Abstract

COFFEY, TODD STIRLING. Temporal and Pseudo-Temporal Numerical Integration Methods. (Under the direction of C. T. Kelley).

Numerical methods for integrating partial differential equations are used in nearly every scientific field. In this dissertation we study two types of numerical integration methods, transient methods and pseudo-transient methods. Transient methods for partial differential equations look for time-accurate solutions that explain the evolution of the equation (although a steady state solution may evolve). Pseudo-transient methods look for steady-state solutions of partial differential equations while paying attention to the transient behavior to aid in stability. In contrast, root-finding methods, e.g. line-search methods, look only for a steady-state solution often not paying attention at all to the transient behavior of the problem.

Pseudo-transient continuation is a method for solving steady state solutions of partial differential equations, and is used when traditional line-search methods fail to converge or converge to non-physical solutions. The method is a hybrid between implicit Euler and Newton's method where variable step-sizes are used to transfer from one method

to the other. We demonstrate the performance of pseudo-transient continuation both numerically and theoretically on a variety of problems. We extend the global convergence theory, which currently covers a class of ordinary differential equations, to include a class of semi-explicit index-1 differential-algebraic equations.

We also studied `CVode`, a transient code for solving nonlinear partial differential equations. In this work, we explain how `CVode` was extended to allow for a two-grid nonlinear solver. The two-grid solver coarsens a given mesh and solves the nonlinear problem on the coarse mesh, which is then moved back to the fine mesh for refining. This approach can be less expensive than computing the full nonlinear solution on the fine mesh. We explore some of the theoretical and computational issues involved in implementing this method for a radiative transfer problem as might be seen in stellar fusion.

TEMPORAL AND PSEUDO-TEMPORAL NUMERICAL INTEGRATION METHODS

BY

TODD S. COFFEY

A DISSERTATION SUBMITTED TO THE GRADUATE FACULTY OF
NORTH CAROLINA STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

APPLIED MATHEMATICS

COMPUTATIONAL MATHEMATICS CONCENTRATION

RALEIGH, NORTH CAROLINA

SEPTEMBER 2002

APPROVED BY:

C. T. KELLEY
CHAIR OF ADVISORY COMMITTEE

D. S. MCRAE

P. A. GREMAUD

M. SHEARER

C. S. WOODWARD

Biography

The author was born early in 1973 in Portland Oregon and raised in Beaverton Oregon. He attended Sunset high school to drop out in 1990, moving on to Portland Community College to receive his high school diploma in December of 1991 and an associates degree in science in June of 1993. Shortly thereafter, he attended Oregon State University in Mathematics and received his Bachelors degree in pure and applied mathematics in December of 1995. Then he moved to North Carolina State University for his graduate education to receive his masters in applied mathematics in May of 1999 and his doctor of philosophy degree in applied computational mathematics in August of 2002.

The author's first experience with computers came in 1978 when he "helped" his father build a Tektronix 4051. His first real interaction came in 1980 with an Apple II+. After learning to touch type, a quick succession of Intel machines followed: an 8086 in 1986, an 80286 in 1988, an 80386 in 1990, and an 80486 in 1992. The author's first experience with *nix came with Coherent on an 80386 in 1993. With his attendance at Oregon State University, he started using Solaris SparcII stations in 1994 and installed Slackware GNU/Linux on an 80486 in 1994. Thereafter *nix became a major player in

his life. In graduate school he switched his main operating system to RedHat on an 80586 in 1996 and installed OpenBSD for a firewall on an 80486. Also in graduate school, he started using MPI on an IBM SP/2 at the North Carolina Supercomputing Center. That experience opened his eyes to the possibilities of parallel processing. Right before his last year in graduate school, he spent a summer at Lawrence Livermore National Laboratory on an internship. It was during this internship that he was first introduced to ASCI Blue Pacific (also an IBM SP/2, but *much* bigger). In the last year of graduate school he switched his main machine to Debian on an AMD-Duron machine in 2002, and that is where we leave him today.

Acknowledgments

First and foremost: thank you Kristy. I certainly would not have made it this far without your encouragement, support and love.

I would like to thank Tim Kelley, who agreed to take me in when I was down and teach me the ways of numerical analysis. And thanks go to Carol Woodward who demonstrated amazing graciousness in letting me work with her over the summer at LLNL and stay at her house while visiting in November (not to mention Robert Woodward who let me occupy my time upgrading their TiVo). Thanks also go to the rest of my committee: Scott McRae, Pierre Gremaud, and Michael Shearer for being impressed with my work.

Most of the more impressive numerical results in this dissertation would not have been possible without the efforts of Jeff McMullan (for the Euler code) and David Keyes (for the DAE results). In addition, I would like to thank Chris Kees and Matthew Farthing for letting me learn from their groundwater code.

Special thanks go to my cronies (as Tim puts it): Matt & Mandy Matthews, Bob Wieman, and Rebecca Segal. Thanks also go out to my other friends and fellow graduate students (in alphabetical order): Brian Adams, Astrid Battermann (for letting

me help with [6]), Peter Berman, David Bortz, Brenda Burns-Williams, Melissa Choi, Cammey Cole, Marilyn Daily, Kimberly Drake, Artem Dyakonov, Jörg Gablonsky, John Haws, Jennifer Hontz, Mike & Michelle Horton, Lea Jenkins, Katie Kavanagh, Matthew Lasater, Rachel Levy, Andrew Malechuk, Jordan Massad, John May, Mike McLean, Kushal Narayanaswamy, John Peach, Laura & Kevin Potter, Rob Ray, Jill Reese, Ric del Rosario, Dewey Taylor, Will Turner, and Shannon Wynne.

My family also deserves great thanks for making me the person I am and supporting my crazy move across the country. Thank you Mom, Dad, Taina & Mark, and Lance!

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The focus of this research is the examination and analysis of pseudo-transient continuation (Ψtc), a numerical method for solving steady state partial differential equations. In the computational fluid dynamics (CFD) community, Ψtc has been used for some time to solve a variety of steady-state problems [46, 34, 39, 49, 60]. Finding specific examples where Ψtc outperforms Newton methods was more difficult than expected, partly due to the drive to solve problems rather than the drive to understand algorithms. The present work is an attempt to understand algorithms better. We carefully examined a variety of small and medium sized problems and found a few good representatives. The first is a steady state one-dimensional Burgers equation problem for which the exact solution is known. Another is a one dimensional Euler equation problem in which different discretizations are used for the residual and the Jacobian. The practice of using two different discretizations in such problems is common in the CFD community [23, 21, 22, 44] and

warranted further study. In following this line of research we extended the Ψtc theory to include a global convergence result that applies to not only ordinary differential equation dynamics but also differential-algebraic equation dynamics.

In the following sections of this chapter, we lay out the groundwork for the examples and analysis to come. First we explain the current theory for Newton's method and pseudo-transient continuation. Then the Euler equations are developed and restricted to one and two dimensions. Many of the numerical methods used by CFD researchers have a long history of development. We explain in detail the background and implementation of these methods where appropriate. Finally, the radiative transfer equations we use in the last chapter are introduced.

1.1 Newton's method

Newton's method, is an iterative solver for nonlinear systems of the form

$$F(u) = 0$$

where $u \in \mathbb{R}^N, F(u) \in \mathbb{R}^N$. Let $F'(u)$ be the Jacobian of F at u. Newton's method in its simplest form is written:

$$u_+ = u_c - F'(u_c)^{-1} F(u_c),$$

where u_c is the current iterate and u_+ is the next iterate. We assume that there exists a solution u^*, $F'(u)$ is Lipschitz continuous, and that $F'(u^*)$ is nonsingular. Under these assumptions, and the local assumption that u_c be sufficiently close to u^*, one can prove that Newton's method will converge quadratically [35, 50]. i.e.

$$\|u_+ - u^*\| \leq C\|u_c - u^*\|^2$$

for some positive constant C. Note that this convergence result only holds locally, which means that if u_c is not sufficiently close to u^*, then we have no guarantee of convergence. We cannot always start *sufficiently close* to the exact solution when we use these methods so it would be nice to have some kind of global convergence result. By a globally convergent algorithm we mean one for which any initial iterate will either converge to a root of F or fail to converge in one of a few failure modes [35].

To establish global convergence, Newton's method must be modified. One way to do this is to use a line-search to adjust the size of the Newton step $s = -F'(u_c)^{-1}F(u_c)$. The particular line search used in this research is the Armijo line-search method [3, 28].

1.1.1 Armijo Line-search

The Armijo line-search method attempts to choose a step size s as large as possible at the same time as sufficiently decreasing the value of $\|F\|$. Let λ be the reduction factor we apply to the Newton step s. Start by choosing $\lambda = 1$ and see if we satisfy the sufficient

decrease condition,

$$\|F(u_c + \lambda s)\| < (1 - \alpha\lambda)\|F(u_c)\|.$$

In this expression, $\alpha \in (0, 1)$ quantifies sufficient decrease. The larger α, the more decrease we demand. It is common to choose $\alpha = 10^{-4}$ [20] and we do so here. If we satisfy the sufficient decrease condition, then we accept the step s, otherwise we reduce λ by a factor $\beta \in (0, 1)$, usually $\frac{1}{2}$, and try the condition again. The Newton-Armijo algorithm is then written:

Newton-Armijo Algorithm

input: u, F, tol

While $\|F(u)\| > \text{tol}$

1. If $F'(u) = 0$ then terminate with failure.
2. Set $s = -F'(u)^{-1}F(u)$
3. Let $\lambda = 1$
4. While $\|F(u + \lambda s)\| > (1 - \alpha\lambda)\|F(u)\|$

   Set $\lambda = \beta\lambda$
5. $u = u + \lambda s$

We stop when we satisfy the sufficient decrease condition or we make λ too small, in which case we have one of the three failure modes of this globally convergent algorithm, line-search stagnation. This usually means the iterates are converging to a local minimum

of $\|F\|$ which is not a root. The other failure modes are: divergence of the u_c iterations and $F'(u_c)$'s becoming singular. Let the sequence $\{u_n\}$ denote the sequence of u_+ iterates. The convergence theorem in [35, 50] states that if $\{u_n\}$ and $\|F'(u_n)\|^{-1}$ are bounded (eliminating the possibility for divergence of u_n and the singularity of F') and F' is Lipschitz continuous in a neighborhood of the entire sequence $\{u_n\}$, then the iteration will converge to a root of F where the original assumptions above on F hold, full steps are taken in the terminal phase, and the convergence is q-quadratic [35], i.e. $u_n \to u^*$ and there is a $K > 0$ such that

$$\|u_{n+1} - u^*\| \leq K\|u_n - u^*\|^2.$$

1.1.2 Inexact Newton

The Newton step s is computed by solving the linear system,

$$F'(u_c)s = -F(u_c).$$

For simple functions, and one-dimensional problems in general, this is solved directly. As problems become larger and functions more complex, it becomes very appealing to not only approximate the inverse of the Jacobian $F'(u)$, but also to approximate $F'(u)$ itself. The inexact Newton condition [19] quantifies which steps will be accepted when we make these approximations. Any approximate step s will be accepted, provided it

satisfies

$$\|F'(u_c)s + F(u_c)\| \leq \eta_c \|F(u_c)\|. \tag{1.1}$$

This demands that the relative residual of the linear equation for s be less than η_c (or η_n as below) which defines how good the approximate step s is. The convergence theorem in [35, 19] states that if u_0 is sufficiently close to u^*, and the sequence $\{\eta_n\} \subset [0, \overline{\eta}]$, for some $\overline{\eta}$, then the inexact Newton iteration

$$u_{n+1} = u_n + s_n,$$

where s_n is defined by (1.1), converges q-linearly to u^*. Moreover, if $\eta_n \to 0$, then the convergence is q-superlinear, and if $\eta_n \leq K_\eta \|F(u_n)\|^p$ for some $K_\eta > 0$ then the convergence is q-superlinear with q-order $1 + p$. By q-linear convergence we mean, for some $\sigma \in (0, 1)$ and n sufficiently large,

$$\|u_{n+1} - u^*\| \leq \sigma \|u_n - u^*\|$$

holds. And q-superlinear convergence of q-order $p > 1$ means, $u_n \to u^*$ and there is a $K > 0$ such that

$$\|u_{n+1} - u^*\| \leq K \|u_n - u^*\|^p.$$

And finally, q-superlinear convergence is defined by,

$$\lim_{n\to\infty} \frac{\|u_{n+1} - u^*\|}{\|u_n - u^*\|} = 0.$$

Now we can write out the full algorithm for the inexact Newton's method with Armijo line-search. This algorithm describes the code `nsola` from [35].

Algorithm 1.1. nsola(u, F, τ)

1. $r_0 = \|F(u)\|$
2. *Do while* $\|F(u)\| > \tau_r r_0 + \tau_a$
   (a) *Find* d *such that* $\|F'(u)d + F(u)\| \le \eta\|F(u)\|$
   if no such d *can be found, terminate with failure.*
   (b) $\lambda = 1$
      i. $u_t = u + \lambda d$
      ii. *If* $\|F(u_t)\| < (1 - \alpha\lambda)\|F(u)\|$ *then* $u = u_t$.
      else
      Choose $\sigma \in [\sigma_0, \sigma_1]$
      $\lambda = \sigma\lambda$
      goto 2(b)i
      fi

In this algorithm, τ_r denotes the desired relative residual error and τ_a denotes the

desired absolute residual error. The only extra trick included in this algorithm is the varying of the reduction factor β between σ_0 and σ_1. In `nsola` $\sigma_0 = 0.1$ and $\sigma_1 = 0.5$. This condition simply allows us to choose a reduction factor between $\frac{1}{10}$ and $\frac{1}{2}$.

1.2 Pseudo-Transient Continuation

We consider the pseudo-transient continuation (Ψtc) algorithm of [36, 39, 48, 49]. This algorithm attempts to find the steady state solution of an ODE by approximately integrating the equation

$$\frac{du}{dt} = -F(u)$$

over time until $\frac{du}{dt} \approx 0$. The algorithm numerically integrates the initial value problem

$$u' = -V^{-1}F(u), \quad u(0) = u_0$$

to steady state using a variable time-step scheme that increases the time-step as $F(u)$ approaches 0. In this context, $u' = \frac{du}{dt}$ and V is a fixed nonsingular scaling matrix for the nonlinear function F.

The Ψtc sequence $\{u_n\}$ is defined by

$$u_{n+1} = u_n - \left(\delta_n^{-1}V + F'(u_n)\right)^{-1} F(u_n),$$

where F' is the Jacobian. Note that if $\delta_n = \infty$, then this is Newton's method. We have

the following algorithm from [36]:

Algorithm 1.2. *Pseudo-transient continuation*

1. *Set* $u = u_0$ *and* $\delta = \delta_0$. *Evaluate* $F(u)$.

2. *While* $\|F(u)\|$ *is too large.*

   (a) *Solve* $(\delta^{-1}V + F'(u))s = -F(u)$.

   (b) *Set* $u = u + s$.

   (c) *Evaluate* $F(u)$.

   (d) *Update* δ.

There are three δ update formulas described in [36]. The first is called "switched evolution relaxation" (SER) [46, 38, 48, 60, 36]:

$$\delta_n = \phi\left(\delta_{n-1}\frac{\|F(u_{n-1})\|}{\|F(u_n)\|}\right). \tag{1.2}$$

The second formula [34, 36] is based on the step size (SSF):

$$\delta_n = \phi\left(\delta_{n-1}\|u_n - u_{n-1}\|^{-1}\right). \tag{1.3}$$

And the third is based on controlling the temporal truncation error estimates (TTE) [39, 36]. We choose δ_n so

$$\left|\frac{\delta_n^2}{2(1+|u_i|)}\frac{d^2u_i}{dt^2}(t_n)\right| \leq \tau \tag{1.4}$$

holds for each i, given some constant τ. This is an approximation to the mixed absolute and relative error tolerance on the second derivative. We approximate $\frac{d^2u_i}{dt^2}(t_n)$ by,

$$\frac{2}{\delta_{n-1}+\delta_{n-2}}\left[\frac{(u_i)_n-(u_i)_{n-1}}{\delta_{n-1}}-\frac{(u_i)_{n-1}-(u_i)_{n-2}}{\delta_{n-2}}\right].$$

The idea with each of these δ update formulas is to keep the time-step small when far away from steady state, and to increase the time-step when approaching the solution to the steady state problem. In these δ update formulas, the function ϕ allows us to either put a cap on how large δ can become or allows us to switch over to steady state once δ becomes large enough. Let ϕ be defined by,

$$\phi(\xi)=\begin{cases}\xi & \xi\leq\xi_t\\ \\ \delta_{max} & \xi>\xi_t\end{cases},$$

where either $\xi_t=\delta_{max}$ or $\xi_t<\infty$ and $\delta_{max}=\infty$.

When δ is small Ψtc approximates forward Euler which tracks the transient behavior of the ODE. When δ is large Ψtc approximates Newton's method with its increased rate of convergence. Using a hybrid of forward Euler and Newton allows us to satisfy the local convergence criteria for Newton at the same time as maintaining Newton's fast convergence rate. The choice of the δ update formula is very important for it must at once force δ to be small for early iterations and allow δ to grow at the right time to recover the benefits of Newton's quadratic convergence rate.

The Ψtc algorithm described lacks something seen in time-accurate solvers [57, 4, 27].

No steps are rejected. In a time-accurate solver we would reject iterates when some measure of local truncation error is too large. In SER Ψtc we just make δ smaller in the next iteration if the norm of F is increased. If δ is too large going into the Ψtc algorithm, we may lose stability, whereas this is not a problem in a standard time-accurate method because we would reduce δ before continuing. This highlights why the choice of a δ update formula and δ_0 is important. The formulas presented here may need modification to get the right performance out of them.

The assumptions necessary for the convergence results of [36] are the following:

Assumption 1.1. *Minimum assumptions for Ψtc:*

1. *There exists a solution u^* of F.*
2. *F is Lipschitz continuously differentiable in a neighborhood of u^*.*
3. *$F'(u^*)$ is nonsingular.*

Assumption 1.1 buys us local q-quadratic convergence [36, 35, 50]. With the following extra assumptions, we can guarantee convergence to steady state from our initial iterate [36], i.e. global convergence.

Assumption 1.2. *Standard assumptions for Ψtc:*

1. *The SER δ update formula is used.*
2. *F is everywhere defined and Lipschitz continuously Fréchet differentiable, and there is an $M > 0$ such that $\|F'(u)\| \leq M$ for all u.*

3. *There is a $\eta > 0$ such that if $\|w - u_0\| < \eta$ then the solution of the initial value problem*

$$u' = -V^{-1}F(u), \quad u(0) = w$$

converges to u^* as $t \to \infty$.

4. *There are $\epsilon, \beta > 0$ such that if $\|u - u^*\| < \epsilon$ then $\|I + \delta V^{-1}F'(u)^{-1}\| \leq (1 + \beta\delta)^{-1}$ for all $\delta \geq 0$.*

Note that Assumption 1.2(3) guarantees that the problem is well conditioned with respect to its initial iterate.

1.3 Euler Equations

The Euler equations are equations of inviscid gas dynamics and are a simplification of the Navier-Stokes equations where the diffusive components have been eliminated [32]. The three-dimensional Euler flow through a volume Ω, enclosed by the surface $\partial\Omega$, in integral conservative formulation is defined by the following three equations [32]:

$$\frac{\partial}{\partial t}\int_\Omega \rho \, d\Omega + \oint_{\partial\Omega} \rho\vec{v} \cdot d\vec{S} = 0$$

$$\frac{\partial}{\partial t}\int_\Omega \rho\vec{v} \, d\Omega + \oint_{\partial\Omega} (\rho\vec{v} \times \vec{v} + p) \, d\vec{S} = \int_\Omega \rho\vec{f_e} \, d\Omega$$

$$\frac{\partial}{\partial t}\int_\Omega \rho E \, d\Omega + \oint_{\partial\Omega} \rho H\vec{v} \, d\vec{S} = \int_\Omega \rho\vec{f_e} \cdot \vec{v} \, d\Omega$$

Where, $\rho(t, \vec{x})$ is density, $p(t, \vec{x})$ is pressure, $\vec{v}(t, \vec{x})$ is velocity, $H(t, \vec{x})$ is stagnation

enthalpy, $E(t, \vec{x})$ is total energy, and $\vec{f_e}$ are the external forces. The first equation represents conservation of total mass, the second is conservation of total momentum, and the third is conservation of total energy. When written in Cartesian coordinates with velocity vector $\vec{v} = (u, v, w)$, the following system is derived:

$$\frac{\partial U}{\partial t} + \frac{\partial f}{\partial x} + \frac{\partial g}{\partial y} + \frac{\partial h}{\partial z} = Q$$

The components f, g, h are defined by

$$f(U) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho uH \end{bmatrix}, \quad g(U) = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho vH \end{bmatrix}, \text{and} \quad h(U) = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho wH \end{bmatrix},$$

with

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix} \text{and} \quad Q = \begin{bmatrix} 0 \\ f_{ex} \\ f_{ey} \\ f_{ez} \\ W_f \end{bmatrix}.$$

In this expression, $\vec{f} = (f_{ex}, f_{ey}, f_{ez})^T$ is an external force and $W_f = \vec{f_e} \cdot (\rho \vec{v})$ is the

work induced by this force. We use the ideal gas assumption:

$$p = (\gamma - 1)(\rho E - \frac{\rho u^2}{2}) \tag{1.5}$$

and thermodynamics to write down constitutive relations for total energy, pressure, and stagnation enthalpy. This assumption also introduces the physical constant γ, the ratio of the specific heat at constant pressure to the specific heat at constant volume for the gas. Air has a γ of approximately 1.4 [59], and that is the value we use in the work here. We also get an expression for the speed of sound:

$$c = \sqrt{\gamma \frac{p}{\rho}}.$$

The constitutive equations are:

$$\begin{aligned} E &= \frac{c^2}{\gamma(\gamma-1)} + \frac{u^2}{2}, \text{ and} \\ \rho H &= \rho E + p, \end{aligned}$$

where the second equation is the definition of stagnation enthalpy. We now linearize this system by applying the chain rule to each of the spatial derivatives. We then extract the three Jacobians $A = \partial f/\partial U, B = \partial g/\partial U$, and $C = \partial h/\partial U$ [32]:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ -u^2+\frac{\gamma-1}{2}\|\vec{v}\|^2 & (3-\gamma)u & -(\gamma-1)v & -(\gamma-1)w & \gamma-1 \\ -uv & v & u & 0 & 0 \\ -uw & w & 0 & u & 0 \\ -u\big(\gamma E-(\gamma-1)\|\vec{v}\|^2\big) & \gamma E-\frac{\gamma-1}{2}(\|\vec{v}\|^2+2u^2) & -(\gamma-1)uv & -(\gamma-1)uw & \gamma u \end{pmatrix}, \tag{1.6}$$

$$B = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ -uv & v & u & 0 & 0 \\ -v^2+\frac{\gamma-1}{2}\|\vec{v}\|^2 & -(\gamma-1)u & (3-\gamma)v & -(\gamma-1)w & \gamma-1 \\ -vw & 0 & w & v & 0 \\ -v\big(\gamma E-(\gamma-1)\|\vec{v}\|^2\big) & -(\gamma-1)uv & \gamma E-\frac{\gamma-1}{2}(\|\vec{v}\|^2+2v^2) & -(\gamma-1)vw & \gamma v \end{pmatrix}, \text{ and} \tag{1.7}$$

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ -uw & w & 0 & u & 0 \\ -vw & 0 & w & v & 0 \\ -u^2+\frac{\gamma-1}{2}\|\vec{v}\|^2 & -(\gamma-1)u & -(\gamma-1)v & -(\gamma-1)w & \gamma-1 \\ -w\big(\gamma E-(\gamma-1)\|\vec{v}\|^2\big) & -(\gamma-1)uw & -(\gamma-1)vw & \gamma E-\frac{\gamma-1}{2}(\|\vec{v}\|^2+2w^2) & \gamma w \end{pmatrix}. \tag{1.8}$$

Further development of the 3-D Euler equations will be explained as needed.

1.3.1 1D Euler Equations

Starting with the integral conservative formulation above, one can derive the governing equations for one-dimensional Euler flow through a nozzle with cross sectional area $S = S(x)$ [32]:

$$\begin{aligned} \frac{\partial(\rho S)}{\partial t} + \frac{\partial(\rho u S)}{\partial x} &= 0, \\ \frac{\partial(\rho u S)}{\partial t} + \frac{\partial\left[(\rho u^2 + p)S\right]}{\partial x} &= p\frac{dS}{dx}, \text{ and} \\ \frac{\partial(\rho E S)}{\partial t} + \frac{\partial(\rho u H S)}{\partial x} &= 0. \end{aligned}$$

In one dimension, U is,

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix}.$$

Let

$$F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ (\rho E + p)u \end{pmatrix} \quad \text{and} \quad W = \begin{pmatrix} 0 \\ p\frac{dS}{dx} \\ 0 \end{pmatrix}.$$

The system then becomes:

$$SU_t + (SF)_x = W.$$

This system is studied further in section 3.4.

1.4 Radiative Transfer Problem

Consider a unit square block of matter to which we apply a source of radiative energy at the center. This energy will transfer into the material and heat it diffusively. We will model the material energy and the radiative energy produced by the source and its energy-material transfer. Our model for radiation diffusion is a simplified version of the full radiation transport equations [51] and is the same problem used in [11, 10]. Please see [7] for more in-depth details. In this formulation, we have assumed isotropic radiation, i.e. no angular dependence, Fick's Law of diffusion, no scattering effects, and that the photon energy is Planckian. The equations are then integrated over frequency to get the diffusion driven model:

$$\frac{\partial E_R}{\partial t} = \nabla \cdot (K(E_R)\nabla E_R) \; + c\rho\kappa_P(T_M)\left(aT_M^4 - E_R\right) + \chi(\mathbf{x})caT_{\text{source}}^4 \qquad (1.9)$$

where $E_R(\mathbf{x}, t)$ is the radiation energy density ($\mathbf{x} = (x, y, z)$), $T_M(\mathbf{x}, t)$ is the material temperature, $\rho(\mathbf{x})$ is the material density, c is the speed of light, and $a = 4\sigma/c$ where σ is the Stephan-Boltzmann constant. The Rosseland opacity, κ_R, is a nonlinear function of the radiation temperature, T_R, which is defined by the relation $E_R = aT_R^4$. The Planck opacity, κ_P, is a nonlinear function of material temperature, T_M, which is related to the material energy through an equation of state, $E_M = \text{EOS}(T_M)$. The last term represents a spatially dependent source term which produces sources or sinks in the radiation field.

The K used in [11, 10] is scalar but nonlinear and incorporates a flux-limiter to prohibit computed solutions from resulting in photon velocities exceeding the speed of light [7],

$$K(E_R) = \frac{c}{3\rho\kappa_R(T_R) + \frac{\|\nabla E_R\|}{E_R}}. \tag{1.10}$$

Equation (1.9) is coupled to an equation expressing conservation of material energy,

$$\frac{\partial E_M}{\partial t} = -c\rho\kappa_P(T_M)\left(aT_M^4 - E_R\right). \tag{1.11}$$

The equations (1.9) and (1.11) represent the system of equations we will solve using the methods to be described in the following sections. The original solution code, `tranpsort3d` (see code 1.5), incorporates Dirichlet, Neumann, and Robin boundary conditions for the system.

1.5 Computer Codes

The following codes were used in this research.

Computer Code 1.1. nsol *Newton root-finder with direct linear solver in* `Matlab` *[35] and maintained by Tim Kelley. (http://www4.ncsu.edu/∼ctk/matlab_roots.html)*

Computer Code 1.2. nsola *Newton root-finder with iterative linear solver in* `Matlab` *[35] and maintained by Tim Kelley. (http://www4.ncsu.edu/∼ctk/matlab_roots.html)*

Computer Code 1.3. fish2d *Fast Poisson solver in 2D based on* `Matlab` `fft` *and maintained by Tim Kelley. Assumes square geometry and homogeneous Dirichlet boundary conditions. (http://www4.ncsu.edu/∼ctk/matlab_roots.html)*

Computer Code 1.4. CVode *Time accurate parallel ODE integrator maintained by the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory [13]. (http://www.llnl.gov/casc/sundials/)*

Computer Code 1.5. transport3d *An in-house development code used at the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory for radiative transfer problems which uses* `CVode` *as the integrator.*

Computer Code 1.6. hypre *Scalable software for solving large sparse linear systems of equations (*`hypre` *stands for high performance preconditioners) developed by the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory [26]. (http://www.llnl.gov/CASC/hypre/)*

Computer Code 1.7. PETSc *The Portable, Extensible Toolkit for Scientific Computation developed by the Mathematics and Computer Science division at Argonne National Laboratory [5]. (http://www-fp.mcs.anl.gov/petsc/)*

Computer Code 1.8. ADIC *Automatic Differentiation for the C language developed by the Mathematics and Computer Science division at Argonne National Laboratory [33]. (http://www-unix.mcs.anl.gov/autodiff/)*

Computer Code 1.9. Matlab *Numeric Computation and Visualization Software developed by The MathWorks (http://www.mathworks.com/).*

Computer Code 1.10. roe *One dimensional Euler equation code for flow in a nozzle developed by Jeff McMullan in the Mechanical and Aerospace Engineering department at North Carolina State University as part of his Ph.D. dissertation [45].*

Computer Code 1.11. LaTeX *A document preparation system built on top of TeX (http://www.latex-project.org/) [41].*

Computer Code 1.12. TeX *A typesetting program designed for high-quality composition of material that contains a lot of mathematical and technical expressions [40].*

Computer Code 1.13. gcc *GNU C compiler developed by the GNU project (http://www.gnu.org)*

Computer Code 1.14. vi *VI text editor, the one true faith. (http://ex-vi.berlios.de/)*

Computer Code 1.15. vim *VI Improved (http://vim.sourceforge.net/)*

Computer Code 1.16. bitkeeper *Distributed source management software that I used for this document (http://www.bitkeeper.com)*

Chapter 2

Illustrative Examples

In our computational experiments we let $\delta_{max} = \infty$. This leaves us with the choice of δ_0. Conventional wisdom says to make δ_0 small, so the iteration is accurate in the beginning but not so small that you spend too many iterations growing δ_n. When using the TTE δ update formula, we have followed the example in [36] and set $\tau = \frac{3}{4}$.

The linear equation

$$(\delta^{-1}V + F'(x))s = -F(x) \tag{2.1}$$

for the Newton step that is solved in step 1.2 of the Ψtc algorithm (Algorithm 1.2), is the discretization of a PDE and is usually very large. As such, it is typically solved by an iterative method which terminates on small linear residuals. In this numerical study, two approaches were taken to solving (2.1). The first was to use the finite difference GMRES code `fdgmres` in [35] to solve (2.1) iteratively. The other was to form the Jacobian using finite differences, and then do a direct solve to compute s in (2.1). The code `fdgmres`

calculates a forward difference approximation to the directional derivative of $F(x)$ (using a fixed Δx of 10^{-7}) for each linear iteration of the GMRES algorithm. This greatly reduces the computational cost of the Newton step since, in most cases, there are far fewer GMRES iterations than there are columns in the Jacobian.

