A Dissertation

entitled

External Verification Analysis: A Code-Independent Approach to Verifying Unsteady Partial Differential Equation Solvers

by

Daniel Ingraham

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Doctor of Philosophy Degree in Mechanical Engineering

Dr. Ray Hixon, Committee Chair

Dr. Sorin Cioc, Committee Member

Dr. James DeBonis, Committee Member

Dr. Mehdi Pourazady, Committee Member

Dr. Chunhua Sheng, Committee Member

Dr. Patricia R. Komuniecki, Dean
College of Graduate Studies

The University of Toledo

May 2015

An Abstract of

External Verification Analysis: A Code-Independent Approach to Verifying Unsteady Partial Differential Equation Solvers

by

Daniel Ingraham

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Doctor of Philosophy Degree in Mechanical Engineering

The University of Toledo
May 2015

External Verification Analysis (EVA), a new approach to verifying unsteady partial differential equation codes, is presented. After a review of the relevant code verification literature, the mathematical foundation and solution method of the EVA tool is discussed in detail. The implementation of the EVA tool itself is verified through an independent Python program. A procedure for code verification with the EVA tool is described and then applied to the three-dimensional form of a high-order non-linear computational aeroacoustics code.

To Owen!

Acknowledgments

The questions and comments from Drs. Cioc, DeBonis, Pourazady, and Sheng provided clarity and motivation, and improved the quality of this work. I'm especially grateful for Dr. DeBonis making the trip from NASA Glenn in Cleveland to UT (twice!).

Graduate school was made a little easier and less lonely through the friendship and advice of Vasanth Allampalli, Adrian Sescu, Nima Mansouri, Soroush Yazdani, and Caleb Ritenour, to name a few.

The success, such as it is, that I've had so far in my professional life is largely due to Dr. Hixon's mentoring and support. Working for him has always been challenging and invigorating, and I'm grateful for the knowledge I've gained as a member of his research group.

Finally, I can't express how essential the love I've received from my family has been to me. But maybe Fortran can?

```
PROGRAM Thank_Family
  IMPLICIT NONE
  DO WHILE (.TRUE.)
    PRINT*, "Mom: thank you for your constant support and encouragement."
    PRINT*, "Dad: thank you for your unfailing poise and optimism."
    PRINT*, "Cassie: thank you for your faith and kindness."
    PRINT*, "Trisha: thank you for the phone calls -- they meant more than you know."
    PRINT*, "Melissa: too much to list. You are still, and will always be, my everything."
  END DO
END PROGRAM Thank_Family
```

Contents

List of Tables

List of Figures

List of Abbreviations

AB4	Fourth-order Adams-Bashforth scheme
ASME	American Society of Mechanical Engineers
BASS	Broadband Aeroacoustic Stator Simulator, a CAA code
C_6	Compact sixth-order finite differencing scheme
C+F	*Computers & Fluids* journal
CAA	Computational Aeroacoustics
CAS	Computer algebra system
CFD	Computational Fluid Dynamics
CFL	Courant–Friedrichs–Lewy
CK	Cauchy-Kowalewski
DRP	Dispersion Relation Preserving, an optimized fourth-order finite differencing stencil
E_2	Explicit second-order finite differencing scheme
E_6	Explicit sixth-order finite differencing scheme
EVA	External Verification Analysis
FSI	Fluid-Structure Interaction
HALE-RK	High-Accuracy Large-step Explicit Runge-Kutta
HOT	Higher-order terms
IC	Initial condition
JCP	*Journal of Computational Physics*
MES	Method of Exact Solutions
MMS	Method of Manufactured Solutions
MTSAB	Multi-time-step Adamas-Bashforth scheme

