q nonlinear systems are independent and may be solved simultaneously, while for parallel explicit schemes, the right-hand-side evaluations may be computed simultaneously. In Table 4.1 we classify these schemes based on the structure of their coefficient matrices.

	Explicit	*Diagonally Implicit*	*Fully Implict*
Parallel	$\mathbf{C} = \mathbf{D} = \mathbf{0}$	$\mathbf{D}$ diagonal, $\mathbf{C} = \mathbf{0}$	no such methods.
Serial	$\mathbf{C}$, $\mathbf{D}$ S.L.T	$\mathbf{C}$, $\mathbf{D}$ lower triagular.	$\mathbf{C}$, $\mathbf{D}$ may be dense

Table 4.1: Block methods (4.1) classified by matrix structure. The abbreviation S.L.T stands for strictly lower triangular.

Recall that for polynomial integrators we parameterize the stepsize as $h = r\alpha$ where r scales the input times and α is the number of radii r per timestep. Applying this parametrization to classical block methods, leads to integrators of the form

$$\mathbf{y}^{[n+1]} = \mathbf{A}(\alpha)\mathbf{y}^{[n]} + r\mathbf{B}(\alpha)\mathbf{f}^{[n]} + \mathbf{C}(\alpha)\mathbf{y}^{[n+1]} + r\mathbf{D}(\alpha)\mathbf{f}^{[n+1]} \tag{4.2}$$

The stepsize for a parametrized block method is given by $h = r\alpha$, where the point radius r acts as a scaling factor for the input and output times. A parametrized block method is a polynomial method if the coefficient matrices $\mathbf{A}(\alpha)$, $\mathbf{B}(\alpha)$, $\mathbf{C}(\alpha)$ and $\mathbf{D}(\alpha)$ can be derived from ODE polynomial. However, we will avoid the coefficient formulation (4.2), in favor of a description that explicitly reveals the underlying polynomial approximations.

4.2.1 General Form

Polynomial block methods (PBMs) compute each output by evaluating an ODE solution polynomial constructed from the method's input and output data. Every polynomial block method depends on the parameters

q	number of inputs/outputs	$\{z_j\}_{j=1}^q$	nodes, $z_j \in \mathbb{C}$, $\lvert z_j \rvert \leq 1$
r	node radius, $r \geq 0$	$\{b_j\}_{j=1}^q$	expansion points
α	extrapolation factor		

and can be written as

$$y_j^{[n+1]} = p_j(z_j + \alpha;\ b_j), \qquad j = 1, \ldots, q, \tag{4.3}$$

where each $p_j(\tau; b)$ is an ODE solution polynomial built from the ODE dataset

$$D(r, t_n) = \begin{cases} \text{inputs} : \left\{ \left(z_j,\ y_j^{[n]},\ r f_j^{[n]} \right) \right\}_{j=1}^q \\ \text{outputs} : \left\{ \left(z_j + \alpha,\ y_j^{[n+1]},\ r f_j^{[n+1]} \right) \right\}_{j=1}^q \end{cases}$$

and an interpolated value set $I(r, t_n)$ generated from $D(r, t_n)$.

We may determine the architecture and degree of implicitness of a PBM based on the method's active value set. This classification provides a simple way to construct PBMs of a particular type. For each type of method listed in Table 4.1, the polynomial $p_j(\tau; b)$ may be constructed using the data

$$\begin{aligned} \text{inputs:} \quad & y_k^{[n]},\ f_k^{[n]} \text{ for } k \in \{1, \ldots, q\} \\ \text{outputs:} \quad & y_k^{[n+1]},\ f_k^{[n+1]} \text{ for } k \in O(j) \end{aligned} \tag{4.4}$$

where the set $O(j)$ is defined in Table 4.2.

	Explicit	*Diagonally Implicit*	*Fully Implicit*
Parallel	$O(j) = \emptyset$	$O(j) = \{j\}$	no such methods
Serial	$O(j) = \{1, \ldots, j-1\}$	$O(j) = \{1, \ldots, j\}$	$O(j) = \{1, \ldots, q\}$

Table 4.2: For a PBM, the *ODE polynomial* $p_j(\tau; b)$ may be formed using all input data together with the output data $y_k^{[n+1]}$, $f_k^{[n+1]}$ for $k \in O(j)$

4.2.1.1 Coarseners and Refiners

In section 4.1.3.2 we introduced polynomial coarsener and refiner methods that take q inputs and produce m outputs where $q \neq m$. Here we write the formula for a polynomial block coarsener or refiner. In addition to the usual r and α parameters, a polynomial block coarsener or refiner also depends on:

q	number of inputs	$\{z_j^{\text{in}}\}_{j=1}^q$ and $\{z_j^{\text{out}}\}_{j=1}^m$	input and output nodes
m	number of outputs	$\{b_j\}_{j=1}^m$	expansion points

A polynomial block coarsener or refiner can be written as

$$y_j^{\text{out}} = p_j(z_j^{\text{out}} + \alpha;\ b_j), \qquad j = 1, \ldots, m \tag{4.5}$$

where each $p_j(x, b)$ is an *ODE polynomial* over the dataset

$$D(r, t_n) = \begin{cases} \text{inputs}: \left\{\left(z_j^{\text{in}},\ y_j^{\text{in}},\ rf_j^{\text{in}}\right)\right\}_{j=1}^q \\ \text{outputs}: \left\{\left(z_j^{\text{out}},\ y_j^{\text{out}},\ rf_j^{\text{out}}\right)\right\}_{j=1}^q \end{cases}$$

4.3 Polynomial General Linear Methods

General linear methods (GLMs) are a large class of multistage, multivalue integrators [13, 14, 16] that encompass Runge-Kutta methods, linear multistep methods, and predictor-correctors methods. The space of GLMs has proved difficult to explore due to the vast number of possibilities. Nevertheless, continuing efforts to study GLMs have produced new methods with favorable properties that cannot be achieved by traditional integrators.

In this section, we briefly describe classical GLMs and introduce their polynomial counterparts (PGLMs). By restricting ourselves to polynomial GLMs we can more easily navigate the space of possibilities using intuition gained from polynomial approximation theory. The PGLM formulation also allows us succinctly express all polynomial time-integrators, including one-step methods and multivalue methods with stages. Furthermore, the space of certain polynomial GLMs is closed under composition, allowing us to construct new methods through composition without introducing additional classes of methods.

If we denote the input approximations for a GLM at the nth timestep as $y_i^{[n]}$, $i = 1, \ldots, q$, and the stage values as Y_i, $i = 1, \ldots, s$, then any GLM can be written as

$$\begin{aligned} Y_i &= h\sum_{j=1}^{s} a_{ij}F_j + \sum_{j=1}^{q} u_{ij}y_j^{[n]}, \quad i = 1, 2, \ldots, s, \\ y_i^{[n+1]} &= h\sum_{j=1}^{s} b_{ij}F_j + \sum_{j=1}^{q} v_{ij}y_j^{[n]}, \quad i = 1, 2, \ldots, q, \end{aligned} \tag{4.6}$$

where the stage derivatives $F_j = f(Y_j)$. By introducing the matrices $A = [a_{ij}]$, $U = [u_{ij}]$, $B = [b_{ij}]$, $V = [v_{ij}]$, and the vectors $Y = [Y_i]$, $F = [F_i]$, we may express general linear methods in the following compact notation

$$\begin{bmatrix} Y \\ y^{[n+1]} \end{bmatrix} = \left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] \begin{bmatrix} hF \\ y^{[n]} \end{bmatrix} \tag{4.7}$$

If $q = 1$, GLMs reduce to a Runge-Kutta methods, while if $s = 1$ GLMs reduces to linear multistep methods. By parametrizing the stepsize as $h = r\alpha$, the underlying GLM takes the form

$$\begin{aligned} Y_i &= r\sum_{j=1}^{s} a_{ij}(\alpha)F_j + \sum_{j=1}^{q} u_{ij}(\alpha)y_j^{[n]} \\ y_i^{[n+1]} &= r\sum_{j=1}^{s} b_{ij}(\alpha)F_j + \sum_{j=1}^{q} v_{ij}(\alpha)y_j^{[n]} \end{aligned} \quad \equiv \quad \begin{bmatrix} Y \\ y^{[n+1]} \end{bmatrix} = \left[\begin{array}{c|c} A(\alpha) & U(\alpha) \\ \hline B(\alpha) & V(\alpha) \end{array}\right] \begin{bmatrix} rF \\ y^{[n]} \end{bmatrix} \tag{4.8}$$

4.3.1 General Form

Polynomial general linear methods (PGLMs) possess any number of inputs and any number of stages. Each method computes output and stage values by evaluating ODE solution polynomials constructed from the input and stage data. To construct a polynomial block method, one must choose two set of nodes (one for inputs and one for stages), and a set of *ODE polynomials*.

Every polynomial general linear method depends on the parameters

q	number of inputs/outputs	$\{z_j\}_{j=1}^{q}$	nodes, $z_j \in \mathbb{C}$, $\lvert z_j\rvert \leq 1$
s	number of stages	$\{b_j\}_{j=1}^{s+q}$	expansion points
r	node radius, $r \geq 0$	$\{c_j(\alpha)\}_{j=1}^{s}$	stage nodes
α	extrapolation factor		

and can be written as

$$\begin{aligned} Y_j &= p_j(c_j(\alpha);\ b_j), & j &= 1,\ldots,s, \\ y_j^{[n+1]} &= p_{j+s}(z_j+\alpha;\ b_{j+s}), & j &= 1,\ldots,q, \end{aligned} \tag{4.9}$$

where each $p_j(\tau; b)$ is an ODE solution polynomial built from the ODE dataset

$$D(r,t_n) = \begin{cases} \text{inputs}: \left\{\left(z_j,\ y_j^{[n]},\ rf_j^{[n]}\right)\right\}_{j=1}^{q} \\ \text{stages}: \left\{\left(c_j(\alpha),\ Y_j,\ rF_j\right)\right\}_{j=1}^{s} \end{cases} \tag{4.10}$$

and any interpolated value set $I(r, t_n)$ generated from $D(r, t_n)$. By computing the weights for evaluating each ODE polynomial we can rewrite (4.9) in coefficient form as

$$\begin{aligned} Y &= \mathbf{A}(\alpha)\mathbf{y}^{[n]} + r\mathbf{B}(\alpha)\mathbf{f}^{[n]} + \mathbf{C}(\alpha)Y + r\mathbf{D}(\alpha)F, \\ \mathbf{y}^{[n+1]} &= \mathbf{U}(\alpha)\mathbf{y}^{[n]} + r\mathbf{V}(\alpha)\mathbf{f}^{[n]} + \mathbf{W}(\alpha)Y + r\mathbf{X}(\alpha)F \end{aligned} \tag{4.11}$$

where the matrices

$$\begin{aligned} &\mathbf{A}(\alpha),\ \mathbf{B}(\alpha) \in \mathbb{C}^{s\times q} && \mathbf{C}(\alpha),\ \mathbf{D}(\alpha) \in \mathbb{C}^{s\times s} \\ &\mathbf{U}(\alpha),\ \mathbf{V}(\alpha) \in \mathbb{C}^{q\times q} && \mathbf{X}(\alpha),\ \mathbf{W}(\alpha) \in \mathbb{C}^{q\times s} \end{aligned}$$

The method (4.11) can be written as a parametrized GLM of the form (4.8) where the input is

$$y^{[n]} = \begin{bmatrix} \mathbf{y}^{[n]} \\ r\mathbf{f}^{[n]} \end{bmatrix}$$

and the coefficient matrices are

$$\begin{aligned} A(\alpha) &= (\mathbf{I} - \mathbf{C}(\alpha))^{-1}\mathbf{D}(\alpha), & U(\alpha) &= (\mathbf{I} - \mathbf{C}(\alpha))^{-1}\left[\mathbf{A}(\alpha) \mid \mathbf{B}(\alpha)\right], \\ B(\alpha) &= \mathbf{W}(\alpha)A(\alpha) + \mathbf{X}(\alpha), & V(\alpha) &= \mathbf{W}(\alpha)U(\alpha) + \left[\mathbf{U}(\alpha),\ \mid \mathbf{V}(\alpha)\right]. \end{aligned}$$

Note that we are using non-bold characters to reference GLM parameters from (4.8) and bold characters to reference parameters from the PGLM (4.11).

4.3.1.1 Coarseners and Refiners

In addition to the usual r and α parameters, a PGLM coarsener or refiner depends on:

q	number of inputs	$\left\{z_j^{\text{in}}\right\}_{j=1}^{q}$ and $\left\{z_j^{\text{out}}\right\}_{j=1}^{m}$	input and output nodes
m	number of outputs	$\{b_j\}_{j=1}^{m}$	expansion points
s	number of stages	$\{c_j(\alpha)\}_{j=1}^{s}$	stage nodes

A polynomial block coarsener or refiner can be written as

$$\begin{aligned} Y_j &= p_j(c_j(\alpha);\ b_j), & j &= 1, \ldots, s, \\ y_j^{[n+1]} &= p_{j+s}(z_j^{\text{out}} + \alpha;\ b_{j+s}), & j &= 1, \ldots, q, \end{aligned} \tag{4.12}$$

where each $p_j(x, b)$ is an *ODE polynomial* over the dataset

$$D(r, t_n) = \begin{cases} \text{inputs} : \left\{\left(z_j^{\text{in}},\ y_j^{\text{in}},\ rf_j^{\text{in}}\right)\right\}_{j=1}^{q} \\ \text{stages} : \left\{(c_j(\alpha),\ Y_j,\ F_j)\right\}_{j=1}^{s} \end{cases}$$

4.3.2 Polynomial Linear Multistep Methods

Linear multistep methods (LMMs) are an important class of time-integrators that compute a single output using solution and derivative values from previous time-steps. The general form for a linear multistep method is

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j f(t_{n-j}, y_{n-j}). \tag{4.13}$$

We cannot parametrize classical LMMs in terms of the node radius r and the extrapolation parameter α since the parametrized methods will fail to produce solution values at equispaced points and would require new method coefficients at each timestep (see Figure 4.4). However, if the output y_{n+1} of a classical LMM (4.13) can be expressed in terms of an ODE solution polynomial, then we can interpret it as a special polynomial general linear methods with one stage, equispaced nodes, and a fixed α.

A polynomial linear multistep method is a special family of polynomial general linear method with

$$\begin{aligned} s &= 1 & \{z_j\}_{j=1}^{q} &= \left\{-1 + \tfrac{2(j-1)}{q-1}\right\}_{j=1}^{q} \\ \alpha &= \tfrac{2}{q-1} & \{c_j(\alpha)\}_{j=1}^{s} &= \left\{\tfrac{q+1}{q-1}\right\} \end{aligned}$$

the free parameters

- q number of inputs/outputs
- r node radius, $r \geq 0$
- b_1 expansion points for the single stage value

The formula for a polynomial LLM can be written as

$$Y_1 = p_1(1+\alpha; b_1) \qquad \text{and} \qquad y_j^{[n+1]} = \begin{cases} y_{j+1}^{[n]} & j < q \\ Y_1 & j = q \end{cases}, \tag{4.14}$$

where $p(\tau; b)$ is an ODE solution polynomial constructed from the ODE dataset

$$D(r, t_n) = \begin{cases} \text{inputs}: \left\{\left(z_j,\ y_j^{[n]},\ r f_j^{[n]}\right)\right\}_{j=1}^{q} \\ \text{stages}: \left\{\left(\dfrac{q+1}{q-1},\ Y_1,\ rF_1\right)\right\} \end{cases}$$

and any interpolated value set $I(r, t_n)$ generated from $D(r, t_n)$. Since all polynomial LLMs can be characterized entirely by the ODE polynomial $p_1(\tau; b)$, we can write 4.14 using the shorthand

$$y_{n+1} = p_1(1+\alpha; b_1). \tag{4.15}$$

4.3.3 Polynomial Runge-Kutta Methods

Runge-Kutta methods are a class of widely-studied one-step methods. They can be constructed to possess favorable stability properties and can be easily implemented with a variable time-step. However, high-order methods require more derivative evaluations per

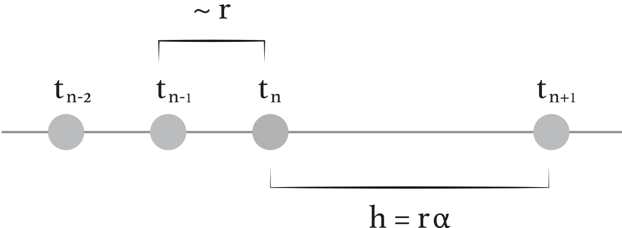


Figure 4.4: Consider an explicit parametrized linear multistep method with $p = 2$. The distance between the inputs is proportional to r, since r takes the place of h for a parametrized method. Furthermore, the method will produce an output at the time $t_{n+1} = t_n + r\alpha$ where α is free. Therefore the output time t_n is independent of the inputs t_{n-2}, t_{n-1}, and t_n. This means we will need the method will require modified coefficients at the next timestep to account for the new temporal locations of the input data. Therefore a method of the form

$$y_{n+1} = \sum_{j=0}^{p} a_j(\alpha) y_{n-j} + r \sum_{j=-1}^{p} b_j(\alpha) f_{n-j}$$

cannot exist since the iteration would only be valid for a simple timestep.

time-step than LMMs and the derivations for these schemes require solving a large number of nonlinear order conditions. A Runge-Kutta method with s stages can be written as

$$\begin{aligned} Y_i &= y_n + h \sum_{j=1} a_{ij} F(t_n + hc_j, Y_j) \qquad\qquad j = 1, \dots s \\ y_{n+1} &= y_n + \sum_{j=1}^{n} b_j F(t_n + hc_j, Y_j) \end{aligned}$$

By parametrizing the stepsize as $h = r\alpha$, a Runge-Kutta method takes the form

$$\begin{aligned} Y_i &= y_n + r \sum_{j=1} a_{ij}(\alpha) F(t_n + rc_j(\alpha), Y_j) \qquad\qquad j = 1, \dots s \\ y_{n+1} &= y_n + r \sum_{j=1}^{n} b_j(\alpha) F(t_n + rc_j(\alpha), Y_j) \end{aligned}$$

If the stages Y_j and the outputs y_n can be expressed in terms of ODE polynomials, then we can write this method as a PGLM (4.9) where

$$q = 1 \qquad \text{and} \qquad \{z_j\}_{j=1}^{q} = \{0\}.$$

4.4 Adams, BDF and GBDF Integrators

In subsection 3.2.1 we introduced the Adams, BDF, and GBDF families of ODE solution polynomial. A polynomial integrator whose outputs and stages are all computed from a single family of ODE solution polynomial, will inherit the same name as the family. For example, the implicit method (2.4) is a BDF polynomial block method since each of its outputs are computed by evaluating BDF ODE solution polynomials. Similarly, the explicit method (2.10) is an Adams integrator.

4.5 Conjugate Inputs, Outputs and Stages

If we are solving a real-valued initial value problem using a polynomial integrator that computes solution values at conjugate points in the complex plane, then we can hope to use the Swartz reflection principle to reduce the total number of required function evaluations and nonlinear solves. Any pairs of inputs, outputs or stages that can be compute via reflection will be called *conjugate*.

Definition 4.5.1 (Conjugate Values)**.** *Let M be a polynomial method with*

$$\begin{array}{lll} \textit{inputs} & y_j^{[n]} & j = 1, \dots q, \\ \textit{outputs} & y_j^{[n+1]} & j = 1, \dots q, \\ \textit{stages} & Y_j & j = 1, \dots, s. \end{array}$$

The inputs $y_j^{[n]}$ and $y_k^{[n]}$ are conjugate if:

1. *The temporal nodes are conjugate s.t. $z_j = z_k^*$.*
2. *The values are conjugate s.t. $y_j^{[n]} = \left(y_k^{[n]}\right)^*$.*

The outputs $y_j^{[n+1]}$ and $y_k^{[n+1]}$ are conjugate if:

1. *The temporal nodes are conjugate s.t. $z_j + \alpha = (z_k + \alpha)^*$.*
2. *The ODE polynomial $p_{j+s}(\tau; b)$ and $p_{k+s}(\tau; b)$ are conjugate.*

The stages Y_j and Y_k are conjugate if:

1. *The temporal stage nodes are conjugate s.t. $c_j(\alpha) = (c_k(\alpha))^*$.*
2. *The ODE polynomial $p_j(\tau; b)$ and $p_k(\tau; b)$ are conjugate.*

If two stages or outputs are conjugate, then we only need to compute one directly. Similarly, if two inputs are conjugate then we only need to compute one right-hand-side evaluation. For certain methods, this can reduce the number of nonlinear solves and total function evaluations in half. Thus certain methods with complex nodes require fewer nonlinear solves and function evaluations per time-step than a real method of equivalent order. However, each of the solves or function evaluations will require complex arithmetic.

4.6 Order of Accuracy

We can determine the order of accuracy of a polynomial integrator by writing it as a general linear method, specifying a starting method, and analyzing the resulting algebraic order conditions [17, Sec. 53]. Unfortunately, this processes is tedious and the result is dependent on the starting method. Alternatively, we can obtain a lower bound for the order of accuracy by inspecting a method's ODE polynomials. It follows that the order of accuracy must be greater than or equal to the lowest order of the ODE polynomials that form a method. For many methods the true order of accuracy will be equal to this lower bound. This fact greatly simplifies the construction of high-order integrators, and leads to the following theorem.

Theorem 4.6.1. *A polynomial method is at least order ρ accurate with respect to the node radius r if:*

1. *The starting method produces inputs that are at least order ρ accurate with respect to the node radius r.*

2. *All of the methods ODE polynomials for computing stages, outputs, and interpolated values are at least order ρ accurate.*

Remark 4.6.1. *To formally prove order of accuracy in the GLM sense [17, Sec. 53], we will require the inputs to be order ρ accurate. However, in practice it may be more convenient to compute the starting values using a low-order integrator with a strict error tolerance.*

In summary, to construct a high-order polynomial time-integrator we must simply choose high-order ODE polynomials. Compare the simplicity of this statement to the corresponding result for obtaining high-order Runge-Kutta and general linear methods.

4.7 Linear Stability

Linear stability for all classical time integrations is determined using the Dahlquist test problem $y' = \lambda y$. When applied to this problem, the timestep iteration for a method that produces q outputs will reduce to

$$\mathbf{y}^{[n+1]} = \mathbf{M}(z)\mathbf{y}^{[n]}$$

where $z = h\lambda$ and $\mathbf{M}(z)$ is a $q \times q$ matrix. The stability region S is the subset of the complex z-plane where $\mathbf{M}(z)$ is power bounded so that

$$S = \left\{ z \ : \ \sup_{n \in \mathbb{N}} \|\mathbf{M}(z)^n\| < \infty \right\}$$

The matrix $\mathbf{M}(z)$ will be power bounded if its eigenvalues lie inside the closed unit disk, and if any eigenvalues of magnitude one are non-defective. For a complete description of linear stability we refer the reader to [17, 16]. In practice, we may obtain approximate linear stability regions by numerically computing the eigenvalues of $\mathbf{M}(z)$ on a grid of complex z values.

4.7.1 Linear Stability for Parametrized Methods

We can trivially extend linear stability analysis to parameterized GLMs (4.8) and parametrized block methods (4.2). When applied to the Dahlquist test problem, these methods reduce to the iteration

$$\mathbf{y}^{[n+1]} = \mathbf{M}(\zeta, \alpha)\mathbf{y}^{[n]}$$

where $\zeta = r\lambda$ and $\mathbf{M}(\zeta, \alpha)$ is a $q \times q$ matrix. We define the stability regions for parametrized methods differently depending on whether the underlying method is a propagator ($\alpha > 0$) or an iterator ($\alpha = 0$). In each case, the stability region is the subset of the complex z-plane defined as

$$S(\alpha) = \left\{ \zeta \ : \ \sup_{n \in \mathbb{N}} \|\mathbf{M}(\zeta/\alpha, \alpha)^n\| < \infty \right\} \text{ for propagators, and}$$

$$S(0) = \left\{ \zeta \ : \ \sup_{n \in \mathbb{N}} \|\mathbf{M}(\zeta, 0)^n\| < \infty \right\} \text{ for iterators.}$$

For propagators we take $\zeta \to \zeta/\alpha = z$ so that the stability regions for parametrized methods are scaled relative to the stepsize h instead of the node radius r. This re-scaling allows us to overlay the stability regions for parametrized methods directly with those of classical methods.

4.7.2 Characterizing Linear Stability Regions

We will characterize methods with stability regions that extend to infinity by considering $A(\theta)$ *stability* and *A-stability*. Conversely, we will characterize methods with finite stability regions by their *negative stability intervals* and *imaginary stability intervals*. These three properties are defined as follows:

1. A method is $A(\theta)$ stable if its stability region $S \supset \{z : |\arg(-z)| < \theta, \ z \neq 0\}$. If a method possesses $A(90°)$ stability, then it is A-stable.

2. A method has negative stability interval β if its stability region $S \supset [-\beta, 0]$.

3. A method has imaginary stability interval β if its stability region $S \supset [-i\beta, i\beta]$.

In Figure (4.5) we illustrate each of these properties. We remark that $A(\theta)$ stability is normally referred to as $A(\alpha)$ stability however we avoid this notation since we have already used the variable α to denote the extrapolation factor.

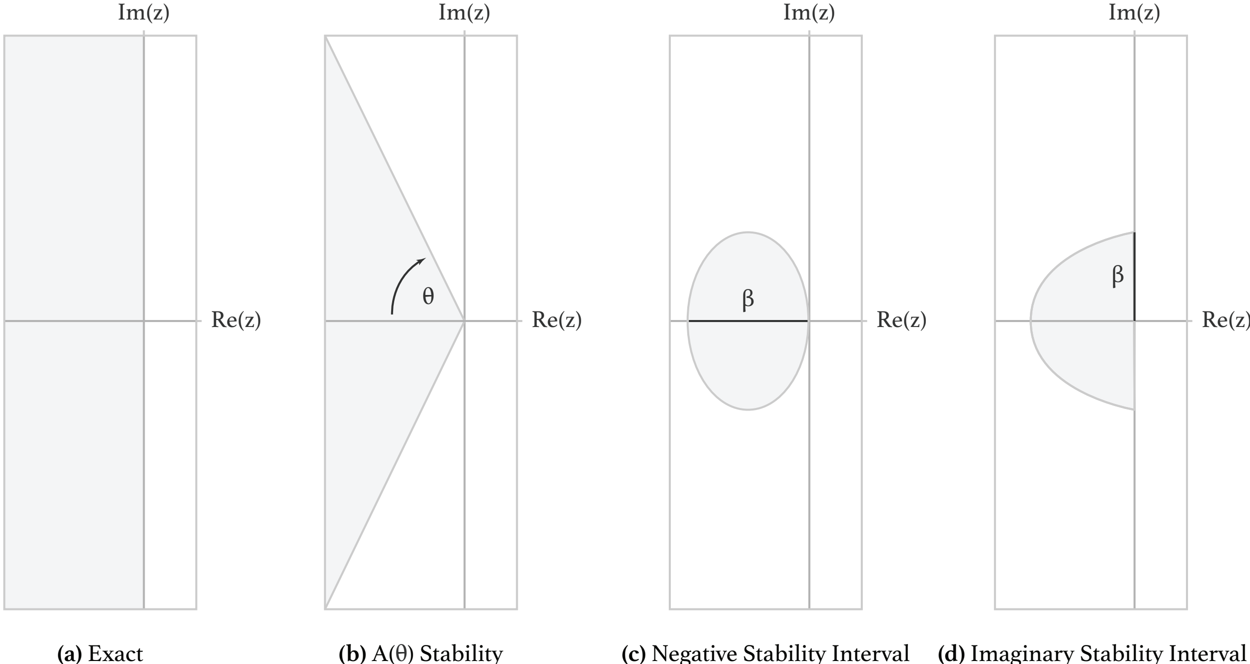


Figure 4.5: Four illustrations depicting different properties of stability regions. **(a)** The exact solution to the Dahlquist test problem remains bounded for all time if $\mathrm{Re}(z) \leq 0$. **(b)** $A(\theta)$ stability requires that a stability region contains a sector of the the left half plane with angle 2θ, centered along the real line. **(c)** The negative stability interval measures the width of the a stability region along the negative real line and relative to the origin. **(d)** The imaginary stability interval measures the width of the a stability region along the positive imaginary line and relative to the origin.

4.8 Diagrams for Geometrically Describing Polynomial Integrators

In Chapter 3 we introduced a set of diagrams and stencils for visualizing the geometric properties of ODE polynomials. With a few minor modifications, we can use these illustrations to highlight the geometric properties of polynomial time integrators. We discuss the modifications below and and introduce two new diagrams.

An **active node diagram** for a polynomial integrator with q outputs and s stages contains $q+s$ node stencils. Each node stencil shows the active node set for one of the ODE solution polynomials used to compute the method's outputs and stages. The temporal node for this output or stage will respectively referred to as the *current output* or *current stage*. In each node stencil, we use different markers to represent the temporal nodes depending on whether the node is active, and on whether the underlying data is an input, output, or stage. The following table shows the different markers that will be used in an active node diagram.

Underlying Data	Active	Inactive
Current Output	○	○
Input or Output	●	●
Stage Value	◉	◉
Current Stage Value	⊙	⊙
Interpolated Solution	⊞	⊞
Interpolated Derivative	⊕	⊕

A **polynomial diagram** for an integrator with q outputs and s stages contains $q+s$ ODE polynomial diagrams. Each diagram represents one of the ODE solution polynomials used to compute the method's outputs and stages. Every temporal node is labeled depending on: (1) whether the corresponding active data is a solution value, a derivative value, or both solution and derivative values, and (2) whether the node corresponds to an input, output, stage, or interpolated value. The markers to designate each of these cases are listed below:

Data Type	Input	Output	Stage	Interpolated
Active Solution Value	■	□	⊡	⊞
Active Derivative Value	●	○	⊙	⊕
Active Solution & Derivative Values	◖	▢	⊡	⊞
Inactive Data	•	•		

Optional Markers	
Expansion Point	•

An empty space indicates that the data type will not be shown. It is possible for solution values, derivative values, and interpolated derivative values to overlap. For the moment, we will avoid introducing additional makers to denote each of these cases.

If a method's ODE solution polynomial are constructed using interpolated values, then the diagram should also include additional ODE polynomial diagrams that describe any ODE solution polynomial and ODE derivative polynomial that were used to form the interpolated value set.

4.8.1 Example Diagrams

We close this section by presenting several simple example diagrams. In Figure 4.6 we show node diagrams for a class of polynomial linear multistep methods and a class of polynomial

block method with real nodes. Even for this simple example, the diagrams describe a broad range of polynomial GBDF, BDF, Adams integrators.

In Figure 4.7 we show the polynomial diagrams for the ODE polynomials used by third-order BDF, fourth-order Adams-Moulton and third-order Adams-Bashforth. Unlike node diagrams, polynomial diagrams are unique to each method because they describe the ODE polynomials for computing each output and stage.

Diagonally-Implicit Explicit
j = 1 j = 1

(a) Polynomial Linear Multistep Methods

Diagonally-Implicit Explicit
j = 1 j = 1
j = 2 j = 2

(b) Polynomial Block Methods

Figure 4.6: Example node diagrams for linear multistep methods and polynomial block methods with $q = 2$. For both types of methods, we show diagrams for a diagonally-implicit method and an explicit method. The node diagram for the diagonally-implicit LMM describes both the 3rd order Adams-Moulton method and the 2nd order backwards differentiation formula while the explicit node diagram describes 2nd order Adams-Bashforth.

3rd order BDF: $H_y^{[1]}$

4th order Adams-Moulton: $L_y^{[1]}$ $L_F^{[1]}$

3rd order Adams-Bashforth: $L_y^{[1]}$ $L_F^{[1]}$

Figure 4.7: Polynomial diagrams for the ODE polynomial used by the third order BDF method, fourth-order Adams-Moulton, and third-order Adams-Bashforth.

Chapter 5

CONSTRUCTING POLYNOMIAL BLOCK METHODS

In this chapter we present a geometric approach for constructing Adams, BDF, and GBDF polynomial block methods with real and imaginary nodes. Our methodology can be used to generate a range of new implicit and explicit integrators with variable order, and with parallel or serial architecture. In total, there are over 180 families of block method presented in this chapter, each of which can be implemented at any order and using any choice of nodes. Remarkably, all of these methods have been derived using geometric intuition, and without considering even a single order condition!

Recall that a polynomial block method can be written as

$$y_j^{[n+1]} = p_j(z_j + \alpha; b_j), \qquad\qquad j = 1, \ldots, q$$

where each $p_j(\tau; b)$ is an ODE solution polynomial that is built from the method's input and output values. To construct a polynomial block method one must choose a set of nodes, a set of ODE Polynomials, and a set of expansion points. In this chapter we will use our geometric intuition to generate a range of possible parameters. To derive a specific method, one must simply choose a combination of these preselected options.

We begin our discussion of parameters by introducing two families of nodes sets, one characterized by real-valued nodes and the other by imaginary nodes that are symmetric about the real axis. For each of these families we then present a variety of different ways to specify the active data values of the ODE solution polynomial for computing a method's outputs. Finally, we describe a simple procedure for constructing BDF, GBDF, or Adams ODE solution polynomials using any combination of the previous parameters. The procedure for constructing a polynomial time-integrator can be summarized in the following three steps:

1. Select a node family described in Section 5.1.

2. Select an active index sex from the list of possibilities listed in Section 5.2.

3. Select a type of method (BDF, GBDF, or Adams) and a node set that belongs to the appropriate family. Then build the method's ODE polynomial according to the procedure in Section 5.3.

Additional Notation For Method Construction

Before discussing parameters, we introduce some additional notation to facilitate the description of active node sets for polynomial time-integrators. Let $p_j(\tau; b)$, be a set of ordered ODE solution polynomials for computing the outputs of a PBM. Recall that each ODE polynomial has an active node set that contains the temporal nodes of the data values used to compute its approximate derivatives.

When discussing PBM construction it is beneficial to: (1) split the active node set into input and output nodes to improve readability, and (2) reference the node index rather than the node value, thereby allowing us to characterize active node sets for PBMs with differing nodes. We will call the resulting sets the *active input index set* and the *active output index set*.

Definition 5.0.1 (Active Input Index Set)**.** *The active input index set $I(j)$ for a polynomial method with q outputs is a set containing integers ranging from 1 to q where $k \in I(j)$ if and only if $p_j(\tau; b_j)$ is constructed using input data $y_j^{[n]}$ or $f_j^{[n]}$.*

Definition 5.0.2 (Active Output Index Set)**.** *The active output index set $O(j)$ for a polynomial method with q outputs is a set containing integers ranging from 1 to q such that $k \in O(j)$ if and only if $p_j(\tau; b_j)$ is constructed using output data $y_j^{[n+1]}$ or $f_j^{[n+1]}$.*

Example 5.0.1. *Let $p_j(\tau; b_j)$ be an ODE polynomial for computing the jth output of a polynomial integrator with q total outputs. If $p_j(\tau; b_j)$ is constructed using the input values $y_1^{[n]}$, $f_2^{[n]}$ and output values $f_j^{[n+1]}$ for $j = 1, \ldots, q$, then it will have active index sets*

$$I(j) = \{1,\ 2\}, \qquad\qquad O(j) = \{1,\ \ldots,\ q\}$$

irregardless of the node locations of the data.

5.1 Selecting Nodes

The first step in our proposed procedure for constructing polynomial block methods is to choose a node family. This choice does not fix the final node set, but rather helps inform future parameter choices and broadly characterizes the type of method we seek to construct.

In this section we present two node families for constructing polynomial block methods; the first is characterized by real-valued nodes and the second by imaginary nodes that are symmetric with respect to the real-axis. Real-valued nodes are desirable in situations where we either want to avoid complex arithmetic or the initial value problem cannot be extended into the complex plane. On the other hand, symmetric imaginary nodes provide a natural setting for constructing complex-valued integrators with conjugate input and outputs.

For each family of nodes we present several orderings that can each be used to construct different types of serial methods. We also provide mappings for converting from one ordering

to another. These mappings allow us to re-express parameters that have been developed for one ordering into other orderings.

In general, there are many additional families of nodes that will lead to interesting polynomial methods. For example, the roots of unity used to construct the integrator (2.10) are neither real or purely imaginary. However the study of additional node families lies outside the scope of this present work.

5.1.1 Real-Valued Nodes

The family of real-valued node sets, is the the family of all $\{z_j\}_{j=1}^q$ where

$$\mathrm{Im}(z_j) = 0 \ \forall j.$$

We consider two orderings for real-valued nodes:

1. **Right-Sweeping**: $z_1 < z_2 < \ldots < z_q$.

2. **Left-Sweeping**: $z_1 > z_2 > \ldots > z_q$.

We illustrate these orderings using the node set $\{z_j\}_{j=1}^3 = \{-1, 0, 1\}$ below.

Right Sweeping — 1 2 3

Left Sweeping — 3 2 1

We can trivially map between right-sweeping and left-sweeping orderings. If the nodes $z_1, \ \ldots, \ z_q$ are in left-sweeping or right-sweeping order then $z_{m(1)}, \ \ldots, \ z_{m(q)}$ will be in the opposite ordering if $m(j) = q - j + 1$.

5.1.2 Real-Symmetric Imaginary Nodes

The family of real-symmetric imaginary node sets, is the the family of all $\{z_j\}_{j=1}^q$ where

$$\mathrm{Re}(z_j) = 0 \qquad \text{and} \qquad \zeta \in \{z_j\}_{j=1}^q \iff \zeta^* \in \{z_j\}_{j=1}^q.$$

We consider three orderings for real-symmetric imaginary nodes:

1. **Classical Ordering**: nodes are ordered from top to bottom in the complex plane so that $iz_1 < iz_2 < \ldots < iz_q$.

2. **Outward Sweeping**: nodes are ordered so that their magnitude monotonically increases:

$$|z_1| \leq |z_2| \leq \ldots \leq |z_q| \qquad \text{where} \qquad |z_j| = |z_{j+1}| \implies iz_j > iz_{j+1}.$$

3. **Inward Sweeping**: nodes are ordered so that their magnitude monotonically decreases:

$$|z_1| \geq |z_2| \geq \ldots \geq |z_q| \qquad \text{where} \qquad |z_j| = |z_{j+1}| \implies iz_j > iz_{j+1}.$$

For inward sweeping and outward sweeping orderings, the second condition prioritizes points in the upper-half plane over conjugate points in the lower-half plane. In Figure 5.1 we illustrate each ordering using a node set consisting of either four or five imaginary equispaced nodes.