To gauge the effectiveness of the iterative version of this algorithm, we used the line search code `nsola` from [35] (see code 1.2). This code utilizes `fdgmres` and is a globally convergent Newton-Krylov solver for $F(x) = 0$ using an inexact-Newton-Armijo iteration, an Eisenstat-Walker forcing term [24, 25], and a parabolic line search via three point interpolation. In all the experiments performed here, the default parameters behaved well and there was no need to adjust any parameters in the `nsola` code.

To gauge the effectiveness of the direct solver version, we used the Newton's method code `nsol` in [35] (modified to use an Armijo step-length). We also used all the default parameters for this code.

These codes are written in `Matlab` where development tools allow easy modification and testing of different problems. The first few test problems are scalar. The first vector problem is a two point boundary value problem followed by a convection-diffusion PDE and Burgers' equation [59, 12, 52]. For most of these examples, we set $V = I$. A scaling matrix for the nonlinear function F could be placed in V.

We observed three principle failure modes for Ψtc. The first is "divergence", which occurs when δ_n goes to zero because the norm of F is going to infinity. The second, is a "cycle", which occurs when the algorithm alternates between two or more x values,

but does not settle down to either. The last is "stagnation" which occurs when the code gets stuck at a non-root. In some cases, more nonlinear iterations may allow the code to terminate, but in most cases, the ratio of successive norms of F, is larger than one, and decreasing at an ever slower rate. This suggests that the iteration may never terminate. The catch-all failure is "max_it", which occurs when the algorithm hits a hard limit for the maximum number of nonlinear iterations.

2.1 Scalar Problems

The problems in the following section are very simple root-finding problems which Ψtc might be applied to. The theory for Ψtc only applies to ordinary differential equations with well defined steady-state solutions. These examples do not satisfy that assumption as stated, but they illustrate how Ψtc behaves on problems for which it wasn't designed. Surprisingly, Ψtc does fairly well on these root-finding problems, but it does have its drawbacks.

2.1.1 $F(x) = \arctan(x)$

The arctan function has a root at $x = 0$, and we start our iterations at $x_0 = 10$. The iterations stop when $|\arctan(x)| < 10^{-10}$ or 5000 iterations occur.

Table 2.1 shows the computational results for all three δ update formulas. Note that for the SER δ update formula, a small δ_0 really slows down the iterations. This is the price for integration that is closer to time-accurate. But with a large δ_0, the code enters

Table 2.1: Computational results for $F(x) = \arctan(x)$.

code	δ update	δ_0	# its	Fevals	error	fail
Ψtc	SER	0.01	689	690	2.40e-15	0
		0.1	75	76	4.51e-15	0
		1	12	13	1.18e-20	0
		10	5000	5001	5.49e+00	orbit
		100	5000	5001	7.22e+01	orbit
Ψtc	SSF	0.01	16	17	4.44e-11	0
		0.1	16	17	4.43e-12	0
		1	15	16	6.92e-15	0
		10	10	11	1.76e-17	0
		100	69	70	6.62e-12	0
Ψtc	TTE	0.01	15	16	7.38e-12	0
		0.1	16	17	1.10e-12	0
		1	14	15	2.39e-13	0
		10	10	11	1.15e-12	0
		100	17	18	4.80e-13	0
`nsol`			12	23	0.00e+00	0

a stable orbit with x alternating between a positive and negative value. For example, when $\delta_0 = 10$, x settles into an orbit of $\{\pm 5.4909\}$ and a fixed δ of 10.58. Neither of these problems arise when using the other two δ update formulas.

This is a theoretical problem with the algorithm for a very broad class of functions. Consider that for each x_0 we can pick an SER δ_0 so that $x_1 = -x_0$:

$$\delta_0 = \left(\frac{F(x_0)}{2x_0} - F'(x_0) \right)^{-1}. \tag{2.2}$$

For $|x_0| \geq 3$ a stable orbit forms in finite precision arithmetic, but for small x_0 the orbit is unstable, and Ψtc converges to the solution. We cannot guarantee the development of an orbit in finite precision arithmetic, but we can say something about the infinite

precision case.

Theorem 2.1. *Let $F(x)$ be an odd continuous function with continuous first derivative. Let x_0 be any nonzero real number such that $F(x_0) \neq 0$. And let (2.2) define δ_0. Then the Ψtc algorithm with the SER δ update formula in infinite precision arithmetic will alternate x_n between $\pm x_0$, and $\delta_n = \delta_0$ for all n.*

Proof. We will use induction on n. First we establish that the result holds for $n = 1$.

$$\begin{aligned} x_1 &= x_0 - \left(\delta_0^{-1} + F'(x_0)\right)^{-1} F(x_0) \\ &= x_0 - \left(\tfrac{F(x_0)}{2x_0} - F'(x_0) + F'(x_0)\right)^{-1} F(x_0) \\ &= x_0 - \tfrac{2x_0}{F(x_0)} F(x_0) \\ &= -x_0. \end{aligned}$$

This shows $x_1 = -x_0$, now we will show $\delta_1 = \delta_0$.

$$\begin{aligned} \delta_1 &= \delta_0 |F(x_0)|/|F(x_1)| \\ &= \delta_0 |F(x_0)|/|F(-x_0)| \\ &= \delta_0 |F(x_0)|/|-F(x_0)| \\ &= \delta_0. \end{aligned}$$

For the induction step, we still need δ_1 to have the form of (2.2).

$$\begin{aligned}\delta_1 &= \left(\tfrac{F(x_0)}{2x_0} - F'(x_0)\right)^{-1} \\ &= \left(\tfrac{-F(x_1)}{-2x_1} - F'(-x_1)\right)^{-1} \\ &= \left(\tfrac{F(x_1)}{2x_1} - F'(x_1)\right)^{-1}.\end{aligned}$$

This is true since $F'(x)$ is even.

Now assume $x_n = -x_{n-1}$ and $\delta_n = \left(\tfrac{F(x_n)}{2x_n} - F'(x_n)\right)^{-1}$.

Show this holds for $n+1$:

$$\begin{aligned}x_{n+1} &= x_n - \left(\delta_n^{-1} + F'(x_n)\right)^{-1} F(x_n) \\ &= x_n - \tfrac{2x_n}{F(x_n)} F(x_n) \\ &= -x_n.\end{aligned}$$

And exactly the same as before,

$$\begin{aligned}\delta_{n+1} &= \delta_n |F(x_n)|/|F(x_{n+1})| \\ &= \delta_n |F(x_n)|/|-F(x_n)| \\ &= \delta_0.\end{aligned}$$

$$\begin{aligned}\delta_{n+1} &= \left(\tfrac{F(x_n)}{2x_n} - F'(x_n)\right)^{-1} \\ &= \left(\tfrac{F(x_1)}{2x_1} - F'(x_1)\right)^{-1}.\end{aligned}$$

□

The numerical code makes many floating point approximations, and these disturbances may cause the code to move x away from the orbit predicted by infinite precision arithmetic.

Table 2.1 also shows the SSF and TTE δ update formula results. We attempted to find a similar stable orbit as before, but were not able to do so. For this problem, both the SSF and TTE δ update formulas performed much better over-all. When `nsol` was applied to this problem with the same tolerance and maximum iterations it found the exact answer very quickly. But the Ψtc code was just as fast if not faster for carefully chosen δ_0.

2.1.2 $F(x) = \cos(x) - x$

This function is monotone decreasing and has a root near $x = 0.74$. We started with $x_0 = 0.5$, tolerance $= 10^{-10}$, and a maximum of 5000 iterations. As seen in Table 2.2 the SER δ update formula behaves very poorly when $\delta_0 \leq 1$. Contrast this with the SSF δ update formula also seen in Table 2.2, where even very small δ_0's yield quick solutions. And the TTE δ update is only slightly better than the SER δ update. For the SER δ update formula, with $\delta_0 <= 1$, the iterations simply march toward $-\infty$. This may seem strange, but analysis of the algorithm predicts this. We can state a general result about the above failure of the Ψtc code.

Theorem 2.2. *Let $F : \mathbb{R} \to \mathbb{R}$ be monotone decreasing and Lipschitz continuous with Lipschitz constant γ. Let $F(x^*) = 0$, $x_0 \leq x^*$, and $\delta_0 \leq \frac{1}{\gamma}$ in the Ψtc algorithm. If $\{x_n\}$*

Table 2.2: Computational results for $F(x) = \cos(x) - x$. For the SER δ update formula, and the first three δ_0 lines, the algorithm attempts to take x_n to $-\infty$.

code	δ update	δ_0	# its	Fevals	rel error	fail
Ψtc	SER	0.01	5000	5001	2.59e+01	diverge
		0.1	5000	5001	2.56e+02	diverge
		1	5000	5001	2.56e+03	diverge
		10	4	5	0	0
		100	4	5	0	0
Ψtc	SSF	0.01	6	7	0	0
		0.1	6	7	1.35e-14	0
		1	8	9	3.00e-16	0
		10	4	5	3.53e-14	0
		100	4	5	0	0
Ψtc	TTE	0.01	2153	2154	Inf	diverge
		0.1	1900	1901	Inf	diverge
		1	9	10	5.50e-14	0
		10	5	6	1.95e-12	0
		100	4	5	9.03e-12	0
`nsol`			4	5	0	0

is the sequence of SER Ψtc iterates, then for all n, $x_n \leq x_0$ and $\delta_n \leq \delta_0$.

Proof. We will proceed by induction on n. First we show the result is true for $n = 1$:

$$x_1 = x_0 - (\delta_0^{-1} + F'(x_0))^{-1} F(x_0).$$

We know $F(x_0) \geq F(x^*) = 0$ and $F'(x_0) \leq 0$ because $F(x)$ is monotone decreasing. Therefore,

$$x_1 - x_0 = -(\delta_0^{-1} - |F'(x_0)|)^{-1} |F(x_0)|.$$

We can show

$$\delta_0^{-1} - |F(x_0)| \geq \gamma - |F'(x_0)| \geq 0$$

since $F(x)$ Lipschitz continuous with Lipschitz constant γ is equivalent to $|F'(x)|$ bounded by γ. Hence,

$$x_1 - x_0 = -\left|(\delta_0^{-1} - |F'(x_0)|)^{-1}\right| |F(x_0)| \leq 0.$$

Which shows that $x_1 \leq x_0$. And, $F(x_1) \geq F(x_0)$, so

$$\delta_1 = \delta_0 \frac{|F(x_0)|}{|F(x_1)|} \leq \delta_0.$$

Now we assume the result is true for n, i.e. $x_n \leq x_0$ and $\delta_n \leq \delta_0$ and show true for $n+1$:

$$x_{n+1} = x_n - (\delta_n^{-1} + F'(x_n))^{-1} F(x_n).$$

Again, $F(x_n) \geq F(x_0) \geq F(x^*) = 0$ and $F'(x_n) \leq 0$, so,

$$x_{n+1} - x_n = -(\delta_n^{-1} - |F'(x_n)|)^{-1}|F(x_n)|.$$

And $\delta_n^{-1} \geq \delta_0^{-1} \geq \gamma$ so,

$$\delta_n^{-1} - |F'(x_n)| \geq \gamma - |F'(x_n)| \geq 0.$$

As before,

$$x_{n+1} - x_n = -\left|(\delta_n^{-1} - |F'(x_n)|)^{-1}\right| |F(x_n)| \leq 0$$

so $x_{n_1} \leq x_n$, which implies $F(x_{n+1}) \geq F(x_n)$, and $\delta_{n+1} \leq \delta_n \leq \delta_0$.

□

The choice of $x_0 = \frac{1}{2}$ and $\delta_0 \approx 0.1$ above was a reasonable initial choice because x_0 is fairly close to the solution and δ_0 is smaller than the difference between x_0 and x^*. As it turns out, for this particular problem, any $x_0 < x^*$ with δ_0 sufficiently small will cause SER Ψtc to fail. It is interesting that $\delta_0 = 1.0$ fails in the SER δ update though, since $F(x)$ is Lipschitz continuous with Lipschitz constant 2. But δ_n^{-1} just needs to be larger than $|F'(x_n)|$ to move x_{n+1} further away from the root. The SSF δ update formula does not *obviously* have this problem. We believe a similar analysis to above cannot be completed for this δ update formula. In order for δ_n to continue to be reduced, the step $|x_{n+1} - x_n|$, must be strictly larger than 1. This condition makes it very difficult to get

the same behavior as in the SER δ update formula. Apparently `nsol` is not as susceptible to this problem since it was able to return an answer very quickly with out any failures. Without user intervention, `nsol` is the clear winner here.

2.1.3 $F(x) = \sin(x)$

This example is very interesting because when we start SER Ψtc at $x_0 = 3$ with $\delta_0 = 0.1$, the code marches x over the hump and down to $x = 0$. This is not what a descent method would do. And when δ_0 becomes larger, the code can actually jump to a root much farther away, see $\delta_0 = 1$ in Table 2.3. We used a tolerance of 10^{-10} and a maximum of 5000 iterations for this problem.

When `nsol` solved this problem, it found the root $x = \pi$ with 3 nonlinear iterations and 4 function calls. In order to get similar performance out of the Ψtc code, δ_0 must be quite large. And especially for the SER δ update formula, a small δ_0 performs very poorly.

2.1.4 $F(x) = x^2$

This is an extremely simple problem that both algorithms solve very easily, which is interesting because this problem doesn't satisfy the theory for either method. For Ψtc, the derivative is not bounded, and for both, the derivative is zero at the root. As before, δ_0 must be quite large in order to get the same performance in Ψtc as in `nsol`. We used $x_0 = 0.5$, tolerance of 10^{-10}, and a maximum of 5000 iterations.

Table 2.3: Computational results for $F(x) = \sin(x)$. Instead of failure column, we have a root column. This column shows which root the iterations were converging to.

code	δ update	δ_0	# its	Fevals	rel error	root
Ψtc	SER	0.01	2135	2136	1.49e-15	0
		0.1	219	220	9.37e-12	0
		1	18	19	1.27e-15	-4π
		10	4	5	1.41e-16	π
		100	3	4	3.96e-15	π
Ψtc	SSF	0.01	5	6	4.78e-14	π
		0.1	5	6	2.65e-13	π
		1	8	9	1.82e-12	-4π
		10	4	5	1.70e-15	π
		100	3	4	3.85e-12	π
Ψtc	TTE	0.01	9	10	1.51e-12	π
		0.1	8	9	8.06e-15	π
		1	8	9	7.48e-13	-4π
		10	5	6	1.24e-14	π
		100	4	5	2.36e-14	π
`nsol`			3	4	0	π

Table 2.4: Computational results for $F(x) = x^2$.

code	δ update	δ_0	# its	Fevals	error
Ψtc	SER	0.01	219	220	6.30e-06
		0.1	38	39	5.34e-06
		1	19	20	5.37e-06
		10	16	17	9.27e-06
		100	16	17	7.78e-06
Ψtc	SSF	0.01	18	19	5.02e-06
		0.1	17	18	9.51e-06
		1	17	18	6.90e-06
		10	16	17	8.76e-06
		100	16	17	7.75e-06
Ψtc	TTE	0.01	19	20	8.73e-06
		0.1	19	20	6.34e-06
		1	18	19	6.12e-06
		10	17	18	5.19e-06
		100	16	17	7.93e-06
`nsol`			16	17	7.63e-06

Table 2.4, shows that the results for this problem are about the same across the board. Only the SER δ update formula performs poorly, and then only when δ_0 is small. When x_0 is closer to 0, SER performs worse but SSF and TTE perform better. And the reverse is true when x_0 is further away from 0.

2.2 Some Vector Problems

2.2.1 Two-Point BVP

Problem 5.7.25 of [35] suggests the following two-point boundary value problem.

$$-u'' = \sin(u) + f(x), \quad u(0) = u(1) = 0,$$

where

$$f(x) = 2 - \sin(x(1-x))$$

which gives a solution of

$$u^*(x) = x(1-x).$$

We discretized the interval [0,1] into N points and described u on the interior, using the known boundary points when necessary. We used centered difference approximations for u''. In the following examples we used: $N = 100$, $u_0 = 0$, and a tolerance of 10^{-10}. This initial condition satisfies the boundary conditions trivially. We used either no preconditioner or the inverse of the second order centered difference operator as a preconditioner in this problem, i.e.

$$\begin{pmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{pmatrix}^{-1}.$$

This is the first case where the SSF update formula has not performed better than

Table 2.5: Computational results for the 2-point BVP with no preconditioner. In the two SER δ update failures, δ_n converged to 0, indicating a failure similar to the $cos(x) - x$ problem.

code	δ update	δ_0	# its	Fevals	rel error	fail
Ψtc	SER	0.01	73	7228	7.09e+00	diverge
		0.1	95	9406	7.52e+01	diverge
		1	4	397	2.21e-02	0
		10	3	298	2.21e-02	0
		100	3	298	2.21e-02	0
Ψtc	SSF	0.01	57	5644	2.14e+02	diverge
		0.1	61	6040	1.82e+02	diverge
		1	4	397	2.21e-02	0
		10	4	397	2.21e-02	0
		100	3	298	2.21e-02	0
Ψtc	TTE	0.01	578	57223	Inf	diverge
		0.1	570	56431	Inf	diverge
		1	6	595	2.21e-02	0
		10	4	397	2.21e-02	0
		100	4	397	2.21e-02	0
`nsol`			4	103	2.21e-02	0

the SER update. In both formulas, small δ_0 causes a serious failure of the algorithm where $\delta_n \to 0$. This is the same error condition as $\cos(x) - x$. For this problem, `nsol` was able to get by with 4 nonlinear iterations, and 103 function calls.

Table 2.6: Computational results for the 2-point BVP with preconditioner as described above.

code	δ update	δ_0	# its	Fevals	rel error	fail
Ψtc	SER	0.01	3	298	2.21e-02	0
		0.1	3	298	2.21e-02	0
		1	3	298	2.21e-02	0
		10	3	298	2.21e-02	0
		100	3	298	2.21e-02	0
Ψtc	SSF	0.01	4	397	2.21e-02	0
		0.1	3	298	2.21e-02	0
		1	3	298	2.21e-02	0
		10	3	298	2.21e-02	0
		100	3	298	2.21e-02	0
Ψtc	TTE	0.01	5	496	2.21e-02	0
		0.1	4	397	2.21e-02	0
		1	4	397	2.21e-02	0
		10	3	298	2.21e-02	0
		100	3	298	2.21e-02	0
`nsol`			4	103	2.21e-02	0

As Table 2.6 shows, N was too small to tax this problem when the preconditioner is applied. All three δ update rules and all choices of δ_0 did about the same with `nsol` the clear winner in terms of number of function calls.

2.2.2 Convection-Diffusion PDE

Section 6.4.2 of [35] contains the following steady convection-diffusion equation:

$$-\nabla^2 u + Cu(u_x + u_y) = f,$$

with homogeneous Dirichlet boundary conditions on the unit square $(0,1) \times (0,1)$ where f has been constructed so that the exact solution was the discretization of the function,

$$10xy(1-x)(1-y)e^{x^{4.5}}.$$

We used $C = 20$, $N = 31$, $u_0(x,y) = 0$, and a tolerance of 10^{-10}. In the first set of results, no preconditioner was used. In second set of results, the fast Poisson solver `fish2d` was used as a preconditioner. This code solves $-\nabla^2 v = g$ for v. In this case $g = f - Cu(u_x + u_y)$.

Table 2.7 shows the computational results without a preconditioner, and Table 2.8 shows the results with the `fish2d` preconditioner. Without the preconditioner, all the δ choices are about the same, and `nsol` performs very similarly to Ψtc, although a bit faster. With the preconditioner, a small δ_0 slows down SER but does not affect the other δ update rules too much. And, as expected, the preconditioner reduces the total cost of computing the solution considerably.

Table 2.7: Computational results for convection-diffusion PDE with iterative linear algebra but without a preconditioner.

code	δ update	δ_0	# its	Fevals	rel error	fail
Ψtc	SER	0.01	8	312	3.23e-14	0
		0.1	7	304	1.04e-13	0
		1	7	366	5.10e-15	0
		10	7	363	7.11e-15	0
		100	7	361	9.37e-15	0
Ψtc	SSF	0.01	8	303	2.65e-14	0
		0.1	7	319	7.59e-14	0
		1	7	363	6.27e-15	0
		10	7	362	1.06e-14	0
		100	7	361	9.37e-15	0
Ψtc	TTE	0.01	9	318	1.10e-13	0
		0.1	8	349	1.92e-14	0
		1	7	369	1.17e-14	0
		10	7	365	5.02e-15	0
		100	7	362	7.70e-15	0
`nsola`			5	289	5.80e-10	0

Table 2.8: Computational results for convection-diffusion PDE with iterative linear algebra and the `fish2d` preconditioner.

code	δ update	δ_0	# its	Fevals	rel error	fail
Ψtc	SER	0.01	116	418	5.13e-11	0
		0.1	17	103	5.80e-11	0
		1	7	60	1.26e-12	0
		10	6	54	2.18e-12	0
		100	6	52	9.05e-12	0
Ψtc	SSF	0.01	8	63	2.80e-13	0
		0.1	8	64	1.29e-13	0
		1	7	60	3.46e-13	0
		10	6	54	3.66e-12	0
		100	6	52	9.34e-12	0
Ψtc	TTE	0.01	10	76	5.70e-12	0
		0.1	10	82	1.68e-13	0
		1	8	69	7.34e-12	0
		10	7	63	9.23e-13	0
		100	6	52	1.64e-11	0
`nsola`			5	46	6.91e-10	0

2.3 Burgers' Equation

Burgers' equation of [59, 12, 52] is

$$u_t + (c + bu)u_x = \mu u_{xx}. \tag{2.3}$$

According to [59] this form has the stationary solution

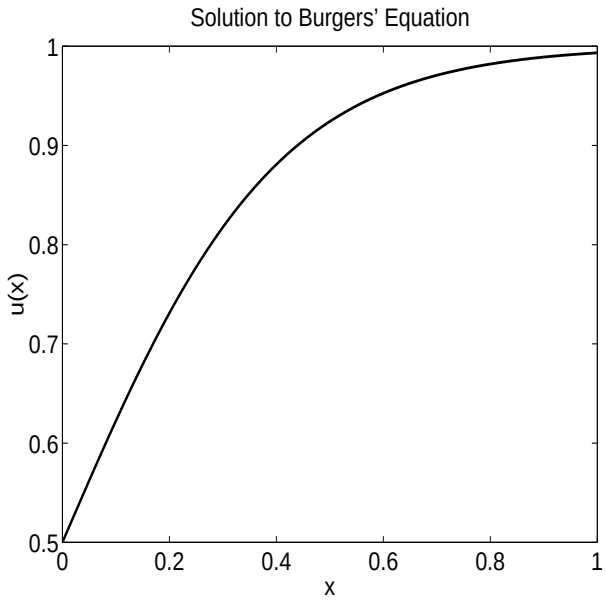
$$u(x) = -\frac{c}{b}\left(1 + \tanh\left(\frac{c(x - x_0)}{2\mu}\right)\right). \tag{2.4}$$

Figure 2.1 shows a typical plot of (2.4) when $x_0 = 0$, $b = -1$, $c = \frac{1}{2}$, and $\mu = 0.1$. As μ becomes smaller, the transition becomes more vertical, and in the limit as $\mu \to 0$, a shock forms at $x = x_0$.

The parameter x_0 in (2.4) handles the fact that Burgers' equation has an infinite number of solutions which differ only by translations along the x-axis. In order to obtain a unique solution, we set up one of our boundary points at $x = 0$, this allows us to specify $x_0 = 0$. To set (2.3) up to solve with Ψtc, we dropped u_t and used forward differences for u_x and centered differences for u_{xx} on the interval $[0, 1]$, with the boundary conditions: $u(0) = \frac{1}{2}$ and $u(1) = 1$. Let N be the number of mesh points in the interval $(0, 1)$ with $h = 1/(N + 1)$. Then,

$$u_i = u(ih), \quad i = 1, \cdots, N \quad \text{and}$$
$$u_0 = 0.5, \quad u_{N+1} = 1 \qquad .$$

Figure 2.1: Plot of equation (2.4) for $x_0 = 0$, $b = -1$, $c = \frac{1}{2}$, and $\mu = 0.1$.



And our function is:

$$F(u)_i = (c + bu_i)\frac{u_{i+1} - u_i}{h} - \mu\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}.$$

Note the use of centered differencing for u_{xx} and forward differencing for u_x. In [59], this is called upwinding, and improves the stability of the numerical method by producing a diagonally dominant iteration matrix. If we were solving this problem on a domain including both sides of x_0, then we would have to deal with a more sophisticated version of upwinding to switch the difference direction on each side of x_0.

There are two variations of our Ψtc code: one where the linear Newton step is computed using a finite difference GMRES solver, and another where it is computed using a

finite difference LU factorization. In the second case, sparse matrices were used to take advantage of the tri-diagonal structure of the Jacobian. Similarly, we have a competitor for the direct method. This one uses the modified `nsol`, code 1.1, [35], which adds an Armijo line search to give global convergence properties.

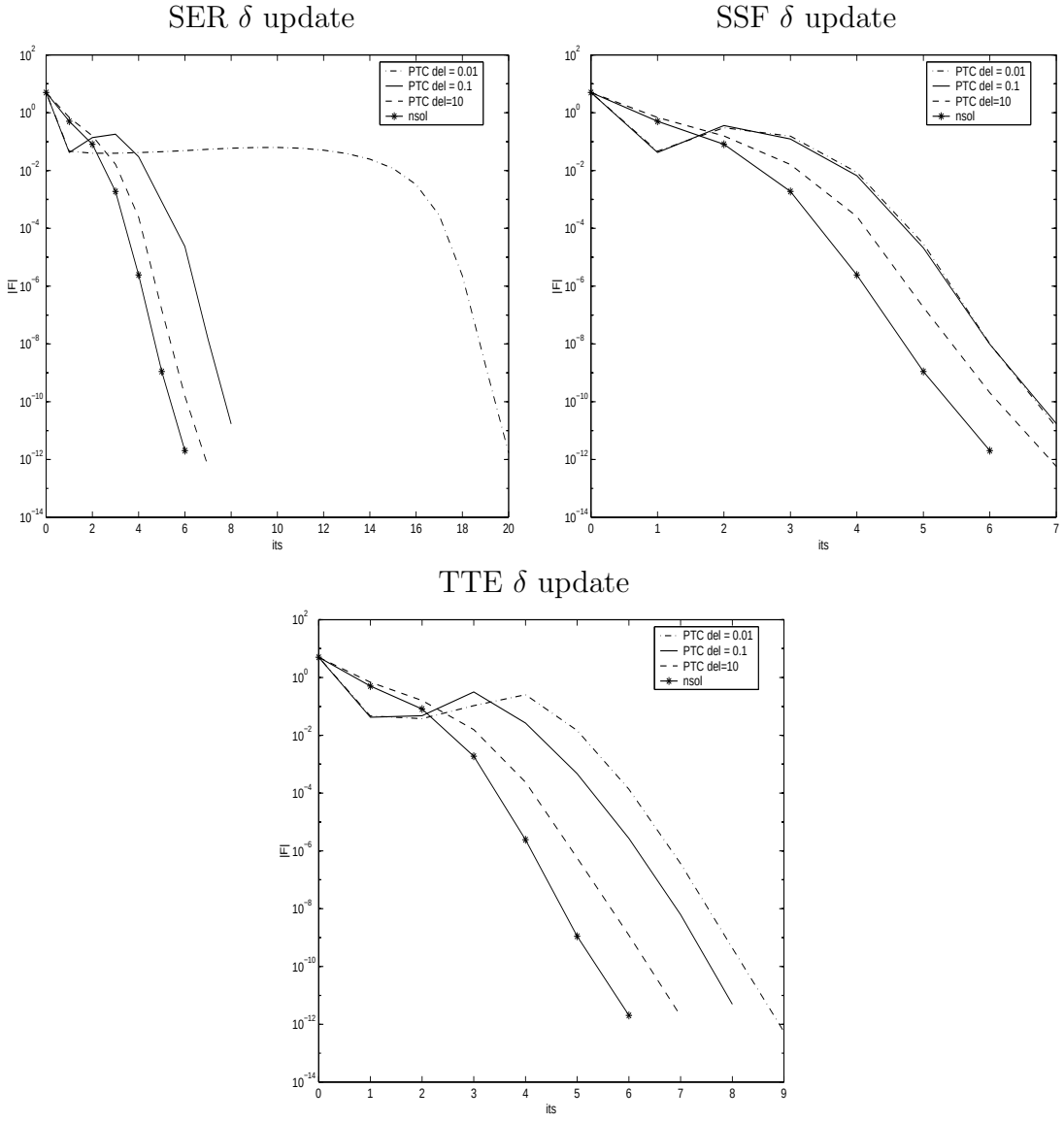
In the following results, $c = 0.5$, $b = -1$ and $N = 1000$ with initial conditions that satisfy the boundary conditions (i.e. a straight line between). Numerical computations are shown for four different μ values. The first is $\mu = 10^{-2}$, for which all the methods do well. The next is $\mu = 10^{-3}$, where Ψtc takes a bit of a lead. And at $\mu = 10^{-4}$, `nsol` really has difficulties. For $\mu = 0$, `nsol` beat Ψtc, but this problem isn't well posed as presented here.

Table 2.9: Burgers' Equation: Direct linear solvers with $\mu = 10^{-2}$.

code	δ update	δ_0	# its	Fevals	rel error	fail
Ψtc	SER	0.01	20	20021	4.1e-03	0
		0.10	8	8009	4.1e-03	0
		10.00	7	7008	4.1e-03	0
Ψtc	SSF	0.01	7	7008	4.1e-03	0
		0.10	7	7008	4.1e-03	0
		10.00	7	7008	4.1e-03	0
Ψtc	TTE	0.01	9	9010	4.1e-03	0
		0.10	8	8009	4.1e-03	0
		10.00	7	7008	4.1e-03	0
`nsol`			6	6008	4.1e-03	0

Note that in Figure 2.2, with $\mu = 10^{-2}$, all the knobs give solutions, and `nsol` returns the fastest solution. This μ value gives a fairly rounded transition which does not give

Figure 2.2: Computational results for Burgers' equation with $\mu = 10^{-2}$, and direct linear solvers.



either method much difficulty.

Table 2.10: Burgers' Equation: Direct linear solvers with $\mu = 10^{-3}$.

code	δ update	δ_0	# its	Fevals	rel error	fail
Ψtc	SER	0.01	275	275276	3.4e-02	0
		0.10	25	25026	3.4e-02	0
		10.00	14	14015	3.4e-02	0
Ψtc	SER2	0.01	8	8009	3.4e-02	0
		0.10	8	8009	3.4e-02	0
		10.00	8	8009	3.4e-02	0
Ψtc	SER3	0.01	10	10011	3.4e-02	0
		0.10	9	9010	3.4e-02	0
		10.00	8	8009	3.4e-02	0
`nsol`			23	23098	3.4e-02	0

The story is considerably different when $\mu = 10^{-3}$. As Figure 2.3 shows, when δ_0 is too small and the SER δ update formula is used, the results are very slow. But notice the striking improvement with the second δ update formula.

