Adaptive Solvers for Partial Differential and Differential-Algebraic Equations

by

Aaditya Viswanath Rangan

AB (Dartmouth College) June 1999

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

in

MATHEMATICS

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor J Strain, Chair
Professor J Neu
Professor P Papadopoulos

Fall 2003

Adaptive Solvers for Partial Differential and Differential-Algebraic Equations

Aaditya Viswanath Rangan

Abstract

Adaptive Solvers for Partial Differential and Differential-Algebraic Equations

by

Aaditya Viswanath Rangan

Doctor of Philosophy in MATHEMATICS

University of California at Berkeley

Professor J Strain, Chair

Numerical methods which solve partial differential and differential-algebraic equations are constructed and analyzed. An implicit Euler scheme in time approximates partial differential equations by a sequence of two-point boundary value problems. A fast spectral integral technique solves these boundary value problems, a residual is computed, and the accuracy is increased by spectral deferred correction. Convergence is proven for special cases. A spectral deferred correction scheme for differential algebraic equations is constructed and analyzed, and convergence is proven for the index-1 case. Numerical results demonstrate the effectiveness of both methods.

Professor J Strain
Dissertation Committee Chair

Contents

1 Introduction 1

2 Mathematical Preliminaries 3
- 2.1 Numerical interpolation and quadrature . . . 3
- 2.2 Linear two-point boundary value problem solver . . . 4
  - 2.2.1 Green's functions for homogeneous two-point boundary value problems . . . 5
  - 2.2.2 Review of the spectral integral approach . . . 7

3 Spectral Deferred Correction Methods 20
- 3.1 Spectral deferred correction for ODEs . . . 20
- 3.2 Spectral deferred correction for PDEs . . . 25
- 3.3 Spectral deferred correction for DAEs . . . 28
  - 3.3.1 Index-1 DAEs . . . 28
  - 3.3.2 Index-2 DAEs . . . 30
  - 3.3.3 Algorithm description . . . 31

4 Automated Coordinate Changes 34

5 Initial-Boundary Value Problem for Evolutionary PDEs 46
- 5.1 Outline of the numerical method . . . 47

6 Convergence Proofs 51
- 6.1 Truncation error . . . 51
- 6.2 Boundary value problem error . . . 52
- 6.3 Convergence for symmetric linear hyperbolic flux . . . 53
- 6.4 Convergence for linear strictly-hyperbolic flux . . . 54
- 6.5 Convergence for one dimensional conservative flux . . . 55

7 Algorithm Description 64

8 Numerical Examples 68
- 8.1 Differential algebraic equations . . . 68
- 8.2 Automated coordinate change . . . 75
- 8.3 Partial differential equations . . . 77

9 Conclusions 87

Bibliography 89

Acknowledgements

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 0209617. Any opinions, findings and conclusions or recomendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

Chapter 1

Introduction

This thesis presents fast accurate new methods for the solution of 1 + 1 dimensional partial differential equations (PDEs) and differential-algebraic equations (DAEs). Our method for solving PDEs has the following properties:

1. Variable Order: Obtaining an ϵ-accurate solution in the 2-norm costs $o(\log(\epsilon))$.

2. Adaptive: The scheme efficiently resolves small-scale features that are localized in space or time.

3. General: The method is easily applied to arbitrary systems with general boundary data. The user need specify only the right-hand side of the PDE, the initial and boundary data and error tolerances.

In contrast to the method of lines [42, 18] and the Galerkin method [27, 29, 6], we discretize time first. Implicit Euler is used to approximate the PDE by a sequence of linear two-point boundary value problems, which may be solved using the best available scheme. We use an adaptive spectral integral scheme to automatically generate a spatial grid and numerical solution at each time-step. Working within a spectral deferred correction framework, we build a high-order temporal scheme by successively correcting our approximate solution. Along the way, we extend spectral deferred correction to solve DAEs and prove convergence for index-1 DAEs. We use the successive improvements

of our numerical solution as error estimates for adapting the time-step and obtaining the desired accuracy. We prove that this method converges for important special cases, and demonstrate its effectiveness with several numerical examples.

Chapter 2

Mathematical Preliminaries

2.1 Numerical interpolation and quadrature

We will approximate functions $f : [a, c] \rightarrow R$ and their integrals $\int_a^c f dt$ by standard interpolation and quadrature schemes based on Chebyshev polynomials. Standard techniques [10] produce weights w_j, S^r_{kj}, S^l_{kj} such that for any polynomial p of degree $n-1$ with values $p_j = p(t_j)$ at Chebyshev nodes

$$t_j = \cos\left(\pi \frac{2j-1}{2n}\right)(c-a)/2 + (c+a)/2 \qquad j = 1, 2, \ldots, n, \tag{2.1.1}$$

the definite and indefinite integrals

$$\int_a^c p(t)dt = \sum_{j=1}^n w_j p_j \tag{2.1.2}$$

$$\int_a^{t_k} p(t)dt = \sum_{j=1}^n S^l_{kj} p_j \tag{2.1.3}$$

$$\int_{t_k}^c p(t)dt = \sum_{j=1}^n S^r_{kj} p_j \tag{2.1.4}$$

are exact. The function values $f_j = f(t_j)$ are used to construct a unique degree $n-1$ polynomial interpolant p_f. Integrating p_f with Eqn. (2.1.2) or Eqns. (2.1.3, 2.1.4) gives a precision n approximation to the integral of f [38, 13]. We will use this quadrature scheme to solve semi-separable integral equations of the form

$$\sigma(x) + M(x)\left(\int_a^x L(t)\sigma(t)dt + \int_x^c R(t)\sigma(t)\right) = f(x) \tag{2.1.5}$$

for the density $\sigma(x)$: We approximate the integral equation by

$$\sigma_k + M_k \sum_{j=1}^{n} (S^l_{kj} L_j \sigma_j + S^r_{kj} R_j \sigma_j) \quad = \quad f_k. \tag{2.1.6}$$

Eqn. (2.1.6) can be rewritten as

$$(I + M(S^l L + S^r R))\hat{\sigma} = \hat{f}, \tag{2.1.7}$$

where $\hat{\sigma} = (\sigma_1, \ldots, \sigma_n)^{\mathrm{T}}$ and $\hat{f} = (f_1, \ldots, f_n)$. L,R and M are diagonal matrices containing values of the kernels L_j,R_j and M_j.

2.2 Linear two-point boundary value problem solver

Two-point boundary value problems can be solved by a fast spectral integral technique [37, 25], and we briefly review the necessary background here. Let $L(R^n)$ be the set of $n \times n$ real matrices. A linear two-point boundary value problem requires a differentiable vector function $\Phi : [a, c] \to R^n$ to satisfy

$$\begin{aligned} \Phi'(x) + p(x)\Phi(x) &= f(x) \\ A\Phi(a) + C\Phi(c) &= \gamma, \end{aligned} \tag{2.2.1}$$

given a continuous coefficient matrix $p : [a, c] \to L(R^n)$, forcing vector $f : [a, c] \to R^n$, boundary matrices $A, C \in L(R^n)$ and boundary vector $\gamma \in R^n$, If γ is zero, then the two-point boundary value problem is homogeneous, and we can construct a solution using Green's functions. If γ is nonzero we can often apply the transformation

$$\begin{aligned} \nu &= (A + C)^{-1}\gamma \\ \hat{f}(x) &= f(x) - p(x)\nu \\ \hat{\Phi}(x) &= \Phi(x) - \nu. \end{aligned} \tag{2.2.2}$$

Then $\hat{\Phi}$ satisfies a homogeneous two-point boundary value problem

$$\begin{aligned} \hat{\Phi}'(x) + p(x)\hat{\Phi}(x) &= \hat{f}(x) \\ A\hat{\Phi}(a) + C\hat{\Phi}(c) &= 0. \end{aligned} \tag{2.2.3}$$

This fails if the matrix $D = A + C$ is singular, but if $[A|C]$ has rank n we can construct an invertible local coordinate transformation

$$\mathcal{T}(x) : [a, c] \to R^{n \times n} \tag{2.2.4}$$

such that $A\mathcal{T}(a) + C\mathcal{T}(c)$ is nonsingular (see Chapter 4). Given $\mathcal{T}$, the transformed solution

$$\tilde{\Phi}(x) = \mathcal{T}(x)^{-1}\Phi(x) \tag{2.2.5}$$

satisfies

$$\begin{gathered}\tilde{\Phi}'(x) + \mathcal{T}(x)^{-1}(\mathcal{T}'(x) + p(x)\mathcal{T}(x))\tilde{\Phi}(x) = \mathcal{T}(x)^{-1}f(x) \\ (A\mathcal{T}(a))\tilde{\Phi}(a) + (C\mathcal{T}(c))\tilde{\Phi}(c) = \gamma.\end{gathered} \tag{2.2.6}$$

2.2.1 Green's functions for homogeneous two-point boundary value problems

The homogenous two-point boundary value problem can be converted into an integral equation via Green's functions. A Green's function for Eqn. (2.2.3) is the matrix-valued function of x and t that satisfies the boundary value problem in x

$$\begin{gathered}\partial_x G(x,t) + p(x)G(x,t) = \delta(x-t) \\ AG(a,t) + CG(c,t) = 0\end{gathered} \tag{2.2.7}$$

for each fixed t. The Green's function has the following properties, which are a direct consequence of the properties of the δ-function:

1. $\partial_x G(x,t)$ is continuous except at $x = t$
2. $\lim_{h\to 0+} G(x+h,x) - G(x-h,x) = I \qquad a \le x < c$
3. $\partial_x G(x,t) + p(x)G(x,t) = 0 \qquad a \le t \le c \qquad t \ne x$
4. $AG(a,t) + CG(c,t) = 0 \qquad a \le t \le c$

Once we have the Green's function, any linear homogeneous problem (2.2.1) has solution

$$\Phi(x) = \int_a^c G(x,t)f(t)dt. \tag{2.2.8}$$

Green's functions for arbitrary p are constructed from a fundamental matrix $\Upsilon(x)$, any nonsingular matrix satisfying the ODE

$$\Upsilon'(x) + p(x)\Upsilon(x) = 0, \tag{2.2.9}$$

and a fundamental boundary matrix

$$D_\Upsilon = A\Upsilon(a) + C\Upsilon(c) \tag{2.2.10}$$

(If D_Υ is singular we transform coordinates or choose a new Υ by the techniques of Chapter 4). The Green's function is then

$$G(x,t) = \begin{cases} \Upsilon(x)D_\Upsilon^{-1}A\Upsilon(a)\Upsilon^{-1}(t) & t \le x \\ -\Upsilon(x)D_\Upsilon^{-1}C\Upsilon(c)\Upsilon^{-1}(t) & t \ge x. \end{cases} \tag{2.2.11}$$

The following general procedure transforms the boundary value problem of Eqn. (2.2.3) into a second-kind integral equation. Suppose Υ is a fundamental matrix for

$$\Upsilon'(x) + p_0\Upsilon(x) = 0. \tag{2.2.12}$$

The background coefficient matrix p_0 is arbitrary (usually we choose $p_0 = 0$). Let G_0 be the Green's function for

$$\begin{aligned} \partial_x G_0(x,t) + p_0 G_0(x,t) &= \delta(x-t) \\ AG_0(a,t) + CG_0(c,t) &= 0, \end{aligned} \tag{2.2.13}$$

assuming that both $D = (A+C)$ and D_Υ are nonsingular. Then we seek the solution to the original boundary value problem Eqn. (2.2.3) as the integral of G against some density function σ to be found later:

$$\Phi(x) = \int_a^c G_0(x,t)\sigma(t)dt \tag{2.2.14}$$

Substituting Eqn. (2.2.14) into Eqn. (2.2.3) and using the fundamental theorem of calculus (or Leibniz' rule for the differentiation of integrals) yields

$$\begin{aligned} f(x) &= \partial_x \textstyle\int_a^c G_0(x,t)\sigma(t)dt + p(x)\int_a^c G_0(x,t)\sigma(t)dt \\ &= \partial_x \textstyle\int_a^x G_0(x,t)\sigma(t)dt + \partial_x \int_x^c G_0(x,t)\sigma(t)dt \\ &\quad + p(x)\textstyle\int_a^c G_0(x,t)\sigma(t)dt = f(x) \\ &= \lim_{h\to 0^+}(G_0(x,x-h)\sigma(x-h) - G_0(x,x+h)\sigma(x+h)) \\ &\quad + \textstyle\int_a^c(\partial_x G_0(x,t) + p(x)G_0(x,t))\sigma(t)dt \\ f(x) &= \sigma(x) + (p(x) - p_0(x))\textstyle\int_a^c G_0(x,t)\sigma(t)dt. \end{aligned} \tag{2.2.15}$$

The most useful background coefficient is usually $p_0 = 0$, with corresponding fundamental matrix $\Upsilon = I$ and background Green's function

$$G = \begin{cases} D^{-1}A & t \leq x \\ D^{-1}C & t \geq x. \end{cases} \tag{2.2.16}$$

The background coefficient

$$p_\gamma = \begin{bmatrix} 0 & -I \\ -\gamma^2 I & 0 \end{bmatrix} \tag{2.2.17}$$

with corresponding fundamental solution

$$\Upsilon_\gamma = \begin{bmatrix} \cosh(x\gamma) & \frac{1}{\gamma}\sinh(x\gamma) \\ \gamma\sinh(x\gamma) & \cosh(x\gamma) \end{bmatrix} \tag{2.2.18}$$

is useful for diffusion problems (Chapter 5).

2.2.2 Review of the spectral integral approach

The homogeneous boundary value problem with nonsingular matrices D and D_Υ of Eqn. (2.2.15) can now be solved numerically by discretizing via the techniques of Section 2.1. Eqn. (2.2.15) can be rewritten as

$$\sigma(x) + M(x)(\int_a^x L(t)\sigma(t)dt + \int_x^c R(t)\sigma(t)dt) = f(x), \tag{2.2.19}$$

where

$$\begin{aligned} M(x) &= (p(x) - p_0(x))\Upsilon(x)D_\Upsilon^{-1} \\ L(t) &= A\Upsilon(a)\Upsilon^{-1}(t) \\ R(t) &= -C\Upsilon(c)\Upsilon^{-1}(t). \end{aligned} \tag{2.2.20}$$

Given n quadrature nodes $x_1, \ldots, x_n$ on the interval $[a, c]$, Eqns. (2.1.5 – 2.1.7) yield

$$\hat{\sigma} = (I + M(S^l L + S^r R))^{-1} f, \tag{2.2.21}$$

where $\hat{\sigma} = (\sigma(x_1), \ldots, \sigma(x_n))^{\mathrm{T}}$ and $f = (f(x_1), \ldots, f(x_n))^{\mathrm{T}}$. L is the matrix $\mathrm{diag}(L(x_1), \ldots, L(x_n))$, R and M are similar. Unfortunately, solving the linear system in Eqn. (2.2.21) requires $O(n^3)$ operations. This is prohibitive for large n so we employ th recursive splitting approach of [37, 25]:

Given two subintervals $[a,b]$ and $[b,c]$ with $a < b < c$, we construct the solution in $[a,c]$ from the solutions of similar problems on $[a,b]$ and $[b,c]$. This essentially decomposes the linear integral operator in Eqn. (2.2.15) into smaller blocks. Each of these blocks is again decomposed by the same procedure. This yields a very fast scheme for solving Eqn. (2.2.15).

We first seek a density σ satisfying Eqn. (2.2.15). For simplicity define $\hat{p} = p - p_0$, so $\sigma : [a,c] \rightarrow R^n$ satisfies

$$\sigma(x) + \hat{p}(x) \int_a^c G(x,t)\sigma(t)dt = f(x). \tag{2.2.22}$$

Split the interval $[a,c]$ into two sub-intervals $I_1 = [a,b]$ and $I_2 = [b,c]$ and let the restrictions of σ to each sub-interval be

$$\sigma_1(x) = \sigma(x)|_{I_1} : I_1 \rightarrow R^n \tag{2.2.23}$$

$$\sigma_2(x) = \sigma(x)|_{I_2} : I_2 \rightarrow R^n. \tag{2.2.24}$$

We calculate σ_1 and σ_2 on the sub-intervals I_1 and I_2 via the following numerical apparatus. Assume that we are capable of applying the operator

$$P(\sigma)(x) = \sigma(x) + \hat{p}(x) \int_a^c G(x,t)\sigma(t)dt. \tag{2.2.25}$$

Similarly, assume that we can apply the following operators

$$\begin{aligned}
&P_{11} : L^2(I_1) \rightarrow L^2(I_1) \qquad P_{11}(\sigma)(x) = \sigma(x) + \hat{p}(x) \textstyle\int_a^b G(x,t)\sigma(t)dt \\
&P_{22} : L^2(I_2) \rightarrow L^2(I_2) \qquad P_{22}(\sigma)(x) = \sigma(x) + \hat{p}(x) \textstyle\int_b^c G(x,t)\sigma(t)dt \\
&P_{12} : L^2(I_1) \rightarrow L^2(I_2) \qquad P_{12}(\sigma)(x) = \hat{p}(x) \textstyle\int_a^b G(x,t)\sigma(t)dt \\
&P_{21} : L^2(I_2) \rightarrow L^2(I_1) \qquad P_{21}(\sigma)(x) = \hat{p}(x) \textstyle\int_b^c G(c,t)\sigma(t)dt.
\end{aligned} \tag{2.2.26}$$

Write for convenience

$$G(x,t) = \begin{cases} \Upsilon(x)v_l(t) & t \leq x \\ \Upsilon(x)v_r(t) & t \geq x. \end{cases} \tag{2.2.27}$$

Now

$$P\sigma = f \tag{2.2.28}$$

is equivalent to two problems

$$\begin{cases} P_{11}\sigma_1 + P_{21}\sigma_2 = f_1 & \text{on } L \\ P_{12}\sigma_1 + P_{22}\sigma_2 = f_2 & \text{on } R \end{cases} \tag{2.2.29}$$

on the subintervals. Inverting the left and right operators P_{11} and P_{22} yields

$$\begin{cases} \sigma_1 + P_{11}^{-1}P_{21}\sigma_2 = P_{11}^{-1}f_1 \\ P_{22}^{-1}P_{12}\sigma_1 + \sigma_2 = P_{22}^{-1}f_2. \end{cases} \tag{2.2.30}$$

Applying block Gaussian elimination to Eqn. (2.2.30) gives

$$\begin{cases} \sigma_1 - P_{11}^{-1}P_{21}P_{22}^{-1}P_{12}\sigma_1 = P_{11}^{-1}f_1 - P_{11}^{-1}P_{21}P_{22}^{-1}f_2 \\ \sigma_2 - P_{22}^{-1}P_{12}P_{11}^{-1}P_{21}\sigma_2 = P_{22}^{-1}f_2 - P_{22}^{-1}P_{12}P_{11}^{-1}f_1. \end{cases} \tag{2.2.31}$$

Following [37], we define local data vectors

$$\begin{aligned} \eta_1 &= P_{11}^{-1}f_1 \\ \eta_2 &= P_{22}^{-1}f_2 \\ \delta_{l1} &= \int_a^b v_l(t)\eta_1(t)dt \\ \delta_{l2} &= \int_b^c v_l(t)\eta_2(t)dt \\ \delta_{r1} &= \int_a^b v_r(t)\eta_1(t)dt \\ \delta_{r2} &= \int_b^c v_r(t)\eta_2(t)dt \end{aligned} \tag{2.2.32}$$

and local matrices

$$\begin{aligned} \psi_1(x) &= p_0(x)\Upsilon(x) \quad x \in I_1 \\ \psi_2(x) &= p_0(x)\Upsilon(x) \quad x \in I_2 \\ \phi_1 &= P_{11}^{-1}\psi_1 \\ \phi_2 &= P_{22}^{-1}\psi_2 \\ \alpha_{l1} &= \int_a^b v_l(t)\phi_1(t)dt \\ \alpha_{l2} &= \int_b^c v_l(t)\phi_2(t)dt \\ \alpha_{r1} &= \int_a^b v_r(t)\phi_1(t)dt \\ \alpha_{r2} &= \int_b^c v_r(t)\phi_2(t)dt \end{aligned} \tag{2.2.33}$$

If the subintervals I_1 and I_2 are small then the values of the various $\eta, \delta, \phi\psi$ and α can be calculated directly via Eqns. (2.2.19 – 2.2.21) with a small to moderate number of nodes.

It is also convenient to adopt the inner product notation

$$\begin{aligned} v^{\mathrm{T}}\sigma &= \int_I v(t)\sigma(t)dt \\ P_{12} &= \psi_2 v_l^{\mathrm{T}} \\ P_{21} &= \psi_1 v_r^{\mathrm{T}} \end{aligned} \tag{2.2.34}$$

where I is the domain of v. The first line of Eqn. (2.2.31) now becomes

$$\sigma_1 - P_{11}^{-1}\psi_1 v_r^{\mathrm{T}} P_{22}^{-1} P_{12}\sigma_1 = \eta_1 - P_{11}^{-1}\psi_1 v_r^{\mathrm{T}}\eta_2 \tag{2.2.35}$$

$$(I - \phi_1 v_r^{\mathrm{T}}\phi_2 v_l^{\mathrm{T}})\sigma_1 = \eta_1 - \phi_1 v_r^{\mathrm{T}}\eta_2 \tag{2.2.36}$$

$$(I - \phi_l \alpha_{r2} v_l^{\mathrm{T}})\sigma_1 = \eta_1 - \phi_l \delta_{r2}. \tag{2.2.37}$$

Similarly the second line of Eqn. (2.2.31) becomes

$$(I - \phi_2 \alpha_{l1} v_r^{\mathrm{T}})\sigma_2 = \eta_2 - \phi_2 \delta_{l1}. \tag{2.2.38}$$

Note that for any two vectors u and v,

$$(I - uv^{\mathrm{T}})^{-1} = (I + u(I - v^{\mathrm{T}}u)^{-1}v^{\mathrm{T}}). \tag{2.2.39}$$

Thus Eqn. (2.2.35) can be solved to give

$$\begin{aligned} \sigma_1 &= (I + \phi_1(I - \alpha_{r2}v_l^{\mathrm{T}}\phi_1)^{-1}\alpha_{r2}v_l^{\mathrm{T}})(\eta_1 - \phi_1\delta_{r2}) \\ &= \eta_1 - \phi_1\delta_{r2} + \phi_1(I - \alpha_{r2}\alpha_{l1})^{-1}\alpha_{r2}(\delta_{l1} - \alpha_{l1}\delta_{r2}). \end{aligned} \tag{2.2.40}$$

Thus if

$$\begin{aligned} \Delta_1 &= I - \alpha_{l1}\alpha_{r2} \\ \Delta_2 &= I - \alpha_{r2}\alpha_{l1}, \end{aligned} \tag{2.2.41}$$

then

$$\sigma_1 = \eta_1 - \phi_1\delta_{r2} + \phi_1\Delta_2^{-1}\alpha_{r2}\delta_{l1} - \phi_l\Delta_2^{-1}\alpha_{r2}\alpha_{l1}\delta_{r2} \tag{2.2.42}$$

$$\sigma_1 = \eta_1 - \phi_1(\Delta_2^{-1}\alpha_{r2}\alpha_{l1} + \Delta_2^{-1}\Delta_2)\delta_{r2} + \phi_1\Delta_2^{-1}\alpha_{r2}\delta_{l1} \tag{2.2.43}$$

$$\sigma_1 = \eta_1 + \phi_1\Delta_2^{-1}(\alpha_{r2}\delta_{l1} - \delta_{r2}). \tag{2.2.44}$$

Similarly, we have

$$\sigma_2 = \eta_2 + \phi_2 \Delta_1^{-1}(\alpha_{l1}\delta_{r2} - \delta_{l1}). \tag{2.2.45}$$

Eqn. (2.2.44) and Eqn. (2.2.45) constitute an explicit formula for the restricted densities σ_1, σ_2 in terms of values from the sub-intervals. Hence σ can be built by solving linear systems on smaller sub-intervals and processing the local information. As seen in Eqn. (2.2.32) and Eqn. (2.2.33), both η_I and ϕ_I are solutions to problems of the form (2.2.25,2.2.28). Hence, by further subdividing $[a, b]$ and $[b, c]$, η_I and ϕ_I can be calculated in the same manner as σ. A fully recursive scheme requires the determination of $\alpha_l, \alpha_r, \delta_l$ and δ_r from values on the sub-intervals. In order to accomplish this, we perform the same manipulations on the matrix equations

$$\begin{aligned} P\chi &= \psi \\ P_{11}\chi_1 + P_{21}\chi_2 &= \psi_1 \\ P_{12}\chi_1 + P_{22}\chi_2 &= \psi_2 \end{aligned} \quad . \tag{2.2.46}$$