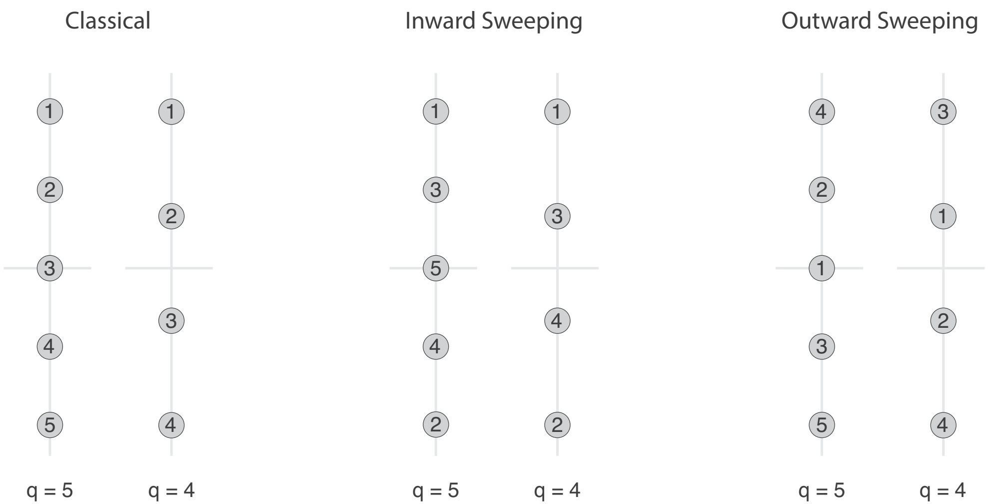


Figure 5.1: Real-symmetric imaginary node orderings shown for the node set $\{z_j\}_{j=1}^4 = \{i,\ i/3,\ i/3,\ i\}$ and $\{z_j\}_{j=1}^5 = \{-i,\ -i/2,\ 0,\ i/2,\ i\}$.

5.1.2.1 Order Mappings for Real-Symmatric Imaginary Nodes

If the nodes $z_1, \ldots, z_q$ are classically ordered nodes, then $z_{m(1)}, \ldots, z_{m(q)}$ will be ordered as inward sweeping or outward sweeping using the following mappings.

1. *Mapping from classical to inward sweeping:*

$$q \text{ odd or even:} \qquad m(j) = \begin{cases} q - \frac{j-1}{2} & j \text{ odd} \\ \frac{j}{2} & j \text{ even} \end{cases} \tag{5.1}$$

2. *Mapping from classical to outward sweeping:*

$$q \text{ odd:} \qquad m(j) = \begin{cases} \lceil \frac{q}{2} \rceil - \frac{j-1}{2} & j \text{ odd} \\ \lceil \frac{q}{2} \rceil + \frac{j}{2} & j \text{ even} \end{cases} \tag{5.2}$$

$$q \text{ even:} \qquad m(j) = \begin{cases} \frac{q}{2} + \frac{j+1}{2} & j \text{ odd} \\ \frac{q}{2} - \frac{j-2}{2} & j \text{ even} \end{cases} \tag{5.3}$$

A mapping $m(j)$ and its corresponding inverse mapping $m^{-1}(j)$ satisfy

$$j = m^{-1}(m(j)).$$

We define the following inverse mapings to reorder nodes in inwards sweeping or outwards sweeping ordering to classical ordering.

1. *Mapping from inward sweeping to classical:*

$$q \text{ odd or even:} \qquad m^{-1}(j) = \begin{cases} 2q - 2j + 1 & \lfloor \frac{q}{2} \rfloor + 1 \leq j \leq q \\ 2j & 1 \leq j \leq \lfloor \frac{q}{2} \rfloor \end{cases} \tag{5.4}$$

2. *Mapping from outward sweep to classical:*

$$q \text{ odd:} \qquad m^{-1}(j) = \begin{cases} 2j - 2\lceil \frac{q}{2} \rceil & \lceil \frac{q}{2} \rceil + 1 \leq j \leq q \\ 2\lceil \frac{q}{2} \rceil + 1 - 2j & 1 \leq j \leq \lceil \frac{q}{2} \rceil \end{cases} \tag{5.5}$$

$$q \text{ even:} \qquad m^{-1}(j) = \begin{cases} 2j - q - 1 & \frac{q}{2} + 1 \leq j \leq q \\ q + 2 - 2j & 1 \leq j \leq \frac{q}{2} \end{cases} \tag{5.6}$$

That these forward mappings $m(j)$ and inverse mappings $m^{-1}(j)$ can be composed to obtain mappings from inwards-sweeping order to outwards-sweeping order or vice versa.

5.2 Choosing An Active Node Index Set

The second step in our proposed procedure for constructing polynomial block methods is to choose an active node index set. Though it might seem peculiar to specify a method's active node sets before choosing its ODE polynomials, this construction strategy allows one to choose a method's architecture and maximum order, before discussing the formulas for its ODE polynomials.

In this section we present five ways to choose the active input index (AII) set and active output index (AOI) set for a PBM with real-valued nodes or real-symmetric imaginary nodes. Each pair of sets can be used to construct either an explicit or diagonally-implicit integrator. For the real-symmetric imaginary nodes, we only discuss AII and AOI sets that produce PBMs whose output times are either real-valued or a member of a conjugate pair. This restriction leads to integrators that require at most $\lceil q/2 \rceil$ nonlinear solves or $\lceil q/2 \rceil$ function evaluations per time-step when solving real-valued initial value problems.

A Naming Convention for Active Index Sets

We introduce a naming convention for each pair of active index sets that describes the underlying method properties. Since active index sets determine the architecture of a method, each name starts with the word *parallel* or *serial*. Next, the names reflect the cardinality of the AII and AOI sets. The theoretical maximum order of a method's ODE polynomials, and the order of the method are both proportional to the cardinality of the active index sets. Larger cardinalities mean that more data values can be used to construct polynomials, leading to increased order of accuracy. Active index sets that achieve a certain property using the maximal amount of data will have the word *maximal* in their name. Finally, the cardinalities of the active index sets can remain fixed or may vary for each of a method's outputs. All names will contain the words *fixed cardinality* or *variable cardinality* to reflect these two possibilities.

Presenting Formula For Active Index Sets

We will write formula for the AII set $I(j)$ directly. However the AOI set $O(j)$ will depend on whether one seeks to construct an explicit or diagonally-implicit integrator. We define this set as

$$O(j) = \begin{cases} B(j) & \text{for explicit methods} \\ B(j) \cup \{j\} & \text{for diagonally-implicit methods} \end{cases}. \tag{5.7}$$

where different formula for the set $B(j)$ are be contained in the following subsection.

For real-symmetric imaginary nodes, the AII set $I(j)$ and the set $B(j)$ will be expressed

in terms of the function $\chi_{\text{in}}(j)$ and $\chi_{\text{out}}(j)$ that are defined as:

$$\begin{aligned}
&\text{inwards ordering:} && \begin{cases} \chi_{\text{in}}(j) = \begin{cases} j & j \text{ odd} \\ j-1 & j \text{ even} \end{cases} \\[2ex] \chi_{\text{out}}(j) = \begin{cases} 1 & j \text{ odd} \\ 2 & j \text{ even} \end{cases} \end{cases}, \\
&\text{outwards ordering:} && \begin{cases} \chi_{\text{in}}(j) = \begin{cases} \max(1, j-1) & q \equiv j \mod 2 \\ j & \text{otherwise} \end{cases} \\[2ex] \chi_{\text{out}}(j) = \begin{cases} 1 & q \equiv j \mod 2 \\ 2 & \text{otherwise} \end{cases} \end{cases}.
\end{aligned} \tag{5.8}$$

Geometrically Interpreting Formula For Computing Active Index Sets

This section also contains active node diagrams that illustrate the geometric properties inherent to each of the proposed active index sets. The formula for each AII and AOI set should be read in tandem with their corresponding node diagram. This will allow the reader to appreciate the simple geometric construction underlying each active index set.

To highlight the properties of methods with real-valued nodes, we provide active node diagrams for PBMs whose nodes are the three equispaced points

$$\{z_j\}_{j=1}^{3} = \{-1, 0, 1\}$$

and whose extrapolation factor α satisfies $\alpha > 2$ (the condition on α simply prevents the nodes from overlapping in the diagrams). To highlight the properties of PBMs with real-symmetric imaginary nodes, we show active node diagrams for methods whose nodes are the four or five imaginary equispaced points

$$\{z_j\}_{j=1}^{4} = \{i,\ i/3,\ i/3,\ i\} \qquad \text{or} \qquad \{z_j\}_{j=1}^{5} = \{-i,\ -i/2,\ 0,\ i/2,\ i\}.$$

For clarity we show an empty node stencil for each of three node sets in Figure 5.2. The node diagrams for each active node set are contained in the following figures:

Figure 5.3: explicit PBMs with real-valued nodes.

Figure 5.4: diagonally-implicit PBMs with real-valued nodes.

Figure 5.5 and Figure 5.7: explicit PBMs with real-symmetric imaginary nodes.

Figure 5.6 and Figure 5.8: diagonally-implicit PBMs with real-symmetric imaginary nodes.

Formula for Five Types of Active Index Sets

The formulas for the AII and AOI sets are:

1. *Parallel Maximal-Fixed-Cardinality* (**PMFC**): all outputs must be computed using solution or derivative data from each input. No output data can be used.

$$\text{all nodes:} \quad \begin{cases} I(j) = \{1, \ \ldots, \ q\} \\ B(j) = \{\} \end{cases} . \tag{5.9}$$

2. *Serial Maximal-Variable-Cardinality* (**SMVC**). All outputs must be computed using solution or derivative from each input and all previously computed outputs. As we add new information the cardinality of the set $B(j)$ grows.

$$\text{real nodes:} \quad \begin{cases} I(j) = \{1, \ \ldots \ , \ q\} \\ B(j) = \{1, \ \ldots \ , \ j-1\} \end{cases} . \tag{5.10}$$

$$\text{imaginary nodes:} \quad \begin{cases} I(j) = \{1, \ \ldots \ , \ q\} \\ B(j) = \{1, \ \ldots \ , \ \chi_{\text{out}}(j)\} \end{cases} . \tag{5.11}$$

3. *Serial Maximal-Fixed-Cardinality* (**SMFC**): all outputs must be computed using solution or derivative from all previously computed outputs and some of the inputs. The cardinality of $I(j) \cup O(j)$ is fixed across all j. For imaginary nodes the active index sets are

$$\text{imaginary nodes:} \quad \begin{cases} I(j) = \{\chi_{\text{in}}(j), \ \ldots \ , \ q\} \\ B(j) = \{1, \ \ldots \ , \ \chi_{\text{out}}(j)\} \end{cases} . \tag{5.12}$$

   For real-valued nodes we can drop input nodes using a *first-in, first-out* (FIFO) approach or a *last-in, last-out* (LIFO) approach. The formula for these two cases are

$$\text{real nodes:} \quad \begin{cases} \textbf{SMFC FIFO} & \begin{cases} I(j) = \{j, \ \ldots \ , \ q\} \\ B(j) = \{1, \ \ldots \ , \ j-1\} \end{cases} \\ \textbf{SMFC LIFO} & \begin{cases} I(j) = \{1, \ \ldots \ , \ q-j+1\} \\ B(j) = \{1, \ \ldots \ , \ j-1\} \end{cases} \end{cases} . \tag{5.13}$$

4. *Parallel Maximal-Fixed-Cardinality minus j* (**PMFCmj**): the jth output must be computed using solution or derivative data from all inputs, excluding the input with index j.

$$\text{all nodes:} \quad \begin{cases} I(j) = \{1, \ \ldots, \ q\} \setminus \{j\} \\ B(j) = \{\} \end{cases} . \tag{5.14}$$

	$I(j)$	$B(j)$
PMFO	$\{1,\ \ldots,\ q\}$	$\{\}$
SMVO	$\{1,\ \ldots,\ q\}$	$\{1,\ \ldots,\ j-1\}$
SMFO FIFO	$\{j,\ \ldots,\ q\}$	$\{1,\ \ldots,\ j-1\}$
SMFO LIFO	$\{1,\ \ldots,\ q-j+1\}$	$\{1,\ \ldots,\ j-1\}$
SMFO$_\ell$ FIFO	$\{j+\ell,\ \ldots,\ q\}$	$\{1,\ \ldots,\ j-1\}$
SMFO$_\ell$ LIFO	$\{1,\ \ldots,\ q-j+\ell-1\}$	$\{1,\ \ldots,\ j-1\}$

Table 5.1: Node index sets for generating implicit and explicit polynomial block methods with real-valued nodes.

5. *Serial Maximal-Fixed-Cardinality minus j* (**SMFCmj**): all outputs must be computed using solution or derivative from all previously computed outputs and some of the inputs excluding the input with the index j. The cardinality of $A(j) \cup B(j)$ is fixed across all j. For imaginary nodes the active index sets are

$$\text{imaginary nodes:} \quad \begin{cases} I(j) = \{\chi_{\text{in}}(j),\ \ldots\ ,\ q\} \setminus \{j\} \\ B(j) = \{1,\ \ldots\ ,\ \chi_{\text{out}}(j)\} \end{cases}. \tag{5.15}$$

For real-valued nodes we can generalize this approach drop ℓ input nodes using a *first-in, first-out* (FIFO) approach or a *last-in, last-out* (LIFO) approach. The formula for these two cases are

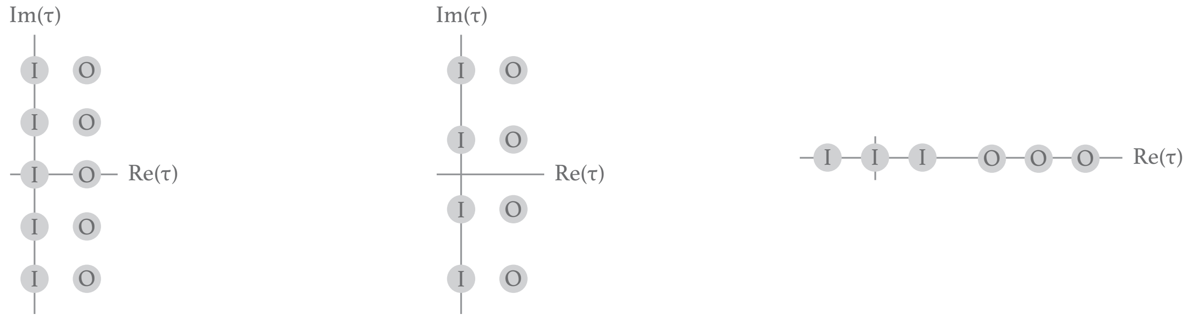
$$\text{real nodes:} \quad \begin{cases} \textbf{SMFC}_\ell\ \textbf{FIFO} & \begin{cases} I(j) = \{j+\ell,\ \ldots\ ,\ q\} \\ B(j) = \{1,\ \ldots\ ,\ j-1\} \end{cases} \\ \textbf{SMFC}_\ell\ \textbf{LIFO} & \begin{cases} I(j) = \{1,\ \ldots\ ,\ q-j-\ell+1\} \\ B(j) = \{1,\ \ldots\ ,\ j-1\} \end{cases} \end{cases}. \tag{5.16}$$

If $\ell = 1$ and FIFO ordering is chosen, then only the jth point is dropped

The formula for the real-valued nodes and real-symmetric imaginary nodes listed above are succinctly listed in Tables 5.1 and 5.2.

	$I(j)$	$B(j)$
PMFC	$\{1,\ \ldots,\ q\}$	$\{\}$
SMVC	$\{1,\ \ldots,\ q\}$	$\{1,\ \ldots,\ \chi_{\text{out}}(j)\}$
SMFC	$\{\chi_{\text{in}}(j),\ \ldots,\ q\}$	$\{1,\ \ldots,\ \chi_{\text{out}}(j)\}$
PMFCmj	$\{1,\ \ldots,\ q\} \setminus \{j\}$	$\{\}$
SMFCmj	$\{\chi_{\text{in}}(j),\ \ldots,\ q\} \setminus \{j\}$	$\{1,\ \ldots,\ \chi_{\text{out}}(j)\}$

Table 5.2: Node index sets for generating implicit and explicit polynomial block methods with real-symmetric imaginary nodes. The functions $\chi_{\text{in}}(j)$ and $\chi_{\text{out}}(j)$ are defined in (5.8)



(a) Five imaginary equispaced points.

(b) Four imaginary equispaced points.

(c) Three real equispaced points.

Figure 5.2: Empty active node stencils for an ODE solution polynomial for a block method with (**a**) nodes $\{z_j\}_{j=1}^{5} = \{-i,\ -i/2,\ 0,\ i/2,\ i\}$ and $\alpha > 0$, (**b**) nodes $\{z_j\}_{j=1}^{4} = \{i,\ i/3,\ i/3,\ i\}$ and $\alpha > 0$, (**c**) nodes $\{z_j\}_{j=1}^{3} = \{-1, 0, 1\}$ with $\alpha > 2$. For clarity, all inactive nodes have been enlarged, and inputs are labeled with the letter I while outputs are labeled with the letter O.

Explicit and Right Sweeping

PMFC SMVC SMFC-FIFO SMFC-LIFO SMFCmj-FIFO SMFCmj-LIFO

j = 1
j = 2
j = 3

Explicit and Left Sweeping

PMFC SMVC SMFC-FIFO SMFC-LIFO SMFOmj-FIFO SMFCmj-LIFO

j = 1
j = 2
j = 3

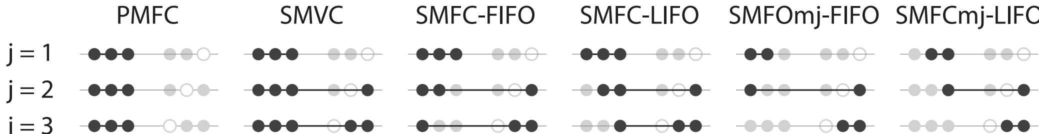


Figure 5.3: Node stencils for diagonally implicit block methods with three real equispaced points and $\alpha > 0$.

Diagonally Implicit and Right Sweeping

PMFC SMVC SMFC-FIFO SMFC-LIFO SMFCmj-FIFO SMFCmj-LIFO

j = 1
j = 2
j = 3

Diagonally Implicit and Left Sweeping

PMFC SMVC SMFC-FIFO SMFC-LIFO SMFCmj-FIFO SMFCmj-LIFO

j = 1
j = 2
j = 3

Figure 5.4: Node stencils for diagonally implicit block methods with three real equispaced points and $\alpha > 0$.

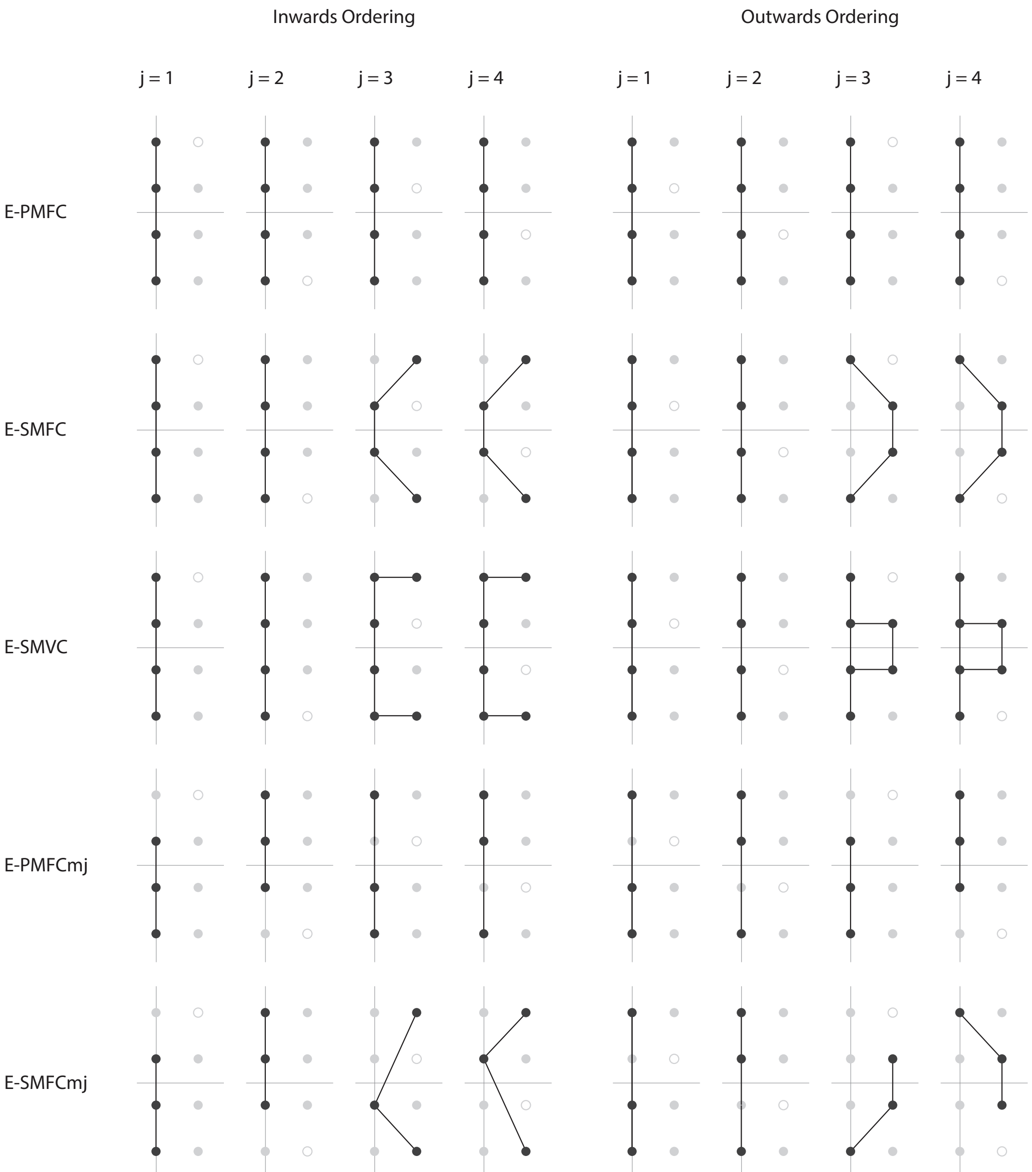


Figure 5.5: Node stencils for explicit block methods with four imaginary equispaced points and $\alpha > 0$.

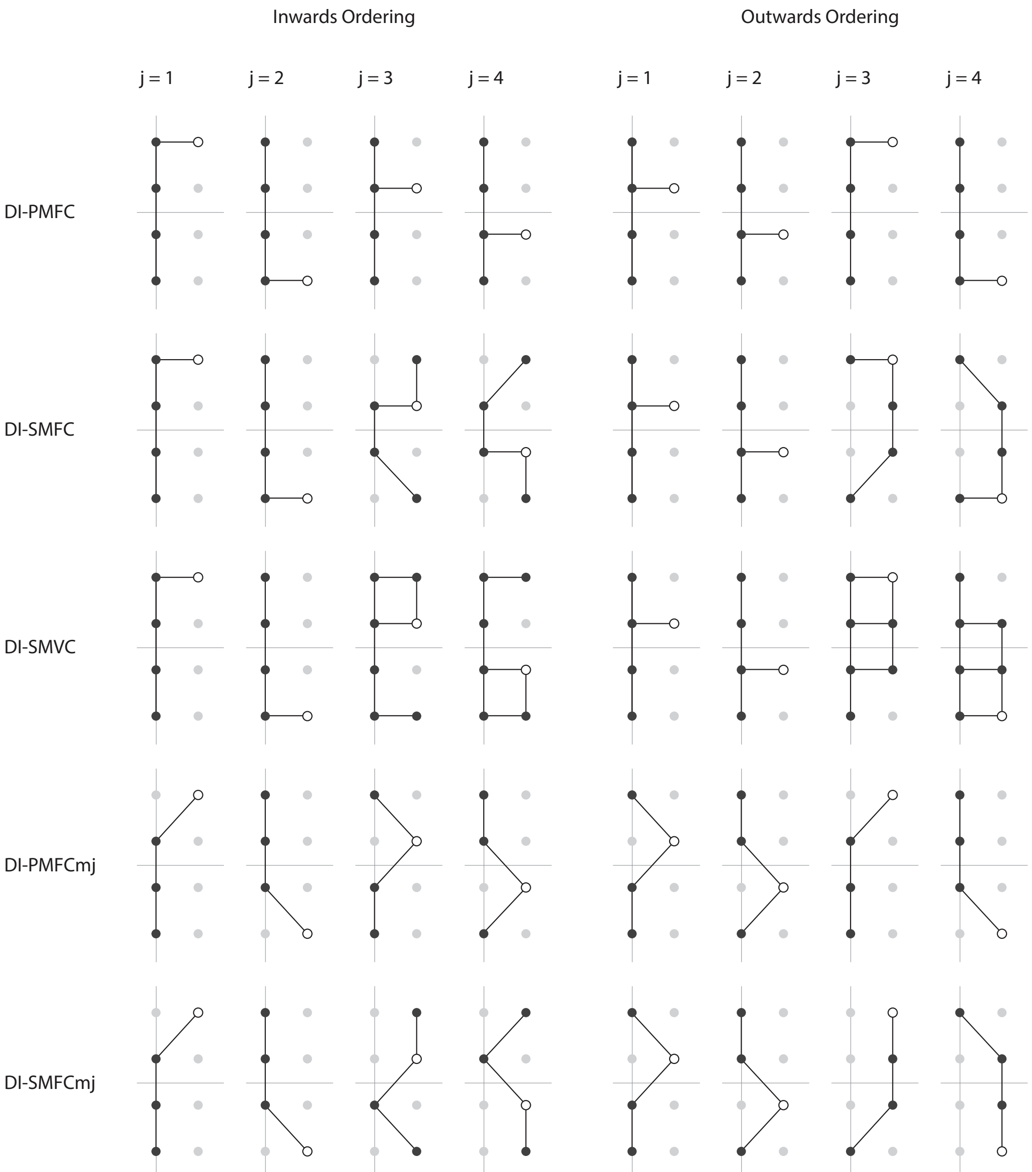


Figure 5.6: Node stencils for diagonally implicit block methods with four imaginary equispaced points and $\alpha > 0$.

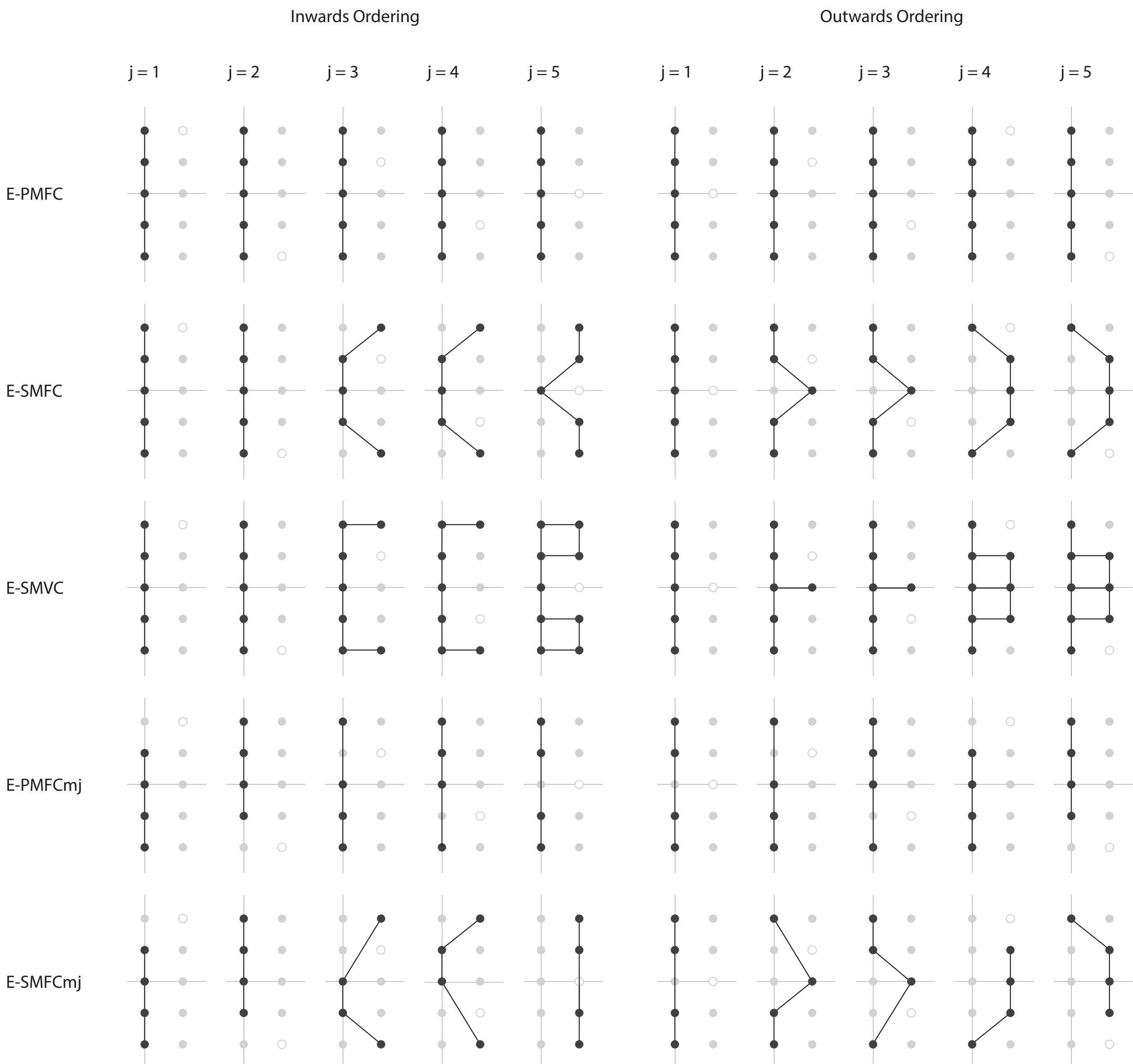


Figure 5.7: Node stencils for explicit block methods with five imaginary equispaced points and $\alpha > 0$.

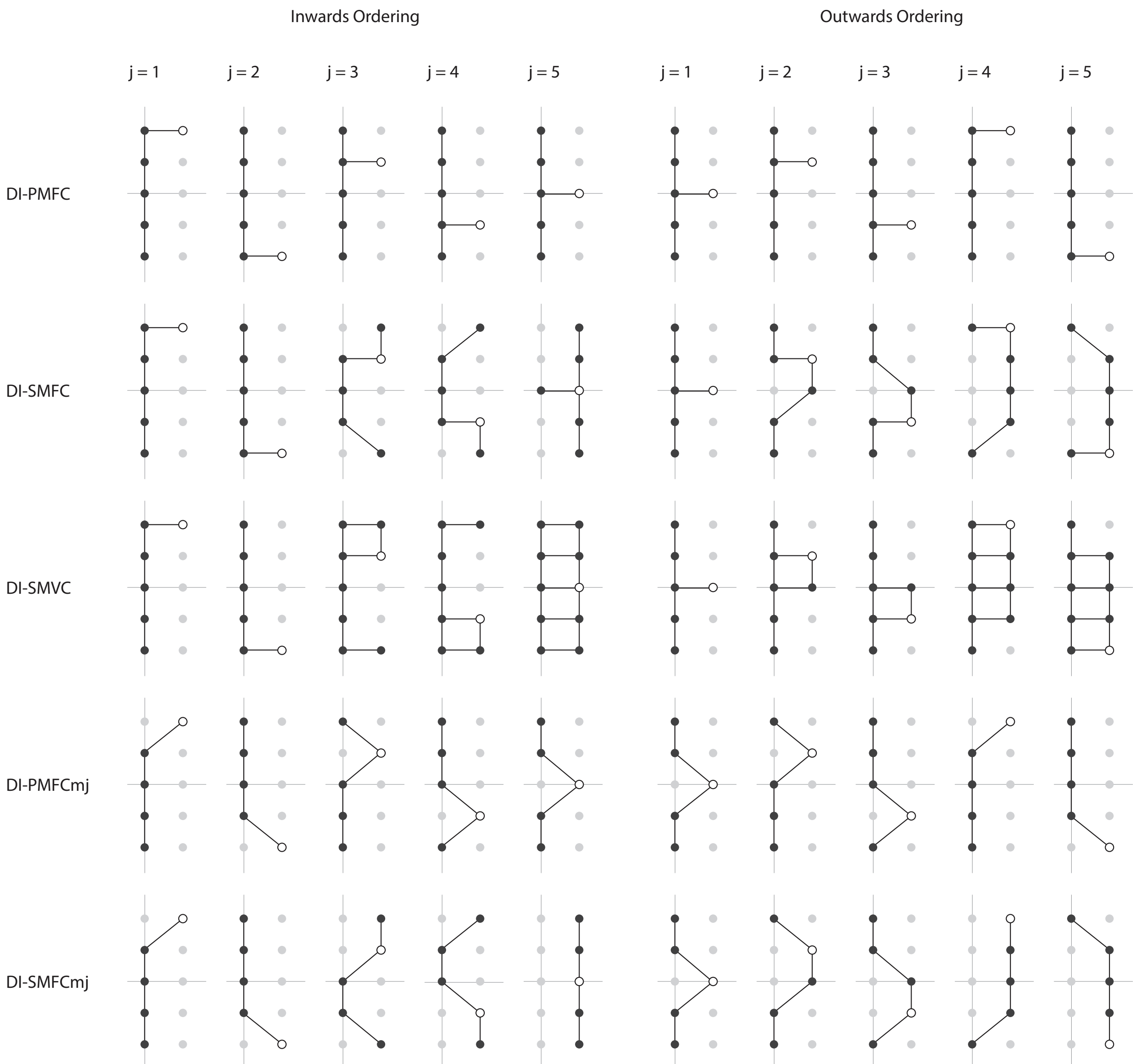


Figure 5.8: Node stencils for diagonally implicit block methods with five imaginary equispaced points and $\alpha > 0$.

5.3 Choosing ODE solution polynomials

The final step in our proposed procedure for constructing polynomial block methods is to choose a type of ODE polynomial for computing the outputs. In this section we provide a set of formulae for choosing Adams, BDF and GBDF ODE solution polynomials. If each of the polynomials is chosen from the same family, then we will obtain an Adams, BDF, or GBDF integrator. Our formulas are expressed in terms of the AII set $I(j)$ the AOI set $O(j)$ and the set $B(j)$ from (5.7). Depending on the desired type of method the formula for a PBMs ODE solution polynomial $p_j(\tau; b)$ can be chosen as follows:

1. To obtain an **Adams** ODE solution polynomial

$$p_j(\tau; b) = L_y^{[j]}(\tau) + \int_b^\tau L_F^{[j]}(\tau)$$

   choose the Lagrange polynomial $L_F^{[j]}(\tau)$ so that it passes through

input derivatives	$f_k^{[n]}$ for $k \in I(j)$,
output derivatives	$f_k^{[n+1]}$ for $k \in O(j)$.

   The Lagrange polynomial $L_y^{[j]}(\tau)$ should be constructed differently depending on the particular choice of expansion points $\{b_j\}_{j=1}^q$. We present several possibilities in Subsection 5.3.1.

2. To obtain a **BDF** ODE solution polynomial

$$p_j(\tau; b) = H_y^{[j]}(\tau)$$

   for an implicit method, choose the polynomial $H^{[j]}(\tau)$ so that it passes through

input values	$y_k^{[n]}$ for $k \in I(j)$,
output values	$y_k^{[n+1]}$ for $k \in B(j)$,
output derivatives	$f_j^{[n+1]}$.

   For an explicit method, we cannot use the output derivative $f_j^{[n+1]}$. Instead we construct an interpolated value set that contains q interpolated derivatives

$$\tilde{f}_j = \dot{p}_j(z_j + \alpha; b) \qquad j = 1, \ldots, q$$

   where $\tilde{f}_j \approx f_j^{[n+1]}$ and $\dot{p}_j(\tau; b) = L_F(\tau)$ where $L_F(\tau)$ is a Lagrange interpolating polynomial passing through

input derivatives	$f_k^{[n]}$ for $k \in I(j)$,
output derivatives	$f_k^{[n+1]}$ for $k \in O(j)$.

Then choose the polynomial $H^{[j]}(\tau)$ so that it passes through

input values	$y_k^{[n]}$ for $k \in I(j)$,
output values	$y_k^{[n+1]}$ for $k \in B(j)$,
output derivatives	$\tilde{f}_j$.

3. To obtain a **GBF** ODE solution polynomial

$$p_j(\tau; b) = H_y^{[j]}(\tau)$$

for an implicit method, choose $H^{[j]}(\tau)$ so that it passes through

input derivatives	$f_k^{[n]}$ for $k \in I(j)$,
output derivatives	$f_j^{[n+1]}$ for $k \in O(j)$.

For an explicit method, construct the interpolated value set that contains q interpolated derivatives

$$\tilde{f}_j = \dot{p}_j(\tau; b) \qquad\qquad j = 1, \ldots, q$$

and choose $\dot{p}_j(\tau; b) = L_F(\tau)$ where $L_F(\tau)$ is a Lagrange interpolating polynomial passing through

input derivatives	$f_k^{[n]}$ for $k \in I(j)$,
output derivatives	$f_k^{[n+1]}$ for $k \in O(j)$.

Finally choose the polynomial $H^{[j]}(\tau)$ so that it passes through

input values	$y_k^{[n]}$ for $k \in I(j)$,
output derivatives	$f_j^{[n+1]}$ for $k \in O(j)$ and $\tilde{f}_j$.

5.3.1 Endpoint Choices For Adams Polynomials

If the jth output of a PBM is computed using an Adams polynomial, then it can be written as

$$y_j^{[n+1]} = p_j(z_j + \alpha; b_j) = L_y^{[j]}(b_j) + \int_{b_j}^{z_j+\alpha} L_F^{[j]}(s)ds.$$

In general, the Lagrange polynomial $L_y^{[j]}(\tau)$ can be chosen independently of the endpoint b_j. However, to obtain the most compact representations for $L_y^{[j]}(\tau)$, and to obtain the highest

order of accuracy for the output, it is generally preferable to consider both parameters simultaneously.

In this section we present three ways to choose these parameters for real-nodes and real-symmetric imaginary nodes. At the end of the section we also provide one formula for choosing $L_y^{[j]}(\tau)$ that can be used for each of the endpoint choices. In addition to the formula, we provide a set of figures for visualizing the parameter choices.

1. *Fixed Input* (**FI**): This expansion point can be used for methods with serial or parallel architectures. For real-valued nodes, the expansion point b_j is equal to the temporal node of a single input. For imaginary nodes the expansion points b_j is equal to the temporal node of a single input or its conjugate (it is necessary to use conjugate endpoints to construct methods with conjugate outputs). There are q possibilities for choosing fixed input endpoints for a method with q outputs. We parametrize this with the variable $\ell \in \{1, \ldots, q\}$.

   For methods with real-valued nodes, the formulas are

   $$b_j = z_\ell \qquad \text{and} \qquad L_y^{[j]} : \left[y_\ell^{[n]} \right].$$

   The expansion diagrams and ODE polynomial for these parameters are shown Figure 5.9.

   For methods with real-symmetric imaginary nodes in classical ordering the formulas are

   $$b_j = z_{\text{ind}(j)} \qquad \text{and} \qquad L_y^{[j]} : \left[y_{\text{ind}(j)}^{[n]} \right]$$

   where the function $\text{ind}(j)$ is defined as

   $$\text{q even} \qquad \text{ind}(j) = \begin{cases} \ell & j = 1, \ldots, \frac{q}{2} \\ q - \ell + 1 & j = \frac{q}{2} + 1, \ldots, q \end{cases},$$

   $$\text{q odd} \qquad \text{ind}(j) = \begin{cases} \ell & j = 1, \ldots, \lceil \frac{q}{2} \rceil \\ \lceil \frac{q}{2} \rceil & j = \lceil \frac{q}{2} \rceil \\ q - \ell + 1 & j = \lceil \frac{q}{2} \rceil + 1, \ldots, q \end{cases}.$$

   To obtain the equivalent formula for nodes $z_1, \ldots, z_q$ with an inwards sweeping or outwards sweeping ordering, we can simply use the mappings defined in Subsection 5.1.2.1. First map the index j from inwards or outwards ordering to classical ordering via $j \leftarrow m^{-1}(j)$, then compute the index of the classically ordered point $j \leftarrow \text{ind}(j)$, and finally map back to the desired ordering via $j \leftarrow m(j)$. The full result is

   $$b_j = z_{m(\text{ind}(m^{-1}(j)))} \qquad \text{and} \qquad L_y^{[j]} : \left[y_{m(\text{ind}(m^{-1}(j)))}^{[n]} \right].$$

   The expansion diagrams and ODE polynomial for these parameters are shown Figure 5.11.