With $\mu = 10^{-4}$, `nsol` is much worse than Ψtc. Forgiving the poor results when δ_0 is too small and the SER δ update formula is used, see Figure 2.4, Ψtc does a fantastic job and `nsol` is quite a bit slower. Figure 2.5 shows how close the computed solution is to the exact solution. When centered differences were used for u_x, a large bump occurred here. Also notice that the computed curve is more gradual than the exact curve. This may be due to the increased diffusion present in the forward Euler approximation to the first derivative.

When $\mu = 0$, we get failures in the Ψtc code for $\delta_0 = 10$. This was expected to happen

Figure 2.3: Computational results for Burgers' equation with $\mu = 10^{-3}$, and direct linear solvers.

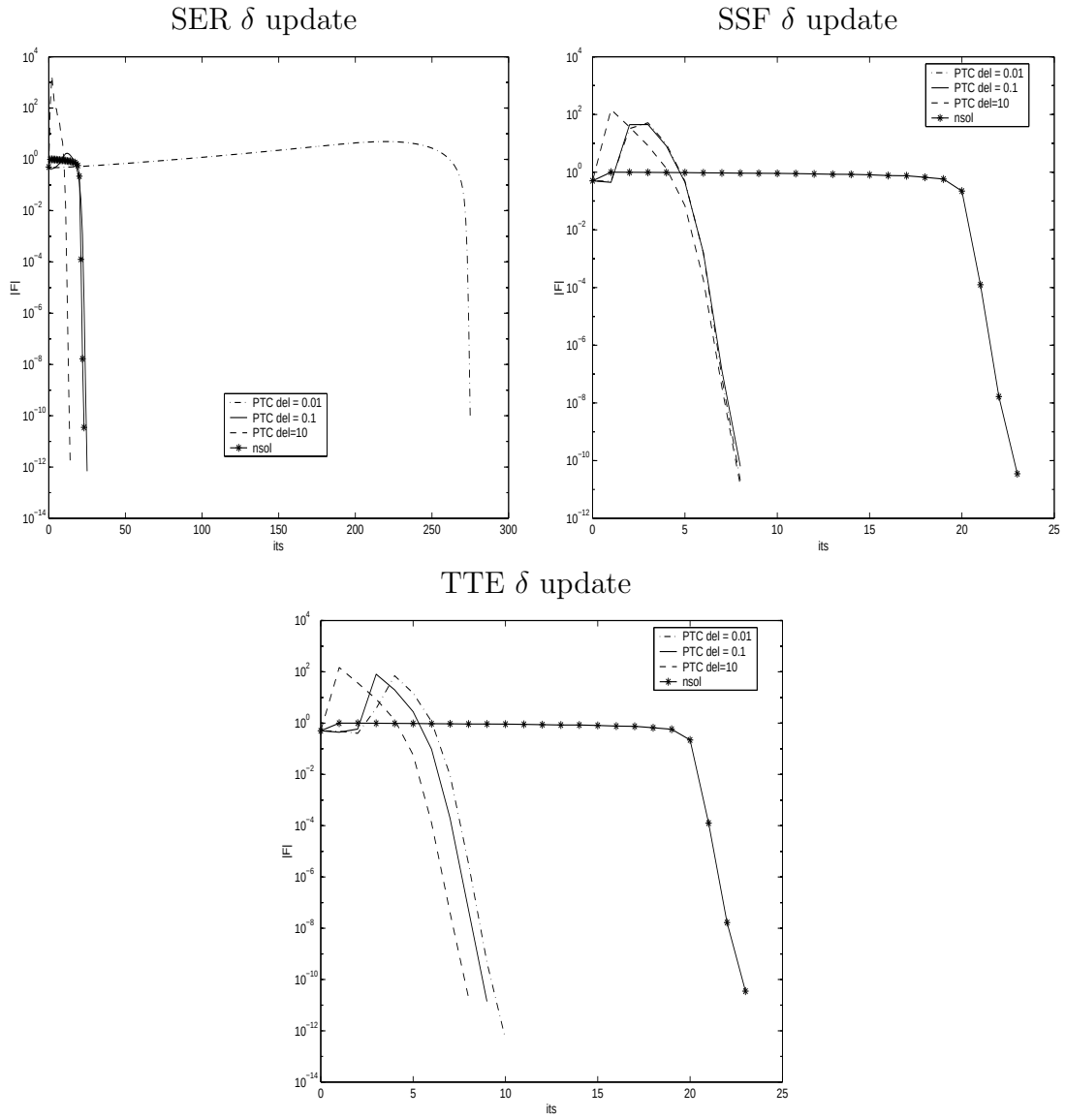


Figure 2.4: Computational results for Burgers' equation with $\mu = 10^{-4}$, and direct linear solvers.

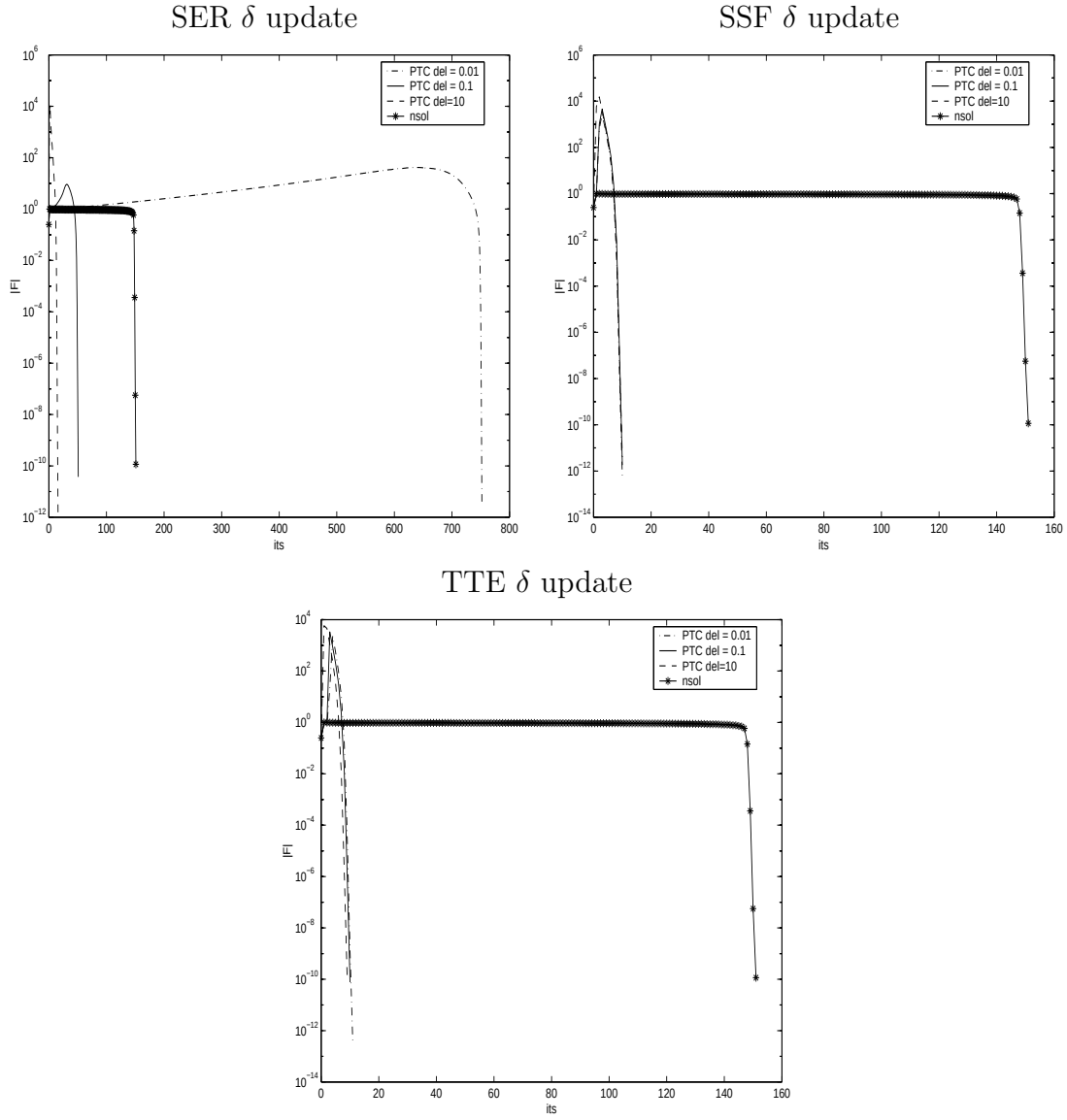


Table 2.11: Burgers' Equation: Direct linear solvers with $\mu = 10^{-4}$.

code	δ update	δ_0	# its	Fevals	rel error	fail
Ψtc	SER	0.01	752	752753	9.0e-02	0
		0.10	51	51052	9.0e-02	0
		10.00	16	16017	9.0e-02	0
Ψtc	SSF	0.01	10	10011	9.0e-02	0
		0.10	10	10011	9.0e-02	0
		10.00	10	10011	9.0e-02	0
Ψtc	TTE	0.01	11	11012	9.0e-02	0
		0.10	10	10011	9.0e-02	0
		10.00	9	9010	9.0e-02	0
`nsol`			151	152416	9.0e-02	0

Figure 2.5: Plot of typical computed solution next to the exact solution for $\mu = 10^{-4}$. This is an enlarged view of the solution between $[0, 0.01]$ taken from a run with 1000 points.

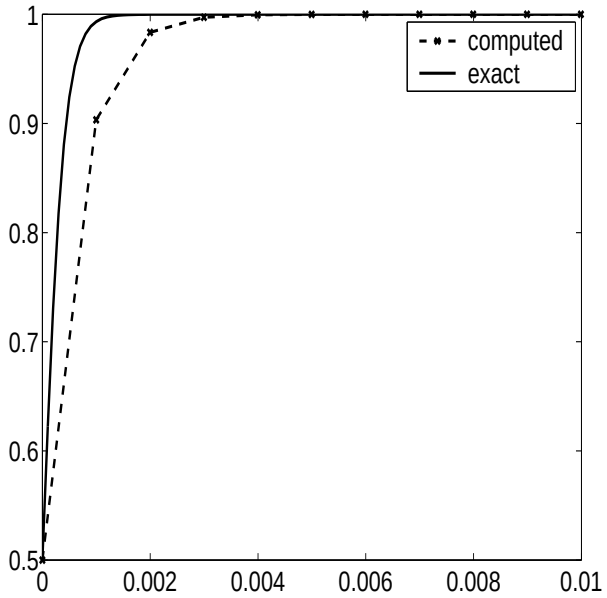


Figure 2.6: Computational results for Burgers' equation with $\mu = 0$, and direct linear solvers.

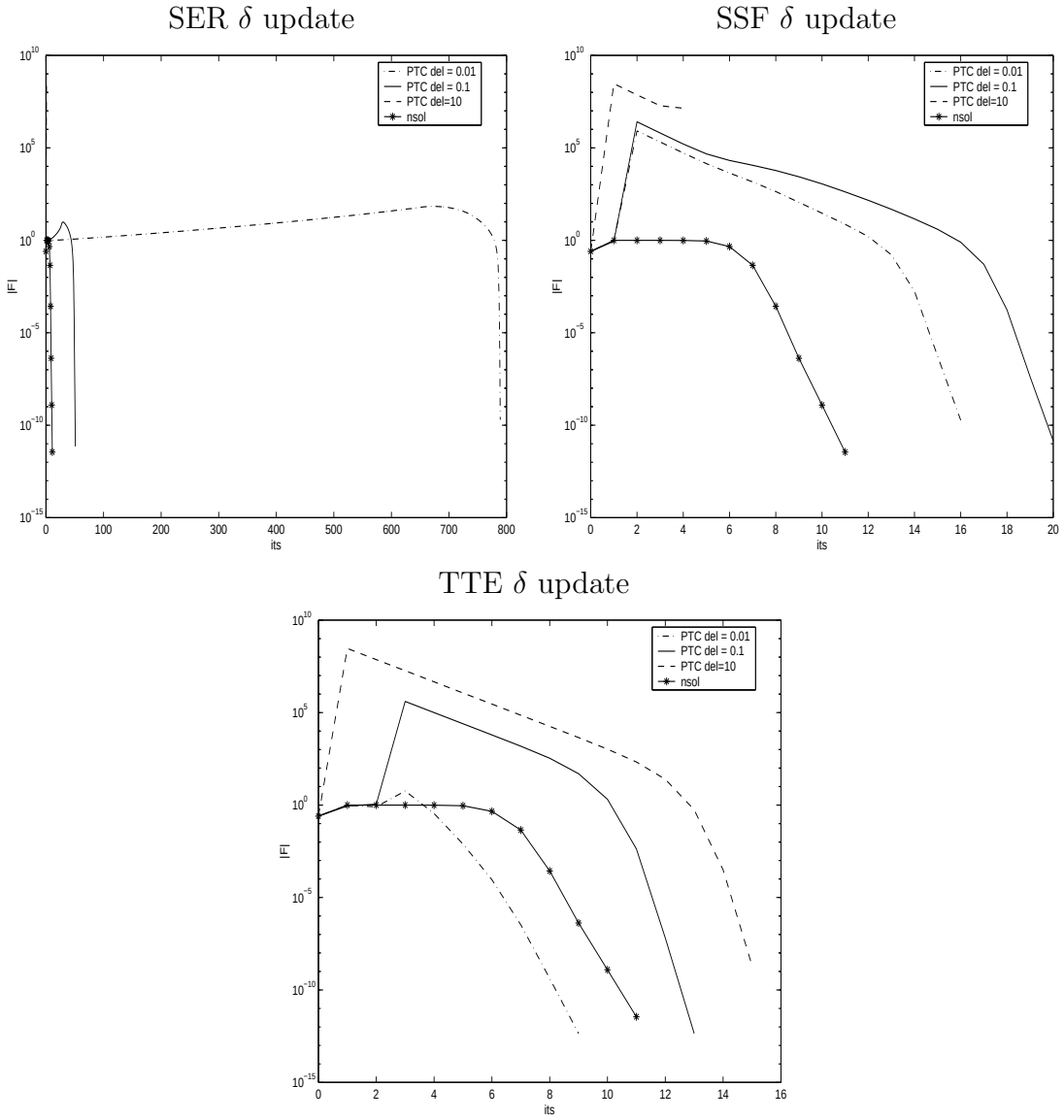


Table 2.12: Burgers' Equation: Direct linear solvers with $\mu = 0$.

code	δ update	δ_0	# its	Fevals	rel error	fail
Ψtc	SER	0.01	789	789790	5.0e-01	shift(1)
		0.10	51	51052	5.0e-01	shift(1)
		10.00	1	1002	3.0e+02	diverge
Ψtc	SSF	0.01	16	16017	5.0e-01	shift(1)
		0.10	20	20021	5.0e-01	shift(1)
		10.00	4	4005	6.9e+01	diverge
Ψtc	TTE	0.01	9	9010	5.0e-01	shift(1)
		0.10	13	13014	5.0e-01	shift(1)
		10.00	16	16017	5.0e-01	shift(1)
`nsol`			11	11068	5.0e-01	shift(1)

eventually. With centered differencing on the first derivative, it is impossible to get any of the codes to converge for $\mu = 0$. The forward differencing here really pays off. The `nsol` code really beats Ψtc here, see Figure 2.6.

2.4 Conclusions

In this chapter we have discussed the use of Pseudo-transient continuation on a variety of fairly simple problems. It would be convenient to replace Newton's method with Ψtc for use in all our problems. Unfortunately, as the examples from section 2.1 showed, it is easy to find problems which cause Ψtc a great deal of trouble. One could say that these problems don't satisfy the assumptions for Ψtc convergence, but then one often applies numerical methods outside where they're proven to converge. In these cases, Newton's method did a much better job of handling these problems than Ψtc. On the

other hand, we have problems like Burgers' equation in section 2.3, where Ψtc performed much better than Newton's method when the viscosity was small. For the two-point boundary value problem in section 2.2.1 and the convection-diffusion equation in section 2.2.2, both Newton and Ψtc performed well, though tuning Ψtc was very important. Newton's method, as formulated here, has no parameters that need adjusting, on the other hand, Ψtc has an initial step-size that must be chosen carefully. In later chapters, a few problem-specific heuristics will be brought up for choosing the initial step-size, but no conclusions have been forthcoming for a general heuristic which would apply at an algorithmic level rather than at a problem specific level. The examples in this chapter hopefully form a basis of intuition about Ψtc and its behavior on simple problems. In the next two chapters, we explore how Ψtc behaves on harder problems.

Chapter 3

1-D Euler Equations

3.1 Introduction

In this chapter Ψtc, as method for solving steady-state compressible fluid flow problems [36, 39, 49] is developed in the context of a global splitting method for non-smooth nonlinear equations [14, 31]. We make a conjecture about a convergence result and compare this conjecture with a numerical experiment. The theory for Ψtc [36] requires smooth nonlinear functions, but problems with non-smooth nonlinear functions have been solved, often by using a Jacobian for a related smooth problem [36, 23, 21, 22]. Also, in many cases derivative data for a related smooth problem are used when the original nonlinear functions are smooth but difficult to differentiate [44]. Both of these cases fall under the context of splitting methods.

In the next section, 3.2, we discuss the general structure of the problem and give the background for splitting methods and Ψtc along with known theoretical results. This

will clarify the gaps between the current theory and the problems presented here. In section 3.3 the hybrid method which will be used in this chapter is described. We make a conjecture about the form of the non-smoothness in our problem and prove a local convergence result using that conjecture. In section 3.4 we describe our problem from compressible fluid flow, supersonic isentropic flow through a nozzle in one dimension. A second order discretization yields a non-smooth nonlinear equation. We find our hybrid method by using a smooth first order discretization. In section 3.5 we substantiate the theoretical claims with a computational study.

3.2 Background

We are solving a nonlinear equation of the form

$$R(U) = 0 \tag{3.1}$$

where R is the spatial discretization of a boundary value problem for a partial differential equation. In the case described here, the one-dimensional steady-state isentropic Euler equations are discretized. Higher order discretizations require modifications to suppress spurious oscillations [32]. In this work, we used slope-limiters. The incorporation of slope limiters makes the higher order discretizations non-smooth.

These non-smooth nonlinear equations cannot be solved by a Newton method that requires differentiating these non-smooth functions. This is why we use related smooth

problems for the Jacobian data. Also, finding good initial data is problematic when complicated shock structures exist in the steady-state solution. So we must also modify the splitting methods described in this chapter so they can handle initial data that are far from the solution. Even for smooth problems, conventional root-finding methods, like line-search methods [35, 20, 50], can stagnate when the Jacobian is singular, or converge to non-physical solutions. Ψtc addresses these issues by accurately integrating the time-dependent equation early in the iteration and increasing the time-step later in the iteration to regain a rapid convergence rate. Complete temporal accuracy is not achieved with this approach, but the quality of the steady-state solution is not affected.

Splitting methods [14, 31] decompose R into smooth and non-smooth parts:

$$R(U) = R_S(U) + R_R(U), \tag{3.2}$$

where R_S is Lipschitz continuously differentiable (the "smooth" part) and R_R Lipschitz continuous (the "rough" part). This decomposition is used in the splitting iteration

$$U_+ = U_c - R'_S(U_c)^{-1} R(U_c), \tag{3.3}$$

where U_c is the current iterate and U_+ is the next iterate. In [14] the assumptions are:

Assumption 3.1. *Let the following hold,*

1. *There is a solution U^*,*

2. *$R'_S(U^*)$ is nonsingular,*

3. R_S is Lipschitz continuously differentiable, and

4. R_R is Lipschitz continuous.

We use Assumption 3.1 to write the splitting iteration (3.3) as a Newton iteration for the function $F(U) = R_S(U) - R_S(U^*)$, which satisfies the local Newton convergence assumptions.

$$U_+ = U_c - F'(U_c)^{-1}F(U_c) + R'_S(U_c)^{-1}\left(R_R(U^*) - R_R(U_c)\right). \tag{3.4}$$

Apply Newton convergence theory [35] to this iteration to get:

$$\|U_+ - U^*\| = O\left(\|U_c - U^*\|^2 + \gamma_R\|U_c - U^*\|\right), \tag{3.5}$$

where γ_R is the Lipschitz constant for $R_R(U)$. If γ_R is small, then the iteration will converge q-linearly.

We rarely compute the Jacobian data exactly, so inexact Newton methods [19] are used instead. These methods accept a Newton step satisfying the inexact Newton condition:

$$\|R'_S(U_c)s + R(U_c)\| \leq \eta_c\|R(U_c)\|. \tag{3.6}$$

If we use an iterative method for the linear systems, then η_c in (3.6) is the tolerance on the relative linear residual. If $U_+ = U_c + s$ with s satisfying the inexact Newton condition

(3.6), then (3.5) becomes

$$\|U_+ - U^*\| = O(\|U_c - U^*\|^2 + (\eta_c + \gamma_R)\|U_c - U^*\|). \tag{3.7}$$

Let $R(U) = 0$ be the steady state equation for the time-dependent initial value problem,

$$\frac{dU}{dt} = -R(U), \quad U(0) = U_0. \tag{3.8}$$

Ψtc starts out accurately integrating the transient behavior of the problem, then once an approximate steady-state is reached, it increases the time-step to achieve a rapid convergence to steady-state.

Recall the iteration for Ψtc from section 1.2 and note that the Ψtc step is:

$$(\delta_c^{-1} I + J_c)s = -R(U_c), \quad \text{where } J_c \approx R'(U_c), \tag{3.9}$$

where δ_c is the current "time step." If δ is fixed, we can view (3.9) as one step of a Rosenbrock method [27]. Note that Ψtc does not vary the step to achieve temporal accuracy, but rather to converge to steady-state as quickly as possible.

Recall the formula for the SER δ update (1.2) from section 1.2. This formula can be rewritten:

$$\delta_n = \min(\delta_0 \|R(U_0)\| / \|R(U_n)\|, \delta_{\max}). \tag{3.10}$$

SER is a common approach for fluid flow problems [36, 49, 38, 60]. We assume Assump-

tions 1.1 and 1.2 hold for Ψtc applied to the problem in equation (3.8). The convergence theory from [36] then tells us that the iteration will converge to the desired steady-state solution, $U^* = \lim_{t \longrightarrow \infty} U(t)$, where U is the solution of (3.8).

The following theorem from [16] summarizes the convergence results in [36] where we use a step s_n satisfying the inexact Newton condition:

$$\|(\delta_n^{-1} I + R'(U_n))s_n + R(U_n)\| \leq \eta_n \|R(U_n)\|. \tag{3.11}$$

Theorem 3.1. *Let the standard Ψtc assumptions 1.2 hold. Then if δ_0 is sufficiently small, s_n satisfies (3.11), and if $\eta_n \leq \overline{\eta} < 1$ for some $\overline{\eta}$ sufficiently small, then $\delta_n \longrightarrow \infty$, $U_n \longrightarrow U^*$, and the convergence in the terminal phase of the iteration is described by*

$$\|U_{n+1} - U^*\| \leq O((\eta_n + \delta_n^{-1})\|U_n - U^*\| + \|U_n - U^*\|^2)$$

for n sufficiently large. In particular, if $\delta_{max} = \infty$ and $\eta_n = 0$, the convergence is q-quadratic.

3.3 Hybrid Algorithm

The hybrid algorithm combines the splitting method with Ψtc by using $J_c = R_S(U_c)$ in both the equation for the step,

$$(\delta_c^{-1} I + R'_S(U_c))s = -R(U_c), \tag{3.12}$$

and in the inexact Newton condition,

$$\|(\delta_c^{-1} I + R'_S(U_c))s + R(U_c)\| \leq \eta_c \|R(U_c)\|. \tag{3.13}$$

This method has been used for some time in the computational fluid dynamics community. Actually, this is the only reasonable formulation if higher order discretizations are used. A sophisticated form of this hybrid was used to generate the numerical results in [36].

Although this method is simple to formulate, no formal analysis has been made in support of the method. The work of this chapter is a step in that direction. We ran a computational study of a simple problem with the aim of verifying a convergence result. We propose a conjecture about the size of the non-smooth component, and using that conjecture, prove a local convergence result. In our Euler equation example, R is the spatial discretization of a partial differential operator. The non-smoothness in this particular operator comes from portions of the method which are required to correctly resolve changes in mathematical character (i.e. changes between hyperbolic, parabolic, and elliptic behavior). R_S is a first order spatial discretization of the same operator which results in a smooth function.

We now make the following conjecture. Let Δx be the spatial mesh width and assume there is a $C > 0$ such that

$$\|R'_S(U_c)^{-1}(R_R(U^*) - R_R(U_c))\| \leq C(\Delta x \|U_c - U^*\|) \tag{3.14}$$

for all U_c sufficiently near U^*, where $\|\cdot\|$ denotes the l^1 norm. This conjecture will hold if the non-smoothness is localized to a small part of the mesh that becomes smaller, and goes away, as the mesh is refined. If we replace the standard assumptions on Ψtc, Assumption 1.2, with the following stronger assumptions we can prove a local convergence result for the hybrid method.

Assumption 3.2. *Let the following hold:*

1. *There is a solution U^*.*
2. *$R'_S(U^*)$ is nonsingular.*
3. *R_S is Lipschitz continuously differentiable.*
4. *Equation (3.14) holds.*

We now state and prove a theorem from [16] that shows $U_n \longrightarrow U^*$ and $\delta_n \longrightarrow \infty$, which implies the convergence rate is the same as for the inexact Newton implementation (3.6). This result corresponds to the theorem about the terminal phase of Ψtc described in [36].

Theorem 3.2. *Let Assumption 3.2 holds. Let $0 \leq \eta_c \leq \overline{\eta} < 1$. Then if U_c is sufficiently near U^*, $\overline{\eta}$ and Δx are sufficiently small, and δ_c, δ_{max} are sufficiently large, then the iteration defined by (3.12), (3.13), and (3.10) satisfies*

$$\|U_+ - U^*\| \leq C_1(\|U_c - U^*\|^2 + (\eta_c + \Delta x + \delta_c^{-1})\|U_c - U^*\|) \tag{3.15}$$

for some $C_1 > 0$. *Moreover, either*

$$\delta_+ = \delta_{max} \quad or \quad \delta_+ > \delta_c. \tag{3.16}$$

Proof. Let U_c be near enough to U^* and let $\overline{\eta}$ and Δx be small enough so that (3.6), (3.7), and (3.14) imply that

- (3.15) holds for all $0 \leq \eta_c \leq \overline{\eta}$,
- $\|U_+ - U^*\| \leq \|U_c - U^*\|/2$, and
- $\|R(U_+)\| < \|R(U_c)\|$.

in that case, (3.10) implies (3.16). □

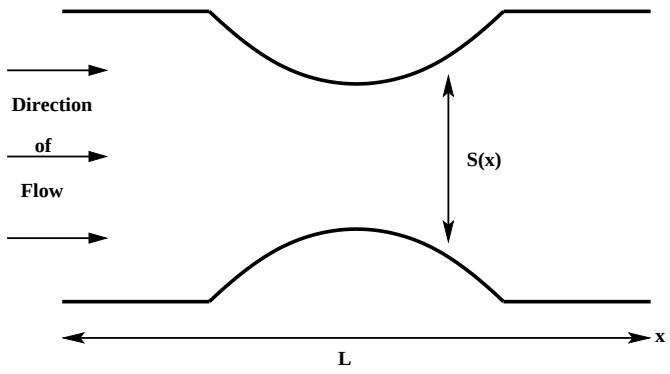
If we assume that $\delta_{\max} > 1/\Delta x$, which is the case in many realistic applications, then Theorem 3.2 implies that the limiting convergence rate will be q-linear with a q-factor proportional to $\Delta x + \overline{\eta}$. In our application in section 3.5, $\overline{\eta} = O(\Delta x)$ is approximately the error in an approximate Jacobian. The following corollary summarizes this:

Corollary 3.1. *Let assumption 3.2 hold. Let* $0 \leq \eta_n \leq \overline{\eta} = O(\Delta x) < 1$. *Then if* U_0 *is sufficiently near* U^*, $\overline{\eta}$ *and* Δx *are sufficiently small,* $\delta_c > 1/\Delta x$ *and* δ_{max} *are sufficiently large, then the iteration given by (3.13) and (3.10) converges q-linearly with q-factor proportional to* Δx.

3.4 Physical Problem and Finite Volume Formulation

In this section we solve the one-dimensional Euler equations in a nozzle at supersonic speeds and investigate the difficulty with which steady state is achieved. Recall section 1.3.1 where the one dimensional Euler equations were introduced. The physical domain of interest for our problem is shown in Figure 3.1. Since $S(x)$ is the cross-sectional area, we can imagine a tube in 3-D to get the radius $Rad(x) = \sqrt{\frac{S(x)}{\pi}}$.

Figure 3.1: Physical Problem



Recall,

$$F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ (\rho E + p)u \end{pmatrix} \quad \text{and} \quad W = \begin{pmatrix} 0 \\ p\frac{dS}{dx} \\ 0 \end{pmatrix}.$$

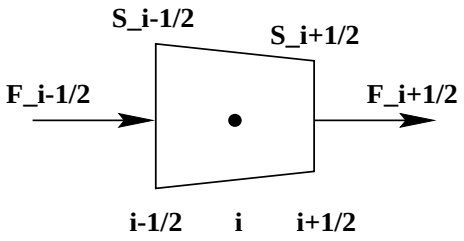
with the system,

$$SU_t + (SF)_x = W.$$

We consider two sets of boundary conditions for this problem. The first is supersonic inlet and supersonic outlet. The second is supersonic inlet and subsonic outlet. For the supersonic inlet condition, we specify all three variables at the inlet. For the supersonic outlet condition, we extrapolate all three variables to the outlet. And when we look at the subsonic outlet condition, we will specify pressure and use the continuity relations to determine the other two variables at the outlet. For initial conditions, we use constant flow defined by the variables at the inlet.

Since we are looking for a steady state solution we will set $U_t = 0$ and write our system as the equation $R(U) = 0$. The system is discretized by use of its finite volume formulation, as in Figure 3.2, with the resulting function:

Figure 3.2: Finite volume formulation for the residual



$$R(U) = \frac{S_{i+\frac{1}{2}}F_{i+\frac{1}{2}} - S_{i-\frac{1}{2}}F_{i-\frac{1}{2}}}{\Delta x} - \begin{bmatrix} 0 \\ p_i \frac{S_{i+\frac{1}{2}} - S_{i-\frac{1}{2}}}{\Delta x} \\ 0 \end{bmatrix}. \tag{3.17}$$

We define the fluxes ($F_{i\pm\frac{1}{2}}$) using first order Roe flux differencing [32, 54, 55, 56]:

$$F_{i+\frac{1}{2}} = \frac{1}{2}\left(F_{i+1} + F_i - |\overline{\overline{A}}(U_i, U_{i+1})|(U_{i+1} - U_i)\right).$$

Where $\overline{\overline{A}}$ is the Roe matrix and will be described in the next section. In this equation, the expression $|\overline{\overline{A}}|$ is defined by:

$$\text{if} \quad \overline{\overline{A}} = T^{-1}\Lambda T \quad \text{then} \quad |\overline{\overline{A}}| = T^{-1}|\Lambda|T,$$

where Λ is the diagonal matrix of the eigenvalues of $\overline{\overline{A}}$.