Just as we obtained Eqn. (2.2.38), we get

$$\chi_1 = (I + \phi_1 \Delta_2^{-1} \alpha_{r2} v_l^{\mathrm{T}})(\phi_1 - \phi_1 \alpha_{r2}) \tag{2.2.47}$$

$$\chi_1 = \phi_1 - \phi_1\alpha_{r2} + \phi_1\Delta_2^{-1}\alpha_{r2}\alpha_{l1} - \phi_1\Delta_2^{-1}\alpha_{r2}\alpha_{l1}\alpha_{r2} \tag{2.2.48}$$

$$\chi_1 = \phi_1(I - \alpha_{r2}) + \phi_1\Delta_2^{-1}\alpha_{r2}\alpha_{l1}(I - \alpha_{r2}) \tag{2.2.49}$$

$$\chi_1 = \phi_1(\Delta_2^{-1}\Delta_2 + \Delta_2^{-1}\alpha_{r2}\alpha_{l1})(I - \alpha_{r2}) \tag{2.2.50}$$

$$\chi_1 = \phi_1\Delta_2^{-1}(I - \alpha_{r2}). \tag{2.2.51}$$

Similarly, we obtain

$$\chi_2 = \phi_2\Delta_1^{-1}(I - \alpha_{l1}). \tag{2.2.52}$$

Eqn. (2.2.44) integrates to

$$\sigma_1 = \eta_1 + \phi_1\Delta_2^{-1}(\alpha_{r2}\delta_{l1} - \delta_{r2}) \tag{2.2.53}$$

$$v_l^{\mathrm{T}}\sigma_1 = \delta_{l1} + \alpha_{l1}\Delta_2^{-1}(\alpha_{r2}\delta_{l1} - \delta_{r2}). \tag{2.2.54}$$

Similarly, we have

$$v_r^{\mathrm{T}}\sigma_1 = \delta_{r1} + \alpha_{r1}\Delta_2^{-1}(\alpha_{r2}\delta_{l1} - \delta_{r2}) \tag{2.2.55}$$

$$v_l^{\mathrm{T}}\sigma_2 = \delta_{l2} + \alpha_{l2}\Delta_1^{-1}(\alpha_{l1}\delta_{r2} - \delta_{l1}) \tag{2.2.56}$$

$$v_r^{\mathrm{T}}\sigma_2 = \delta_{r2} + \alpha_{r2}\Delta_1^{-1}(\alpha_{l1}\delta_{r2} - \delta_{l1}). \tag{2.2.57}$$

We compute

$$\begin{aligned} \delta_l &= v_l^{\mathrm{T}}(\sigma_1 + \sigma_2) \\ &= \delta_{l1} + \alpha_{l1}\Delta_2^{-1}(\alpha_{r2}\delta_{l1} - \delta_{r2}) + \\ &\quad \delta_{l2} + \alpha_{l2}\Delta_1^{-1}(\alpha_{l1}\delta_{r2} - \delta_{l1}) \end{aligned} \tag{2.2.58}$$

$$\begin{aligned} \delta_r &= v_r^{\mathrm{T}}(\sigma_1 + \sigma_2) \\ &= \delta_{r1} + \alpha_{r1}\Delta_2^{-1}(\alpha_{r2}\delta_{l1} - \delta_{r2}) + \\ &\quad \delta_{r2} + \alpha_{r2}\Delta_1^{-1}(\alpha_{l1}\delta_{r2} - \delta_{l1}). \end{aligned} \tag{2.2.59}$$

Eqn. (2.2.58) and Eqn. (2.2.59) allow us to calculate the values of the parent interval $[a, c]$ from the properties of the sub-intervals $[a, b], [b, c]$. We need a way to calculate α_l, α_r from the values of the sub-intervals. As before, we integrate Eqn. (2.2.51) to yield

$$\chi_1 = \phi_1\Delta_2^{-1}(I - \alpha_{r2}) \tag{2.2.60}$$

$$v_l^{\mathrm{T}}\chi_1 = \alpha_{l1}\Delta_2^{-1}(I - \alpha_{r2}). \tag{2.2.61}$$

similarly, we have

$$v_r^{\mathrm{T}}\chi_1 = \alpha_{r1}\Delta_2^{-1}(I - \alpha_{r2}) \tag{2.2.62}$$

$$v_l^{\mathrm{T}}\chi_2 = \alpha_{l2}\Delta_1^{-1}(I - \alpha_{l1}) \tag{2.2.63}$$

$$v_r^{\mathrm{T}}\chi_2 = \alpha_{r2}\Delta_1^{-1}(I - \alpha_{l1}). \tag{2.2.64}$$

Thus

$$\alpha_l = v_l^{\mathrm{T}}(\chi_1 + \chi_2) = \alpha_{l1}\Delta_2^{-1}(I - \alpha_{r2}) + \alpha_{l2}\Delta_1^{-1}(I - \alpha_{l1}) \tag{2.2.65}$$

$$\alpha_r = v_r^{\mathrm{T}}(\chi_1 + \chi_2) = \alpha_{r1}\Delta_2^{-1}(I - \alpha_{r2}) + \alpha_{r2}\Delta_1^{-1}(I - \alpha_{l1}). \tag{2.2.66}$$

Eqn. (2.2.65) and Eqn. (2.2.66) express α_l, α_r on the full interval using values from the sub-intervals, as required by a fully recursive scheme.

One more analytical shortcut is possible: From Eqn. (2.2.44) and Eqn. (2.2.45), the global density restricted to a subinterval I is given by some linear combination of the local density η_I and ϕ_I

$$\sigma_I = \eta_I + \phi_I \cdot \lambda_I \tag{2.2.67}$$

for some vector λ_I. So the global density σ, restricted to I, is a simple linear combination of the local density on that interval (η) and matrix ϕ. A recursive determination of the coefficients λ_I saves a lot of work. Given a function L defined on the interval $[a, c]$ which is a linear combination

$$L = \sigma + \chi\lambda, \tag{2.2.68}$$

Eqn. (2.2.44),Eqn. (2.2.45),Eqn. (2.2.51) and Eqn. (2.2.52) imply that the restriction of L to the interval $[a, b]$ is a linear combination

$$L_1 = \eta_1 + \phi_1\lambda_1 \tag{2.2.69}$$

for some vector λ_1 which depends on the original λ as well as the various α matrices and δ vectors. On the other hand, Eqn. (2.2.23),Eqn. (2.2.24) and Eqn. (2.2.46) give

$$L_1 = \sigma_1 + \chi_1\lambda \tag{2.2.70}$$

Eqn. (2.2.44) and Eqn. (2.2.51) yield

$$\begin{aligned} L_1 &= \eta_1 + \phi_1\Delta_2^{-1}(\alpha_{r2}\delta_{l1} - \delta_{r2}) + \phi_1\Delta_2^{-1}(I - \alpha_{r2})\lambda \\ &= \eta_1 + \phi_1\Delta_2^{-1}(\alpha_{r2}\delta_{l1} - \delta_{r2} + (I - \alpha_{r2})\lambda). \end{aligned} \tag{2.2.71}$$

Thus λ_1 and λ_2 are given by

$$\lambda_1 = \Delta_2^{-1}(\alpha_{r2}\delta_{l1} - \delta_{r2} + (I - \alpha_{r2})\lambda) \tag{2.2.72}$$

$$\lambda_2 = \Delta_1^{-1}(\alpha_{l1}\delta_{r2} - \delta_{l1} + (I - \alpha_{l1})\lambda). \tag{2.2.73}$$

These analytical tools allow us to solve a boundary problem recursively: Split $[a, c]$ into 2^N adjacent intervals of equal length $[a_0, a_1], [a_1, a_2], \ldots, [a_{2^N-1}, a_{2^N}]$ with equispaced endpoints such that $a_0 < a_1 < \ldots < a_{2^N}$. Organize these intervals into a binary tree structure [2, 36] with levels $n = 0, 1, \ldots, N$. Level n of the tree consists of the 2^n intervals $I_i^n = [a_{i\cdot 2^{N-n}}, a_{(i+1)\cdot 2^{N-n}}]$ each of length $(c-a)2^{N-n}$ (i ranges from 0 to $2^n - 1$). On level n, the interval I_i^n (for fixed i) has children on level $n+1$ of the tree. These children are the intervals $I_{2i}^{n+1}, I_{2i+1}^{n+1}$ obtained by subdividing the original interval in half. Similarly, on level n, the interval I_i^n has a parent $I_{[i/2]}^{n-1}$ on level $n-1$ of the tree.

The boundary value problem is solved on this binary tree as follows: At the finest level N, there are 2^N intervals. For each of the intervals $I_i^N = [a_i, a_{i+1}]$ on this level, build the localized operator

$$P_i^N(\sigma)(x) = \sigma(x) + (p(x) - p_0(x)) \int_{a_i}^{a_{i+1}} G_0(x,t)\sigma(t) \tag{2.2.74}$$

and determine the local values f_i^N and ψ_i^N of f and ψ within I_i^N. Solve the local equations

$$\begin{aligned} \eta_i^N &= [P_i^N]^{-1} f_i^N \\ \phi_i^N &= [P_i^N]^{-1} \psi_i^N \end{aligned} \tag{2.2.75}$$

to find η_i^N and ϕ_i^N on I_i^N. Similarly, find $v_{l,i}^N, v_{r,i}^N, \alpha_{l,i}^N, \alpha_{r,i}^N, \delta_{l,i}^N$ and $\delta_{r,i}^N$. Once all of these local values have been computed for each subinterval on the finest level, move one level up the tree. There are 2^{N-1} intervals at this level. Each interval I_i^{N-1} on level $N-1$ has children I_{2i}^N, I_{2i+1}^N on level N of the tree with local α and δ already computed. Using the values of $\alpha_{l,2i}^N$, $\alpha_{l,2i+1}^N$, $\alpha_{r,2i}^N$, $\alpha_{r,2i+1}^N$ and $\delta_{l,2i}^N$, $\delta_{l,2i+1}^N$, $\delta_{r,2i}^N$, $\delta_{r,2i+1}^N$ computed for the children on level N, compute the values of $\alpha_{l,i}^{N-1}$, $\alpha_{r,i}^{N-1}$, $\delta_{l,i}^{N-1}$, $\delta_{r,i}^{N-1}$ for the parent interval I_i^{N-1} by Eqns. (2.2.58 - 2.2.66). Repeat this process up to level 0 of the tree, computing $\alpha_{l,i}^n$, $\alpha_{r,i}^n$, $\delta_{l,i}^n$, $\delta_{r,i}^n$ for every interval i on every level n of the tree.

Now sweep downward from level 0, where there is only 1 subinterval, namely $[a, c]$ itself. The localized operator P_0^0 for $[a, c]$ is actually the global integral operator we want to invert. Thus, given η_0^0 and ϕ_0^0 for this interval, the solution at this level is $\sigma = \eta_0^0 + \phi_0^0 \cdot 0$. So set $\lambda_0^0 = 0$ for this interval. Eqns. (2.2.72, 2.2.73) provide λ_0^1 and λ_1^1 for the 2 child intervals at level 1 of the tree, then λ_0^2, λ_1^2, λ_2^2, λ_3^2 for the 4 child intervals at level 2 of the tree, and so forth until λ_i^N is determined on each subinterval at the finest level. The solution σ on each subinterval on the finest level is given by

$$\sigma = \eta_i^N + \phi_i^N \lambda_i^N. \tag{2.2.76}$$

This corresponds to the following algorithm:

Algorithm 2.2.1 *Binary Tree Solver for Two-Point Boundary Value Problem*

1. *for each interval I on the finest level N*

   Find P on interval I,

   then, by evaluating $\hat{p}$ and f and inverting P directly,

   calculate the values of η and ϕ for I,

   as well as the values of α and δ.

   end for

2. *for $l = N-1, N-2, \ldots, 0$*

   for each interval I on level l

   Find the children L and R of I on level $l+1$

   then, using the values of α and δ for L and R,

   calculate values of α and δ for interval I.

   end for

   end for

3. *Set $\lambda = 0$ for the single interval on the coarsest level*

4. *for $l = 0, 1, 2, \ldots, N$*

for each interval I on level l

Find children L and R of I on level $l+1$

then, using the values of α and δ for L and R,

as well as the value of λ for I,

calculate values of λ for L and R.

end for

end for

5. *for each interval I on the finest level N*

Using the values of η, ϕ and λ for I,

calculate the values of σ for I.

end for

6. *for each interval I on the finest level N*

Using the values of σ for I,

calculate the values of Φ for I.

end for

This procedure can easily be made solution-adaptive [9]. Let a single interval $[a, c]$ be the top (coarsest) level of the tree. Use a coarse grid (q Chebyshev nodes) to estimate P, η, ϕ, α and δ on this interval. Compute a rough estimate $\hat{\sigma}$ on the coarse grid by inverting $P\hat{\sigma} = f$. Now split the top interval, creating two child intervals of half the size to form the second level of the tree. Find P, η, ϕ, α and δ for each of these child intervals. Then recompute α and δ for the parent interval at the top of the tree using Eqns. (2.2.58 – 2.2.66). Use Eqns. (2.2.72, 2.2.73, 2.2.76) to compute λ and $\hat{\sigma}$ for each of the child intervals. If $\hat{\sigma}$ is not sufficiently resolved within a particular interval, split that interval and create two smaller subintervals. After this splitting process, there may be some old subintervals that have not been divided, and other new subintervals that have just been created. Find P, η, ϕ, α and δ for each of these newly created subintervals. Then work up the

tree recomputing α and δ for all the ancestors of these newly created subintervals. Then, starting at the top of the tree and working our way down again, recompute λ for every subinterval. After finding λ for every subinterval, construct the density $\hat{\sigma}$ for the deepest subintervals at the bottom of the tree. Once again, check whether the density $\hat{\sigma}$ is well resolved in each childless subinterval and split the subintervals accordingly. Then calculate P, η, ϕ, α and δ for each of these newly created subintervals, and repeat the process. Stop when the density $\hat{\sigma}$ has been sufficiently resolved in each of the childless subintervals. Finally, calculate Φ for each of the deepest subintervals at the bottom of the tree.

This corresponds to the following

Algorithm 2.2.2 *Adaptive Tree Solver for Two-Point Boundary Value Problems*

1. *Choose an integration order q and tolerance ϵ.*
2. *Choose a preliminary binary tree structure that sufficiently resolves p and f.*
3. *Run Setup($[a, c]$).*
4. *Run Update($[a, c]$).*
5. *Run FindL($[a, c]$).*
6. *do*

   nsplits = Split($[a, c]$,q,ϵ)

   Setup($[a, c]$)

   Update($[a, c]$)

   FindL($[a, c]$)

   while nsplits > 0
7. *Calculate Φ for each of the intervals on the finest level of the tree.*

where we have defined

Algorithm 2.2.3 *Setup(interval I)*

1. *If I has children L, R, then Setup(L) and Setup(R).*

2. *Otherwise, if I has no children, but has values of $P, \eta, \phi, \alpha, \delta$ already computed and stored, then just exit.*

3. *Otherwise, compute and store values of $P, \eta, \phi, \alpha, \delta$ for I.*

Algorithm 2.2.4 *Update(interval I)*

1. *If I has no children, then just exit.*

2. *Otherwise, if I has children L, R, then Update(L) and Update(R).*

3. *Now use the values of α, δ for L, R and Eqns. (2.2.58 - 2.2.66) to compute and store α, δ for I.*

Algorithm 2.2.5 *FindL(interval I)*

1. *If I has no children, then compute and store $\hat{\sigma} = \eta + \phi \cdot \lambda$ for I.*

2. *If I has children L, R, then use the values of λ for I and the values of α, δ for L, R and Eqns. (2.2.72, 2.2.73) to compute and store λ for L, R.*

3. *FindL(L) and FindL(R).*

Algorithm 2.2.6 *output = Split(interval I,integration order q,tolerance ϵ)*

1. *If I has children L, R, then return Split(L) + Split(R).*

2. *Otherwise, if I has no children and SigmaResolved(I,q,ϵ) = TRUE then return 0.*

3. *Otherwise, if I has no children and SigmaResolved(I,q,ϵ) = FALSE then create two new child intervals of I by splitting I in half, and return 1.*

There are several ways to check whether $\hat{\sigma}$ is resolved in a given interval. One way is to compare the $\hat{\sigma}$ in the child interval to the $\hat{\sigma}$ of its immediate parent. If their values differ significantly, then more resolution is required. Another way is to expand $\hat{\sigma}$ in terms of the Chebyshev polynomials used to perform the quadrature, and examine the 2 highest-order expansion coefficients. If these high-order coefficients of $\hat{\sigma}$ are small relative to the rest of the coefficients, we assume that $\hat{\sigma}$ is well approximated by the current basis function expansion. In other words, we hope that the neglected terms in the Chebyshev expansion of $\hat{\sigma}$ are negligible. However, if the high-order coefficients of $\hat{\sigma}$ are relatively large, we suspect that the next few terms in the basis function expansion are non-negligible, and we split the interval. We examine the highest 2 expansion coefficients in case $\hat{\sigma}$ is predominantly even or odd. Such a technique is employed in [25]. This corresponds to the following

Algorithm 2.2.7 *output = SigmaResolved(interval I,integration order q,tolerance ϵ)*

1. *Use q Chebyshev nodes on the interval I to construct a Chebyshev expansion $\{c_1, \ldots, c_q\}$ of $\hat{\sigma}$ on I.*
2. *Let $C_q = \left(\sum_{i=0}^{q} c_i^2\right)^{1/2}$.*
3. *if $C_q \leq \epsilon$ then set $C_q = 1$.*
4. *Let $C_2 = \left(c_{q-1}^2 + c_q^2\right)^{1/2}$.*
5. *If $C_2/C_q \leq \epsilon$ then return TRUE.*
6. *Otherwise return FALSE.*

Chapter 3

Spectral Deferred Correction Methods

Deferred correction is a general idea that can be applied to ordinary differential equations (ODEs), PDEs and DAEs. The idea is to take a low order scheme and promote it to a high-order scheme by calculating a residual and solving for the error. The general steps are

1. Use a numerical scheme to construct an approximate solution $u(t)$ to the problem.
2. Calculate a residual $R'(t)$ in terms of $u(t)$ and the data of the problem.
3. Formulate an equation for the error $e = \text{exact} - \text{computed}$ in terms of $R(t)$ and u.
4. Solve the error equation for e.
5. Update $u \leftarrow u + e$.
6. Quit if ϵ or R is small, otherwise repeat.

3.1 Spectral deferred correction for ODEs

A classical deferred correction procedure was developed in [34, 33, 32, 31, 3, 11]. The spectral deferred correction procedure was developed in [7]. We review the procedure of [7], because it is more accurate and easier to understand.

Consider the initial value problem

$$\begin{aligned} y'(t) &= f(t, y(t)) \\ y(0) &= y_0. \end{aligned} \tag{3.1.1}$$

A multistep or Runge-Kutta method can be used to compute the solution on a time interval $[0, T]$. This yields a sequence of time values t_j and an approximate solution u_j at these times with $u_0 = y_0$. We usually choose the time values t_j in the interval $[0, T]$ to correspond to Chebyshev quadrature nodes. This allows us to easily build a continuous interpolant u from the discrete values u_j on $[0, T]$. Then we construct the residual

$$r(t) = f(t, u(t)) - u'(t), \tag{3.1.2}$$

which measures how far u is from satisfying Eqn. (3.1.1). Then the exact solution y to Eqn. (3.1.1) yields the initial value problem for the error

$$\begin{aligned} e(t) &= y(t) - u(t); \\ e'(t) &= y'(t) - u'(t) \\ &= f(t, y(t)) - f(t, u(t)) + r(t) \\ &= f(t, u(t) + e(t)) - f(t, u(t)) + r(t) \\ &= F(t, u(t), e(t)) + r(t) \\ e(0) &= 0, \end{aligned} \tag{3.1.3}$$

where $F(t, u(t), e(t)) = f(t, u(t) + e(t)) - f(t, u(t))$. This problem is of the same form as Eqn. (3.1.1) and we solve it (usually by the same numerical method), to obtain an approximation $\hat{e}$ to e. The updated solution

$$u^{\text{new}} = u + \hat{e} \tag{3.1.4}$$

should be more accurate than u.

Note: In practice we never directly evaluate $r(t)$. Evaluating $r(t)$ may involve numerically differentiating a discrete numerical solution, which is numerically unstable [5]. Instead, we form linear combinations of r which can be approximated by evaluating $R(t)$

$$\begin{aligned} R(t) &= \int_0^t r(s)ds \\ &= y_0 - u(t) + \int_0^t f(s, u(s))ds. \end{aligned} \tag{3.1.5}$$

Computing $R(t)$ only involves the numerical integration of a discrete numerical solution. This is numerically stable if we know the discrete solution at the nodes of an accurate quadrature scheme.

For example, assume we solve Eqn. (3.1.1) with a k-step linear multistep method [1, 16] such as implicit Euler ($k = 1$). The time values t_j in the interval $[0, T]$ may not be equispaced, so we let $h_j = t_{j+1} - t_j$. The method is given by

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = \sum_{j=0}^{k} h_j \beta_j f(t_{n+j}, y_{n+j}), \tag{3.1.6}$$

where the coefficients $\{\beta_j\}$ are usually linked to a quadrature scheme

$$\sum_{j=0}^{k} h_j \beta_j y'_{n+j} \approx y_{n+b} - y_{n+a} \tag{3.1.7}$$

for some a and b. Usually $b = 1$ and $a = 0$ as in BDF or Adams methods. This method produces a discrete numerical solution u_j. A continuous approximate $u(t)$ is then constructed by interpolation. Solving Eqn. (3.1.3) by the multistep scheme (3.1.6) gives

$$\sum_{j=0}^{k} \alpha_j e_{n+j} = \sum_{j=0}^{k} h_j \beta_j F(t_{n+j}, u_{n+j}, e_{n+j}) + h_j \beta_j r(t_{n+j}). \tag{3.1.8}$$

Then Eqn. (3.1.7) yields

$$\sum_{j=0}^{k} \alpha_j e_{n+j} = R(t_{n+b}) - R(t_{n+a}) + \sum_{j=0}^{k} h_j \beta_j F(t_{n+j}, u_{n+j}, e_{n+j}). \tag{3.1.9}$$

Plugging in Eqn. (3.1.5) we get

$$\sum_{j=0}^{k} \alpha_j e_{n+j} = \int_{t_{n+a}}^{t_{n+b}} f(s, u(s))ds + u_{n+a} - u_{n+b} + \sum_{j=0}^{k} h_j \beta_j F(t_{n+j}, u_{n+j}, e_{n+j}). \tag{3.1.10}$$

Note that we never explicitly construct the continuous $u(t)$. Rather, we use the discrete solution u_j with an appropriate quadrature scheme to evaluate the required interpolants and definite integrals.

Alternatively, we can solve Eqn. (3.1.1) and Eqn. (3.1.3) with an s-stage FSAL Runge-Kutta method [22, 12, 23]. These methods are useful in spectral deferred correction of DAEs. Each step of such a method computes s stages

$$Y_i = y_n + h \sum_{j=1}^{s} a_{ij} f(t + c_j h, Y_j) \qquad i = 1, \ldots, s. \tag{3.1.11}$$

The numerical solution at the next step is given by the last stage of the current step

$$y_{n+1} = Y_s, \tag{3.1.12}$$

so the method is said to be 'first same as last', or FSAL, and saves one function evaluation. The coefficients a_{ij} and c_i are a set of quadrature rules

$$y(t + c_i h) \approx y(t) + h \sum_{j=1}^{s} a_{ij} y'(t + c_j h). \tag{3.1.13}$$

Applying the method (3.1.11,3.1.12) to Eqn. (3.1.3) gives

$$E_i = e_n + h \sum_{j=1}^{s} a_{ij} F(t + c_j h, u(t + c_j h), E_j) - a_{ij} r(t + c_j h) \tag{3.1.14}$$

$$e_{n+1} = E_s. \tag{3.1.15}$$

Plugging in Eqn. (3.1.13) gives

$$E_i = e_n + u(t + c_i h) - u(t) + \int_t^{t+c_i h} f(s, u(s)) ds + h \sum_{j=1}^{s} a_{ij} F(t + c_j h, u(t + c_j h), E_j) \tag{3.1.16}$$

$$e_{n+1} = E_s. \tag{3.1.17}$$

When applying Eqn. (3.1.16), interpolation and quadrature schemes based on the discrete values of $u_j, f(t_j, u_j)$ calculate the required values of $u_{t+c_i h}$ as well as the required integrals $\int_t^{t+c_i h} f ds$.

This process is easily iterated. Once we update our solution we calculate a new residual and a new error and repeat the process. The accuracy of the solution increases with each correction. In practice we use the magnitude of the computed residual R and error $\hat{e}$ as an estimation of the accuracy of the current solution. If R or $\hat{e}$ is large we perform another correction step. If R and $\hat{e}$ are small we assume our current solution is accurate and move on.

This assumption will not be valid unless the original ODE is well conditioned when written in integral form. In other words, we assume that the operator

$$\mathcal{R}[u](t) = y_0 - u(t) + \int_0^t f(s, u(s)) ds \tag{3.1.18}$$

is well conditioned with respect to the input function u in a neighborhood of the exact solution y. We hope that

$$|\mathcal{R}[y+\delta] - \mathcal{R}[y]| = |\mathcal{R}[y+\delta]| = O(|\delta|). \tag{3.1.19}$$

If this is the case, then y is a local minimum of $|\mathcal{R}[u]|$. Therefore if our approximate solution u is sufficiently close to y then we can safely associate a small residual with an accurate numerical solution. This motivates the following definitions

$$\begin{array}{lcl} \mu & = & \text{iteration index} \\ u^{[\mu]} & = & \text{solution after } \mu \text{ steps of deferred correction} \\ R^{[\mu]} & = & \text{computed residual of } u^{[\mu]} \\ e^{[\mu]} & = & \text{approximate error of } u^{[\mu]} \end{array} \tag{3.1.20}$$

And the following

Algorithm 3.1.1 *Iterated deferred correction for ODEs*

1. *Choose an order q of Chebyshev integration (Section 2.1), a maximum number of allowed corrections μ_{max}, and a tolerance ϵ.*

2. *Choose a Runge-Kutta method or a multistep method that is compatible with the Chebyshev nodes.*

3. *Set $T_0 \leftarrow 0$ and choose an initial time-step ΔT.*

4. *Let $\{t_0, \ldots, t_q\}$ be Chebyshev nodes on $[T_0, T_0 + \Delta T]$ corresponding to the quadrature scheme of step (1).*

5. *Set $\mu \leftarrow 0$.*

6. *Calculate the numerical solution $u_j^{[\mu]}$ at time points t_j, with the scheme of step (2).*

7. *Calculate the residual $R_j^{[\mu]}$ at time points t_j using the quadrature scheme chosen in step (1).*

8. *If $|R^{[\mu]}| < \epsilon$ the numerical solution is sufficiently well resolved. Go to step (14).*

9. *Otherwise a correction step is necessary. Calculate the error* $e_j^{[\mu]}$ *with the scheme of step (2), using Eqns. (2.1.2 - 2.1.4) to compute the values of* $u_j^{[\mu]}, R_j^{[\mu]}$ *if necessary.*

10. *Form the improved solution* $u_j^{[\mu+1]} = u_j^{[\mu]} + e_j^{[\mu]}$.