2. *Sliding Inputs* (**SI**): This choice of endpoint can be used for methods with serial or parallel architectures. The expansion point b_j is equal to the temporal node of the jth input.

$$\text{all nodes:} \qquad b_j = z_j \text{ and } L_j^{[j]} : [y_j^{[n]}].$$

The expansion diagrams and ODE polynomial for these real-valued nodes and real-symmetric imaginary nodes are respectively shown in Figures 5.9 and 5.12.

3. *Sliding Outputs* (**SO**): This expansion point is for serial methods only. For real-valued nodes the expansion point is equal to the temporal node of the last computed output. For real-symmetric imaginary nodes, the expansion point is equal to the newest previously compute inputs that has an already computed conjugate pair.

For PBMs with real-valued nodes, there are q possible endpoints for a method with q outputs. If we let $\ell \in \{1, \ldots, q\}$ the formula are

$$\text{all nodes:} \qquad b_j = \begin{cases} z_\ell & j = 1 \\ z_{j-1} & j > 1 \end{cases}, \qquad L_y^{[j]} : \begin{cases} [y_\ell^{[n]}] & j = 1 \\ [y_{j-1}^{[n+1]}] & j > 1 \end{cases}.$$

The expansion diagrams and ODE polynomial for these parameters are shown Figure 5.10.

For PBMs real-symmetric imaginary nodes in inwards ordering the formulas are

$$\text{q even:} \qquad b_j = \begin{cases} z_j & j \leq 2 \\ z_{j-2} + \alpha & j > 2 \end{cases}, \qquad L_y^{[j]} : \begin{cases} [y_j^{[n]}] & j \leq 2 \\ [y_{j-2}^{[n+1]}] & j > 2 \end{cases}.$$

$$\text{q odd:} \qquad b_j = \begin{cases} z_j & j \leq 2 \\ z_{j-2} + \alpha & 2 \leq j < q \\ z_q & j = q \end{cases}, \qquad L_y^{[j]} : \begin{cases} [y_j^{[n]}] & j \leq 2 \\ [y_{j-2}^{[n+1]}] & 2 \leq j < q \\ [y_q^{[n]}] & j = q \end{cases}.$$

For PBMs with real-symmetric imaginary nodes in outwards ordering the formulas are

$$\text{q even:} \qquad b_j = \begin{cases} z_j & j \leq 2 \\ z_{j-2} + \alpha & j > 2 \end{cases}, \qquad L_y^{[j]} : \begin{cases} [y_j^{[n]}] & j \leq 2 \\ [y_{j-2}^{[n+1]}] & j > 2 \end{cases}.$$

$$\text{q odd:} \qquad b_j = \begin{cases} z_1 & j = 1 \\ z_1 + \alpha & 1 < j \leq 3 \\ z_{j-2} & j > 3 \end{cases}, \qquad L_y^{[j]} : \begin{cases} [y_1^{[n]}] & j = 1 \\ [y_1^{[n+1]}] & 1 < j \leq 3 \\ [y_{j-2}^{[n]}] & j > 3 \end{cases}.$$

The expansion diagrams and ODE polynomial for these parameters are shown Figure 5.13.

A More General Set of Polynomials $L_y^{[j]}(\tau)$

We can choose a single set of polynomials

$$L_y^{[j]}(\tau) \qquad\qquad j = 1, \ldots, q$$

to construct all of the methods described above. The drawback to this representation is that for certain expansion point choices, the polynomials will be constructed with unused data points. Once these extra values are discarded, the formula for $L_y^{[j]}(\tau)$ will reduced to the respective formula described above.

The formula for the new polynomials $L_y^{[j]}(\tau)$ is

$$L_y^{[j]} : \begin{cases} \left[y_1^{[n]}, \;\ldots,\; y_q^{[n]}\right] & \text{ind}(j) < 1 \\ \left[y_1^{[n]}, \;\ldots,\; y_q^{[n]},\; y_1^{[n+1]}, \;\ldots,\; y_{\text{ind}(j)}^{[n+1]}\right] & \text{ind}(j) \geq 1 \end{cases} \tag{5.17}$$

where for real-valued nodes

$$\text{ind}(j) = j - 1$$

and for real-symmetric imaginary nodes

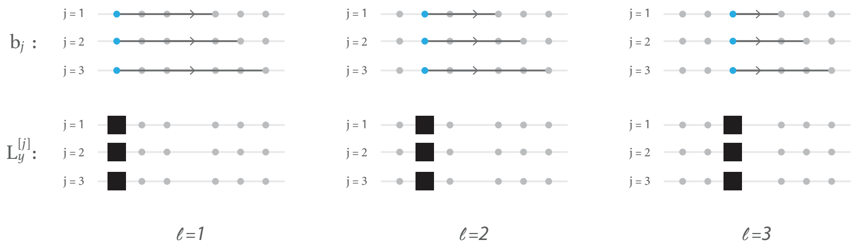
$$\text{inwards sweeping} \qquad \text{ind}(j) = \begin{cases} \text{q even} & \begin{cases} 0 & j \leq 2 \\ j-2 & j > 2 \end{cases} \\ \\ \text{q odd} & \begin{cases} 0 & j \leq 2 \\ j-2 & 2 < j < q \\ 0 & j = q \end{cases} \end{cases}$$

$$\text{outwards sweeping} \qquad \text{ind}(j) = \begin{cases} \text{q even} & \begin{cases} 0 & j \leq 2 \\ j-2 & j > 2 \end{cases} \\ \\ \text{q odd} & \begin{cases} 0 & j = 1 \\ 1 & 1 < j \leq 3 \\ j-2 & j > 3 \end{cases} \end{cases}$$

The ODE polynomial diagrams for each of these cases are shown in Figures 5.15 and 5.14.

These formulas are serial since they are constructed sequentially using output values. However, it is possible that the resulting method will become parallel once unused values are discarded. Because of this ambiguity we do not recommend using this general representation to describe a PBM. Nevertheless, these polynomials are useful when implementing several methods since they can be used to span a wide range of parameters.

Fixed Input - All Orderings



Sliding Input - All Orderings

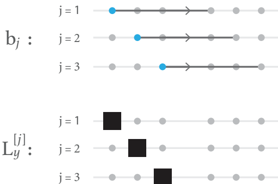


Figure 5.9: (**Fixed Input & Sliding Input, Real Nodes**) Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with three real equispaced nodes.

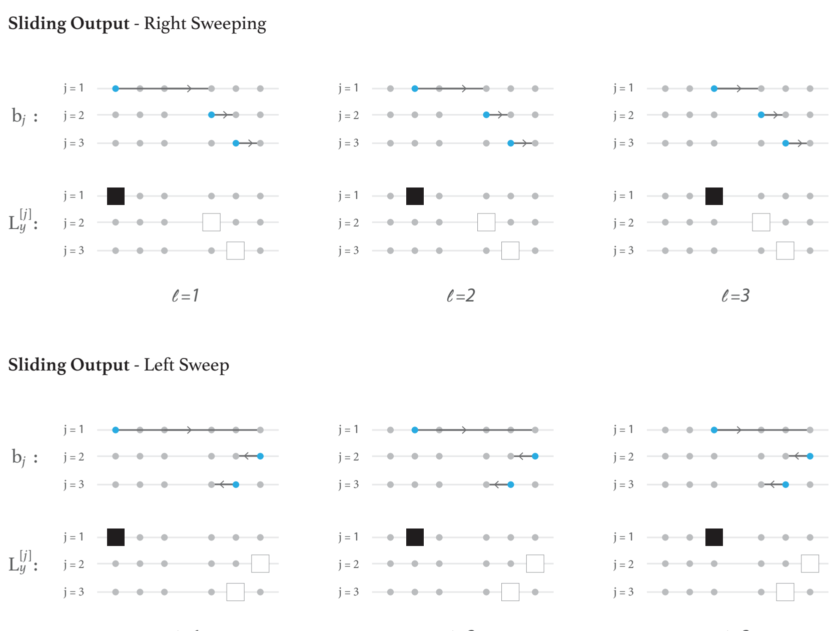
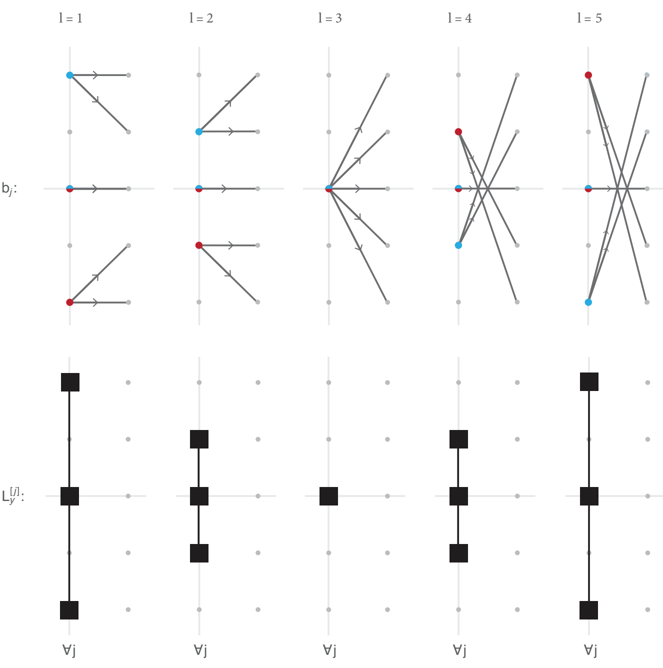


Figure 5.10: (Sliding Output, Real Nodes) Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with three real equispaced nodes.



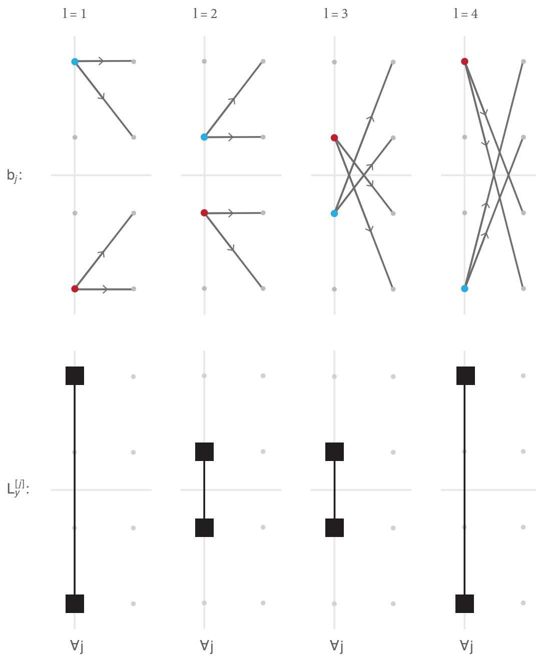
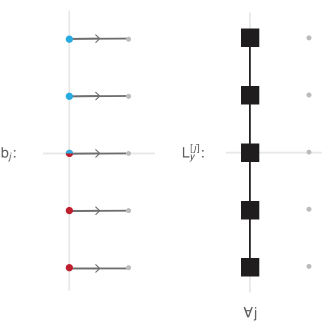


Figure 5.11: (**Fixed Input, Real-Symmetric Imaginary Nodes**) Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with four and five imaginary equispaced nodes. Conjugate expansion points are shown in red circles instead of blue circles.



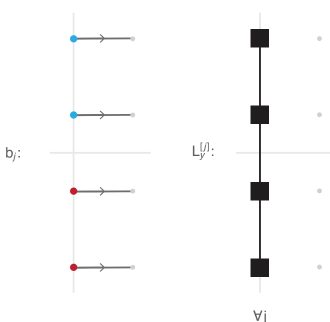
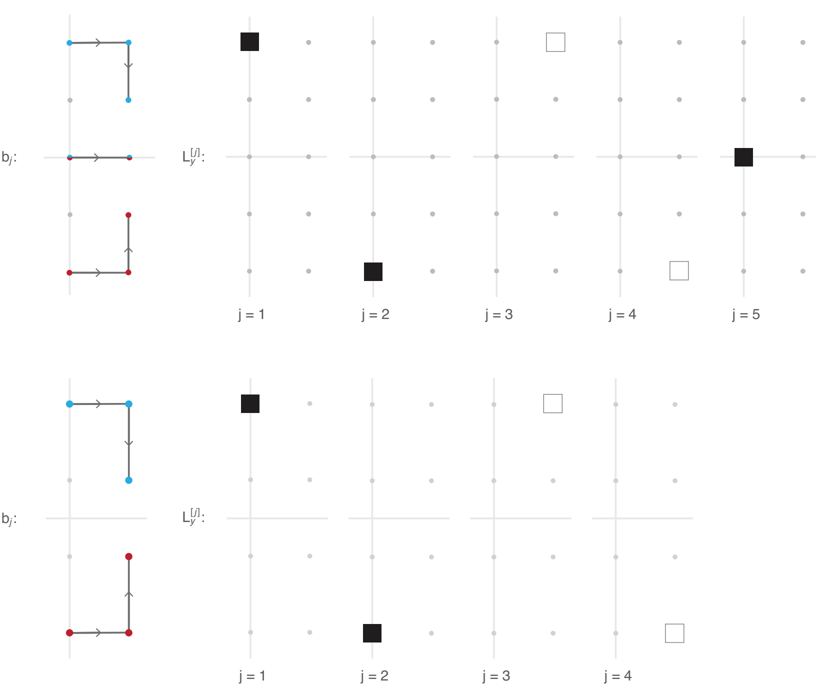


Figure 5.12: (Sliding Input, Real-Symmetric Imaginary Nodes) Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with with four and five imaginary equispaced nodes. Conjugate expansion points are shown in red circles instead of blue circles.



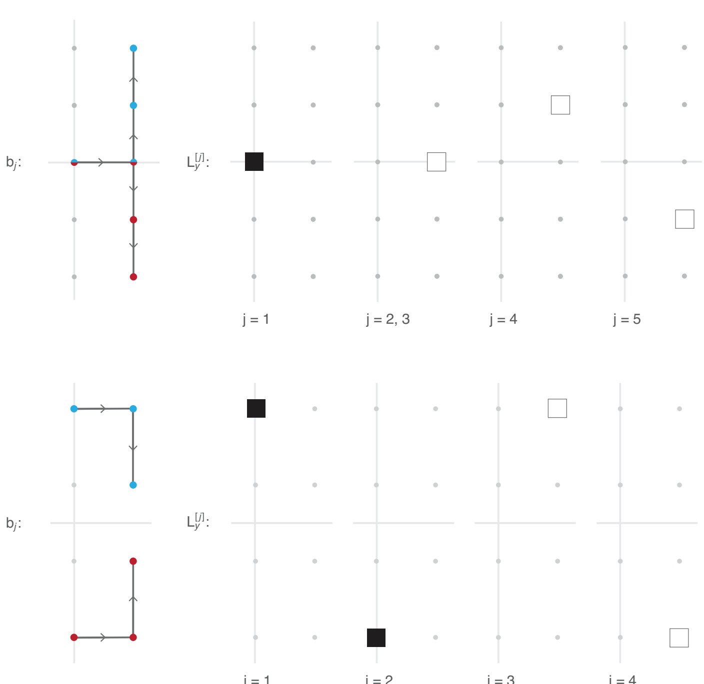


Figure 5.13: (Sliding Output, Real-Symmetric Imaginary Nodes) Expansion point diagrams and ODE polynomial diagrams of $L_y^{[j]}$ for Adams PBMs with with four and five imaginary equispaced nodes. Conjugate expansion points are shown in red circles instead of blue circles.

(a) Five Equispaced Points

(b) Four Equispaced Points

Figure 5.14: ODE polynomial diagrams for $L_y^{[j]}$ constructed using (5.17) and either four and five imaginary equispaced nodes. This non-compact representation of can be used with all endpoint listed in Subsection 5.3.1

Figure 5.15: ODE polynomial diagrams for $L_y^{[j]}$ constructed using (5.17) and three real equispaced nodes. This non-compact representation of can be used with all endpoint listed in Subsection 5.3.1

5.4 A Collection of Proposed Methods

We close this chapter by presenting a collection of methods that can be derived by using the parameters presented in the previous section. For each method we list the parameter choices and show a method diagram. The diagrams for PBMs with real-symmetric nodes depict methods with four outputs and with nodes that in classical ordering are

$$\{z_j\}_{j=1}^{4} = \{i,\ i/3,\ \ -i/3,\ \ -i\}.$$

Similarly, the diagrams for PBMs with real nodes depict methods with three outputs and with nodes that in left-sweeping ordering are

$$\{z_j\}_{j=1}^{4} = \{-1,\ 0,\ 1\}.$$

Counting The Methods

At the beginning of this chapter we claimed that we would present the formula for 180 methods. We can compute this number by simply considering all the possible combinations of the parameters presented in this section. See Figure 5.16 for an illustration using a graph. Also remember that this number excludes the order of each method and the specific choice of nodes!

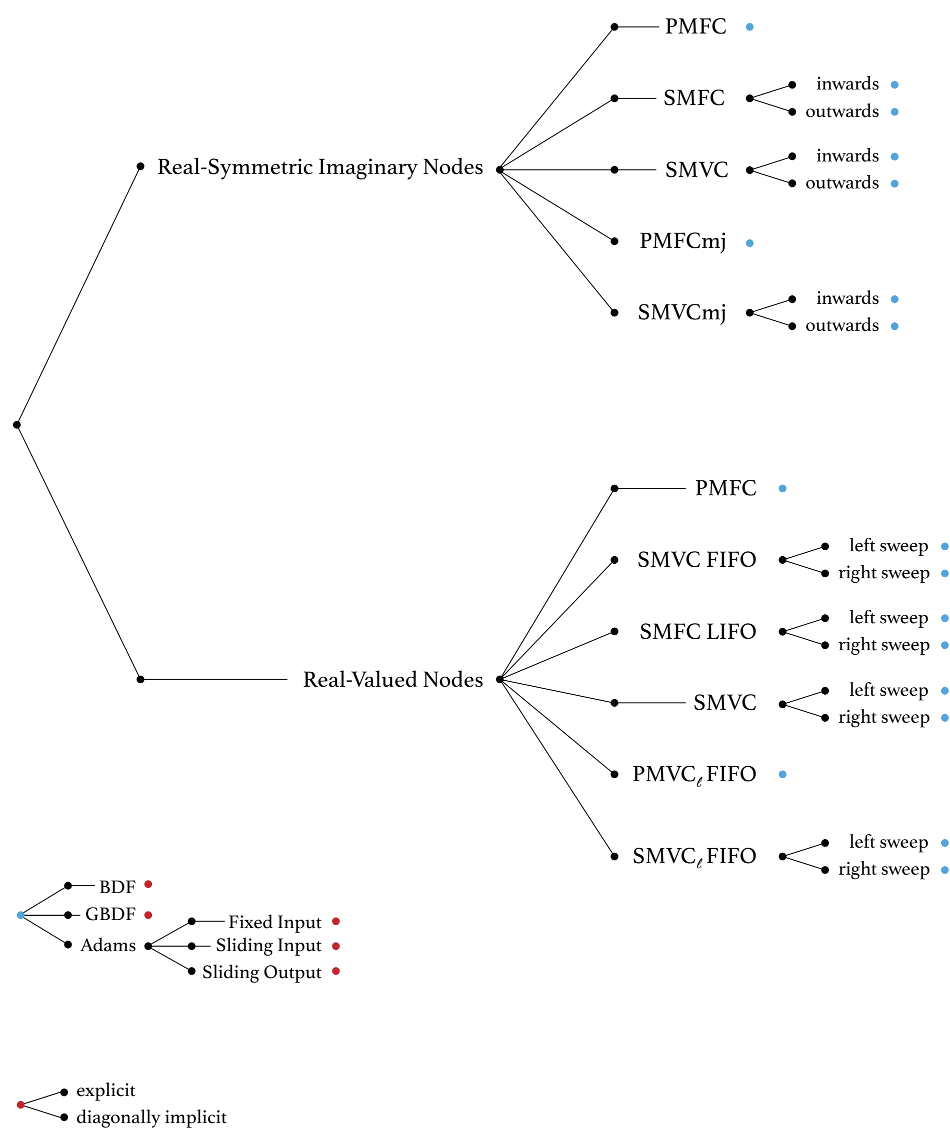


Figure 5.16: A graph displaying the parameter choices listed in this chapter for constructing a polynomial block method. Each blue vertex leads to ten possible method families; in total there 180 families. Instead of trees representing order conditions, we now find ourselves with trees of methods.

5.4.1 The Methods List

The following pages contain diagrams and descriptions for a collection of five polynomial block methods with imaginary nodes. Each method can be implemented at any order and with any node set that belongs to the appropriate node family. The name, along with a brief description of each block method, is provided here for convenience. These particular five methods were selected because they each posses an advantageous property. We describe each of these properties in the method descriptions.

1. **IB-BDF-PMFC**

   A parallel method with imaginary nodes based on BDF. When implemented in parallel, this scheme offers improved stability in comparison to BDF without increasing computational time. It should never be run in serial, since there are other more efficient serial methods.

2. **IB-BDF-SMFC**

   A serial method with imaginary nodes based on BDF. This method has improved error properties in comparison to other imaginary schemes. At orders up to eight this method is A-stable for sufficiently small α.

3. **IB-GBDF-SMVC**

   A serial method with imaginary nodes. This method has improved error properties in comparison to other imaginary schemes. At orders up to eight this method is A-stable for sufficiently small α.

4. **IB-A-PMFC-SI**

   A parallel method with imaginary nodes and based on the Adams-Moulton. When implemented in parallel this scheme offers improved stability in comparison to Adams-Moulton without increasing computational time. The stability regions for this method are bounded, thus this method should be avoided when solving very stiff equations.

5. **IB-A-PMFCmj-SI**

   A parallel Adams method with imaginary nodes. This scheme offers unbounded regions and can be used to solve stiff dissipative equations.

Each method is presented on a separate page.

1. **A Parallel BDF PBM with imaginary nodes**

   Name: IB-BDF-PMFC

   Node Family: real-symmetric imaginary.

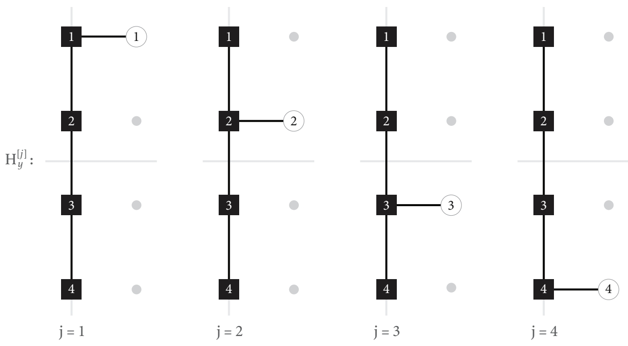
   Ordering: classical.

   Active Index Sets: PMFC.

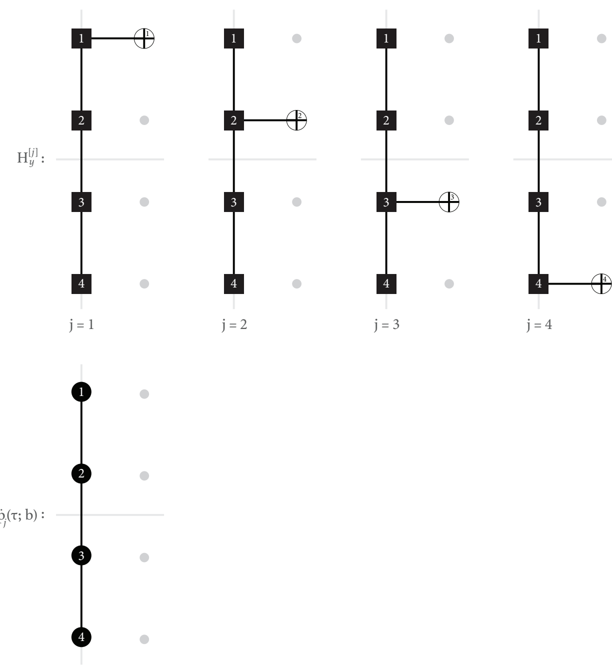
   Method Type: BDF.

   Order of Accuracy: q (diagonally implicit or explicit).

   Diagonally-Implicit Method Diagram (q = 4):



   Explicit Method Diagram (q = 4):



2. **A Serial BDF PBM with imaginary nodes**

Name: IB-BDF-SMFC

Node Family: real-symmetric imaginary.

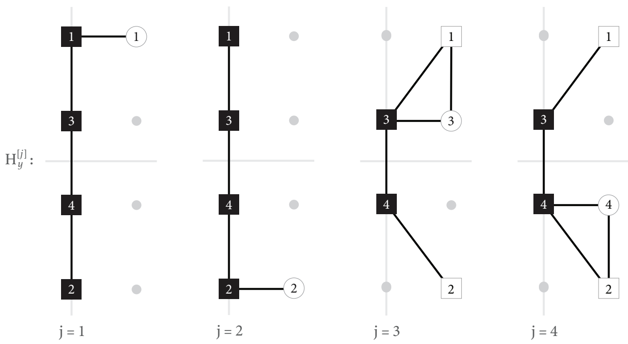
Ordering: inwards sweeping.

Active Index Sets: SMFC.

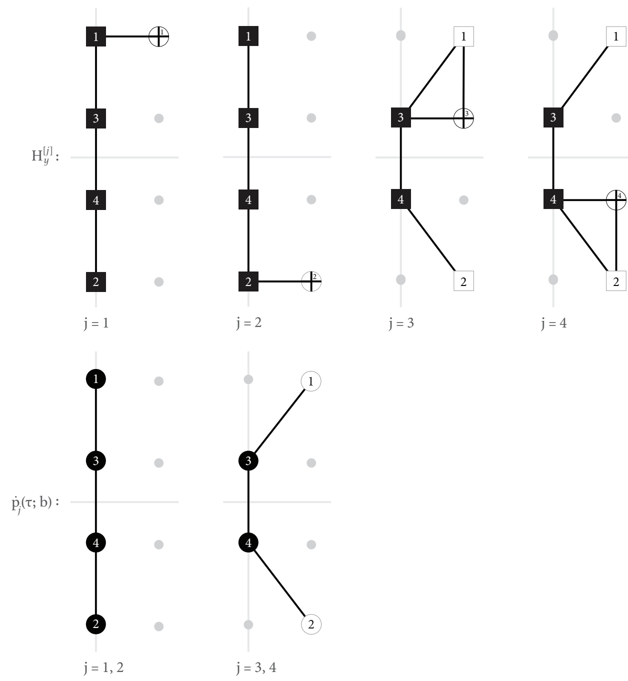
Method Type: BDF.

Order of Accuracy: q (diagonally implicit or explicit).

Diagonally-Implicit Method Diagram ($q = 4$):



Explicit Method Diagram ($q = 4$):



3. **A Serial GBDF PBM with imaginary nodes**

Name: IB-GBDF-SMVC

Node Family: real-symmetric imaginary.

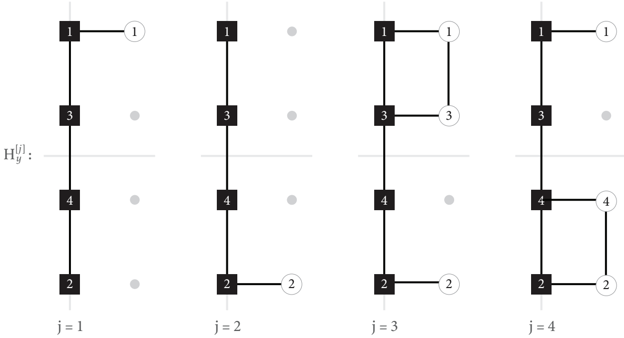
Ordering: inwards sweeping.

Active Index Sets: SMVC.

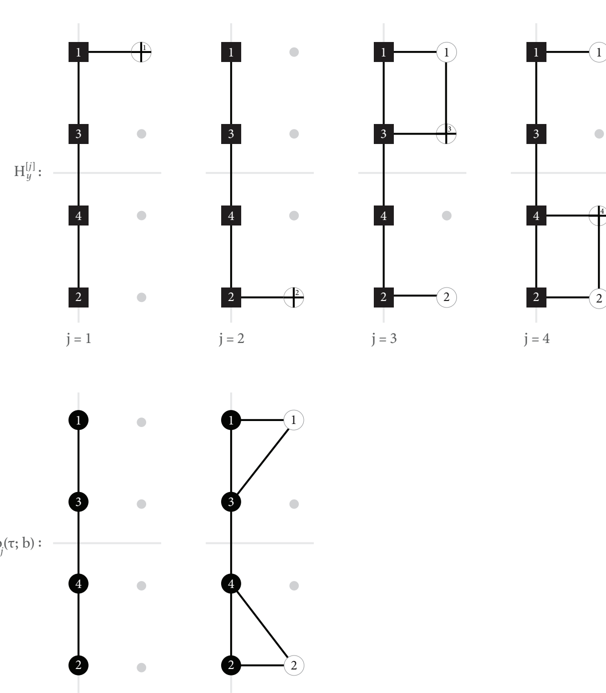
Method Type: GBDF.

Order of Accuracy: q (diagonally implicit or explicit).

Diagonally-Implicit Method Diagram ($q = 4$):



Explicit Method Diagram ($q = 4$):



4. **A Parallel Adams PBM with imaginary nodes**

Name: IB-A-PMFC-SI

Node Family: real-symmetric imaginary.

Ordering: classical.

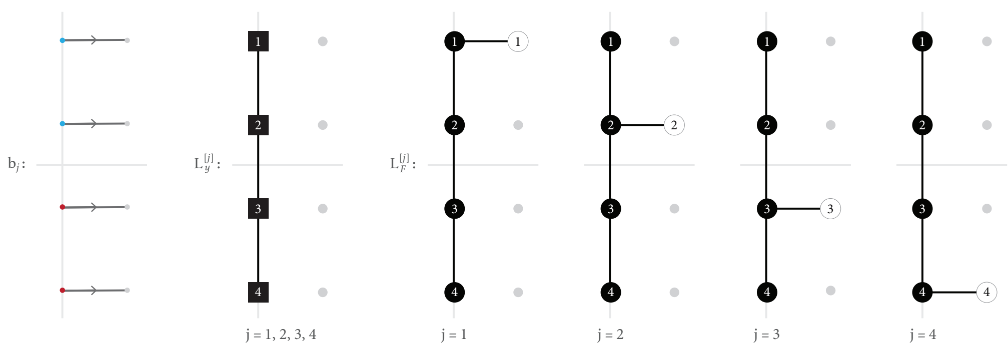
Active Index Sets: PMFC.

Method Type: Adams.

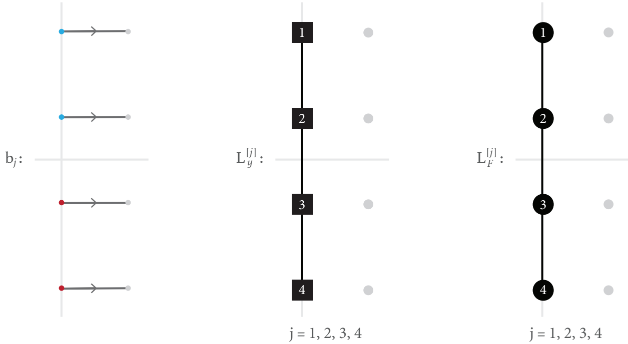
Endpoints: Sliding Input.

Order of Accuracy: $q + 1$ (diagonally implicit) and q (explicit).

Diagonally-Implicit Method Diagram ($q = 4$):



Explicit Method Diagram ($q = 4$):



5. **A Parallel Adams PBM with imaginary nodes**

Name: IB-A-PMFCmj-SI

Node Family: real-symmetric imaginary.

Ordering: classical.

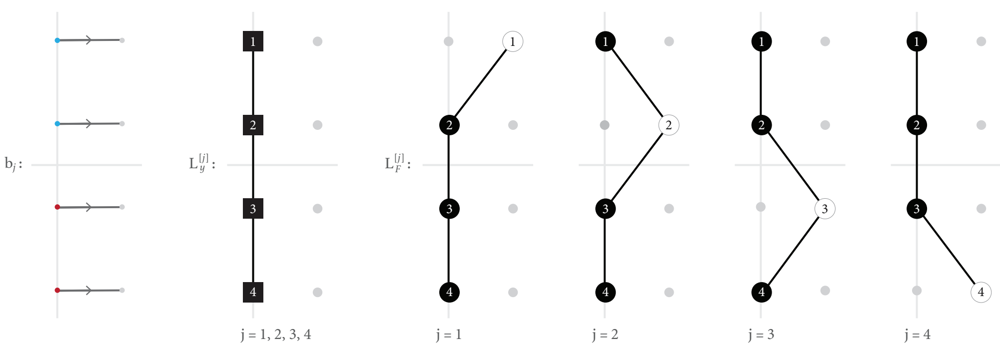
Active Index Sets: PMFCmj.

Method Type: ADAMS.

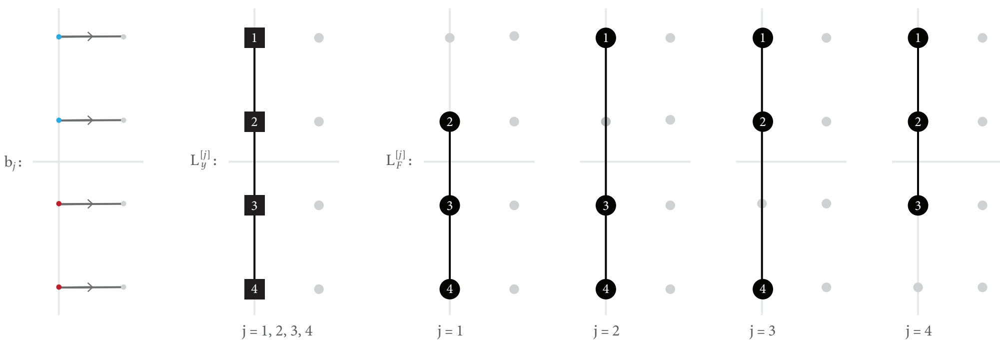
Endpoints: Sliding Input.

Order of Accuracy: $q+1$ (diagonally implicit) and q (explicit).

Diagonally-Implicit Method Diagram (q = 4):



Explicit Method Diagram (q = 4):



Chapter 6

CONSTRUCTING POLYNOMIAL GENERAL LINEAR METHODS

Recall that general linear methods [13, 14, 16], which we first discussed in section 4.3, are a large class of multistage, multivalue integrators that encompass Runge-Kutta methods, linear multistep methods, and predictor-correctors methods. There are three primary reasons why classical general linear methods are difficult to derive. First, the space of all GLMs is extremely large and there is no metric for rapidly identifying promising families of integrators. Second, the inputs for GLMs are expressed using elementary differentials. This generalization increases the number of total parameters, and requires the derivation of a specialized Runge-Kutta integrator for producing the initial values. Finally, the large number of nonlinear order conditions require the use of simplifying assumptions in order to make method derivation feasible.

Polynomial GLMs are simpler to derive because they do not suffer from many of these drawbacks. The inputs for a PGLM are expressed using full derivatives, eliminating the need for a special starting method, and enabling one to implicitly satisfy order conditions by construting ODE polynomials. In short, to construct a PGLM one must choose two sets of nodes (one for inputs and one for stages), a set of ODE polynomials, and a set of expansion points. Though this is considerably simpler than the procedure required for constructing classical GLMs, it still leaves us with a vast sea of possibilities.

To remedy this problem, we present a strategy for deriving PGLMs that builds on the approach introduced in the previous chapter. Our proposed methodology is not affected by any of challenges describe above, and can be used to derive new PGLMs with improved stability, accuracy and computational efficiency in comparison to PBMs. Our approach uses method composition to construct PGLMs from any number of polynomial block methods, polynomial block refiners, or polynomial block coarseners. By composing propagators with specially constructed iterators, we can also derive composite methods that sequentially improve the accuracy of their stage and output values. This technique allows for the derivation of polynomial Runge-Kutta methods and polynomial iterators that improve the accuracy of their inputs by more than one order.

We begin by defining method composition, and by discussing a simple technique for producing polynomial iterators that improve the order of accuracy of their inputs. We then demonstrate the simplicity and effectiveness of our construction strategy by deriving an Adams PGLM through the composition of three block methods.

6.1 *The Composition Product*

The composition product has proven invaluable for both analyzing and deriving many types of time-integrators. For Runge-Kutta methods it provides the group law and is used to derive methods with effective order; for GLMs, the composition product is used to express the order of accuracy of a method [17]. For any two classical time-integrators the composition product can be defined broadly as follows.

Definition 6.1.1 (Composition Product)**.** *Let M_1 and M_2 be two time-integration methods that accept q inputs and produce q outputs. The composition product of M_1 and M_2 produces the composite method*

$$M_3 = M_2 \circ M_1$$

where M_3 computes its outputs by first taking a step with M_1 and then using this new output to take a second step with M_2.

By defining the composition rule for polynomial GLMs, we can turn this theoretical concept into a powerful tool for method construction.

6.1.1 The Composition Product For Polynomial Integrators

We start by specifying which polynomial integrators can be composed (we will call such methods *composable*) before providing a definition for the composition product. In this subsection the words polynomial integrator should be taken to mean either a polynomial method, a polynomial refiner, or a polynomial coarsener. We will use the notation $M(r, \alpha)$ to denote a polynomial integrator M characterized by the node radius r and the extrapolation factor α.

Definition 6.1.2 (Composable Methods)**.** *We say that the polynomial integrator $M_1(r, \alpha)$ is composable with the polynomial integrator $M_2(r, \alpha)$ if the input nodes of M_2 are equal to the output nodes of M_1.*

Definition 6.1.3 (Composition Product for Polynomial Integrators)**.** *Let $M_1(r, \alpha)$ be composable with $M_2(r, \alpha)$. The composition product of M_1 and M_2 produces the composite integrator*

$$M_3(r, \alpha) = M_2(r, (1-d)\alpha) \circ M_1(r, d\alpha) \qquad \text{for any } d \in \mathbb{R}.$$

where $M_3(r, \alpha)$ computes its outputs by first taking a step with $M_1(r, d\alpha)$ and then using this new output to take a second step with $M_2(r, (1-d)\alpha)$. The method M_3 accepts the same number of inputs as M_1 and produce the same number of outputs as M_2. The constant d insures that the stepsize of the composite method $M_3(r, \alpha)$ is always $h = r\alpha$.