3.4.1 Roe Flux Differencing

As described in [32], the matrix $\overline{\overline{A}}(U_i, U_{i+1})$ must have the following properties:

1. For any pair U_i, U_{i+1} one should have *exactly*

$$F_{i+1} - F_i = \overline{\overline{A}}(U_i, U_{i+1})(U_{i+1} - U_i).$$

2. For $U_i = U_{i+1} = U$ the matrix $\overline{\overline{A}}(U, U) = A(U) \equiv \partial F/\partial U$.

3. $\overline{\overline{A}}$ has real eigenvalues with linearly independent eigenvectors.

Roe discovered that if $\partial F/\partial U$ is written in terms of ρ, u, H and these variables are weighted by the densities in a particular way (3.19), then this matrix satisfies the conditions above on $\overline{\overline{A}}$. The Jacobian matrix $\partial F/\partial U$ is a restricted version of the 3-d Jacobian (1.6):

$$\frac{\partial F}{\partial U} = \begin{bmatrix} 0 & 1 & 0 \\ -(3-\gamma)\frac{u^2}{2} & (3-\gamma)u & \gamma-1 \\ (\gamma-1)u^3 - \gamma u E & \gamma E - \frac{3}{2}(\gamma-1)u^2 & \gamma u \end{bmatrix}. \tag{3.18}$$

When this Jacobian is written in terms of ρ, u, H, it looks like:

$$\begin{bmatrix} 0 & 1 & 0 \\ -(3-\gamma)\frac{u^2}{2} & (3-\gamma)u & \gamma-1 \\ (\gamma-1)u^3 - u(H+(\gamma-1)\frac{u^2}{2\rho^2}) & H+(\gamma-1)\frac{u^2}{2\rho^2} - \frac{3}{2}(\gamma-1)u^2 & \gamma u \end{bmatrix}.$$

The weighted variables are:

$$\begin{aligned} R_{i+\frac{1}{2}} &= \sqrt{\tfrac{\rho_{i+1}}{\rho_i}}, \\ \overline{\overline{\rho}}_{i+\frac{1}{2}} &= R_{i+\frac{1}{2}}\rho_i, \\ \overline{\overline{u}}_{i+\frac{1}{2}} &= \tfrac{R_{i+\frac{1}{2}}u_{i+1}+u_i}{R_{i+\frac{1}{2}}+1}, \text{ and} \\ \overline{\overline{H}}_{i+\frac{1}{2}} &= \tfrac{R_{i+\frac{1}{2}}H_{i+1}+H_i}{R_{i+\frac{1}{2}}+1}. \end{aligned} \tag{3.19}$$

This gives us the following Roe matrix $\overline{\overline{A}}$ where each variable is evaluated at $i+\frac{1}{2}$:

$$\overline{\overline{A}}(U_i, U_{i+1}) = \begin{bmatrix} 0 & 1 & 0 \\ -(3-\gamma)\frac{\overline{\overline{u}}^2}{2} & (3-\gamma)\overline{\overline{u}} & \gamma - 1 \\ (\gamma-1)\overline{\overline{u}}^3 - \overline{\overline{u}}(\overline{\overline{H}} + (\gamma-1)\frac{\overline{\overline{u}}^2}{2\overline{\overline{\rho}}^2}) & \overline{\overline{H}} + (\gamma-1)\frac{\overline{\overline{u}}^2}{2\overline{\overline{\rho}}^2} - \frac{3}{2}(\gamma-1)\overline{\overline{u}}^2 & \gamma\overline{\overline{u}} \end{bmatrix}.$$

Roe's flux differencing evaluates $|\overline{\overline{A}}(U_i, U_{i+1})|(U_{i+1} - U_i)$ by the following steps (all variables are evaluated at $i + \frac{1}{2}$):

1. For each cell $(i, i+1)$ calculate the averages above, $\overline{\overline{\rho}}, \overline{\overline{u}}, \overline{\overline{H}}$, and the averaged speed of sound at $i + \frac{1}{2}$:

$$\overline{\overline{c}} = (\gamma - 1)\left(\overline{\overline{H}} - \frac{\overline{\overline{u}}^2}{2}\right).$$

2. Calculate the eigenvalues and eigenvectors of the Roe matrix $\overline{\overline{A}}(U_i, U_{i+1})$:

$$\overline{\overline{\lambda}}_{(1)} = \overline{\overline{u}}, \quad \overline{\overline{\lambda}}_{(2)} = \overline{\overline{u}} + \overline{\overline{c}}, \text{ and } \quad \overline{\overline{\lambda}}_{(3)} = \overline{\overline{u}} - \overline{\overline{c}},$$

with the eigenvectors,

$$\overline{\overline{r}}^{(1)} = \begin{vmatrix} 1 \\ \overline{\overline{u}} \\ \frac{\overline{\overline{u}}^2}{2} \end{vmatrix}, \quad \overline{\overline{r}}^{(2)} = \begin{vmatrix} 1 \\ \overline{\overline{u}} + \overline{\overline{c}} \\ \overline{\overline{H}} + \overline{\overline{uc}} \end{vmatrix} \frac{\overline{\overline{\rho}}}{2\overline{\overline{c}}}, \text{ and } \quad \overline{\overline{r}}^{(3)} = \begin{vmatrix} 1 \\ \overline{\overline{u}} - \overline{\overline{c}} \\ \overline{\overline{H}} - \overline{\overline{uc}} \end{vmatrix} \frac{\overline{\overline{\rho}}}{2\overline{\overline{c}}}.$$

3. Calculate the wave amplitudes [32],

$$\begin{aligned}
\partial w_1 &= \delta p - \frac{\delta p}{\overline{\overline{c}}^2}, \\
\partial w_2 &= \delta u + \frac{\delta p}{\overline{\overline{\rho c}}}, \\
\partial w_3 &= \delta u - \frac{\delta p}{\overline{\overline{\rho c}}}, \\
\delta u_{i+\frac{1}{2}} &= u_{i+1} - u_i, \\
\delta p_{i+\frac{1}{2}} &= p_{i+1} - p_i, \text{ and} \\
\delta \rho_{i+\frac{1}{2}} &= \rho_{i+1} - \rho_i.
\end{aligned}$$

4. Evaluate the numerical flux [32],

$$F_{i+\frac{1}{2}} = \frac{1}{2}\left(F_{i+1} + F_i - \sum_{j=1}^{3} |\overline{\overline{\lambda}}_{(j)}| \partial w_j \overline{\overline{r}}^{(j)}\right).$$

We will henceforth write $\overline{\overline{A}}_{i+\frac{1}{2}}$ for $\overline{\overline{A}}(U_i, U_{i+1})$. This fully defines Roe's Flux differencing as a first order upwind method.

3.4.2 MUSCL

To make the method more accurate we will use quadratic approximations to the cell centered data. The following scheme is called MUSCL [43, 2, 32] (Monotone Upstream-

centered Schemes for Conservation Laws).

We define $U^L_{i+\frac{1}{2}}$ and $U^R_{i+\frac{1}{2}}$ according to [32]:

$$U^L_{i+\frac{1}{2}} = U_i + \frac{1}{4}\left[(1-\kappa)\Phi(r^+_{i-\frac{1}{2}})(U_i - U_{i-1}) + (1+\kappa)\Phi(r^-_{i+\frac{1}{2}})(U_{i+1} - U_i)\right], \qquad (3.20)$$

where r is defined by the ratio of differences of U:

$$r^+_{i-\frac{1}{2}} = \frac{U_{i+2} - U_{i+1}}{U_{i+1} - U_i},$$

$$r^-_{i+\frac{1}{2}} = \frac{U_i - U_{i-1}}{U_{i+1} - U_i},$$

and Φ is a flux limiter. A flux limiter will scale the slopes to avoid introducing new local extrema and to reduce the method to first order near shocks. The flux limiters we used are described in detail in the next section. Similarly for $U^R_{i+\frac{1}{2}}$ we have:

$$U^R_{i+\frac{1}{2}} = U_{i+1} - \frac{1}{4}\left[(1-\kappa)\Phi(r^-_{i+\frac{3}{2}})(U_{i+2} - U_{i+1}) + (1+\kappa)\Phi(r^+_{i+\frac{1}{2}})(U_{i+1} - U_i)\right], \qquad (3.21)$$

$$r^-_{i+\frac{3}{2}} = \frac{U_{i+1} - U_i}{U_{i+2} - U_{i+1}}, \text{ and}$$

$$r^+_{i+\frac{1}{2}} = \frac{U_{i+2} - U_{i+1}}{U_{i+1} - U_i}.$$

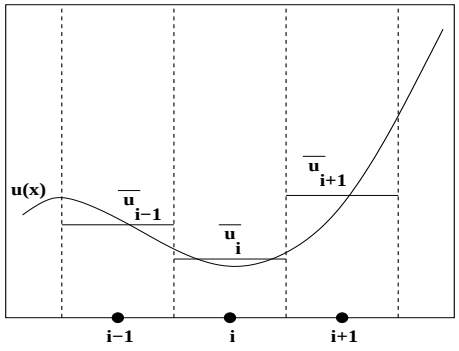
Explanation of MUSCL formulation:

Let $\overline{u}_i$ be the average value of $u(x)$ over the cell i defined by

$$\overline{u}_i = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} u(x)dx$$

and shown visually in Figure 3.3. When we use a first order method like Roe's scheme

Figure 3.3: MUSCL cell average notation



above, we approximate $u(x)$ with $\overline{u}_i$ on cell i which represents a piecewise constant approximation to u. With MUSCL extrapolation we instead consider piecewise linear or quadratic approximations on each cell. But we demand that the average value over the cell remains the same. We need this condition to maintain conservation of mass. If we allow the average value to change after replacing the constant approximation with a linear or quadratic approximation on the cell, then we have changed the mass in that

cell. This condition is satisfied easily for the linear approximation

$$u(x) = b(x - x_i) + \overline{u}_i$$

for any b, since the area under $u(x)$ remains $\overline{u}_i$. We select b by fitting our linear approximation to the data we have. Since we only store cell centered cell averaged quantities $\overline{u}_i$, we can approximate the slope in cell i by $\left(\overline{u}_{i+1} - \overline{u}_{i-1}\right)/(2\Delta x)$. Hence,

$$b = \frac{\overline{u}_{i+1} - \overline{u}_{i-1}}{2\Delta x} \approx u'(x_i).$$

This gives us the following linear approximation for $u(x)$:

$$u(x) = \frac{\overline{u}_{i+1} - \overline{u}_{i-1}}{2\Delta x}(x - x_i) + \overline{u}_i.$$

We are really interested in evaluating this approximation at the cell edges. For example, $u_{i+\frac{1}{2}}$ can be linearly approximated by:

$$u(x_{i+\frac{1}{2}}) = \overline{u}_i + \frac{1}{4}\left[(\overline{u}_i - \overline{u}_{i-1}) + (\overline{u}_{i+1} - \overline{u}_i)\right].$$

If we add in the flux limiter as above, then we get a piecewise constant approximation near the shocks and a linear approximation away from the shocks.

A similar formula can be constructed for quadratic approximations to $u(x)$. Consider

the general quadratic centered about x_i:

$$u(x) = a(x - x_i)^2 + b(x - x_i) + c.$$

When we find what values for a, b, and c will give an average value of $\overline{u}_i$, we find that a and b are independent, but

$$c = -\frac{a\Delta x^2}{12} + \overline{u}_i.$$

Thus, for any values of a and b, the quadratic approximation,

$$u(x) = a(x - x_i)^2 + b(x - x_i) + \overline{u}_i - \frac{a\Delta x^2}{12}$$

has the average value $\overline{u}_i$ over cell i. To specify values for a and b consider the first and second derivative of this quadratic approximation at x_i:

$$u'(x_i) = 2a(x_i - x_i) + b = b$$

and

$$u''(x_i) = 2a.$$

Therefore, let $b \approx u'(x_i)$ and $a \approx u''(x_i)/2$. We use the same approximation for $u'(x_i)$ as above and

$$u''(x_i) \approx \frac{\overline{u}_{i+1} - 2\overline{u}_i + \overline{u}_{i-1}}{\Delta x^2}.$$

As before, we can now approximate $U_{i+\frac{1}{2}}$ by:

$$u(x_{i+\frac{1}{2}}) = \overline{u}_i + \frac{1}{4}\left[\frac{2}{3}(\overline{u}_i - \overline{u}_{i-1}) + \frac{4}{3}(\overline{u}_{i+1} - \overline{u}_i)\right].$$

Note that when $\kappa = \frac{1}{3}$, this formulation is identical to the one presented above for $U_{i+\frac{1}{2}}$ (3.20) with the exception of the flux limiters. And as in the linear case, the flux limiters simply reduce the accuracy to first order when near shocks.

It is now clear that choosing $\kappa = \frac{1}{3}$ will result in piecewise quadratic approximations on the cells and hence local third order spatial accuracy [32]. To implement these approximations, for the $i + \frac{1}{2}$ interface, U_i in (3.19) is replaced by $U^L_{i+\frac{1}{2}}$ and U_{i+1} in (3.19) is replaced by $U^R_{i+\frac{1}{2}}$. And when ρ and p are used, they are also derived from this MUSCL approximation. A similar substitution holds for $i - \frac{1}{2}$. Our scheme is then:

$$F_{i\pm\frac{1}{2}} = \frac{1}{2}\left(F^L_{i\pm\frac{1}{2}} + F^R_{i\pm\frac{1}{2}} - |\overline{\overline{A}}|_{i\pm\frac{1}{2}}(U^R_{i\pm\frac{1}{2}} - U^L_{i\pm\frac{1}{2}})\right) \tag{3.22}$$

Where,

$$F^L_{i\pm\frac{1}{2}} = F(U^L_{i\pm\frac{1}{2}}) \quad \text{and} \quad F^R_{i\pm\frac{1}{2}} = F(U^R_{i\pm\frac{1}{2}}).$$

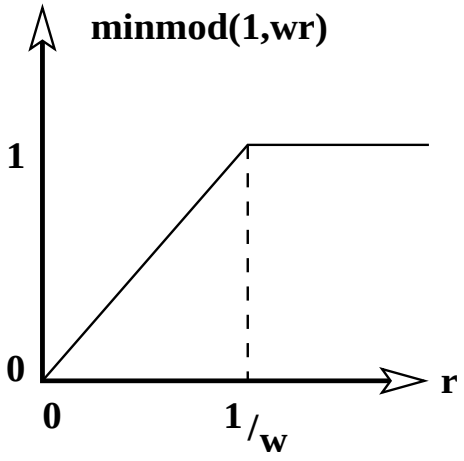
3.4.3 Flux Limiters

We define several different flux limiters here. First consider the minmod limiter:

$$\Phi^M(r) = \left\{ \begin{array}{cc} min(1, r) & \text{if } r > 0 \\ \\ 0 & \text{if } r \leq 0 \end{array} \right\}.$$

Figure 3.4 shows the shape of the minmod limiter as a function of r. This function can also be described in terms of two parameters. In this two parameter formulation, it is given slopes on either side of an interface, and the minmod limiter returns the smallest slope if they are the same sign, and zero otherwise [32, 59]. This two parameter formulation makes it easier to understand what the limiter does. When the slopes differ in sign, that indicates a local max or min and therefore zero is the appropriate slope. Using the smallest of the slopes guarantees that no new local max or min will be introduced through the use of higher order approximations for the solution on the interior of the cells.

Figure 3.4: Minmod flux limiter



$$
\begin{aligned}
f_{\text{minmod}}(a,b) &= \begin{cases} a & \text{if} \quad |a| < |b| \quad \text{and} \quad ab > 0 \\ b & \text{if} \quad |a| > |b| \quad \text{and} \quad ab > 0 \\ 0 & \text{if} \quad\quad\quad\quad\quad\quad\quad ab < 0 \end{cases} \\
&= \tfrac{1}{2}\left[sign(a) + sign(b)\right] \min\left(|a|, |b|\right).
\end{aligned}
$$

When written this way, Φ^M above is rewritten:

$$\Phi^M(r) = \text{minmod}(1, r).$$

The minmod function is very diffusive so the parameter ω is added to its argument to get:

$$\Phi^M(r) = \text{minmod}(1, \omega r).$$

Notice that for any $r > 0$,

$$\lim_{\omega \to 0} \text{minmod}(1, \omega r) = 0 \quad \text{and} \quad \lim_{\omega \to \infty} \text{minmod}(1, \omega r) = 1.$$

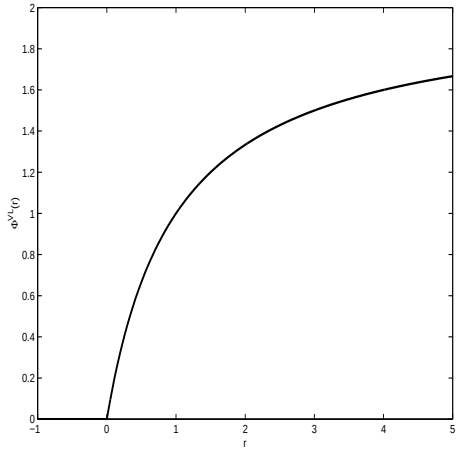
When we say diffusive above, we mean that $\text{minmod}(1, r) < 1$ for a large range of r. By placing $\omega \geq 1$ into the minmod function we can reduce this set and therefore cause the limiter to be one more often. On the other hand if ω is too large, then this function will not act as a flux limiter because it will not limit enough values. Therefore, ω must satisfy, $1 \leq \omega \leq 2.5$ for the limiter to exclude the introduction of new local extrema. The smaller ω is, the more diffusive this limiter will be.

We also use the VanLeer limiter:

$$\Phi^{VL}(r) = \frac{r + |r|}{1 + r},$$

whose graph is in Figure 3.5. This limiter only has one point where it is non-differentiable, whereas the minmod limiter has two points. This extra smoothness can make analyzing the method easier. This completes the discretization and definition of the residual. Since

Figure 3.5: VanLeer flux limiter



this discretization results in a non-differentiable function due to the flux-limiter, we will use a different discretization of F for the Jacobian [23, 21, 22, 44]. Note that this non-smooth nonlinear function is Lipschitz continuous and therefore satisfies the requirements for the hybrid splitting method results to apply.

3.4.4 Lax-Friedrich's Discretization

In order to find a related differentiable problem for our Jacobian data in the hybrid splitting method, we consider the first order Lax-Friedrich's discretization. This discretization of F gives the following fluxes [32, 42]:

$$\begin{aligned}\widehat{F}_{i+\frac{1}{2}} &= \tfrac{1}{2}(F_{i+1}+F_i) - \tfrac{1}{2}|\lambda|_{i+\frac{1}{2}}(U_{i+1}-U_i)\\ \widehat{F}_{i-\frac{1}{2}} &= \tfrac{1}{2}(F_i+F_{i-1}) - \tfrac{1}{2}|\lambda|_{i-\frac{1}{2}}(U_i-U_{i-1}).\end{aligned}$$

Note, in these fluxes $|\lambda|$ is evaluated at cell edges. In our computation of the Jacobian we will approximate $|\lambda_{i\pm\frac{1}{2}}|$ with cell centered data, $|\lambda_{i-1}|$,$|\lambda_i|$, and $|\lambda_{i+1}|$. In this case,

$$|\lambda|_i = |u|_i + c_i.$$

We now write the residual equation $\widehat{R}$ in groups of three equations stacked into a $3N\times 1$ vector. Each group of three equations is denoted:

$$\begin{aligned}\widehat{R}(U)_i &= \frac{S_{i+\frac{1}{2}}\widehat{F}_{i+\frac{1}{2}} - S_{i-\frac{1}{2}}\widehat{F}_{i-\frac{1}{2}}}{\Delta x} - \begin{bmatrix} 0 \\ p_i\frac{S_{i+\frac{1}{2}}-S_{i-\frac{1}{2}}}{\Delta x} \\ 0\end{bmatrix},\\ &= \frac{S_{i+\frac{1}{2}}}{2\Delta x}\left(F_{i+1}+F_i - |\lambda|_{i+\frac{1}{2}}(U_{i+1}-U_i)\right)\\ &\quad -\frac{S_{i-\frac{1}{2}}}{2\Delta x}\left(F_i+F_{i-1} - |\lambda|_{i-\frac{1}{2}}(U_i-U_{i-1})\right) - \begin{bmatrix} 0 \\ p_i\frac{S_{i+\frac{1}{2}}-S_{i-\frac{1}{2}}}{\Delta x} \\ 0\end{bmatrix},\end{aligned}$$

for i from 1 to N. This defines the smooth residual,

$$R_S(U) = \widehat{R}(U). \tag{3.23}$$

Note that $\widehat{R}(U)_i$ will only use data from U_{i-1}, U_i, and U_{i+1}, so the Jacobian will be block tridiagonal ($3N \times 3N$). We will make two assumptions in the computation of the Jacobian. The first change is already noted above, we will approximate the cell edge data in the residual with cell center data. This can be described by writing down three new residuals which are simply variations of the residual above with cell edge data moved to cell centers.

$$\begin{aligned}
\widehat{R}^{(1)}(U)_i &= \tfrac{S_{i+1}}{2\Delta x}\left(F_{i+1} + F_i - |\lambda|_{i+1}(U_{i+1} - U_i)\right) \\
&\quad -\tfrac{S_i}{2\Delta x}\left(F_i + F_{i-1} - |\lambda|_i(U_i - U_{i-1})\right) - \begin{bmatrix} 0 \\ p_i \frac{S_{i+1}-S_i}{\Delta x} \\ 0 \end{bmatrix}, \\
\widehat{R}^{(0)}(U)_i &= \tfrac{S_i}{2\Delta x}\left(F_{i+1} + F_i - |\lambda|_i(U_{i+1} - U_i)\right) \\
&\quad -\tfrac{S_i}{2\Delta x}\left(F_i + F_{i-1} - |\lambda|_i(U_i - U_{i-1})\right) - \begin{bmatrix} 0 \\ p_i \frac{S_i-S_i}{\Delta x} \\ 0 \end{bmatrix}, \text{ and} \\
\widehat{R}^{(-1)}(U)_i &= \tfrac{S_i}{2\Delta x}\left(F_{i+1} + F_i - |\lambda|_i(U_{i+1} - U_i)\right) \\
&\quad -\tfrac{S_{i-1}}{2\Delta x}\left(F_i + F_{i-1} - |\lambda|_{i-1}(U_i - U_{i-1})\right) - \begin{bmatrix} 0 \\ p_i \frac{S_i-S_{i-1}}{\Delta x} \\ 0 \end{bmatrix}.
\end{aligned}$$

Now consider the derivative of $\widehat{R}(U)_i$ with respect to U:

$$\frac{\partial \widehat{R}(U)_i}{\partial U} = \begin{bmatrix} \vdots \\ 0 \\ \frac{\partial \widehat{R}(U)_i}{\partial U_{i-1}} \\ \frac{\partial \widehat{R}(U)_i}{\partial U_i} \\ \frac{\partial \widehat{R}(U)_i}{\partial U_{i+1}} \\ 0 \\ \vdots \end{bmatrix} \approx \begin{bmatrix} \vdots \\ 0 \\ \frac{\partial \widehat{R}^{(-1)}(U)_i}{\partial U_{i-1}} \\ \frac{\partial \widehat{R}^{(0)}(U)_i}{\partial U_i} \\ \frac{\partial \widehat{R}^{(1)}(U)_i}{\partial U_{i+1}} \\ 0 \\ \vdots \end{bmatrix}.$$

Our second assumption is that λ is not a function of U for these derivatives. This reduces the complexity of the Jacobian. Given these assumptions we can write the individual

blocks of the Jacobian:

$$\begin{aligned}
\frac{\partial \widehat{R}^{(-1)}(U)_i}{\partial U_{i-1}} &= d_i^{(-1)} = -\frac{1}{2\Delta x} S_{i-1} \left(\frac{\partial F}{\partial U}\Big|_{i-1} + \lambda_{i-1} I \right), \\
\frac{\partial \widehat{R}^{(0)}(U)_i}{\partial U_i} &= d_i^{(0)} = \frac{1}{\Delta x} S_i \lambda_i I, \text{ and} \\
\frac{\partial \widehat{R}^{(1)}(U)_i}{\partial U_{i+1}} &= d_i^{(1)} = \frac{1}{2\Delta x} S_{i+1} \left(\frac{\partial F}{\partial U}\Big|_{i+1} - \lambda_{i+1} I \right),
\end{aligned}$$

where $\partial F/\partial U$ is the matrix given by (3.18). Now when we put $\lambda = |u| + c$ back, we get the derivatives,

$$\begin{aligned}
\frac{\partial \widehat{R}^{(-1)}(U)_i}{\partial U_{i-1}} &= d_i^{(-1)} = -\frac{1}{2\Delta x} S_{i-1} \left(\frac{\partial F}{\partial U}\Big|_{i-1} + (|u| + c)_{i-1} I \right), \\
\frac{\partial \widehat{R}^{(0)}(U)_i}{\partial U_i} &= d_i^{(0)} = \frac{1}{\Delta x} S_i (|u| + c)_i I, \text{ and} \\
\frac{\partial \widehat{R}^{(1)}(U)_i}{\partial U_{i+1}} &= d_i^{(1)} = \frac{1}{2\Delta x} S_{i+1} \left(\frac{\partial F}{\partial U}\Big|_{i+1} - (|u| + c)_{i+1} I \right).
\end{aligned}$$

In the above equations the d_i^j are 3×3 matrices with the following limits:

$$\begin{aligned}
d_i^{(-1)} &: \quad 2 \leq i \leq N, \\
d_i^{(0)} &: \quad 1 \leq i \leq N, \text{ and} \\
d_i^{(1)} &: \quad 1 \leq i \leq N - 1.
\end{aligned}$$

The resulting Jacobian $\partial\widehat{R}/\partial U$ can now be written concisely:

$$J(U) = \frac{\partial \widehat{R}}{\partial U} = \begin{bmatrix} d_1^{(0)} & d_1^{(1)} & & \\ d_2^{(-1)} & \ddots & \ddots & \\ & \ddots & \ddots & d_{N-1}^{(1)} \\ & & d_N^{(-1)} & d_N^{(0)} \end{bmatrix}_{[3N\times 3N]} . \tag{3.24}$$

This Jacobian is a simplification of the Jacobian for the smooth nonlinear Lax-Friedrich's discretization which is Lipschitz continuously differentiable as needed by the hybrid splitting method results.

3.4.5 Boundary and Initial Conditions

We consider two different sets of boundary conditions for this problem. The first is supersonic inlet and supersonic outlet. Since the eigenvalues for this problem are u, $u+c$ and $u-c$, all three eigenvalues are positive at both ends of the nozzle. This means we must prescribe three conditions at the inlet and may not provide any conditions on the outlet. The second set of boundary conditions are supersonic inlet and subsonic outlet. So we provide three conditions at the inlet and one condition at the outlet.

We will specify the inlet boundary conditions in terms of density (ρ), temperature (T) and Mach number (M). From these three conditions we derive inlet boundary conditions in terms of the primary variables of the system: density, velocity (u), and pressure (p). On the subsonic outlet boundary problem it is common to specify pressure. The variables

ρ, u, and p are computed and stored at each cell center and the conservative variables (ρ, ρu, ϵ) in U are derived from these when needed. The following two formulas are used in these conversions:

$$p = \rho RT \text{ and} \qquad M = \frac{|u|}{c}.$$

R is a physical constant equal to 287 $J/(kg\,K)$ [59] and the relation for p results from the ideal gas law.

The initial conditions are simply the inlet boundary conditions at each cell in the domain. Note that since the volume is changing in the domain (through $S(x)$) this initial condition may introduce shock waves to the solution on its way to steady-state as mass is redistributed in the domain. These transient effects do not change the steady-state solution.

Since both sets of boundary conditions use supersonic inlet conditions, we will describe that boundary first. We define the far-field boundary data as p_∞, T_∞, and M_∞. We then calculate ρ_∞ and u_∞ from the formulas above (note u is always positive for these boundary conditions). We then compute the ghost cell value for ρ by using the MUSCL formula (3.21) ($\Phi \equiv 1$) with $\rho^R_{0+\frac{1}{2}} = \rho_\infty$ and solving for ρ_0:

$$\rho_0 = \rho_1 - \frac{4(\rho_1 - \rho_\infty) - (1-\kappa)(\rho_2 - \rho_1)}{1+\kappa}.$$

We follow exactly the same procedure for u_0 and p_0. This defines how the far-field boundary conditions are used to compute the ghost-cell boundary data for the supersonic inlet boundary conditions. The next two headings discuss the two different outlet boundary

conditions.

Supersonic outlet

For the supersonic outlet boundary condition, we cannot specify any conditions on the boundary, but we still need ghost cell data. To determine the outlet ghost cell value for ρ, we used the MUSCL formula (3.20) ($\Phi \equiv 1$) with $\rho^L_{N+\frac{1}{2}} = \rho_N$ and solved for ρ_{N+1}:

$$\rho_{N+1} = \rho_N - \left(\frac{1-\kappa}{1+\kappa}\right)(\rho_N - \rho_{N-1}).$$

The remaining variables, u_{N+1} and p_{N+1}, follow the same way.

Subsonic outlet

As described above, we will specify a pressure, p_e, at the subsonic outlet, which provides a back-pressure to the system. This back-pressure induces a shock just past the center of the throat in the nozzle. This problem is much harder to solve because there is a real discontinuity in all the variables.

After we specify the back-pressure we need to compute the far-field boundary values for density and velocity. We do this by using a discretized form of the compatibility equations. We then take the far-field boundary conditions at the outlet and use MUSCL extrapolation as in the supersonic inlet case to compute the ghost cell values. Let the outlet far-field boundary values be ρ_{out}, u_{out}, and p_{out}. We also need to keep track of the speed of sound at the outlet, and will use a_{out} for that. The compatibility equations give

us the following equations for the far-field subsonic outlet conditions:

$$\begin{aligned} p_{out} &= p_e \\ a_{out} &= \sqrt{\frac{\gamma p_{out}}{\rho_{out}}} \\ \rho_{out} &= \rho_{out} - \left(\frac{2\delta_0 u_{out}}{\Delta x}\right)(\rho_{out} - \rho_N) - \left(\frac{1}{a_{out}}\right)(p_{out} - p_N) \\ u_{out} &= u_{out} - \left(\frac{2\delta_0}{\Delta x}\right)(u_{out} + a_{out})(u_{out} - u_N) + \left(\frac{1}{\rho_{out} a_{out}}\right)(p_{out} - p_N) \end{aligned} \tag{3.25}$$

To compute the outlet ghost cell value for ρ, we used the MUSCL formula 3.20 ($\Phi \equiv 1$) with $\rho^L_{N+\frac{1}{2}} = \rho_N$ and solved for ρ_{N+1}:

$$\rho_{N+1} = \rho_N - \frac{4(\rho_{out} - \rho_N) - (1-\kappa)(\rho_N - \rho_{N-1})}{1+\kappa}$$

The exact same procedure produces the u_{N+1} and p_{N+1} ghost cell data values.