11. *If* $\mu < \mu_{max}$ *then set* $\mu \leftarrow \mu + 1$ *and go back to step (7).*

12. *If* $\|R^{[\mu max]}\| > \epsilon$ *then the numerical solution is not sufficiently accurate. Restart from step (4) with* $\Delta T \leftarrow \Delta T/2$.

13. *Otherwise, if* $|R^{[\mu max]}| \leq \epsilon$ *the numerical solution* $u_j^{[\mu max]}$ *is accurate. Go to step (14).*

14. *Record the numerical solution* $u_j^{[\mu]}$. *Set* $T_0 \leftarrow t_q$. *If* ΔT *was not halved recently, set* $\Delta T \leftarrow 2\Delta T$. *Go back to step (4).*

Iterated deferred correction works for ODEs in the sense that every correction step increases the order of the solution. In fact, if we use a single method of order p to calculate a first approximation to the solution and perform $k-1$ correction steps, then we can think of the resulting solution as the output of an order kp method. More precisely, we have the following theorem [17].

Theorem 3.1.1 *Assume we are given an initial value problem 3.1.1 and a numerical method 3.1.6 (or 3.1.11) with global convergence of order p. Assume we solve the initial value problem with the numerical method and obtain an approximate solution* u_j. *If we then use a quadrature scheme of order q to apply deferred correction via Eqn. (3.1.10) (or Eqn. (3.1.16) respectively), we obtain a new solution* $u_j + e_j$ *of order* $\min(2p, q)$. *This process can be iterated, and each iteration increases the order by p, up to a maximum of q.*

3.2 Spectral deferred correction for PDEs

Semi-implicit deferred correction schemes for the time integration of PDEs were developed in [4] and [24]. These schemes divide the PDE into stiff and nonstiff parts, and solve the stiff parts (ie, diffusion) implicitly and the nonstiff parts (ie, advection) explicitly. We will treat the entire parabolic

PDE implicitly. The correction process for PDEs is almost completely analogous to the ODE case. Assume we have some well posed initial-boundary value problem

$$\partial_t y(x,t) = F(x,t,y(x,t),\partial_x y(x,t),\partial_{xx} y(x,t)) \tag{3.2.1}$$

$$y(x,0) = y_0(x) \tag{3.2.2}$$

$$A(t)\begin{bmatrix} y(a,t) \\ \partial_x y(a,t) \end{bmatrix} + C(t)\begin{bmatrix} y(c,t) \\ \partial_x y(c,t) \end{bmatrix} = \gamma(t). \tag{3.2.3}$$

We apply the implicit Euler method of Chapter 6 and obtain a numerical solution $u_n(x)$ at each time t_n by solving

$$\begin{gathered} u_{n+1}(x) = u_n(x) + kF(x,t_{n+1},u_{n+1}(x),\partial_x u_{n+1}(x),\partial_{xx} u_{n+1}(x)) \\ A(t_{n+1})\begin{bmatrix} u_{n+1}(a) \\ \partial_x u_{n+1}(a) \end{bmatrix} + C(t_{n+1})\begin{bmatrix} u_{n+1}(c) \\ \partial_x u_{n+1}(c) \end{bmatrix} = \gamma(t_n) \\ u_0(x) = y_0(x) \end{gathered} \tag{3.2.4}$$

for $n = 0,1,\ldots,N$. Once we know $u_n(x)$ we may form a continuous $u(x,t)$ as well as a residual $r(x,t)$ such that

$$\partial_t u(x,t) = F(x,t,u(x,t),\partial_x u(x,t),\partial_{xx} u(x,t)) - r(x,t). \tag{3.2.5}$$

The error $e(x,t) = y - u$ then satisfies the PDE

$$\begin{aligned} e &= y - u \\ \partial_t e &= F(x,t,y,\partial_x y,\partial_{xx} y) - F(x,t,u,\partial_x u,\partial_{xx} u) + r(x,t) \\ &= F(x,t,u+e,\partial_x u + \partial_x e,\partial_{xx} u + \partial_{xx} e) - F(x,t,u,\partial_x u,\partial_{xx} u) + r(x,t) \end{aligned} \tag{3.2.6}$$

subject to homogeneous initial values and boundary conditions

$$A(t)\begin{bmatrix} e(a,t) \\ \partial_x e(a,t) \end{bmatrix} + C(t)\begin{bmatrix} e(c,t) \\ \partial_x e(c,t) \end{bmatrix} = 0. \tag{3.2.7}$$

This problem is just like Eqn. (3.2.1). We linearize and apply Euler's method to obtain:

$$\begin{aligned} e_{n+1} = {} & e_n(x) + \\ & k\partial_y F(x,t_{n+1},u_{n+1},\partial_x u_{n+1},\partial_{xx} u_{n+1}) e_{n+1} + \\ & k\partial_{y_x} F(x,t_{n+1},u_{n+1},\partial_x u_{n+1},\partial_{xx} u_{n+1})\partial_x e_{n+1} + \\ & k\partial_{y_{xx}} F(x,t_{n+1},u_{n+1},\partial_x u_{n+1},\partial_{xx} u_{n+1})\partial_{xx} e_{n+1} + \\ & (u_n - u_{n+1}) + \\ & \textstyle\int_{t_n}^{t_{n+1}} F(x,s,u,\partial_x u,\partial_{xx} u)ds, \end{aligned} \tag{3.2.8}$$

along with homogeneous initial values and boundary conditions that agree with 3.2.7:

$$\begin{aligned} e_0 &= 0 \\ A(t_{n+1}) \begin{bmatrix} e_{n+1}(a) \\ \partial_x e_{n+1}(a) \end{bmatrix} &+ C(t_{n+1}) \begin{bmatrix} e_{n+1}(c) \\ \partial_x e_{n+1}(c) \end{bmatrix} = 0. \end{aligned} \tag{3.2.9}$$

Note that we do not directly evaluate $r(x,t)$. Instead we construct and evalute

$$\begin{aligned} R(x,t) &= \int_0^t r(x,s)ds \\ &= y_0(x) - u(x,t) + \int_0^t F(x,s,u(x,s),\partial_x u(x,s),\partial_{xx}u(x,s))ds, \end{aligned} \tag{3.2.10}$$

which is analagous to Eqn. (3.1.5). We define vectors

$$\begin{aligned} e &= e_{n+1}(x) \\ e' &= \partial_x e_{n+1} \\ \Delta R &= R(x,t_{n+1}) - R(x,t_n) \\ &= (u_n(x) - u_{n+1}(x)) \\ &\quad + \int_{t_n}^{t_{n+1}} F(x,s,u,\partial_x u,\partial_{xx}u)ds, \end{aligned} \tag{3.2.11}$$

and matrices

$$\begin{aligned} H &= \partial_y F(x,t_{n+1},u_{n+1},\partial_x u_{n+1},\partial_{xx}u_{n+1}) \\ J &= \partial_{y_x} F(x,t_{n+1},u_{n+1},\partial_x u_{n+1},\partial_{xx}u_{n+1}) \\ K &= \partial_{y_{xx}} F(x,t_{n+1},u_{n+1},\partial_x u_{n+1},\partial_{xx}u_{n+1}) \\ M &= K^{-1}, \end{aligned} \tag{3.2.12}$$

and rewrite Eqn. (3.2.8) as a first order system:

$$\partial_x \begin{bmatrix} e \\ e' \end{bmatrix} + \begin{bmatrix} 0 & -I \\ M(H - I/k) & MJ \end{bmatrix} \cdot \begin{bmatrix} e \\ e' \end{bmatrix} = \begin{bmatrix} 0 \\ -(\Delta R + e_n)/k \end{bmatrix}. \tag{3.2.13}$$

Together Eqn. (3.2.13) and Eqn. (3.2.9) form a linear two-point boundary value problem which is solved adaptively using the methods of Section 2.2. As before, a quadrature scheme is used to compute ΔR. Once we have e_j we form $u^{\text{new}} = u_j + e_j$. We repeat the process as necessary, applying Algorithm 3.1.1. The solution $u_j^{[0]}$ at each time t_j is computed on a different spatial grid. Therefore we cannot immediately precompute the residual in step (7) of Algorithm 3.1.1. A similar difficulty arises when we add on the corrections in step (10). This requires us to have some way of combining two different functions on two different spatial grids.

3.3 Spectral deferred correction for DAEs

Spectral deferred correction can also be applied to DAEs. The procedure is very similar to the procedure for ODEs and PDEs. We solve the original problem with a low order method, then calculate the residual, then formulate an equation for the error. In this case, the error equation will be a DAE (as opposed to an ODE or PDE).

3.3.1 Index-1 DAEs

Consider the constrained initial value problem:

$$\begin{aligned} y'(t) &= f(t, y(t), z(t)) \\ 0 &= g(t, y(t), z(t)) \\ \\ y(0) &= y_0 \\ z(0) &= z_0 \\ 0 &= g(0, y_0, z_0). \end{aligned} \tag{3.3.1}$$

We assume that g_z has bounded inverse. This is a simple index-1 DAE. The solution has 2 components $y(t), z(t)$. We solve this DAE with an s-stage FSAL runge-kutta method similar to Eqn. (3.1.11):

$$\begin{aligned} y_{n+1} &= Y_s \\ z_{n+1} &= Z_s \\ Y_i &= y_n + h\textstyle\sum_{j=1}^{s} a_{ij} f(t + c_i h, Y_i, Z_i) \\ 0 &= g(t + c_i h, Y_i, Z_i). \end{aligned} \tag{3.3.2}$$

For each time-step we solve the system of $2s$ equations using newton's method with an initial guess of $Y_j = y_n$ and $Z_j = z_n$ for each j. Note that we are imposing the constraint at each stage of the runge kutta method. Since the solution at the next step is actually one of the stages, the solution at the next step will also satisfy the constraint. The application of Eqn. (3.3.2) to Eqn. (3.3.1) yields a (2-component) numerical solution u_j, v_j.

Once we obtain u_j and v_j, we may form continuous numerical solutions $u(t)$ and $v(t)$ and construct the residual

$$r(t) = u'(t) - f(t, u(t), v(t)). \tag{3.3.3}$$

Once we compute the residual we focus on the error d, e

$$\begin{aligned} d(t) &= y(t) - u(t) \\ e(t) &= z(t) - v(t). \end{aligned} \tag{3.3.4}$$

The error equations are

$$\begin{aligned} d'(t) &= y'(t) - u'(t) \\ d'(t) &= f(t, y(t), z(t)) - f(t, u(t), v(t)) - r(t) \\ d'(t) &= f(t, u(t) + d(t), v(t) + e(t)) - f(t, u(t), v(t)) - r(t) \\ 0 &= g(t, u(t) + d(t), v(t) + e(t)), \end{aligned} \tag{3.3.5}$$

along with initial conditions

$$\begin{aligned} d(0) &= 0 \\ e(0) &= 0. \end{aligned} \tag{3.3.6}$$

Eqns. (3.3.5, 3.3.6) form a DAE similar to Eqn. (3.3.1). We linearize and solve it the same way:

$$\begin{aligned} d_{n+1} &= D_s \\ e_{n+1} &= E_s \\ D_i &= d_n \\ &\quad + h \textstyle\sum_{j=1}^{s} a_{ij} \partial_y f(t + c_j h, u(t + c_j h), v(t + c_j h)) \cdot D_j \\ &\quad + h \textstyle\sum_{j=1}^{s} a_{ij} \partial_z f(t + c_j h, u(t + c_j h), v(t + c_j h)) \cdot E_j \\ &\quad + \textstyle\int_t^{t+c_i h} f(s, u(s), v(s)) ds - u(t + c_i h) + u(t) \\ 0 &= g(t + c_i h, u(t + c_i h) + D_i, v(t + c_i h) + E_i). \end{aligned} \tag{3.3.7}$$

We never construct $r(t)$, instead we evaluate

$$\begin{aligned} R(t) &= \textstyle\int_0^t r(s) ds \\ &= y_0 - u(t) + \textstyle\int_0^t f(s, u(s), v(s)) ds, \end{aligned} \tag{3.3.8}$$

which is almost identical to Eqn. (3.1.5). A quadrature scheme is used to compute the values of R_j as well as $u(t + c_i h)$ and $\int_t^{t+c_i h} f ds$. We use newton's method to solve Eqn. (3.3.7), starting with an initial guess of $D_j = d_n$ and $E_j = e_n$ for each j.

This process works, and we prove the following

Theorem 3.3.1 *Assume we are given an index-1 DAE in the form of Eqn. (3.3.1) where $g(t, y, z)$ is uniquely solvable for z in terms of y. Also assume we are given a numerical method in the form of Eqn. (3.3.2) with global convergence of order p. We use the numerical method to solve the DAE,*

obtaining an approximate solution u_j, v_j. If we then use a quadrature scheme of order q and apply deferred correction via Eqn. (3.3.7), we obtain a new solution $u_j + d_j, v_j + e_j$ of order $\min(2p, q)$. *This process can be iterated, and each iteration increases the order by p to a maximum of q.*

PROOF. Since g is uniquely solvable for z in terms of y, we can write

$$z(t) = G(t, y(t)) \tag{3.3.9}$$

for some function $G(t, y)$. Eqn. (3.3.1) can be rewritten as

$$y'(t) = f(t, y(t), G(t, y(t))) = F(t, y(t)). \tag{3.3.10}$$

Since the constraint is imposed at every step of Eqn. (3.3.2) and Eqn. (3.3.7), $u_n = G(t, v_n)$. Therefore applying Eqn. (3.3.2) and Eqn. (3.3.7) to Eqn. (3.3.1) is equivalent to applying a standard s-stage FSAL deferred correction methods to Eqn. (3.3.10). Therefore by Theorem 3.1.1 we know that $(u_n + d_n)$ is a solution of order $\min(2p, q)$. If G is lipshitz then $(v_n + e_n)$ is also a solution of order $\min(2p, q)$. □

3.3.2 Index-2 DAEs

The formulation for index-2 DAEs is similar. Consider the DAE

$$\begin{aligned} y'(t) &= f(t, y(t), z(t)) \\ 0 &= g(t, y(t)) \\ \\ y(0) &= y_0 \\ z(0) &= z_0 \\ 0 &= g(0, y_0). \end{aligned} \tag{3.3.11}$$

We assume that $g_y f_z$ has bounded inverse, and so Eqn. (3.3.11) is a simple index-2 DAE. We apply the same procedure as above, and solve Eqn. (3.3.11) with an s-stage FSAL runge kutta method

$$\begin{aligned} y_{n+1} &= Y_s \\ z_{n+1} &= Z_s \\ Y_i &= y_n + h \textstyle\sum_{j=1}^{s} a_{ij} f(t + c_i h, Y_i, Z_i) \\ 0 &= g(t + c_i h, Y_i). \end{aligned} \tag{3.3.12}$$

We obtain a numerical solution u_j, v_j, and calculate a residual using Eqn. (3.3.3). We solve for the errors d_j, e_j by applying a linearized version of Eqn. (3.3.12)

$$\begin{aligned} d_{n+1} &= D_s \\ e_{n+1} &= E_s \\ D_i &= d_n \\ &\quad + h\sum_{j=1}^{s} a_{ij}\partial_y f(t+c_j h, u(t+c_j h), v(t+c_j h)) \cdot D_j \\ &\quad + h\sum_{j=1}^{s} a_{ij}\partial_z f(t+c_j h, u(t+c_j h), v(t+c_j h)) \cdot E_j \\ &\quad + \int_t^{t+c_i h} f(s, u(s), v(s))ds - u(t+c_i h) + u(t) \\ 0 &= g(t+c_i h, u(t+c_i h) + D_i). \end{aligned} \tag{3.3.13}$$

3.3.3 Algorithm description

For reference we list the entire spectral deferred correction algorithm for DAEs.

Algorithm 3.3.1 *Adaptive DAE Solver*

1. *Define user input:*
   - *Set up the DAE right hand side* $f, \partial_y f, \partial_z f, g, \partial_y g, \partial_z g$.
   - *Set up initial values* y_0, z_0, T_0.
   - *Choose a set of nodes to use (ie, Chebyshev nodes) and a degree* q *for the corresponding quadrature scheme.*
   - *Choose a runge-kutta FSAL method to use during the correction process.*
   - *Choose a tolerance* ϵ *and initial time-step* ΔT.
2. *Construct* q *nodes* $\{t_1, \ldots, t_q\}$ *within the interval* $[T_0, T_0 + \Delta T]$.
3. *Obtain an initial estimate for the solution:*

   for $j = 1, 2, \ldots, q$

   solve for u_j, v_j *using (3.3.2) or (3.3.12)*

   using the runge-kutta method chosen in step (1)

   end for

4. *Set* $u_j^{[0]} = u_j$ *and* $v_j^{[0]} = v_j$ *and* $\mu = 0$.

5. *Calculate* $R_j^{[\mu]}$ *for each* t_j.

6. *Determine if a correction is necessary:*

   - *If* $|R^{[\mu]}|$ *is less than* ϵ, *then our current solution is sufficiently resolved. Jump forward to step (9).*
   - *Otherwise, if* $|R^{[\mu]}|$ *is too large, or* $|R^{[\mu]}| > |R^{[\mu-1]}|$ *we assume the corrections are not converging. Go back to step (2) and halve* ΔT.
   - *If neither of these apply, perform a step of correction:*

   for $j = 1, 2, \ldots, q$

   Solve for $d_j^{[\mu]}$ *and* $e_j^{[\mu]}$ *using Eqn. (3.3.7) or Eqn. (3.3.13)*

   and the runge-kutta method of step (1).

   end for

7. *Set* $u_j^{[\mu+1]} = u_j^{[\mu]} + d_j^{[\mu]}$ *and* $v_j^{[\mu+1]} = v_j^{[\mu]} + e_j^{[\mu]}$.

8. *Test to see if the solution is accurate:*

   - *If* $\mu < \mu_{max}$, *set* $\mu = \mu + 1$ *and go back to step (5).*
   - *Otherwise if* $\mu = \mu_{max}$, *set* $\mu = \mu + 1$ *and calculate the residual* $R^{[\mu]}$. *If* $|R^{[\mu]}| > \epsilon$ *go back to step (2) and halve* ΔT.

9. *Accept* $u^{[\mu]}$ *as the correct solution on the interval* $[T_0, T_0 + \Delta T]$. *Move on to a new time interval* $[t_q, t_q + \Delta T]$ *and repeat the process.*

10. *If the last few time intervals were resolved without halving* ΔT, *double* ΔT *for the next time interval.*

The final numerical solution generated by this algorithm is guaranteed to have a residual less than ϵ on each time interval.

Chapter 4

Automated Coordinate Changes

We will use the two-point boundary value problem solver of Section 2.2 later, so we need a way to automatically convert an inhomogeneous boundary value problem into a homogeneous boundary value problem. Here we develop a general change of coordinates that can be applied to any two-point boundary value problem of the form

$$\begin{aligned} \Phi'(x) + p(x)\Phi(x) &= f(x) \\ A\Phi(a) + C\Phi(c) &= \gamma. \end{aligned} \tag{4.0.1}$$

The goal is to apply the transformation

$$\Gamma(x) = \Phi(x) - \delta, \tag{4.0.2}$$

where

$$(A + C)\delta = \gamma, \tag{4.0.3}$$

thus eliminating γ from the right hand side of the boundary equation. For this to be possible for general γ, the boundary matrix $D = (A + C)$ must be invertible. If D is singular we apply a preliminary transformation

$$\hat{\Phi}(x) = \mathcal{T}(x)^{-1}\Phi(x) \tag{4.0.4}$$

to obtain a new two-point boundary value problem

$$\begin{aligned} \hat{\Phi}'(x) + \mathcal{T}(x)^{-1}(\mathcal{T}'(x) + p(x)\mathcal{T}(x))\hat{\Phi}(x) &= \mathcal{T}(x)^{-1}f(x) \\ (A\mathcal{T}(a))\hat{\Phi}(a) + (C\mathcal{T}(c))\hat{\Phi}(c) &= \gamma. \end{aligned} \tag{4.0.5}$$

with new boundary matrix $\hat{D} = A\mathcal{T}(a) + C\mathcal{T}(c)$. We choose $\mathcal{T}$ so that $\hat{D}$ is invertible, and $|\mathcal{T}(x)^{-1}|$ is bounded. This allows us to transform Eqn. (4.0.5) into a homogeneous boundary value problem:

$$\Gamma(x) = \hat{\Phi} - \hat{D}^{-1}\gamma, \tag{4.0.6}$$

$$\begin{aligned} \hat{p}(x) &= \mathcal{T}(x)^{-1}(\mathcal{T}'(x) + p(x)\mathcal{T}(x)) \\ \hat{f}(x) &= \mathcal{T}(x)^{-1}f(x) - \hat{p}\hat{D}^{-1}\gamma, \end{aligned} \tag{4.0.7}$$

$$\begin{aligned} \Gamma'(x) + \hat{p}(x)\Gamma(x) &= \hat{f} \\ A\mathcal{T}(a)\Gamma(a) + C\mathcal{T}(c)\Gamma(c) &= 0. \end{aligned} \tag{4.0.8}$$

Before we solve Eqn. (4.0.8) using the two-point boundary value problem solver of Section 2.2 we must choose some background Green's function $p_0(x)$. This background Green's function produces a fundamental solution matrix $\Upsilon(x)$ that satisfies

$$\Upsilon'(x) + p_0(x)\Upsilon(x) = 0. \tag{4.0.9}$$

The two-point boundary value problem solver requires that

$$\hat{D}_\Upsilon = A\mathcal{T}(a)\Upsilon(a) + C\mathcal{T}(c)\Upsilon(c) \tag{4.0.10}$$

be nonsingular. Here a problem arises. If our original p_0 was close to p, then there is no guarantee that p_0 will be close to $\hat{p}$. In fact, if p_0 is close to p, then a natural choice for a background Green's function for Eqn. (4.0.8) is

$$\hat{p_0}(x) = \mathcal{T}(x)^{-1}(\mathcal{T}'(x) + p_0(x)\mathcal{T}(x)). \tag{4.0.11}$$

This produces a fundamental solution matrix $\hat{\Upsilon}(x)$ that satisfies

$$\begin{aligned} &\hat{\Upsilon}'(x) + \hat{p_0}(x)\hat{\Upsilon}(x) = 0 \\ &\hat{\Upsilon}'(x) + \mathcal{T}(x)^{-1}(\mathcal{T}(x) + p_0(x)\mathcal{T}(x))\hat{\Upsilon}(x) = 0 \\ &\mathcal{T}(x)\hat{\Upsilon}'(x) + \mathcal{T}'(x)\hat{\Upsilon}(x) + p_0(x)\mathcal{T}(x)\hat{\Upsilon}(x) = 0 \\ &(\mathcal{T}(x)\hat{\Upsilon}(x))' + p_0(x)(\mathcal{T}(x)\hat{\Upsilon}(x)) = 0 \\ &\mathcal{T}(x)\hat{\Upsilon}(x) = \Upsilon(x) \\ &\hat{\Upsilon}(x) = \mathcal{T}(x)^{-1}\Upsilon(x). \end{aligned} \tag{4.0.12}$$

So $\hat{D}_{\hat{\Upsilon}} = A\mathcal{T}(a)\hat{\Upsilon}(a) + C\mathcal{T}(c)\hat{\Upsilon}(c) = A\Upsilon(a) + C\Upsilon(c) = D_{\Upsilon}$, which does not depend on $\mathcal{T}$ at all. Therefore if we insist on choosing $\hat{p_0}$ close to $\hat{p}$, $\hat{D}_{\hat{\Upsilon}}$ may be singular. For example, consider the two-point boundary value problem given by

$$\begin{aligned} &\Phi'(x) + (i + \epsilon(x))\Phi(x) = 0 \\ &\Phi(0) + \Phi(\pi) = 1. \end{aligned} \tag{4.0.13}$$

After inspecting the boundary data, the natural choice for $\mathcal{T}$ is 1. However, if we choose $\hat{p_0} = i$, we obtain $\hat{D}_{\hat{\Upsilon}} = 0$, which is unacceptable. Another example is given by the two-point boundary value problem

$$\begin{aligned} &\Phi'(x) + \epsilon(x)\Phi(x) = 0 \\ &\Phi(0) - \Phi(1) = 1. \end{aligned} \tag{4.0.14}$$

In this case $D = 0$, and so we must choose $\mathcal{T}$ so that $\hat{D} \neq 0$ ($\mathcal{T}(x) = (2 - x)$ will suffice). However, no matter what $\mathcal{T}$ is, if we choose $p_0 = 0$ and $\hat{p_0} = \mathcal{T}^{-1}(\mathcal{T}')$, we end up with $\hat{D}_{\hat{\Upsilon}} = 0$.