The composition product can be used to combine any number of methods. For example, if the integrator M_j is composable with M_{j+1} for $j = 1, \ldots, n-1$, then we may form the composite product

$$M_{n+1}(r, \alpha) = M_n(r, d_n\alpha) \circ M_{n-1}(r, d_{n-1}\alpha) \circ \ldots \circ M_1(r, d_1\alpha),$$

where d_j, $j = 1, \ldots, n-1$, are free and $d_n = \sum_{j=1}^{n-1}(1 - d_j)$. The following shorthand can be used to write this product more conveniently:

$$M_{n+1} = M_n \circ M_{n-1} \circ \ldots \circ M_1 \qquad\qquad \{d_{n-1}, \ldots, d_1\}.$$

6.1.1.1 Closure Under Composition for PGLMs

The set of PGLMs is closed under composition. We can show this by taking two PGLM integrators $M_1(r, \alpha)$ and $M_2(r, \alpha)$, where M_1 is composable with M_2, and rewriting the composite method $M_3(r, \alpha) = M_1(r, (1 - d)\alpha) \circ M_2(r, d\alpha)$ as a PGLM integrator.

Let the integrators $M_1(r, \alpha)$ and $M_2(r, \alpha)$ be defined as

$$M_1(r, \alpha) \qquad \begin{cases} Y_j = p_j(c_j(\alpha);\ b_j), & j = 1, \ldots, s, \\ y_j^{[n+1]} = p_{j+s}(z_j + \alpha;\ b_{j+s}), & j = 1, \ldots, q, \end{cases}$$

$$M_2(r, \alpha) \qquad \begin{cases} Y_j = \widehat{p}_j(c_j(\alpha);\ \widehat{b}_j), & j = 1, \ldots, \widehat{s}, \\ y_j^{[n+1]} = \widehat{p}_{j+\widehat{s}}(\widehat{z}_j + \alpha;\ \widehat{b}_{j+\widehat{s}}), & j = 1, \ldots, \widehat{q}, \end{cases}.$$

The composite method $M_3(r, \alpha)$ can also be written as a PGLM with input nodes equal to those of M_1, output nodes equal to those of M_2, and a total of $s + q + \widehat{s}$ stage values with stage nodes

$$\{c_j(\alpha)\}_{j=1}^{s} \cup \{z_j + d\alpha\}_{j=1}^{q} \cup \{c_j((1 - d)\alpha) + d\alpha\}_{j=1}^{\widehat{s}}.$$

The stages values can be computed as follows:

$$M_3(r, \alpha) \quad \begin{cases} Y_j = \begin{cases} p_j(c_j(d\alpha); b_j) & j = 1, \ldots, s \\ p_j(z_{j-s} + d\alpha; b_{j-s}) & j = s + 1, \ldots, s + q \\ \widehat{p}_{j-s-q}(\widehat{c}_{j-s-q}((1 - d)\alpha); \widehat{b}_{j-s-q}) & j = s + q + 1, \ldots, s + q + 1 + \hat{s} \end{cases} \\ y_j^{[n+1]} = \widehat{p}_{j+\widehat{s}}(\widehat{z}_j + (1 - d)\alpha;\ \widehat{b}_{j+\widehat{s}}), \qquad j = 1, \ldots, \widehat{q}, \end{cases}.$$

Remark 6.1.1. *Since all polynomial integrators can be written as a polynomial GLMs, it follows that composition does not create any new types of methods. If we compose PGLMs, then we create new PGLMS. If we compose other types of polynomial integrators, then we obtain a PGLM. In this way, the composition rule is a tool for constructing PGLMs.*

6.2 Iterators For Improving Accuracy

By using the composition product we can easily derive methods with higher orders of accuracy than their individual parts. In this section we address the construction of polynomial iterators (recall that all iterators are characterized by $\alpha = 0$) that map an input vector with at least one high-order input to an output vector with more than one high-order output. This simple idea is found in many existing time integration methods; for example, consider a Runge-Kutta method with s stages that possesses stage order k and order $p > k$. This method accepts a single high-order input and produces $s-1$ new values of order k and one value of order p.

Here we generalize this concept for polynomial methods. We present a single theorem and two subsequent corollaries. The theorem demonstrates the existence of ODE polynomials for improving the accuracy of dataset values using a construction based proof. We then use this result to prove the existence of polynomial iterators that produce high-order outputs.

Theorem 6.2.1 (ODE Polynomials For Iterators)**.** *Let $D(r, s)$ be an ODE dataset of size w defined as*

$$D(r,s) = \{(\tau_j, y_j, rf_j)\}_{j=1}^{w}$$

where the order of accuracy of each element with respect to r is given by

$$\{\rho_1, \ldots, \rho_w\} = \text{order}(D(r,s); r), \qquad \textit{and} \qquad \gamma = \min_j \rho_j$$

If there exists at least one index k such that $\rho_k > \gamma$ and if $w \geq \gamma$, then we can construct an ODE solution polynomial that is order $\gamma + 1$ accurate with respect to r.

Proof. We will demonstrate this result using a construction-based proof. First let us isolate all data elements satisfying the following properties

$$\begin{cases} \text{all solution values } y_k \text{ where } \rho_k > \gamma \\ \text{all derivative values } rf_k \text{ where } \rho_k \geq \gamma \end{cases}. \tag{6.1}$$

Since $w \geq \gamma$, the total number of such data elements is bounded below by $\gamma + 1$.

Next, let us define any number (including zero) of implicitly defined values

$$(\hat{\tau}_j, \hat{y}_j, r\hat{f}_j) \qquad j = 1, \ldots, l$$

where each $\hat{y}_j$ is formed from an ODE solution polynomial that: (1) is created using any subset of the data elements (6.1), and (2) whose truncation error order with respect to r is at least $\gamma + 1$. Since there are at least $\gamma + 1$ total data elements in (6.1), it will always be possible to construct implicit values with the requisite truncation error.

Finally, let $p(\tau; b)$ be an ODE solution polynomial of degree $d > \gamma$ whose approximate derivatives $a_j(b)$, $j = 1, \ldots, d$, are constructed by differentiating interpolating polynomials

that pass through at least $\gamma+1$ of the following data elements

$$\begin{cases} \text{the solution values } y_k \text{ for any } k \text{ such that } \rho_k > \gamma \\ \text{the derivative values } f_k \text{ for any } k \text{ such that } \rho_k \geq \gamma \\ \text{the implicitly defined solution values } \hat{y}_k \\ \text{the implicitly defined solution values } r\hat{f}_k \end{cases}$$

The ODE solution polynomial $p(\tau; b)$ will always exists since: (1) there is always at least one k such that $\rho_k > \gamma$ and (2) there are a total of $2w + l \geq 2\gamma + l \geq \gamma + 1$ data elements.

Since each approximate derivative is constructed using polynomials of degree $\gamma+1$, then the order of the truncation error of $p(\tau; b)$ with respect to r is $\gamma+1$. Finally, since the active solution values y_j, the active derivatives rf_j (See Remark 3.1.1), and any active interpolated values $\hat{y}_j$, $\hat{f}_j$ are each at least order $\gamma+1$ accurate with respect to r, then the order of $p(\tau; b)$ must also be at least $\gamma+1$ accurate. □

Corollary 6.2.1.1 (Iterators for Improving Accuracy)**.** *There exists at least one polynomial iterator $M(r, 0)$ that when applied to inputs satisfying*

$$y_j^{[n]} = y(t_n + rz_j) + \mathcal{O}(r^{\rho_j}) \qquad j = 1, \ldots, q$$

where $\gamma = \min_j \rho_j$ and $q \geq \gamma$, produces the outputs

$$y_j^{[n+1]} = \begin{cases} y_j^{[n]} & \text{if } \rho_j > \gamma \\ y(t_n + rz_j) + \mathcal{O}(r^{\rho_j+1}) & \text{if } \rho_j = \gamma \end{cases} \qquad j = 1, \ldots, q.$$

Proof. Choose the input nodes for $M(r, \alpha)$ to be $\{z_j\}_{j=1}^q$. To compute an output $y_j^{[n+1]}$ where $\rho_j > \gamma$, choose the corresponding ODE solution polynomials for computing the output to be $p(\tau; b) = y_j^{[n]}$. To compute an output $y_j^{[n+1]}$ where $p_j = \gamma$, construct an ODE polynomial from the dataset containing the method's inputs, outputs and stages in a manner that is consistent with the rules described in the proof of Theorem 6.2.1. When $\alpha = 0$ we obtain the desired properties. □

Corollary 6.2.1.2 (Composite Iterators for Improving Accuracy)**.** *There always exists a sequence of composable polynomial iterators $M_1, \ldots, M_\nu$, such that the composite method*

$$M = M_\nu \circ \ldots \circ M_1 \qquad \{0, \ldots, 0\}$$

can be applied to the inputs

$$y_j^{[n]} = y(t_j) + \mathcal{O}(r^{\rho_j}) \qquad j = 1, \ldots, q$$

where $\gamma = \min_j \rho_j$ and $\Gamma = \max_j \rho_j$ to produce the outputs

$$y_j^{[n+1]} = \begin{cases} y_j^{[n]} & \text{if } \rho_j > \gamma \\ y(t_n + rz_j) + \mathcal{O}(r^{\min(q+1,\Gamma)}) & \text{if } \rho_j = \gamma \end{cases} \qquad j = 1, \ldots, q.$$

Proof. Choose the input nodes for all methods $M_k(r, \alpha)$ to be $\{z_j\}_{j=1}^q$, then sequentially apply Corollary 6.2.1.1 to construct the methods. □

6.3 Example Polynomial GLMs

We now briefly discuss polynomial GLMs and introduce a single fourth-order method by using the composition rule. Recall that the computation for a PGLM with q outputs, s stages, and nodes $\{z_j\}_{j=1}^{q}$ is

$$\begin{aligned} Y_j &= p_j(c_j(\alpha),\ b_j), && j = 1, \ldots, s, \\ y_j^{[n+1]} &= p_{j+s}(z_j + \alpha,\ b_{j+s}), && j = 1, \ldots, q, \end{aligned}$$

where each $p_j(x, b)$ is an *ODE polynomial* over the data set (4.10). A simple example of PGLMs with $q = 1$ and $s = 1$, is the class of second order explicit Runge-Kutta methods characterized by

$$\text{nodes: } z_1 = 0, \qquad \text{stages: } c_1 \text{ (free)} \qquad \text{expansion points: } b_1 = b_2 = 0, \qquad \alpha = 1,$$

and the *ODE polynomials*

$$\underbrace{\begin{aligned} p_1(\tau; b) &= y_1^{[n]} + r\int_b^\tau f_1^{[n]} d\xi \\ p_2(\tau; b) &= y_1^{[n]} + r\int_b^\tau \frac{(c_1 - x)}{c_1} f_1^{[n]} + \frac{x}{c_1} F_1 d\xi \end{aligned}}_{\text{explicit coefficient formula}} \equiv \underbrace{\begin{aligned} &L_y^{[1]} : \left[y_1^{[n]}\right] \quad L_F^{[1]} : \left[f_1^{[n]}\right] \\ &L_y^{[2]} : \left[y_1^{[n]}\right] \quad L_F^{[2]} : \left[f_1^{[n]},\ F_1\right] \end{aligned}}_{\text{Adams notation}}$$

If $c_1 = 1/2$ then we obtain the midpoint method, and if $c = 1$ we obtain the Heun method.

6.3.1 A New Fourth-Order Polynomial GLM

We propose a new fourth-order Adams PGLM with $q = 2$ and $s = 4$ that can be derived by composing three Adams block methods. This method, which we will refer to as APGLM4, can be written as

$$\text{APGLM4}(r, \alpha) = M_1(r, \alpha) \circ M_2(r, 0) \circ M_3(r, 0). \tag{6.2}$$

where each $M_k(r, \alpha)$ is a PBM. The intuition for constructing APGLM4 is as follows: 1) M_1 is a refiner that advances the initial solution using an implicit method from Subsection 5.4 while also retaining the original inputs, 2) M_2 is an iterator that improves the order of accuracy of the new solution values by one, 3) M_3 is a coarsener that discards all initial values leaving only the new ones. We describe each of these three block methods below:

- $M_1(r, \alpha_1)$: *diagonally implicit block refiner*

  nodes: $q = 2, \quad m = 4, \quad \{z^{\text{in}}\} = \{-i,\ i\}, \quad \{z^{\text{out}}\} = \{-i,\ i,\ -i + \alpha_1,\ i + \alpha_1\}$

  endpoints: $\{b\} = \{-i,\ i,\ -i,\ i\}$

  Adams polynomials: $L_y^{[j]} : \left[y_1^{[\text{in}]},\ y_2^{[\text{in}]}\right] \qquad L_F^{[j]} : \begin{cases} \left[f_1^{[\text{in}]},\ f_2^{[\text{in}]}\right] & j = 1, 2 \\ \left[f_1^{[\text{in}]},\ f_2^{[\text{in}]},\ f_j^{[\text{out}]}\right] & j = 3, 4 \end{cases}$

This method sets the first two outputs equal to the inputs ($y_j^{\text{out}} = y_j^{\text{in}}$ for $j = 1, 2$), and computes the remaining outputs identically to the implicit IB-A-PMFC-SI method with $q = 2$ from Section 5.4.

- $M_2(r, \alpha_2)$: *diagonally implicit block method*

nodes:	$q = 4, \quad \{z\} = \{-i,\ i,\ -i+\alpha_1,\ i+\alpha_1\}$
endpoints:	$\{b\} = \{-i,\ i,\ -i,\ i\}$
Adams polynomials:	$L_y^{[j]} : \left[y_1^{[n]},\ y_2^{[n]}\right]$
	$L_F^{[j]} : [\gamma_1(j),\ \gamma_2(j),\ \gamma_3(j), \gamma_4(j)] \qquad \gamma_i(j) = \begin{cases} f_i^{[n+1]} & i = j \\ f_i^{[n]} & i \neq j \end{cases}$

This method computes the jth output using $f_j^{[n+1]}$ at node j and $f^{[n]}$ at all other nodes. When $\alpha_2 = 0$, the first two outputs will be equal to the first two inputs (i.e. $y_j^{[\text{n}+1]} = y_j^{[\text{n}]}$ for $j = 1, 2$).

- $M_3(r, \alpha_3)$: *parallel explicit block coarsener*

nodes:	$q = 4, \quad m = 2, \quad \begin{array}{l} \left\{z^{\text{in}}\right\} = \{-i,\ i,\ -i+\alpha_1,\ i+\alpha_1\} \\ \left\{z^{\text{out}}\right\} = \{-i+\alpha_1+\alpha_3,\ i+\alpha_1+\alpha_3\} \end{array}$
endpoints:	$\{b\} = \{-i+\alpha_1,\ i+\alpha_1\}$
Adams polynomials:	$L_y^{[j]} : \left[y_3^{[\text{in}]},\ y_4^{[\text{in}]}\right] \qquad L_F^{[j]} : [\ ]$

This method discards the first two inputs and produces outputs equal to the last two inputs ($y_j^{\text{out}} = y_{j+2}^{\text{in}}$ for $j = 1, 2$). This method is exact if $\alpha_3 = 0$.

APGLM4 is a serial PGLM with parallel stages since M_1, M_2, and M_3 are each parallel methods. If the ODE is real-valued, then we may apply the Swartz reflection principle to reduce APGLM4 to a serial method that requires two sequential, nonlinear solves per time-step. We may also express APGLM4 compactly in the form (4.9) with $q = 2$ and $s = 4$, by choosing

nodes:	$\{z\}_{j=1}^{q} = \{-i,\ i\}$
stages:	$\{c\}_{j=1}^{s} = \{-i+\alpha,\ i+\alpha,\ -i+\alpha,\ i+\alpha\}$
expansion points:	$\{b\}_{j=1}^{q+s} = \{-i,\ i,\ -i,\ i,\ -i+\alpha,\ i+\alpha\}$

along with the Adams polynomials

$$L_y^{[j]} : \begin{cases} \left[y_1^{[n]},\ y_2^{[n]}\right] & j = 1,2,3,4 \\ [Y_3,\ Y_4] & j = 5,6 \end{cases} \qquad L_F^{[j]} : \begin{cases} \left[f_1^{[n]},\ f_2^{[n]},\ F_j\right] & j = 1,2 \\ \left[f_1^{[n]},\ f_2^{[n]},\ F_3,\ F_2\right] & j = 3 \\ \left[f_1^{[n]},\ f_2^{[n]},\ F_1,\ F_4\right] & j = 4 \\ [] & j = 5,6 \end{cases} .$$

Chapter 7

LINEAR STABILITY

In Chapter 5 we introduced over 180 different families of polynomial block methods, and in Chapter 6 we proposed a fourth order PGLM. In this chapter we investigate the linear stability properties for several of these integrators and compare them with classical BDF and Adams linear multistep methods. When comparing methods we will emphasize the properties of $A(\theta)$ stability, real stability interval, and negative stability interval which were all previously defined in Subsection 4.7.2.

7.0.0.1 Selecting A Set of Methods To Compare

We will only show results for a small subset of the 190 families of methods. In this section we will list the integrators that we propose to test. Since this list is small, we will use a simplified naming convention for these schemes. We briefly discuss each method below then summarize the names and properties of the methods in Table 8.2.

For all diagonally-implicit block methods with imaginary nodes, we will choose the node set consisting of q equispaced points on the imaginary interval $[-i,\ i]$, given by

$$z_j = -i + 2i(j-1)/(q-1) \qquad j = 1, \ldots, q. \tag{7.1}$$

We make this choice since these nodes are merely a rotation of those used to obtain the classical BDF and Adams-Moulton methods. We then consider the following methods:

1. *BBDF* - A parallel, diagonally implicit block BDF method with the construction IB-BDF-PMFC from Subsection 5.4, the node set (7.1), and $q > 1$.

2. *BAM* - A parallel, diagonally implicit block Adams-Moulton method with the construction IB-A-PMFC-SI from Subsection 5.4, the node set (7.1), and $q > 1$.

For diagonally-implicit PGLMS we test the APGLM4 method (6.2) which also has nodes 7.1 for $q = 2$.

For explicit schemes we will make a minor modification to the integrator (2.10) that improves the order of accuracy by one without increasing the total number of function evaluations. This is accomplished by introducing an additional node at $\tau = 0$ and using only data at the roots of unity to form $L_F^{[j]}(\tau)$. We describe this modified method as follows:

3. *BAB* - A parallel, explicit block Adams method of order $q-1$ which for $q>2$ has the parameters

$$\text{ODE Polynomials} \quad L_y^{[j]} : \left[y_1^{[n]},\ y_2^{[n]},\ \ldots,\ y_q^{[n]}\right], \qquad L_F^{[j]} : \left[f_1^{[n]},\ f_2^{[n]},\ \ldots,\ f_{q-1}^{[n]}\right],$$

$$\text{Nodes} \quad z_j = \begin{cases} \exp\left(\frac{2\pi i(j-1)}{q-1}\right) & j<q \\ 0 & j=q \end{cases},$$

$$\text{expansion points} \quad b_j = 0.$$

Short Name	Construction	Properties
BBDF	IB-BDF-PMFC	A diagonally-implicit parallel BDF-PBM of order q using imaginary equispaced nodes.
BAM	IB-A-PMFC-SI	A parallel Adams-PBM of order $q+1$ using imaginary equispaced nodes.
APGLM4	APGLM4	A diagonally-implicit serial Adams type GLM with nodes $\{i,-i\}$.
BAB	n.a.	An explicit parallel Adams-PBM $q+1$ using the roots of unity and zero.

Table 7.1: Simplified subset of polynomial block and GLM integrators that will be tested. The full list of construction stategies can be found in Subsection 5.4

7.1 Linear Stability Results for BBDF, BAM, and BAB schemes

The BBDF, BAM and BAB methods are block generalizations of BDF, Adams-Moulton and Adams-Bashforth. In this section we will compare the stability properties of each of these six schemes to highlight the advantage of polynomial integrators versus classical linear multistep schemes.

For BDF methods of order one through six, $A(\theta)$ stability decreases monotonically, while methods of order greater than six are no longer root stable (See Table 7.2). On the other hand, Adams-Moulton methods have bounded linear stability regions that contract rapidly with order and are often smaller than those of explicit methods (See Table 7.3).

BBDF and BAM methods improve upon these limitations, leading to high-order methods with more favorable stability properties. In comparison to BDF, BBDF schemes satisfy the

root condition past order six, and provide improved $A(\theta)$ stability especially for small α. In Table 7.2 we present $A(\theta)$ stability for BBDF, and in Figures 7.1 and 7.2 we present plots of BBDF stability regions overlaid with those of the classical BDF methods.

order	2	3	4	5	6	7	8
BDF	90°	86.03°	73.35°	51.84°	17.84°		
$\text{BBDF}_{\alpha=1}$	90.00°	89.54°	88.51°	87.58°	86.89°		
$\text{BBDF}_{\alpha=\frac{1}{2}}$	90.00°	89.88°	89.32°	88.51°	87.72°	87.05°	83.58°
$\text{BBDF}_{\alpha=\frac{1}{4}}$	90.00°	89.99°	89.90°	89.68°	89.31°	88.83°	88.33°
$\text{BBDF}_{\alpha=\frac{1}{8}}$	90.00°	89.99°	89.99°	89.98°	89.94°	89.86°	89.75°

Table 7.2: A(θ) stability for classical BDF and BBDF. The table lists θ values for methods of orders two through eight and for BBDF methods with α values of 1, 1/2, 1/4, and 1/8. All angles are rounded to two decimals. Empty positions indicate the underlying method is not root stable.

Like Adams-Moulton methods BAM methods have bounded stability regions. However they do possess significantly larger linear stability regions than their classical counterparts, but do not include the imaginary axis. In Table 7.3 we present real stability intervals for BAM Schemes and in Figure 7.3 we present plots of BAM stability regions overlaid with those of the classical AM methods.

The stability gains for the BBDF and BAM methods are possible because of the extrapolation parameter α. As a general rule we find that smaller α values lead to improved stability for both methods. However, recall that choosing a smaller α requires a greater amount of analyticity relative to the timestep $h = r\alpha$.

Next we consider explicit integrators. High-order Adams-Bashforth methods have small stability regions and are only efficient when high-accuracy solutions are desired. In contrast, the parallel BAB methods have stability regions that converge to a half circle. In situations where each of the q ODE right-hand side evaluations can be evaluated efficiently in parallel, high-order BAB methods will provide significantly improved stability at no additional cost. In Table 7.4 we present real stability intervals for AB and BAB Schemes and in Figure 7.5 we present plots of BAB stability regions and those of the classical AB methods.

order	3	4	5	6	7	8
AM	6.00	3.00	1.84	1.18	0.77	0.49
$\text{BAM}_{\alpha=1}$	58.01	11.66	7.24	5.68	4.81	4.23
$\text{BAM}_{\alpha=\frac{1}{2}}$	202.01	29.66	14.34	9.29	7.21	5.90
$\text{BAM}_{\alpha=\frac{1}{4}}$	778.01	101.67	42.77	23.60	15.94	11.88
$\text{BAM}_{\alpha=\frac{1}{8}}$	3082.01	389.67	156.55	81.17	51.19	35.31

Table 7.3: Negative stability intervals β for AM and BAM schemes. The table lists β values for methods of orders two through eight and for BAM methods with α values of 1, 1/2, 1/4, and 1/8. All values are rounded to two decimals.

order	2	3	4	5	6	7	8
AB	1.00	0.55	0.30	0.16	0.088	0.047	0.024
$\text{BAB}_{\alpha=4}$	1.34	0.75	0.55	0.47	0.44	0.41	0.41
$\text{BAB}_{\alpha=2}$	1.01	0.71	0.54	0.47	0.44	0.41	0.41
$\text{BAB}_{\alpha=1}$	0.67	0.61	0.52	0.47	0.44	0.41	0.41

Table 7.4: Negative real stability intervals β values for AB and BAB schemes. The table lists β values for methods of orders two through eight and for BAM methods with α values of 4, 2, 1/4, and 1. All values are rounded to two decimals.

7.2 *Linear Stability Results for APGLM4*

In Figure 7.4 we present the linear stability regions for APGLM4 for several values of the extrapolation factor α. It is clear from the stability plots that APGLM4 has a comparable stability region to the BBDF methods. The stability regions of APGLM4 are also less sensitive to changes in α than those of the BBDF method, and stability at $\text{Re}(z) = -\infty$ is preserved even for α as large as four. Using a large α requires less analyticity per stepsize, which is useful for solving systems with limited analyticity off the real line.

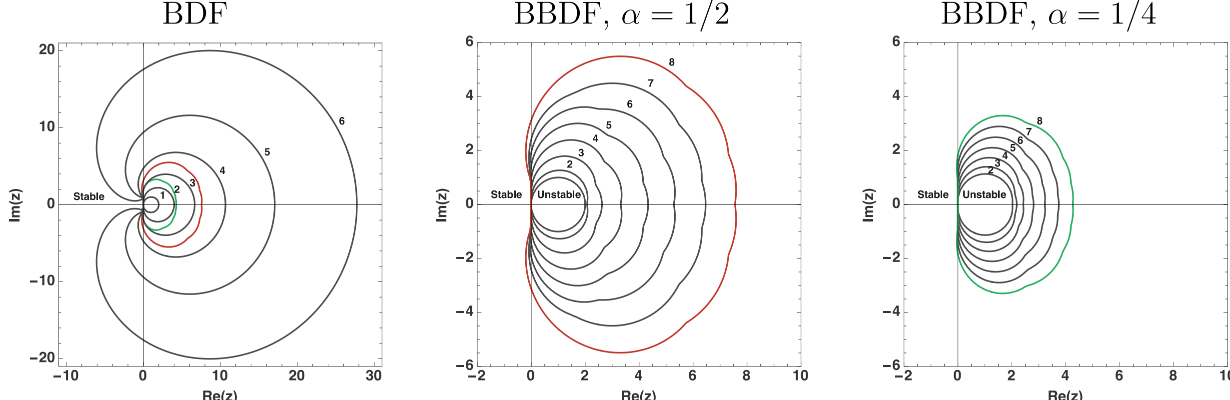


Figure 7.1: Stability regions for BDF methods of order two through six and block BDF (BBDF) methods of orders two through eight with rotated equispaced nodes. The stability boundary for eighth order BBDF with $\alpha = 1/2$ and $\alpha = 1/4$ are respectively labeled in red and green. Decreasing the extrapolation parameter α, consistently improves $A(\theta)$ stability for BBDF schemes.

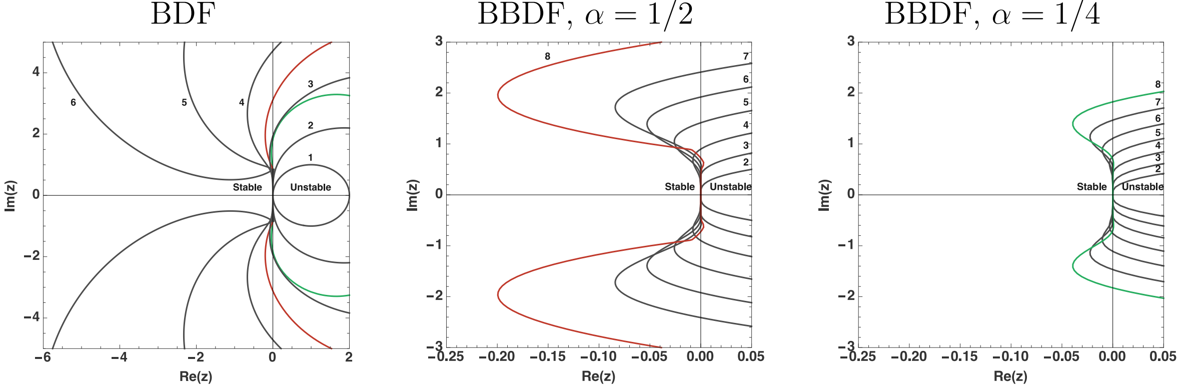


Figure 7.2: Magnification of Figure 7.1 showing stability regions near the imaginary axis.

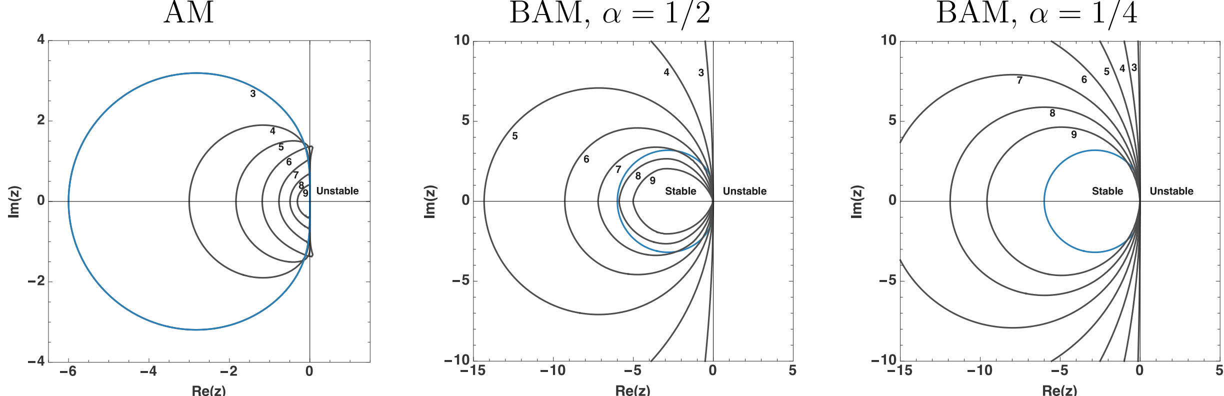


Figure 7.3: Stability regions for Adams-Moulton (AM) and block Adams-Moulton (BAM) of orders three through nine. The stability boundary for third-order Adams-Moulton appears as a blue contour in all figures. Block methods have significantly larger stability regions but do not include any part of the imaginary axis. Smaller α leads to larger and more circular stability regions.

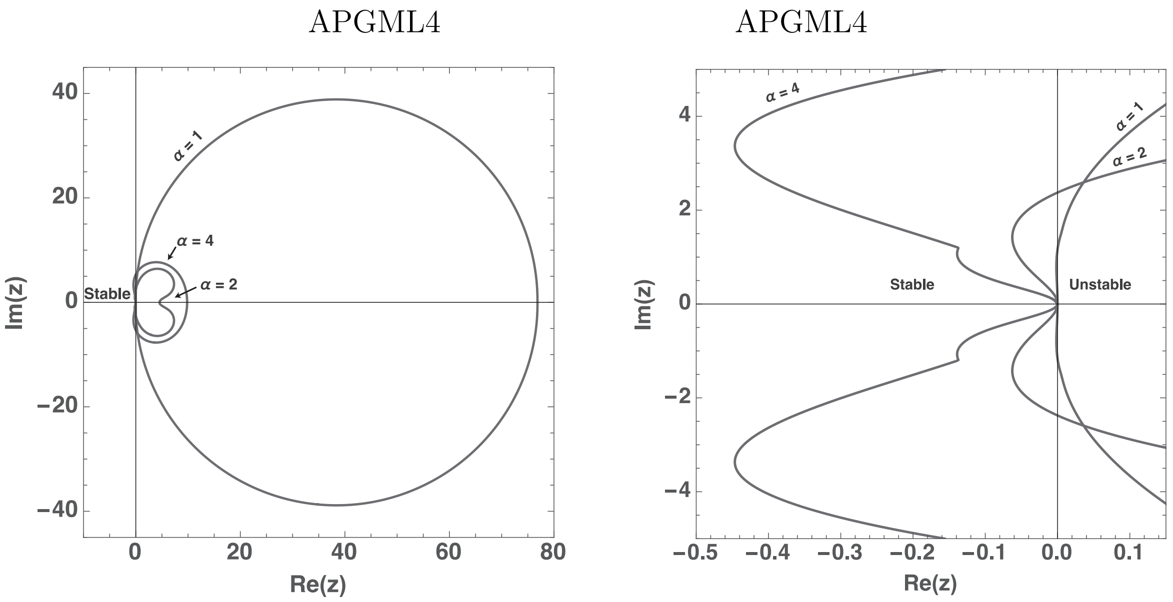


Figure 7.4: Left: stability regions for APGML4 for $\alpha = 1$, 2, and 4. Right: magnified stability regions around imaginary axis. The imaginary axis is never fully included in the stability region for any of these α values.

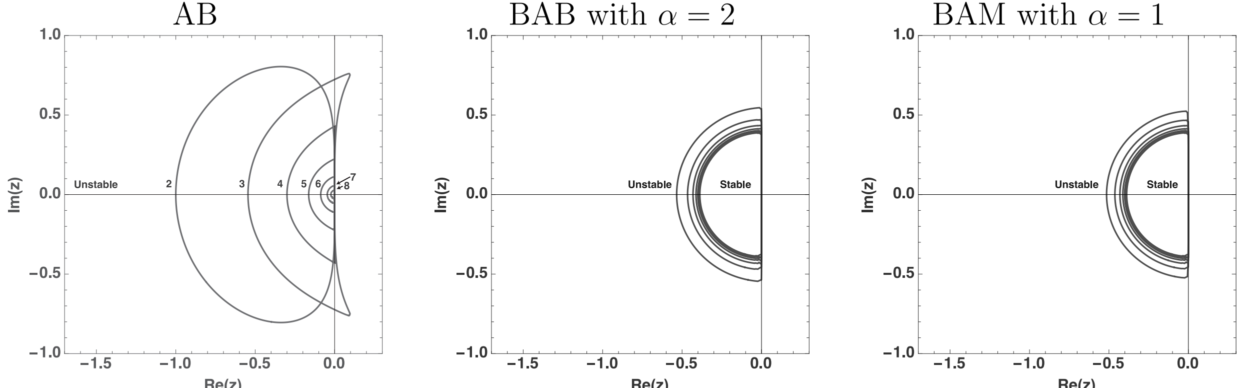


Figure 7.5: Stability regions for Adams-Bashforth (AB) and block Adams-Bashforth (BAB) schemes of orders two through eight. The stability regions for BAB methods monotonically decrease and appear to converge to a half circle, while the stability regions for AB converge to zero. For both $\alpha = 1$ and $\alpha = 1/2$ BAB methods of orders 4, 5, and 8 are stable along a non-empty interval of the imaginary axis. Changing α has significantly less effect on stability regions then was the case for implicit methods.

Chapter 8

NUMERICAL EXPERIMENTS

In this chapter, we present numerical experiments to validate the BBDF, BAM, BAB, and APGLM schemes described in Table (8.2). Our numerical experiments address three points. First we demonstrate the improved accuracy and stability properties of BBDF, BAM, and BAB in comparison to classical BDF, Adams-Moulton and Adams-Bashforth. Second we show the improved error properties of APGLM4 in comparison to BBDF methods and its improved stability in comparison to BAM methods. Finally we numerically investigate the effects of the extrapolation factor α on the accuracy of BBDF and BAM.

We compare implicit methods on two stiff partial differential equations, and explicit methods on an ordinary differential equation. We present plots of absolute error vs. stepsize, and absolute error vs. parallel computational time. For each equation, we compute the reference solutions using MATLAB's *ode15s* integrator with a tolerance of 1e-14. Inputs for all multivalue methods are computed at the initial time-step using the exponential integrator EPIRK43a [82]. For implicit methods, we solve all nonlinear systems using Newton's method with an exact Jacobian, and MATLAB's backslash to solve the underlying linear systems. All results presented in this chapter have been run on a single thread using a 3.5 Ghz Intel i7 Processor.

Measuring Computational Time for Parallel Schemes

For parallel, diagonally implicit methods, the q nonlinear systems can be solved simultaneously at each time-step. For parallel, explicit methods, the q right-hand-side evaluations can be computed in parallel. If these methods are run on a distributed memory system, it is necessary to communicate all solution values between the nodes after each nonlinear solve or function evaluation. If the communication cost is high, then optimal parallel efficiency will not be obtained. On a shared memory system communication is not necessary, but frequent cache misses may degrade parallel performance.

In general, the efficiency for any parallelized code will depend on the problem size and computational architecture. To avoid this technicality, we present our timing results in terms of *optimal parallel time* that ignores communication overhead and assumes perfect parallelization.

The *optimal parallel time* for a parallel diagonally implicit integrator is the sum of the slowest nonlinear system solve and all additional processing. We illustrate this with the following pseudocode for a single time-step:

```
for j = 1 ... q
```

```
     solve jth nonlinear system on processor j; % let running time be τ_{j,0}
end
process results on processor 1; % let running time be τ_{1,1}
```

The *optimal parallel time* for this single step is $\tau_{1,1}+\max(\tau_{1,0},\ldots,\tau_{q,0})$. Similarly, the *optimal parallel time* for parallel explicit methods is the sum of the slowest function evaluation and all additional processing. In short, our timing results for parallel integrators are representative of situations where communication overhead is insignificant compared to the nonlinear solve times or to the function evaluation times.

Computing Outputs At Real-Valued Times

When q is even, BBDF and BAM methods have no real-valued nodes. We must therefore compute an additional output at the local coordinate $\tau = \alpha$ during the final time-step, or at any time-step where we wish to obtain the solution at a real time point. This can be accomplished by forming and evaluating a new *ODE polynomial* $p_{\text{out}}(\tau; b)$ so that

$$y_{\text{out}}^{[n+1]} = p_{\text{out}}(\alpha; b_{\text{out}}).$$

A possible choice for the output polynomials for BBDF and BAM are:

1. $BBDF_{\text{out}}$ $\qquad H_F^{[j]} : \left[y_1^{[\text{n}]},\ldots,y_q^{[\text{n}]},\ f_{\text{out}}^{[\text{n}+1]}\right]$

2. BAM_{out} $\qquad \begin{cases} L_y^{[j]} : \left[y_1^{[\text{n}]},\ldots,y_q^{[\text{n}]}\right], \quad L_F^{[j]} : \left[f_1^{[n]},\ldots,\ f_q^{[n]}, f_{\text{out}}^{[n+1]}\right] \\ b_{\text{out}} = z_{q/2}. \end{cases}$

Computing Conjugate Outputs

The nodes $\{z_j\}_{j=1}^q$ and endpoints $\{b_j\}_{j=1}^q$ for BBDF, and BAM are symmetric with respect to the real line. If the underlying ODE is real-valued then the Swartz reflection principle implies that the solution $y(t)$ and its derivative $F(t, y(t))$ satisfy the relations

$$y(z^*) = y(z)^* \qquad \text{and} \qquad F(z^*, y(z^*)) = F(z, y(z))^*.$$

where * denotes complex conjugate. For BBDF and BAM methods, this implies that we only need to compute the first $\lceil q/2 \rceil$ outputs and function evaluations, since it follows that

$$y_j^{[n+1]} = \left(y_{q-j+1}^{[n+1]}\right)^* \qquad \text{and} \qquad f_j^{[n+1]} = \left(f_{q-j+1}^{[n+1]}\right)^* \qquad \text{for} \qquad j = 1,\ldots \lfloor q/2 \rfloor.$$

This simplification eliminates redundant computation, but does not reduce the parallel time of the method since each of the q systems can be solved independently. However, this

simplification allows us to double the order of our methods without requiring additional computational nodes.

When implementing these integrators on real-valued problems, it is important to clip any imaginary rounding errors at each time-step from all outputs that lie on the real time-line. If the rounding errors are not removed, we find that the method can become unstable.

8.1 Test Problems

Below we describe two stiff partial differential equations, and one non-stiff ordinary differential equation used for our numerical experiments. For each partial differential equation, we discretize space using standard second-order finite differences.