3.5 Numerical Results

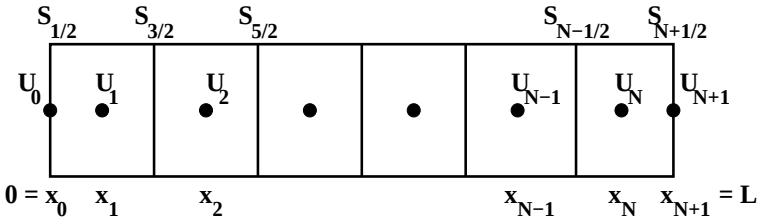
The physical domain in Figure 3.1 is discretized in space with a uniform mesh containing N cells with $\Delta x = L/N$. There are two ghost cells for a total of $N+2$ data points. Boundary conditions are applied at x_0 and x_{N+1}. Figure 3.6 shows the location of data

in the cells. The physical domain is a nozzle with the parameters:

$$\begin{aligned} L &= 2, \\ N &= 1200, \text{ and} \\ S(x) &= \left\{ \begin{array}{cc} 1+4(x-1)^2 & \text{if } 0.5 < x < 1.5 \\ 2 & \text{otherwise} \end{array} \right\}. \end{aligned}$$

For the following experiments, we specified the inlet far-field boundary conditions as:

Figure 3.6: Numerical setup for nozzle



$$\begin{aligned} p_\infty &= 101325, \\ T_\infty &= 300, \text{ and} \\ M_\infty &= 2.5 \end{aligned}$$

and the outlet back-pressure as

$$p_e = 900000.$$

Our numerical method is defined through the equations for $\frac{\partial \widehat{R}}{\partial U}$, (3.24), $R(U)$, (3.17),

and $F_{i\pm\frac{1}{2}}$, (3.22) resulting in the Ψtc iteration,

$$U_+ = U_c - \left(\delta_n^{-1} V + \frac{\partial \widehat{R}}{\partial U}(U_c)\right)^{-1} R(U_c) \tag{3.26}$$

and the Newton iteration,

$$U_+ = U_c - \left(\frac{\partial \widehat{R}}{\partial U}(U_c)\right)^{-1} R(U_c)$$

We used the hybrid Ψtc with the SER delta update formula and the hybrid Newton's method with the Armijo line search formula in this comparison. We did not attempt to solve this problem with a straight Newton's method, as the Newton-Armijo method had enough difficulty with this problem already. To choose the initial step size for Ψtc, we found the largest δ_0 that satisfied the local CFL condition for every cell, and we never allowed δ_n to shrink any smaller than this. This choice for δ_0 also appeared in [58] and is a problem-specific heuristic for choosing the initial step-size as was discussed in the conclusions of Chapter 2.

For a smooth problem, existing theory tells us that the iteration will either converge to a solution, diverge to infinity, or stagnate at a point where the Jacobian is singular [35]. For this hybrid Ψtc method, we have proven a similar result for γ_R sufficiently small (3.2).

3.5.1 Supersonic outlet

In the supersonic outlet case, the Newton line search fails by stagnation, as seen in the residual history and step size curves in Figure 3.7. In addition, the solution for pressure and density are clearly not symmetric. On the other hand the hybrid splitting method succeeds as seen by the residual history in Figure 3.8. The residual converges to zero rapidly, consistent with the conclusions of Theorem 3.2 when

$$\overline{\eta} \approx \max_n \|J(U_n) - R'_S(U_n)\| = O(\Delta x).$$

In comparison with the results from Ψtc in Figure 3.8, it appears that the residual had to increase at some point in order to resolve the solution correctly. This increase in the residual in not possible with the Armijo line search and therefore the step size decreased to zero.

Exact numerical solution

To check our numerical results for the supersonic outlet case, we computed the exact steady state solution using the isentropic gas equations along with conservation of total pressure and total temperature [1, 30]. One of the resulting equations is nonlinear and we used Newton's method to determine its solution on each cell. The solutions from Ψtc for the supersonic outlet case are qualitatively and quantitatively correct according to this exact steady state solution. Next we explain how the exact solution was computed.

Following thermodynamic relations, [1, 30] show that the area Mach number relation

is

$$\left(\frac{A}{A^*}\right)^2 = \frac{1}{M^2}\left[\frac{2}{\gamma+1}\left(1+\frac{\gamma-1}{2}M^2\right)\right]^{\frac{\gamma+1}{\gamma-1}}. \tag{3.27}$$

This relation tells us that the Mach number at any point in the nozzle is a function of the ratio of the area at that point and the area where the Mach number is unity, A^*. We use this area as a reference to compute the Mach number at other areas in the domain.

To compute A^*, we use the inlet Mach number, $M_\infty = 2.5$, and inlet area, $A_\infty = 2$.

$$A^* = A_\infty M_\infty \left[\frac{2}{\gamma+1}\left(1+\frac{\gamma-1}{2}M_\infty^2\right)\right]^{\frac{2(\gamma-1)}{\gamma+1}} \tag{3.28}$$

Now that we have A^*, we can compute the Mach number at any other point in the domain using the area at that point. Equation (3.27) is a nonlinear equation relating the area A and the Mach number M. We used Newton's method directly on this problem to solve for M using a Mach number of 2.5 as the initial iterate. Newton's method converged to a solution, $M = M_2$, in very few iterations with a relative tolerance of 10^{-8}.

To compute the primary variables (ρ_2, p_2, T_2) at each point in the domain we first need the primary variables at the inlet where we specified M_∞.

$$\begin{aligned} p_0 &= p_\infty \left(1+\tfrac{\gamma-1}{2}M_\infty^2\right)^{\frac{\gamma}{\gamma-1}}, \\ \rho_0 &= \rho_\infty \left(1+\tfrac{\gamma-1}{2}M_\infty^2\right)^{\frac{1}{\gamma-1}}, \text{ and} \\ T_0 &= T_\infty \left(1+\tfrac{\gamma-1}{2}M_\infty^2\right), \end{aligned}$$

where we used the following boundary values,

$$\begin{aligned} \rho_\infty &= \tfrac{p_\infty}{RT_\infty}, \\ p_\infty &= 101325, \\ T_\infty &= 300, \\ R &= 287, \text{ and} \\ \gamma &= 1.4. \end{aligned}$$

We then computed the primary variables at every point in the domain using the Mach number, M_2, from Newton's method above.

$$\begin{aligned} p_2 &= p_0 \left(1 + \tfrac{\gamma-1}{2} M_2^2\right)^{-\frac{\gamma}{\gamma-1}}, \\ \rho_2 &= \rho_0 \left(1 + \tfrac{\gamma-1}{2} M_2^2\right)^{-\frac{1}{\gamma-1}}, \text{ and} \\ T_2 &= T_0 \left(1 + \tfrac{\gamma-1}{2} M_2^2\right)^{-1}. \end{aligned}$$

And finally we computed velocity at each point using,

$$u_2 = M_2 \sqrt{\gamma R T_2}.$$

This procedure describes the supersonic inlet and supersonic outlet solution, and allows us to calculate a numerical exact solution to compare with.

3.5.2 Subsonic outlet

For the subsonic outlet case, the Newton line-search method was also not able to solve this problem. It failed with a line-search failure on the second iteration. On the other hand, Ψtc succeeded, though it took many iterations to do so, partly because it had to grow the shock from smooth initial conditions. Figure 3.10 shows the Ψtc results for this shock problem. Figure 3.9 shows the Newton results. The results for Ψtc are qualitatively correct according to the theory for fluid flow through a converging-diverging nozzle [1].

3.6 Conclusions

In this chapter we have examined the one dimensional Euler equations applied to a converging-diverging nozzle with supersonic inlet and supersonic or subsonic outlet. This is a classical problem in computational fluid dynamics and has had much analysis applied to it. It is a very interesting problem from an algorithmic point of view because Newton's method was not able to solve the problem at all. It is most likely failures like this that caused many people in the computational fluid dynamics community to leave Newton's method in favor of Ψtc. This formulation is also a nice problem to work with because it exhibits a smooth underlying problem with a non-smooth variation added. This theme of Ψtc behaving well with non-smooth effects will continue into the next chapter. Chapter 4 is a departure from the ODE work so far and delves into differential-algebraic equations (DAEs). DAEs come up in many fields and often have the same non-smooth effects which give Newton's method problems.

Figure 3.7: Newton's method for 1-D Euler Equations with supersonic outlet

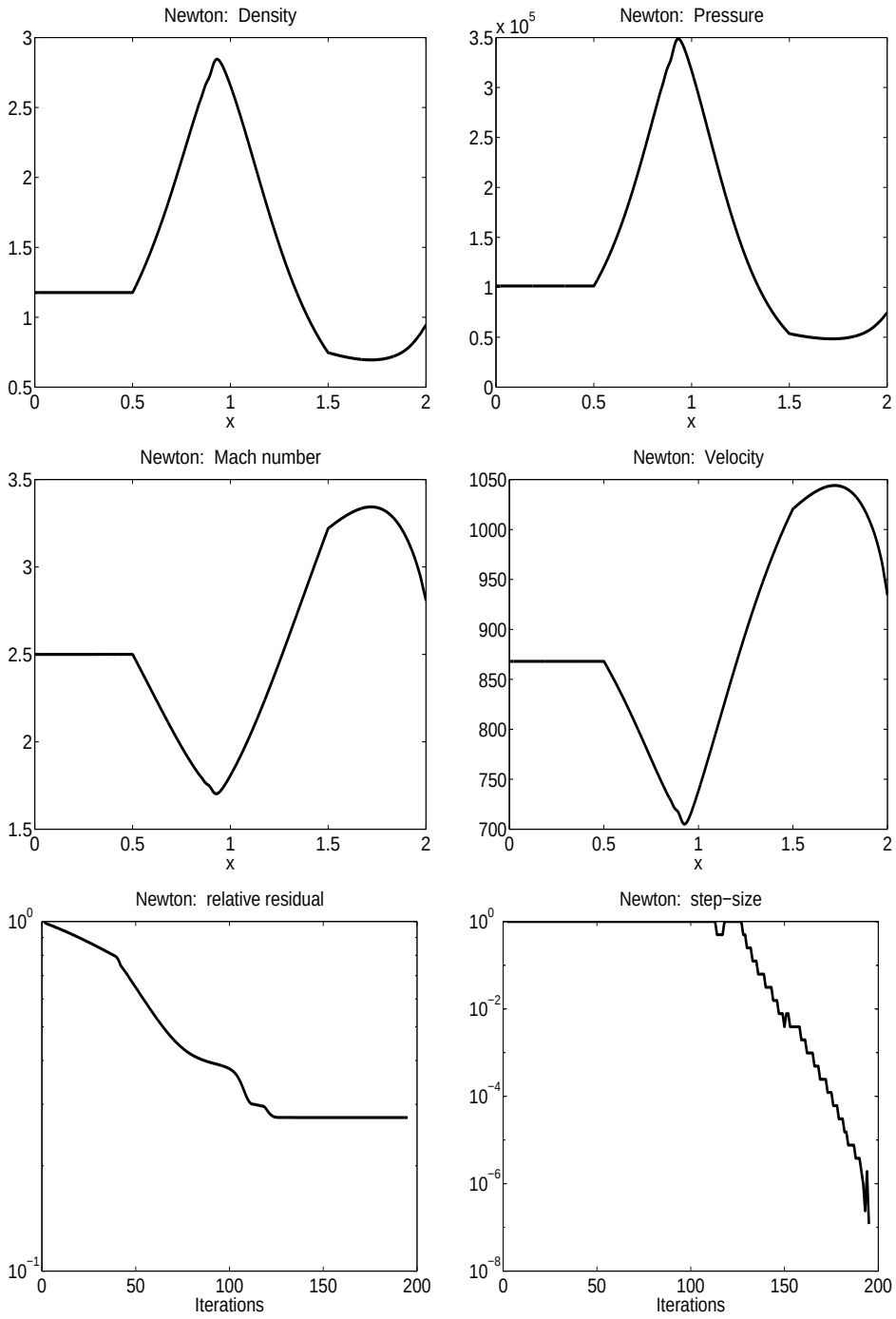


Figure 3.8: Ψtc method for 1-D Euler Equations with supersonic outlet

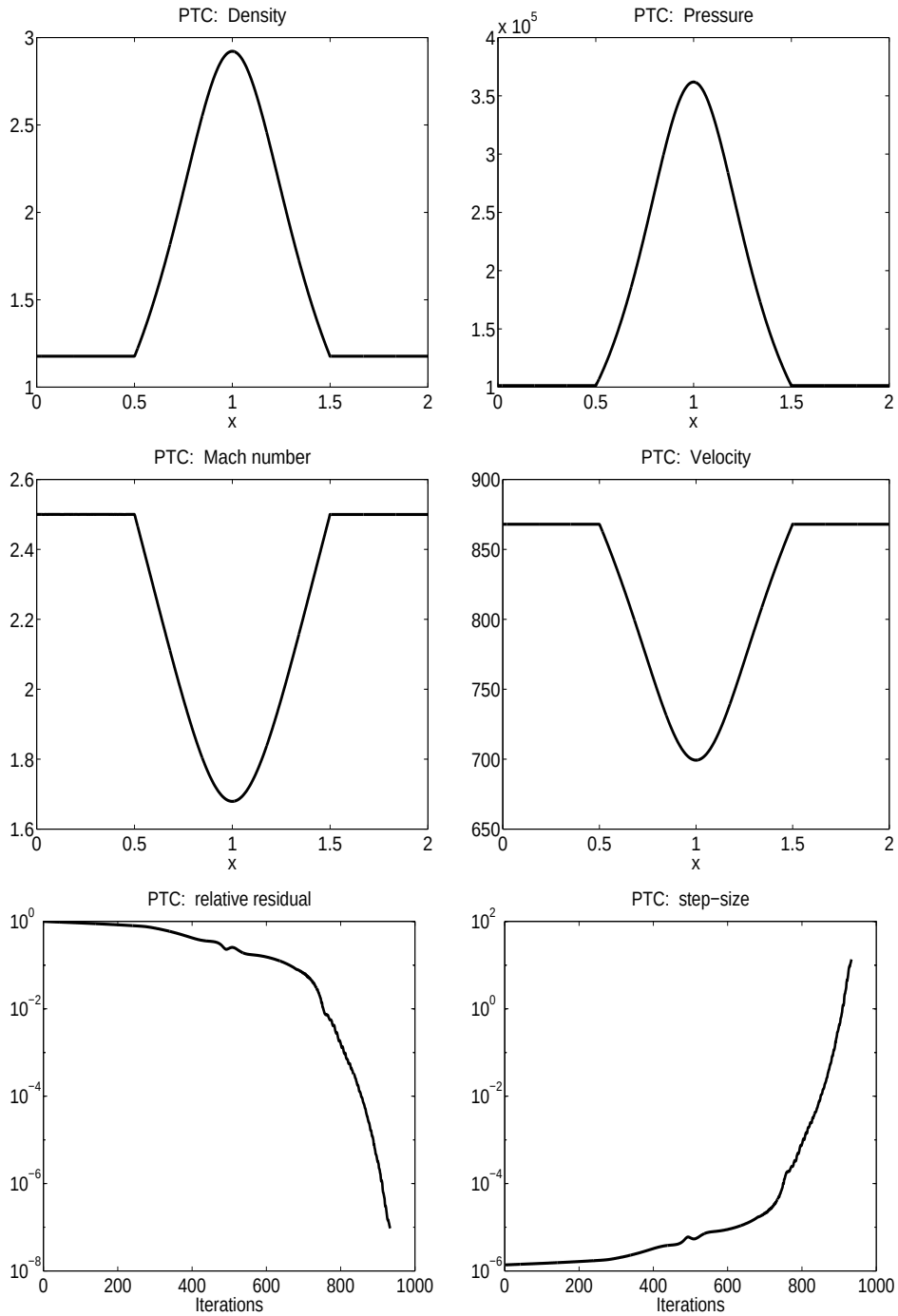


Figure 3.9: Newton-Armijo method for the 1-D Euler equations in a nozzle with subsonic outlet boundary condition

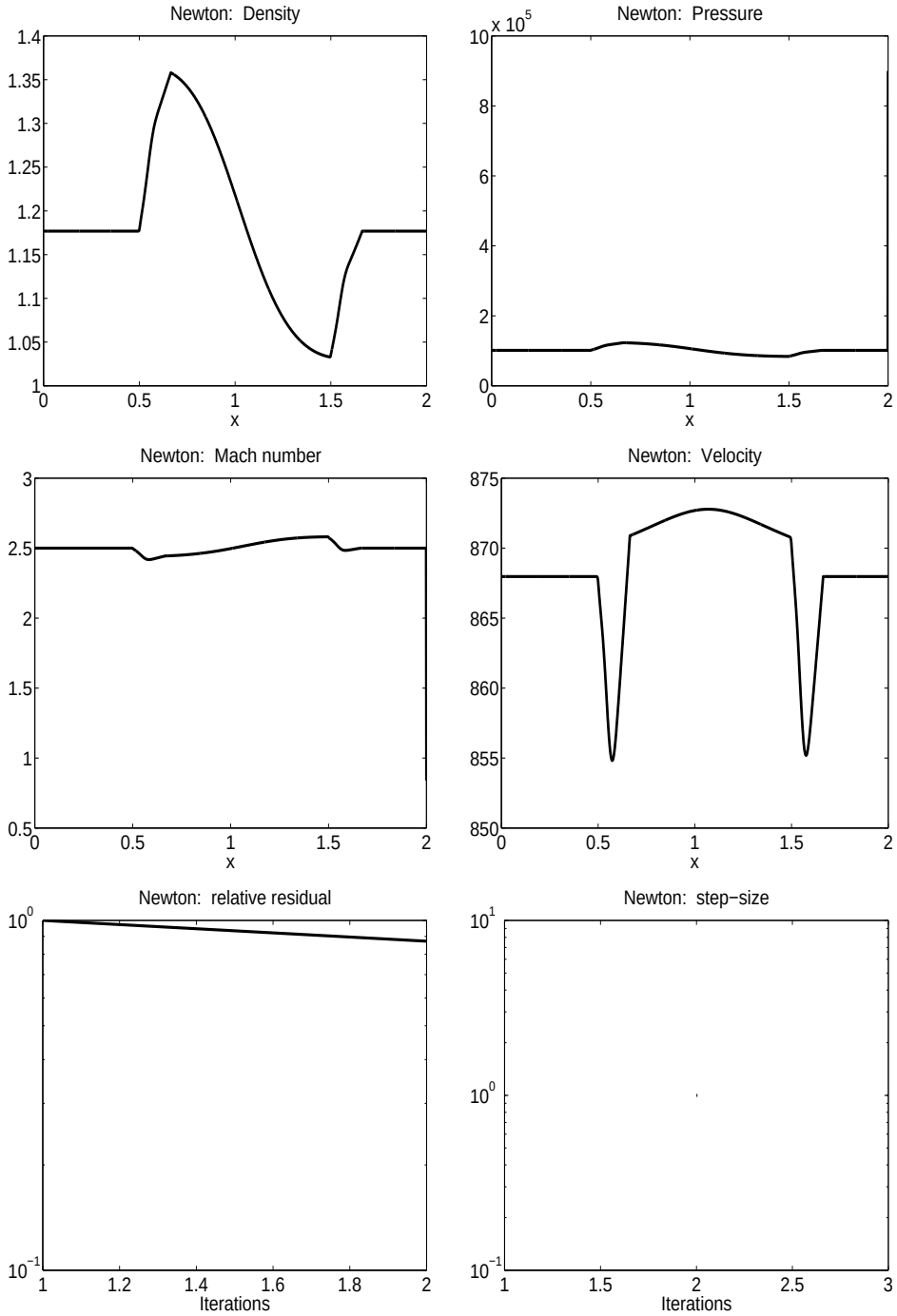
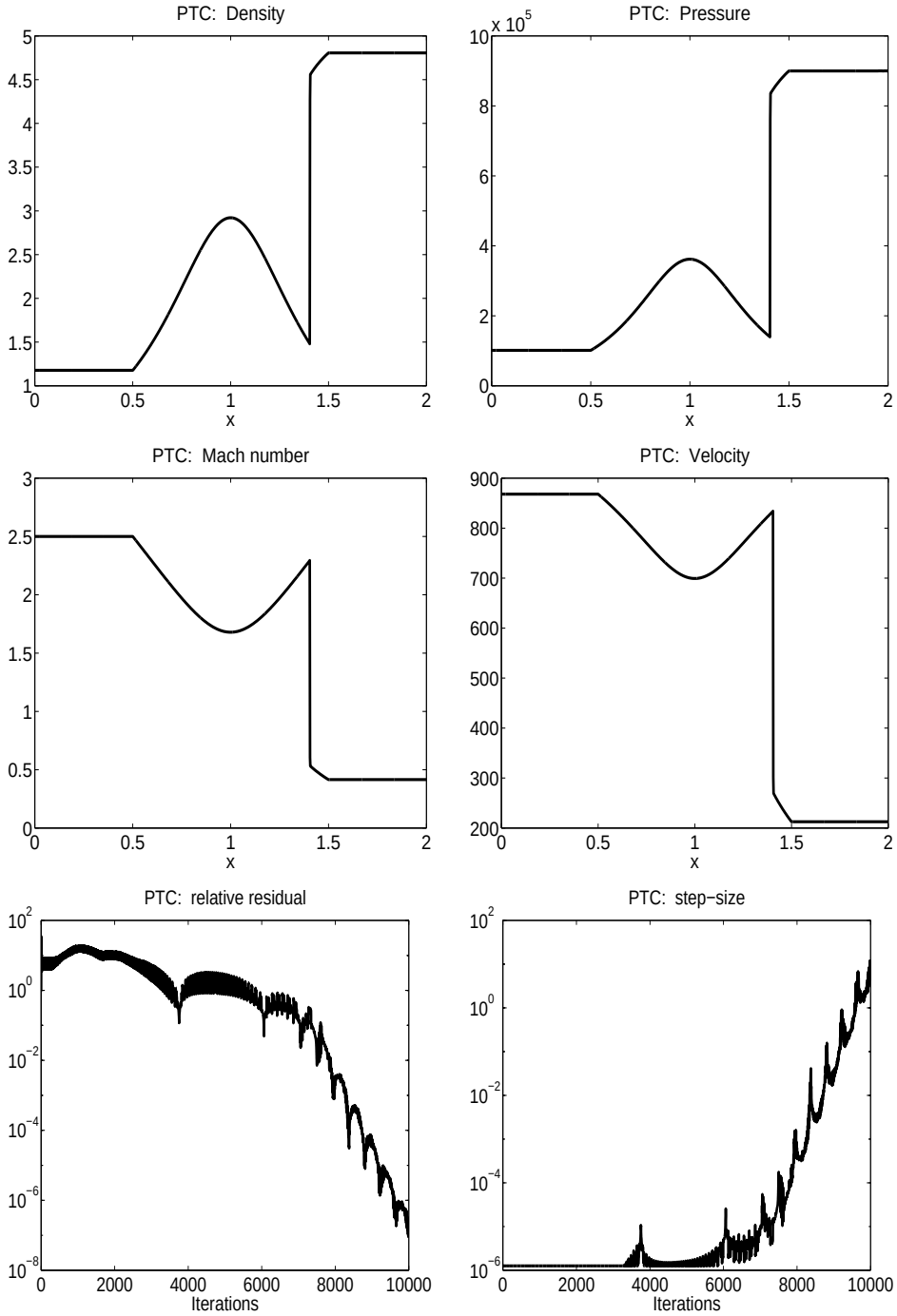


Figure 3.10: Ψtc method for the 1-D Euler equations in a nozzle with subsonic outlet boundary condition



Chapter 4

Differential-Algebraic Equation Extension

4.1 Introduction

Many problems are best formulated, or can only be formulated, as differential-algebraic equations (DAEs) [9]. These are differential equations with algebraic constraints. As an example, we provide a combustion model from computational fluid dynamics in section 4.3.2 that is cast as a DAE for physical reasons. Pseudo-transient continuation is a good method for this problem, but no theory exists to support its use here.

In this chapter we extend the current theory for Ψtc to include a global convergence

result for semi-explicit index-1 DAEs:

$$\begin{aligned} Vu' &= -f(u,v) \\ 0 &= g(u,v) \end{aligned} \tag{4.1}$$

with initial value $(u(0), v(0)) = (u_0, v_0)$, $u \in \mathbb{R}^{N_1}$, $v \in \mathbb{R}^{N_2}$, where V is an $N_1 \times N_1$ fixed nonsingular scaling matrix and $\partial g/\partial v$ is nonsingular (which defines it as index-1). Our result follows from the work in [36], and we prove two theorems establishing global convergence of the Ψtc iteration.

4.2 Global Convergence

We write the DAE (4.1) as a system in matrix form, $x = [u, v]^T \in \mathbb{R}^N$, $N = N_1 + N_2$, $F = [f(u,v), g(u,v)]^T$ and $D = \left(\begin{smallmatrix} V & 0 \\ 0 & 0 \end{smallmatrix}\right)$. The system is then,

$$Dx' = -F(x), \quad x(0) = x_0. \tag{4.2}$$

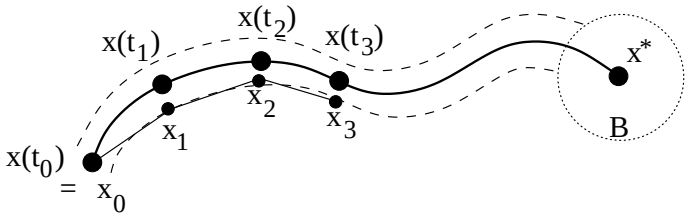
The pseudo-transient continuation procedure (Ψtc) is defined by the iteration

$$x_{n+1} = x_n - (\delta_n^{-1} D + F'(x_n))^{-1} F(x_n). \tag{4.3}$$

The ODE convergence proof in [36] is broken into three phases. The first phase is far from steady-state, where we iterate in a time-marching mode to get the solution closer

to steady-state. In the second phase, we establish convergence by growing our step-size while maintaining temporal stability. In the third phase, we establish asymptotic Newton convergence using the large step-size and solution provided by the previous phase. These phases are explained graphically in Figure 4.1. In this figure, we start out at x_0 and Ψtc produces the iterates x_1, x_2, x_3, etc. The solid line corresponds to the exact solution with corresponding exact solution points $x(t_1), x(t_2), x(t_3)$, etc. The dotted lines represent how accurate we must be to remain stable with the corresponding volume denoted S. The first phase is responsible for staying inside the dotted lines. The second phase starts when we get inside the ball B which represents the ball of local Newton convergence. When the sequence of Ψtc iterates move into this ball, we can switch over to Newton's method and it will converge to the steady-state solution x^*. The third phase establishes asymptotic Newton convergence from inside the ball B. In this DAE convergence analysis, we will lump phase II and phase III together and then deal with phase I on its own.

Figure 4.1: Schematic for the phases of convergence for Ψtc



We present the local convergence theory first (phase II and III), then the global theory (phase I). First we assume that our DAE is index-1, has a global solution in time, and

that this solution converges to a steady-state.

Assumption 4.1. *Let the following hold,*

1. *The initial values* (u_0, v_0) *are consistent, i.e.* $g(u_0, v_0) = 0$.

2. F *is twice Lipschitz continuously differentiable.*

3. *The solution* $x(t) = [u(t), v(t)]^T$ *of (4.3) exists for all* $t > 0$ *and* $\lim_{t\to\infty} x(t) = x^*$.

4. $g_v(u(t), v(t)) = g_v(x(t))$ *is a nonsingular* $N_2 \times N_2$ *matrix for all* $t \geq 0$ *and* $\|g_v(x(t))^{-1}\|$ *is uniformly bounded on* $(0, \infty)$.

The global analysis rests on the fact that the Ψtc iteration remains close to the exact solution of (4.2), so we define a ball S around the exact solution,

$$S(\epsilon) = \{z | \inf_{t\geq 0} \|z - x(t)\| \leq \epsilon\}. \tag{4.4}$$

Given $\epsilon > 0$, we will show that provided δ_0 is sufficiently small, the Ψtc iteration will stay within $S(\epsilon)$ for all n and accurately approximates the exact solution $\{x(t_n)\}$ until the iteration is within the local ball of convergence for Newton's method. This corresponds to phase I of the iteration as described above for the ODE theory [36]. Once the Ψtc iterates are near the x^*, one must show that δ_n will increase to improve our convergence rate without causing the iteration to diverge. Once this intermediate phase is complete, the final phase establishes local Newton convergence rates [36].

First we need to assume that the linear systems to be solved, are nonsingular and that the steady-state solution of the DAE is stable.

Assumption 4.2. *There are $\epsilon_S, \epsilon_G > 0$ such that*

1. *For all $z_0 \in S(\epsilon_S)$, the solution of $Dz' = -F(z), z(0) = z_0$ exists, $z(t) \in S(\epsilon_G)$ for all t, and $\lim_{t\to\infty} z(t) = x^*$.*

2. *Moreover, there are $M_D, M_I > 0$ such that for all $\delta > 0$.*

   (a) $(\delta^{-1}D + F'(x))$ is nonsingular for all $x \in S(\epsilon_G)$.

   (b) $\|(\delta^{-1}D + F'(x))\| \leq M_D$ for all $x \in S(\epsilon_G)$, and

   (c) $\|(\delta^{-1}D + F'(x))^{-1}\| \leq M_I$ and

   $\|(D + \left(\begin{smallmatrix} \delta I & 0 \\ 0 & I \end{smallmatrix}\right) F'(x))^{-1}\| \leq M_I$ *for all $x \in S(\epsilon_G)$.*

4.2.1 Phase II and III: Local Convergence

Let,

$$B(\epsilon) = \{x | \, \|x - x^*\| < \epsilon\} \tag{4.5}$$

be the ball representing local Newton convergence (for ϵ small enough). As in the ODE theory, we need to assume continuity and differentiability of F and nonsingularity of F' near x^*.

Assumption 4.3. *There are $\beta, \epsilon_L > 0$ such that for all $x \in B(\epsilon_L)$ and for all $\delta > 0$,*

$$\|(D + \delta F'(x))^{-1} D\| \leq \frac{1}{1 + \beta\delta}.$$

This assumption is the DAE analog of Assumption 2.1.3 in [36], and is critical for

establishing stability in phase II of the iteration. The following theorem describes phase II and phase III of the iteration in the same way that Theorems 2.1 and 2.3 of [36] do for the ODE case.