Therefore we abandon any hope of choosing $\hat{p_0}$ close to $\hat{p}$. Instead, we concentrate on the case of $p_0 = 0$, and $\hat{D} = \hat{D}_{\Upsilon}$. For the remainder of this section we assume that $[a, c] = [-1, 1]$, and that we are given boundary data with singular D. We focus on the problem of constructing $\mathcal{T}$ such that $\hat{D} = A\mathcal{T}(-1) + C\mathcal{T}(+1)$ is nonsingular. The column space of $\hat{D}$ lies within the column space of $[A|C]$. So $[A|C]$ must have full rank in order for $\mathcal{T}(x)$ to exist. We will show that this simple necessary condition for the existance of $\mathcal{T}$ is also sufficient. The apparatus we construct revolves around permutations, reflections and scaling:

Consider first, the case

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, C = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}. \tag{4.0.15}$$

We need $\hat{D}$ to be nonsingular, so we require

$$\hat{D}\mathcal{T}(-1)^{-1} = A + C\mathcal{T}(+1)\mathcal{T}(-1)^{-1} \tag{4.0.16}$$

to be invertible. So $\mathcal{T}(+1)\mathcal{T}(-1)^{-1}$ must permute the columns of C. If $\mathcal{T}(-1) = I$, a good choice for $\mathcal{T}(+1)$ is

$$\mathcal{T}(+1) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \tag{4.0.17}$$

A reflection is necessary, since the continuity and invertibility of $\mathcal{T}(x)$ requires the sign of $\det(\mathcal{T}(+1))$ to match the sign of $\det(\mathcal{T}(-1))$.

Now consider the case

$$A = 1, C = -1. \tag{4.0.18}$$

Here we only have one dimension to work with, and there is no room to rotate. Our transformation $\mathcal{T}$ must scale, and $\mathcal{T}(-1)$ must be different from $\mathcal{T}(+1)$. In addition, since $\mathcal{T}(x)$ must be continuous and invertible for all x, $\mathcal{T}(-1)$ and $\mathcal{T}(+1)$ must be of the same sign. Any simple scaling will do, such as $\mathcal{T}(x) = 1 + (\epsilon - 1)(x + 1)/2$.

Our algorithm for constructing $\hat{D}$ and $\mathcal{T}$ accommodates permutations, reflections and scalings, and we ensure that the determinant of $\mathcal{T}$ is nonzero over the interval $[-1, +1]$. The basic idea behind our algorithm is this:

- $[A|C]$ has $2n$ columns, but only n of them are needed for $[A|C]$ to have full rank.
- We find these n relevant 'dominant' columns. Some of them will come from A, and some will come from C.
- Sometimes (as in the case of our second example), all (or most) of them will come from one of the boundary matrices. In this case we apply a scaling that mitigates the effect of the other irrelevant 'nondominant' columns from the other boundary matrix.
- Sometimes (as in the case of the first example), some of the dominant columns will come from one boundary matrix, and some will come from the other boundary matrix. In this case we permute the relevant dominant columns into the correct positions so that $\hat{D}$ has full rank.
- Most of the time both of these effects happen simultaneously. We choose a $\mathcal{T}$ that permutes the dominant columns into the correct positions and scales the nondominant columns so that they do not affect the rank of $\hat{D}$.

First we perform an LU decomposition

$$\begin{aligned} P \cdot [A|C]^{\mathrm{T}} &= L \cdot U \\ P &= 2n \times 2n \quad \text{permutation matrix} \\ L &= 2n \times n \quad \text{lower triangular matrix} \\ U &= n \times n \quad \text{upper triangular matrix,} \end{aligned} \tag{4.0.19}$$

We inspect the first n rows of P. The nonzero entries correspond to the n dominant columns of $[A|C]$. We record these column indices and divide them into two groups:

1. $i_1^A, \ldots, i_p^A$ correspond to the p dominant columns of A

2. $i_{p+1}^C, \ldots, i_n^C$ correspond to the $n-p$ dominant columns of C.

We then record the column indices of the irrelevant columns of A and C:

1. $i_{p+1}^A, \ldots, i_n^A$ correspond to the $n-p$ irrelevant columns of A

2. $i_1^C, \ldots, i_p^C$ correspond to the p irrelevant columns of C.

We define permutations

$$\eta_A(j) = i_j^A \tag{4.0.20}$$

$$\eta_C(j) = i_j^C \tag{4.0.21}$$

and permutation matrices

$$P_{ij}^{\eta_A} = \delta_{i,\eta_A(j)} \tag{4.0.22}$$

$$P_{ij}^{\eta_C} = \delta_{i,\eta_C(j)}. \tag{4.0.23}$$

The permutation matrix P^{η_A} shuffles the p dominant columns of A to the front, and P^{η_C} shuffles the $n-p$ dominant columns of C to the back. The first p columns of AP^{η_A} and the last $n-p$ columns of CP^{η_C} can be combined to make an invertible matrix.

Now we concentrate on scaling down the effect of the irrelevant columns of $[A|C]$. We define scaling matrices

$$S_{ij}^{p,\epsilon,\nu}(x) = \begin{cases} \epsilon\delta_{ij} & j \leq p \\ \nu\delta_{ij} & j > p. \end{cases} \tag{4.0.24}$$

The nonzero columns of $AP^{\eta_A}S^{p,1,0}$ are the p dominant columns of A. Similarly, the nonzero columns of $CP^{\eta_C}S^{p,0,1}$ are the $n-p$ dominant columns of C. The sum

$$AP^{\eta_A}S^{p,1,0} + CP^{\eta_C}S^{p,0,1} \tag{4.0.25}$$

is invertible. Because the determinant is a continuous function, the sum

$$AP^{\eta_A}S^{p,1,\epsilon} + CP^{\eta_C}S^{p,\epsilon,1} \tag{4.0.26}$$

is invertible for some $\epsilon > 0$. Finding the maximum ϵ is difficult, so we simply try different values until we find one that works. In practice we almost always find $\epsilon \in [1/8, 1/2]$. This suggests a choice of $\mathcal{T}(-1) = P^{\eta_A}S^{p,1,\epsilon}$ and $\mathcal{T}(+1) = P^{\eta_C}S^{p,\epsilon,1}$. However, if we make this choice we may not be able to construct an invertible matrix path $\mathcal{T}(x)$ between these two boundary values. In particular, $P^{\eta_A}S^{p,1,\epsilon}$ and $P^{\eta_C}S^{p,\epsilon,1}$ must have the same determinant.

To ensure this we design reflection matrices:

$$F_{ij}^{\mathcal{L}} = \begin{cases} \delta_{ij} & j \notin \mathcal{L} \\ -\delta_{ij} & j \in \mathcal{L}, \end{cases} \tag{4.0.27}$$

where $\mathcal{L}$ can be any set of indices. We form the matrices

$$\mathcal{T}(-1) = P^{\eta_A}S^{p,1,\epsilon} \tag{4.0.28}$$

$$\mathcal{T}(+1) = P^{\eta_C}S^{p,\epsilon,1}F^{\mathcal{L}}, \tag{4.0.29}$$

where $\mathcal{L}$ has not yet been chosen. We construct the scalar paths

$$\begin{aligned} c(x) &= \cos(\pi(x+1)/4) \\ s(x) &= \sin(\pi(x+1)/4) \\ \gamma(x) &= \epsilon + (x+1)(1-\epsilon)/2 \end{aligned} \tag{4.0.30}$$

and the matrix paths

$$\begin{aligned} P(x) &= c(x)P^{\eta_A} + s(x)P^{\eta_C}F^{\mathcal{L}} \\ S(x) &= S^{p,\gamma(x),\gamma(-x)} \\ \mathcal{T}(x) &= P(x)S(x). \end{aligned} \tag{4.0.31}$$

With this choice of $\mathcal{T}$ we have

$$\hat{D} = A\mathcal{T}(-1) + C\mathcal{T}(+1) = AP^{\eta_A}S^{p,1,0} + CP^{\eta_C}S^{p,0,1}F^{\mathcal{L}}. \tag{4.0.32}$$

This is the invertible boundary matrix we want. Note that $\det(S(x)) \geq \epsilon^n$ for every x. So $\mathcal{T}(x)$ is invertible if and only if $P(x)$ is invertible. We will show how to choose $\mathcal{L}$ such that $P(x)$ is invertible for all x.

We apply a preliminary transformation

$$(P^{\eta_A})^{-1}P(x) = c(x)I + s(x)(P^{\eta_A})^{-1}P^{\eta_C}F^{\mathcal{L}}. \tag{4.0.33}$$

We note the following

Lemma 4.0.2

$$P^{\eta_2}P^{\eta_1} = P^{\eta_2 \cdot \eta_1}$$

PROOF. Definitions 4.0.22 and 4.0.23 yield

$$\begin{aligned} [P^{\eta_2}P^{\eta_1}]_{ik} &= \sum_{j=1}^{n} P_{ij}^{\eta_2}P_{jk}^{\eta_1} \\ &= \sum_{j=1}^{n} \delta_{i,\eta_2(j)}\delta_{j,\eta_1(k)} \\ &= \delta_{i,\eta_2(\eta_1(k))} \\ &= P^{\eta_2 \cdot \eta_1}. \end{aligned}$$

□

Using Lemma 4.0.2 we define

$$\hat{\eta} = \eta_A^{-1}\eta_C, \tag{4.0.34}$$

and write

$$\hat{P}(x) = (P^{\eta_A})^{-1}P(x) = c(x)I + s(x)P^{\hat{\eta}}F^{\mathcal{L}}. \tag{4.0.35}$$

We need to show that $\hat{P}(x)$ is invertible. To do this we decompose $\hat{\eta}$ into k disjoint cycles (cyclic permutations) [30]

$$\hat{\eta} = \alpha_k \cdot \alpha_{k-1} \cdots \alpha_1, \tag{4.0.36}$$

where each α_i has length l_i. We associate to each α_i the set of indices (coordinates) β_i that it permutes. The β_i form a partition of the set $\{1, \ldots, n\}$. We note the following

Lemma 4.0.3 *If we write $\hat{\eta}$ as the product of k disjoint cycles $\hat{\eta} = \Pi_{i=k}^{1} \alpha_i$ each of length l_i, then $P^{\hat{\eta}} = P^{\alpha_k} \cdots P^{\alpha_1}$ is a matrix with k blocks, each of size $l_i \times l_i$. If we associate with each α_i its set β_i of permuted indices, then the blocks of P^{η} are those associated with the coordinate groups β_j.*

PROOF. Repeatedly apply Lemma 4.0.2. This block matrix becomes visually obvious if we reorder coordinates such that the indices in each cycle α_{j+1} are larger than the indices in the previous cycle α_j. (Alternately we could require the entries of β_{j+1} to be larger than the entries of β_j). □

Armed with this cycle decomposition of $\hat{\eta}$, we write:

$$\hat{P} = c(x)I + s(x)P^{\alpha_k} \cdots P^{\alpha_1} F^{\mathcal{L}}. \tag{4.0.37}$$

It is convenient to define the restriction

$$[P|_\beta]_{ij} = \begin{cases} P_{ij} & i, j \in \beta \\ \delta_{ij} & \text{otherwise.} \end{cases} \tag{4.0.38}$$

Note that

$$P^{\alpha_i}|_{\beta_i} = P^{\alpha_i} \tag{4.0.39}$$

and

$$F^{\mathcal{L}}|_{\beta_i} = F^{\mathcal{L} \cap \beta_i}. \tag{4.0.40}$$

Note also that since the β_i form a partition of $\{1,\ldots,n\}$, the product $\Pi_{i=k}^1 A|_{\beta_i}$ is simply the matrix A restricted to the blocks β_i (with zeros elsewhere). Using Eqn. (4.0.38) we rewrite Eqn. (4.0.37) as

$$\begin{aligned}\hat{P} &= \Pi_{i=k}^1(c(x)I)|_{\beta_i} + \Pi_{i=k}^1(s(x)P^{\alpha_i}F^{\mathcal{L}})|_{\beta_i} \\ &= \Pi_{i=k}^1(c(x)I + s(x)P^{\alpha_i}F^{\mathcal{L}})|_{\beta_i} \\ &= \Pi_{i=k}^1(c(x)|_{\beta_i} + s(x)|_{\beta_i}P^{\alpha_i}F^{\mathcal{L}\cap\beta_i}).\end{aligned} \tag{4.0.41}$$

So the determinant of $\hat{P}$ is given by the product of the determinant of the coordinate blocks

$$\det(\hat{P}) = \Pi_{i=k}^1 \det(c(x)|_{\beta_i} + s(x)|_{\beta_i}P^{\alpha_i}F^{\mathcal{L}\cap\beta_i}). \tag{4.0.42}$$

To prove that each of these coordinate blocks has nonzero determinant we use the following

Lemma 4.0.4 *If α is a single cycle of length n operating on every coordinate, then*

$$\det(c(x)I + s(x)P^{\alpha}F^{\mathcal{L}}) = c^n(x) + s^n(x)\mathit{parity}(\alpha)\det(F^{\mathcal{L}}).$$

PROOF. We use the following definition of determinant: Given an $n\times n$ matrix M, let $\varepsilon(\zeta)$ be the fully alternating antisymmetric n-tensor:

$$\varepsilon(\zeta) = \begin{cases} 0 & \zeta \text{ not an } n\text{-cycle} \\ \text{parity of } \zeta & \zeta \text{ is an } n\text{-cycle.}\end{cases} \tag{4.0.43}$$

The determinant of M is given by

$$\begin{aligned}\det(M) &= \textstyle\sum_{\zeta\in S_n}\varepsilon(\zeta)M_{1,\zeta(1)}M_{2,\zeta(2)}\cdots M_{n,\zeta(n)} \\ &= \textstyle\sum_{\zeta\in S_n}\varepsilon(\zeta)\Pi_{j=1}^{j=n}M_{j,\zeta(j)}.\end{aligned} \tag{4.0.44}$$

Now when considering terms of

$$M = c(x)I + s(x)P^{\alpha}F^{\mathcal{L}}$$

we have the following possibilities:

1. the single term $c^n(x)$ corresponds to $\zeta(j) = j$, and so $\varepsilon(\zeta) = 1$
2. the single term $\det(F^{\mathcal{L}})s^n(x)$ corresponds to $\zeta(j) = \alpha(j)$ and so $\varepsilon(\zeta) = \text{parity}(\alpha)$
3. the cross terms $\pm c^m(x)s^{n-m}(x)$ for $0 < m < n$ means that we have taken some $s(x)$ terms and some $c(x)$ terms from M.

We examine this last case. Since α is a full n-cycle we know that each row and each column of M has one $s(x)$ term and one $c(x)$ term. We also know that the $c(x)$ terms are on the diagonal. So any term of the form $c^m(x)s^{n-m}(x)$ must take m $c(x)$ terms from the diagonal of M. Now there are at least m $s(x)$ terms in those m particular rows and columns of M. If there were only m $s(x)$ terms in those particular m rows and columns of M, then they would all have to be in the $m \times m$ subblock of M which contains the m $c(x)$ terms. However, since α is an n-cycle, no $m \times m$ subblock of P^α can have m s(x) terms in it. So there must be at least $m+1$ $s(x)$ terms in those m particular rows and columns of M. That leaves only $n-m-1$ $s(x)$ terms in the remaining $n-m \times n-m$ block submatrix of M. In order for $\varepsilon(\zeta)$ to be nonzero, the $n-m$ $s(x)$ terms must have come from the remaining $n-m \times n-m$ block submatrix of M. This is not possible, and so $\varepsilon(\zeta)$ must be zero. So

$$\det(M) = c^n(x) + \det(F^{\mathcal{L}})\text{parity}(\alpha)s^n(x)$$

and we are done. □

Under the assumptions of the lemma, we can easily choose $\mathcal{L}$ so that M is invertible. If α has positive parity, we choose $\mathcal{L}$ to be the empty set. Otherwise, if α has negative parity, we choose $\mathcal{L}$ to be the singleton set $\{i\}$, where i is any one index permuted by α. A direct consequence of Lemma 4.0.4 is

Lemma 4.0.5 *If $\hat{\eta} = \alpha_k \cdots \alpha_1$ is a disjoint cycle decomposition with corresponding coordinate blocks $\{\beta_k, \ldots, \beta_1\}$ of respective lengths $\{l_k, \ldots, l_1\}$, then*

$$\det(\hat{P}(x)) = \pm\Pi_{i=k}^{1} c^{l_i}(x) + \det(F^{\mathcal{L}\bigcap\beta_i})\textit{parity}(\alpha_i)s^{l_i}(x).$$

We choose $\mathcal{L}$ to contain one element of each β_j that corresponds to an α_j with negative parity. This ensures that $\det(F^{\mathcal{L}\bigcap\beta_i})$ has the same sign as parity(α_i), and that $\hat{P}(x)$ is invertible. This corresponds to the simple

Algorithm 4.0.2 *Build $\mathcal{L}$*

1. *Initialize $\mathcal{L}$ to be the empty set*

2. *Determine a disjoint cycle decomposition $\hat{\eta} = \alpha_k \cdots \alpha_1$*

3. *for $i = 0, 1, \ldots, k$*

   if α_i has negative parity

   take an index permuted by α_i and append it to $\mathcal{L}$

   end if

end for

This establishes the invertibility of $\hat{P}(x)$, $P(x)$ and $\mathcal{T}(x)$. The construction of Eqn. (4.0.31) is useful since $\mathcal{T}$ is as smooth as c and s. However, inversion of $\mathcal{T}$ is not especially easy. For reference we construct an alternate coordinate transformation that is easy to invert: Given a transposition τ of indices l, m, we define the matrix path $P^{\tau}_{[x_1,x_2]}(x)$ which connects the transposition over a small interval

$$P^{\tau}_{[x_1,x_2]}(x) = c((x - \frac{x_1+x_2}{2})\frac{x_2-x_1}{2})I + s((x - \frac{x_1+x_2}{2})\frac{x_2-x_1}{2})P^{\tau}F^{\{l\}}. \tag{4.0.45}$$

This path is designed so $P^{\tau}_{[x_1,x_2]}(x_1) = I$ is the identity matrix, and $P^{\tau}_{[x_1,x_2]}(x_2) = P^{\tau}F^{l_\tau}$ is the identity matrix with two rows swapped (those specified by τ) and one negated. We decompose $\hat{\eta} = \tau_k \cdots \tau_1$ as the product of k transpositions, and let l_i be one of the indices swapped by τ_i. We select $k+1$ points x_i in the interval $[-1, 1]$, and write

$$P(x) = \begin{cases} P^{\eta_A} P^{\tau_1}_{[x_0,x_1]}(x) & x \in [x_0, x_1] \\ P^{\eta_A} P^{\tau_1} F^{l_1} P^{\tau_2}_{[x_1,x_2]}(x) & x \in [x_1, x_2] \\ P^{\eta_A} P^{\tau_2\tau_1} F^{l_1,l_2} P^{\tau_3}_{[x_2,x_3]}(x) & x \in [x_2, x_3] \\ \vdots & \vdots \\ P^{\eta_A} P^{\tau_k \cdots \tau_1} F^{l_1,\ldots,l_k} P^{\tau_k}_{[x_{k-1},x_k]}(x) & x \in [x_{k-1}, x_k]. \end{cases} \tag{4.0.46}$$

The path $P(x)$ goes through k steps, swapping two columns with each step. It is convenient to choose c and s such that $c+s = 1$ so each path $P^{\tau}_{[x_1,x_2]}$ acts only on the columns swapped by τ, and does not

scale any other columns. (Note that $c(x) = \cos^2((x+1)pi/4)$ and $s(x) = \sin^2((x+1)pi/4)$ will do the job). The construction of Eqn. (4.0.46) is useful because it is especially easy to invert. Unfortunately, the $\mathcal{T}$ constructed is not necessarily as smooth as c and s. (in the case of $c = \cos^2$,$s = \sin^2$, $\mathcal{T}$ is only C^1). However, we can choose c and s such that $\mathcal{T}$ is as smooth as necessary.

Chapter 5

Initial-Boundary Value Problem for Evolutionary PDEs

Given a differentiable function

$$F : [a, c] \times [0, T] \times R^n \times R^n \times R^n \to R^n, \tag{5.0.1}$$

boundary matrices

$$A, C : [0, T] \to R^{2n \times 2n}, \tag{5.0.2}$$

boundary vector

$$\gamma : [0, T] \to R^{2n}, \tag{5.0.3}$$

and consistent smooth initial data

$$u_0 : [a, c] \to R^n, \tag{5.0.4}$$

a well-posed 1+1 dimensional initial-boundary value problem requires a differentiable function

$$u : [a, c] \times [0, T] \to R^n \tag{5.0.5}$$

to satisfy

$$\begin{gathered} u_t(x,t) = F(x,t,u(x,t),u_x(x,t),u_{xx}(x,t)) \\ A(t)\begin{bmatrix} u(a,t) \\ u_x(a,t) \end{bmatrix} + C(t)\begin{bmatrix} u(c,t) \\ u_x(c,t) \end{bmatrix} = \gamma(t) \\ u(x,0) = u_0(x). \end{gathered} \tag{5.0.6}$$

Eqn. (5.0.6) is quite general, and accommodates many physical systems including fluid flow and the 1-dimensional Schrödinger equation. If we assume a density u and a corresponding flux with a small diffusive component controlled by a parameter ϵ, we obtain a viscous conservation law

$$\begin{gathered} u_t(x,t) + \tfrac{d}{dx}(f(x,t,u(x,t)) - \epsilon u_x(x,t)) = 0 \\ A(t)\begin{bmatrix} u(a,t) \\ u_x(a,t) \end{bmatrix} + C(t)\begin{bmatrix} u(c,t) \\ u_x(c,t) \end{bmatrix} = \gamma(t) \\ u(x,0) = u_0(x). \end{gathered} \tag{5.0.7}$$

This is a special case of Eqn. (5.0.6), and can be used to approximate the vanishing viscosity limit of a hyperbolic conservation law [14]. As long as ϵ remains positive, smooth solutions exist for all time. The limit of these solutions as $\epsilon \to 0$ is the physically relevant weak solution of the hyperbolic conservation law

$$u_t(x,t) + \tfrac{d}{dx} f(x,t,u(x,t)) = 0. \tag{5.0.8}$$

The conservation law 5.0.8 only requires n independent boundary relations. Therefore we must carefully assign boundary conditions to avoid spurious boundary layers. In general, we choose the boundary conditions for Eqn. (5.0.7) to be consistent with the corresponding inviscid problem.

5.1 Outline of the numerical method

Our method approximates initial-boundary problems by a sequence of boundary value problems. We choose a time interal $[0,T]$, and a sequence of time values $t_0,\ldots,t_p$ corresponding to the Chebyshev nodes on that interval. Then we formally apply implicit Euler to Eqn. (5.0.6):

$$\frac{u_{n+1}(x) - u_n(x)}{t_{n+1} - t_n} = F(x,t_{n+1},u_{n+1}(x),\partial_x u_{n+1}(x),\partial_{xx} u_{n+1}(x)). \tag{5.1.1}$$

This is equivalent to replacing the time-derivative in the PDE with a first order implicit finite difference. We approximate the true solution $u(x,t_n)$ at time t_n by the numerical solution $u_n(x)$ (a function of x). Letting

$$k_{n+1} = t_{n+1} - t_n, \tag{5.1.2}$$

we rewrite Eqn. (5.1.1) as

$$u_{n+1}(x) = u_n(x) + k_{n+1}F(x, t_{n+1}, u_{n+1}(x), \partial_x u_{n+1}(x), \partial_{xx} u_{n+1}(x)), \tag{5.1.3}$$

with boundary conditions

$$A(t_{n+1}) \begin{bmatrix} u_{n+1}(a) \\ \partial_x u_{n+1}(a) \end{bmatrix} + C(t_{n+1}) \begin{bmatrix} u_{n+1}(c) \\ \partial_x u_{n+1}(c) \end{bmatrix} = \gamma(t_{n+1}). \tag{5.1.4}$$

One step of newton's method applied to Eqn. (5.1.3) yields

$$(I - k_{n+1}H - k_{n+1}J\partial_x - k_{n+1}K\partial_{xx})(u_{n+1} - u_n) = k_{n+1}F, \tag{5.1.5}$$

where we define

$$\begin{array}{rcll} F & = & F_{n+1}(x) & = F(x, t_{n+1}, u_n(x), \partial_x u_n(x), \partial_{xx} u_n(x)) \\ H & = & H_{n+1}(x) & = D_u F(x, t_{n+1}, u_n(x), \partial_x u_n(x), \partial_{xx} u_n(x)) \\ J & = & J_{n+1}(x) & = D_{u_x} F(x, t_{n+1}, u_n(x), \partial_x u_n(x), \partial_{xx} u_n(x)) \\ K & = & K_{n+1}(x) & = D_{u_{xx}} F(x, t_{n+1}, u_n(x), \partial_x u_n(x), \partial_{xx} u_n(x)) \\ M & = & M_{n+1}(x) & = K_{n+1}(x)^{-1}. \end{array} \tag{5.1.6}$$

Note that because u_n is our initial guess for newton's method F, H, J, K, M are all evaluated at the previous time-step u_n. We define solution vectors

$$\begin{array}{rcl} v_n(x) & = & u_n(x) \\ w_n(x) & = & \partial_x u_n(x), \end{array} \tag{5.1.7}$$

and rewrite Eqn. (5.1.5) as a first order system:

$$\partial_x \begin{bmatrix} v_{n+1} \\ w_{n+1} \end{bmatrix} + \begin{bmatrix} 0 & -I \\ M(H - \frac{1}{k}I) & MJ \end{bmatrix} \cdot \begin{bmatrix} v_{n+1} \\ w_{n+1} \end{bmatrix} = \begin{bmatrix} 0 \\ M(H - \frac{1}{k}I)v_n + MJw_n + \partial_x w_n - MF \end{bmatrix} \tag{5.1.8}$$

with boundary conditions

$$A_{n+1}\begin{bmatrix} v_{n+1}(a) \\ w_{n+1}(a) \end{bmatrix} + C_{n+1}\begin{bmatrix} v_{n+1}(c) \\ w_{n+1}(c) \end{bmatrix} = \gamma_{n+1}. \tag{5.1.9}$$

Note that there is no spatial discretization. Rather, $u_n(x)$ is a function of x at each time t_n. Space will eventually be discretized by the two-point boundary value problem solver of Section 2.2, but we treat the two-point boundary value problem solver as a black box which returns a differentiable function.