Stiff Partial Differential Equations

The **Viscous Burgers Equation** is a classical partial differential equation appearing in fluid dynamics. We consider the one dimensional Burgers equation with homogenous boundary conditions [92],

$$\begin{aligned} & u_t = \nu u_{xx} - u u_x, \qquad (8.1)\\ & u(x, t=0) = (\sin(3\pi x))^2 (1-x)^{3/2}, \\ & x \in [0,1], \quad t \in [0,1]. \end{aligned}$$

where we take $\nu = 3 \times 10^{-4}$. We discretize space using a 2000 point grid, and test all time-integrators using fifteen different stepsizes logarithmically spaced between 5×10^{-3} and 5×10^{-4}.

The **Advection-Reaction-Diffusion** (ADR) equation can be used to model the evolution of a substance in a medium. We consider the two dimension ADR equation with homogeneous Neumann boundary conditions [82],

$$\begin{aligned} & u_t = \epsilon (u_{xx} + u_{yy}) + \delta (u_x + u_y) + \gamma u(u - 1/2)(1-u), \qquad (8.2)\\ & u(x, t=0) = 256(xy(1-x)(1-y))^2 + 0.3 \\ & x, y \in [0,1], \quad t \in [0, 0.05] \end{aligned}$$

where we take $\epsilon = 1/100$, $\delta = -10$ and $\gamma = 100$. We discretize space using a 400×400 point grid, and test all time-integrators using fifteen different stepsizes logarithmically spaced between 1×10^{-3} and 1×10^{-4}.

Non-Stiff Ordinary Differential Equations

The **Van der Pol** oscillator is a simple ordinary differential equation describing certain

electrical circuits. We consider a transformed Van der Pol oscillator from [42] given by

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \left(\left(1 - y_1^2\right) y_2 - y_1 \right) / \epsilon \\ y_1(0) &= 2, \quad y_2(0) = 0, \quad t \in [0, 1] \end{aligned} \tag{8.3}$$

where $\epsilon = 0.1$. This particular choice of ϵ leads to a non-stiff problem that can be used to test explicit methods. We test all time-integrators using thirty different stepsizes logarithmically spaced between 1×10^{-3} and 1×10^{-4}.

8.2 Results for Diagonally Implicit Polynomial Block Methods

We compare the BBDF and BAM methods from Table with an extrapolation parameter of $\alpha = 1/2$, against BDF and Adams-Moulton. In Figures 8.1 and 8.2 we present error vs stepsize and error vs *optimal parallel time* diagrams for Burgers equation and the ADR equation. We draw the following conclusions from our numerical experiments:

1. Regarding Accuracy and Stability: High-order BDF and AM methods were either completely unstable or only stable at smaller stepsizes. BBDF methods were stable at all orders and BAM methods showed improved stability over AM methods of equivalent order. All four classes of methods demonstrated or exceeded their expected orders of accuracy at small stepsizes. When solving viscous Burgers, BBDF and BAM did not converge at the coarsest stepsizes due to insufficient analyticity of the solution. We suspect that analyticity for Burgers equation is inversely proportional to the parameter ν and that all methods will converge for larger timesteps if ν is increased (See for example [8]).

We find that AM and BAM methods have superior error properties compared to BDF and BBDF, making them more efficient so long as they remain stable. Rounding errors presented significant problems for both BBDF8, which was unable to achieve an error below 1e-7, and BAM8, which performed worse than the BAM7 on the Burgers equation. In section 8.2.3, we resolve this issue by choosing a smaller α.

2. Regarding Performance: When implemented in parallel, BBDF and BAM methods with $\alpha = 1/2$ are comparable in efficiency to BDF and AM schemes. The primary reason to consider BBDF and BAM over BDF or AM schemes is their improved stability at high orders of accuracy. For large step-sizes, high-order BDF schemes are unstable for even mildly dispersive PDEs, and AM methods are not appropriate for stiff equations. In contrast, high-order BBDF methods can be applied to a wider range of problems, and BAM methods will outperform BBDF on dissipative PDEs that are not too stiff.

In our implementation, a single step with a BBDF or BAM takes roughly twice as long in optimal parallel time as a single step with BDF or AM. We find that the additional cost is due to the complex-valued linear solves at each Newton iteration. We use MATLAB's mldivide to solve linear systems, however, we expect that for large problems one can match the

performance of BDF of AM by using a linear solver that allows for additional parallelization. For example, one may rewrite the complex linear systems into a real system of twice the dimension, and apply GMRES where the larger matrix multiplications have been parallelized to offset the additional cost.

8.2.1 Results for Diagonally Implicit Polynomial GLM

Here we present numerical experiments to showcase the stability and performance of the APGLM4 method (6.2) with $\alpha = 4$. This method has *ODE polynomials* that pass through solution and derivative data at subsets of the local time points $\tau \in \{-i, i, -i+\alpha, i+\alpha\}$. We compare APGLM4 to BAM3 and BBDF2, since the *ODE polynomials* for all these methods pass through the same temporal nodes. We also compare PGML4 to BDF2 and AM2, which can be interpreted as two polynomial methods with *ODE polynomials* that pass through subsets of the local time points $\tau \in \{-1, 1, -1+\alpha, 1+\alpha\}$. Together, these methods help illustrate the effects on error and stability when choosing different polynomial approximations over similar nodes.

Since the ADR and Burgers' equation are real-valued, we may use the Swartz reflection principle to reduce APGLM4, BBDF2, and BAM3 to serial methods. For this experiment, our timing results now represent actual running time, not optimal parallel time. In Figure 8.3 we present error vs stepsize and error vs computation time for Burgers equation and the ADR equation.

Our results show that APGLM4 provides stability on par with BBDF and accuracy comparable to that of Adams-Moulton. In terms of efficiency the method is comparable to BAM3 and AM3, but its improved stability makes it useful for solving stiff PDEs. At small timesteps the method is comparable in efficiency to BBDF5. However, since it retains good stability properties with $\alpha = 4$, APGLM4 requires less analyticity and less storage per timestep than BBDF5.

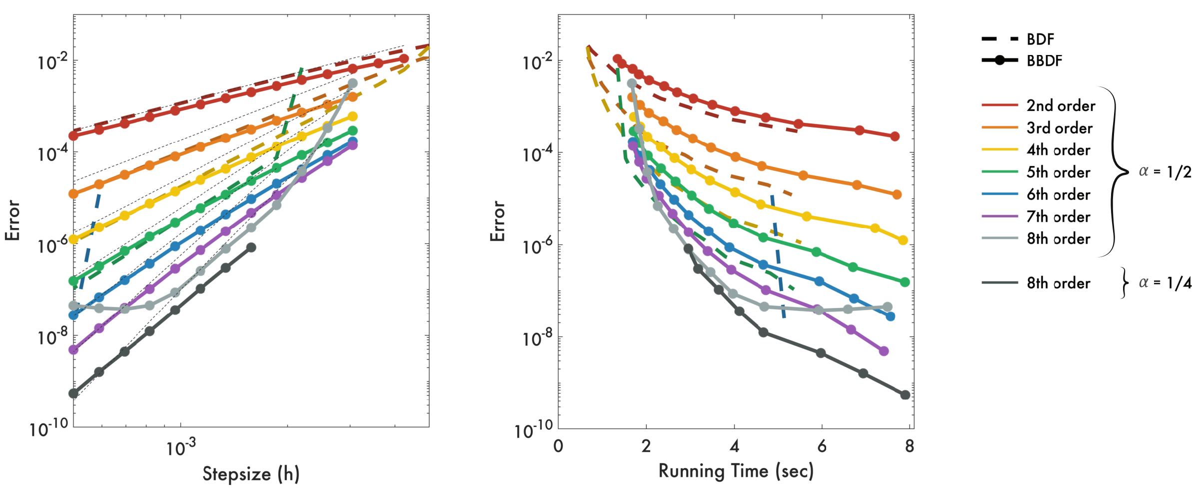
8.2.2 Results for Explicit Polynomial Block Methods

We compare the BAB method from Table (8.2) with an extrapolation parameter of $\alpha = 1$, against Adams-Bashforth. In Figure 8.4 we present error vs stepsize and error vs optimal parallel runtime for the Van der Pol oscillator. It is clear from the results that BAB methods have superior stability properties and better efficiency when implemented in parallel. In contrast, high-order AB schemes are only stable for smaller stepsizes and have inferior error constants. Our results demonstrate that BAB methods are efficient in situations where each of the ODE right-hand-sides can be computed in parallel.

8.2.3 Effects of Choosing Alpha on Computational Accuracy

Choosing different values of the extrapolation parameter α can have a significant effect on stability and accuracy. Here we present two numerical experiments using the parallel,

Viscous Burgers: BDF and BBDF with $\alpha = 1/2$



Viscous Burgers: AM and BAM with $\alpha = 1/2$

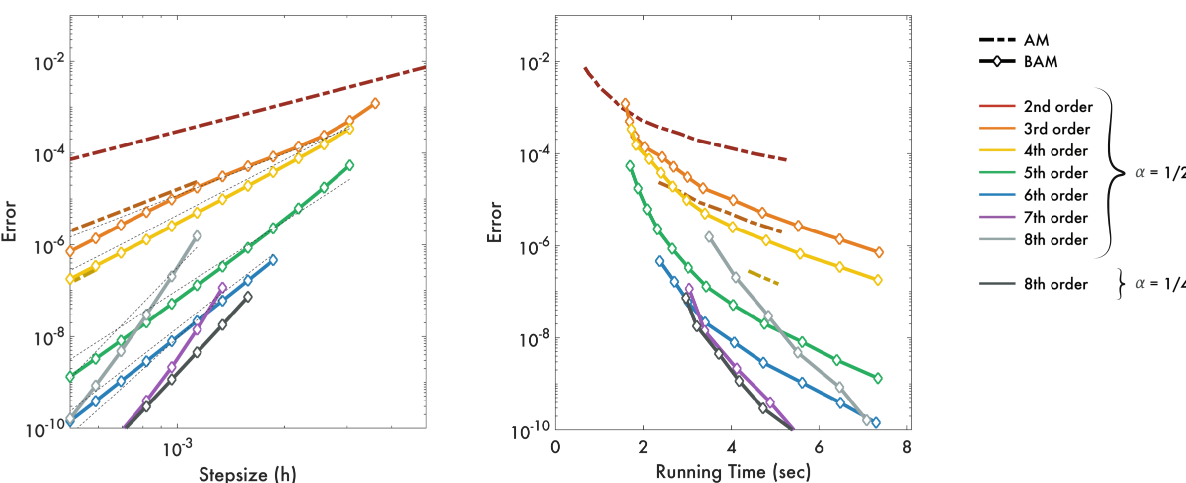
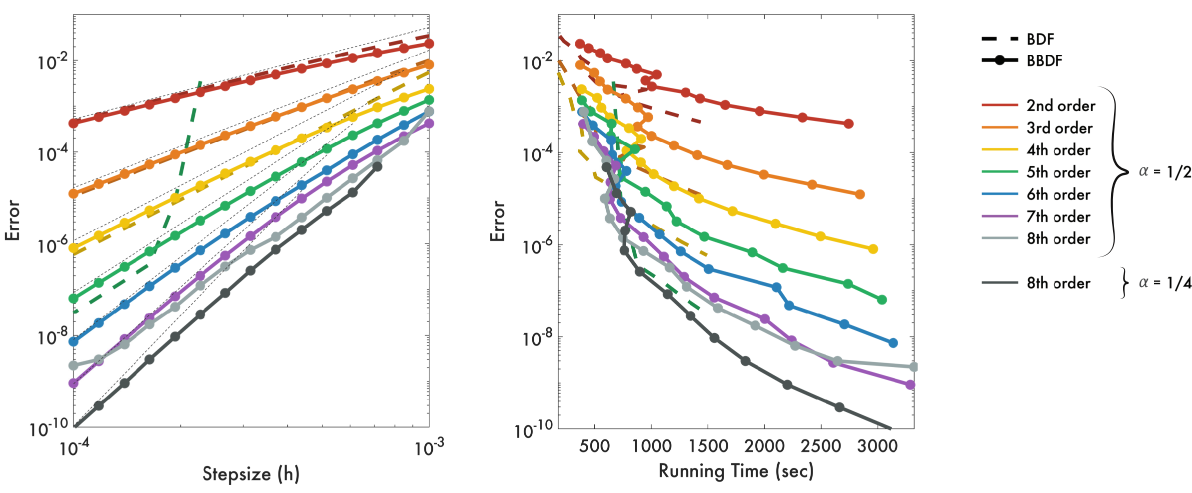


Figure 8.1: Error vs stepsize and error vs running time diagrams for the Viscous Burgers equation. We present results for BDF and BBDF in the top two plots, and results for BAM and Adams-Moulton (AM) in the bottom two plots. All BAM and BBDF methods are run using $\alpha = 1/2$. The seven thinly-dashed grey lines on the error vs stepsize plots show the slopes expected in these log-log plots for 2nd through 8th order convergence.

2D ADR: BDF and BBDF with $\alpha = 1/2$



2D ADR: AM and BAM with $\alpha = 1/2$

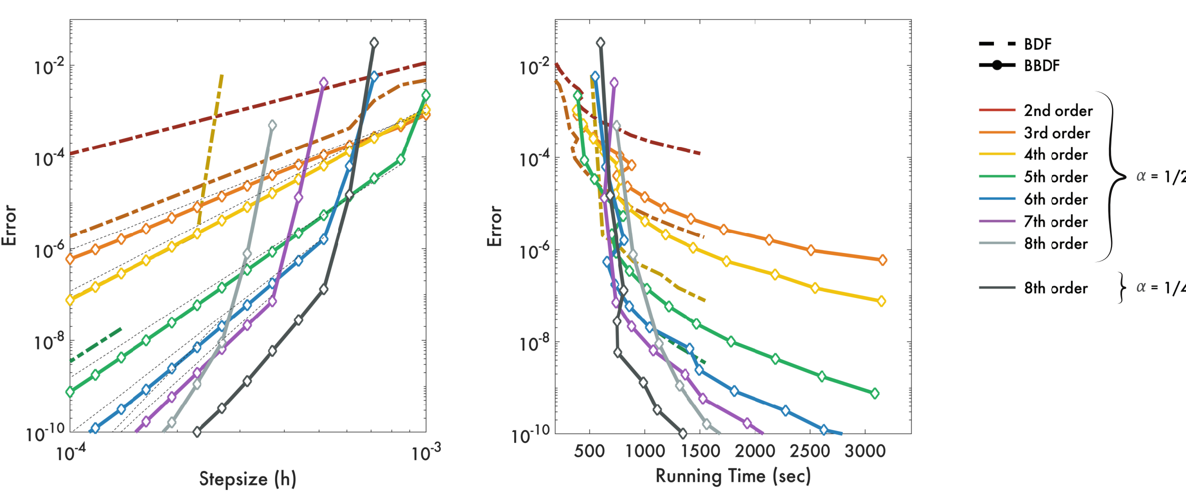


Figure 8.2: Error vs stepsize and error vs running time diagrams for the 2D ADR equation. We present results for BDF and BBDF in the top two plots, and results for BAM and Adams-Moulton (AM) in the bottom two plots. All BAM and BBDF methods are run using $\alpha = 1/2$. The seven thinly-dashed grey lines on the error vs stepsize plots show the slopes expected in these log-log plots for 2nd through 8th order convergence.

Viscous Burgers: BDF, AM and BBDF, BAM, APGLM4 with $\alpha = 4$

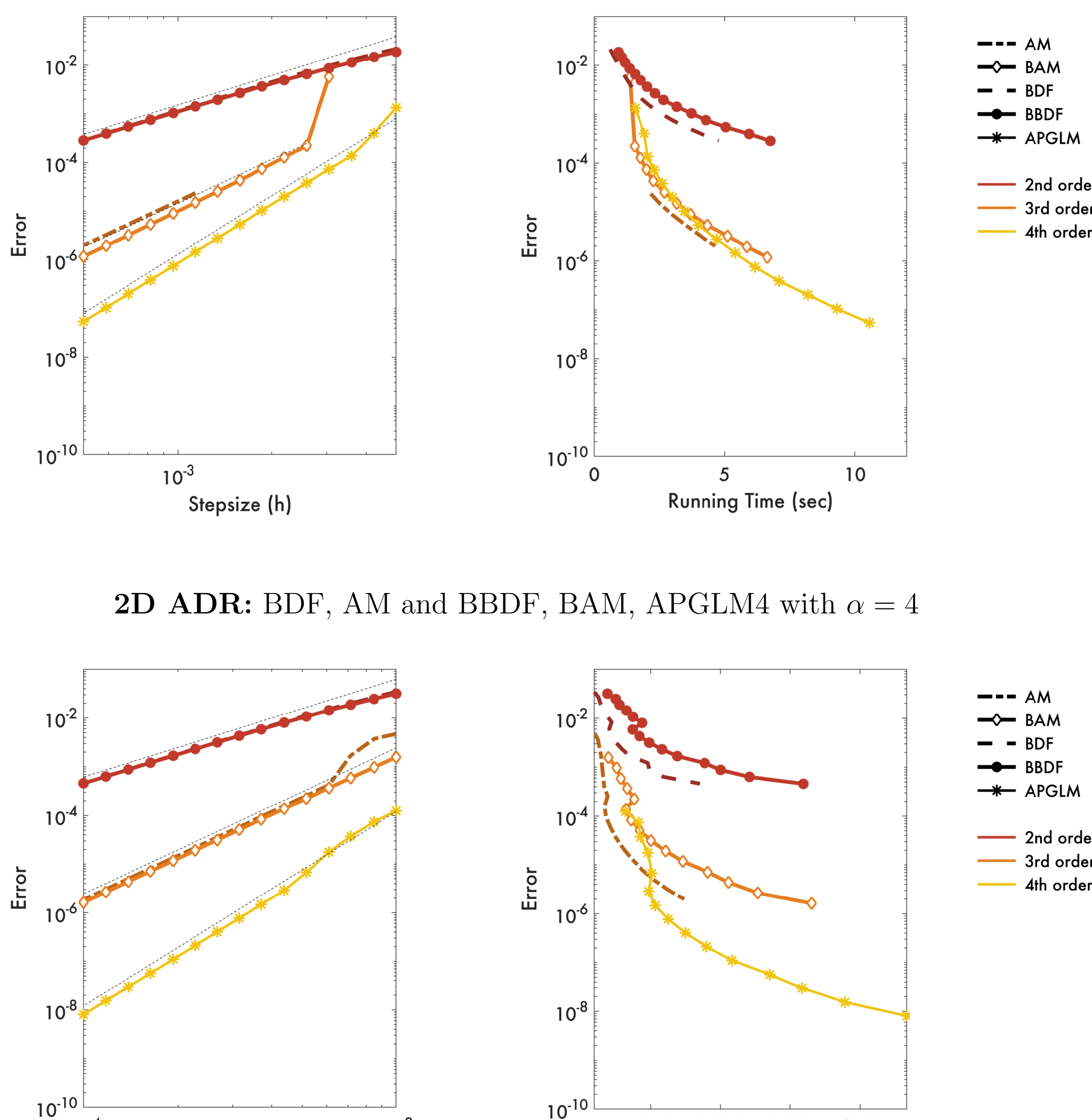


Figure 8.3: Error vs stepsize and error vs running time diagrams for the Burgers and 2D ADR equation. We present results for APGLM4 and the block methods BBDF2, BAM3 all with $\alpha = 4$. We also include second order BDF and third order Adams-Moulton in our plots. The three thinly-dashed grey lines on the error vs stepsize plots show the slopes expected in these log-log plots for 2nd, 3rd and 4th order convergence.

Van der Pol: AB and BAB with $\alpha = 1$

Figure 8.4: Error vs stepsize and error vs running time diagrams for the Van der Pol equation. We present results for AB and BAB. All BAB methods are run using $\alpha = 1$. The thin dashed black lines on Error vs Stepsize plots correspond to order $2, 3, \ldots, 8$ accuracy. The seven thinly-dashed grey lines on the error vs stepsize plots show the slopes expected in these log-log plots for 2nd through 8th order convergence.

diagonally implicit schemes BBDF and BAM to numerically investigate the effects of varying α. In a future publication we will provide a more complete analysis.

Choosing a smaller α for BBDF and BAM schemes leads to better stability at the cost of requiring more analyticity per timestep. Changing α will also have an effect on the error constants and on the severity of rounding errors. We demonstrate the effects of α by solving the viscous Burgers equation using fourth, sixth and eight order BBDF and BAM methods where we take α to be either 1/2, 1/4 or 1/8. We present our numerical results in Figure 8.5.

Our results reveal that α has a significant effect on the performance of certain methods. For high-order methods, large α leads to catastrophic rounding errors, and should be avoided for both BBDF and BAM methods. By choosing $\alpha = 1/8$ we see that both BBDF8 and BAM8 are able to achieve much better errors than in our previous numerical experiment where $\alpha = 1/2$. However, the lack of analyticity in the solution prevents methods with small α from converging at larger time-steps.

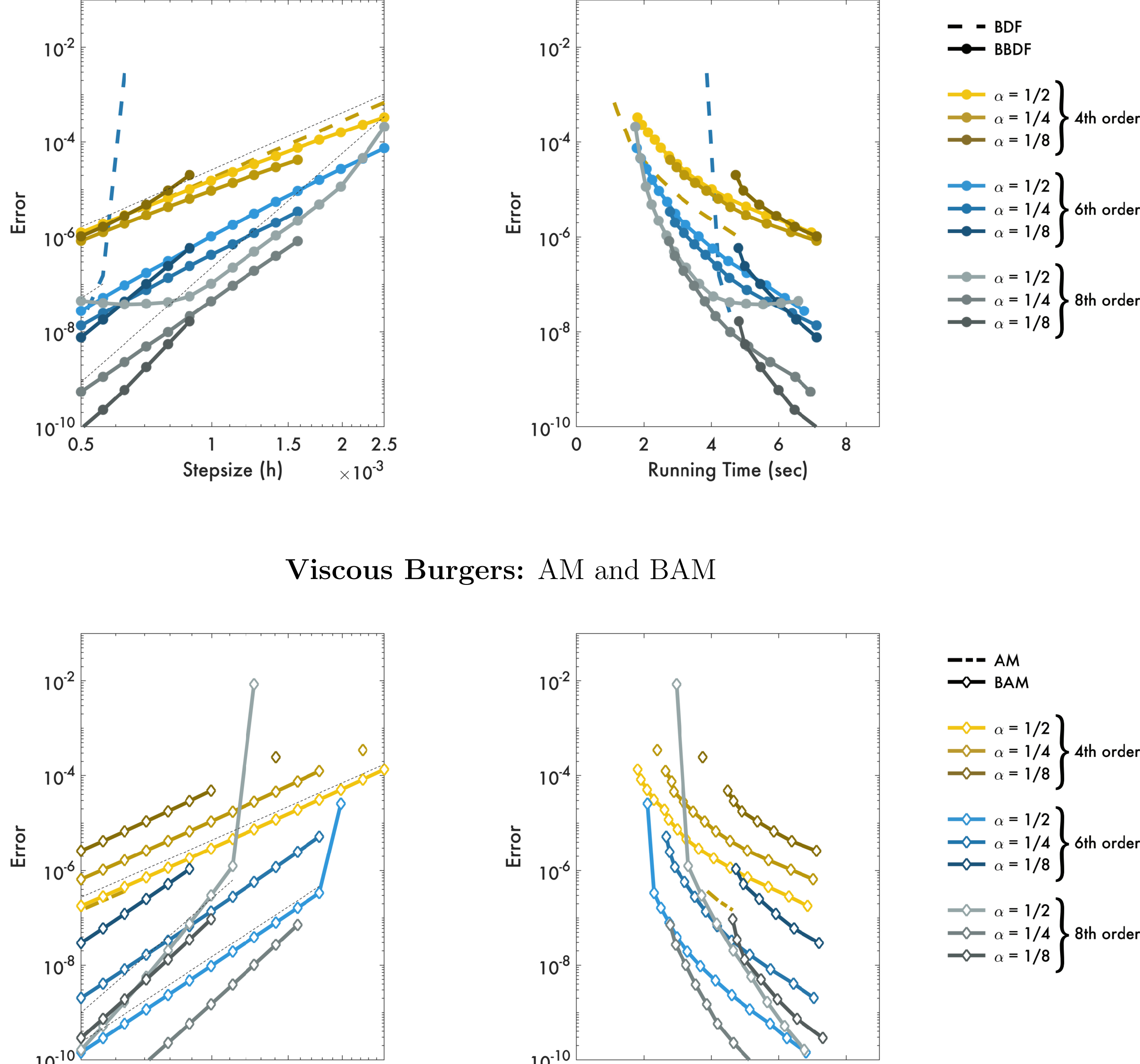


Figure 8.5: Error vs stepsize and error vs running time diagrams for the viscous Burgers equation. We present results for BDF, BBDF, AM, and BAM methods. For BBDF and BAM methods we test methods with α values of 1/2, 1/4, and 1/8. The three thinly-dashed grey lines on the error vs stepsize plots correspond to 4th, 6th and 8th order convergence.

Chapter 9

SPECIAL FAMILIES OF POLYNOMIAL INTEGRATORS

Stiff initial value problems arising from the spatial discretization of partial differential equations will continue to pose computational challenges in the foreseeable future. In addition to classical implicit methods, there are several specialized integrators for solving these problems including additive integrators [56, 57, 79, 2, 72, 83], exponential integrators [74, 92, 81, 82, 55, 51, 53], Rosenbrock integrators [84, 77], and Lawson integrators [60, 63]. The broad aim of this work is to pave the foundation for developing specialized polynomial integrators possessing high-orders of accuracy and parallelism.

The need to develop more accurate time-integrators is driven by the increasing popularity of high-order spatial discretizations for solving stiff partial differential equations. These new discretizations require a better balance between spatial and temporal accuracy, especially in the case of pseudo-spectral methods and high-order spatial discretizations like spectral elements and discontinuous Galerkin. Even in situations where high-order is not required, it remains advantageous to develop families of numerical methods that can be implemented at any order. This creates opportunities for adaptive-order codes that maintain efficiency for a wide range of problems.

In the last decade there also has been a significant push to develop parallel time-integrators [89, 29, 30, 23, 65, 36]. This is a direct response to leveling clock-rates and the increasing prevalence of parallel architectures. In short, time-parallelization can offer additional avenues for speedup in situations where spatial parallelism has been saturated. We aim to extend the ideas presented for implicit methods to develop parallel specialized integrators that can leverage modern advances in computational hardware.

9.1 Model Problems

In this chapter we discuss polynomial integrators for solving stiff *partitioned* and *unpartitioned* initial value problems. We define these two problem types as:

Unpartitioned System $$\mathbf{y}'(t) = \mathbf{F}(t, \mathbf{y}(t)), \tag{9.1}$$

Partitioned System $$\mathbf{y}'(t) = \sum_{k=1}^{m} \mathbf{F}^{\{k\}}(t, \mathbf{y}(t)). \tag{9.2}$$

In the following subsections we present a well-known approach for reducing an unpartitioned system to a partitioned system, and discuss an important special case of the partitioned system.

9.1.1 Linearizing Unpartitioned Systems

Any unpartitioned system can be reduced to a partitioned system via local linearization. This is accomplished by rewriting the right-hand-side $\mathbf{F}(t, \mathbf{y})$ as

$$\begin{aligned} \mathbf{F}(t, \mathbf{y}(t)) &= \mathbf{F}_\nu + \mathbf{J}_\nu\,(\mathbf{y} - \mathbf{y}_\nu) + \mathbf{R}(t, \mathbf{y}(t); \nu) \\ \mathbf{R}(t, \mathbf{y}(t); \nu) &= \mathbf{F}(t, \mathbf{y}(t)) - \left[\mathbf{F}_\nu + \frac{\partial \mathbf{F}}{\partial \mathbf{y}}(\mathbf{y}_\nu)\,(\mathbf{y}(t) - \mathbf{y}_\nu)\right] \end{aligned}$$

where $\mathbf{y}_\nu = \mathbf{y}(\nu)$, $\mathbf{F}_\nu = \mathbf{F}(\nu, \mathbf{y}_\nu)$, and $\mathbf{J}_\nu = \frac{\partial \mathbf{F}}{\partial \mathbf{y}}(\nu, \mathbf{y}_\nu)$ is the Jacobian of $\mathbf{F}$ at the point $t = \nu$. Then the resulting expression can be partitioned in several ways, the most common being

$$\mathbf{y}'(t) = \sum_{k=1}^{2} \mathbf{F}^{\{k\}}(t, \mathbf{y}(t)) \qquad \begin{cases} \mathbf{F}^{\{1\}}(t, \mathbf{y}(t)) = \mathbf{J}_\nu \mathbf{y} \\ \mathbf{F}^{\{2\}}(t, \mathbf{y}(t)) = \mathbf{F}_\nu - \mathbf{J}_\nu \mathbf{y}_\nu + \mathbf{R}(t, \mathbf{y}(t); \nu) \end{cases} \tag{9.3}$$

Many exponential integrators and Rosenbrock methods [41] are derived by applying the linearization $\nu = t_n$ at each time-step [52, 92].

9.1.2 Semilinear Systems

Discrete semilinear systems are an important subclass of partitioned systems. They can be written as

$$\mathbf{y}'(t) = \mathbf{L}\mathbf{y} + \mathbf{N}(t, \mathbf{y}(t)), \tag{9.4}$$

where $\mathbf{L}$ is a square matrix. In many cases, $\mathbf{L}$ captures the majority of the stiffness such that

$$\rho(\mathbf{L}) \gg \rho\left(\frac{\partial \mathbf{N}}{\partial \mathbf{y}}\right)$$

where ρ denotes spectral radius and $\frac{\partial \mathbf{N}}{\partial \mathbf{y}}$ is the Jacobian of $\mathbf{N}(t, \mathbf{y})$. In Table 9.1 we present several partial differential equations whose discretizations produce split systems of this form. The system (9.19) is also a semilinear system where the Jacobian $\mathbf{J}_\ell$ plays the role of $\mathbf{L}$ and the remaining terms form $\mathbf{N}(t, \mathbf{y}(t))$.

9.2 Additive Integrators

Additive integrators are designed to solve systems of additively partitioned first-order ordinary differential equations (9.2). Additive Integrators have the ability to treat each partition differently depending on properties like stiffness, smoothness, accuracy requirements, and computational cost. This can lead to improved efficiency for equations where the derivative components evolve on different timescales, or in situations where it is prohibitively expensive to treat the full derivative implicitly.

Nonlinear Schrödinger	$iu_t = u_{xx} + \|u\|^2 u$
Viscous Burgers	$u_t = u_{xx} - uu_x$
Kuramoto-Sivashinsky	$u_t = -u_{xx} - u_{xxxx} - \frac{1}{2}\left(u^2\right)_x$

Table 9.1: Several one-dimensional semilinear partial differential equations which produce discrete semilinear systems if spatially discretized using finite differences or a pseudo-spectral method.

9.2.1 Overview

A simple example of an additively partitioned problem is the scaler system

$$y'(t) = f^{\{1\}}(t, y(t)) + f^{\{2\}}(t, y(t)).$$

A classical integrator will treat both $f^{\{1\}}(t, y(t))$ and $f^{\{2\}}(t, y(t))$ explicitly or implicitly; however, this is not the only possibility. For example, suppose we first integrate our initial value problem to obtain

$$y(t) = y(b) + \int_b^t f^{\{1\}}(s, y(s))ds + \int_b^t f^{\{2\}}(s, y(s))ds$$

Next suppose we approximate the integrands using an zeroth-order implicit approximation for $f^{\{1\}}(t, y(t))$ and a zeroth-order explicit approximation for $f^{\{2\}}(t, y(t))$ to obtain

$$y(t) \approx y(b) + \int_b^t f^{\{1\}}(t, y(t))ds + \int_b^t f^{\{2\}}(b, y(b))ds.$$

By taking $b = t_n$ and $t = t_n + h$, this equation produces an additive Euler method

$$y_{n+1} = y_n + hf_{n+1}^{\{1\}} + hf_n^{\{2\}} \tag{9.5}$$

If applied to the semilinear system (9.4), this method reduces to the IMEX Euler method.

In general it is possible to construct more accurate additive integrators that can be used to solve systems with any number of partitions. One notable example is the general additive Runge-Kutta (GARK) framework [83, 39]. In the subsequent sections we describe the construction of polynomial integrators for additively partitioned initial value problems.

9.2.2 Extending ODE Datasets and ODE Polynomials

To develop additive polynomial integrators, we must first adapt our definitions for ODE dataset, ODE polynomial, and interpolated value set. To simplify our notation, we will

introduce these definitions using a scaler first-order ordinary differential equation

$$y'(t) = F(y(t)) = \sum_{j=1}^{m} F^{\{j\}}(t, y(t)). \tag{9.6}$$

Definition 9.2.1 (Partitioned ODE Dataset)**.** *An ODE dataset $D(r,s)$ of size w is an ordered set of tuples of the form*

$$\left\{ \left(\tau_j,\ y_j,\ rf_j^{\{1\}},\ \ldots,\ rf_j^{\{m\}} \right) \right\}_{j=1}^{w} \tag{9.7}$$

where $t(\tau) = r\tau + s$, $y_j \approx y(t(\tau_j))$, and $f_j^{\{k\}} = F^{\{k\}}(t(\tau_j), y_j)$.

Definition 9.2.2 (Partitioned ODE Solution Polynomial)**.** *A partitioned ODE solution polynomial $p(\tau; b)$ approximates the truncated Taylor series for the local solution $y(t(\tau))$, where $t(\tau) = r\tau + s$, expanded around the point $\tau = b$. A partitioned ODE solution polynomial of degree g can be expressed as*

$$p(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!}, \qquad g \leq 2w \tag{9.8}$$

where each approximate derivative $a_j(b)$ is computed using values from an ODE dataset $D(r,s)$ and any associated interpolated value set $I(r,s)$, in one of the following ways:

1. Differentiate a polynomial approximation to $y(t(\tau))$: *Construct a local interpolating polynomial $h_j(\tau)$ passing through at least one of the approximate solution values y_k in $D(r,s)$ or $I(r,s)$ and any number of approximate full derivatives. Any approximate full derivative F must be formed by summing component derivatives in $D(r,s)$ or $I(r,s)$ in the following way*

$$F = rf_{k_1}^{\{1\}} + rf_{k_2}^{\{2\}} + \ldots + rf_{k_m}^{\{m\}} \qquad \text{where} \qquad \tau_{k_1} = \tau_{k_2} = \ldots = \tau_{k_m}.$$

Then,

$$h_j(\tau) \approx y(t(\tau)) \qquad \text{and} \qquad a_j(b) = \frac{d^j}{d\tau^j} h_j(\tau)\bigg|_{\tau=b}.$$

2. Differentiate a polynomial approximation to $ry'(t(\tau))$: *Construct the set of local interpolating polynomials*

$$l_j^{\{k\}}(\tau), \qquad k = 1, \ldots, m$$

where $l_j^{\{k\}}(\tau)$ *passes through a subset of the derivative component values* $rf_l^{\{k\}}$ *in* $D(r,s)$ *or* $I(r,s)$. *Then,*

$$l_j(\tau) = \sum_{k=1}^{m} l_j^{\{k\}}(\tau) \approx ry'(t(\tau))$$

and the approximate derivatives

$$a_j(b) = \frac{d^{j-1}}{d\tau^{j-1}} l_j(\tau)\bigg|_{\tau=b} \qquad \text{(only valid for } j \geq 1\text{)}.$$

Definition 9.2.3 (Partitioned ODE Derivative Polynomial)**.** *A partitioned ODE derivative polynomial approximates the Taylor series for the local solution derivative* $ry'(t(\tau))$, *where* $t(\tau) = r\tau + s$, *expanded around the point* $\tau = b$. *A partitioned ODE derivative polynomial of degree* g *can be expressed as*

$$\dot{p}(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!} \tag{9.9}$$

where each approximate derivative $a_j(b)$ *is computed using values from an ODE dataset* $D(r,s)$ *and any associated interpolated value set* $I(r,s)$, *in either of the following ways:*

1. Differentiate a polynomial approximation to $y(t(\tau))$: *Construct a local interpolating polynomial* $h_j(\tau)$ *passing through at least one of the approximate solution values* y_k *in* $D(r,s)$ *or* $I(r,s)$ *and any number of approximate full derivative values. Any approximate full derivative* F *must be formed by summing component derivatives in* $D(r,s)$ *or* $I(r,s)$ *in the following way*

$$F = rf_{k_1}^{\{1\}} + rf_{k_2}^{\{2\}} + \ldots + rf_{k_m}^{\{m\}} \qquad \text{where} \qquad \tau_{k_1} = \tau_{k_2} = \ldots = \tau_{k_m}.$$

   Then,

$$h_j(\tau) \approx y(t(\tau)) \qquad \text{and} \qquad a_j(b) = \frac{d^{j+1}}{d\tau^{j+1}} h_j(\tau)\bigg|_{\tau=b}.$$

2. Differentiate a polynomial approximation to $ry'(t(\tau))$: *Construct the set of local interpolating polynomials*

$$l_j^{\{k\}}(\tau), \qquad k = 1, \ldots m$$

   where $l_j^{\{k\}}(\tau)$ *passes through a subset of the derivative component values* $rf_l^{\{k\}}$ *in* $D(r,s)$ *or* $I(r,s)$. *Then,*

$$l_j(\tau) = \sum_{k=1}^{m} l_j^{\{k\}}(\tau) \approx y'(t(\tau)) \qquad \text{and} \qquad a_j(b) = \frac{d^j}{d\tau^j} l_j(\tau)\bigg|_{\tau=b}.$$

Every ODE derivative polynomial approximates the local derivative such that

$$\dot{p}(\tau; b) \approx r y'(t(\tau)) \quad \forall b.$$

Definition 9.2.4 (ODE Derivative Component Polynomials). *An ODE derivative component polynomial approximates the Taylor series for the derivative component* $rF^{\{k\}}(t(\tau), y(t(\tau)))$*, where* $t(\tau) = r\tau + s$*, expanded around the point* $\tau = b$*. A ODE derivative component polynomial of degree* g *can be expressed as*

$$\dot{p}^{\{k\}}(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!} \tag{9.10}$$

where each approximate derivative value must be formed by summing component derivatives in $D(r, s)$ *or* $I(r, s)$ *in the following way:*

1. Differentiate a polynomial approximation to $rF^{\{k\}}(t(\tau), y(t(\tau)))$: *Construct a local interpolating polynomial* $l_j^{\{k\}}(\tau)$ *that passes through a subset of the derivative values* $f_l^{\{k\}}$ *in* $D(r, s)$ *or* $I(r, s)$*. Then,*

$$a_j(b) = \frac{d^j}{d\tau^j} l_j^{\{k\}}(\tau)\bigg|_{\tau=b}.$$

Every ODE derivative component polynomial approximates the local component derivative such that

$$\dot{p}^{\{k\}}(\tau; b) \approx rF^{\{k\}}(t(\tau), y(t(\tau))) \quad \forall b.$$

Definition 9.2.5 (Interpolated Value Set for Partitioned Systems). *An* interpolated value set $I(r, s)$ *is an ordered set of tuples* $I(r, s) = \{d_j\}_{j=1}^{w}$ *generated from an ODE dataset* $D(r, s)$*, where each tuple* d_j *contains a temporal node* $\tilde{\tau}$ *and either:*

1. *An interpolated solution* $\tilde{y}_j = p_j(\tau_j; b_j)$ *where* $p_j(\tau_j; b_j)$ *is a partitioned ODE solution polynomial formed from* $D(r, s)$ *or* $I(r, s)$*.*

2. *An interpolated derivative* $\tilde{f}_j = \dot{p}_j(\tau_j; b_j)$ *where* $\dot{p}_j(\tau_j; b_j)$ *is a partitioned ODE derivative polynomial formed from* $D(r, s)$ *or* $I(r, s)$*.*

3. *An interpolated derivative component* $\tilde{f}_j^{\{k\}} = \dot{p}_j^{\{k\}}(\tau_j; b_j)$*, for any* $k = 1, \ldots, m$*, where* $\dot{p}_j^{\{k\}}(\tau_j; b_j)$ *is a partitioned ODE derivative component polynomial formed from* $D(r, s)$ *or* $I(r, s)$*.*

If the tuple d_j contains an interpolated solution value, it has the form

$$d_j = (\tilde{\tau}_j,\ \tilde{y}_j,\ \emptyset) \qquad \textit{where } \tilde{y}_j \approx y(t(\tilde{\tau}_j)).$$

and $t(\tau) = r\tau + s$. If the tuple d_j contains an interpolated derivative value it has the form

$$d_j = (\tilde{\tau}_j,\ \emptyset,\ r\tilde{f}_j) \qquad \textit{where } \tilde{f}_j \approx y'(t(\tilde{\tau}_j)).$$

If the tuple d_j contains an interpolated derivative component value it has the form

$$d_j = (\tilde{\tau}_j,\ \emptyset,\ r\tilde{f}_j^{\{k\}}) \qquad \textit{where } \tilde{f}_j^{\{k\}} \approx F^{\{k\}}(t(\tilde{\tau}_j), y(t(\tilde{\tau}_j))).$$

9.2.2.1 Families of ODE Solution Polynomials for Partitioned Systems

We can extend the definitions for Adams, BDF, and GBDF ODE polynomial to their partitioned counterparts. To shorten our discussion we introduce several interpolating polynomials that are each constructed from data values in an ODE dataset $D(r, s)$ and any associated interpolated value set $I(r, s)$.