ODE	Ordinary differential equation
PDE	Partial differential equation
PVSV	Primitive variable specific volume, i.e. the variables σ, u, v, w, p
RANS	Reynolds-Averaged Navier-Stokes
RK-2N	$2N$-storage Runge-Kutta scheme
RK4	Classic fourth-order Runge-Kutta scheme
RK4L	RK-2N version of RK4
RK56	Stanescu and Habashi's five-six-stage RK scheme
RK5L	Jameson's second-order five-stage RK scheme
RK67	Six-seven-stage HALE-RK scheme
RK7S	Seven-stage HALE-RK scheme
V&V	Verification and Validation

List of Symbols

x, u	Independent and dependent variables of a generic PDE
F	Generic PDE operator
f	Manufactured solution to a generic PDE
G	MMS source term for a generic PDE
H	Cauchy problem operator
y	Independent spatial variables of a generic Cauchy problem PDE
t, τ	Independent variable normal to a Cauchy surface, and location of the surface in the t-direction
v	Initial solution on Cauchy surface
U	Approximate solution to a Cauchy problem
ξ, η	Transformed coordinates
a, b	Coefficients of a first-order PDE
U	Collection of dependent variables of a PDE system
X	Collection of independent variables of a PDE system
x, y, z	The Cartesian coordinates
t	Time
t_0	Initial time level
Δt	Time step size
$\vec{Q}$	Vector of conserved solution variables of a PDE system
$\vec{E}$, $\vec{F}$, $\vec{G}$	Vector of Cartesian fluxes in the x, y, z directions, respectively
ρ	Density
σ	Specific volume
u, v, w	Velocities in the x-, y, and z-directions, respectively
V	Velocity vector
p	Pressure
E_{total}	Total energy
γ	Ratio of specific heats
k	Thermal conductivity
R	Ideal gas constant
μ	Dynamic viscosity

ν ..... Kinematic viscosity
Θ ..... Navier-Stokes conduction term
Φ ..... Navier-Stokes mechanical dissipation term

E, I ..... Envelope and inner parts of the EVA initial condition

l_1, l_2, $l_{\max}$ ... Error norms
ϵ ..... Generic error norm
ϕ, Φ ..... PDE code and EVA solutions
h ..... Discretization measure (i.e., grid spacing or time step size)
p ..... Observed order-of-accuracy

Chapter 1

Introduction and review of the literature

1.1 Verification and validation

The terms verification and validation (V&V), nearly synonymous in common usage, are defined differently in the verification and validation community. Verification, in the V&V context, refers to activities meant to develop an understanding of how closely a code's solution corresponds to the exact solution of the partial differential equation (PDE) it solves. Validation, on the other hand, is a comparison between the PDE's solution and reality, i.e., data from experiment. This distinction is inspired by terminology from the software development research field, specifically, by Bohem [1] in a textbook on software economics:

Verification "Are we building the product right?"

Validation "Are we building the right product?"

Bohem justifies these definitions by pointing out that the word verification has its roots in the Latin *veritas* (truth), while validation originates from *valere* (to be worth), though Roache [2] maintains that the reversed definitions would be equally valid. Blottner [3] adapts Bohem's V&V definitions for a scientific computing context:

Verification "[Are we] solving the governing equations right?"

Validation "[Are we] solving the right governing equations?"

Blottner's terminology nicely captures the difference between verification and validation, and has been embraced by other researchers [2, 4, 5].

This work is focused on the *verification* of PDE codes, and does not address validation.

Verification is further subdivided into two separate procedures: code verification and solution verification [2, 5]. Code verification (or "ascertaining code correctness" [6]) is concerned with *error evaluation*: the goal is to ensure that the numerical algorithms used in a PDE code are implemented properly. Solution verification (also called "verification of calculations" [2], "numerical error estimation" [7], or "discretization error estimation" [4]), on the other hand, is an *error estimation* process — the aim of solution verification is to estimate or band the inherent numerical error of a particular calculation of a PDE code. (See Roache's article [6] for a particularly clear explanation of the relationships between code verification, solution verification, and validation.) Code and solution verification, while separate activities, are usually not thought of as independent in the V&V community: attempts at solution verification have meaning only after the code has been verified.

This work is entirely focused on code verification, specifically on code verification through the order-of-accuracy test.