Eqns. (5.1.8, 5.1.9) can be applied to Eqn. (5.0.7) (a viscous conservation law). In this case M is simply $1/\epsilon$, and the numerical solution solves

$$\partial_x \begin{bmatrix} v_{n+1} \\ w_{n+1} \end{bmatrix} + \begin{bmatrix} 0 & -I \\ H/\epsilon - I/(k\epsilon) & J/\epsilon \end{bmatrix} \cdot \begin{bmatrix} v_{n+1} \\ w_{n+1} \end{bmatrix} = \begin{bmatrix} 0 \\ (H/\epsilon - I/(k\epsilon))v_n + Jw_n/\epsilon + \partial_x w_n - F/\epsilon \end{bmatrix}. \tag{5.1.10}$$

with boundary contions given by Eqn. (5.1.9).

We use the spectral deferred correction methods of Section 3 to calculate the residual, solve for the error and improve our approximate solution to the initial-boundary problem. The residual is

$$R_j(x) = u_0(x) + \int_0^{t_j} F(x, s, u(x,s), \partial_x u(x,s), \partial_{xx} u(x,s))ds - u_j(x), \tag{5.1.11}$$

and can be computed accurately because the time values t_j admit accurate time-wise integration. After constructing the residual we solve the error equations.

$$E_{n+1} = \Delta R + E_n + kHE_{n+1} + kJ\partial_x E_{n+1} + kK\partial_{xx}E_{n+1} \tag{5.1.12}$$

where our definitions have changed:

$$\begin{array}{rcl} \Delta R = \Delta R_{n+1}(x) & = & u_n(x) - u_{n+1}(x) + \int_{t_n}^{t_{n+1}} F(x, s, u, \partial_x u, \partial_{xx} u)ds \\ F = F_{n+1}(x) & = & F(x, t_{n+1}, u_{n+1}(x), \partial_x u_{n+1}(x), \partial_{xx} u_{n+1}(x)) \\ H = H_{n+1}(x) & = & D_u F(x, t_{n+1}, u_{n+1}(x), \partial_x u_{n+1}(x), \partial_{xx} u_{n+1}(x)) \\ J = J_{n+1}(x) & = & D_{u_x} F(x, t_{n+1}, u_{n+1}(x), \partial_x u_{n+1}(x), \partial_{xx} u_{n+1}(x)) \\ K = K_{n+1}(x) & = & D_{u_{xx}} F(x, t_{n+1}, u_{n+1}(x), \partial_x u_{n+1}(x), \partial_{xx} u_{n+1}(x)) \\ M = M_{n+1}(x) & = & K_{n+1}(x)^{-1}. \end{array} \tag{5.1.13}$$

Note that in Eqn. (5.1.12), F, H, J, K, M are evaluated at the current time-step u_{n+1} as opposed to the previous time-step u_n. We introduce

$$E'_n(x) = \partial_x E_n(x), \tag{5.1.14}$$

and turn Eqn. (5.1.12) into a first order system:

$$\partial_x \begin{bmatrix} E_{n+1} \\ E'_{n+1} \end{bmatrix} + \begin{bmatrix} 0 & -I \\ M(H - \frac{1}{k}I) & MJ \end{bmatrix} \cdot \begin{bmatrix} E_{n+1} \\ E'_{n+1} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{k}M(R_n - R_{n+1} - E_n) \end{bmatrix} \tag{5.1.15}$$

with boundary conditions

$$A_{n+1} \begin{bmatrix} E_{n+1}(a) \\ E'_{n+1}(a) \end{bmatrix} + C_{n+1} \begin{bmatrix} E_{n+1}(c) \\ E'_{n+1}(c) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{5.1.16}$$

Each step of spectral deferred correction is again a two-point boundary value problem, and we solve the error equations with the same two-point boundary value problem solver used for Eqn. (5.1.8). The correction process is iterated until we are satisfied with the solution. Note that if each two-point boundary value problem is solved adaptively a different spatial grid will be used for each fixed time and for each subsequent correction. This will require us to combine solutions calculated on different grids before determining the residual $R(x,t)$ on our domain.

Chapter 6

Convergence Proofs

Here we prove the convergence of the backward Euler scheme for a few special cases. For the remainder of this section we assume we are given a viscous conservation law of form Eqn. (5.0.7). The implicit Euler scheme is given by

$$\begin{gathered} k_{j+1} = t_{j+1} - t_j \\ u_{j+1}(x) = u_j + k_{j+1}\frac{d}{dx}(\epsilon\partial_x u_{j+1}(x) - f(x, t_{j+1}, u_{j+1}(x))) \\ A(t_{j+1}) \cdot \begin{bmatrix} u_{j+1}(a) \\ \partial_x u_{j+1}(a) \end{bmatrix} + C(t_{j+1}) \cdot \begin{bmatrix} u_{j+1}(c) \\ \partial_x u_{j+1}(c) \end{bmatrix} = \gamma(t_{j+1}). \end{gathered} \tag{6.0.1}$$

The t_j correspond to quadrature nodes, and are not equispaced. Each $u_j(x)$ is a function of x. Eqn. (6.0.1) is a two-point boundary value problem , and the successive solution of Eqn. (6.0.1) for $j = 0, 1, \ldots$ yields a numerical solution for every t_j.

6.1 Truncation error

If we plug the exact solution $y = y(x,t)$ into the method given by Eqn. (6.0.1), and we assume appropriate bounds on the derivatives of u, then we obtain a local truncation error τ_j which is $O(k_{j+1})$:

$$\begin{gathered} \frac{y(x,t_{j+1}) - y(x,t_{j+1})}{k_{j+1}} + \frac{d}{dx} f(x, t_{j+1}, y(x, t_{j+1})) \\ = \epsilon\partial_{xx} y(x, t_{j+1}) + \tau_j(x) \end{gathered} \tag{6.1.1}$$

$$\begin{aligned} &\partial_t y(x,t_{j+1}) + \tfrac{1}{2}\partial_{tt} y(x,t_{j+1})k_{j+1} + O(k_{j+1}^2) \\ &\quad + \tfrac{d}{dx} f(x,t_{j+1},y(x,t_{j+1})) \\ &\quad = \epsilon \partial_{xx} y(x,t_{j+1}) + \tau_j(x) \end{aligned} \tag{6.1.2}$$

$$\begin{aligned} &\tfrac{1}{2}\partial_{tt} y(x,t_{j+1})k_{j+1} + O(k_{j+1}^2) \\ &\quad = \tau_j(x) = O(k_{j+1}). \end{aligned} \tag{6.1.3}$$

6.2 Boundary value problem error

We account for the errors in the solution of the boundary value problem by assuming that our boundary value problem solver is backwards stable. More specifically, we rewrite Eqn. (6.0.1) as a first order system

$$\begin{aligned} v &= u_{j+1} \\ w &= \partial_x u_{j+1} \end{aligned} \tag{6.2.1}$$

$$\begin{aligned} \partial_x v - w &= 0 \\ \partial_x w + \tfrac{u_j - v}{k\epsilon} - \tfrac{1}{\epsilon}\tfrac{d}{dx} f(x,t_{j+1},v) &= 0 \\ A_{j+1} \cdot \begin{bmatrix} v(a) \\ w(a) \end{bmatrix} + C_{j+1} \cdot \begin{bmatrix} v(c) \\ w(c) \end{bmatrix} &= \gamma_{j+1}, \end{aligned} \tag{6.2.2}$$

and assume that the computed solution $\hat{v}, \hat{w}$ satisfies the following nearby two-point boundary value problem:

$$\begin{aligned} \partial_x \hat{v} - \hat{w} &= \delta_1 \\ \partial_x \hat{w} + (\hat{v} - u_j)/(k\epsilon) + \tfrac{1}{\epsilon}\partial_x f(x,t_{j+1},\hat{v}) &= \delta_2 \\ A(t_{j+1}) \cdot \begin{bmatrix} \hat{v}(a) \\ \hat{w}(a) \end{bmatrix} + C(t_{j+1}) \cdot \begin{bmatrix} \hat{v}(c) \\ \hat{w}(c) \end{bmatrix} &= \gamma(t_{j+1}) + \begin{bmatrix} \delta_3 \\ \delta_4 \end{bmatrix}, \end{aligned} \tag{6.2.3}$$

with $\delta_1, \delta_2, \delta_3, \delta_4$ all small. If this is the case, we define our numerical solution to be

$$\breve{v}(x) = \hat{v}(a) + \int_a^x \hat{w}(s)ds. \tag{6.2.4}$$

This guarantees that

$$\begin{aligned} \partial_x \breve{v}(x) &= \hat{w}(x) \\ \breve{v}(x) - \hat{v}(x) &= \textstyle\int_a^x \delta_1(s)ds. \end{aligned} \tag{6.2.5}$$

Assuming the appropriate bounds on the derivatives of f, we have

$$\begin{array}{c} \partial_x \hat{w} + (\breve{v} - u_j)/(k\epsilon) - \\ \frac{1}{\epsilon}(\partial_x f(x, t_{j+1}, \breve{v}) + \partial_u f(x, t_{j+1}, \breve{v})\hat{w}) = \\ \delta_2(x) + (1 + O(k)) \int_a^x \delta_1(s) ds/(k\epsilon), \end{array} \tag{6.2.6}$$

which implies that $\breve{v}$ satisfies

$$\begin{array}{c} \breve{v} = u_j + k\epsilon \partial_{xx} \breve{v} - k\partial_x (f(x, t_{j+1}, \breve{v})) + \zeta_j \\ \|\zeta_j\| \leq (1 + O(k))(\|\delta_1\| + \|\delta_2\|). \end{array} \tag{6.2.7}$$

6.3 Convergence for symmetric linear hyperbolic flux

We prove that backwards Euler converges in the 2-norm, given periodic boundary conditions and a dissipative flux. We assume $A(t) = I$, $C(t) = -I$, and that $f(x, t, u(x, t)) = F(x, t)u(x, t)$ where $F(x, t)$ is a symmetric matrix with distinct real eigenvalues. Using Eqn. (6.0.1), we obtain a numerical solution u_j satisfying Eqn. (6.2.3) at each time-step. We define the error

$$e_{j+1}(x) \quad = \quad y(x, t_{j+1}) - u_{j+1}(x), \tag{6.3.1}$$

and use Eqn. (6.1.3) and Eqn. (6.2.7) to form

$$\begin{array}{rcl} e_{j+1}(x) & = & e_j(x) + k\epsilon \partial_{xx} e_{j+1}(x) \\ & & -k\partial_x (f(x, t_{j+1}, y(x, t_{j+1})) - f(x, t_{j+1}, u_{j+1}(x))) \\ & & +\tau_j(x) + \zeta_j(x) \\ e_{j+1}(x) & = & e_j(x) + k\epsilon \partial_{xx} e_{j+1}(x) \\ & & -k\partial_x (F(x, t) e_{j+1}(x)) + \tau_j(x) + \zeta_j(x), \end{array} \tag{6.3.2}$$

along with boundary conditions

$$A(t_{j+1}) \cdot \begin{bmatrix} e(a) \\ \partial_x e(a) \end{bmatrix} + C(t_{j+1}) \cdot \begin{bmatrix} e(c) \\ \partial_x e(c) \end{bmatrix} = \begin{bmatrix} \delta_3 \\ \delta_4 \end{bmatrix}. \tag{6.3.3}$$

Let $e = e_{j+1}(x)$ and $d = e_j(x)$. We take inner products on the interval $[a, c]$ to get

$$e^{\mathrm{T}} e = d^{\mathrm{T}} e + k\epsilon \partial_{xx} e^{\mathrm{T}} e - k\partial_x (Fe)^{\mathrm{T}} e + \tau_j^{\mathrm{T}} e + \zeta_j^{\mathrm{T}} e. \tag{6.3.4}$$

Integration by parts yields

$$\begin{array}{rcl} \partial_{xx} e^{\mathrm{T}} e & = & -\partial_x e^{\mathrm{T}} \partial_x e + \partial_x e(a)^{\mathrm{T}} e(a) - \partial_x e(c)^{\mathrm{T}} e(c) \\ & = & -\|\partial_x e\|^2 + |\partial_x e(c) + e(c)| O(\delta_3 + \delta_4) \\ & \leq & |\partial_x e(c) + e(c)| O(\delta_3 + \delta_4) \end{array} \tag{6.3.5}$$

and

$$\begin{aligned}
\partial_x(Fe)^{\mathrm{T}}e &= -(Fe)^{\mathrm{T}}\partial_x e + (F(a)e(a))^{\mathrm{T}}e(a) - (F(c)e(c))^{\mathrm{T}}e(c) \\
&= -e^{\mathrm{T}}F\partial_x e + |F(c)||e(c)|O(\delta_3) \\
&= -e^{\mathrm{T}}\partial_x(Fe) + e^{\mathrm{T}}(\partial_x F)e + |F(c)||e(c)|O(\delta_3) \\
\partial_x(Fe)^{\mathrm{T}}e &= \tfrac{1}{2}e^{\mathrm{T}}(\partial_x F)e + |F(c)||e(c)|O(\delta_3) \\
&\leq \|\partial_x F\| \cdot \|e\|^2 + |F(c)||e(c)|O(\delta_3).
\end{aligned} \tag{6.3.6}$$

Plugging Eqn. (6.3.5) and Eqn. (6.3.6) into Eqn. (6.3.4) gives

$$\begin{aligned}
\|e\|^2 &\leq \|d\| \cdot \|e\| + k\|\partial_x F\| \cdot \|e\|^2 + (\|\tau_j\| + \|\zeta_j\|) \cdot \|e\| \\
&\quad + |\partial_x e(c) + (1 + |F(c)|)e(c)|O(\delta_3 + \delta_4) \\
\|e\| &\leq (1 + k\|\partial_x F\| + \|\tau_j\| + \|\zeta_j\| + O(k^2 + \tau_j^2 + \zeta_j^2)) \cdot \|d\| \\
&\quad + |\partial_x e(c) + (1 + |F(c)|)e(c)|O(\delta_3 + \delta_4).
\end{aligned} \tag{6.3.7}$$

If τ_j and ζ_j are both $O(k)$, and δ_3, δ_4 are small enough (and the solution and its derivatives are bounded at the endpoints) then we have convergence.

6.4 Convergence for linear strictly-hyperbolic flux

We prove a similar result for a linear strictly-hyperbolic (but not necessarily symmetric) flux with zero viscosity and periodic boundary conditions. Assume that $f = A(x,t)u(x,t)$, and that the eigenvalues of A are distinct and real. Let $A = VDV^{-1}$ be an eigenvalue decomposition. Assume that $\epsilon = 0$, and that solutions exist and are differentiable. Implicit Euler yields

$$\begin{aligned}
u_{j+1} &= u_j - k\partial_x(Au_{j+1}) \\
y_{j+1} &= y_j - k\partial_x(Ay_{j+1}) + \tau_j \\
e_{j+1} &= e_j - k\partial_x(Ae_{j+1}) + \tau_j.
\end{aligned} \tag{6.4.1}$$

We define

$$\begin{aligned}
e(x) &= e_{j+1}(x) \\
d(x) &= e_j(x) \\
g(x) &= V(x, t_{j+1})^{-1}e(x) \\
h(x) &= V(x, t_j)^{-1}d(x),
\end{aligned} \tag{6.4.2}$$

and rewrite Eqn. (6.4.1) as

$$\begin{aligned}
e &= d - k\partial_x(Ae) + \tau_j \\
V^{-1}e &= V^{-1}d - kV^{-1}\partial_x(Ae) + V^{-1}\tau_j \\
(V^{-1}e)^{\mathrm{T}}(V^{-1}e) &= (V^{-1}d - k\partial_x(V^{-1}Ae) \\
&\quad + k\partial_x(V^{-1})Ae + V^{-1}\tau_j)^{\mathrm{T}}(V^{-1}e).
\end{aligned} \tag{6.4.3}$$

We rewrite Eqn. (6.4.3) as

$$\begin{aligned} g^{\mathrm{T}}g &= d^{\mathrm{T}}V^{-\mathrm{T}}g - k\partial_x(Dg)^{\mathrm{T}}g \\ &\quad +g^{\mathrm{T}}(\partial_x(V^{-1})V^{-1}D)^{\mathrm{T}}g + \tau_j^{\mathrm{T}}(V^{-\mathrm{T}})g \\ \|g\|^2 &\leq \|h\| \cdot \|g\| \cdot \|V(x,t_j)^{\mathrm{T}}V(x,t_{j+1})^{-\mathrm{T}}\| \\ &\quad +kg^{\mathrm{T}}D(x,t_{j+1})\partial_x(g) \\ &\quad +\|g\|^2 k\|\partial_x(V(x,t_{j+1})^{-1})V(x,t_{j+1})^{-1}D(x,t_{j+1})\| \\ &\quad +\|\tau_j\| \cdot \|g\| \cdot \|V(x,t_{j+1})^{-\mathrm{T}}\| \\ \|g\| &\leq \|h\| \cdot \|V(x,t_j)^{\mathrm{T}}V(x,t_{j+1})^{-\mathrm{T}}\| \\ &\quad +k\|g\|(\|\partial_x(D(x,t_{j+1}))\| + \|\partial_x(V(x,t_{j+1})^{-1})V(x,t_{j+1})^{-1}D(x,t_{j+1})\|) \\ &\quad +\|\tau_j\| \cdot \|V(x,t_{j+1})^{-\mathrm{T}}\|. \end{aligned} \tag{6.4.4}$$

So assuming the appropriate bounds on the norms of $V, D, \partial_x V, \partial_x D$, we have

$$\|g\| \leq \|h\|(1 + O(k)) + O(\tau). \tag{6.4.5}$$

We combine this with the inequality

$$\|e\| \leq \|g\| \cdot \|V(x,t_{j+1})\| \tag{6.4.6}$$

to get convergence (assuming a perfect boundary value problem solver).

6.5 Convergence for one dimensional conservative flux

Here we prove a form of convergence for the 1-norm. We show that the implicit Euler scheme has a conservative numerical flux and produces numerical solutions which are total variation diminishing (TVD). This implies that the numerical solutions converge, and we show that the limit of the numerical solutions is a solution of the original PDE. For the remainder of this section, we assume that we are given a one dimensional conservative flux of the following form:

$$\partial_t u(x,t) + \partial_x(f(u(x,t))) = \epsilon\partial_{xx}u(x,t), \tag{6.5.1}$$

with periodic boundary conditions

$$u(a,t) = u(c,t) \quad \partial_x u(a,t) = \partial_x u(c,t). \tag{6.5.2}$$

We define a flux function

$$F(u(x,t)) = f(u(x,t)) - \epsilon\partial_x u(x,t), \tag{6.5.3}$$

such that

$$\partial_t u(x,t) + \partial_x F(u(x,t)) = 0. \tag{6.5.4}$$

We write the conservation law 6.5.1 in integral (conservation) form:

$$\begin{array}{c} \int_{x_1}^{x_2} u(x,t_{j+1})dx = \int_{x_1}^{x_2} u(x,t_j)dx + \\ \int_{t_j}^{t_{j+1}} F(u(x_1,t))dt - \int_{t_j}^{t_{j+1}} F(u(x_2,t))dt. \end{array} \tag{6.5.5}$$

Applying the implicit Euler scheme

$$\begin{array}{rcl} u_{j+1} & = & u_j + k\epsilon\partial_{xx}u_{j+1} - k\partial_x(f(u_{j+1})) \\ & = & u_j + k\partial_x F(u_{j+1}) \end{array} \tag{6.5.6}$$

to Eqn. (6.5.5) gives

$$\begin{array}{rcl} \int_{x_1}^{x_2} u_{j+1}(x)dx & = & \int_{x_1}^{x_2} u_j(x)dx \\ & & +k\int_{x_1}^{x_2} \partial_x F(u_{j+1})dx \\ \int_{x_1}^{x_2} u_{j+1}(x)dx & = & \int_{x_1}^{x_2} u_j(x)dx \\ & & +k(F(u_{j+1}(x_1))) - k(F(u_{j+1}(x_2))). \end{array} \tag{6.5.7}$$

Note that the flux term

$$\int_{t_j}^{t_{j+1}} (f(u(x,t)) - \epsilon\partial_x u(x,t))dt = \int_{t_j}^{t_{j+1}} F(u(x,t)) \tag{6.5.8}$$

is approximated by the numerical flux term

$$k(f(u_{j+1}(x)) - \epsilon\partial_x u_{j+1}(x)) = kF(u_{j+1}). \tag{6.5.9}$$

Eqn. (6.5.9) reduces to the actual flux in the case of constant flow, and is a lipshitz function of u. Therefore the numerical flux is consistent and the numerical solution satisfies a discrete conservation law:

$$\begin{array}{rcl} \int_{x_1}^{x_2} u_{j+1}(x)dx & = & \int_{x_1}^{x_2} u_j(x)dx \\ & & +kF(u_{j+1}(x_1)) - kF(u_{j+1}(x_2)). \end{array} \tag{6.5.10}$$

With periodic boundary contitions the total integral $\int_a^c u_{j+1}$ is equal to the total integral $\int_a^c u_j$ at the previous step, and the total density is conserved.

A version of the lax-wendroff theorem holds in this situation: if Euler's method converges and certain conditions are satisfied, then the limit function is a solution of Eqn. (6.5.1). We make

this precise by specifying the notion of convergence. Assume we are given a sequence of time grids indexed by l. For each fixed l we have a set of time values $\{t_0^l, t_1^l, \ldots, t_j^l, t_{j+1}^l, \ldots, t_{N^l}^l\}$. We let $k_j^l = t_{j+1}^l - t_j^l$ be the j-th time-step of the l-th grid, and $k^l = \sup_j k_j^l$ be the upper bound for the time-steps of the l-th grid. We assume $k^l \to 0$ as $l \to \infty$. Implicit Euler yields:

$$u_{j+1}^l(x) = u_j^l(x) - k_{j+1}^l \partial_x F(u_{j+1}^l). \tag{6.5.11}$$

Let $u^l(x,t)$ be a function that is piecewise constant in time such that $u^l(x, t_j^l) = u_j^l(x)$. The sequence of solutions $u^l(x,t)$ converges to a limit $u(x,t)$ if

$$\int_0^T \int_a^c |u^l(x,t) - u(x,t)| dx dt \to 0 \text{ as } l \to \infty. \tag{6.5.12}$$

Now we prove that the limit solution $u(x,t)$ is a solution of the original conservation law.