1. Let $L_y(\tau) \approx y(t(\tau))$ be a Lagrange interpolating polynomial passing exclusively though approximate solution data in $D(r, s)$ and $I(r, s)$.

2. Let $L_F^{\{k\}}(\tau) \approx rF^{\{k\}}(t(\tau), y(t(\tau)))$ for $k = 1, \ldots, m$, be a set of Lagrange interpolating polynomial of degree $g - 1$ passing exclusively through derivative data for the kth component.

3. Let $H_y(\tau) \approx y(t(\tau))$ be an interpolating polynomial of degree g passing through: (1) at least one approximate solution value, and (2) at least one approximate full derivative. Any approximate full derivative F must be formed by summing derivative components values

$$rf_{k_1}^{\{1\}} + rf_{k_2}^{\{2\}} + \ldots + rf_{k_m}^{\{m\}} \qquad \text{where} \qquad \tau_{k_1} = \tau_{k_2} = \ldots = \tau_{k_m}.$$

1. *Families of Partitioned ODE Solution Polynomial.* Let $p(\tau; b)$ be a partitioned ODE solution polynomial. We say that:

   (a) the polynomial $p(\tau; b)$ is of **Adams** type if its approximate derivatives

$$a_j(b) = \begin{cases} L_y(b) & j = 0 \\ \frac{d^{j-1}}{d\tau^{j-1}} \sum_{k=1}^{m} L_F^{\{k\}}(\tau)\Big|_{\tau=b} & 1 \le j \le g \end{cases} \tag{9.11}$$

   All partitioned Adams ODE solution polynomial can be expressed in integral form as

$$p(\tau; b) = L_y(b) + \int_b^{\tau} \sum_{j=1}^{m} L_F^{\{j\}}(\xi) d\xi. \tag{9.12}$$

(b) the polynomial $p(\tau; b)$ is of **GBDF** type if

$$p(\tau; b) = H_y(\tau)$$

or equivalently if it has degree g with approximate derivatives

$$a_j(b) = \frac{d^j}{d\tau^j} H_y(\tau)\bigg|_{\tau=b} \qquad j = 1, \ldots, g. \tag{9.13}$$

(c) the polynomial $p(\tau; b)$ is of **BDF** type if it is a partitioned GBDF polynomial and if $H_y(\tau)$ passes through only a single full derivative.

2. *Families of Partitioned ODE Derivative Polynomial.* Let $\dot{p}(\tau; b)$ be a partitioned ODE derivative polynomial. We say that:

(a) The polynomial $\dot{p}(\tau; b)$ is of **Adams** type if

$$\dot{p}(\tau; b) = \sum_{k=1}^{m} L_F^{\{k\}}(\tau)$$

or equivalently if it has degree $g - 1$ and approximate derivatives

$$a_j(b) = \frac{d^j}{d\tau^j} \sum_{k=1}^{m} L_F^{\{k\}}(\tau)\Bigg|_{\tau=b} \qquad j = 0, \ldots, g-1. \tag{9.14}$$

(b) the polynomial $\dot{p}(\tau; b)$ is of **GBDF** type if

$$p(\tau; b) = H_y'(\tau)$$

or equivalently if it has degree $g - 1$ with approximate derivatives

$$a_j(b) = \frac{d^{j+1}}{d\tau^{j+1}} H_y(\tau)\bigg|_{\tau=b} \qquad j = 1, \ldots, g-1. \tag{9.15}$$

(c) the polynomial $\dot{p}(\tau; b)$ is of **BDF** type if it is a partitioned GBDF ODE derivative polynomial and if $H_y(\tau)$ passes through only a single full derivative.

3. *Families of Partitioned ODE Derivative Component Polynomial.* Let $\dot{p}^{\{k\}}(\tau; b)$ be a partitioned ODE derivative component polynomial. We say that:

(a) The polynomial $\dot{p}^{\{k\}}(\tau; b)$ is of **Adams** type if

$$\dot{p}^{\{k\}}(\tau; b) = L_F^{\{k\}}(\tau)$$

or equivalently if it has degree $g - 1$ with approximate derivatives

$$a_j(b) = \frac{d^j}{d\tau^j} L_F^{\{k\}}(\tau)\bigg|_{\tau=b} \qquad j = 0, \ldots g-1. \tag{9.16}$$

9.2.3 Additive Polynomial Block Methods

An additive polynomial block method depends on the parameters:

q	number of inputs/outputs	$\{z_j\}_{j=1}^q$	nodes, $z_j \in \mathbb{C}$, $\lvert z_j\rvert \le 1$
r	node radius, $r \ge 0$	$\{b_j\}_{j=1}^q$	expansion points
α	extrapolation factor		

It can be written as

$$y_j^{[n+1]} = p_j(z_j + \alpha;\ b_j), \qquad j = 1, \ldots, q, \tag{9.17}$$

where each $p_j(\tau; b)$ is a partitioned ODE solution polynomial built from the partitioned ODE dataset

$$D(r, t_n) = \begin{cases} \text{inputs}: \left\{\left(z_j,\ y_j^{[n]},\ f_j^{\{1\}[n]},\ \ldots\ ,\ f_j^{\{m\}[n]}\right)\right\}_{j=1}^q \\ \text{outputs}: \left\{\left(z_j,\ y_j^{[n+1]},\ f_j^{\{1\}[n+1]},\ \ldots\ ,\ f_j^{\{m\}[n+1]}\right)\right\}_{j=1}^q \end{cases}$$

and any interpolated value set $I(r, t_n)$ generated from $D(r, t_n)$. Any additive polynomial block method can be written in coefficient form as

$$\mathbf{y}^{[n+1]} = \mathbf{A}(\alpha)\mathbf{y}^{[n]} + r\sum_{k=1}^{m} \mathbf{B}^{\{j\}}(\alpha)\mathbf{f}^{\{j\}[n]} + r\sum_{k=1}^{m} \mathbf{D}^{\{j\}}(\alpha)\mathbf{f}^{\{j\}[n+1]} \tag{9.18}$$

where the matrices $\mathbf{A}^{\{k\}}(\alpha)$, $\mathbf{B}^{\{k\}}(\alpha)$, $\mathbf{D}(\alpha) \in \mathbb{R}^{q\times q}$, the input and output vectors $\mathbf{y}^{[n]}$, $\mathbf{y}^{[n]} \in \mathbb{R}^q$, and the kth input and output derivative component vectors are defined as

$$\begin{aligned} \mathbf{f}_j^{\{k\}[n]} &= f_j^{\{k\}[n]} \\ \mathbf{f}_j^{\{k\}[n+1]} &= f_j^{\{k\}[n+1]} \end{aligned} \qquad j = 1, \ldots, q.$$

9.2.4 Implicit-Explicit Polynomial Block Methods

We can combine the implicit and explicit PBM construction strategies described in Chapter 5 to construct 90 families of Implicit-Explicit (IMEX) PBMs for solving the semilinear equation (9.4). Each of these integrators can be implemented at any order of accuracy, and can also be used to solve the unpartitioned system (9.1) via local linearization at each timestep.

To simplify our notation we will present the polynomial IMEX integrators using the scalar problem

$$y'(t) = \sum_{k=1}^{2} F^{\{k\}}(t, y(t)) \qquad \begin{cases} F^{\{1\}}(t, y(t)) = Ly(t) \\ F^{\{2\}}(t, y(t)) = N(t, y(t)) \end{cases}. \tag{9.19}$$

The first two steps of IMEX PBM construction (node selection, and active node set selection) are identical to those for classical PBMs. However, we require a modified procedure for generating the partitioned Adams, BDF and GBDF ODE solution polynomial for computing an IMEX method's outputs. This new procedure blends the implicit and explicit ODE polynomials described in Section 5.3, by using the implicit polynomial for the linear term and the explicit polynomial for the nonlinearity. This leads to the following polynomial constructions:

1. To obtain a **partitioned Adams** ODE solution polynomial

$$p_j(\tau; b) = L_y^{[j]}(\tau) + \int_b^\tau L_F^{\{1\}[j]}(s) + L_F^{\{2\}[j]}(s)ds$$

   choose the Lagrange polynomial $L_F^{\{1\}[j]}(\tau)$ so that it passes through

input derivatives	$f_k^{\{1\}[n]}$ for $k \in I(j)$,
output derivatives	$f_k^{\{1\}[n+1]}$ for $k \in O(j)$.

   and choose the Lagrange polynomial $L_F^{\{2\}[j]}(\tau)$ so that it passes through

input derivatives	$f_k^{\{2\}[n]}$ for $k \in I(j)$,
output derivatives	$f_k^{\{2\}[n+1]}$ for $k \in B(j)$.

   The choices for the Lagrange polynomial $L_y^{[j]}(\tau)$ are identical to those listed in subsection 5.3.1.

2. To obtain a **partitioned BDF** or a **partitioned GBDF** ODE solution polynomial

$$p_j(\tau; b) = H_y^{[j]}(\tau)$$

   Construct an interpolated value set that contains q interpolated component derivatives for the nonlinear term

$$\tilde{f}_j^{\{2\}} = \dot{p}_j^{\{2\}}(z_j + \alpha; b) \qquad j = 1, \ldots, q$$

   where $\tilde{f}_j^{\{2\}} \approx f_j^{\{2\}[n+1]}$ and $\dot{p}_j^{\{2\}}(\tau; b) = L_{F^{\{2\}}}(\tau)$ where $L_{F^{\{2\}}}(\tau)$ is a Lagrange interpolating polynomial passing through

input derivatives	$f_k^{\{2\}[n]}$ for $k \in I(j)$,
output derivatives	$f_k^{\{2\}[n+1]}$ for $k \in O(j)$.

(a) For a **partitioned BDF** polynomial choose $H^{[j]}(\tau)$ so that it passes through

input values	$y_k^{[n]}$ for $k \in I(j)$,
output values	$y_k^{[n+1]}$ for $k \in B(j)$,
output derivatives	$f_j^{\{1\}[n+1]} + \tilde{f}_j^{\{2\}}$.

(b) For a **partitioned GBDF** polynomial choose $H^{[j]}(\tau)$ so that it passes through

input values	$y_k^{[n]}$ for $k \in I(j)$,
output derivatives	$\left(f_k^{\{1\}[n+1]} + f_k^{\{2\}[n+1]}\right)$ for $k \in B(j)$ and $\left(f_j^{\{1\}[n+1]} + \tilde{f}_j^{\{2\}}\right)$.

9.3 Exponential Integrators

Exponential integrators are a class of numerical methods for solving stiff systems of ordinary differential equations. In this section we extend our polynomial framework to allow for the construction of exponential integrators for the solving the semilinear system (9.4). These integrators can also be used on unpartitioned systems that have been locally linearized around the timestep center $t = t_n$ as shown in Subsection 9.1.1.

9.3.1 Overview

All exponential integrators are derived using a discrete variation of constants formula that treats the linear term exactly and approximates the nonlinear term. Using variation of constants, the exact solution to (9.4) is given by

$$\mathbf{y}(t) = e^{\mathbf{L}(t-b)}\mathbf{y}(t_n) + \int_b^t e^{\mathbf{L}(t-s)}\mathbf{N}(s, \mathbf{y}(s))ds \qquad \forall b$$

To derive the exponential Euler method we replace the nonlinearity with a constant, equal to $\mathbf{N}(t, \mathbf{y}(s))$ at $t = b$, leading to

$$\mathbf{y}(t) \approx e^{\mathbf{L}(t-b)}\mathbf{y}(b) + \int_b^t e^{\mathbf{L}(t-s)}\mathbf{N}(b, \mathbf{y}(b))ds$$

This discretization leads naturally to the exponential forward Euler method

$$\mathbf{y}_{n+1} = e^{h\mathbf{L}}\mathbf{y}_n + \int_{t_n}^{t_n+h} e^{\mathbf{L}(t_n+h-s)}\mathbf{N}(t_n, \mathbf{y}(t_n))ds$$

Had we replaced $\mathbf{N}(t, \mathbf{y}(t))$ with a more accurate approximation formed using past solution values then we would arrive at an exponential multistep method. On the other hand, by

combining stage values that are computed using an exponential Euler-like method, we arrive at exponential Runge-Kutta methods.

The computational efficiency of exponential integrators is dependent on the algorithm used to compute the matrix exponentials. If the underlying matrix is small or diagonal then the exponential functions can be efficiently computed using techniques based on contour integration or Taylor series with scaling and squaring. If the underlying matrix is sparse and characterized by large, negative eigenvalues then Krylov subspace methods are highly effective. However, it remains challenging to efficiently initialize exponential functions for large, dense matrices or matrices with purely imaginary spectrum.

9.3.1.1 Variation of Constants and $\varphi(x)$ Expansions

The solution to the semilinear system (9.2) is

$$\mathbf{y}(t) = e^{\mathbf{L}(t-b)}\mathbf{y}(b) + \int_b^t e^{\mathbf{L}(t-s)}\mathbf{N}(s, \mathbf{y}(s))ds \qquad \forall b \tag{9.20}$$

By introducing the φ functions

$$\varphi_n(z) = \begin{cases} e^z & n = 0 \\ \frac{1}{(n-1)!}\int_0^1 e^{z(1-s)}s^{n-1}ds & n > 0 \end{cases}$$

and Taylor expanding $\mathbf{N}(s, \mathbf{y}(s))$ we can express the exact solution as an infinite series.

$$\mathbf{y}(t) = \mathbf{c}(0) + \sum_{j=1}^{\infty}(t-b)\varphi_j((t-b)\mathbf{L})\mathbf{c}(j) \tag{9.21}$$

where the vectors $\mathbf{c}(j)$ are given by

$$\mathbf{c}(j) = \begin{cases} y(b) & j = 0 \\ \left.\dfrac{d^{j-1}}{dt^{j-1}}\mathbf{N}(t, \mathbf{y}(t))\right|_{t=b} & j > 0 \end{cases}.$$

This series representation is convenient for extending the ODE Polynomial for solving the integral equation (9.20).

9.3.2 Extending ODE datasets and ODE polynomials

To derive polynomial-based exponential integrators we must extend our definitions for ODE dataset, ODE polynomial, and the interpolated value set. The dataset definition is trivially extended, by replacing full derivates with derivative components for $N(t, y(t))$. Unlike the

datasets for additive integrators, the dataset for exponential integrators do not include linear derivative components since the linear term is always treated exactly.

For simplicity, we will present our definitions using the scalar semilinear system

$$y'(t) = Ly(t) + N(t, y(t)). \tag{9.22}$$

Definition 9.3.1 (Exponential ODE Dataset)**.** *An ODE dataset $D(r,s)$ of size w is an ordered set of tuples of the form*

$$\{(\tau_j,\ y_j,\ rn_j)\}_{j=1}^{w} \tag{9.23}$$

where $t(\tau) = r\tau + s$, $y_j \approx y(t(\tau_j))$, and $n_j = N(t(\tau_j), y_j)$.

The ODE polynomial requires a complete redefinition; in order to solve the the integral equation (9.20), we must extend beyond the space of Taylor series in favor of φ series (9.21). Since the coefficients are still based on polynomial approximations, we will name these new approximations polynomial φ-expansions.

Definition 9.3.2 (Polynomial φ-Expansion)**.** *A polynomial φ-expansion $p(\tau; b)$ approximates the φ series for the local solution $y(t(\tau))$, where $t(\tau) = r\tau + s$, expanded around the point $\tau = b$. A polynomial φ-expansion of degree g can be expressed as*

$$p(\tau; b) = a_0(b) + \sum_{j=1}^{g} a_j(b)\varphi_j(r(\tau - b)\mathbf{L}) \tag{9.24}$$

where each approximate derivative $a_j(b)$ is computed using values from an ODE dataset $D(r,s)$ and any associated interpolated value set $I(r,s)$, in one of the following ways:

1. If $j = 0$, then differentiate a polynomial approximation to $y(t(\tau))$*: Construct a local Lagrange interpolating polynomial $L_y(\tau)$ passing through any approximate solution values in $D(r,s)$ or $I(r,s)$. Then*

$$L_y(\tau) \approx y(t(\tau)) \qquad \textit{and} \qquad a_0(b) = h_j(b).$$

2. If $j > 0$, differentiate a polynomial approximation to $rN(t(\tau), y(t(\tau)))$*: Construct a local Lagrange interpolating polynomial $L_j(\tau)$ passing through any approximate derivative component values in $D(r,s)$ or $I(r,s)$. Then,*

$$L_j(\tau) \approx N(t(\tau), y(t(\tau))) \qquad \textit{and} \qquad a_j(b) = \frac{d^{j-1}}{d\tau^{j-1}} L_j(\tau)\bigg|_{\tau=b}$$

In order to allow for interpolated value sets that contain approximate derivative components, we must also introduce the Exponential ODE Derivative Component Polynomial for Integral Equation (9.20).

Definition 9.3.3 (Exponential ODE Derivative Component Polynomials)**.** *An ODE derivative component polynomial approximates the Taylor series for the derivative component $N(t(\tau), y(t(\tau)))$, where $t(\tau) = r\tau + s$, expanded around the point $\tau = b$. Every ODE derivative polynomial can be expressed as*

$$\dot{p}(\tau; b) = \sum_{j=0}^{g} \frac{a_j(b)(\tau - b)^j}{j!} \tag{9.25}$$

where each approximate derivative value must be formed by summing component derivatives in $D(r, s)$ or $I(r, s)$ in the following way:

- Differentiate a polynomial approximation to $rN(t(\tau), y(t(\tau)))$*: Construct a local interpolating polynomial $l_j(\tau)$ that passes through a subset of the derivative values in $D(r, s)$ or $I(r, s)$. Then,*

$$a_j(b) = \frac{d^j}{d\tau^j} l_j(\tau)\bigg|_{\tau=b}.$$

Every ODE derivative component polynomial approximates the local component derivative such that

$$\dot{p}(\tau; b) \approx rN(t(\tau), y(t(\tau))) \quad \forall b.$$

Using the definition for ODE derivative component polynomial, we can now interpret any polynomial φ-expansion in integral form as the following approximation to the integral equation (9.20):

$$\mathbf{y}(r\tau + s) = e^{\mathbf{L}(t-b)} \underbrace{\mathbf{y}(b)}_{\text{Replace with a Lagrange interpolant for } y(t) \text{ evaluated at } b.} + \int_b^{r(\tau-s)} e^{\mathbf{L}r(\tau-s)} \overbrace{\mathbf{N}(s, \mathbf{y}(s))}^{\text{Replace with an ODE derivative component polynomial.}} ds \qquad \forall b.$$

Finally, we also introduce the exponential interpolated value set

Definition 9.3.4 (Exponential Interpolated Value Set)**.** *An* interpolated value set *$I(r, s)$ is an ordered set of tuples $I(r, s) = \{d_j\}_{j=1}^{w}$ generated from an ODE dataset $D(r, s)$, where each tuple d_j contains a temporal node $\tilde{\tau}$ and either an interpolated solution $\tilde{y}_j$ or an interpolated derivative $\tilde{n}_j$. If the tuple d_j contains an interpolated solution value, it has the form*

$$d_j = (\tilde{\tau}_j,\ \tilde{y}_j,\ \emptyset) \qquad \text{where} \qquad \tilde{y}_j \approx y(t(\tilde{\tau}_j))$$

and $t(\tau) = r\tau + s$. If the tuple d_j contains an interpolated derivative component value it has the form

$$d_j = (\tilde{\tau}_j,\ \emptyset,\ r\tilde{n}_j) \qquad \text{where} \qquad \tilde{n}_j \approx N(t(\tilde{\tau}_j), y(t(\tilde{\tau}_j)))$$

Each of the interpolated values must be computed via $\tilde{y}_j = p_j(\tau_j; b_j)$ and $\tilde{f}_j = \dot{p}_j(\tau_j; b_j)$ where $p_j(\tau; b)$ are polynomial φ-series and $\dot{p}_j(\tau; b)$ are ODE derivative component polynomials for $N(t, y(t))$ formed using the data in $D(r, s)$ or $I(r, s)$.

9.3.2.1 Types of ODE Solution Polynomial

We introduce an important family of polynomial φ-expansions that is related to the Adams ODE polynomial for classical integrators.

1. The polynomial φ-expansion $p(\tau; b)$ is of **Adams** Type if its approximate derivatives

$$a_j(b) = \begin{cases} L_y(b) & j = 0 \\ \frac{d^{j-1}}{d\tau^{j-1}} L_N(\tau)\Big|_{\tau=b} & 1 \leq j \leq g \end{cases} \tag{9.26}$$

   where $L_N(x) \approx rN(t(\tau), y(t(\tau)))$ is a Lagrange interpolating polynomial of degree $g-1$ passing exclusively though derivative component data from an ODE dataset $D(r, s)$ and an associated interpolated value set $I(r, s)$.

9.3.3 Exponential Polynomial Block Methods

An exponential polynomial block method depends on the parameters:

q	number of inputs/outputs	$\{z_j\}_{j=1}^q$	nodes, $z_j \in \mathbb{C}$, $\lvert z_j \rvert \leq 1$
r	node radius, $r \geq 0$	$\{b_j\}_{j=1}^q$	expansion points
α	extrapolation factor		

It can be written as

$$y_j^{[n+1]} = p_j(z_j + \alpha;\ b_j), \qquad j = 1, \ldots, q, \tag{9.27}$$

where each $p_j(\tau; b)$ is a polynomial φ-expansion built from the ODE dataset

$$D(r, t_n) = \begin{cases} \text{inputs}: \left\{ \left(z_j,\ y_j^{[n]},\ n_j^{[n]} \right) \right\}_{j=1}^q \\ \text{outputs}: \left\{ \left(z_j,\ y_j^{[n+1]},\ n_j^{[n+1]} \right) \right\}_{j=1}^q \end{cases}$$

and any interpolated value set $I(r, t_n)$ generated from $D(r, t_n)$. Any additive polynomial block method can be written in coefficient form as

$$y_j^{[n+1]} = \varphi_0(\eta_j \mathbf{L}) \sum_{k=1}^{q} \left(a_{jk} y_k^{[n]} + c_{jk} y_k^{[n+1]} \right) + \sum_{k=1}^{2q} \varphi_k(\eta_j \mathbf{L}) \sum_{l=1}^{q} \left(c_{jkl} n_l^{[n]} + b_{jkl} n_l^{[n+1]} \right) \quad j = 1, \ldots, q$$

9.3.3.1 Constructing Exponential Methods

Exponential polynomial integrators are closely related to classical Adams polynomial integrators. Recall, that each output for a classical Adams polynomial integrator is computed by evaluating an ODE polynomial formed from two Lagrange polynomials (one for $y(t)$ and one

for the derivative $F(t, y(t))$). Similarly, the outputs for exponential integrators are computed by evaluating a polynomial φ-expansion formed from two Lagrange polynomials (one for $y(t)$ and one for the derivative component $N(t, y(t))$).

To construct an exponential PBM we can reuse any construction strategy provided in Chapter 5 that produces an Adams PBM. We are still free to choose between real-symmetric imaginary nodes or real nodes, and any of the active node sets. However, when we construct the φ-expansions we cannot choose BDF or GBDF type since these families do not exist. This restriction still leaves us with a total of 108 possible methods, though in practice only 54 will be useful since exponential integrators are generally explicit.

As was the case for additive integrators we need to modify the procedure for determining the ODE polynomials (now called polynomial φ-expansions) for computing the methods outputs. To obtain an Adams polynomial φ-expansion

$$p_j(\tau; b) = L_y^{[j]}(b) + \sum_{j=1}^{g} a_j(b)\varphi_j(r(\tau - b)\mathbf{L}) \qquad a_j(b) = \left.\frac{d^{j-1}}{d\tau^{j-1}} L_F^{[j]}(\tau)\right|_{\tau=b}$$

choose the Lagrange polynomial $L_F^{[j]}(\tau)$ so that it passes through

input derivatives	$f_k^{[n]}$ for $k \in I(j)$,
output derivatives	$f_k^{[n+1]}$ for $k \in O(j)$.

The choices for the Lagrange polynomial $L_y^{[j]}(\tau)$ and expansion points $\{b_j\}_{j=1}^{q}$ are identical to those listed in subsection 5.3.1.

Chapter 10

CONCLUSIONS

We introduced a polynomial-based time-integration framework that combines ideas from approximation theory and complex analysis. Our approach eliminates the complexity of order conditions enabling simplified construction of methods with a specific architecture (parallel or serial), degree of implicitness (explicit, diagonally implicit, fully implicit) and desired order of accuracy. We have shown how this framework can be used to derive many types of polynomial block methods, and also provided a more general strategy for easily constructing polynomial general linear methods. Through numerical experiments and linear stability analysis we illustrated the utility of our new approach by constructing generalized BDF, Adams-Moulton, and Adams-Bashforth integrators with superior properties to classical LMM counterparts. There are still many open questions and interesting research directions relating to polynomial-based time-integration. In the following years, I will continue to research this area and plan to further develop the ideas presented in this work.

Part II

EXPONENTIAL SPECTRAL DEFERRED CORRECTION

Chapter 11

EXPONENTIAL INTEGRATORS BASED ON SPECTRAL DEFERRED CORRECTION

11.1 Introduction

Spectral deferred correction schemes are a special family of Runge-Kutta methods that can be derived without explicitly considering order conditions. In this chapter we describe the spectral deferred correction framework and expand it to include exponential integration. All the content from this chapter is taken directly from [19].

We introduce a new class of arbitrary-order exponential time differencing (ETD) methods for solving nonlinear evolution equations of the form

$$\phi_t = \Lambda\phi + \mathcal{N}(t, \phi)$$

where Λ is a stiff linear operator and $\mathcal{N}$ is a nonlinear operator. Such systems commonly arise when discretizing nonlinear wave equations including Burgers', nonlinear Schrödinger, Korteweg-de Vries, Kuramoto, Navier-Stokes, and the quasigeostrophic equation. ETD Adams methods [9, 52], ETD Runge-Kutta methods [26, 55, 61, 49, 50, 58, 52], and ETD general linear methods [78, 52] are well-understood, and many of these schemes perform competitively when integrating nonlinear evolution equations [38, 55, 67]. Despite these advances, there has been limited success developing exponential integrators with order greater than five. High-order exponential Adams methods are largely unusable due to their poor stability properties, and no exponential Runge-Kutta methods have been derived of order greater than five [68].

Nevertheless, high-order exponential integrators could prove useful if paired with spatial spectral discretizations, especially on periodic domains. Spectral methods exhibit exceptional accuracy and have been shown to be remarkably successful when applied to nonlinear wave equations [34, 93, 12]. When applying spectral methods on PDEs with smooth solutions, the time integrator often limits the overall order of accuracy. The development of stable, high-order integrators will allow for more accurate numerical simulations at reduced computational costs and will better balance spatial and temporal accuracy.

In order to develop high-order ETD schemes, we turn our attention to spectral deferred correction methods (SDC), originally developed by Dutt, Greengard, and Rokhlin [28]. SDC methods are a class of high-order, self-starting time integrators for solving ordinary differential equations. By pairing Euler's method with a Picard integral equation, SDC methods achieve an arbitrary order of accuracy and favorable stability properties. Remarkably, they are simple to implement, even at high order. In the past decade, there has been a continuing

effort to analyze and improve these methods [89, 91, 43, 22, 54, 66, 64, 40]. In particular, Minion introduced implicit-explict spectral deferred correction schemes (IMEXSDC) for integrating stiff semilinear systems [73].

In this chapter, we present a new exponential integrator based on spectral deferred correction methods. Our new integrator, which we call ETDSDC, allows for an arbitrary-order of accuracy, has favorable stability properties, and outperforms state-of-the-art ETD schemes when low error tolerances are required. In Section 11.2, we provide a brief introduction to spectral deferred correction methods before deriving our ETDSDC method and discussing IMEXSDC. In Section 11.3, we analyze and compare the stability and accuracy regions of these two methods. In Section 11.4, we discuss two techniques for accurately initializing the coefficients for our ETDSDC method. Finally, in Section 11.5, we perform numerical experiments comparing our ETDSDC method against IMEXSDC and ETDRK4, a well-known fourth-order exponential integrator [26].

11.2 Spectral Deferred Correction Methods

In this section, we provide a review of Euler-based spectral deferred correction methods [28], before deriving our ETDSDC method in Section 11.2.3 and the IMEXSDC method [73] in Section 11.2.4. To introduce SDC methods, we consider a first-order initial value problem of the form

$$\begin{aligned} \phi'(t) &= F(t,\phi) \\ \phi(a) &= \phi_a \end{aligned} \tag{11.1}$$

where $\phi \in \mathbb{C}^d$ and $F(t,\phi)$ is ν times differentiable for $\nu \gg 1$. We then shift our attention to a semi-linear first-order initial value problem of the form

$$\begin{aligned} \phi'(t) &= \Lambda\phi + \mathcal{N}(t,\phi) \\ \phi(a) &= \phi_a \end{aligned} \tag{11.2}$$

where again $\phi \in \mathbb{C}^d$, $\mathcal{N} \in C^\nu$, and Λ is a $d \times d$ matrix (not necessarily diagonal). The continuity conditions on $\mathcal{N}(t,\phi)$ and $F(t,\phi)$ are stronger than the Lipschitz continuity required for existence and uniqueness, but they ensure that high-order methods can be applied successfully.

11.2.1 Preliminaries

Spectral deferred correction schemes iteratively improve the accuracy of an approximate solution to Eq. (11.1) by repeatedly solving an integral equation that governs error. This integral equation is of the form

$$y(t) = y(a) + \int_a^t g(s, y(s))ds + r(t), \tag{11.3}$$

where $r(a) = 0$. As first proposed by Dutt et al. [28], we can approximate the solution to Eq. (11.3) at points t_0, t_1, …, t_m using the implicit ($\ell = 1$) or explicit ($\ell = 0$) Euler-like method

$$y(t_{n+1}) = y(t_n) + h_n g(t_{n+\ell}, y(t_{n+\ell})) + r(t_{n+1}), \tag{11.4}$$

where $h_n = t_{n+1} - t_n$.

To arrive at the error equation of the form (11.3), we let $\phi^k(t)$ be an approximate solution to Eq. (11.1), and let the error be $E(t) = \phi(t) - \phi^k(t)$. By considering the integral form of Eq. (11.1), one arrives at

$$\phi(t) = \phi(a) + \int_a^t F(s, \phi(s))ds.$$

Substituting $\phi(t) = \phi^k(t) + E(t)$ leads to the integral equation

$$E(t) = -\phi^k(t) + \phi^k(a) + E(a) + \int_a^t F(s, \phi^k(s) + E(s))ds. \tag{11.5}$$

Introducing the residual

$$R(t, a, \phi^k) = \left[\phi^k(a) + \int_a^t F(s, \phi^k(s))ds\right] - \phi^k(t) \tag{11.6}$$

allows us to rewrite Eq. (11.5) as

$$E(t) = E(a) + \int_a^t G(s, E(s))ds + R(t, a, \phi^k), \tag{11.7}$$

where

$$G(s, E(s)) = F(s, \phi^k(s) + E(s)) - F(s, \phi^k(s)). \tag{11.8}$$

Rewriting Eq. (11.5) in this manner isolates the residual and the error terms and leads to an equation of the form (11.3). The residual $R(t, a, \phi^k)$ depends only on known quantities and can be approximated to arbitrary accuracy via numerical quadrature of the function $F(t, \phi^k(t))$. If we consider a single timestep of method (11.4) applied to Eq. (11.7), and suppose that $\phi^k(t)$ is a sufficiently good approximation so that

$$\sup_{t \in [t_{n+1}, t_n]} \|E(t)\| = O(h^m) \quad \text{for} \quad h = t_{n+1} - t_n \text{ and } m \in \mathbb{N},$$

then, since $F(t, \phi)$ is Lipchitz continuous in ϕ, we have that

$$\|hG(s, E(s))\| = h\|F(s, \phi^k(s) + E(s)) - F(s, \phi^k(s))\| = O(h^{m+1}).$$

Thus, the Euler-like method (11.4) is sufficient for estimating $E(t)$ to $O(h^{m+1})$ in the interval $[t_n, t_{n+1}]$. This approximate error, which we denote by $E^k(t)$, can be used to obtain an $O(h^{n+1})$ accurate solution $\phi^{k+1}(t) = \phi^k(t) - E^k(t)$. This process can be repeated M times to obtain a sequence of increasingly accurate approximations to Eq. (11.1).

To implement this strategy numerically, Dutt et al. proposed to divide each timestep $[t_n, t_{n+1}]$ into N substeps or quadrature nodes which we denote via $t_{n,1}, \ldots, t_{n,N}$ [28]. This enables us to represent the approximate solution $\phi^k(t)$ as an interpolating polynomial which passes through the quadrature points. We can then calculate a provisional solution $\phi^1(t)$ at each node using either forward or backward Euler, and obtain a sequence of higher-order approximations $\phi^k(t_{n,j}) = \phi^{k-1}(t_{n,j}) + E^{k-1}(t_{n,j})$ by repeatedly approximating the error $E(t)$ at each quadrature node using (11.4).

The choice of the nodes $t_{n,1}, \ldots, t_{n,N}$ affects the quality of the quadrature approximation used to determine Eq. (11.6). Dutt et al. use Gauss-Legendre points, and Minion has studied the implications of using different quadrature nodes [64]. After M correction sweeps, the order of accuracy at each node is $\min(N, M+1)$, regardless of the choice of quadrature nodes [43, 91].

To simplify our discussion, we consider only a single timestep of spectral deferred correction from $t_n = 0$ to t_{n+1}. We find it most convenient to describe SDC methods in terms of normalized quadrature points which reduce to the quadrature points if the stepsize $h = 1$. Throughout the rest of this chapter we will make extensive use of the following definitions:

Stepsize:	$h = t_{n+1} - t_n$	Normalized nodes:	$\tau_i = t_{n,i}/h$
Substeps:	$h_i = t_{n,i+1} - t_{n,i}$	Normalized substeps:	$\eta_i = h_i/h$

We will use the notation SDC_N^M to denote a spectral deferred correction method which uses the quadrature points $\{\tau_i\}_{i=1}^N$, and performs M correction sweeps. For brevity we also use the variables $\phi_i^k = \phi^k(t_{n,i})$ and $E_i^k = E^k(t_{n,i})$ to denote the approximate solution and the error at the ith quadrature node after k correction sweeps.

11.2.2 Euler-Based Spectral Deferred Correction Methods

We now describe Euler-based spectral deferred correction methods in detail. Implicit and Explicit SDC methods use Implicit or Explicit Euler respectively to determine the provisional solution $\phi^1(t)$ at the quadrature points $h\tau_i$. Applying the Euler-like method (11.4) to Eq. (11.7) one obtains an approximation of the error $E(t)$ at each of the quadrature points. Every step of this Euler-like method requires approximating the residual term; we describe this process below.

Approximating the Residual Term: During the kth correction sweep, $\phi^k(t)$ is known at the quadrature points. The residual term (11.6) can be approximated for $t = h\tau_{i+1}$ and $a = h\tau_i$ at the cost of N function evaluations $F(h\tau_i, \phi_i^k)$ via

$$\hat{R}(h\tau_{i+1}, h\tau_i, \phi^k) = \phi^k(h\tau_i) - \phi^k(h\tau_{i+1}) + I_i^{i+1}(\phi^k)$$

where $I_i^{i+1}(\phi^k)$ denotes the Nth order numerical quadrature approximation to

$$\int_{h\tau_i}^{h\tau_{i+1}} F(s, \phi^k(s))ds. \tag{11.9}$$

The coefficients for this numerical quadrature can be obtained for general quadrature points using an algorithm which we propose in Section 11.4. For Chebyshev quadrature points, a fast $O(N\log(N))$ matrix-free algorithm exists for computing (11.9) [76].

Given the initial condition $\phi_1^1 = \phi(a)$, we can express a single timestep of an SDC_N^M method algorithmically:

Implicit ($\ell = 1$) **or Explicit** ($\ell = 0$) $\mathbf{SDC_N^M}$	Note: $E_1^k = 0$

- *Initial Solution (Euler):*

  for i=1 to N-1
  $\phi_{i+1}^1 = \phi_i^1 + h_i F(h\tau_{i+\ell}, \phi_{i+\ell}^1)$

- *Correction & Update:*

  for k=1 to M
  for i=1 to N-1
  $E_{i+1}^k = E_i^k + h_i G(h\tau_{i+\ell}, E^k, \phi^k) + \hat{R}(h\tau_{i+1}, h\tau_i, \phi^k)$
  $\phi_{i+1}^{k+1} = \phi_{i+1}^k + E_{i+1}^k$

By substituting the expression for E_{i+1}^k into the update formula for ϕ_{i+1}^{k+1}, noting that $\phi_i^{k+1} = \phi_i^k + E_i^k$, and using Eq. (11.8), one arrives at the following direct update formula:

$$\phi_{i+1}^{k+1} = \phi_i^{k+1} + h_i \left[F(h\tau_{i+\ell}, \phi_{i+\ell}^{k+1}) - F(h\tau_{i+\ell}, \phi_{i+\ell}^k) \right] + I_i^{i+1}(\phi^k).$$

This compact form for spectral deferred correction methods was first mentioned in [73] but was not recommended due to potential numerical rounding errors. However, in our numerical experiments, we find that this compact formula leads to simpler codes and equally accurate results. We therefore make use of this compact update formula in all of our codes.