1.2 Code verification and the order-of-accuracy test

As discussed above, code verification is a process by which one attempts to show that a PDE code is bug-free — that the numerical methods it uses are implemented correctly. Oberkampf and Roy [5] point out that there is more than one way to obtain evidence that a PDE code is working properly: one can subject the code to "simple

tests" (e.g., checking for symmetry, or conservation in conserved quantities, if applicable), code-to-code comparisons ("is Code A 'close' to Code B?"), or convergence tests (i.e., determining if the code's error goes to zero, but not checking the rate). But these researchers argue (and others agree [4, 8, 2]) that the order-of-accuracy test is the most sensitive and rigorous code verification method. In this approach (also referred to as "order-verification" [8] or "code order verification" [4]), a series of numerical solutions are compared to a highly-accurate reference solution as a "discretization measure" is varied, and the rate of change of the error between the numerical and reference solution is calculated. This procedure is commonly referred to as a grid refinement study, though the "discretization measure" could also be a time step size for unsteady calculations. The goal of code verification is to prove (hopefully) that, as a discretization measure is decreased, the PDE code's error approaches zero at the expected rate: the formal order-of-accuracy of the PDE code's numerical scheme(s).

1.2.1 Obtaining reference solutions

Calculating the convergence rate of a PDE solver's numerical error, the goal of the order-of-accuracy test, is straightforward — the difficulty lies in calculating the numerical error *itself*. To find the error in any sort of quantity, one needs some sort of "reference solution" — values that are considered "exact," or at least considerably more accurate than the numerical results. This is the challenging part of the order-of-accuracy test, and code verification generally: obtaining reference solutions to a code's PDEs that are general enough to exercise all aspects of the code, and are accurate enough to allow for a good estimation of the code's numerical error. There are currently two approaches to constructing reference solutions for code verification: the Method of Exact Solutions, and the Method of Manufactured Solutions. Each of these methods will be discussed in this section.

1.2.2 The Method of Exact Solutions

Perhaps the most straightforward approach to obtaining a reference solution for code verification purposes is to use an exact solution from the available literature. For example, to verify a PDE code that solves the linear advection equation,

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0, \tag{1.1}$$

the exact solution

$$u(x,t) = f(x - at), \tag{1.2}$$

where f is any differentiable function, is taken as the reference solution and used to calculate the PDE code's error.

Equation (1.2) is a quite general solution to Equation (1.1): one has considerable flexibility in choosing f, and thus the form of the solution. Solutions to PDEs of more practical interest are usually not as flexible. For example, consider the available solutions to the incompressible Navier-Stokes equations, i.e.,

$$\begin{aligned} \nabla \cdot \vec{V} &= 0 \\ \frac{\partial \vec{V}}{\partial t} + \vec{V} \cdot \nabla \vec{V} &= -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{V}, \end{aligned} \tag{1.3}$$

where $\nu = \frac{\mu}{\rho}$. White [9, Chapter 3] distinguishes between two types of solutions to (1.3): linear and nonlinear, referring to the linearity of the PDE that is eventually solved. Many of the common linear solutions to the incompressible Navier-Stokes are parallel flows, i.e., the velocity field in Cartesian coordinates takes the form [10, Chapter 5]

$$u = u(y,z,t); \quad v = 0; \quad w = 0 \tag{1.4}$$

which, after considering the y- and z-momentum equations, implies that the pressure

depends only on x, and the x-momentum equation reduces to

$$\frac{\partial u}{\partial t} = -\frac{1}{\rho}\frac{dp}{dx} + \nu\left(\frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right), \tag{1.5}$$

the unsteady heat equation with a source term ($-\frac{1}{\rho}\frac{dp}{dx}$). If one applies the conditions

$$\begin{aligned} \frac{\partial u}{\partial t} = 0; \quad & \frac{\partial p}{\partial x} = 0 \\ u(x, y = 0, z) = 0; \quad & u(x, y = H, z) = U = \text{const.}, \end{aligned} \tag{1.6}$$