Theorem 4.1. *Let assumptions 4.1, 4.2, 4.3 hold. Let* $\{\delta_n\}$ *be given by (1.2). Then there are* $C_T, \epsilon_T > 0$ *so that if* $x_0 \in B(\epsilon_T)$ *then either* $\inf_n \delta_n = 0$ *or* $\delta_n \to \delta_{max}$*, the* Ψtc *iteration converges, and for* n *sufficiently large*

$$\|x_{n+1} - x^*\| \leq C_T \|x_n - x^*\| \left(\delta_{max}^{-1} + \|x_n - x^*\|\right). \tag{4.6}$$

Proof. We describe the progress of the iteration in terms of the transition from the current iterate x_c to a new one x_+ [35]. Let the error be denoted by $e = x - x^*$. Let $\epsilon_T < \epsilon_L$ be small enough so that the local convergence theory for Newton's method holds for $F(x) = 0$. If $x_c \in B(\epsilon_T)$ then,

$$\begin{aligned} e_+ &= e_c - (\delta_c^{-1} D + F'(x_c))^{-1} F(x_c) \\ &= e_c - (\delta_c^{-1} D + F'(x_c))^{-1} (\delta_c^{-1} D + F'(x_c)) e_c \\ &\quad + (\delta_c^{-1} D + F'(x_c))^{-1} (F'(x_c) e_c - F(x_c)) \\ &\quad + (\delta_c^{-1} D + F'(x_c))^{-1} \delta_c^{-1} D e_c. \end{aligned} \tag{4.7}$$

The standard local convergence theory for Newton's method and Assumption 4.2 imply that there is $C_N > 0$ such that

$$\|(\delta_c^{-1} D + F'(x_c))^{-1} (F'(x_c) e_c - F(x_c))\| \leq C_N \|e_c\|^2$$

and

$$\|e_c - (\delta_c^{-1}D + F'(x_c))^{-1}(\delta_c^{-1}D + F'(x_c))e_c\| \leq C_N\|e_c\|^2.$$

Assumption 4.3 implies that

$$\|(\delta_c^{-1}D + F'(x_c))^{-1}\delta_c^{-1}De_c\| \leq \frac{\|e_c\|}{1+\beta\delta_c}.$$

Now, assuming $\{\delta_n\}$ does not converge to zero, and is bounded from below by δ^*,

$$\|e_+\| \leq (2C_N\|e_c\| + (1+\beta\delta^*)^{-1})\|e_c\|.$$

Reduce ϵ_T if needed so that

$$(2C_N\epsilon_t + (1+\beta\delta^*)^{-1}) < (1+\beta\delta^*/2)^{-1}.$$

Then, provided δ_n is bounded from below, the local convergence is q-linear. This q-linear convergence rate will eventually drive δ_n to δ_{max}, if $\delta_{max} < \infty$. Hence, the standard local convergence theory for Newton's method [35] establishes our conclusion, (4.6). □

4.2.2 Phase I: Global Convergence

In this phase, we start with a small time-step δ_n, and a solution that is far from steady-state. Our goal is to get our solution into the ball of Newton's convergence. We now state and prove our theorem corresponding to the initial phase of the Ψtc iteration.

Theorem 4.2. *Let $\{\delta_n\}$ be given by the SER delta update formula (1.2) and let Assumptions 4.1, 4.2, and 4.3 hold. For any $\epsilon > 0$ there is a $\widehat{\delta}$ such that if $\delta_0 \leq \widehat{\delta}$ then there is an n such that $x_n \in B(\epsilon)$.*

Proof. Let $\epsilon < \epsilon_T$ be sufficiently small to satisfy local Newton convergence. Then there exists $\epsilon_N < \epsilon_S$ and ϵ_F such that if $x \in S(\epsilon_N)$ and $\|F(x)\| < \epsilon_F$ then $x \in B(\epsilon)$. Let,

$$M = \sup_{x \in S(\epsilon_N)} \|F(x)\|.$$

We will show that if δ_0 is sufficiently small, then $x_n \in S(\epsilon_N)$ until $\|F(x_n)\| < \epsilon_F$. By the SER δ update formula, if

$$\delta_0 \|F(x_0)\|/\epsilon_F < \xi_t,$$

then

$$C_L \delta_0 \equiv \frac{\|F(x_0)\|}{M}\delta_0 \leq \delta_n \leq \frac{\|F(x_0)\|}{\epsilon_F}\delta_0 \equiv C_U \delta_0 \tag{4.8}$$

holds for $\|F(x_n)\| \geq \epsilon_F$ and $x_n \in S(\epsilon_N)$. Let $0 < T < \infty$ be such that for all $t > T$,

$$\|x - x(t)\| < \epsilon_N \Longrightarrow \|F(x)\| < \epsilon_F.$$

Consider the approximate integration of (4.2) by (4.3),

$$t_{n+1} = \sum_{l=0}^{n} \delta_l > (nC_L + 1)\delta_0.$$

If $\|x_n - x(t_n)\| < \epsilon_N$ and $\|F(x_n)\| \geq \epsilon_F$ then (4.8) holds. But (4.8) does not hold if $n > (T - \delta_0)/(C_L\delta_0)$ (which implies $t_n > T$). Therefore the proof will be complete if we can show that

$$\|x_n - x(t_n)\| < \epsilon_N$$

for all

$$n \leq T/(C_L\delta_0). \tag{4.9}$$

Let $E_n = \|x_n - x(t_n)\|$. We will show by induction that if δ_0 is sufficiently small then $E_n < \epsilon_N$. The Ψtc iteration (4.3) can be written,

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} - \begin{bmatrix} V + \delta_n f_u & \delta_n f_v \\ g_u & g_v \end{bmatrix}^{-1} \begin{pmatrix} \delta_n f \\ g \end{pmatrix}, \tag{4.10}$$

where each function is evaluated at (u_n, v_n). From here on, functions with no arguments are evaluated at (u_n, v_n). We now write out E_{n+1}, replace $(u_{n+1}, v_{n+1})^T$ with the Ψtc iteration above and expand $(u(t_{n+1}), v(t_{n+1}))^T$ in a Taylor series about t_n.

$$E_{n+1} \leq E_n + \left\| \begin{bmatrix} V + \delta_n f_u & \delta_n f_v \\ g_u & g_v \end{bmatrix}^{-1} \begin{pmatrix} \delta_n f \\ g \end{pmatrix} + \begin{pmatrix} u'(t_n) \\ v'(t_n) \end{pmatrix} \delta_n + O(\delta_n^2) \right\|.$$

We now factor out the inverted matrix (2c) in Assumption 4.2 and expand:

$$E_{n+1} \quad \leq \quad E_n + M_I \left\| \begin{pmatrix} \delta_n f \\ g \end{pmatrix} + \begin{bmatrix} V + \delta_n f_u & \delta_n f_v \\ g_u & g_v \end{bmatrix} \begin{pmatrix} u'(t_n) \\ v'(t_n) \end{pmatrix} \delta_n \right\| + O(\delta_n^2)$$

$$\begin{aligned} E_{n+1} \quad \leq \quad & E_n + O(\delta_n^2) + M_I \, (&& \text{(4.11a)} \\ & \left\| \delta_n f + (V + \delta_n f_u) u'(t_n) \delta_n + \delta_n^2 f_v v'(t_n) \right\| && \text{(4.11b)} \\ & + \| g + g_u u'(t_n) \delta_n + g_v v'(t_n) \delta_n \|) \, . && \text{(4.11c)} \end{aligned}$$

To arrive at a bound on this vector, we will need the ODE that results from solving for v' directly,

$$\begin{aligned} u' \quad &= \quad -V^{-1} f(u, v) \\ v' \quad &= \quad g_v^{-1}(u, v) g_u(u, v) V^{-1} f(u, v). \end{aligned} \tag{4.12}$$

By substituting u' from (4.12) into (4.11b) and rewriting we get the bound on (4.11b),

$$\| f(u_n, v_n) - f(u(t_n), v(t_n)) \| \delta_n + \left\| \begin{pmatrix} f_u \\ f_v \end{pmatrix}^T \begin{pmatrix} u'(t_n) \\ v'(t_n) \end{pmatrix} \right\| \delta_n^2$$

Let γ be the Lipschitz continuity parameter for f. We then have the final bound on (4.11b),

$$(4.11b) \leq \gamma \delta_n E_n + C \delta_n^2,$$

where C includes M_I and the bound on $\|(u'(t), v'(t))^T\|$, which comes from (4.12). Now, consider (4.11c). The second two terms in this norm appear very similar to the total derivative of g. We will add and subtract two terms to bring this out and then bound the remaining terms. Replace g_u and g_v with,

$$g_u = g_u + g_u(u(t_n), v(t_n)) - g_u(u(t_n), v(t_n)),$$
$$g_v = g_v + g_v(u(t_n), v(t_n)) - g_v(u(t_n), v(t_n)),$$

and rewrite,

$$\|g + g_u u'(t_n)\delta_n + g_v v'(t_n)\delta_n\| \leq \|g\| + \left\| \begin{pmatrix} g_u - g_u(u(t_n), v(t_n)) \\ g_v - g_v(u(t_n), v(t_n)) \end{pmatrix}^T \begin{pmatrix} u'(t_n) \\ v'(t_n) \end{pmatrix} \right\| \delta_n,$$

where we note that the remaining terms are zero because the total derivative of g is zero along the solution. Now, we use Lipschitz continuity of the gradient of g, with Lipschitz parameter Γ to get:

$$\|g + g_u u'(t_n + \zeta)\delta_n + g_v v'(t_n + \zeta)\delta_n\| \leq \|g\| + \Gamma\delta_n E_n C.$$

Now, we just need a bound on $\|g(u_n, v_n)\|$. Let θ_n be defined by the implicit function theorem so that for all u near u_n, $g(u, \theta_n(u)) = 0$. We will show in Lemma 4.1 that

$\|v_n - \theta_n(u_n)\| = O(\delta_0^2)$. With this result we have,

$$g(u_n, v_n) = g(u_n, \theta_n(u_n)) + g_v(u_n, \theta_n(u_n))(v_n - \theta_n(u_n)) + O(\|v_n - \theta_n(u_n)\|^2)$$

and,

$$\|g(u_n, v_n)\| \leq O(\delta_0^2) + O(\delta_0^4),$$

where we have used a first order Taylor expansion of $g(u_n, v_n)$ about $(u_n, \theta_n(u_n))$. Therefore, there exist constants M_1 and M_2 such that

$$E_{n+1} \leq (1 + M_1\delta_0)E_n + M_2\delta_0^2.$$

Here we have used the fact that $\delta_n \leq C_U\delta_0$ from (4.8). The next Lemma establishes the claim above that $\|v_n - \theta_n(u_n)\| = O(\delta_0^2)$.

Following standard practices in [27] we argue that,

$$E_n \leq \delta_0 \frac{M_2}{M_1} \left(e^{nM_1\delta_0} - 1\right).$$

Since n is bounded by (4.9),

$$E_n \leq \delta_0 \frac{M_2}{M_1} \left(e^{TM_1/C_L} - 1\right).$$

Therefore, if

$$\delta_0 < \epsilon_N \frac{M_1}{M_2[\exp(TM_1/C_L) - 1]}$$

then $F(x_n) < \epsilon_F$ or $t_n > T$. □

Lemma 4.1. *Let (u_n, v_n) be defined by the DAE Ψtc sequence (4.3) and let Assumptions 4.1 and 4.2 hold. Let $\theta_n \in C^1(W_n)$ be defined by the implicit function theorem on the open neighborhood W_n about u_n such that for all $u \in W_n$, $g(u, \theta_n(u)) = 0$. Then,*

$$\forall N \ s.t. \ t_N \leq T, \exists K \ s.t. \ \forall n \leq N, \ \|v_n - \theta_n(u_n)\| \leq K\delta_0^2 + O(\delta_0^3).$$

Proof. The Ψtc iteration can be viewed as a predictor-corrector iteration where the predictor is the previous iteration $(u_n, v_n)^T$ and the corrector is a single Newton iteration to solve:

$$G \begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} V(u_{n+1} - u_n) + \delta_n f(u_{n+1}, v_{n+1}) \\ g(u_{n+1}, v_{n+1}) \end{pmatrix} = 0. \tag{4.13}$$

Since we are only applying one Newton iteration, we do not converge to the root of G, $(u^*_{n+1}, v^*_{n+1})^T$. For δ_n sufficiently small, $(u_n, v_n)^T$ is in the ball of quadratic Newton convergence to $(u^*_{n+1}, v^*_{n+1})^T$. So,

$$\begin{aligned} \|u_{n+1} - u^*_{n+1}\| &\leq K_n \|u_n - u^*_{n+1}\|^2 \\ \|v_{n+1} - v^*_{n+1}\| &\leq K_n \|v_n - v^*_{n+1}\|^2. \end{aligned}$$

Consider the following string of inequalities:

$$\begin{aligned} \|v_{n+1} - \theta_{n+1}(u_{n+1})\| &\leq \|v_{n+1} - v^*_{n+1}\| + \|v^*_{n+1} - \theta_{n+1}(u_{n+1})\| \\ &= \|v_{n+1} - v^*_{n+1}\| + \|\theta^*_{n+1}(u^*_{n+1}) - \theta_{n+1}(u_{n+1})\|. \end{aligned}$$

We used the fact that $(u^*_{n+1}, v^*_{n+1})^T$ solves G, and hence solves g. If δ_n is made sufficiently smaller, $\theta^*_{n+1}(u^*_{n+1}) = \theta_{n+1}(u^*_{n+1})$. This reduces us to the expression,

$$\|v_{n+1} - \theta_{n+1}(u_{n+1})\| \leq \|v_{n+1} - v^*_{n+1}\| + \|\theta_{n+1}(u^*_{n+1}) - \theta_{n+1}(u_{n+1})\|. \tag{4.14}$$

We now establish uniform Lipschitz continuity of $\{\theta_n\}$. For $\theta_n \in C^1(W_n)$ and $u \in W_n$, $g(u, \theta_n(u)) = 0$. Take the derivative of this expression with respect to u to obtain,

$$\|\theta'_n(u)\| \leq \left\| \frac{\partial g}{\partial u}(u, \theta_n(u)) \right\| \left\| \frac{\partial g}{\partial v}(u, \theta_n(u))^{-1} \right\| \leq C,$$

where C includes the bound on F' and g_v^{-1}. Therefore, since θ'_n is bounded independently of n (provided $t_n \leq T$), the $\{\theta_n\}$ sequence is Lipschitz continuous with parameter σ, where σ is independent of n. Using Lipschitz continuity of θ_n and quadratic convergence of Newton's method, our expression (4.14) becomes,

$$\begin{aligned} \|v_{n+1} - \theta_{n+1}(u_{n+1})\| &\leq K_n \|v_n - v^*_{n+1}\|^2 + \sigma K_n \|u^*_{n+1} - u_n\|^2. \\ &\leq K_n \left(\|v_n - \theta_n(u_n)\| + \|\theta_n(u_n) - v^*_{n+1}\| \right)^2 \\ &\quad + \sigma K_n \|u^*_{n+1} - u_n\|^2. \end{aligned} \tag{4.15}$$

The first term in (4.15) is recursive and after bounding the remaining two terms we will bound the recursion in terms of δ_0. For the last term in (4.15), note that since $(u^*_{n+1}, v^*_{n+1})^T$ solves G (4.13),

$$\|u^*_{n+1} - u_n\| \leq \|V^{-1}\| M \delta_n,$$

where M is the same as in Theorem 4.2. Using this, note that if δ_n is sufficiently small, $\theta^*_{n+1}(u^*_{n+1}) = \theta_n(u^*_{n+1})$. Therefore the second term becomes,

$$\|\theta_n(u_n) - \theta_n(u^*_{n+1})\| \leq \sigma \|u_n - u^*_{n+1}\| \leq \sigma \|V^{-1}\| M \delta_n.$$

Our recursion is then,

$$L_{n+1} \leq K_n \left((L_n + M_R \delta_0)^2 + (M_R \delta_0)^2 \right). \tag{4.16}$$

where $L_n = \|v_n - \theta_n(u_n)\|$ and $M_R = \max(\sqrt{\sigma}, \sigma)\|V^{-1}\| M C_U$. Note that $L_0 = 0$ since $(u_0, v_0)^T$ is consistent. To simplify the expression, let $\overline{K} = \max\{K_1, \cdots, K_N\}$ for N such that $t_N \leq T$. Expanding the recursion (4.16) and canceling terms we derive the expression,

$$\|v_n - \theta_n(u_n)\| \leq K \delta_0^2 + O(\delta_0^3),$$

where $K = 2\overline{K} M_R^2$. The O term contains terms of the form: $\overline{K}^{n-1}(M_R \delta_0)^n$. □

4.3 Applications and Numerics

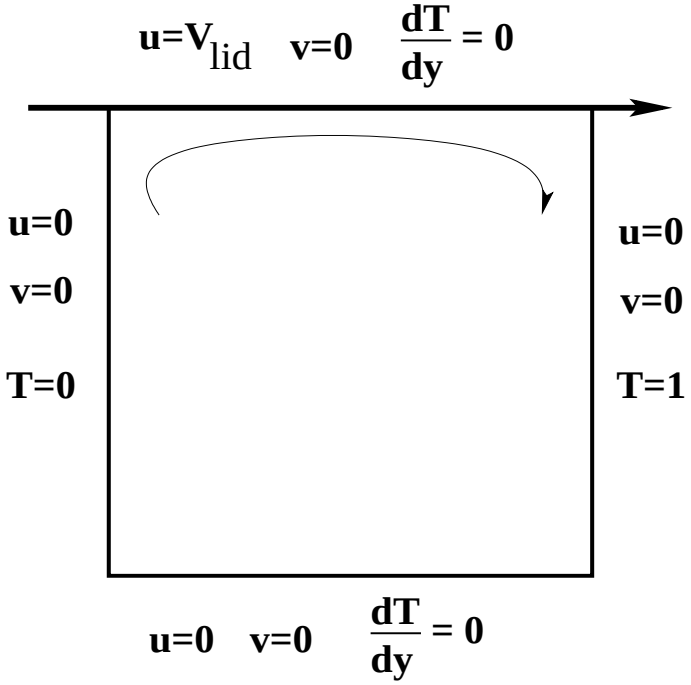
In this section we present two numerical experiments with differential-algebraic equations. The first is a lid-driven buoyancy-driven cavity problem that can be configured as a DAE or as a fully-parabolized set of partial differential equations. The second is a much more difficult problem from combustion. The next two sections describe the problems and the algorithmic numerical results.

4.3.1 Lid Problem

The diagram in Figure 4.2 shows the configuration of the lid-driven buoyancy-driven cavity problem. The lid moves with a constant velocity V_{lid}, and there is a temperature differential from the left vertical wall to the right vertical wall. This problem is part of the release of the `PETSc` [5] toolkit. This numerical experiment was conducted by David Keyes while in production of [15]. The motion of the lid sets up a primary vortex and the viscous forces set up secondary vortices. In opposition to the primary vortex, the temperature differential of the vertical walls drives a buoyant vortex flow. The development and interaction of these vortices is governed by the following equations:

$$\begin{aligned} -\Delta u - \frac{\partial \omega}{\partial y} &= 0, \\ -\Delta v + \frac{\partial \omega}{\partial x} &= 0, \\ -\Delta \omega + u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} - \mathrm{Gr}\frac{\partial T}{\partial x} &= 0, \text{ and} \\ -\Delta T + \mathrm{Pr}\left(u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}\right) &= 0. \end{aligned} \tag{4.17}$$

Figure 4.2: Lid-Driven Buoyancy-Driven Problem – boundary conditions



In these equations, u and v are the x and y velocity fields, ω is the vorticity normal to the xy plane, T is temperature, Gr is the Grashof number (a problem specific constant which is proportional to the ratio of buoyancy force to viscous force), and Pr is the Prandtl number (a problem specific constant which is proportional to the ratio of momentum diffusivity to thermal diffusivity). The boundary conditions are explained in Figure 4.2, and are written out here:

1. Along the bottom $(0 < x < 1, y = 0)$: $u = v = 0, \frac{\partial T}{\partial y} = 0$.

2. Along the top $(0 < x < 1, y = 1)$: $u = V_{lid}, v = 0, \frac{\partial T}{\partial y} = 0$.

3. Along the left $(x = 0, 0 < y < 1)$: $u = v = 0, T = 0$.

4. Along the right ($x = 1, 0 < y < 1$): $u = v = 0, T = 1$.

In addition, we use the definition of $\omega(x, y) = -\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$ on the boundaries.

For initial conditions (or initial iterate), we use a motionless, vorticity free initial flow field. Note that this initial condition is incompatible with the boundary conditions on a set of measure zero for the continuous space problem. To discretize the problem, we used a standard uniform Cartesian grid with a vertex-centered 5-point stencil for each component. First order upwinding was used on the convective terms, and centered differencing on the other first-order derivatives.

`PETSc` has a variety of nonlinear solvers to choose from, and we used the Preconditioned Jacobian free Newton-Krylov solver, with block ILU preconditioner. The computation of the Jacobian vector product was accomplished by use of automatic differentiation with `ADIC` [33].

We solved this problem twice, once as a fully parabolized partial differential-algebraic equation and once as a partial differential-algebraic equation. When we write the equations (4.17) in a form suitable for Ψtc:

$$u = (u, v, \omega, T), \qquad F = (f_1, f_2, f_3, f_4), \text{ and}$$

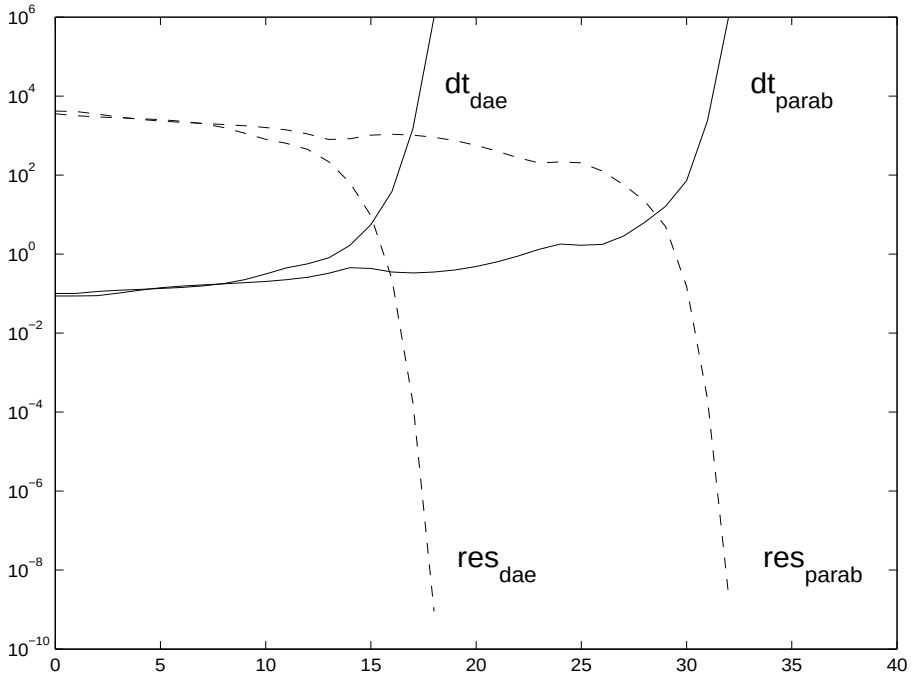
$$Eu' = -F(u).$$

The fully parabolized form corresponds to $E = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}$ and the differential-algebraic form corresponds to $E = \begin{pmatrix} 0 & & & \\ & 0 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}$.

The residual norm ($\|F(x_n)\|$) and step-size (δ_n) histories for Ψtc applied to the steady-state solution of this cavity problem are in Figure 4.3. This figure contains both the fully parabolized form, labeled (parab), and the differential-algebraic form, labeled (dae). Along the x-axis is the iteration number, and along the y-axis is the value of the residual and step-size. As you can see the (dae) form requires nearly half the number of iterations as the (parab) form. This alone justifies the use of the differential-algebraic form of the equations and the need for theory to back up the use of these numerical tools. Also note, that the (parab) curves are not monotone, but the (dae) curves are. This is an indication that the (dae) equations were easier to solve and exhibited better algorithm performance than the (parab) equations.

The traditional approach for using Newton's method on this problem is mesh sequencing, where a very coarse initial grid is used (e.g. 4x4), with interpolation providing a path to initial iterates on finer meshes. But even for a coarse 32x32 grid, Newton's method is not able to solve this problem from the cold start initial iterate, whereas Ψtc performed very well at the very fine meshes used for these results.

Figure 4.3: Lid-Driven Buoyancy-Driven Problem – algorithm results as a DAE and as a fully-parabolized PDE.



In this experiment, $V_{lid} = 100$, $Pr = 1$, and $\mathrm{Gr} = 10^5$.

4.3.2 Combustion

As a harder problem for the DAE theory provided above, consider the combustion model from [29] (which will soon become a `PETSc` example problem too). The following equations describe a two-dimensional axisymmetric isobaric laminar diffusion flame. This problem is expressed as a system of three partial differential equations. In the notation of [37], we use r and z for the radial and axial directions in cylindrical coordinates and v_r and v_z as the radial and axial velocities respectively. The three equations are, mass conservation:

$$f_1(\psi,\omega,S) = -\frac{\partial}{\partial z}\left(\frac{1}{r\rho}\frac{\partial \psi}{\partial z}\right) - \frac{\partial}{\partial r}\left(\frac{1}{r\rho}\frac{\partial \psi}{\partial r}\right) - \omega = 0, \tag{4.18}$$

momentum conservation:

$$\begin{aligned} f_2(\psi,\omega,S) \quad = \quad & -\tfrac{\partial}{\partial r}\left(r^3\tfrac{\partial}{\partial r}\left(\tfrac{\mu\omega}{r}\right)\right) - \tfrac{\partial}{\partial z}\left(r^3\tfrac{\partial}{\partial z}\left(\tfrac{\mu\omega}{r}\right)\right) + r^2\left[\tfrac{\partial}{\partial z}\left(\tfrac{\omega}{r}\tfrac{\partial \psi}{\partial r}\right) - \tfrac{\partial}{\partial r}\left(\tfrac{\omega}{r}\tfrac{\partial \psi}{\partial z}\right)\right] \\ & + r^2 g\tfrac{\partial \rho}{\partial r} + r^2\nabla\left(\tfrac{v_r^2+v_z^2}{2}\right)\cdot \operatorname{iso}\rho = 0, \end{aligned} \tag{4.19}$$

and species conservation,

$$f_3(\psi,\omega,S) = -\frac{\partial}{\partial r}\left(r\rho D\frac{\partial S}{\partial r}\right) - \frac{\partial}{\partial z}\left(r\rho D\frac{\partial S}{\partial z}\right) = 0. + \frac{\partial}{\partial z}\left(S\frac{\partial \psi}{\partial r}\right) - \frac{\partial}{\partial r}\left(S\frac{\partial \psi}{\partial z}\right) \tag{4.20}$$

In these equations, ψ is the Stokes stream function, ρ is density, S is species concentration by mass, $\omega = \frac{\partial v_r}{\partial z} - \frac{\partial v_z}{\partial r}$ is vorticity, μ is mixture viscosity, and D is molecular diffusivity. The notation $\operatorname{iso}\rho = \left(-\frac{\partial \rho}{\partial z}, -\frac{\partial \rho}{\partial r}\right)^T$. The Stokes stream function ψ satisfies

the equations:

$$\rho r v_r = -\frac{\partial \psi}{\partial z} \qquad \rho r v_z = \frac{\partial \psi}{\partial r}.$$

The system is then closed by equations of state that express ρ, μ, and D as nonlinear functions of S. Following on the results from the previous section and in contrast with the Euler equations of section 3.4, the most appropriate time-dependent formulation of the system (4.18), (4.19), and (4.20) is a partial differential-algebraic equation. In this problem, the sonic time scale for (4.18) is much faster than the convective scale for which the system evolves. By enforcing (4.18) as an algebraic constraint we remove the fastest time-scale from consideration in the time step-size selection. This allows us to take larger steps. In addition, when we use (4.18) as an algebraic constraint we equilibrate pressure and suppress acoustic stiffness.

To formulate this system into the Ψtc method, let $u = (\psi, \omega, S)^T$ and $F = (f_1, f_2, f_3)^T$. The steady-state equation is then $F(u) = 0$ and the time-dependent equation is

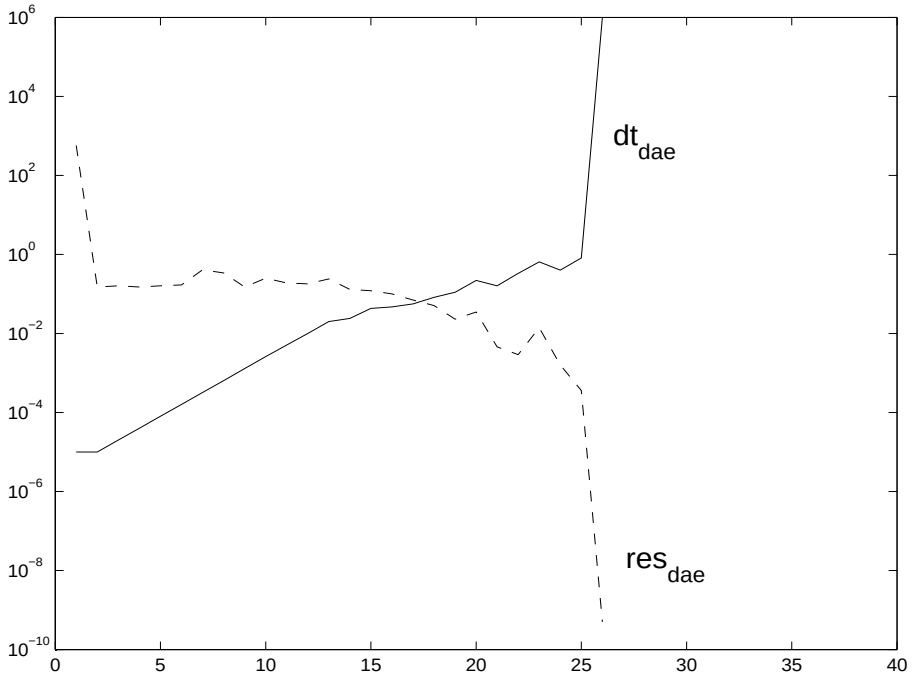
$$Eu' = -F(u) \tag{4.21}$$

where $E = \begin{pmatrix} 0 & & \\ & r^3\rho & \\ & & r\rho \end{pmatrix}$.

Figure 4.4 shows the residual norm and step-size as a function of iteration counter for Ψtc applied to the differential algebraic equation discussed above. The immediate rapid drop in the residual is due to the method obtaining consistent initial conditions for the differential-algebraic equations. Note also that there is quite a bit of non-monotonicity

in both curves. As in the previous example, this is one expression of how difficult this problem is. Attempting to solve this problem with Newton's method is even more difficult than in the previous example. Here, the coarsest physically reasonable mesh is 16x16, and Newton's method is simply not able to solve the cold-start conditions on this mesh. It takes many doublings of the mesh density before the interpolated solution initially lies in the ball of convergence for Newton's method applied directly to the steady-state system.

Figure 4.4: Combustion Problem – algorithmic results.



4.4 Conclusions

The first few sections of this chapter established a convergence theory for Ψtc applied to differential-algebraic equations. Both of the examples in this chapter demonstrate the utility of Ψtc. The first provided compelling evidence for why DAE formulations are important and why Ψtc needed a theory to support its use there. The second example demonstrated how difficult these problems can be for Newton's method.

Note that the two examples presented here appeared to exhibit super-linear convergence when examining the residual plot histories. Unfortunately, both problems have non-smooth properties that violate the assumptions of the theory presented in the early sections of this chapter. The assumptions we used might be relaxed to allow for non-smoothness. This is a topic for future research.

The previous chapters have discussed a variety of problems and the effectiveness of Ψtc. The current chapter has extended the theory for Ψtc from ODEs to DAEs in addition to discussing two new problems for which Ψtc performs very well. As in previous chapters though, more work is needed for non-smooth functions. The next chapter discusses an implicit time-accurate method, and as in all implicit methods, a nonlinear problem must be solved. This nonlinear problem could be solved with Ψtc as in [58], but the time-accurate solution methods discussed in the next chapter don't incorporate Ψtc yet.