11.2.3 ETD Spectral Deferred Correction Methods

We now introduce a new class of exponential integrators based on spectral deferred correction for solving Eq. (11.2), which we repeat here for convenience:

$$\phi'(t) = \Lambda\phi + \mathcal{N}(t, \phi),$$
$$\phi(a) = \phi_a.$$

To derive ETD spectral deferred correction schemes, we seek an error equation of the form

$$y(t) = y(a)e^{\Lambda(t-a)} + \int_a^t e^{\Lambda(t-s)} g(s, y(s))ds + r(t). \tag{11.10}$$

We propose to approximate the solution to Eq. (11.10) by replacing $g(s, y(s))$ with a one-point approximation, leading to the explicit ($\ell = 0$) or implicit ($\ell = 1$) ETD Euler-like method

$$y(t_{n+1}) = y(t_n)e^{h\Lambda} + \Lambda^{-1}\left[e^{h\Lambda} - I\right] g(t_{n+\ell}, y(t_{n+\ell})) + r(t_{n+1}). \tag{11.11}$$

To arrive at an error equation of the form (11.10), we let $\phi^k(t)$ be an approximate solution of Eq. (11.2), and define the error to be $E(t) = \phi(t) - \phi^k(t)$. Applying variation of constants, we obtain the integral form of Eq. (11.2),

$$\phi(t) = \phi(a)e^{\Lambda(t-a)} + \int_a^t e^{\Lambda(t-s)}\mathcal{N}(s, \phi(s))ds.$$

Substituting $\phi(t) = \phi^k(t) + E(t)$ leads to the integral equation

$$E(t) = -\phi^k(t) + \left(\phi^k(a) + E(a)\right) e^{\Lambda(t-a)} + \int_a^t e^{\Lambda(t-s)}\mathcal{N}(s, \phi^k(s) + E(s))ds. \tag{11.12}$$

Introducing the residual

$$R_e(t, a, \phi^k) = \left[\phi^k(a)e^{\Lambda(t-a)} + \int_a^t e^{\Lambda(t-s)}\mathcal{N}(s, \phi^k(s))ds\right] - \phi^k(t) \tag{11.13}$$

allows us to rewrite Eq. (11.12) as

$$E(t) = E(a)e^{\Lambda(t-a)} + \int_a^t e^{\Lambda(t-s)}H(s, E(s))ds + R_e(t, a, \phi^k), \tag{11.14}$$

where

$$H(s, E(s)) = \mathcal{N}(s, \phi^k(s) + E(s)) - \mathcal{N}(s, \phi^k(s)). \tag{11.15}$$

Now that we have obtained an error equation of the form (11.10), we are free to proceed in the same manner as Euler-based spectral deferred correction. The provisional solution $\phi^1(t)$ is calculated at the quadrature points using either implicit or explicit ETD Euler and the error at each quadrature point is estimated using (11.11). As before, we describe the computation of the residual term.

Approximating the Residual Term: During the kth correction sweep, $\phi^k(t)$ is known at the quadrature points. The residual (11.13) can be approximated for $t = h\tau_{i+1}$ and $a = h\tau_i$ at the cost of N function evaluations via

$$\hat{R}_e(h\tau_{i+1}, h\tau_i, \phi^k(t)) = \phi^k(h\tau_i)e^{h_i\Lambda} - \phi^k(h\tau_{i+1}) + W_i^{i+1}(\phi^k) \tag{11.16}$$

where $W_i^{i+1}(\phi^k)$ denotes the weighted N point numerical quadrature approximation to

$$\int_{h\tau_i}^{h\tau_{i+1}} e^{\Lambda(h\tau_{i+1}-s)}\mathcal{N}(s, \phi^k(s))ds \tag{11.17}$$

where the weight function is $w(s) = e^{\Lambda(\tau_{i+1}-s)}$. We describe in detail how to obtain the coefficients for this weighted quadrature in Section 11.4.

We use ETDSDC_N^M to denote an ETD spectral deferred which performs M correction sweeps on the quadrature points $\{\tau_i\}_{i=1}^N$. Given the initial condition $\phi_1^1 = \phi(a)$, we can express a single timestep of an ETDSDC_N^M method algorithmically:

Implicit ($\ell = 1$) **or Explicit** ($\ell = 0$) $\mathbf{ETDSDC_N^M}$ — Note: $E_1^k = 0$

- *Initial Solution (ETD Euler):*

  for i=1 to N-1
  $\phi_{i+1}^1 = \phi_i^1 e^{h_i\Lambda} + \Lambda^{-1}\left[e^{h_i\Lambda} - I\right]\mathcal{N}(h\tau_{i+\ell}, \phi_{i+\ell}^1)$

- *Correction & Update:*

  for k=1 to M
  for i=1 to N-1
  $E_{i+1}^k = E_i^k e^{h_i\Lambda} + \Lambda^{-1}\left[e^{h_i\Lambda} - I\right] H(h\tau_{i+\ell}, E^k, \phi^k) + \hat{R}_e(h\tau_{i+1}, h\tau_i, \phi^k)$
  $\phi_{i+1}^{k+1} = \phi_{i+1}^k + E_{i+1}^k$

By substituting the expression for E_{i+1}^k into the update formula for ϕ_{i+1}^{k+1}, noting that $\phi_i^{k+1} = \phi_i^k + E_i^k$, and using Eq. (11.15), one arrives at the following direct update formula:

$$\phi_{i+1}^{k+1} = \phi_i^{k+1} e^{h_i\Lambda} + \Lambda^{-1}\left[e^{h_i\Lambda} - 1\right]\left[\mathcal{N}(h\tau_{i+\ell}, \phi_{i+\ell}^{k+1}) - \mathcal{N}(h\tau_{i+\ell}, \phi_{i+\ell}^k)\right] + W_i^{i+1}(\phi^k). \quad (11.18)$$

Though we have derived both an implicit and explicit exponential integrator, we will be solely considering the explicit exponential integrator throughout the rest of this chapter.

11.2.4 IMEX Spectral Deferred Correction

We now briefly discuss Minion's IMEXSDC_N^M method for solving Eq. (11.2) [73]. The provisional solution $\phi^1(t)$ is calculated using IMEX Euler. The error and residual equations can be derived by repeating the procedure outlined in Section 11.2.1 with $F(t, y) = \Lambda y + \mathcal{N}(t, y)$. This leads to

$$E(t) = E(a) + \int_a^t \left[\Lambda E(s) + G(s, E(s))\right] ds + R(t, a, \phi^k), \quad (11.19)$$

$$H(s, E(s)) = \mathcal{N}(s, E(s) + \phi^k(s)) - \mathcal{N}(s, \phi^k(s)) \quad (11.20)$$

$$R(t, a, \phi^k) = \left[\phi^k(a) + \int_a^t \left[\Lambda\phi^k(s) + \mathcal{N}(s, \phi^k(s))\right] ds\right] - \phi^k(t). \quad (11.21)$$

Notice that Eq. (11.19) is of the form

$$y(t) = y(a) + \int_a^t \left[\Lambda y(s) + g(s, y(s))\right] ds + r(t). \quad (11.22)$$

We can approximate Eq. (11.22) by treating the linear term implicitly and the nonlinear term explicitly, yielding the IMEX Euler-like scheme

$$y(t_{n+1}) = (I - h\Lambda)^{-1}\left[y(t_n) + hg(t_{n+\ell}, y(t_{n+\ell})) + r(t_{n+1})\right].$$

The residual term (11.21) is approximated exactly as described in Section 11.2.2, except the integrand in Eq. (11.9) is now $\Lambda\phi^k(s) + N(s, \phi^k(s))$. We denote the quadrature approximation to the residual for IMEXSDC by $\tilde{R}(t, a, \phi)$. Given the initial condition $\phi_1^1 = \phi(a)$, we can express a single timestep of an IMEXSDC$_N^M$ method algorithmically:

IMEXSDC$_\mathbf{N}^\mathbf{M}$ Method	Note: $E_1^k = 0$
• *Initial Solution (IMEX Euler):* **for** i=1 to N-1 $\phi_{i+1}^1 = [I - h_i\Lambda]^{-1}\left[\phi_i^1 + h_i N(h\tau_i, \phi_i^1)\right]$ • *Correction & Update:* **for** k=1 to M **for** i=1 to N-1 $E_{i+1}^k = [I - h_i\Lambda]^{-1}\left[E_i^k + h_i H(h\tau_i, E, \phi^k) + \tilde{R}(h\tau_{i+1}, h\tau_i, \phi^k)\right]$ $\phi_{i+1}^{k+1} = \phi_{i+1}^k + E_{i+1}^k$	

By rewriting the error formula implicitly so that

$$E_{i+1}^k = \left[E_i^k + h_i(\Lambda E_{i+1}^k + H(h\tau_i, E, \phi^k)) + \tilde{R}(h\tau_{i+1}, h\tau_i, \phi^k)\right],$$

substituting this expression into the update formula for ϕ_{i+1}^{k+1}, and noting that

$$E_{i+1}^k = \phi_{i+1}^{k+1} - \phi_{i+1}^k, \qquad \phi_i^{k+1} = \phi_i^k + E_i^k$$

one arrives at the following direct update formula:

$$\phi_{i+1}^{k+1} = [I - h_i\Lambda]^{-1}\left[\phi_i^{k+1} - (h_i\Lambda)\phi_{i+1}^k + h_i(\mathcal{N}(h\tau_i, \phi_i^{k+1}) - \mathcal{N}(h\tau_i, \phi_i^k)) + \tilde{I}_i^{i+1}(\phi^k)\right]$$

where $\tilde{I}_i^{i+1}(\phi^k)$ denotes the numerical quadrature approximation to

$$\int_{h\tau_i}^{h\tau_{i+1}} \Lambda\phi^k(s) + \mathcal{N}(s, \phi^k(s))ds.$$

11.3 Stability and Accuracy

Determining the stability properties of IMEX and ETD integrators is non-trivial. A commonly used approach is to consider the model problem

$$\begin{aligned} \phi' &= \mu\phi + \lambda\phi \\ \phi(0) &= 1 \end{aligned} \tag{11.23}$$

where $\mu, \lambda \in \mathbb{C}$ and the terms $\mu\phi$, $\lambda\phi$ act as the linear and nonlinear term respectively. This model problem highlights stability for Eq. (11.2) when it is possible to simultaneously diagonalize both the linear and nonlinear operators around a fixed point. Though this analysis does not extend to general linear systems, it has proven useful for predicting stability properties of IMEX and ETD methods on a variety of partial differential equations [38].

Applying an ETDSDC_N^M or $\mathrm{IMEXSDC}_N^M$ method on Eq. (11.23) leads to a recursion relation of the form

$$\phi(t_{n+1}) = \psi_N^M(r, z)\phi(t_n)$$

where $r = \mu h$, $z = \lambda h$, and h denotes the timestep. As with all one-step methods, the stability region is defined as

$$\mathcal{S} = \{(r, z) \in \mathbb{C}^2, |\psi_N^M(r, z)| \le 1\}.$$

We list the stability functions $\psi_N^M(r, z)$ for ETDSDC_N^M and $\mathrm{IMEXSDC}_N^M$ schemes in Table 11.1.

We choose to analyze stability for PDEs with linear dispersion and dissipation; thus, $r = h\mu$ and $z = h\lambda$ are complex-valued. Several strategies have been proposed for effectively visualizing the resulting four-dimensional stability region. As in [9, 26, 61], we choose to overlay two-dimensional slices of the stability regions, each corresponding to a fixed r value. For the sake of brevity, we focus our attention on 8th order methods where $N = 8,\ M = 7$ and on 16th order methods where $N = 16,\ M = 15$. For all methods, we select the Chebyshev quadrature nodes

$$\tau_i = \frac{1}{2}\left(1 - \cos\left(\frac{\pi(i-1)}{N-1}\right)\right) \qquad i = 1, \ldots, N.$$

We pick a range of real, imaginary, and complex r values to simulate nonlinear PDEs with varying degrees of linear dispersion and dissipation. We plot stability regions pertaining to

$$r \in -1 \cdot [0, 30], \qquad r \in 1i \cdot [0, 30], \text{ and } \qquad r \in \exp(3\pi i/4) \cdot [0, 30] \tag{11.26}$$

in Figure 11.1. For these three r ranges, we find that the stability regions of all methods grow as $|r|$ increases. For imaginary r, the stability regions for ETDSDC methods temporarily decrease before growing. Though all methods exhibit satisfactory stability properties, IMEXSDC methods allow for coarser timesteps on a wider range of (r, z). Overall, our results suggest that both IMEXSDC methods and ETDSDC methods exhibit good stability properties on a wide range of stiff nonlinear evolution equations.

ETDSDC Stability Functions $\psi_1^k(r,z) = 1$

$$\begin{aligned}
\psi_{i+1}^1 &= e^{r\eta_i}\psi_i^1 + \frac{e^{r\eta_i}-1}{r} z\psi_i^1 \\
\psi_{i+1}^{k+1} &= e^{r\eta_i}\psi_i^{k+1} + \frac{e^{r\eta_i}-1}{r} z(\psi_i^{k+1} - \psi_i^k) + z\sum_{j=1}^{N}\mathbf{W}_{i,j}\psi_j^k
\end{aligned} \tag{11.24}$$

$$\text{where} \qquad \mathbf{W}_{ij} = \int_{\tau_i}^{\tau_{i+1}} e^{r(\tau_{i+1}-s)} L_j(s)ds, \qquad L_j(s) = \prod_{\substack{l=1 \\ l\neq j}}^{N} \frac{(s-\tau_l)}{(\tau_j-\tau_l)}.$$

IMEXSDC Stability Functions $\psi_1^k(r,z) = 1$

$$\begin{aligned}
\psi_{i+1}^1 &= \left(\frac{1+z\eta_i}{1-r\eta_i}\right)\psi_i^1 \\
\psi_{i+1}^{k+1} &= \left(\frac{\psi_i^{k+1} + \eta_i z\left(\psi_i^{k+1}-\psi_i^k\right) - r\eta_i\psi_{i+1}^k + (r+z)\sum_{j=1}^{N}\mathbf{I}_{i,j}\psi_j^k}{1-r\eta_i}\right)
\end{aligned} \tag{11.25}$$

$$\text{where} \qquad \mathbf{I}_{ij} = \int_{\tau_i}^{\tau_{i+1}} L_j(s)ds, \qquad L_j(s) = \prod_{\substack{l=1 \\ l\neq j}}^{N} \frac{(s-\tau_l)}{(\tau_j-\tau_l)}.$$

Table 11.1: Stability functions for ETDSDC$_N^M$ and IMEXSDC$_N^M$ methods. As $r \to 0$ the stability functions of both methods limit to that of an explicit SDC$_N^M$ method.

When analyzing spectral deferred correction methods, it is also common to plot accuracy regions. Accuracy regions highlight the restrictions on the stepsize h so that error after one timestep is smaller than $\epsilon > 0$. They are simply defined as

$$\mathcal{A}_\epsilon = \{(r,z) \in \mathbb{C}^2, |\psi_N^M(r,z) - \exp(r+z)| \leq \epsilon\}.$$

They were introduced in [28] for comparing the efficiency of high-order methods, and provide a more detailed picture than stability regions which solely differentiate between convergent and divergent (r,z) pairs.

We find that as $|r|$ increases, the accuracy region containing $z = 0$ decreases rapidly for ETDSDC$_N^M$ methods and vanishes entirely for IMEXSDC$_N^M$ methods. This behavior can be understood from Eq. (11.24) and Eq. (11.25). For the ETDSDC$_N^M$ methods it follows that $\psi_N^M(r,0) = \exp(rh)$; moreover, since the stability function $\psi_N^M(r,z)$ is continuous, then for any $\epsilon > 0$, there exists a nontrivial accuracy region surrounding $z = 0$. The same cannot be said for IMEXSDC schemes since Eq. (11.25) satisfies the weaker relation $\psi(r,0) = \exp(rh) + O(rh)$; hence, as r becomes sufficiently large, there need not exist an accuracy region around $z = 0$.

We present accuracy regions for $\epsilon = 1 \times 10^{-8}$ in Figure 11.2. We consider the three ranges of r values in (11.26), but due to rapidly shrinking accuracy regions, we are only able to visualize different subsets of r values for each numerical method. ETDSDC$_N^M$ schemes outperform IMEXSDC$_N^M$ schemes for all tested values. Accuracy regions for the ETD methods decrease more slowly, and the non-vanishing accuracy regions around $z = 0$ guarantee accuracy for any r so long as z is chosen sufficiently small. The MATLAB code used to generate these figures can be found in [18] and can be easily modified to generate stability and accuracy plots for other ETDSDC or IMEXSDC methods.

11.4 *Calculating* $W_i^{i+1}(\phi^k)$

Every iteration of an ETDSDC$_N^M$ method requires computing $W_i^{i+1}(\phi^k)$, which denotes the weighted quadrature approximation to

$$\int_{h\tau_i}^{h\tau_{i+1}} e^{\Lambda(h\tau_{i+1}-s)} \mathcal{N}(s, \phi^k(s)) ds.$$

To arrive at a formula for $W_i^{i+1}(\phi^k)$, we let $\mathbf{N}_l(\phi) = \mathcal{N}(h\tau_l, \phi(h\tau_l))$ and replace $\mathcal{N}(s, \phi^k(s))$ in Eq. (11.17) with the Lagrange interpolating polynomial $L(s)$ that passes through the quadrature points $\{(h\tau_l, \mathbf{N}_l(\phi^k))\}_{l=1}^N$ so that

$$W_i^{i+1}(\phi^k) = \int_{h\tau_i}^{h\tau_{i+1}} e^{\Lambda(h\tau_{i+1}-s)} L(s) ds = \sum_{l=1}^{N} w_{i,l} \mathbf{N}_l(\phi^k). \quad (11.27)$$

For low-order methods, explicit formulae for $w_{i,l}$ can be derived by forming $L(s)$ and repeatedly applying integration by parts. Unfortunately, this direct calculation leads to increasingly

Stability Region Plots

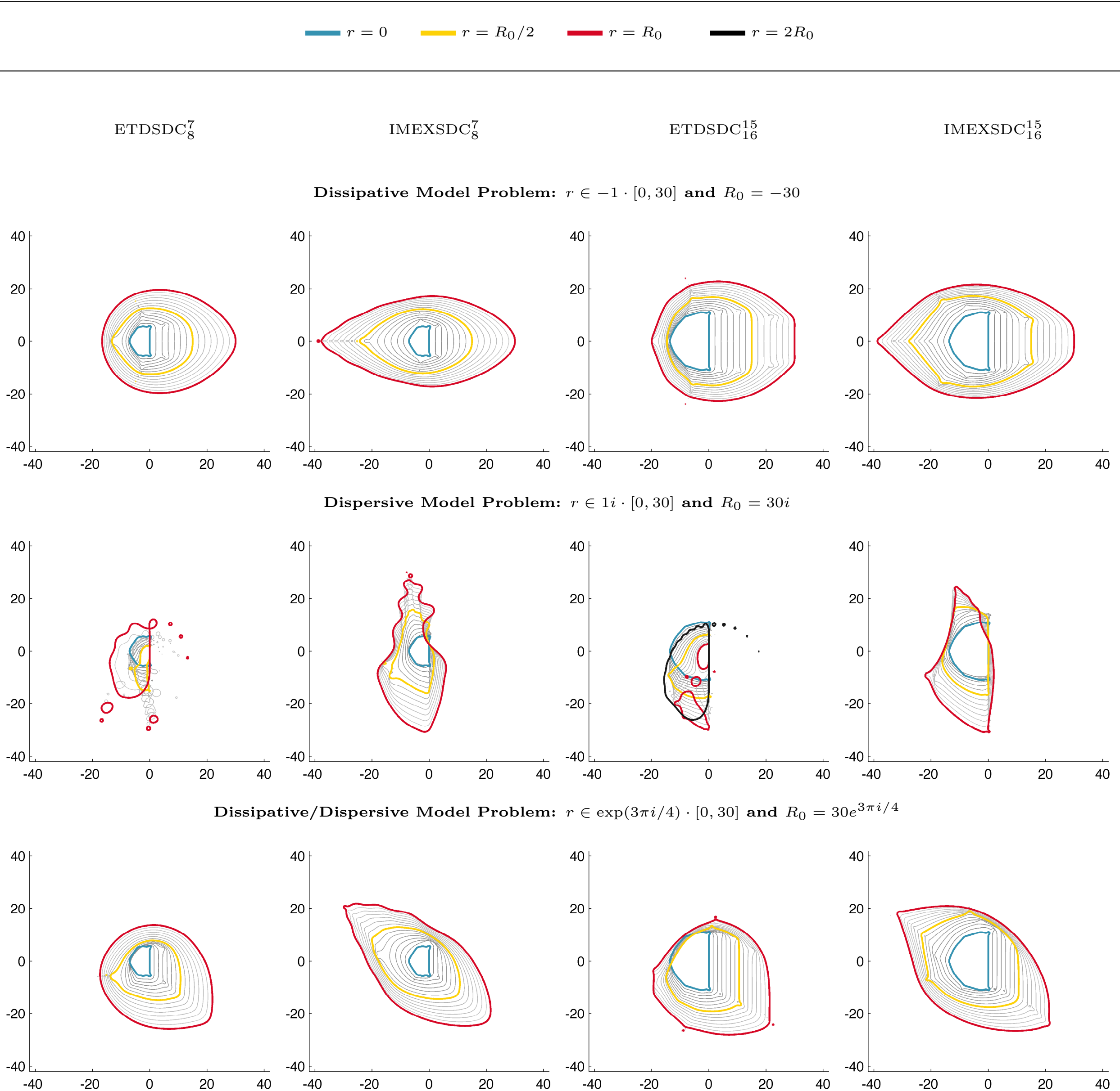


Figure 11.1: Stability regions for 8th order and 16th order methods with Chebyshev quadrature nodes. Colored contours correspond to different r values as described in the legend. We plot an additional black contour for the ETDSDC_{16}^{15} method on the dispersive model problem to show that stability regions eventually grow for sufficiently large imaginary r. For large $|r|$, increasing the order of the ETD and IMEX methods does not lead to significantly larger stability regions.

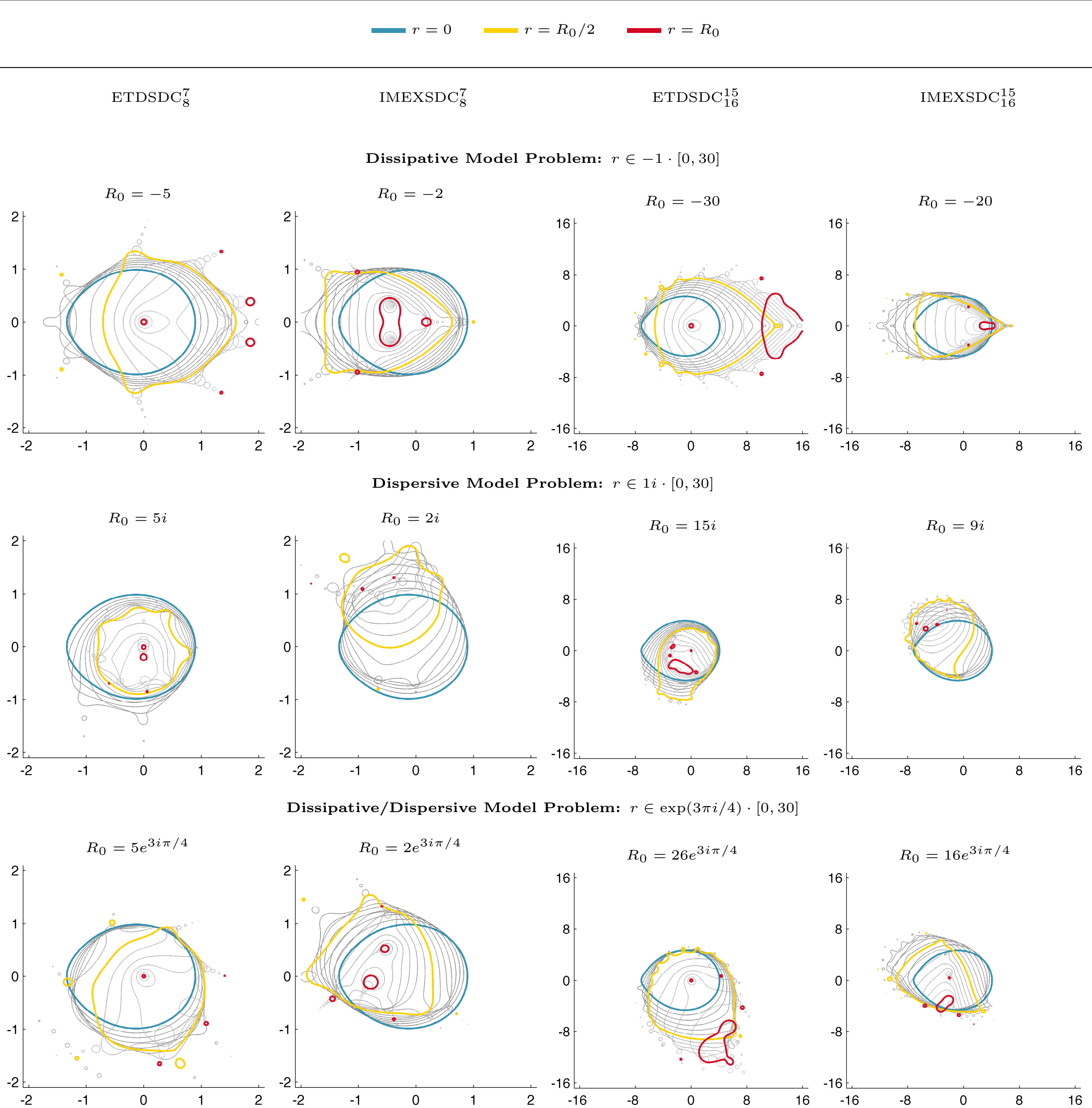


Figure 11.2: Accuracy regions corresponding to $\epsilon = 1 \times 10^{-8}$ for 8th order and 16th order methods with Chebyshev quadrature nodes. Colored contours correspond to different r values as described in legend. We choose R_0 in each figure so that the red contour marks a near vanishing accuracy region around $z = 0$. As expected, 16th order methods possess larger accuracy regions for a wider range of r than 8th order methods.

involved formulae for large N. We therefore seek a general procedure for determining $w_{i,l}$ for any N. We propose to express the weights $w_{i,l}$ in terms of the well-known functions

$$\varphi_n(z) = \frac{1}{(n-1)!}\int_0^1 e^{z(1-\sigma)}\sigma^{n-1}d\sigma.$$

using a stable algorithm developed by Fornberg for determining finite difference coefficients [32]. We describe our algorithm in Section 11.4.1, before discussing φ functions and two well-known methods for initializing them in Section 11.4.2.

11.4.1 Proposed Algorithm

To arrive at a convenient expression for $W_i^{i+1}(\phi^k)$, we propose to apply the change of variables

$$s = h\left[(\tau_{i+1}-\tau_i)\sigma + \tau_i\right] = h_i\sigma + h\tau_i, \tag{11.28}$$

to the integral term in (11.27), expand the Lagrange interpolating polynomial $L(s(\sigma))$ as a Taylor polynomial, and rewrite the result in terms of φ functions. Applying the change of variables (11.28) leads to

$$h_i\int_0^1 e^{h_i\Lambda(1-\sigma)}L(s(\sigma))d\sigma = h_i\int_0^1 e^{h_i\Lambda(1-\sigma)}P_i(\sigma)d\sigma,$$

where $P_i(\sigma)$ is the Lagrange interpolating polynomial which passes through the points

$$\{(q_{i,l},\mathbf{N}_l(\phi^k))\}_{l=1}^N \quad \text{and} \quad q_{i,l} = (\tau_l-\tau_i)/(\tau_{i+1}-\tau_i)$$

denote the scaled, translated quadrature nodes $h\tau_i$ under the transformation (11.28). Next, we define the finite difference coefficients $a_{j,l}^{(i)}$ so that

$$\left.\frac{d^j}{d\sigma^j}P_i(\sigma)\right|_{\sigma=0} = \sum_{l=1}^N a_{j,l}^{(i)}\mathbf{N}_l(\phi^k).$$

Expanding $P_i(\sigma)$ as a Taylor polynomial we obtain

$$W_i^{i+1}(\phi^k) = h_i\int_0^1 e^{h_i\lambda(1-\sigma)}\sum_{j=0}^{N-1}\left[\frac{\sigma^j}{j!}\sum_{l=1}^N a_{j,l}^{(i)}\mathbf{N}_l(\phi^k)\right]d\sigma.$$

Reordering terms we arrive at

$$\begin{aligned} W_i^{i+1}(\phi^k) &= h_i\sum_{l=1}^N\left[\mathbf{N}_l(\phi^k)\sum_{j=0}^{N-1}\left[\frac{a_{j,l}^{(i)}}{j!}\int_0^1 e^{h_i\Lambda(1-\sigma)}\sigma^j d\sigma\right]\right] \\ &= h_i\sum_{l=1}^N\left[\mathbf{N}_l(\phi^k)\sum_{j=0}^{N-1}\left[a_{j,l}^{(i)}\varphi_{j+1}(h_i\Lambda)\right]\right]. \end{aligned}$$

By defining the functions

$$w_{i,l}(z) = h_i \sum_{j=0}^{N-1} a_{j,l}^{(i)} \varphi_{j+1}(z), \tag{11.29}$$

we obtain a convenient expression for the weighted quadrature rule:

$$W_i^{i+1}(\phi^k) = \sum_{l=1}^{N} w_{i,l}(h_i\Lambda)\mathbf{N_k}(\phi^k).$$

To successfully implement this procedure, we must determine the finite difference coefficients $a_{j,l}^{(i)}$ and the matrix functions $\varphi_n(h_i\Lambda)$. The coefficients $a_{j,l}^{(i)}$ can be rapidly obtained using the stable algorithm presented in [32]. We define the functions:

- $weights(z_0, [q_1, \ldots, q_n], m)$: returns a finite difference matrix $\mathbf{a}$ for computing m derivatives at z_0, assuming q_j are the quadrature points. This calling sequence is consistent with the implementation in [33].

- $initPhi(z, n)$: returns the functions $\varphi_i(z)$ for $i = 0, \ldots, n$. We discuss two possible implementations in Section 11.4.2.

The algorithm for computing $w_{i,l}(z)$ for an ETDSDC$_N^M$ method can be written as:

Computing $w_{i,l}(h_i\Lambda)$

for i=1 to N
  $[\varphi_0(h_i\Lambda), \ldots, \varphi_N(h_i\Lambda)] = \text{initPhi}(h_i\Lambda, N)$
  for j=1 to N
    $q_j = (\tau_j - \tau_i)/(\tau_{i+1} - \tau_i)$
  $a^{(i)} = \text{weights}(0, [q_1, \ldots q_N], N-1)$
  for l=1 to N
    for j=0 to N-1
      $w_{i,l}(h_i\Lambda) = w_{i,l}(h_i\Lambda) + a_{j,l}^{(i)}\varphi_{j+1}(h_i\Lambda)$

When computing $w_{i,l}(h_i\Lambda)$, it is convenient to save $\varphi_0(h_i\Lambda)$ and $\varphi_1(h_i\Lambda)$ since both are required for the ETD Euler method.

11.4.2 φ Functions

The coefficients of all exponential integrators can be expressed in terms of φ functions [52, 7, 71, 59]. The nth φ function can be defined in the following ways:

$$\textit{Integral Form:} \qquad \varphi_n(z) = \begin{cases} e^z & n = 0 \\ \dfrac{1}{(n-1)!}\displaystyle\int_0^1 e^{z(1-s)} s^{n-1} ds & n > 0 \end{cases} \tag{11.30}$$

$$\textit{Series Form:} \qquad \varphi_n(z) = \sum_{k=0}^{\infty} \frac{z^k}{(k+n)!} \tag{11.31}$$

$$\textit{Recursion Relation:} \qquad \varphi_n(z) = \frac{\varphi_{n-1}(z) - \frac{1}{(n-1)!}}{z}, \quad \varphi_0(z) = e^z \tag{11.32}$$

The first few $\varphi_n(z)$ are given by

$$\varphi_0(z) = e^z, \quad \varphi_1(z) = \frac{e^z - 1}{z}, \quad \varphi_2(z) = \frac{e^z - 1 - z}{z^2}, \quad \varphi_3(z) = \frac{e^z - 1 - z - \frac{1}{2}z^2}{z^3}.$$

We can now rewrite the compact update formula (11.18) as

$$\phi_{i+1}^{k+1} = \varphi_0(h_i\Lambda)\phi_i^{k+1} + \varphi_1(h_i\Lambda)\left[N(h\tau_{i+\ell}, \phi_{i+\ell}^{k+1}) - N(h\tau_{i+\ell}, \phi_{i+\ell}^{k})\right] + W_i^{i+1}(\phi^k).$$

From their series definition, it follows that the functions $\varphi_n(z)$ are entire; nevertheless, it is well-known that explicit formula for $\varphi_n(z)$ are prone to catastrophic numerical roundoff error for small $|z|$. Various strategies for overcoming this difficulty have been compared extensively [3]. We briefly outline a method based on scaling and squaring [59] and a method based on contour integration [55]. Other approaches involve Krylov subspace approximations [48, 47] and improved contour integrals [94] but we do not consider them in this thesis.

11.4.2.1 Taylor/Padé Scaling and Squaring Algorithm

The scaling and squaring algorithm for calculating φ functions is a generalization of a well-known algorithm for computing matrix exponentials [46]. For small $|z|$, $\varphi_n(z)$ can be accurately evaluated via the Taylor series (11.31) or via the diagonal (m, m) Padé approximation, whose explicit formula is given in [88]. This initial approximation can be used to obtain $\varphi_n(z)$ for large $|z|$ by repeatedly applying the well-known scaling relation

$$\varphi_n(z) = \frac{1}{2^n}\left[\varphi_0\left(\tfrac{z}{2}\right)\varphi_n\left(\tfrac{z}{2}\right) + \sum_{i=1}^{n} \frac{\varphi_i\left(\frac{z}{2}\right)}{(n-i)!}\right]. \tag{11.33}$$

We present pseudocode for an m-term Taylor series procedure for initializing $\phi_i(\Lambda)$ in Table 11.2. A MATLAB implementation of the Padé scaling and squaring algorithm is freely available in [7] and can be easily used to initialize $\varphi_n(\Lambda)$ for both scalar and matrix Λ.

11.4.2.2 Contour Integration Algorithm

An alternative algorithm for initializing ETD coefficients was first suggested in [55]. Since the functions $\varphi_n(z)$ are entire, Cauchy's integral formula can be used to obtain $\varphi(z)$ at problematic regions near $z = 0$. We highlight this procedure for both scalar and matrix Λ in Table 11.2 assuming that the explicit formula for $\varphi_n(z)$ is known. If this is not the case, then it is convenient to combine Eq. (11.32) with the discretized contour integral so that

$$\varphi_n(\Lambda) = \frac{1}{P}\sum_{j=0}^{P-1} \frac{\varphi_{n-1}(\Lambda + re^{i\theta}) - 1/(n-1)!}{\Lambda + Re^{i\theta}}. \tag{11.34}$$

This allows one to progressively evaluate $\varphi_n(\Lambda)$ for $n = 1, \ldots, N$. For scalar Λ we use Eq. (11.34) when $|\Lambda| < 1$ and Eq. (11.32) when $|\Lambda| \geq 1$. For matrix Λ we find that the technique based on scaling and squaring is faster and more accurate, especially for matrices with large norm.

11.5 Numerical Experiments

In this section, we numerically solve four partial differential equations in order to compare ETDSDC$_N^M$ and IMEXSDC$_N^M$ methods of orders $4, 8, 16, 32$ against the fourth-order Runge-Kutta method (ETDRK4) developed in [26]. We have chosen to include ETDRK4 in our tests since it was shown to perform competitively [38, 55], and provides a good reference for comparing SDC based schemes to existing ETD and IMEX methods. We provide our MATLAB and Fortran implementation of ETDSDC$_N^M$, IMEXSDC$_N^M$ and ETDRK4 in [18] along with code for reproducing our numerical experiments.

In all our numerical experiments, we apply a fine spectral spatial discretization so that the error is primarily due to the time integrator. In our first three experiments we impose periodic boundary conditions and solve the PDEs in Fourier space. This is convenient since it leads to an evolution equation of the form (11.2) where the matrix Λ is diagonal. In our final experiment we consider a more challenging example where Λ is a dense matrix. We base our first three numerical experiments from [55, 38] so that our results can be compared with those obtained using other IMEX and ETD schemes.

Since we consider methods of varying order, our experiments are based on the number of function evaluations rather than the step size h. We compute reference solutions by using four times as many function evaluations as used in the experiment. To avoid biased results, we average the solutions of at least two convergent methods when forming our reference solutions. For each PDE, we present plots of relative error vs. function evaluations, relative error vs. stepsize, and relative error vs. computational time, where the relative error between two solution vectors $\mathbf{x}$ and $\mathbf{y}$ is $\|\mathbf{x}-\mathbf{y}\|_\infty/\|\mathbf{x}\|_\infty$. Though we solve equations in Fourier space, we compute relative errors in physical space. We do not count the time required to initialize ETD coefficients in our time plots. We also make no specific efforts to optimize our code, thus timing results only serve as an indication and may vary under different implementations.

<table>
<tr><th colspan="2">m-Term Taylor Scaling & Squaring Procedure for Matrix/Scalar Λ</th></tr>
<tr><td colspan="2">

• *Select Scaling Factor:*

Let $s \in \mathbb{N}$ so that $\|\Lambda/2^s\|_\infty < \delta(m)$ See [59] for choosing $\delta(m)$.

• *Initialize $\varphi_i(\Lambda/2^s)$ via Horner's Method:*

for i=0 to N
    $P_i = \frac{\Lambda}{(m+i)!} + \frac{\mathbf{I}}{(m+i-1)!}$
    for k=0 to m-2
        $P_i = \Lambda P_i + \frac{\mathbf{I}}{(m+i-2-k)!}$
    $\varphi_i(\Lambda/2^s) = P_i$

• *Obtain $\varphi_i(\Lambda)$ via Eq. (11.33):*

for i=1 to s
    for n=0 to N
        $\varphi_n(\Lambda/2^{s-i}) = \frac{1}{2^n}\left[\varphi_0\left(\Lambda/2^{s-i+1}\right)\varphi_n\left(\Lambda/2^{s-i+1}\right) + \sum_{i=1}^{n}\frac{\varphi_i\left(\Lambda/2^{s-i+1}\right)}{(n-i)!}\right]$

</td></tr>
<tr><th>Contour Integral for Scalar $|\Lambda| < 1$</th><th>Contour Integral for Matrix Λ</th></tr>
<tr><td>

• *Cauchy Integral Formula:*

$$\varphi_n(\Lambda) = \frac{1}{2\pi i}\oint_\Gamma \frac{\varphi_n(z)}{(z-\Lambda)}dz$$

• *Choosing Γ:*

Let $\Gamma = Re^{i\theta} + \Lambda$ for $\theta \in [0, 2\pi]$. The radius R should be chosen so that contour never comes near the origin.

$$\varphi_n(\Lambda) = \frac{1}{2\pi}\int_0^{2\pi} \varphi_n(Re^{i\theta} + \Lambda)d\theta$$

• *Discretization via Trapezoidal Rule:*

Let $\theta_j = 2\pi j/P$, then for P sufficiently large, $\varphi_n(\Lambda)$ is approximately

$$\frac{1}{P}\sum_{j=0}^{P-1}\varphi_n(\Lambda + Re^{i\theta_j}).$$

For scalar $|\Lambda| \geq 1$, use Eq. (11.32).