Theorem 6.5.1 *Assume we apply the implicit Euler scheme of Eqn. (6.5.11) and obtain a sequence of numerical solutions $u^l(x,t)$. If these solutions converge to a function $u(x,t)$ as $l \to 0$ (in the sense of Eqn. (6.5.12)), then $u(x,t)$ is a solution of the original conservation law (Eqn. (6.5.1)).*

Proof. We show that $u(x,t)$ is a weak solution to the conservation law. Given any smooth test function $\phi(x,t)$ which satisfies the periodic boundary conditions (smooth across the boundary), and is supported compactly within $[0,T)$, then:

$$\begin{gathered} \int_0^T \int_a^c (u\partial_t \phi + F(u)\partial_x \phi) dx dt = \\ \int_a^c [u(x,t)\phi(x,t)]_{t=0}^{t=T} + \int_0^T [F(u)\phi(x,t)]_{x=a}^{x=c} dt. \end{gathered} \tag{6.5.13}$$

The periodic boundary conditions cancel out the spatial boundary terms, and $\phi(x,T) = 0$, leaving

$$\begin{gathered} \int_0^T \int_a^c (u\partial_t \phi + F(u)\partial_x \phi) dx dt = \\ - \int_a^c u(x,0)\phi(x,0) dx. \end{gathered} \tag{6.5.14}$$

We cancel the boundary terms to get

$$\begin{gathered} \int_0^T \int_a^c (u\partial_t \phi + f(u)\partial_x \phi + \epsilon u \partial_{xx} \phi) dx dt \\ = - \int_a^c u(x,0)\phi(x,0) dx. \end{gathered} \tag{6.5.15}$$

We multiply the implicit Euler method (Eqn. (6.5.11)) by the test function ϕ

$$\begin{gathered} u_{j+1}^l(x)\phi(x, t_{j+1}^l) = \\ \phi(x, t_{j+1}^l) u_j^l(x) - \phi(x, t_{j+1}^l) k_{j+1}^l \partial_x F(u_{j+1}^l(x)), \end{gathered} \tag{6.5.16}$$

and integrate with respect to x and sum with respect to j

$$\begin{array}{c} \int_a^c \sum_{j=0}^{N^l-1} \phi(x,t_{j+1}^l)(u_{j+1}^l(x)-u_j^l(x))dx = \\ -\sum_{j=0}^{N^l-1} \int_a^c \phi(x,t_{j+1}^l)k_{j+1}^l \partial_x F(u_{j+1}^l(x)). \end{array} \tag{6.5.17}$$

The summation by parts formula:

$$\sum_{j=0}^{N-1} a_{j+1}(b_{j+1}-b_j) = a_N b_N - a_1 b_0 - \sum_{j=1}^{N-1}(a_{j+1}-a_j)b_j, \tag{6.5.18}$$

as well as integration by parts yields

$$\begin{array}{c} \int_a^c \phi(x,T)u_{N^l}^l(x)dx - \int_a^c \phi(x,0)u_0^l(x)dx - \\ \int_a^c \sum_{j=1}^{N^l-1}(\phi(x,t_{j+1}^l)-\phi(x,t_j^l))u_j^l(x)dx = \\ \sum_{j=0}^{N^l-1} k_{j+1}^l(\phi(a,t_{j+1}^l)F(u_{j+1}^l(a)) - \phi(c,t_{j+1}^l)F(u_{j+1}^l(c))) \\ -\sum_{j=0}^{N^l-1}\int_a^c k_{j+1}^l F(u_{j+1}^l(x))\partial_x\phi(x,t_{j+1}^l)dx. \end{array} \tag{6.5.19}$$

Due to periodic boundary conditions and the compact support of ϕ, many spatial boundary terms cancel out, and we are left with

$$\begin{array}{c} \int_a^c \phi(x,0)u_0^l(x)dx + \\ \int_a^c \sum_{j=1}^{N^l-1} k_{j+1}^l \frac{\phi(x,t_{j+1}^l)-\phi(x,t_j^l)}{k_{j+1}^l} u_j^l(x)dx = \\ -\sum_{j=0}^{N^l-1}\int_a^c k_{j+1}^l(\epsilon u_{j+1}^l(x)\partial_{xx}\phi(x,t_{j+1}^l) + f(u_{j+1}^l(x))\partial_x\phi(x,t_{j+1}^l))dx. \end{array} \tag{6.5.20}$$

Since each grid starts with the same initial data, the first line of Eqn. (6.5.20) is

$$\int_a^c \phi(x,0)u_0^l(x)dx = \int_a^c \phi(x,0)u(x,0)dx. \tag{6.5.21}$$

Because u^l converges to u in the 1-norm, and ϕ is smooth, we have

$$\begin{array}{c} \lim_{l\to\infty} \int_a^c \sum_{j=1}^{N^l-1} k_{j+1}^l \frac{\phi(x,t_{j+1}^l)-\phi(x,t_j^l)}{k_{j+1}^l} u_j^l(x)dx \\ = \int_a^c \int_0^T u(x,t)\partial_t\phi(x,t)dxdt, \end{array} \tag{6.5.22}$$

and

$$\begin{array}{c} \lim_{l\to\infty} \int_a^c \sum_{j=0}^{N^l-1} k_{j+1}^l(\epsilon u_{j+1}^l(x)\partial_{xx}\phi(x,t_{j+1}^l) + f(u_{j+1}^l(x))\partial_x\phi(x,t_{j+1}^l))dx = \\ \int_0^T \int_a^c (\epsilon u(x,t)\partial_{xx}\phi(x,t) + f(u(x,t))\partial_x\phi(x,t))dxdt. \end{array} \tag{6.5.23}$$

By combining these results we see that Eqn. (6.5.15) holds for all appropriate test functions ϕ. Since the original conservation law has a unique smooth solution, each weak solution is actually the (strong) solution, and therefore $u(x,t)$ is the (strong) solution to the original conservation law. □

Now we prove that the u^l do indeed converge to some u in the 1-norm. The basic idea is to show that the u^l all lie in a compact subset of $L_1([a,c]\times[0,T])$. The total variation of a function $v(x):[a,c]\to R$ is

$$TV(v)=\limsup_{\epsilon\to 0}\frac{1}{\epsilon}\int_a^c |v(x+\epsilon)-v(x)|dx. \tag{6.5.24}$$

For differentiable functions this is equivalent to

$$TV(v)=\int_a^c |\partial_x v(x)|dx. \tag{6.5.25}$$

We extend this definition to a function $v(x,t):[a,c]\times[0,T]\to R$ as follows

$$\begin{aligned} TV(v)=&\limsup_{\epsilon\to 0}\tfrac{1}{\epsilon}\int_0^T\int_a^c |v(x+\epsilon,t)-v(x,t)|dxdt\\ &+\limsup_{\epsilon\to 0}\tfrac{1}{\epsilon}\int_0^T\int_a^c |v(x,t+\epsilon)-v(x,t)|dxdt. \end{aligned} \tag{6.5.26}$$

For the time-wise piecewise constant numerical solution $u^l(x,t)$, this reduces to:

$$TV(u^l(x,t))=\textstyle\sum_j\left[k_j^l TV(u_j^l(x))+\|u_{j+1}^l-u_j^l\|\right]. \tag{6.5.27}$$

The set

$$TVB(R)=\{v\in L_1([a,c]\times[0,T]):TV(v)\le R\} \tag{6.5.28}$$

is compact in $L_1([a,c]\times[0,T])$ (for each R), and hence any sequence of functions with uniformly bounded total variation must contain a subsequence which converges in the 1-norm.

A method is TV-stable if it produces a sequence of numerical solutions u^l with $k^l\le\bar{k}$ that lie within $TVB(R)$ for some R that depends only on $\bar{k}$, the conservation law, and the initial conditions. We will show that implicit Euler is TV-stable.

The first step is to show that the solutions have uniformly bounded total variation in space. That is, we show that $TV(u_j^l(x))<R$ for all l and j. Recall that we have periodic boundary conditions and a conservative flux $\partial_x(f(u(x,t))-\epsilon\partial_x u(x,t))$. Implicit Euler yields

$$u_{j+1}^l=u_j^l-k_{j+1}^l\partial_x(f(u_{j+1}^l)-\epsilon\partial_x u_{j+1}^l). \tag{6.5.29}$$

Differentiating this expression yields:

$$\partial_x u^l_{j+1} = \partial_x u^l_j - k^l_{j+1}\partial_{xx}(f(u^l_{j+1}) - \epsilon\partial_x u^l_{j+1}). \tag{6.5.30}$$

For simplicity we drop the l and j indices and let $w = u^l_{j+1}$ and $v = u^l_j$. We are interested in the quantity $TV(w)$. Since $w(x)$ is smooth, we use Eqn. (6.5.25). To compute this integral, we consider the zeros of $\partial_x w(x)$. Let $z_1 < z_2 < \ldots < z_K$ be the roots of $\partial_x w(x)$. In between these zeros, $\partial_x w(x)$ has definite sign (always positive or always negative). Therefore

$$\int_{z_k}^{z_{k+1}} |\partial_x w(x)|dx = \begin{cases} \int_{z_k}^{z_{k+1}} \partial_x w(x)dx & \partial_x w(x) > 0 \quad \forall x \in [z_k, z_{k+1}] \\ -\int_{z_k}^{z_{k+1}} \partial_x w(x)dx & \partial_x w(x) < 0 \quad \forall x \in [z_k, z_{k+1}]. \end{cases} \tag{6.5.31}$$

If $\partial_x w(x)$ is positive between the two roots, then

$$\begin{aligned} \int_{z_k}^{z_{k+1}} |\partial_x w(x)|dx &= \int_{z_k}^{z_{k+1}} \partial_x w(x)dx \\ &= \int_{z_k}^{z_{k+1}} \left[\partial_x v(x) + k^l_{j+1}\partial_{xx}(\epsilon\partial_x w(x) - f(w(x)))\right] dx \\ &= \int_{z_k}^{z_{k+1}} \partial_x v(x)dx + \left[\epsilon k^l_{j+1}\partial_{xx} w(x)\right]_{x=z_k}^{x=z_{k+1}} \\ &\quad - \left[k^l_{j+1}\partial_x f(w(x))\right]_{x=z_k}^{x=z_{k+1}}. \end{aligned} \tag{6.5.32}$$

Since $\partial_x w$ is positive in the interval and zero at the the endpoints,

$$\partial_{xx} w(z_{k+1}) \leq 0 \leq \partial_{xx} w(z_k) \tag{6.5.33}$$

and so

$$\left[\epsilon k^l_{j+1}\partial_{xx} w(x)\right]_{x=z_k}^{x=z_{k+1}} \leq 0. \tag{6.5.34}$$

We also have that

$$\partial_x f(w(z_k)) = \partial_w f(w(z_k))\partial_x w(z_k) = 0 \tag{6.5.35}$$

and

$$\partial_x f(w(z_{k+1})) = \partial_w f(w(z_{k+1}))\partial_x w(z_{k+1}) = 0. \tag{6.5.36}$$

Therefore

$$\left[k^l_{j+1}\partial_x f(w(x))\right]_{x=z_k}^{x=z_{k+1}} = 0 \tag{6.5.37}$$

and

$$\int_{z_k}^{z_{k+1}} |\partial_x w(x)|dx \leq \int_{z_k}^{z_{k+1}} \partial_x v(x)dx \leq \int_{z_k}^{z_{k+1}} |\partial_x v(x)|dx. \tag{6.5.38}$$

The case where $\partial_x w(x)$ is negative is similar, and we obtain

$$\int_{z_k}^{z_{k+1}} |\partial_x w(x)|dx \leq -\int_{z_k}^{z_{k+1}} \partial_x v(x)dx \leq \int_{z_k}^{z_{k+1}} |\partial_x v(x)|dx. \tag{6.5.39}$$

We use the periodic boundary conditions and express the entire integral as a sum:

$$\int_a^c |\partial_x w(x)|dx = \left[\sum_{k=1}^{K} \int_{z_k}^{z_{k+1}} |\partial_x w(x)|dx\right], \tag{6.5.40}$$

where $z_{K+1} = z_1$, and the final integral stretches over the boundary point. Applying Eqn. (6.5.38) and Eqn. (6.5.39) to Eqn. (6.5.40) yields

$$\int_a^c |\partial_x u_{j+1}^l(x)|dx \leq \int_a^c |\partial_x u_j^l(x)|dx, \tag{6.5.41}$$

which implies that

$$TV(u_j^l(x)) \leq TV(u(x,0)) = R. \tag{6.5.42}$$

Thus the functions $u^l(x)$ have uniformly (in l) bounded total variation in space, and we can easily bound the first part of Eqn. (6.5.27).

To bound the second part of Eqn. (6.5.27) we prove that the difference $|u_{j+1}^l - u_j^l|$ is small in the 1-norm. We need to show that

$$\|u_{j+1}^l(x) - u_j^l(x)\| = O(k_{j+1}^l). \tag{6.5.43}$$

Implicit Euler yields:

$$\begin{array}{lcl} w - v & = & k_{j+1}^l \partial_x(\epsilon\partial_x w - f(w)) \\ \int_a^c |w - v|dx & = & k_{j+1}^l \int_a^c |\partial_x(\epsilon\partial_x w - f(w))|dx \\ \int_a^c |w - v|dx & = & k_{j+1}^l TV(\epsilon\partial_x w - f(w)), \end{array} \tag{6.5.44}$$

where, as before, $w = u_{j+1}^l$ and $v = u_j^l$. Since the u_j^l satisfy a discrete conservation law (Eqn. (6.5.10)), $\int_a^c w = \int_a^c v$. Thus $(w - v)$ cannot be everywhere positive, or everywhere negative. So

$(w-v)$ must be positive in some regions, negative in other regions, and zero on the boundaries. Let $z_1, z_2, \ldots, z_K$ be the zeros of $|w-v|$ which are on the boundary between regions of positive $(w-v)$ and negative $(w-v)$. These z_k will also be zeros of $|\partial_x(\epsilon\partial_x w - f(w))|$. If $(w-v)$ is positive in between z_k and z_{k+1}, then $(w-v)$ is negative in between z_{k-1} and z_k as well as in between z_{k+1} and z_{k+2}, and so

$$\begin{aligned} \partial_x v(z_k) &\leq \partial_x w(z_k) \\ \partial_x v(z_{k+1}) &\geq \partial_x w(z_{k+1}). \end{aligned} \tag{6.5.45}$$

Since ϵ is positive this implies

$$\begin{aligned} \epsilon\partial_x v(z_k) - f(v(z_k)) &\leq \epsilon\partial_x w(z_k) - f(w(z_k)) \\ \epsilon\partial_x v(z_{k+1}) - f(v(z_{k+1})) &\geq \epsilon\partial_x w(z_{k+1}) - f(w(z_{k+1})). \end{aligned} \tag{6.5.46}$$

Therefore

$$\begin{aligned} \int_{z_k}^{z_{k+1}} |w-v| &= \int_{z_k}^{z_{k+1}} w - v \\ &= \int_{z_k}^{z_{k+1}} k_{j+1}^l \partial_x(\epsilon\partial_x w - f(w)) \\ &= k_{j+1}^l [\epsilon\partial_x w - f(w)]_{z_k}^{z_{k+1}} \\ &\leq k_{j+1}^l [\epsilon\partial_x v - f(v)]_{z_k}^{z_{k+1}} \\ &\leq (k_{j+1}^l / k_j^l) \int_{z_k}^{z_{k+1}} |v - u_{j-1}^l|. \end{aligned} \tag{6.5.47}$$

Similarly, if $(w-v)$ is negative in between z_k and z_{k+1} then we have

$$\begin{aligned} \int_{z_k}^{z_{k+1}} |w-v| &= -\int_{z_k}^{z_{k+1}} w - v \\ &= -\int_{z_k}^{z_{k+1}} k_{j+1}^l \partial_x(\epsilon\partial_x w - f(w)) \\ &= -k_{j+1}^l [\epsilon\partial_x w - f(w)]_{z_k}^{z_{k+1}} \\ &\leq -k_{j+1}^l [\epsilon\partial_x v - f(v)]_{z_k}^{z_{k+1}} \\ &\leq (k_{j+1}^l / k_j^l) \int_{z_k}^{z_{k+1}} |v - u_{j-1}^l|. \end{aligned} \tag{6.5.48}$$

Therefore

$$\begin{aligned} \int_a^c |w-v| &= \sum_{k=1}^{k=K} \int_{z_k}^{z_{k+1}} |w-v| \\ &\leq (k_{j+1}^l / k_j^l) \sum_{k=1}^{k=K} \int_{z_k}^{z_{k+1}} |v - u_{j-1}^l| \\ &\leq (k_{j+1}^l / k_j^l) \int_a^c \int_{z_k}^{z_{k+1}} |v - u_{j-1}^l|, \end{aligned} \tag{6.5.49}$$

where we let $z_{K+1} = z_1$. So if $\Pi_j (k_{j+1}^l / k_j^l)$ is bounded, then

$$\int_a^c |w-v| \leq R k_1^l \int_a^c |\partial_x(\epsilon\partial_x u_0 - f(u_0))| \leq k_j^l R. \tag{6.5.50}$$

Using the estimates of Eqn. (6.5.50) and Eqn. (6.5.42) in Eqn. (6.5.27) yields

$$\begin{aligned} TV(u^l(x,t)) &= \sum_j \left[k_j^l TV(u_j^l(x)) + \|u_{j+1}^l - u_j^l\|\right] \\ &\leq \sum_j \left[k_j^l R + k_j^l R\right] \\ &\leq R. \end{aligned} \tag{6.5.51}$$

Hence implicit Euler is TV-stable, and all the $u^l(x,t)$ belong to a compact set of $L_1([a,c]\times[0,T])$. Therefore, this set has convergent subsequences, and by Theorem 6.5.1 we conclude that the limit of each of these convergent subsequences is the solution to the original conservation law.

Chapter 7

Algorithm Description

We describe in more detail our algorithm for solving initial-boundary problems. As stated earlier in Section 5.1, we first discretize time and reduce the PDE to a sequence of (inhomogeneous) two-point boundary value problems. Then we use the coordinate transform of Chapter 4 to make each boundary value problem homogeneous. We solve all these boundary value problems with the method of Section 2.2. After forming an initial numerical solution, we use the formalism of Section 3.2 to construct an initial-boundary problem for the error and solve it the same way. We repeat this correction process until we are satisfied with our solution. Then we move on, using our previously calculated (and corrected) solution as initial values for the next time interval.

Algorithm 7.0.1 *Adaptive PDE Solver*

1. *Define user input:*
   - *Set up the PDE right hand side* F, H, J, M.
   - *Set up the boundary conditions* A, C, γ.
   - *Set up the initial conditions* $u_0, \partial_x u_0, \partial_{xx} u_0, T_0$.
   - *Choose a set of basis functions to use (ie, Chebyshev Polynomials) and a degree* q *for the corresponding quadrature scheme.*
   - *Choose a tolerance* ϵ *and initial time-step* ΔT.

2. *Construct the q quadrature nodes $\{t_1, \ldots, t_q\}$ corresponding to the interval $[T_0, T_0 + \Delta T]$.*

3. *Obtain an initial estimate for the solution:*

   for $j = 1, 2, \ldots, q$

   Adaptively solve for u_j at time t_j by applying the coordinate change of Chapter 4 and Algorithm 2.2.2 to Eqn. (5.1.8) and Eqn. (5.1.9). (u_0 at time T_0 is given).

   end for

4. *Set $u_j^{[0]} = u_j$ and $\mu = 0$.*

5. *Calculate $R_j^{[\mu]}$ for each t_j (using Algorithm 7.0.2).*

6. *Test to see if a correction is necessary:*

   - *If $|R^{[\mu]}| < \epsilon$ then our current solution is accurate enough. Jump forward to step (9).*
   - *Otherwise if $|R^{[\mu]}| > |R^{[\mu-1]}|$ assume the corrections are not converging. Go back to step (2) and halve ΔT.*
   - *Otherwise continue with a correction step:*

     for $j = 1, 2, \ldots, q$

     Adaptively solve for $e_j^{[\mu]}$ at time t_j by applying the coordinate change of Chapter 4 and Algorithm 2.2.2 to Eqn. (3.2.13) and Eqn. (3.2.9). ($e_0^{[\mu]}$ at time T_0 is zero).

     end for

7. *Set $u_j^{[\mu+1]} = u_j^{[\mu]} + e_j^{[\mu]}$ (using Algorithm 7.0.3).*

8. *Test to see if the solution is accurate:*

- *If $\mu < \mu_{max}$, set $\mu = \mu + 1$ and go back to step (5).*
- *Otherwise if $\mu = \mu_{max}$, set $\mu = \mu + 1$ and calculate $R_j^{[\mu]}$. If $|R^{[\mu]}| > \epsilon$ go back to step (2) and halve ΔT.*

9. *Accept $u_j^{[\mu]}$ as the correct solution on $[T_0, t_q]$. Move on to a new time interval $[t_q, t_q + \Delta T]$ and repeat the process.*

10. *If the previous few time intervals were resolved without halving ΔT, double ΔT for the next time interval.*

While conceptually clean, this algorithm has a few problematic implementation details. Every boundary value problem is solved adaptively and each $u_j^{[\mu]}$ is stored on a different tree of subintervals. In order to calculate the residual we need to integrate across different subinterval trees. This requires the following algorithm, which is used during during step (5) of Algorithm 7.0.1.

Algorithm 7.0.2 *Calculation of the Residual*

1. *After computing $u_j^{[\mu]}$ for times t_j, collect and store each corresponding subinterval tree $\mathcal{B}_j$.*
2. *Form a tree $\mathcal{B}$ which is the union of all the $\mathcal{B}_j$:*
   - *Initialize $\mathcal{B}$ to the interval $[a, c]$.*
   - *Recursively examine each deepest subinterval of $\mathcal{B}$, and if that subinterval is split in any of the $\mathcal{B}_j$, split that subinterval in $\mathcal{B}$.*
3. *$\mathcal{B}$ will be finer than every $\mathcal{B}_j$.*
4. *Use the values of $u_j^{[\mu]}$ on $\mathcal{B}_j$ to construct values of $u_j^{[\mu]}$ on the deepest subintervals of $\mathcal{B}$ for each j.*
5. *Use the values of $u_j^{[\mu]}$ on the deepest subintervals of $\mathcal{B}$ to compute $R_j^{[\mu]}$ on the deepest subintervals of $\mathcal{B}$ for each j.*

6. *Store* $R_j^{[\mu]}$ *on* $\mathcal{B}$.

We also need to merge $e_j^{[\mu]}$ and $u_j^{[\mu]}$ into $u_j^{[\mu+1]}$ since each correction $e_j^{[\mu]}$ is stored on a separate tree. The following algorithm is used during step (7) of Algorithm 7.0.1.

Algorithm 7.0.3 *Calculation of the Corrected Solution*

1. *After computing* $u_j^{[\mu]}$ *and* $e_j^{[\mu]}$ *for a particular* t_j, *collect and store the corresponding subinterval trees* $\mathcal{B}_u$ *and* $\mathcal{B}_e$.

2. *Form a tree* $\mathcal{B}$ *which is the union of* $\mathcal{B}_u$ *and* $\mathcal{B}_e$:

   - *Initialize* $\mathcal{B}$ *to the interval* $[a, c]$.
   - *Recursively examine each deepest subinterval of* $\mathcal{B}$, *and if that subinterval is split in either* $\mathcal{B}_u$ *or* $\mathcal{B}_e$, *split that subinterval in* $\mathcal{B}$.

3. $\mathcal{B}$ *will be finer than* $\mathcal{B}_u$ *and* $\mathcal{B}_e$.

4. *Use the values of* $u_j^{[\mu]}$ *and* $e_j^{[\mu]}$ *on* $\mathcal{B}_u$ *and* $\mathcal{B}_e$ *(respectively) to construct values of* $u_j^{[\mu]}$ *and* $e_j^{[\mu]}$ *on the deepest subintervals of* $\mathcal{B}$.

5. *Use the values of* $u_j^{[\mu]}$ *and* $e_j^{[\mu]}$ *on the deepest subintervals of* $\mathcal{B}$ *to compute* $u_j^{[\mu+1]}$ *on the deepest subintervals of* $\mathcal{B}$.

6. *Store* $u_j^{[\mu+1]}$ *on* $\mathcal{B}$.

Chapter 8

Numerical Examples

We measure the performance of Algorithm 3.3.1 and Algorithm 7.0.1 on several challenging DAEs and PDEs. Many of our examples are chosen so that the solutions are nearly discontinuous. We use the function

$$S(l,h,w,x) = 0.5h\left(1+\text{erf}\left(4\frac{x-l}{w}\right)\right) \tag{8.0.1}$$

to model a shock at location l with height h and smoothing over length scale w. We use the function

$$R(l,h,W,w,x) = \begin{cases} \frac{h}{W}(x-l+\frac{W}{2})(S(l-\frac{W}{2},1,w,x)-S(l+\frac{W}{2},1,w,x)) & W>0 \\ \qquad +S(l+\frac{W}{2},h,w,x) & \\ S(l,h,w,x) & W=0 \end{cases} \tag{8.0.2}$$

to model a rarefaction wave centered at location l with height h width W and smoothing over length scale w. Examples of these functions are plotted in Figure (8.0.1). Each numerical example is programmed in C and tested on a Toshiba 2455 running Linux with a 2.3 GHz Pentium 4 processor. BLAS libraries are not used.