Chapter 5

Temporal Integration

5.1 Introduction

In this chapter we discuss an implicit time-accurate solver for ordinary differential equations (or spatially discretized partial differential equations). Implicit methods require nonlinear solves at each iteration and are often quite expensive to use as a result. We investigate a method for reducing this cost which involves solving the expensive nonlinear problem on a coarser mesh to capture the main nonlinear behavior of the solution and reduce the computational work. This coarse mesh solution is then transfered back to the fine mesh and refined with linear solves. This strategy is commonly called a Two-Grid method [64, 65, 61, 62, 17, 18].

Our application is nonlinear radiation diffusion problems in three dimensions, coming from e.g. stellar fusion. These problems are generally very large and require parallel computers to compute accurate solutions. This method has been successfully applied to

similar problems arising in ground water applications [63, 61, 62]. The main theoretical development for this method comes from [65], and the extensions to our nonlinear system come from [18]. The radiation diffusion problem comes from [11].

In the next section we demonstrate how the method is applied within the context of an implicit Euler integration. We then proceed to develop those same ideas for the full problem. In section 5.2 we explain how the two-grid method was implemented in the context of `CVode`. In section 1.4 we explained the radiation diffusion problem we used as our model problem for this method. In section 5.3 we discuss discretization issues, the fine grid linear problem, and preliminary numerical results.

5.1.1 Implicit Euler Example

This method is easily explained when applied to implicit Euler. Let

$$x' = -f(x), \quad x(0) = x_0 \tag{5.1}$$

be our model ODE initial value problem. When we apply implicit Euler to this problem on the fine mesh (h),

$$\frac{x_h^{n+1} - x_h^n}{\delta} = -f(x_h^{n+1}) \tag{5.2}$$

we need to solve a nonlinear equation to determine the next step. If we use Newton's method to solve this nonlinear problem, then we will end up solving many linear problems on the fine mesh, which can be very expensive. Instead we will find a coarse mesh (H)

on which to solve the nonlinear problem and use grid transfer functions I_h^H and I_H^h as restriction and prolongation operators between the grids respectively. With this notation, our two-grid algorithm is:

Algorithm 5.1. *input:* x_h^n, δ *output:* x_h^{n+1}

1. *Restrict:* $x_H^n = I_h^H x_h^n$.

2. *Nonlinear solve:* $G_H(x_H^{n+1}) \equiv x_H^{n+1} + \delta f(x_H^{n+1}) - x_H^n = 0$ *for* x_H^{n+1}.

3. *Prolong:* $z_h^{n+1} = I_H^h x_H^{n+1}$.

4. *Linearize* $G_h(\xi) = \xi + \delta f(\xi) - x_h^n$ *about* z_h^{n+1} *and*
   Linear solve: $x_h^{n+1} = -\widetilde{G_h'}(z_h^{n+1})^{-1} G_h(z_h^{n+1}) + z_h^{n+1}$.

In the last step, we allow for an approximate Jacobian of G_h by using $\widetilde{G_h'}$.

There are three main parts to this algorithm. First, grid-transfers must be devised to move the solution data between the grids. The functions must also be defined on both meshes. Second, a coarse grid nonlinear solve must be developed. This can be computed using a Newton's method (or Ψtc possibly). Third, a fine grid linear solve is computed using the linearization of G_h about the interpolated coarse grid solution. The next section explains our model radiation diffusion problem. The following sections will explain in detail how each of the steps above are completed in the context of `CVode`.

5.2 Two-Grid Implementation

We implemented this method by modifying an existing code for solving this problem. The existing code employs `CVode` (see section 1.5) as an adaptive time-stepping ODE solver (fixed leading BDF coefficient time integrator with variable order & variable step-size and Newton nonlinear solver with scaled preconditioned GMRES linear solver), `transport3d` [11, 10], a radiative transfer in-house development code (also in section 1.5), providing the right hand side function along with `hypre` [26] to help with parallelization and the preconditioner.

We modified `CVode` to handle two grids and added that capability to `transport3d`. With these changes, we then solve the coarse mesh problem using the same machinery as before. The grid transfer routine was written using tri-linear interpolation on the `transport3d` side. Finally, a fine-grid linear solve was written that uses the scaled preconditioned GMRES (`SPGMR`) solver from `CVode` and a new right hand side function from `transport3d`. The changes to both codes abstract the method so that it can be applied to different problems. Eventually this method may migrate to `KINSOL` and `IDA` (a nonlinear system solver and an implicit DAE system solver: http://www.llnl.gov/casc/sundials/). The following three sections explain the algorithmic and coding changes made to `CVode` to implement the two-grid method into `CVode`

5.2.1 Coarse Solve

In `CVode`, the nonlinear problem to be solved at each time step is

$$G(y) = y - \gamma f(t_n, y) - a_n = 0, \tag{5.3}$$

where a_n is the BDF sum of previous iterations and γ is a coefficient formed from the temporal step-size. The user supplies f and this nonlinear problem is solved at each time-step. In the two-grid method, we solve this nonlinear problem on a coarser mesh using the exact same method and machinery described here, which corresponds to step 2 in the Implicit Euler (IE) two-grid algorithm 5.1. Then this coarse grid solution is interpolated back to the fine-grid (step 3 in IE) and refined by linearizing the nonlinear problem (5.3) and solving the resulting linear problem (step 4 in IE).

`CVode` was designed for modular linear solvers so that a particular application could use its own linear solver easily. Unfortunately `CVode` was not written with modular nonlinear solvers in mind. Extensive changes were necessary to get `CVode` to interoperate with a different nonlinear solver than the ones provided internally. In particular `CVode` uses a single data structure (`cv_mem`) to hold all the data necessary for every routine. This data structure is passed to every function and subroutine. For the linear solver there is a chunk of memory set aside specifically for linear solver data, but the nonlinear solver data is interspersed with all the other data in the main data structure. One of the main challenges was to separate out the data necessary for the nonlinear solver and

to put it all in a single chunk of memory that can be passed to all the nonlinear solver functions.

In this work, we made every effort to preserve the current functioning state of the code and simply add a new nonlinear solver. This effort has isolated the input and output data used between `CVode` and the nonlinear solver. Building on this information, `CVode` could be rewritten to handle a user-supplied nonlinear solver.

5.2.2 Fine Solve

Now that we have step 2 of the Implicit Euler example completed for the two-grid method, we will move on to step 4. Results in [18] indicate that we can achieve an error on the order of the discretization error by the application of one linear solve on the fine grid. We could still apply more corrector iterations if necessary.

The coarse grid solver returns y_H, which we then prolong to the fine mesh and use linear solves to clean up. Recall the nonlinear function which `CVode` solves at each step (5.3):

$$G(y) = y - \gamma f(t_n, y) - a_n = 0.$$

To form the linear problem on the fine mesh, we linearize G about $I_H^h y_H$.

$$G(y_h) = G(I_H^h y_H) + \widetilde{J}_G(I_H^h y_H)(y_h - I_H^h y_H) = 0 \tag{5.4}$$

where $\widetilde{J}_G$ is an approximate Jacobian of G. Our fine grid linear system is then,

$$\widetilde{J}_G(I_H^h y_H)(y_h - I_H^h y_H) = -G(I_H^h y_H). \tag{5.5}$$

We can formulate this problem in terms of the step $S = y_h - I_H^h y_H$. The approximate Jacobian of G comes from approximating the Jacobian of f in (5.3):

$$\widetilde{J}_G(I_H^h y_H)S \equiv S - \gamma \widetilde{J}_f(t_n, I_H^h y_H)S.$$

This reduces the user input to f and $\widetilde{J}_f S$ with the system,

$$\widetilde{J}_G(I_H^h y_H)S = -G(I_H^h y_H) \tag{5.6}$$

solved by the built in scaled preconditioned GMRES (`SPGMR`) solver in `CVode`. This part of the coding was fairly straight forward, once this "step" formulation of the problem came about. The linear solver is very modular and required very little setup to run on this new problem. Since this is an iterative solver, we need a preconditioner. Currently the same preconditioner for the coarse solve is used on the fine solve, but this will soon change to allow a different preconditioner for each problem.

5.2.3 Grid Transfers

All of the grid transfers must be computed by user supplied routines. Since `CVode` is a general purpose ODE solver which is often used for very complicated PDE problems, we cannot anticipate how the grid transfers should be computed. The model problem in section 1.4 uses structured 3-D grids, so we implemented a generic 3-D structured-grid linear grid transfer routine as an example. This grid transfer routine assumes that the two grids lie on the same rectangular domain. The user then supplies the x, y, and z coordinates of the vertices where the data is coming from and going to. The grid transfer routine then linearly interpolates the data from the input coordinates to the output coordinates. The only caveat is that the output coordinates must lie inside the convex hull of the input coordinates. The next two sections discuss the 2-D and 3-D linear interpolation and extrapolation we used.

Bi-Linear Interpolation

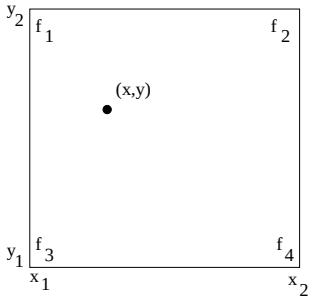


Figure 5.1: Basic 2-D cell: Bi-linear interpolation can be used to find a function value at (x, y) given function values at the corners.

Bi-linear interpolation consists of using Lagrange interpolating polynomials to approximate the function with linear polynomials in each of the coordinate directions. Figure 5.1 shows a square with numbered points $1, 2, 3, 4$, and (x, y). To approximate the point (x, y), we use the bi-linear interpolating function:

$$\begin{aligned} f(x,y) \quad = \quad & \left[\left(\tfrac{x-x_1}{x_2-x_1}\right) f_2 + \left(\tfrac{x_2-x}{x_2-x_1}\right) f_1\right] \left(\tfrac{y_2-y}{y_2-y_1}\right) + \\ & \left[\left(\tfrac{x-x_1}{x_2-x_1}\right) f_4 + \left(\tfrac{x_2-x}{x_2-x_1}\right) f_3\right] \left(\tfrac{y-y_1}{y_2-y_1}\right) \end{aligned}$$

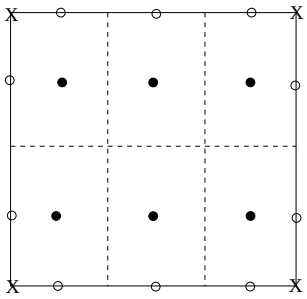


Figure 5.2: 2-D Domain cell-centered layout: This shows three cells in the x direction and two cells in the y direction. The solid dots denote solution data, the empty dots denote computable boundary conditions, and the "X" points are unknown.

Figure 5.2 shows what our domain looks like for the 2-D problem. Since we are given the cell centered data and the boundary conditions, we need only compute the corners now. Taylor series expansions allow us to write these corner points in terms of data we already have. Consider the lower left corner:

$$f_{i+1,j} = f_{i,j} + \left.\frac{\partial f}{\partial x}\right|_{i,j} \frac{H^x}{2} + O((H^x)^2)$$

$$f_{i,j+1} = f_{i,j} + \left.\frac{\partial f}{\partial y}\right|_{i,j} \frac{H^y}{2} + O((H^y)^2)$$

$$f_{i+1,j+1} = f_{i,j} + \left.\frac{\partial f}{\partial x}\right|_{i,j} \frac{H^x}{2} \left.\frac{\partial f}{\partial y}\right|_{i,j} \frac{H^y}{2} + O((H^x)^2 + (H^y)^2)$$

And we can take a linear combination of these to get the following approximation to the upper left corner:

$$f_{i,j} = f_{i+1,j} + f_{i,j+1} - f_{i+1,j+1} + O((H^x)^2 + (H^y)^2)$$

We can do this for the other three corners and we have an extrapolation which is 2nd order accurate in space, and allows us to bi-linearly interpolate any point on the interior of our domain.

Tri-Linear Interpolation

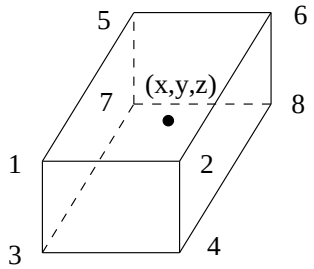


Figure 5.3: 3-D cell with eight corners set up for tri-linear interpolation.

Now consider Figure 5.3 which shows the points $1-8$, and (x, y, z). We can tri-linearly interpolate the point (x, y, z) by use of the following function:

$$
\begin{aligned}
f(x,y,z) \quad = \quad & \left\{ \left[\left(\tfrac{x-x_1}{x_2-x_1} \right) f_2 + \left(\tfrac{x_2-x}{x_2-x_1} \right) f_1 \right] \left(\tfrac{y_2-y}{y_2-y_1} \right) + \right. \\
& \left. \left[\left(\tfrac{x-x_1}{x_2-x_1} \right) f_4 + \left(\tfrac{x_2-x}{x_2-x_1} \right) f_3 \right] \left(\tfrac{y-y_1}{y_2-y_1} \right) \right\} \left(\tfrac{z_2-z}{z_2-z_1} \right) + \\
& \left\{ \left[\left(\tfrac{x-x_1}{x_2-x_1} \right) f_6 + \left(\tfrac{x_2-x}{x_2-x_1} \right) f_5 \right] \left(\tfrac{y_2-y}{y_2-y_1} \right) + \right. \\
& \left. \left[\left(\tfrac{x-x_1}{x_2-x_1} \right) f_8 + \left(\tfrac{x_2-x}{x_2-x_1} \right) f_7 \right] \left(\tfrac{y-y_1}{y_2-y_1} \right) \right\} \left(\tfrac{z-z_1}{z_2-z_1} \right)
\end{aligned}
$$

We have the same boundary data as before, but now we are missing much more data because we need edge data in addition to the corner data.

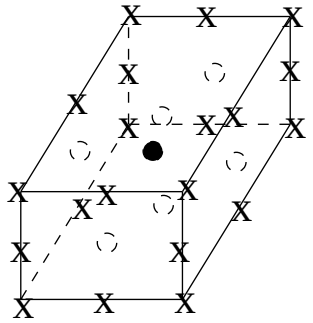


Figure 5.4: A single 3-D cell for the entire domain with the solid dot denoting solution data, the empty dots denoting boundary conditions, and the "X" points denoting unknowns.

Figure 5.4 shows a 1-cell domain in 3-d with the interior and boundary data displayed with solid dots, and the unknown edges and corners with "X" labels. The same types of Taylor expansions as before give us the extrapolation formulas we need. Any number of extrapolation formulas could be developed, but we preferred here to only use data from the interior or the boundary conditions. This allows us to compute the edges and corners in any order we desire with no dependence on each other.

There are 12 edges in total that must be extrapolated. We can divide all 12 edges into three groups. The first group contains those edges for which x varies and y, z are held constant. Similarly, the second set contains the y-varying edges, and the third set contains the z-varying edges. The following formula shows how to extrapolate the x-varying edges when j and k are both 1. A similar formula is used for the other x-varying edges. Five points contribute to each edge extrapolation: the face values for the four faces which touch the edge, and the center node value. This leads us to the following x-varying edge extrapolation formula:

$$f_{i,j,k} = f_{i,j+1,k+1} + f_{i,j+1,k} + f_{i,j,k+1} - f_{i+1,j+1,k+1} - f_{i-1,j+1,k+1} + O((H^x)^2 + (H^y)^2 + (H^z)^2)$$

The y- and z-varying edges can be extrapolated similarly by using the heuristic above. I.e. use the face data for faces which touch the edge, and the center node value.

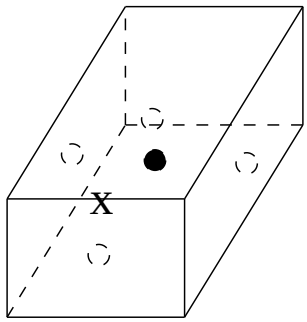


Figure 5.5: X-varying edge extrapolation. There are four boundary conditions and one cell centered data point that contribute to the edge computation. The face data comes from the top, both sides, and the front.

The corner points are extrapolated using only four points: all three faces which touch the corner, and the center node value. Edge data could have been used here, but we opted to stick with primary data from the cell centers and the boundaries. The corner listed here is the $(1, 1, 1)$ corner. The other corner formulas are similarly computed. This leads us to the following $(1, 1, 1)$ corner extrapolation formula:

$$f_{i,j,k} = f_{i+1,j+1,k} + f_{i,j+1,k+1} + f_{i+1,j,k+1} - 2f_{i+1,j+1,k+1} + O((H^x)^2 + (H^y)^2 + (H^z)^2)$$

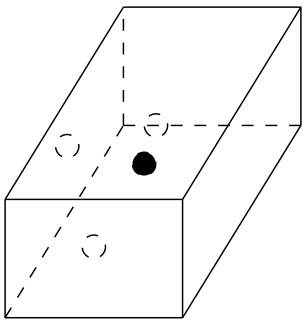


Figure 5.6: Corner extrapolation. There are three boundary conditions and one cell centered data point that contribute to the corner computation. The face data comes from the top, the side, and the front.

To help explain which points these are, Figure 5.5 shows which points contribute to an x-varying edge; Figure 5.6 shows which points contribute to the $(1, 1, 1)$ corner.

Note that the above analysis has been for the one-cell case, and when there are multiple cells, the face values will not come from primary data but will have to be

averaged from the two adjacent cells.

5.3 Model Problem

Recall the radiative transfer problem from section 1.4. The following sections develop the discretization used and numerical results from applying the two-grid method to this problem.

5.3.1 Discretization

In this section we formulate an expanded mixed method finite difference discretization for the radiation diffusion problem (1.9) and (1.11). So, we will first introduce the notation for the finite element spaces and the discrete inner products we used. For more details, please see [18, 8].

Let Ω be a rectangular domain in $\mathbb{R}^d$ ($d = 1, 2,$ or 3) with boundary Γ. Let $V = H(\Omega, div) = \{\mathbf{v} \in (L^2(\Omega))^d : \nabla \cdot \mathbf{v} \in L^2(\Omega)\}$ and $W = L^2(\Omega)$. We will consider two quasi-uniform triangulations of Ω, a coarse triangulation with mesh size H denoted by $\mathcal{T}_H$, and a refinement of this triangulation with mesh size h denoted by $\mathcal{T}_h$. Both of these triangulations consist of rectangles in two dimensions or bricks in three dimensions. We consider the lowest-order Raviart-Thomas-Nédélec (RTN) space on rectangles [53, 47].

Thus, on an element $T \in \mathcal{T}_k$, $\quad k = h$ or H, we have

$$\begin{aligned} V_k(T) &= \{(\alpha_1 x_1 + \beta_1, \alpha_2 x_2 + \beta_2, \alpha_3 x_3 + \beta_3)^T : \alpha_i, \beta_i \in \mathbb{R}\}, \\ W_k(T) &= \{\alpha : \alpha \in \mathbb{R}\} \end{aligned}$$

where the last component in V_k should be deleted in two dimensions. We use the standard nodal basis, where for V_k the nodes are at the midpoints of the edges or faces of the elements, and for W_k the nodes are at the centers of the elements. We denote the grid points of the fine grid by

$$(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}), \quad i = 0, \cdots, N_x, \quad j = 0, \cdots, N_y$$

and define

$$\begin{aligned} x_i &= \tfrac{1}{2}\left(x_{i+\frac{1}{2}} + x_{i-\frac{1}{2}}\right), && i = 1, \cdots, N_x \\ y_j &= \tfrac{1}{2}\left(y_{j+\frac{1}{2}} + y_{j-\frac{1}{2}}\right), && j = 1, \cdots, N_y \\ h^x_{i+\frac{1}{2}} &= x_{i+1} - x_i, && i = 1, \cdots, N_x - 1 \\ h^y_{j+\frac{1}{2}} &= y_{j+1} - y_j, && j = 1, \cdots, N_y - 1 \\ h^x_i &= x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}, && i = 1, \cdots, N_x \\ h^y_i &= y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}}, && j = 1, \cdots, N_y \\ h &= \max_{i,j}\left(h^x_i, h^y_j\right) \end{aligned}$$

with corresponding notation for a third dimension. For the coarse grid, similar quantities are defined, but the number of points in each direction are denoted $\hat{N}_x$ and $\hat{N}_y$. We define discrete inner products corresponding to applications of the midpoint (M), trapezoidal

(T), and midpoint by trapezoidal (TM) quadrature rules by

$$(r,s)_{\mathrm{M}} = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} h_i^x h_j^y r_{i,j} s_{i,j}$$

$$(\mathbf{v},\mathbf{q})_{\mathrm{TM}} = \sum_{i=0}^{N_x}\sum_{j=1}^{N_y} h^x h^y \mathbf{v}^x_{i+\frac{1}{2},j}\mathbf{q}^x_{i+\frac{1}{2},j} + \sum_{i=1}^{N_x}\sum_{j=0}^{N_y} h^x h^y \mathbf{v}^y_{i,j+\frac{1}{2}}\mathbf{q}^y_{i,j+\frac{1}{2}}$$

$$\begin{aligned}(\mathbf{v},\mathbf{q})_{\mathrm{T}} = &\sum_{i=0}^{N_x}\sum_{j=1}^{N_y} h^x h^y \frac{1}{2}\left(\mathbf{v}^x_{i+\frac{1}{2},j-\frac{1}{2}}\mathbf{q}^x_{i+\frac{1}{2},j-\frac{1}{2}} + \mathbf{v}^x_{i+\frac{1}{2},j+\frac{1}{2}}\mathbf{q}^x_{i+\frac{1}{2},j+\frac{1}{2}}\right) + \\ &\sum_{i=1}^{N_x}\sum_{j=0}^{N_y} h^x h^y \frac{1}{2}\left(\mathbf{v}^y_{i-\frac{1}{2},j+\frac{1}{2}}\mathbf{q}^y_{i-\frac{1}{2},j+\frac{1}{2}} + \mathbf{v}^y_{i+\frac{1}{2},j+\frac{1}{2}}\mathbf{q}^y_{i+\frac{1}{2},j+\frac{1}{2}}\right)\end{aligned}$$

where we add a third sum in each for the case of three dimensions. We denote the associated norms by $\|\cdot\|_R$, where $R = M, T$, or TM.

Now that the notation is established we will define our expanded mixed method and the variational version of (1.9) and (1.11). In the following analysis, we will focus on the first equation (1.9) because the second equation is simply a subset of the first. Let $\widetilde{U}$ and U be,

$$\begin{aligned}\widetilde{\mathbf{U}} &= -\nabla E_R \quad \text{and} \\ \mathbf{U} &= K(E_R)\widetilde{\mathbf{U}}.\end{aligned} \tag{5.7}$$

We define the mixed variational version of (1.9) by,

$$(d_t E_R, w)_{\mathrm{M}} = -(\nabla \cdot \mathbf{U}, w)_{\mathrm{M}} + (C(E), w)_{\mathrm{M}} + \left(\chi c a T^4_{source}, w\right)_{\mathrm{M}}, \quad \forall w \in W_h,$$

(5.8)

$$\left(\widetilde{\mathbf{U}}(E_R), \mathbf{v}\right)_{\mathrm{TM}} = (E_R, \nabla \cdot \mathbf{v})_{\mathrm{M}}, \qquad \forall \mathbf{v} \in V_h^0 \tag{5.9}$$

$$(\mathbf{U}, \mathbf{v})_{\mathrm{TM}} = \left(K(E_R)\widetilde{\mathbf{U}}(E_R), \mathbf{v}\right)_{\mathrm{T}}, \qquad \forall \mathbf{v} \in V_h. \tag{5.10}$$

Where $E = (E_R, E_M)^T$, and C is defined by,

$$C(E) = c\rho\kappa_P(T_M)\left(aT_M^4 - E_R\right). \tag{5.11}$$

The rest of this section expands out the full discretization in two dimensions. Three dimensions follows easily.

2-D Formulation

Using the quadratures above we can discretize equation (5.8). We pick a very specific $w \in W_h$ that has value one on cell (i, j), and zero elsewhere. Figure 5.7 shows the support for the four hat functions in $\mathbf{U}$ that touch cell (i, j).

$$(d_t E_R, w)_{\mathrm{M}} = h^x h^y d_t E_{Rij}$$

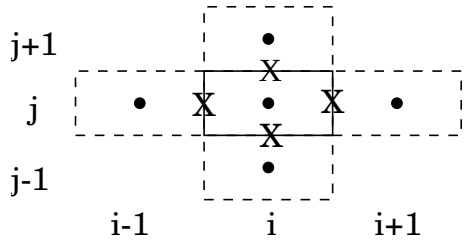


Figure 5.7: Cell (i, j): The dots denote where E, w are defined and the X's denote where $\mathbf{U}, \widetilde{\mathbf{U}}, \mathbf{v}$ are defined. The dashed boxes indicate that in the following integration, w is zero outside cell (i, j)

We integrate over all four hat functions in $\mathbf{U}$ for the second term in equation (5.8):

$$\begin{aligned}(\nabla \cdot \mathbf{U}, w)_{\mathrm{M}} \quad &= \quad h^x h^y U^x_{i+\frac{1}{2},j}\left(\tfrac{1}{h^x}\right) + h^x h^y U^x_{i-\frac{1}{2},j}\left(\tfrac{-1}{h^x}\right) + \\ &\qquad h^x h^y U^y_{i,j+\frac{1}{2}}\left(\tfrac{1}{h^y}\right) + h^x h^y U^y_{i,j-\frac{1}{2}}\left(\tfrac{-1}{h^y}\right)\end{aligned}$$

Then C is simply rewritten in discrete form,

$$(C(E), w)_{\mathrm{M}} = h^x h^y C(E)_{ij}.$$

And the last term is the same way,

$$\left(\chi c a T^4_{source}, w\right)_{\mathrm{M}} = h^x h^y \chi(x_{i,j}) c a T^4_{source}.$$

Therefore, the discretized version of equation (5.8) is,

$$d_t E_{Rij} = -\left(\frac{U^x_{i+\frac{1}{2},j} - U^x_{i-\frac{1}{2},j}}{h^x}\right) - \left(\frac{U^y_{i,j+\frac{1}{2}} - U^y_{i,j-\frac{1}{2}}}{h^y}\right) + C(E)_{i,j} + \chi(x_{i,j}) c a T^4_{source}. \quad (5.12)$$

For equation (5.9-5.10) we pick $\mathbf{v}^{(1)} \in V_h$ as the hat function in the x direction with height one (zero in the y direction) and support in cells (i, j) and $(i + 1, j)$. Figure 5.8 shows the support for $\mathbf{v}^{(1)}$. Then we will pick a different $\mathbf{v}^{(2)} \in V_h$ that is a hat function in the y direction with height one (zero in the x direction) and support in cells (i, j) and $(i, j + 1)$. Figure 5.9 shows the support for $\mathbf{v}^{(2)}$.

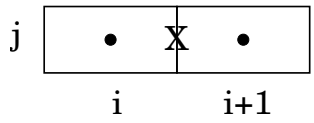


Figure 5.8: Support for $\mathbf{v}^{(1)}$: cells (i, j) and $(i + 1, j)$. The black dot denotes storage for E, and the X's denotes the storage location for $\mathbf{U}, \widetilde{\mathbf{U}}$, and $\mathbf{v}$.

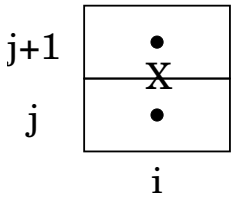


Figure 5.9: Support for $\mathbf{v}^{(2)}$: cells (i, j) and $(i, j + 1)$.

$$\begin{aligned}\left(\widetilde{\mathbf{U}}(E_R),\mathbf{v}^{(1)}\right)_{\mathrm{TM}} &= h^x h^y\left(0+\frac{\widetilde{\mathbf{U}}^x_{i+\frac{1}{2},j}}{2}\right)+h^x h^y\left(\frac{\widetilde{\mathbf{U}}^x_{i+\frac{1}{2},j}}{2}+0\right)\\ &= h^x h^y\widetilde{\mathbf{U}}^x(E_R)_{i+\frac{1}{2},j}\end{aligned}$$

$$\left(\widetilde{\mathbf{U}}(E_R),\mathbf{v}^{(2)}\right)_{\mathrm{TM}} = h^x h^y\widetilde{\mathbf{U}}^y(E_R)_{i,j+\frac{1}{2}}$$

$$\begin{aligned}\left(E_R,\nabla\cdot\mathbf{v}^{(1)}\right)_{\mathrm{M}} &= h^x h^y E_{R,i,j}\left(\tfrac{1}{h^x}\right)+h^x h^y E_{R,i+1,j}\left(\tfrac{-1}{h^x}\right)\\ &= -h^y\left(E_{R,i+1,j}-E_{R,i,j}\right)\end{aligned}$$

$$\left(E_R,\nabla\cdot\mathbf{v}^{(2)}\right)_{\mathrm{M}} = -h^x\left(E_{R,i,j+1}-E_{R,i,j}\right)$$

Therefore, equation (5.9) gives the relations:

$$\begin{aligned}\widetilde{\mathbf{U}}^x(E_R)_{i+\frac{1}{2},j} &= -\left(\tfrac{E_{R,i+1,j}-E_{R,i,j}}{h^x}\right)\\ \widetilde{\mathbf{U}}^y(E_R)_{i,j+\frac{1}{2}} &= -\left(\tfrac{E_{R,i,j+1}-E_{R,i,j}}{h^y}\right)\end{aligned}\qquad(5.13)$$

Now consider equation (5.10).

$$\begin{aligned}\left(\mathbf{U},\mathbf{v}^{(1)}\right)_{\mathrm{TM}} &= h^x h^y\left(0+\frac{\mathbf{U}^x_{i+\frac{1}{2},j}}{2}\right)+h^x h^y\left(\frac{\mathbf{U}^x_{i+\frac{1}{2},j}}{2}+0\right)\\ &= h^x h^y\mathbf{U}^x_{i+\frac{1}{2},j}\end{aligned}$$

$$\left(\mathbf{U},\mathbf{v}^{(2)}\right)_{\mathrm{TM}} = h^x h^y\mathbf{U}^y_{i,j+\frac{1}{2}}$$

$$\begin{aligned}\left(K(E_R)\widetilde{\mathbf{U}}(E_R), v^{(1)}\right)_T &= \tfrac{1}{4}h^x h^y\left(\left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j+\frac{1}{2}} + \left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j-\frac{1}{2}}\right)\\ &\quad +\tfrac{1}{4}h^x h^y\left(\left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j+\frac{1}{2}} + \left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j-\frac{1}{2}}\right)\\ &= \tfrac{1}{2}h^x h^y\left(\left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j+\frac{1}{2}} + \left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j-\frac{1}{2}}\right)\end{aligned}$$

$$\left(K(E_R)\widetilde{\mathbf{U}}(E_R), v^{(2)}\right)_T = \tfrac{1}{2}h^x h^y\left(\left[K(E_R)\widetilde{\mathbf{U}}^y(E_R)\right]_{i+\frac{1}{2},j+\frac{1}{2}} + \left[K(E_R)\widetilde{\mathbf{U}}^y(E_R)\right]_{i-\frac{1}{2},j+\frac{1}{2}}\right)$$

Equation (5.10) gives the relations:

$$\begin{aligned}\mathbf{U}^x_{i+\frac{1}{2},j} &= \tfrac{1}{2}\left(\left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j+\frac{1}{2}} + \left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j-\frac{1}{2}}\right)\\ \mathbf{U}^y_{i,j+\frac{1}{2}} &= \tfrac{1}{2}\left(\left[K(E_R)\widetilde{\mathbf{U}}^y(E_R)\right]_{i+\frac{1}{2},j+\frac{1}{2}} + \left[K(E_R)\widetilde{\mathbf{U}}^y(E_R)\right]_{i-\frac{1}{2},j+\frac{1}{2}}\right)\end{aligned} \tag{5.14}$$

Equations (5.12-5.14) define the 2-D not linearized fine-grid discretization for the expanded mixed method. Putting this all together we have the following discretized

problem:

$$\begin{aligned}
d_t E_{Rij} &= -\left(\frac{U^x_{i+\frac{1}{2},j} - U^x_{i-\frac{1}{2},j}}{h^x}\right) - \left(\frac{U^y_{i,j+\frac{1}{2}} - U^y_{i,j-\frac{1}{2}}}{h^y}\right) \\
&\quad + C(E)_{i,j} + \chi(x_{i,j}) c a T^4_{source} \\
d_t E_{Mij} &= -C(E)_{i,j} \\
\mathbf{U}^x_{i+\frac{1}{2},j} &= \frac{1}{2}\left(\left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j+\frac{1}{2}} + \left[K(E_R)\widetilde{\mathbf{U}}^x(E_R)\right]_{i+\frac{1}{2},j-\frac{1}{2}}\right) \\
\mathbf{U}^y_{i,j+\frac{1}{2}} &= \frac{1}{2}\left(\left[K(E_R)\widetilde{\mathbf{U}}^y(E_R)\right]_{i+\frac{1}{2},j+\frac{1}{2}} + \left[K(E_R)\widetilde{\mathbf{U}}^y(E_R)\right]_{i-\frac{1}{2},j+\frac{1}{2}}\right) \\
\widetilde{\mathbf{U}}^x(E_R)_{i+\frac{1}{2},j} &= -\left(\frac{E_{R,i+1,j} - E_{R,i,j}}{h^x}\right) \\
\widetilde{\mathbf{U}}^y(E_R)_{i,j+\frac{1}{2}} &= -\left(\frac{E_{R,i,j+1} - E_{R,i,j}}{h^y}\right) \\
C(E) &= c\rho\kappa_P(T_M)\left(aT^4_M - E_R\right).
\end{aligned} \tag{5.15}$$

Once extended to 3D, we can define $\mathcal{D}$ as,

$$\begin{aligned}
(\mathcal{D}_x A)_{i,j,k} &= \frac{1}{h_x}\left[\frac{1}{4}\left(A_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} + A_{i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}} + A_{i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}} + A_{i+\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}\right)\right. \\
&\quad \left. -\frac{1}{4}\left(A_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} + A_{i-\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}} + A_{i-\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}} + A_{i-\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}\right)\right] \\
(\mathcal{D}_y A)_{i,j,k} &= \frac{1}{h_y}\left[\frac{1}{4}\left(A_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} + A_{i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}} + A_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} + A_{i-\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}}\right)\right. \\
&\quad \left. -\frac{1}{4}\left(A_{i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}} + A_{i+\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}} + A_{i-\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}} + A_{i-\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}\right)\right] \\
(\mathcal{D}_z A)_{i,j,k} &= \frac{1}{h_z}\left[\frac{1}{4}\left(A_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} + A_{i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}} + A_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} + A_{i-\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}}\right)\right. \\
&\quad \left. -\frac{1}{4}\left(A_{i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}} + A_{i+\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}} + A_{i-\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}} + A_{i-\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}\right)\right]
\end{aligned} \tag{5.16}$$

and rewrite these equations as an f for `CVode`:

$$\begin{aligned} f(t,E)_{i,j,k} = \Big[& -\mathcal{D}_x(K(E_R)\widetilde{U}^x(E_R))_{i,j,k} - \mathcal{D}_y(K(E_R)\widetilde{U}^y(E_R))_{i,j,k} \\ & - \mathcal{D}_z(K(E_R)\widetilde{U}^z(E_R))_{i,j,k} + c\rho\kappa_P(T_M)_{i,j,k}\left(a(T_M)^4_{i,j,k} - E_{R,i,j,k}\right) \\ & + \chi(x_{i,j,k})caT^4_{\text{source}}, -c\rho\kappa_P(T_M)_{i,j,k}\left(a(T_M)^4_{i,j,k} - E_{R,i,j,k}\right)\Big]^T \end{aligned} \quad (5.17)$$

The next section linearizes this system of equations to define the fine-grid linear problem.