</td><td>

• *Cauchy Integral Formula:*

$$\varphi_n(\Lambda) = \frac{1}{2\pi i}\oint_\Gamma \varphi_n(z)(z\mathbf{I} - \Lambda)^{-1}dz$$

• *Choosing Γ:*

Let $\Gamma = Re^{i\theta} + z_0$ for $\theta \in [0, 2\pi]$. The radius R and center z_0 must be chosen so that contour encloses spectrum of Λ.

$$\varphi_n(\Lambda) = \frac{1}{2\pi}\int_0^{2\pi} \varphi_n(Re^{i\theta} + \Lambda)d\theta$$

• *Discretization via Trapezoidal Rule:*

Let $\theta_j = 2\pi j/P$, $\gamma_j = R\exp(\theta_j) + z_0$, then for P sufficiently large, $\varphi_n(\Lambda)$ is approximately

$$\frac{1}{P}\sum_{j=0}^{P-1}\varphi_n(\gamma_j)\left(\mathbf{I} + \frac{(z_0\mathbf{I} - \Lambda)}{Re^{i\theta_j}}\right)^{-1}$$

where $\varphi_n(\gamma_j)$ initialized like scalar Λ.

</td></tr>
</table>

Table 11.2: Scaling & squaring, and contour integral methodology for initializing $\varphi_n(\Lambda)$.

The results presented in this chapter have been run on a 3.5 Ghz Intel i7 Processor using our double precision Fortran implementation. We describe each of the four problems below.

The **Kuramoto-Sivashinsky** (KS) equation models reaction-diffusion systems [62]. As originally presented in [55], we consider the KS equation with periodic boundary conditions:

$$\begin{aligned} u_t &= -u_{xx} - u_{xxxx} - \tfrac{1}{2}\left(u^2\right)_x, \\ u(x,t=0) &= \cos\left(\tfrac{x}{16}\right)\left(1+\sin\left(\tfrac{x}{16}\right)\right), \quad x \in [0, 64\pi]. \end{aligned} \tag{11.35}$$

We numerically integrate Eq. (11.35) using a 1024 point Fourier spectral discretization in x and run the simulation out to $t = 60$. The KS equation has a dispersive linear term Λ with eigenvalues given by $\lambda(k) = k^2 - k^4$, where k denotes the Fourier wavenumber. We present our numerical results in Figure 11.3.

The **Nikolaevskiy** equation was originally developed for studying seismic waves [75] and now serves as a model for pattern formation in a variety of systems [87]. As originally presented in [38], we consider the Nikolaevskiy equation with periodic boundary conditions:

$$\begin{aligned} u_t &= \alpha\partial_x^3 u + \beta\partial_x^5 u - \partial_x^2\left(r - (1+\partial_x^2)^2\right)u - \tfrac{1}{2}\left(u^2\right)_x, \\ u(x,t=0) &= \sin(x) + \epsilon\sin(x/25), \quad x \in [-75\pi, 75\pi] \end{aligned} \tag{11.36}$$

where $r = 1/4$, $\alpha = 2.1$, $\beta = 0.77$, and $\epsilon = 1/10$. We solve the Nikolaevskiy equation using a 4096 point Fourier spectral discretization in x and run the simulation out to $t = 50$. The Nikolaevskiy equation has a dissipative and dispersive linear term with eigenvalues given by $\lambda(k) = k^2(r - (1-k^2)^2) - i\alpha k^3 + i\beta k^5$, where k denotes the Fourier wavenumber. We present our numerical results in Figure 11.3.

The **quasigeostrophic** (QG) equations model a variety of atmospheric and oceanic phenomena [80]. As originally presented in [38], we consider the barotropic QG equation on a β-plane with linear Ekman drag and hyperviscous diffusion of momentum with periodic boundary conditions,

$$\begin{aligned} \partial_t \nabla^2\psi &= -\left[\beta\partial_x\psi + \epsilon\nabla^2\psi + \nu\nabla^{10}\psi + \mathbf{u}\cdot\nabla(\nabla^2\psi)\right] \\ \psi(x,y,t=0) &= \frac{1}{8}\exp\left(-8\left(2y^2 + x^2/2 - \pi/4\right)^2\right), \\ (x,y) &\in [-\pi,\pi] \end{aligned} \tag{11.37}$$

where $\psi(x, y)$ is the stream function for two-dimensional velocity $\mathbf{u} = (-\partial_y\psi, \partial_x\psi)$, $\epsilon = 1/100$, and $\nu = 10^{-14}$. We run the simulation to time $t = 5$ using a 256×256 point Fourier discritization. We consider a different initial condition than the one presented in [38], since $\nabla^2\psi(x, y)$ was originally chosen to be discontinuous at the point $(0, 0)$. We note that Eq. (11.37) describes the change in the vorticity $\omega = \nabla^2\psi$ in terms of the stream

function ψ. In order to obtain ψ at each timestep, it is necessary to solve Poisson's equation $\nabla^2\psi = \omega$. Since we are solving in Fourier space, it follows that

$$\hat{\psi}_{k,l} = \begin{cases} 0 & k = l = 0 \\ -\frac{\hat{\omega}}{k^2+l^2} & \text{otherwise} \end{cases}$$

where k and l are the Fourier wave numbers and $\hat{\psi}$, $\hat{\omega}$ denote the discrete Fourier transforms of ψ and ω. The QG equation has a linear term with strong dissipation and mild dispersion with eigenvalues given by $\lambda(k,l) = \frac{-ik-\epsilon k^2}{k^2+l^2} - \nu(k^8 + l^8)$. We present our numerical results in Figure 11.4.

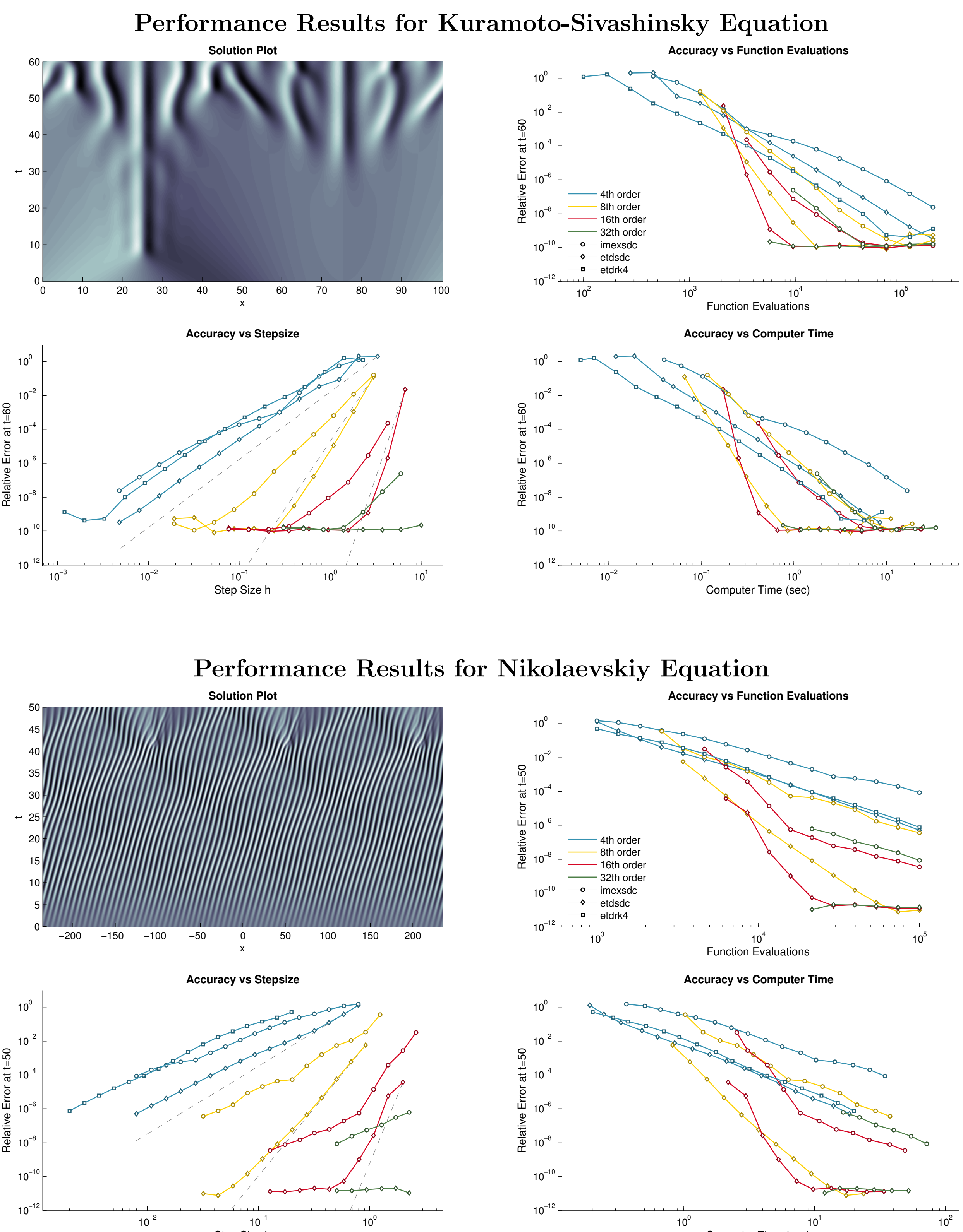
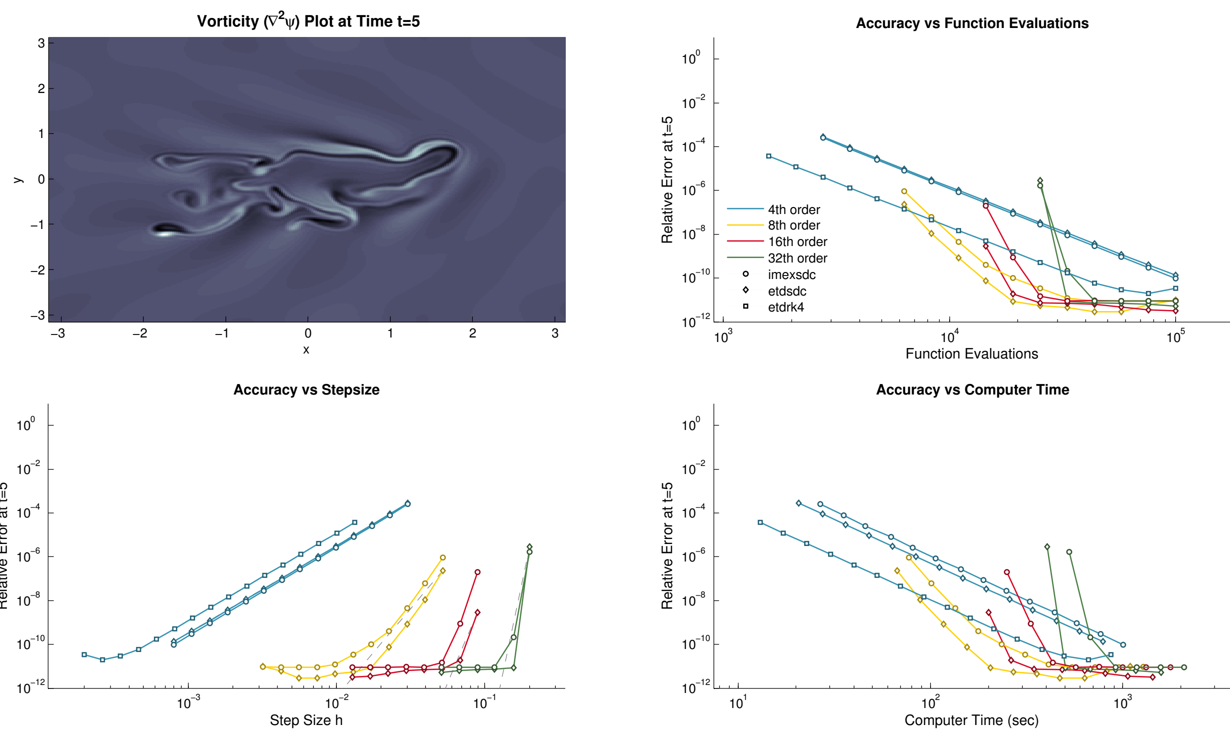


Figure 11.3: Performance results for the Kuramoto-Sivianshi and Nikolaevskiy equations. Gray dashed lines of increasing steepness in the accuracy vs stepsize plots correspond to $O(h^4)$, $O(h^8)$ and $O(h^{16})$, respectively. IMEXSDC schemes experience significant order reduction on both problems.

Performance Results for Quasigeostrophic Equation



Performance Results for Korteweg-de Vries Equation

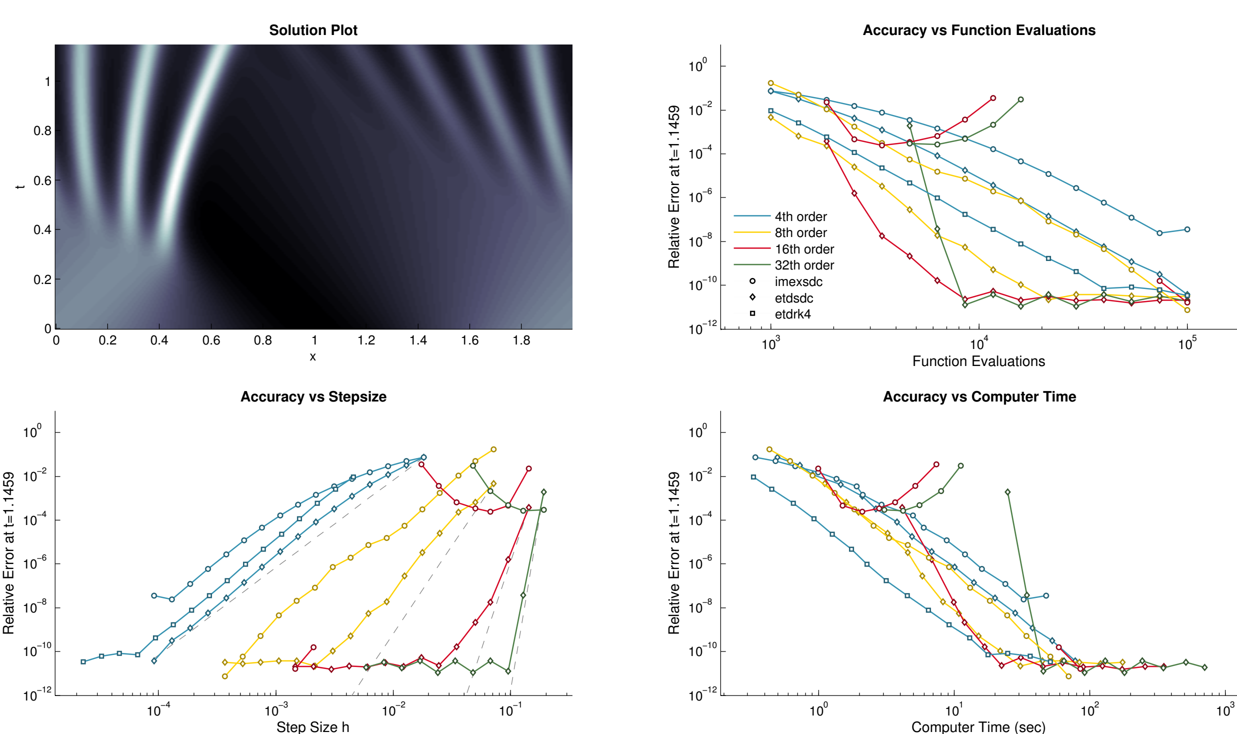


Figure 11.4: Performance results for the Quasigeostrophic and Korteweg-de Vries equations. Dashed lines of increasing steepness in the accuracy vs stepsize plots correspond to $O(h^4)$, $O(h^8)$ $O(h^{16})$ and $O(h^{32})$, respectively. Notice that high-order IMEXSDC schemes are unstable on the KDV equation. Order reduction does not occur for any method on the

The **Korteweg-de Vries** (KDV) equation describes weakly nonlinear shallow water waves. In 1965 Kruskal and Zabusky observed that smooth initial conditions could give rise to soliton solutions [99]. As in their original numerical experiment, we consider the KDV equation on a periodic domain

$$\begin{aligned} &u_t = -\left[\delta u_{xxx} + \tfrac{1}{2}(u^2)_x\right] \\ &u(x, t=0) = \cos(\pi x), \quad x \in [0, 2] \end{aligned}$$

where $\delta = 0.022$ and the simulation is run out to time $t = 3.6/\pi$. The eigenvalues of the linear terms are given by $\lambda(k, l) = \delta i k^3$; thus this equation possess a purely dispersive linear term. Unlike our previous examples, we solve this PDE in physical space where the resulting differentiation matrix is no longer diagonal. The nondiagonal case is more challenging since the coefficients $w_{i,n}$ in Eq. (11.27) are now matrix functions. In practice it would be more efficient to consider a lower-order spatial description and apply Krylov space or contour integral techniques that avoid explicitly initializing the requisite ETD matrices. Nevertheless, we consider this example to test the robustness of the scaling and squaring algorithm. For IMEXSDC_N^M schemes it is necessary to repeatedly solve the system $\Lambda x = f$ at each timestep. We perform an initial LU factorization of Λ to expedite this process. We present our numerical results for the KDV equation in Figure 11.4.

11.5.1 Discussion

Our results demonstrate that high-order methods can lead to significant speedup when solving nonlinear wave equations to high accuracy. Methods of order 8 and 16 were able to achieve the smallest error using the fewest function evaluations and the least overall CPU time. Interestingly, the error threshold separating good and bad performance for high and low order methods varied significantly in each experiment. Overall, ETDSDC methods consistently achieved better accuracy than corresponding IMEXSDC methods, and did not suffer from crippling order reduction on any of the problems we tested. Amongst the fourth order methods, ETDRK4 is more efficient than either ETDSDC_4^3 or IMEXSDC_4^3. Moreover, ETDRK4 is the fastest method for computing solutions if error tolerances are large. Methods of order 32 were generally less competitive than those of 8th or 16th order, and should only be considered in situations where extreme precision is necessary and quad/arbitrary-precision arithmetic allow for relative errors significantly below 1×10^{-12}. Finally, for diagonal Λ, the time required to initialize the ETD coefficients was insignificant as compared to overall computational time even for 32nd order method.

High-order ETDSDC methods continued to perform well even in the non-diagonal case, and we found no evidence of catastrophic roundoff error when forming the ETD matrix coefficients $w_{i,l}(h_i\Lambda)$. For nondiagonal Λ, high-order ETDSDC_N^M schemes require large amounts of memory and time to initialize and store the $N^2 - N$ requisite matrices. Moreover, the expensive matrix multiplications at each timestep reduced their overall competitiveness. To improve the performance of ETDSDC schemes on higher dimensional problems with non-

diagonal linear operators, it becomes essential to use techniques that avoid explicitly storing the ETD matrices.

High-order IMEXSDC schemes were unstable when solving the KDV equation on fine grids in both physical and Fourier space. Through additional numerical testing we find that IMEXSDC schemes can be unstable when integrating other nonlinear wave equations with dispersive linear terms such as the nonlinear Schrödinger equation.

We make several additional comments regarding our numerical experiments. The benefits of using high-order methods is greatly reduced if the initial conditions are not smooth, though in certain situations we found that high-order methods are rendered no less efficient than lower-order counterparts. The size of the integration window also affects the difference in performance between high and low-order methods, with the high-order methods generally benefiting on larger time domains. Chaotic equations can cause additional complications, as small perturbations due to rounding errors grow exponentially and contaminate overall accuracy. This was the case for the the KS equation where we were not able to integrate further without damaging the quality of the reference solution.

11.6 Conclusion

We have demonstrated that high-order ETD spectral deferred correction schemes possess excellent accuracy/stability properties and outperform existing ETD and IMEX methods when solving nonlinear wave equations to high accuracy. Our proposed methodology for initializing ETD coefficients is robust and can be successfully applied to ETDSDC schemes up to 32 order accuracy, even for equations with non-diagonal linear operator Λ. We have also highlighted the advantages of ETD spectral deferred correction methods as compared with IMEXSDC schemes. Our new ETD schemes consistently outperform their IMEX counterparts, do not appear to suffer from crippling order reduction, and retain stability on equations with dispersive linear terms.

Part III

CONCLUSIONS AND FUTURE WORK

Chapter 12

SUMMARY, CONCLUSIONS, AND FUTURE WORK

12.1 Summary

In this work we combined ideas from time-integration and approximation theory to develop a simple and elegant framework for constructing polynomial-based time-integrators. We began by introducing the ODE polynomial and the ODE dataset in chapter 3. These two objects form the basis of our framework, and have potential applications outside the realm of time-stepping. Additionally, the properties and definitions for ODE polynomials and ODE datasets arise naturally from approximation theory and are accessible to anyone with an understanding of elementary numerical analysis.

Next, in chapter 4 we applied the ODE polynomial and the ODE dataset to develop polynomial time integrators based on linear multistep methods, Runge-Kutta methods, block methods, and general linear methods. We also divided the space of polynomial integrators into iterators and propagators, and introduced special refiner and coarsener methods that provide important building blocks for constructing multistage methods.

Throughout this work, we have seen that polynomial block methods form an important pillar of our framework, since they are used to derive all other polynomial integrators. In chapter 5 we introduced a geometric construction strategy for polynomial block methods and derived a large parameter set that can be used to construct a range of integrators with either real or imaginary nodes. This procedure was extended in Chapter 6 to simplify the construction of polynomial general linear methods. Specifically, we introduced composition and successive iteration; two powerful and effective ways to combine block methods.

Next, we proceeded to illustrate a range of stability results and numerical experiments in chapters 7 and 8. The results of these studies showcase the superior properties of polynomial methods over classical linear multistep methods. Finally, in Chapter 9 we closed this work by extending the ODE polynomial and the ODE dataset to partitioned initial value problems, and to integral equations used to derive exponential integrators. These new definitions allowed us to construct new high-order implicit-explicit and exponential polynomial methods.

In Part II we extended the spectral deferred correction framework to encompass exponential integration, and compared the new exponential SDC schemes against existing IMEX SDC methods and existing exponential Runge-Kutta methods.

12.2 Conclusions

In this thesis, we introduced a methodology for constructing time integrators that combines ideas from approximation theory and complex analysis. Our approach eliminates the complexity of order conditions enabling the simplified construction of methods with a specific architecture (parallel or serial), degree of implicitness (explicit, diagonally implicit, fully implicit) and desired order of accuracy. Our geometric approach to method construction enabled us to derive various classes of high-order methods with improved linear stability regions compared to their classical counterparts. We illustrated the utility of this methodology by describing over 180 different generalizations of the classical BDF and Adams integrators.

In its totality, this work makes four primary contributions. First it offers a novel viewpoint for constructing, describing, and understanding time-integrators. This polynomial framework is accessible to a wide range of mathematicians and engineers, enabling non-experts to easily design custom integrators for their specific application problems. Second, this work provides a strong foundation for exploring polynomial-based exponential integrators, additive integrators, and other specialized methods. Third, this work pushes the boundaries of high-order integration for both classical and specialized integrators, and opens avenues for more easily exploring general linear methods. Finally, we have extended the utility of the spectral deferred correction framework and shown that it is possible to construct efficient high-order exponential SDC integrators.

12.3 Future Work

The generality of this polynomial framework raises many interesting theoretical and practical questions about the properties and utility of this approach. In particular there are three key areas that we plan to explore in future work:

1. **A careful analytical and numerical comparison of methods**

   We introduced 180 families of polynomial block methods but did not discuss their comparative performance. With such a large number of methods it is essential to develop an effective comparison metric to identify the advantages and drawbacks of each scheme and to characterize the most effective methods for certain problem classes. We are also interested in carefully comparing polynomial integrators to other state-of-the-art integrators.

2. **Variable time-stepping**

   Time integrators with an adaptive time step allow for computational savings in addition to guaranteeing local error properties. We plan to investigate automated strategies for choosing the extrapolation parameter α, and discuss how our framework can be used to simplify adaptive time-stepping for multivalue methods.

3. **Strategies for time-parallelization**

   Time-parallelization can offer additional avenues for speedup in situations where spatial parallelism has been saturated. In this work, we constructed time-integrators that allow for small-scale parallelism within the method. We plan to continue developing parallel polynomial time-integrators, and to carefully test the efficiency of parallelization on a range of application problems.

BIBLIOGRAPHY

[1] M. J. Ablowitz and A. S. Fokas. *Complex variables: introduction and applications.* Cambridge University Press, 2003.

[2] U. M. Ascher, S. J. Ruuth, and B. Wetton. Implicit-explicit methods for time-dependent partial differential equations. *SIAM Journal on Numerical Analysis*, 32(3):797–823, 1995.

[3] H. A. Ashi, L. J. Cummings, and P. C. Matthews. Comparison of methods for evaluating functions of a matrix exponential. *Applied Numerical Mathematics*, 59(3):468–486, 2009.

[4] H. A. Ashi, L. J. Cummings, and P. C. Matthews. Comparison of methods for evaluating functions of a matrix exponential. *Applied Numerical Mathematics*, 59(3):468–486, 2009.

[5] A. P. Austin, P. Kravanja, and L. N. Trefethen. Numerical algorithms based on analytic function values at roots of unity. *SIAM Journal on Numerical Analysis*, 52(4):1795–1821, 2014.

[6] H Berland, B Owren, and B Skaflestad. B-series and order conditions for exponential integrators. *SIAM Journal on Numerical Analysis*, 43(4):1715–1727, 2005.

[7] H. Berland, B. Skaflestad, and W. M. Wright. EXPINT—a MATLAB package for exponential integrators. *ACM Transactions on Mathematical Software (TOMS)*, 33(1):4, 2007.

[8] D. Bessis and J.D. Fournier. Pole condensation and the Riemann surface associated with a shock in Burgers' equation. *Journal de Physique Lettres*, 45(17):833–841, 1984.

[9] G. Beylkin, J. M. Keiser, and L. Vozovoi. A new class of time discretization schemes for the solution of nonlinear PDEs. *Journal of Computational Physics*, 147(2):362–387, 1998.

[10] F. Bornemann. Accuracy and stability of computing high-order derivatives of analytic functions by Cauchy integrals. *Foundations of Computational Mathematics*, 11(1):1–63, 2011.

[11] F. Bornemann and G. Wechslberger. Optimal contours for high-order derivatives. *IMA Journal of Numerical Analysis*, 33(2):403, 2013.

[12] J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover, New York, 2013.

[13] J. C. Butcher. On the convergence of numerical solutions to ordinary differential equations. *Mathematics of Computation*, 20(93):1–10, 1966.

[14] J. C. Butcher. General linear method: A survey. *Applied Numerical Mathematics*, 1(4):273–284, 1985.

[15] J. C. Butcher. An introduction to DIMSIMs. *Comp. Appl. Math*, 14:59–72, 1995.

[16] J. C. Butcher. General linear methods. *Acta Numerica*, 15:157–256, 2006.

[17] J. C. Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[18] T. Buvoli. Codebase for ETD Spectral Deferred Correction Methods. Nov 2014.

[19] T. Buvoli. A class of exponential integrators based on spectral deferred correction. *arXiv preprint arXiv:1504.05543*, 2015.

[20] T. Buvoli and M. Tokman. Constructing time integrators using interpolating polynomials. *Submitted*, 2018.

[21] A. Christlieb, M. Morton, B. Ong, and J. Qiu. Semi-implicit integral deferred correction constructed with additive Runge-Kutta methods. *Commun. Math. Sci*, 9:879–902, 2011.

[22] A. Christlieb, B. Ong, and J. Qiu. Spectral deferred correction methods with high order Runge-Kutta schemes in prediction and correction steps. *Commun. Appl. Math. Comput. Sci*, 4:27–56, 2009.

[23] A. J. Christlieb, C. B. Macdonald, and B. W. Ong. Parallel high-order integrators. *SIAM Journal on Scientific Computing*, 32(2):818–835, 2010.

[24] M. T. Chu and H. Hamilton. Parallel solution of ODEs by multiblock methods. *SIAM journal on scientific and statistical computing*, 8(3):342–353, 1987.

[25] G. F. Corliss. Integrating ODEs in the complex plane—pole vaulting. *Mathematics of Computation*, 35(152):1181–1189, 1980.

[26] S. M. Cox and P. C. Matthews. Exponential time differencing for stiff systems. *Journal of Computational Physics*, 176(2):430–455, 2002.

[27] P. J. Davis and P. Rabinowitz. *Methods of numerical integration*. Courier Corporation, 2007.

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

[29] 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.

[30] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder. Parallel time integration with multigrid. *SIAM Journal on Scientific Computing*, 36(6):C635–C661, 2014.

[31] B. Fornberg. Numerical differentiation of analytic functions. *ACM Transactions on Mathematical Software (TOMS)*, 7(4):512–526, 1981.

[32] B. Fornberg. Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of Computation*, 51(184):699–706, 1988.

[33] B. Fornberg. Calculation of weights in finite difference formulas. *SIAM review*, 40(3):685–691, 1998.

[34] B. Fornberg. *A Practical Guide to Pseudospectral Methods*, volume 1. Cambridge University Press, 1998.

[35] B. Fornberg and J.A.C. Weideman. A numerical methodology for the Painlevé equations. *Journal of Computational Physics*, 230(15):5957–5973, 2011.

[36] M. J. Gander. 50 years of time parallel time integration. In *Multiple Shooting and Time Domain Decomposition Methods*, pages 69–113. Springer, 2015.

[37] C. W. Gear. Parallel methods for ordinary differential equations. *Calcolo*, 25(1):1–20, 1988.

[38] I. Grooms and K. Julien. Linearly implicit methods for nonlinear PDEs with linear dispersion and dissipation. *Journal of Computational Physics*, 230(9):3630–3650, 2011.

[39] M. Günther and A. Sandu. Multirate generalized additive runge kutta methods. *Numerische Mathematik*, 133(3):497–524, Jul 2016.

[40] S. Güttel and G. Klein. Efficient high-order rational integration and deferred correction with equispaced data. *In preparation*, 2013.

[41] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving ordinary differential equations I: nonstiff problems*, volume 1. Springer Science & Business, 2008.

[42] E. Hairer and G. Wanner. *Solving ordinary differential equations II: Stiff and differential-algebraic problems*. Springer, 1996.

[43] A. C. Hansen and J. Strain. On the order of deferred correction. *Applied Numerical Mathematics*, 61(8):961–973, 2011.

[44] E. Hansen and A. Ostermann. High order splitting methods for analytic semigroups exist. *BIT Numerical Mathematics*, 49(3):527–542, Sep 2009.

[45] Ch. Hermite. Sur la formule d'interpolation de lagrange.(extrait d'une lettre de M. Ch. Hermite à M. Borchardt). *Journal für die reine und angewandte Mathematik*, 84:70–79, 1877.

[46] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM review*, 51(4):747–764, 2009.

[47] M. Hochbruck and C. Lubich. On Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 34(5):1911–1925, 1997.

[48] M. Hochbruck, C. Lubich, and H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM Journal on Scientific Computing*, 19(5):1552–1574, 1998.

[49] M. Hochbruck and A. Ostermann. Explicit exponential Runge–Kutta methods for semilinear parabolic problems. *SIAM Journal on Numerical Analysis*, 43(3):1069–1090, 2005.

[50] M. Hochbruck and A. Ostermann. Exponential Runge–Kutta methods for parabolic problems. *Applied Numerical Mathematics*, 53(2):323–339, 2005.

[51] M. Hochbruck and A. Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

[52] M. Hochbruck and A. Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

[53] M. Hochbruck, A. Ostermann, and J. Schweitzer. Exponential Rosenbrock-type methods. *SIAM Journal on Numerical Analysis*, 47(1):786–803, 2009.

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

[55] A. Kassam and L.N Trefethen. Fourth-order time stepping for stiff PDEs. *SIAM J. Sci. Comput*, 26:1214–1233, 2005.

[56] C. A. Kennedy and M. H. Carpenter. Additive Runge-Kutta schemes for convection-diffusion-reaction equations. 2001.

[57] C. A. Kennedy and M. H. Carpenter. Additive Runge-Kutta schemes for convection–diffusion–reaction equations. *Applied Numerical Mathematics*, 44(1):139 – 181, 2003.

[58] S. Koikari. Rooted tree analysis of Runge–Kutta methods with exact treatment of linear terms. *Journal of computational and applied mathematics*, 177(2):427–453, 2005.

[59] S. Koikari. An error analysis of the modified scaling and squaring method. *Computers & Mathematics with Applications*, 53(8):1293–1305, 2007.

[60] S. Krogstad. Generalized integrating factor methods for stiff pdes. *Journal of Computational Physics*, 203(1):72–88, 2005.

[61] S. Krogstad. Generalized integrating factor methods for stiff PDEs. *Journal of Computational Physics*, 203(1):72–88, 2005.

[62] Y. Kuramoto and T. Tsuzuki. Persistent propagation of concentration waves in dissipative media far from thermal equilibrium. *Progress of theoretical physics*, 55(2):356–369, 1976.

[63] J. D. Lawson. Generalized Runge-Kutta processes for stable systems with large Lipschitz constants. *SIAM Journal on Numerical Analysis*, 4(3):372–380, 1967.

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

[65] J. L. Lions, Y. Maday, and G. Turinici. A "parareal" in time discretization of pdes ,. *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 332(7):661–668, 2001.

[66] Y. Liu, C. Shu, and M. Zhang. Strong stability preserving property of the deferred correction time discretization. *Journal of Computational Mathematics*, 26(5), 2008.

[67] J. Loffeld and M. Tokman. Comparative performance of exponential, implicit, and explicit integrators for stiff systems of ODEs. *Journal of Computational and Applied Mathematics*, 241:45–67, 2013.

[68] V. T. Luan and A. Ostermann. Explicit exponential rungekutta methods of high order for parabolic problems. *Journal of Computational and Applied Mathematics*, 256:168 – 179, 2014.

[69] V. T. Luan and A. Ostermann. *Stiff Order Conditions for Exponential Runge-Kutta Methods of Order Five*, pages 133–143. Springer International Publishing, Cham, 2014.

[70] J. N. Lyness and C. B. Moler. Numerical differentiation of analytic functions. *SIAM Journal on Numerical Analysis*, 4(2):202–210, 1967.

[71] B. V. Minchev and Wright W. M. A review of exponential integrators for first order semi-linear problems. *Preprint, NTNU Trondheim*, 2005.

[72] M. Minion. Semi-implicit spectral deferred correction methods for ordinary differential equations. *Communications in Mathematical Sciences*, 1(3):471–500, 09 2003.

[73] M. Minion. Semi-implicit spectral deferred correction methods for ordinary differential equations. *Communications in Mathematical Sciences*, 1(3):471–500, 09 2003.

[74] H. Montanelli and Y. Nakatsukasa. Fourth-order time-stepping for stiff pdes on the sphere. *SIAM Journal on Scientific Computing*, 40(1):A421–A451, 2018.

[75] V. N. Nikolaevskiy. Dynamics of viscoelastic media with internal oscillators. In *Lecture Notes in Engineering*, volume 39, pages 210–221. Springer-Verlag, Berlin, 1989.

[76] G. F. Norris. Spectral integration and the numerical solution of two-point boundary value problems. *Master's Thesis, Oregon State University*, 1999.

[77] S. P. Nørsett and A. Wolfbrandt. Order conditions for Rosenbrock type methods. *Numerische Mathematik*, 32(1):1–15, 1979.

[78] A. Ostermann, M. Thalhammer, and W. M. Wright. A class of explicit exponential general linear methods. *BIT Numerical Mathematics*, 46(2):409–431, 2006.

[79] L. Pareschi and G. Russo. Implicit-explicit Runge-Kutta schemes for stiff systems of differential equations. *Recent trends in numerical analysis*, 3:269–289, 2000.

[80] J. Pedlosky. *Geophysical Fluid Dynamics*. Springer-Verlag, New York, 1987.

[81] G. Rainwater and M. Tokman. A new class of split exponential propagation iterative methods of Runge-Kutta type (sEPIRK) for semilinear systems of odes. *Journal of Computational Physics*, 269:40–60, 2014.

[82] G. Rainwater and M. Tokman. A new approach to constructing efficient stiffly accurate EPIRK methods. *Journal of Computational Physics*, 323(Supplement C):283 – 309, 2016.

[83] A. Sandu and M. Günther. A class of generalized additive Runge-Kutta methods. *CoRR*, abs/1310.5573, 2013.

[84] L. F. Shampine. Implementation of Rosenbrock methods. *ACM Transactions on Mathematical Software (TOMS)*, 8(2):93–113, 1982.

[85] L. F. Shampine and H. A. Watts. Block implicit one-step methods. *Mathematics of Computation*, 23(108):731–740, 1969.

[86] C. E. Shin. Generalized Hermite interpolation and sampling theorem involving derivatives. *Communications-Korean Mathematical Society*, 17(4):731–740, 2002.

[87] E. Simbawa, P. C. Matthews, and S. M. Cox. Nikolaevskiy equation with dispersion. *Physical Review E*, 81(3):036220, 2010.

[88] B. Skaflestad and W. M. Wright. The scaling and modified squaring method for matrix functions related to the exponential. *Applied Numerical Mathematics*, 59(3):783–799, 2009.

[89] R. Speck, D. Ruprecht, M. Emmett, M. Minion, M. Bolten, and R. Krause. A multilevel spectral deferred correction method. *BIT Numerical Mathematics*, 55(3):843–867, 2015.

[90] J. Strikwerda. *Finite Difference Schemes and Partial Differential Equations, Second Edition*. Society for Industrial and Applied Mathematics, 2004.

[91] 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):1–13, 2013.

[92] M. Tokman. Efficient integration of large stiff systems of odes with exponential propagation iterative (epi) methods. *Journal of Computational Physics*, 213(2):748–776, 2006.

[93] L. N. Trefethen. *Spectral Methods in MATLAB*, volume 10. SIAM, Philadelphia, 2000.

[94] L. N. Trefethen. Evaluating matrix functions for exponential integrators via Carathéodory–Fejér approximation and contour integrals. *Electronic Transactions on Numerical Analysis*, 29:1–18, 2007.

[95] L. N. Trefethen. *Approximation theory and approximation practice*. SIAM, Philadelphia, 2013.

[96] L. N. Trefethen and J.A.C. Weideman. The exponentially convergent trapezoidal rule. *SIAM Review*, 56(3):385–458, 2014.

[97] P. van der Houwen and E. Messina. Parallel Adams methods. *Journal of Computational and Applied Mathematics*, 101(1-2):153–165, 1999.

[98] P. J. van der Houwen, B. P. Sommeijer, and W. A. van der Veen. Parallelism across the steps in iterated Runge-Kutta methods for stiff initial value problems. *Numerical Algorithms*, 8(2):293–312, 1994.

[99] N. J. Zabusky and M. D. Kruskal. Interaction of solitons in a collisionless plasma and the recurrence of initial states. *Phys. Rev. Lett*, 15(6):240–243, 1965.