8.1 Differential algebraic equations

We apply the spectral deferred correction algorithm (Algorithm 3.3.1) to DAEs of the form (3.3.1) and (3.3.11). The same code is used for index-1 and index-2 DAEs. The user needs to supply the following:

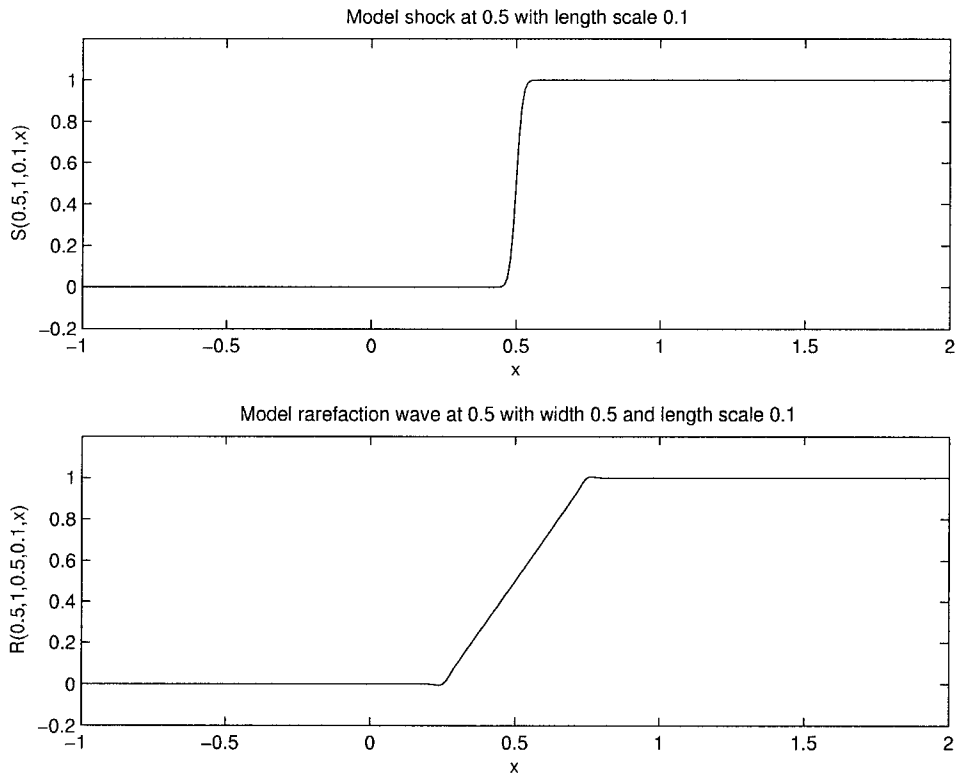

Figure 8.0.1: Model shock and rarefaction wave

1	1
	1

$\frac{1}{3}$	$\frac{5}{12}$	$\frac{-1}{12}$
1	$\frac{3}{4}$	$\frac{1}{4}$
	$\frac{3}{4}$	$\frac{1}{4}$

Table 8.1.1: Radau IIA methods of order 1 and 3

1. Functions $f, \partial_y f, \partial_z f, g, \partial_y g, \partial_z g$ which can be called for arbitrary t, y, z.

2. Initial conditions y_0, z_0.

3. An FSAL Runge-Kutta method which will be used for each correction step.

4. Control parameters ϵ and $\mu_{\max}$.

5. Initial time T_0, initial time-step ΔT and final time T_F.

To simplify the matrix manipulations our code requires that vectors y and z be of the same dimension. We restrict ourselves to the low order FSAL Radau IIA methods shown in Table (8.1.1) [15]. The first method is merely implicit Euler, and the second method is of order 3. With an FSAL Runge-Kutta method of order p, the order of the algorithm should be no more than $p\mu_{\max}$. Hence we choose the order q of the quadrature scheme to be the smallest power of 2 greater than $q\mu_{\max}$. We want to determine if the magnitude of the residual is an accurate indicator of the error, and if decreasing the tolerance ϵ corresponds to a decrease in the error. We also check if the algorithm is performing multiple corrections per step (resulting in a high-order scheme).

Example 8.1.1 *The index-1 DAE*

$$\begin{aligned} y_1' &= -ty_2 - (1+t)z_1 \\ y_2' &= ty_1 - (1+t)z_2 \\ 0 &= (y_1 - z_2)/5 - \cos(t^2/2) \\ 0 &= (y_2 + z_1)/5 - \sin(t^2/2) \end{aligned} \tag{8.1.1}$$

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Error	Total # of Steps	Total Time (s)
$1e-2$	8	4.2	$1e-1$	86	16
$1e-4$	8	5.3	$3e-4$	141	42
$1e-6$	8	7.3	$4e-6$	221	79
$1e-8$	8	8.0	$9e-8$	375	136

Table 8.1.2: Solution of Example (8.1.1) using Implicit Euler

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Error	Total # of Steps	Total Time (s)
$1e-2$	16	1.0	$3e-3$	23	35
$1e-4$	16	8.1	$5e-5$	31	159
$1e-6$	16	10.7	$9e-7$	41	277
$1e-8$	16	12.1	$3e-9$	51	376

Table 8.1.3: Solution of Example (8.1.1) using Order 3 Radau IIA

with initial conditions

$$\begin{aligned} y_1(0) &= 5 \\ y_2(0) &= 1 \\ z_1(0) &= -1 \\ z_2(0) &= 0 \end{aligned} \tag{8.1.2}$$

Has the exact solution

$$\begin{aligned} y_1(t) &= \sin(t) + 5\cos(t^2/2) \\ y_2(t) &= \cos(t) + 5\sin(t^2/2) \\ z_1(t) &= -\cos(t) \\ z_2(t) &= \sin(t). \end{aligned} \tag{8.1.3}$$

Tables (8.1.2) and (8.1.3) contain the results of several trials with different ϵ and $\mu_{\max}$ on $0 \leq t \leq 4\pi$ with an initial time-step of $\Delta T = 0.1\pi$. The first set of trials are performed using implicit Euler for each correction step. The second set of trials are performed using the order 3 Radau IIA method for each correction step. Listed are the average number of corrections, average error, number of time-steps and total time for each trial. As the tolerance decreases the order of the algorithm increases and the error decreases.

Example 8.1.2 *The following index-1 DAE is used in [15] and [35] to model a two-transistor am-*

plifier with a forcing term:

$$\begin{aligned}
V_1' &= \tfrac{1000}{9}(9U_e - 9V_1 - 9V_6) + F_1 \\
V_2' &= \tfrac{1000}{18}(9000f(V_6 - V_2) - V_2) + F_2 \\
V_3' &= \tfrac{1000}{27}(6 - 8910f(V_6 - V_2) - V_3 - V_7) + F_3 \\
V_4' &= \tfrac{1000}{36}(9000f(V_7 - V_4)) + F_4 \\
V_5' &= \tfrac{1000}{45}(6 - 8910f(V_7 - V_4) - V_5 - V_8) + F_5 \\
0 &= 9V_1 + 11V_6 + 90f(V_6 - V_2) - 9U_e - 6 + F_6 \\
0 &= 8910f(V_6 - V_2) + V_3 + 3V_7 + 90f(V_7 - V_4) - 12 + F_7 \\
0 &= 8910f(V_7 - V_4) + V_5 + 2V_8 - 6 + F_8
\end{aligned} \tag{8.1.4}$$

where

$$f(V) = 10^{-6}(\exp(38.4615V) - 1) \tag{8.1.5}$$

and the initial signal is chosen as

$$U_e(t) = \begin{cases} \sin(200\pi t)/10 & t > 0 \\ 0 & t < 0. \end{cases} \tag{8.1.6}$$

The initial conditions $V_j(0)$ and forcing terms F_k are chosen so that the exact solution is:

$$\begin{aligned}
V_1 &= 3.5e^{-100t} - 3 + U_e(t - 0.0125) \\
V_2 &= 2.85e^{-100t} + U_e(t - 0.0125) \\
V_3 &= 6e^{-100t} + U_e(t - 0.0125) \\
V_4 &= 2 + \textstyle\sum_{j=1}^{\infty} S(0.0125 + 0.01j, 0.25, 0.001, t) \\
V_5 &= 6 - \textstyle\sum_{j=1}^{\infty} S(0.0125 + 0.01j, 0.1, 0.002, t) \\
V_6 &= 3e^{-100t} + U_e(t - 0.0125) \\
V_7 &= 2 - 2000te^{-1000t} - 100te^{-100t} - 25te^{-50t} + 5U_e(t - 0.0125) \\
V_8 &= 0.1 - 3.61\exp(-(10U_e(t - 0.0125))^2/0.27884) + 0.3t
\end{aligned} \tag{8.1.7}$$

with S given by Eqn. (8.0.1).

The physical quantities of interest are the gate potentials V_4 and V_5 and exit voltage V_8. These voltages undergo perturbations at the same frequency as U_e, and are plotted in Figures (8.1.1,8.1.2). Tables (8.1.4) and (8.1.5) contain the results of several trials with different ϵ and $\mu_{\max}$ on $0 \le t \le 0.1$ with an initial time-step of $\Delta T = 0.005$. Again, as the tolerance decreases the order increases and the error decreases.

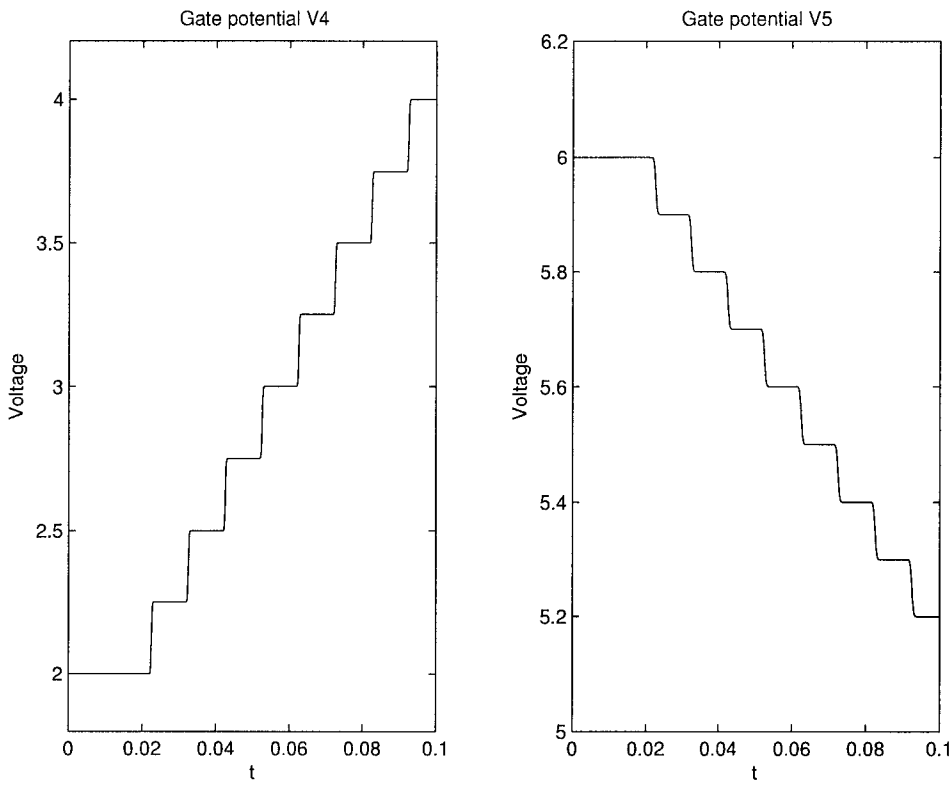

Figure 8.1.1: Gate voltages V4 and V5 for Example (8.1.2)

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Error	Total # of Steps	Total Time (s)
$1e-2$	8	1.9	$5e-1$	125	25
$1e-4$	8	3.7	$9e-2$	197	74
$1e-6$	8	5.7	$2e-4$	249	141
$1e-8$	8	7.2	$6e-6$	348	252

Table 8.1.4: Solution of Example (8.1.2) using Implicit Euler

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Error	Total # of Steps	Total Time (s)
$1e-2$	16	1.5	$7e-1$	20	36
$1e-4$	16	2.2	$4e-2$	27	85
$1e-6$	16	2.5	$5e-2$	43	188
$1e-8$	16	3.4	$8e-3$	58	362
$1e-10$	16	4.3	$9e-5$	79	691

Table 8.1.5: Solution of Example (8.1.2) using Order 3 Radau IIA

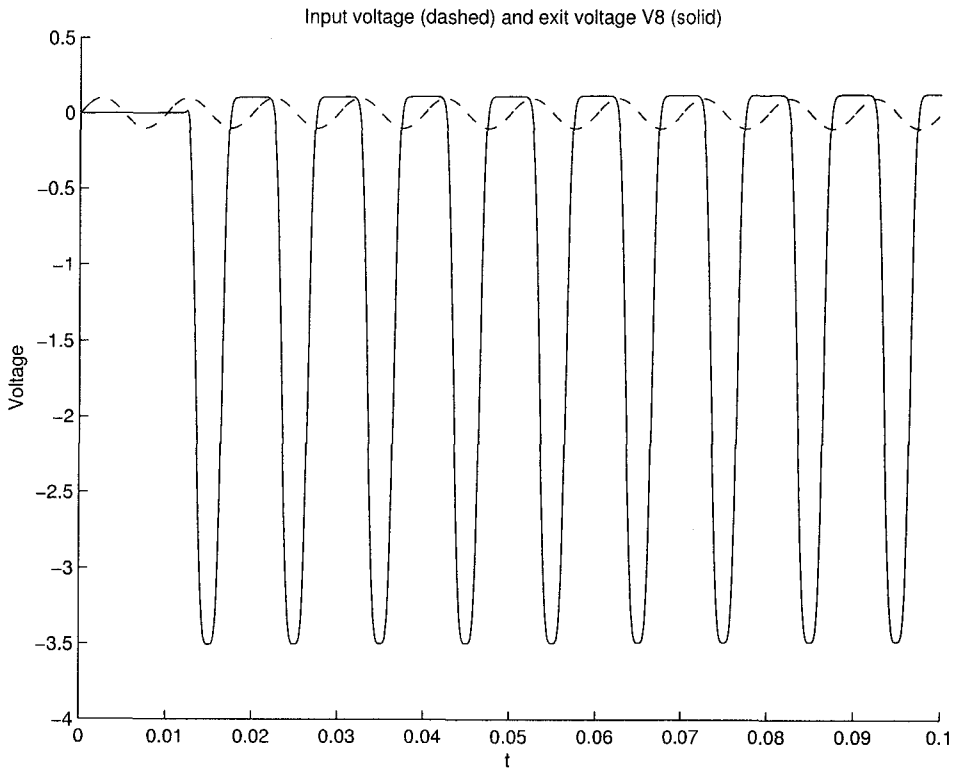

Figure 8.1.2: Exit voltage V8 for Example (8.1.2)

Example 8.1.3 *The following index-2 DAE is studied in [15] and [28] as a model for a compressor, with a valve to control discharge and a forcing term:*

$$\begin{aligned} k' &= 0.05(c-n)+F_1 \\ s' &= 1.32+0.2(c-s)+F_2 \\ M' &= \mu - m + F_3 \\ 0 &= n-k \\ 0 &= q(m)+300(c-s)/M+F_4 \\ 0 &= 0.0025M^2+0.482\mu^2/k^2-49.58^2+F_5 \end{aligned} \tag{8.1.8}$$

where

$$\begin{aligned} \mu(t) &= 15+5\tanh(t-10) \\ q(m) &= 0.01m^2-0.075m+3.35. \end{aligned} \tag{8.1.9}$$

The initial conditions and forcing terms have been chosen so that the exact solution is equal to

$$\begin{aligned} k &= 0.25+0.1\sqrt{t}\cdot S(10,0.45,2,t) \\ s &= 6.85+2(t-10)e^{-0.5(t-10)}S(10,0.2,1,t)+S(10,0.5,3,t) \\ M &= 734-S(9,25,2,x)+150(1-\exp(-0.05(x-11)))S(11,1,1,x) \\ n &= k \\ c &= 0.3+S(10,0.5,7,x)+2.3\exp(-0.2(x-10)^2) \\ m &= 10-S(9,5,6,x)+20\exp(-0.33(x-10)^2) \\ &\quad +10(1-\exp(-0.1(x-12)))S(12,1,1,x) \end{aligned} \tag{8.1.10}$$

with S given by Eqn. (8.0.1).

The variable n is retained so that the jacobian of the right hand side is square. The exact solution is plotted in Figure (8.1.3). Tables (8.1.6) and (8.1.7) contain the results of several trials on $0 \le t \le 40$ with an initial time-step of $\Delta T = 1$. As seen in Table (8.1.6), the order of the algorithm reaches its maximum of $\mu_{\max} = 8$ for tolerance $\epsilon = 1e-6$. The algorithm is forced to take many more steps when the tolerance is $\epsilon = 1e-8$. This is not a problem when we increase $\mu_{\max}$ to 16. As seen in Table (8.1.7), the algorithm is able to increase its order to match the decrease in tolerance.

8.2 Automated coordinate change

We use the method of Chapter 4 to automatically construct a change of coordinates for Algorithm 2.2.2. We test the performance of this method by constructing several boundary matrices A and C

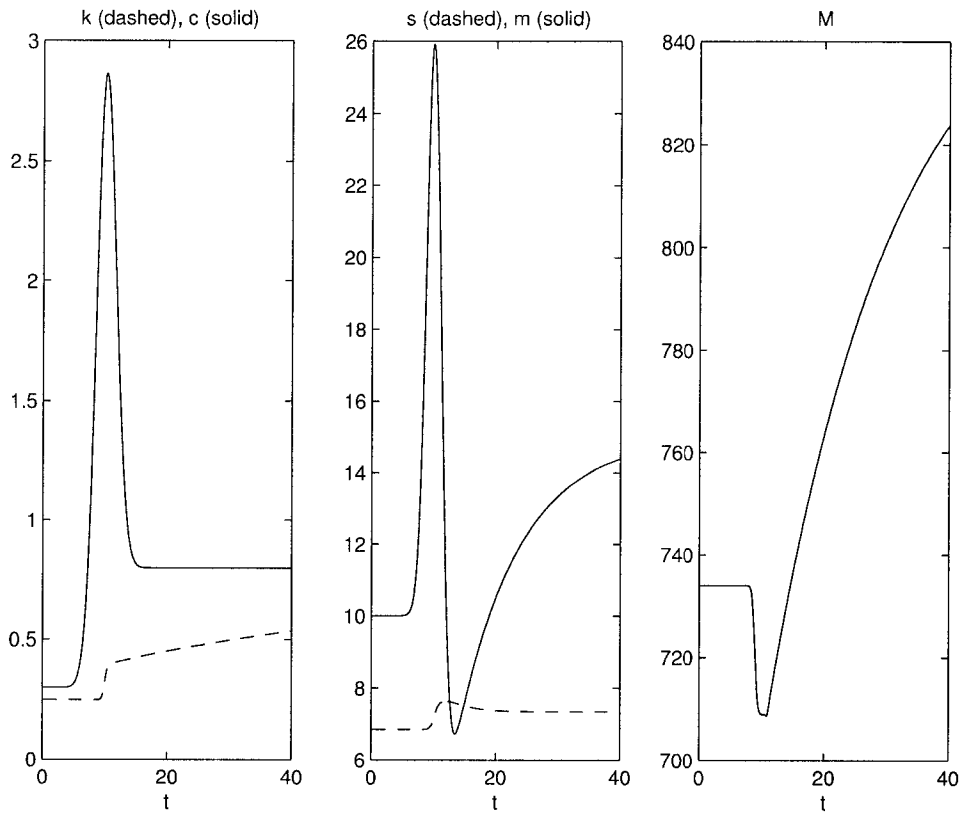

Figure 8.1.3: Forced exact solution for Example (8.1.3)

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Error	Total # of Steps	Total Time (s)
$1e-2$	8	4.2	$1e-0$	20	6
$1e-4$	8	7.1	$8e-3$	39	17
$1e-6$	8	7.9	$4e-5$	307	150
$1e-8$	8	8.0	$6e-7$	29680	10213

Table 8.1.6: Solution of Example (8.1.3) using Implicit Euler

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Error	Total # of Steps	Total Time (s)
$1e-2$	16	1.5	$3e-1$	10	21
$1e-4$	16	2.4	$4e-3$	14	51
$1e-6$	16	9.4	$8e-6$	320	245
$1e-8$	16	15.8	$2e-8$	1282	1873

Table 8.1.7: Solution of Example (8.1.3) using Order 3 Radau IIA

such that their sum D is singular. We construct a coordinate transformation $\mathcal{T}(x)$ on the interval $[-1,1]$ for each pair of boundary matrices. We measure the average condition number and maximum condition number of $\hat{D} = A\mathcal{T}(-1) + C\mathcal{T}(+1)$, $\mathcal{T}(x)$ and $\mathcal{T}'(x)$ over several trials. The process used to construct singular boundary matrices is as follows:

1. Choose the dimension n of the boundary matrices.
2. Choose the rank $j \leq n$ of A and C.
3. Choose an even number d for the rank of $D = A + C$.
4. Let $k = j - \frac{d}{2}$.
5. Construct $2j - k$ rank-1 outer products $u_i \cdot v_i^{\mathrm{T}}$ at random.
6. Set $A = \sum_{i=1}^{j} u_i \cdot v_i^{\mathrm{T}}$.
7. Set $C = \sum_{i=j-k+1}^{j} -u_i \cdot v_i^{\mathrm{T}} + \sum_{i=j+1}^{2j-k} u_i \cdot v_i^{\mathrm{T}}$.
8. Now in general, $D = A + C$ will have rank $2(j-k) = d$.

In general, $[A|C]$ will have full rank if $2j - k \geq n$. This is equivalent to requiring $2j + d \geq 2n$. Table (8.2.1) contains the results for $n = 6$ and values of j ranging from 6 to 4. Each test was done with 500 trials. The condition numbers of $\hat{D}$,$\mathcal{T}(x)$ and $\mathcal{T}'(x)$ are all fairly low, even in the worst cases. The condition number of $\hat{D}$ is highest when D is singular. However, in this case the condition numbers of $\mathcal{T}$ and $\mathcal{T}'$ are at their lowest.

8.3 Partial differential equations

We apply Algorithm 7.0.1 to initial boundary problems of the form (5.0.6), where the time derivative of the solution is a function of space, time, the solution and its first two spatial derivatives. We use the 2-norm of the residual (instead of the sup-norm or relative sup-norm) to choose timesteps and estimate the error. The sup-norm tends to remain high for the first few correction steps (in the

Rank of A and C	Rank of D	$\bar{\kappa}(\hat{D})$	$\kappa_{\max}(\hat{D})$	$\bar{\kappa}(\mathcal{T}(x))$	$\kappa_{\max}(\mathcal{T}(x))$	$\bar{\kappa}(\mathcal{T}'(x))$	$\kappa_{\max}(\mathcal{T}'(x))$
6	4	4e1	2e2	8e0	2e1	1e2	6e4
6	2	7e1	3e3	8e0	2e1	2e2	1e4
6	0	4e3	9e5	6e0	6e0	8e0	9e0
5	4	5e1	1e3	8e0	2e1	1e2	5e3
5	2	3e2	2e4	8e0	3e1	1e2	7e3
4	4	2e2	6e3	8e0	2e1	2e2	5e3

Table 8.2.1: Results for 6 by 6 boundary matrices, 500 trials each

vicinity of shocks). This makes it difficult to guage the adequacy of our time-step early on. On the other hand, the 2-norm seems to be a better indicator of global accuracy, and decreases with each correction when the time-step is reasonably small. Our code allows for general diffusion terms and general boundary conditions. The user needs to supply the following:

1. Functions F, H, J, M which can be called for arbitrary $x, t, u, \partial_x u, \partial_{xx} u$.

2. Initial solution $u_0, \partial_x u_0, \partial_{xx} u_0$ which can be called for arbitraray x.

3. Boundary functions A, C, γ which can be called for arbitrary t.

4. Control parameters ϵ and $\mu_{\max}$, starting time T_0 and initial time-step ΔT.

Since the implicit Euler correction process is first order, we choose the order of our quadrature scheme q to be the smallest power of 2 greater than $\mu_{\max}$. We use the refinement scheme of [25] in Algorithm 2.2.2 with a relative tolerance of $10^{-\log_2(q)-1}$. This requires a more stringent resolution of each subinterval if more quadrature points are used. As before, we are interested in determining if the magnitude of the residual is an accurate indicator of the error, and if the error decreases as we decrease the tolerance ϵ. We also check to see if our algorithm is performing multiple corrections per time-step.

Example 8.3.1 *Viscous Buckey-Leverett equations. The PDE models two-phase fluid flow, and contains a forcing term. The density of one of the fluids is given by u, and the density of the other*

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Sup Norm Error	2 Norm Error	Total # of Steps	Total Time (s)
$1e-2$	2	1.9	$1e-2$	$4e-7$	8	49
$1e-4$	4	3.2	$3e-3$	$6e-9$	13	119
$1e-6$	6	5.8	$8e-4$	$8e-11$	24	1202
$1e-8$	8	6.5	$7e-5$	$9e-14$	31	2849
$1e-10$	10	8.0	$2e-6$	$3e-15$	32	5762

Table 8.3.1: Solution of Example (8.3.1)

fluid is $1-u$.

$$\begin{aligned} f(u) &= u^2 \cdot (u^2 + 0.5(1-u)^2)^{-1} \\ u_t + \tfrac{d}{dx} f(u) &= 0.005 u_{xx} + F. \end{aligned} \tag{8.3.1}$$

The input data are chosen so that the exact solution is:

$$u(x,t) = S(-1.366t, 0.5774, 0.01, x) + R(-0.5 \cdot 1.366t, 1 - 0.5774, 1.366t, 0.01, x) \tag{8.3.2}$$

with S, R *given by Eqns. (8.0.1, 8.0.2).*

This problem simulates the solution of the Buckey-Leverett equations with step-function initial data. The unforced solution involves a shock wave followed by a rarefaction wave, both propagating into the region of lower density. The solution is computed on the interval $-1 \leq x \leq 1$ and $0 \leq t \leq 0.5$ with an initial time-step of $\Delta T = 0.01$. The exact solution is plotted in Figure (8.3.1). Table (8.3.1) contains the average number of corrections, error magnitude, total number of steps and total time for several trials with ϵ ranging from $1e-2$ to $1e-10$ and $\mu_{\max}$ ranging from 2 to 10. The computation becomes more difficult as ϵ decreases and $\mu_{\max}$ increases. The order of the algorithm increases to compensate.