5.3.2 Fine-Grid Linear Problem

This section discusses exactly how f and $\widetilde{J}_f S$ are computed for the radiative transfer problem in `transport3d`. Equations (5.17) are linear except for the $K(E_R)\widetilde{U}$ terms and the $C(E)$ terms. $K(E_R)$ is nonlinear by itself, but it is also multiplied by a linear function of E_R. We now need to distinguish which grid we are on. We will denote E_h and E_H as the energy on the fine and coarse grids respectively ($E = (E_R, E_M)^T$). The fine-grid linear system's right hand side, f is then,

$$\begin{aligned} f(t, I^h_H E_H)_{i,j,k} = \Big[& -\mathcal{D}_x(K(I^h_H E_{R,H})\widetilde{U}^x(I^h_H E_{R,H}))_{i,j,k} \\ & - \mathcal{D}_y(K(I^h_H E_{R,H})\widetilde{U}^y(I^h_H E_{R,H}))_{i,j,k} - \mathcal{D}_z(K(I^h_H E_{R,H})\widetilde{U}^z(I^h_H E_{R,H}))_{i,j,k} \\ & + c\rho\kappa_P(I^h_H T_{M,H})_{i,j,k}\left(a(I^h_H T_{M,H})^4_{i,j,k} - I^h_H E_{R,H,i,j,k}\right) \\ & + \chi(x_{i,j,k})caT^4_{\text{source}}, \\ & -c\rho\kappa_P(I^h_H T_{M,H})_{i,j,k}\left(a(I^h_H T_{M,H})^4_{i,j,k} - I^h_H E_{R,H,i,j,k}\right)\Big]^T. \end{aligned} \quad (5.18)$$

The $K\widetilde{U}$ product is linearized as follows:

$$\begin{aligned} K(E_{R,h})\widetilde{U}(E_{R,h}) \quad &\approx \quad K(I_H^h E_{R,H})\widetilde{U}(E_{R,h}) \\ &\quad +\tfrac{\partial K}{\partial E_R}(I_H^h E_{R,H}) I_H^h \widetilde{U}(E_{R,H}) \left(E_{R,h} - I_H^h E_{R,H}\right). \end{aligned} \tag{5.19}$$

The material-energy transfer relation $C(E)$ is linearized:

$$\begin{aligned} C(E_h) \quad &\approx \quad C(I_H^h E_H) + \tfrac{\partial C}{\partial E}(I_H^h E_H)(E_h - I_H^h E_H) \\ &= \quad c\rho\kappa_P(I_H^h T_{M,H}) \left(a(I_H^h T_{M,H})^4 - I_H^h E_{R,H}\right) \\ &\quad -c\rho\kappa_P(I_H^h T_{M,H}) \left(E_{R,h} - I_H^h E_{R,H}\right) \\ &\quad +c\rho \left[\tfrac{\partial \kappa_P}{\partial T_M}(I_H^h T_{M,H}) \tfrac{\partial T_M}{\partial E_M}(I_H^h E_{M,H}) \left(a(I_H^h T_{M,H})^4 - I_H^h E_{R,H}\right)\right. \\ &\quad \left. +\kappa_P(I_H^h T_{M,H}) \left(4a(I_H^h T_{M,H})^3 \tfrac{\partial T_M}{\partial E_M}(I_H^h E_{M,H})\right)\right] \left(E_{M,h} - I_H^h E_{M,H}\right). \end{aligned} \tag{5.20}$$

If we substitute (5.19) and (5.20) into (5.18) then we get our fine-grid linearized problem. The Jacobian of this system is seen in equation (5.21). The extension to three dimensions is a straightforward application of the previous analysis and does not provide any new insight.

The discrete Jacobian of the linearized problem, $\widetilde{J}_f S$ is written,

$$
\begin{aligned}
(\widetilde{J}_f S)_{i,j,k} = \Bigg[& -\mathcal{D}_x \left(K(I_H^h E_{R,H}) \widetilde{U}^x(S_R) + \frac{\partial K}{\partial E_R}(I_H^h E_{R,H}) I_H^h \widetilde{U}^x(E_{R,H}) S_R \right)_{i,j,k} \\
& - \mathcal{D}_y \left(K(I_H^h E_{R,H}) \widetilde{U}^y(S_R) + \frac{\partial K}{\partial E_R}(I_H^h E_{R,H}) I_H^h \widetilde{U}^y(E_{R,H}) S_R \right)_{i,j,k} \\
& - \mathcal{D}_z \left(K(I_H^h E_{R,H}) \widetilde{U}^z(S_R) + \frac{\partial K}{\partial E_R}(I_H^h E_{R,H}) I_H^h \widetilde{U}^z(E_{R,H}) S_R \right)_{i,j,k} \\
& - c\rho\kappa_P (I_H^h T_{M,H})_{i,j,k} S_{R,i,j,k} \\
& + c\rho \Bigg[\frac{\partial \kappa_P}{\partial T_M}(I_H^h T_{M,H})_{i,j,k} \frac{\partial T_M}{\partial E_M}(I_H^h E_{M,H})_{i,j,k} \left(a(I_H^h T_{M,H})^4 - I_H^h E_{R,H} \right)_{i,j,k} \\
& + \kappa_P (I_H^h T_{M,H})_{i,j,k} \left(4a(I_H^h T_{M,H})^3 \frac{\partial T_M}{\partial E_M}(I_H^h E_{M,H}) \right)_{i,j,k} \Bigg] S_{M,i,j,k}, \\
& c\rho\kappa_P (I_H^h T_{M,H})_{i,j,k} S_{R,i,j,k} \\
& - c\rho \Bigg[\frac{\partial \kappa_P}{\partial T_M}(I_H^h T_{M,H})_{i,j,k} \frac{\partial T_M}{\partial E_M}(I_H^h E_{M,H})_{i,j,k} \left(a(I_H^h T_{M,H})^4 - I_H^h E_{R,H} \right)_{i,j,k} \\
& + \kappa_P (I_H^h T_{M,H})_{i,j,k} \left(4a(I_H^h T_{M,H})^3 \frac{\partial T_M}{\partial E_M}(I_H^h E_{M,H}) \right)_{i,j,k} \Bigg] S_{M,i,j,k} \Bigg]^T, \quad (5.21)
\end{aligned}
$$

where,

$$
S = \begin{pmatrix} S_R \\ S_M \end{pmatrix} = \begin{pmatrix} E_{R,h} - I_H^h E_{R,H} \\ E_{M,h} - I_H^h E_{M,H} \end{pmatrix}.
$$

Up until this point, it may not be clear why we are using tildes to represent approximations of the Jacobians. Why not just compute the Jacobian directly? The theory presented in [18] requires that we linearize our original problem and compute the Jacobian of that system for use in the fine-grid linear problem. The Jacobian of the not linearized system evaluated at $I_H^h y_H$ is not the same as the Jacobian of the linearized

system.

5.3.3 Grid Transfers

The two-grid method depends heavily on grid transfers between the coarse grid and the fine grid. For this model problem with structured grids, we used the generic 3-D grid transfer routine discussed in section 5.2.3 for both the prolongation and restriction operators. In this application there are several different data structures used and we need to be able to move data between many of them. These include: cell centered data, cell interface data, cell corner data, and ghosted varieties of each. The generic 3-D grid transfer routine was able to perform all of these transfers, we just needed to supply the right coordinate information.

5.3.4 Choice of H

There are a variety of convergence estimates and numerical results for how to choose H for different types of problems. For example, Dawson and Wheeler [17] analyzed a lowest order mixed finite element method for a nonlinear diffusion problem and showed a convergence rate of order $O(h + H^{3-d/2})$, where d is the dimension of the problem. Wu and Allen [61, 62] also considered a lowest order mixed finite element method, but for a semi-linear problem and showed numerically an order of convergence of $O(h + H^2)$. Our problem, as discretized above, is a cell-centered finite difference method for a nonlinear diffusion problem. This case has been analyzed by Dawson, Wheeler, and Woodward

in [18] where the order of convergence of the two-grid method is $O(h^2 + H^{4-d/2})$. That means that for a 2-D problem $H = h^{2/3}$ and for a 3-D problem $H = h^{4/5}$ will retain optimal $O(h^2)$ convergence. As an example if $h = 1/64,000$ then $H = 1/1,600$ in 2-D and $H = 1/7,000$ in 3-D would be acceptable. These are significantly coarser meshes and should reduce the computational work of the coarse mesh problem considerably.

5.3.5 Numerical Results

The implementation details for the two-grid method were substantially more complicated than originally anticipated. The inter-grid transfers between grids with no special relation was especially time-consuming. In addition, the numerous same-grid transfers where cell centered data was moved to the the corners slowed down development time. At this point, the code is in a functional state, but has not been optimized. Therefore, we will not demonstrate comparisons between this two-grid code and the original method. Instead we will demonstrate its effectiveness on the radiative transfer problem.

A plot of the original Newton method solution is shown in Figure 5.10. This picture shows you what the solution looks like on the fine grid when the nonlinear problem is solved directly on the fine grid. Contrast this picture with the one in Figure 5.11 which shows a plot of the same nonlinear solve but on a coarser mesh. In Figure 5.12, the solution in Figure 5.11 has been interpolated to the fine mesh. This is the data that goes into the fine grid linear solve in the two-grid method. Figure 5.13 shows the effect of the fine grid linear solve. This last picture is the output generated by the two-grid method.

Figure 5.10: Original Newton solution plot

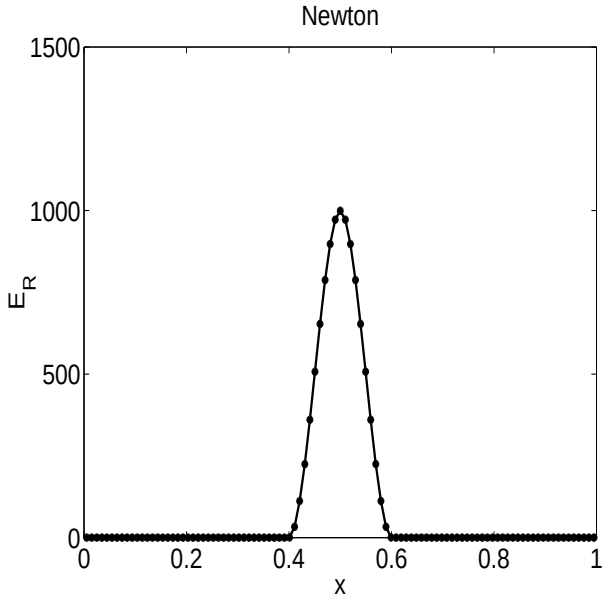


Figure 5.11: Two-grid Coarse solve solution plot

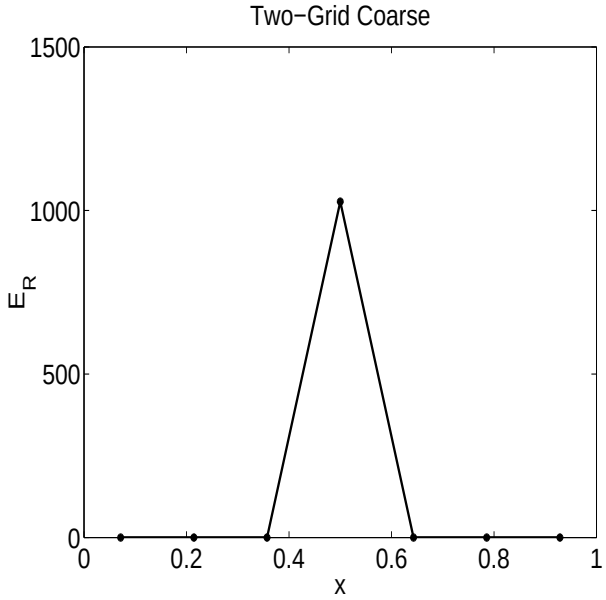


Figure 5.12: Two-grid Fine interpolated solution plot

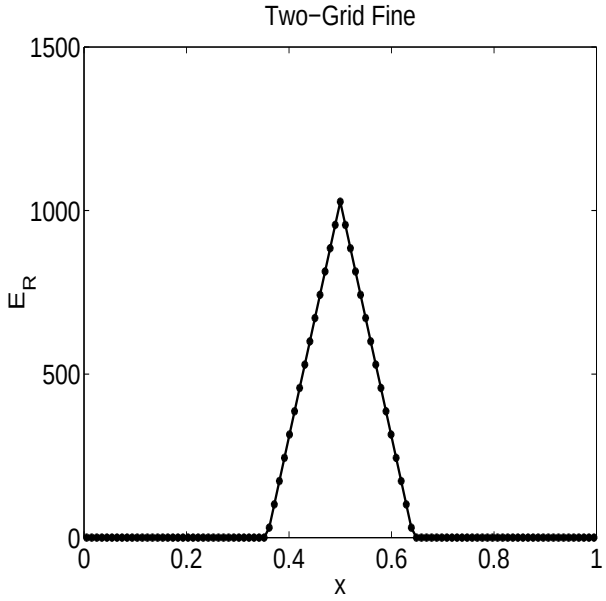
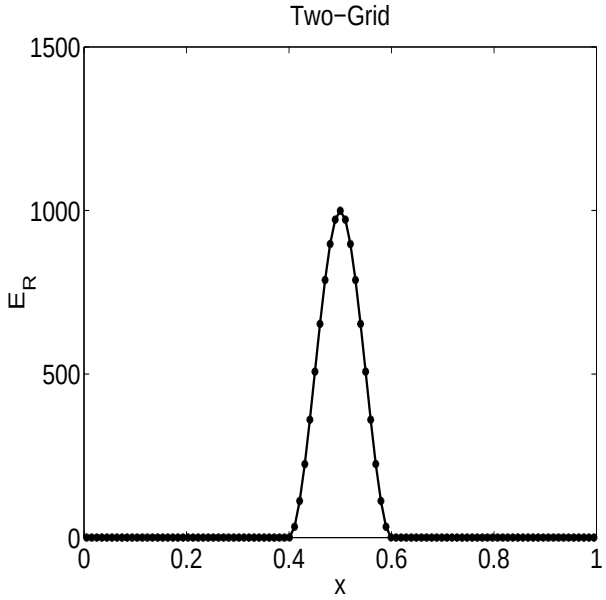


Figure 5.13: Two-grid Fine solve solution plot



5.4 Conclusions

The two-grid method is appealing because it offers the potential for much less work in solving very fine-grid nonlinear problems. On the other hand, the inter-grid transfers can be quite difficult to implement and can be expensive on top of that. The code was demonstrated on a model radiation diffusion problem and after optimization of the code, comparison runs with the original solution method will be made.

Chapter 6

Conclusions

We started out with a variety of very simple one dimensional root-finding examples which we blindly applied pseudo-transient continuation (Ψtc) to. These problems demonstrated that Ψtc really is not a general purpose root-finding method. We then applied Ψtc to some ordinary and partial differential equations (with real steady-state solutions), and we started to see why Ψtc is such a great method for finding the steady-state solutions to these problems. The Euler equations presented a very nice example where Ψtc worked wonders on a problem that caused Newton's method to fail. It also gave us an opportunity to explore the hybrid methods used so commonly in the computational fluid dynamics community.

Then we considered how Ψtc is applied to differential-algebraic equations, which come up often in combustion problems among other places. We established a global convergence result for Ψtc applied to semi-explicit index-1 differential algebraic equations, and

demonstrated its effectiveness on these problems. Of particular interest, was the lid driven problem that was formulated as a PDE and as a DAE. The DAE formulation performed quite a bit better than the PDE formulation and justified the use of Ψtc on the DAE.

And finally, we considered an implicit time-accurate two-grid method and its application in radiation diffusion (used e.g. in stellar fusion problems). The theory for this method has not been tested in many places, and getting a working code was more difficult than expected. We showed numerical results of the two-grid method on a nonlinear elliptic PDE, something that, to our knowledge, has not been done before. This method uses nonlinear solves at each time-step and could be combined with Ψtc.

Pseudo-transient continuation now has theory to support its use in ordinary differential equations and differential-algebraic equations. It is a very effective method for the problems it was designed for, and we demonstrated that with several numerical experiments.

References

[1] John D. Anderson. *Modern compressible flow with historical perspective*. McGraw-Hill, second edition, 1990.

[2] W. K. Anderson, J. L. Thomas, and B. Van Leer. Comparison of finite volume flux vector splittings for the Euler equations. *AIAA Journal*, 24:1453–1460, 1986.

[3] L. Armijo. Minimization of functions having Lipschitz-continuous first partial derivatives. *Pacific Journal of Mathematics*, 16:1–3, 1966.

[4] U. M. Ascher and L. R. Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*. SIAM, Philadelphia, 1998.

[5] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Lois Curfman McInnes, and Barry F. Smith. PETSc home page. http://www.mcs.anl.gov/petsc, 2002.

[6] A. Battermann, J. M. Gablonsky, A. Patrick, C. T. Kelley, K. R. Kavanagh, T. Coffey, and C. T. Miller. Solution of a groundwater flow problem with implicit filtering.

Technical Report CRSC-TR00-30, North Carolina State University, Dec 2000. To appear in Optimization and Engineering.

[7] Richard L. Bowers. *Numerical modeling in applied physics and astrophysics*. Jones and Bartlett, Boston, first edition, 1991.

[8] Dietrich Braess. *Finite elements: Theory, fast solvers, and applications in solid mechanics*. Cambridge University Press, Cambridge UK, second edition, 2001.

[9] K. E. Brennan, S. L. Campbell, and L. R. Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*, volume 14 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, first edition, 1996.

[10] Peter N. Brown, Frank Graziani, Ivan Otero, and Carol S. Woodward. Implicit solution of large-scale radiation diffusion problems. *Nuclear Explosives Code Developers Collaborations (NECDC)*, 2000. http://www.llnl.gov/CASC/nsde/pubs/Brown_Graziani_Otero_Woodward.pdf.

[11] Peter N. Brown and Carol S. Woodward. Preconditioning strategies for fully implicit radiation diffusion with material-energy transfer. *SIAM Journal on Scientific Computing*, 23(2):499–516, 2001.

[12] J. M. Burgers. A mathematical model illustrating the theory of turbulence. *Advanced Applications of Mechanics*, 1:171–199, 1948.

[13] George D. Byrne and Alan C. Hindmarsh. PVODE, an ODE solver for parallel

computers. *International Journal of High Performance Computing Applications*, pages 354–365, 1999.

[14] X. Chen and T. Yamamoto. Convergence domains of certain iterative methods for solving nonlinear equations. *Numerical Functional Analysis and Optimization*, 10:37–48, 1989.

[15] T. S. Coffey, C. T. Kelley, and D. E. Keyes. Psuedo-transient continuation and differential-algebraic equations. *Special Issue of SIAM Journal on Scientific Computing*, 2002. submitted 07/02.

[16] T. S. Coffey, R. J. McMullan, C. T. Kelley, and D. S. McRae. Globally convergent algorithm for nonsmooth nonlinear equations in computational fluid dynamics. Technical Report CRSC-TR01-28, North Carolina State University, November 2001. Submitted to Journal on Computational Applied Mathematics.

[17] Clint N. Dawson and Mary F. Wheeler. Two-grid methods for mixed finite element approximations of nonlinear parabolic equations. *Contemporary Mathematics*, 180:191–203, 1994.

[18] Clint N. Dawson, Mary F. Wheeler, and Carol S. Woodward. A two-grid finite difference scheme for nonlinear parabolic equations. *SIAM Journal on Numerical Analysis*, 35(2):435–452, April 1998.

[19] R. Dembo, S. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM Journal on Numerical Analysis*, 19:400–408, 1982.

[20] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, first edition, 1996.

[21] Jack R. Edwards. Numerical implementation of a modified Liou-Steffen upwind scheme. *AIAA Journal*, 32:2120–2122, 1994.

[22] Jack R. Edwards. Development of an upwind relaxation multigrid method for computing three-dimensional, viscous internal flows. *AIAA Journal of Propulsion and Power*, 12:146–154, 1996.

[23] Jack R. Edwards. A low-diffusion flux-splitting scheme for Navier-Stokes calculations. *Computers & Fluids*, 26(6):635–659, 1997.

[24] S. C. Eisenstat and H. F. Walker. Globally convergent inexact Newton methods. *SIAM Journal on Optimization*, (4):1–30, 1994.

[25] S. C. Eisenstat and H. F. Walker. Choosing the forcing terms in an inexact Newton method. *SIAM Journal on Scientific Computing*, (17):16–32, 1996.

[26] R. D. Falgout and U. M. Yang. *hypre*: a library of high performance preconditioners. In P.M.A. Sloot, C.J.K. Tan, J.J. Dongarra, and A.G. Hoekstra, editors, *Lecture notes in computational science - ICCS 2002 Part III*, volume 2331, pages 632–641. Springer-Verlag, 2002.

[27] C. W. Gear. *Numerical initial value problems in ordinary differential equations*. Prentice-Hall, Englewood Cliffs, New Jersey, first edition, 1971.

[28] A. A. Goldstein. *Constructive real analysis*. Harper and Row, New York, 1967.

[29] A. D. Gosman, W. N. Pun, A. K. Runchal, D. B. Spalding, and M. Wolfshtein. *Heat and mass transfer in recirculating flows*. Academic Press, New York, 1969.

[30] W. P. Graebel. *Engineering fluid mechanics*. Taylor & Francis, 2001.

[31] M. Heinkerschloss, C. T. Kelley, and H. T. Tran. Fast algorithm for nonsmooth compact fixed point problems. *SIAM Journal on Numerical Analysis*, 29:1769–1792, 1992.

[32] Charles Hirsch. *Numerical computation of internal and external flows*, volume 2. John Wiley & Sons, first edition, 1984.

[33] Paul Hovland, Boyana Norris, and Christian Bischof. ADIC home page. http://www.mcs.anl.gov/adic, 2002.

[34] H. Jiang and P. A. Forsyth. Robust linear and nonlinear strategies for solution of the transonic Euler equations. *Computers and Fluids*, 24:753–770, 1995.

[35] C. T. Kelley. *Iterative methods for linear and nonlinear equations*, volume 16 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, PA, first edition, 1995.

[36] C. T. Kelley and D. E. Keyes. Convergence analysis of pseudo-transient continuation. *SIAM Journal on Numerical Analysis*, 35(2):508–523, April 1998.

[37] D. E. Keyes. Domain decomposition methods for the parallel computation of reacting flows. *Computational Physics Comm.*, 53:181–200, 1989.

[38] D. E. Keyes. Aerodynamic applications of Newton-Krylov-Schwarz solvers. In R. N. et al., editor, *Proceedings of the 14th International Conference on Numerical Methods in Fluid Dynamics*, pages 1–20, New York, 1995. Springer.

[39] D. E. Keyes and M. D. Smooke. A parallelized elliptic solver for reacting flows in parallel computations and their impact on mechanics. *American Society of Mechanical Engineers*, pages 375–402, 1987.

[40] D.E. Knuth. *The TEX book*. Addison-Wesley, Reading, MA, 1984.

[41] Helmut Kopka and Patrick W. Daly. *A Guide to LATEX: Document Preparation for Beginners and Advanced Users*. Addison-Wesley, Reading, MA, second edition, 1995.

[42] P. D. Lax. Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Comm Pure and Applied Mathematics*, 7:159–193, 1954.

[43] B. Van Leer. Towards the ultimate conservative difference scheme. V. A second order sequel to Godunov's method. *Journal of Computational Physics*, 32:101–136, 1979.

[44] R. W. MacCormack and G. V. Candler. The solution of the Navier-Stokes equations using Gauss-Seidel line relaxation. *Computers and Fluids*, 17:135–150, 1989.

[45] R. J. McMullan. *Numerical boundary conditions simulating the interaction between upstream disturbances and an axial compressor.* PhD thesis, North Carolina State University, 2002. http://www.lib.ncsu.edu/etd/temp-list.html.

[46] W. Mulder and B. V. Leer. Experiments with implicit upwind methods for the Euler equations. *Journal of Computational Physics*, 59:232–246, 1985.

[47] J. Nédélec. Mixed finite elements in R^3. *Numerical Mathematics*, 35:315–341, 1980.

[48] P. D. Orkwis and D. S. McRae. Newton's method solver for the axisymmetric Navier-Stokes equations. *AIAA Journal*, 30:1507–1514, 1992.

[49] Paul D. Orkwis. *A Newton's method solver for the 2-dimensional and axisymmetric Navier-Stokes equations.* PhD thesis, North Carolina State University, 1990.

[50] J. M. Ortega and W. C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*, volume 30 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, first edition, 2000.

[51] G. C. Pomraning. *The equations of radiation hydrodynamics*, volume 54 of *International Series of Monographs in Natural Philosophy.* Pergamon Press, New York, first edition, 1973.

[52] J. V. Rakich. *Computational fluid mechanics.* North Carolina State University, 1978. Course Notes.

[53] P.A. Raviart and J.M. Thomas. A mixed finite element method for second order elliptic problems. In I. Galligani and E. Magenes, editors, *Mathematical aspects of finite element methods: Lecture notes in mathematics 606*, pages 292–315. Springer-Verlag, Berlin, 1977.

[54] P. L. Roe. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[55] P. L. Roe. *Upwind schemes using various formulations of the Euler equations.* SIAM, Philadelphia, 1985.

[56] P. L. Roe and J. Pike. Efficient construction and utilization of approximate Riemann solutions. In R. Glowinski and J. L. Lions, editors, *Computing Methods in Applied Sciences and Engineering*, Amsterdam, 1984. Norh Holland.

[57] L. F. Shampine. *Numerical solution of ordinary differential equations.* Chapman & Hall, New York, 1994.

[58] A. I. Shestakov, J. L. Milovich, and A. Noy. Solution of nonlinear Poisson-Boltzmann equation using pseudo-transient continuation and the finite element method. *Journal of Colloid & Interface Science*, 247(1):62–79, March 1 2002.

[59] John C. Tannehill, Dale A. Anderson, and Richard Pletcher. *Computational fluid mechanics and heat transfer.* Taylor & Francis, second edition, 1997.

[60] V. Venkatakrishnan. Newton solution of inviscid and viscous problems. *AIAA Journal*, 27:885–891, 1989.

[61] Li Wu and Myron B. Allen. A two-grid method for mixed finite-element solution of reaction diffusion equations. *Numerical Methods Partial Differential Equations*, 15:317–332, 1999.

[62] Li Wu and Myron B. Allen. Two-grid methods for mixed finite-element solution of coupled reaction-diffusion systems. *Numerical Methods Partial Differential Equations*, 15:589–604, 1999.

[63] Li Wu, Myron B. Allen, and Eun-Jae Park. Mixed finite-element solution of reaction-diffusion equations using a two-grid method. In *Computer Models in Water Resources XII*, pages 617–623, 1998.

[64] Jinchao Xu. A novel two-grid method for semilinear elliptic equations. *SIAM Journal on Scientific Computing*, 15:231–237, 1994.

[65] Jinchao Xu. Two-grid discretization techniques for linear and nonlinear PDEs. *SIAM Journal on Numerical Analysis*, 33(5):1759–1777, October 1996.