Example 8.3.2 *Viscous shallow-water equations. This system models low amplitude waves of an incompressible fluid in a channel.* ν *represents the velocity and* ϕ *the height of the fluid.*

$$\begin{aligned} \nu_t + \nu\nu_x + \phi_x &= \nu_{xx}/512 + F_1 \\ \phi_t + \nu_x\phi + \nu\phi_x &= \phi_{xx}/512 + F_2 \end{aligned} \tag{8.3.3}$$

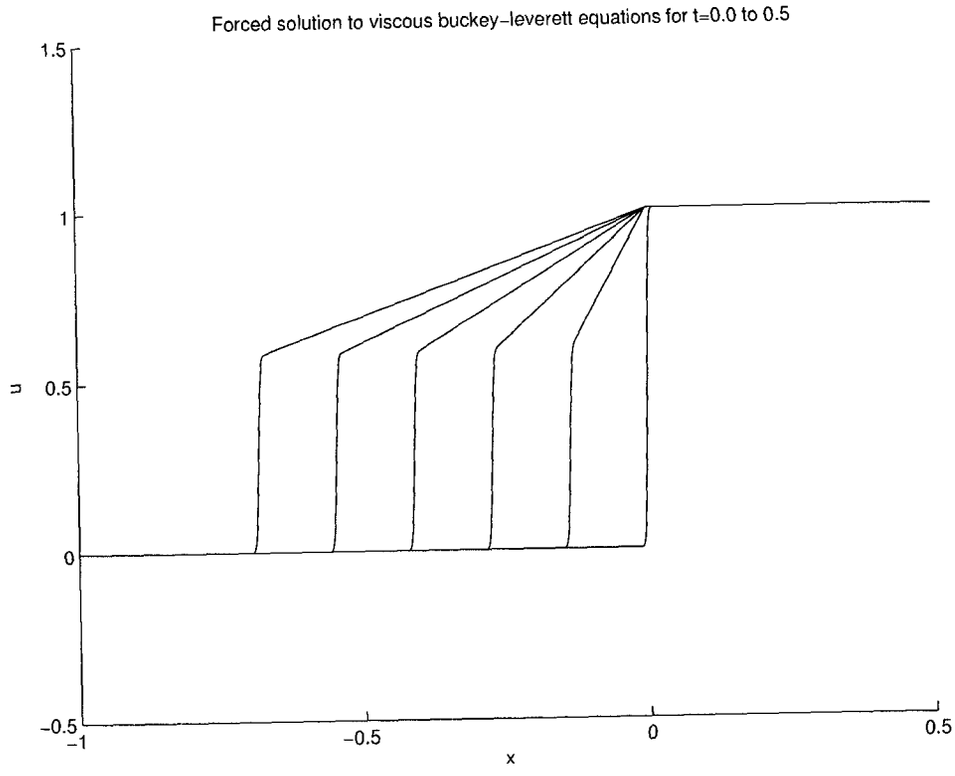

Figure 8.3.1: Forced exact solution for Example (8.3.1)

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Sup Norm Error	2 Norm Error	Total # of Steps	Total Time (s)
$1e-2$	2	0.3	$3e-1$	$9e-4$	10	123
$1e-4$	4	1.5	$1e-2$	$1e-6$	32	978
$1e-6$	6	2.8	$7e-3$	$8e-7$	130	12341
$1e-8$	8	5.8	$6e-4$	$4e-9$	412	148921
$1e-10$	10	7.9	$9e-5$	$2e-11$	620	236946

Table 8.3.2: Solution of Example (8.3.2)

The initial conditions, boundary values and forcing terms are chosen so that the exact solution is

$$\begin{aligned} \nu &= R(-1.03t, 0.94, 1.4t, 0.002, x) - S(1.8t, 0.94, 0.002, x) \\ \phi &= 3 - R(-1.03t, 1.41, 1.4t, 0.002, x) - S(1.8t, 0.59, 0.001, x), \end{aligned} \tag{8.3.4}$$

with S, R given by Eqns. (8.0.1, 8.0.2).

This is intended to simulate the 'dam break' problem which involves initially still water at two different heights and zero velocity separated by a barrier. At $t = 0$ the barrier is removed (or broken) and the water is allowed to flow into the area of lower height. The unforced solution involves a shock propagating into the region of lower height, and a rarefaction wave propagating in the opposite direction [41, 39]. The solution is computed on the interval $-1 \leq x \leq 1$ and $0 \leq t \leq 0.3$ with an initial time-step of $\Delta T = 0.01$. The exact solution is plotted in Figure (8.3.2). Table (8.3.2) contains the average number of corrections, error magnitude, total number of steps and total time for several trials with ϵ ranging from $1e-2$ to $1e-10$ and $\mu_{\max}$ ranging from 2 to 10. Again, as the tolerance decreases the order increases and the error decreases.

Example 8.3.3 *The PDE*

$$u_t = (2 + x^2(2e^{-x^2/2} - 1) + t^2 e^{x^2/2})u + \frac{3x}{t}u_x + \frac{1}{t}e^{x^2/2}u_{xx} \tag{8.3.5}$$

has the exact solution

$$u(x,t) = t\cos(xt)\exp(\exp(\frac{-x^2}{2})). \tag{8.3.6}$$

The solution is computed on the interval $-3 \leq x \leq 3$ and $1 \leq t \leq 3$ with an initial time-step of $\Delta T = 0.1$. The advection remains roughly order unity, but the growth and diffusion are quite large

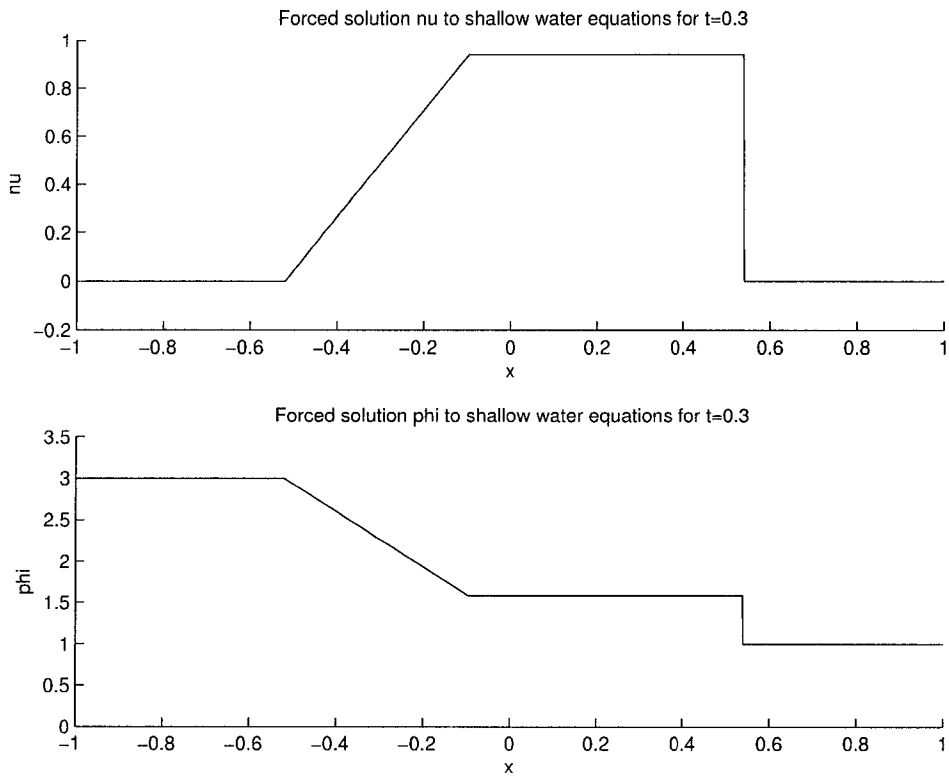

Figure 8.3.2: Forced exact solution to Example (8.3.2)

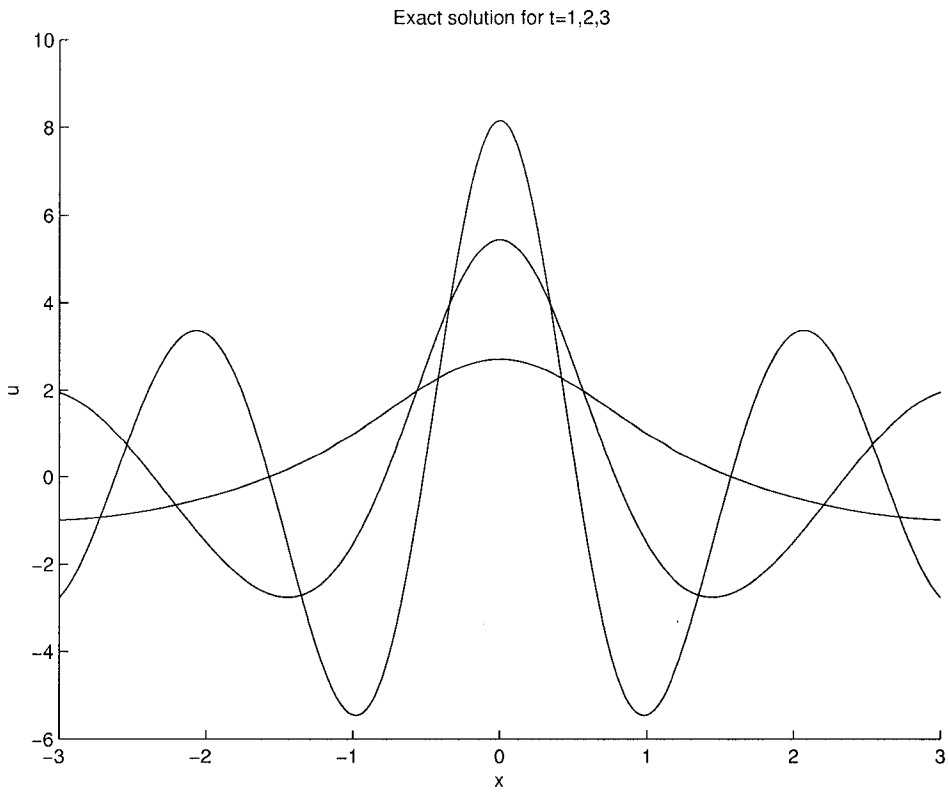

Figure 8.3.3: Exact solution for Example (8.3.3)

near the endpoints of the interval. The exact solution is plotted in Figure (8.3.3). Table (8.3.3) contains the average number of corrections, error magnitude, total number of steps and total time for several trials with ϵ ranging from $1e-2$ to $1e-14$ and $\mu_{\max}$ ranging from 2 to 14.

Example 8.3.4 *One difficult test case is the viscous Euler equations in one spatial dimension [26]. The density ρ, velocity ν and energy E of a 1-d tube of gas evolve according to:*

$$\begin{bmatrix} \rho \\ \rho\nu \\ E \end{bmatrix}_t + \begin{bmatrix} \rho\nu \\ \rho\nu^2 + p \\ \nu(E+p) \end{bmatrix}_x = D \begin{bmatrix} \rho \\ \rho\nu \\ E \end{bmatrix}_{xx} + \begin{bmatrix} F_\rho \\ F_{\rho\nu} \\ F_E \end{bmatrix} \tag{8.3.7}$$

where the pressure p is given by

$$p = \frac{2}{3}(E - \frac{1}{2}\rho\nu^2) \tag{8.3.8}$$

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Sup Norm Error	2 Norm Error	Total # of Steps	Total Time (s)
$1e-2$	2	1.2	$2e-1$	$5e-3$	149	65
$1e-4$	4	2.5	$8e-2$	$3e-3$	153	83
$1e-6$	6	3.4	$1e-3$	$5e-6$	152	162
$1e-8$	8	5.5	$5e-4$	$7e-7$	168	295
$1e-10$	10	6.1	$6e-5$	$4e-8$	193	587
$1e-12$	12	8.0	$7e-6$	$3e-9$	230	3397
$1e-14$	14	9.1	$1e-6$	$2e-9$	275	7229

Table 8.3.3: Solution of Example (8.3.3)

and the diffusion constant D is small. The input data are chosen so that the exact solution is:

$$\begin{aligned} \rho &= 3 - R(-s_r t, 1.0, t, D, x) - S(s_c t, 0.6, D, x) - S(s_s t, 0.4, D, x) \\ \nu &= R(-s_r t, 0.46, t, D, x) - S(s_s t, 0.46, D, x) \\ p &= 3 - R(-s_r t, 1.34, t, D, x) - S(s_s t, 0.66, D, x), \end{aligned} \tag{8.3.9}$$

with S, R given by Eqns. (8.0.1, 8.0.2) and speeds

$$s_s = 1.5 \qquad s_c = 0.5 \qquad s_r = 1.0. \tag{8.3.10}$$

The solution is computed on the interval $-1 \leq x \leq 1$ and $0 \leq t \leq 0.5$ with an initial time-step of $\Delta T = 0.1D$. This example simulates the solution of the shock tube problem. A shock tube is a tube filled with gas, initially divided into two parts. The density and pressure have distinct constant values in the parts of the tube. The velocity is initially zero everywhere. At time $t = 0$, the gas is allowed to flow between the two parts. The solution to this problem contains a shock wave followed by a contact discontinuity, both propagating into the low-pressure region. A rarefaction wave propagates into the high-pressure region. Figure (8.3.4) contains graphs of the exact solutions for diffusion constants ranging from 2^{-4} to 2^{-6} and $t = 0.5$. Table (8.3.4) contains the average number of corrections, error magnitude, total number of steps and total time for several trials on $0 \leq t \leq 0.3$ with $D = 2^{-4}$ and ϵ ranging from $1e-2$ to $1e-8$ and $\mu_{\max}$ ranging from 2 to 8. Tables (8.3.5) and (8.3.6) contain the results for $D = 2^{-5}$ and $D = 2^{-6}$ respectively. As D decreases the exact solution sharpens and the problem becomes more difficult to solve. This is reflected by increases in error and computation time (for the same tolerance ϵ) as D decreases.

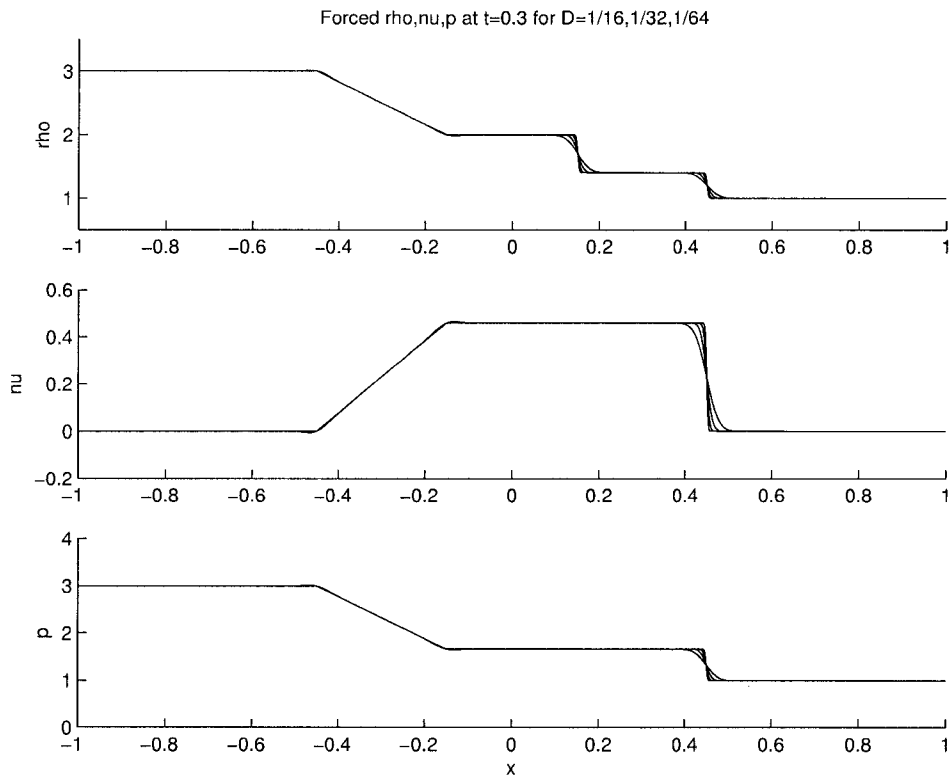

Figure 8.3.4: Forced exact solution for Example (8.3.4)

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Sup Norm Error	2 Norm Error	Total # of Steps	Total Time (s)
$1e-2$	8	2.1	$6e-3$	$2e-7$	10	65
$1e-4$	8	4.3	$2e-2$	$4e-8$	10	83
$1e-6$	8	5.4	$2e-4$	$4e-11$	15	162
$1e-8$	8	7.9	$1e-5$	$2e-13$	17	295

Table 8.3.4: Solution of Example (8.3.4) with D=1/16

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Sup Norm Error	2 Norm Error	Total # of Steps	Total Time (s)
$1e-2$	8	2.5	$3e-2$	$4e-7$	10	417
$1e-4$	8	5.3	$2e-2$	$2e-7$	11	2591
$1e-6$	8	6.7	$1e-4$	$3e-9$	22	5577
$1e-8$	8	8.0	$6e-5$	$2e-10$	33	10621

Table 8.3.5: Solution of Example (8.3.4) with D=1/32

Tolerance ϵ	$\mu_{\max}$	Average # of Corrections	Sup Norm Error	2 Norm Error	Total # of Steps	Total Time (s)
$1e-2$	16	2.7	$3e-2$	$5e-7$	24	1689
$1e-4$	16	5.7	$2e-2$	$2e-7$	23	8360
$1e-6$	16	7.9	$9e-3$	$1e-9$	26	26409
$1e-8$	16	10.3	$2e-4$	$6e-12$	36	57772

Table 8.3.6: Solution of Example (8.3.4) with D=1/64

Chapter 9

Conclusions

Algorithm 3.3.1 is a very general method that can be applied to index-1 and index-2 DAEs. If high accuracy is desired the method performs at high-order. The residual is a good indicator of the error, and can be used to effectively control the step-size. This allows the algorithm to spend time refining the solution where the problem is stiff. This method is fast except when the maximum order is reached. This can be overcome by switching to a higher-order quadrature scheme (and a higher maximum order).

It should be possible to prove the convergence of spectral deferred correction applied to index-2 DAEs, although the proof may be considerably more complicated than the proof of Theorem 3.3.1. It may also be possible to apply spectral deferred correction directly to index-1 and index-2 DAEs of the form

$$F(y(t), y'(t)) = 0. \tag{9.0.1}$$

This will allow for an even more general method that can be applied without any preliminary change of coordinates.

Algorithm 7.0.1 is a general method that can be applied to a number of PDEs. The automated coordinate transformation is robust, and can handle general boundary conditions easily. By using an adaptive boundary value problem solver and controlling the step-size, shocks and other small-scale features are resolved locally. High accuracy is achieved by performing multiple correc-

tions. As long as the order of the method is not limited by the order of the quadrature scheme, it seems as though ϵ-accuracy is achieved for the 2-norm in $o(\log(\epsilon))$ time.

The techniques used to reduce a 1+1 dimensional PDE to a sequence of boundary value problems can also be applied to PDEs with more than one spatial dimension. This reduction produces a sequence of elliptic PDEs [8] along with boundary data. Each elliptic PDE may be solved with the best available scheme [20, 19, 40, 21]. Hopefully spectral deferred correction can be used to improve the order (and accuracy) of the resulting numerical solution. If this all works, it would extend spectral deferred correction to initial-boundary problems on more complicated (higher-dimensional) domains.

Our algorithm only uses first-order implicit Euler in time. It is possible to apply higher-order multistep or Runge-Kutta methods instead. This may increase the order of accuracy of the initial approximate solution, and each subsequent correction. Algorithm 7.0.1 spends most of its computation time solving boundary value problems. Therefore if solving each boundary value problem takes roughly the same amount of time, then increasing the order of accuracy of each correction would increase the efficiency of the method considerably.

Bibliography

[1] U. M. Ascher, R. M. M. Mattheij, and R. D. Russel. *Numerical solution of Boundary Value Problems for Ordinary Differential Equations.* siam, 1995.

[2] R. Bayer and A. E. M. McCreight. Symmetric binary b-trees: data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, 1972.

[3] K. Böhmer, P. Hemker, and H. J. Stetter. *The defect correction approach, in: K. B. H. J. Stetter (Ed.), Defect Correction Methods. Theory and Applications.* Springer-Verlag, 1984.

[4] A. Bourlioux, A. Layton, and M. Minion. Multi-implicit spectral deferred correction methods for problems of reactive flow. *Journal of Computational Physics*, in preparation.

[5] K.S. Breuer and R. M. Everson. On the errors incurred calculating derivatives using chebyshev polynomials. *Journal of Computational Physics*, 99:56–67, 1992.

[6] Clint Dawson and Robert Kirby. Solution of parabolic equations by backward euler-mixed finite element methods on a dynamically changing mesh. *SIAM Journal on Numerical Analysis*, 37(2):423–442, 1999.

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

[8] L. C. Evans. *Partial Differential Equations.* American Mathematical Society, 1998.

[9] J. Flaherty, P. Paslow, M. Shepard, and J. Vasilakis. *Adaptive Methods for Partial Differential Equations.* SIAM, 1989.

[10] B. Fornberg. *A Practical Guide to Pseudospectral Methods.* Cambridge University Press, 1998.

[11] R. Frank and C. W. Ueberhuber. Iterated defect correction for the efficient solution of stiff systems of ordinary differential equations. *BIT*, 17:146–159, 1977.

[12] G. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, 1971.

[13] D. Gottlieb and S. A. Orszag. Numerical analysis of spectral methods: Theory and applications. *CBMS-NSF Regional Conference Series Applied Mathematics*, 26, 1977.

[14] B. Guo, H. Ma, and E. Tadmor. Spectral vanishing viscosity method for nonlinear conservation laws. *SIAM Journal on Numerical Analysis*, 39(4):1254–1268, 2001.

[15] E. Hairer, C. Lubich, and M. Roche. *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods.* Springer-Verlag, 1989.

[16] E. Hairer, S. P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I.* Springer, 2000.

[17] Anders C. Hansen and John Strain. On the order of spectral deferred correction. in preparation.

[18] J. S. Hicks and J. Wei. Numerical solution of parabolic partial differential equations with two-point boundary conditions by use of the method of lines. *Journal of the ACM*, 14(3):549–562, 1967.

[19] R. W. Hockney. A fast direct solution to poisson's equation using fourier analysis. *J. Assoc. Comput. Mach.*, 12:95–113, 1965.

[20] E. N. Houstis and T. S. Papatheodorou. A high order fast elliptic equation solver. *ACM Trans. Math. Software*, 5:431–441, 1979.

[21] J. Huang and L. Greengard. A fast direct solver for elliptic partial differential equations on adaptively refined meshes. *Journal on Scientific Computing*, 21(4):1551–1556, 2000.

[22] H. B. Keller. *Numerical Methods for Two-Point Boundary Value Problems*. Dover, 1992.

[23] J.D. Lambert. *Numerical Methods for Ordinary Differential Systems*. John Wiley & Sons, 1991.

[24] A. Layton and M. Minion. Conservative multi-implicit spectral deferred correction methods for reacting gas dynamics. *Journal of Computational Physics*, in preparation.

[25] June-Yub Lee and Leslie Greengard. A fast adaptive numerical method for stiff two-point boundary value problems. *SIAM Journal on Scientific Computing*, 18(2):403–428, 1997.

[26] R. J. LeVeque. *Numerical Methods for Conservation Laws*. Birkhauser Verlag, Basel, Switzerland, 1992.

[27] Mohammad Majidi and Gerhard Starke. Least-squares galerkin methods for parabolic problems ii: The fully discrete case and adaptive algorithms. *SIAM Journal on Numerical Analysis*, 39(5):1648–1666, 2002.

[28] M.Berzins, A.J.Preston, P.M.Dew, and L.E.Scales. Towards efficient d.a.e. solvers for the solution of dynamic simulation problems. *Proc of I.M.A. 1989 O.D.E. Conference, (Eds.) I.*, 1:299–308, 1992.

[29] Peter K. Moore. An adaptive finite element method for parabolic differential systems: Some algorithmic considerations in solving in three space dimensions. *SIAM Journal on Scientific Computing*, 21(4):1567–1586, 2000.

[30] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover, 1998.

[31] V. Pereyra. Iterated deferred corrections for nonlinear operator equations. *Numer. Math.*, 10:316–323, 1966.

[32] V. Pereyra. On improving an approximate solution of a functional equation by deferred corrections. *Numer. Math.*, 8:376–391, 1966.

[33] V. Pereyra. On improving the approximate solution of a functional equation by deferred corrections. *Numer. Math.*, 8:376–391, 1966.

[34] V. Pereyra. Iterated deferred correction for nonlinear boundary value problems. *Numer. Math.*, 11:111–125, 1968.

[35] P. Rentrop, M. Roche, and G. Steinebach. The application of rosenbrock-wanner type methods with stepsize control in differential-algebraic equations. *Numer. Math*, 55:545–563, 1989.

[36] R. Sedgwick. *Algorithms*. Addison-Wesley Publishing Company, 1988.

[37] P. Starr and V. Rokhlin. On the numerical solution of two-point boundary value problems ii. *Appl. Math*, 79:271–289, 1988.

[38] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis 2nd edition*. Springer-Verlag, Berlin, New York, 1993.

[39] J. J. Stoker. *Water Waves*. Interscience, New York, 1957.

[40] P. Swarztrauber and R. Sweet. Efficient fortran subprograms for the solution of elliptic partial differential equations. *NCAR-TN/IA*, 109:135–137, 1975.

[41] V.C.Fang and T.W.H.Sheu. Two element-by-element iterative solutions for shallow water equations. *SIAM Journal on Scientific Computing*, 22(6):2075–2092, 2001.

[42] A. Zafarullah. Application of the method of lines to parabolic partial differential equations with error estimates. *Journal of the ACM*, 17(2):294–302, 1970.