Equation (1.5) reduces to

$$\mu\frac{\partial^2 u}{\partial y^2} = 0 \tag{1.7}$$

with the solution

$$u = \frac{y}{H}U, \tag{1.8}$$

the simplest form of Couette flow [9, pp. 98–106][10, pp. 83–85][11, pp. 182–183], the steady viscous flow between infinite parallel plates with moving at a constant velocity U. If, instead, the conditions

$$\begin{aligned} \frac{\partial u}{\partial t} = 0; \quad & \frac{\partial p}{\partial x} = \text{const.} \\ u(x, y = -H, z) = 0; \quad & u(x, y = H, z) = 0, \end{aligned} \tag{1.9}$$

are used with Equation (1.5), one finds

$$\mu\frac{\partial^2 u}{\partial y^2} = \frac{dp}{dx} \tag{1.10}$$

with the solution

$$u = -\frac{1}{2\mu}\frac{dp}{dx}\left(H^2 - y^2\right), \tag{1.11}$$

the flow between infinite parallel plates subject to a constant pressure gradient ($\frac{dp}{dx}$), commonly known as Poiseuille flow [9, pp. 106–125][10, pp. 85–87][11, pp. 180–182]. Alternatively, if Equation (1.5) is recast in cylindrical coordinates with x the axial direction, the parallel flow equation becomes

$$\rho\frac{\partial u}{\partial t} = -\frac{dp}{dx} + \mu\left(\frac{1}{r}\frac{\partial}{\partial r}\left[r\frac{\partial u}{\partial r}\right] + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}\right), \tag{1.12}$$

and the conditions equivalent to Equation (1.10) in cylindrical coordinates are

$$\begin{aligned} \frac{\partial u}{\partial t} = 0; \quad & \frac{\partial p}{\partial x} = \text{const.} \\ u(x, r = R, \theta) = 0; \quad & u(x, r = 0, \theta) = \text{finite} \end{aligned} \tag{1.13}$$

which reduce Equation (1.12) to

$$\frac{dp}{dx} = \mu\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) \tag{1.14}$$

yielding the solution

$$u = -\frac{1}{4\mu}\frac{dp}{dx}\left(R^2 - r^2\right). \tag{1.15}$$

The three preceding parallel flow solutions (Equation (1.8), Equation (1.11), Equation (1.15)) are very simple: each is a function of only one spatial variable (the transverse coordinate y or r), and the form of this function is also quite simple (linear or quadratic). More complex solutions would provide more convincing verification test cases — for example, an unsteady solution (i.e., one that depends on t, the time coordinate) would be a significant improvement. For example, if one subjects Equation (1.5) to the boundary conditions

$$u(y, t \leq 0) = 0; \quad u(y = 0, t > 0) = U; \quad u(y \to \infty, t) = 0, \tag{1.16}$$

corresponding to the flow of an initially stationary fluid near a flat plate suddenly moving at a constant speed U in the x-direction, the solution to Equation (1.5) is

$$u = U \operatorname{erfc}\left(\frac{y}{2\sqrt{\nu t}}\right) \tag{1.17}$$

which is known as the solution to "Stokes' First Problem" [9, pp. 129–131][10, pp. 90–93][11, pp. 189–190]. The boundary conditions for Stokes' second problem [9, pp. 131–133][10, pp. 93–94][11, pp. 191–193], involving a flat plate oscillating in its own plane, are

$$u(y, t \leq 0) = 0; \quad u(y = 0, t > 0) = U\cos(\omega t); \quad u(y \to \infty, t) = 0, \tag{1.18}$$

with solution

$$u(y, t) = Ue^{-ky}\cos(\omega t - ky) \tag{1.19}$$

where

$$k = \sqrt{\frac{\omega}{2\nu}}. \tag{1.20}$$

The proceeding solutions all are well-known incompressible Navier-Stokes exact solutions that are easy to evaluate: once the appropriate input parameters are chosen (e.g., the viscosity μ or ν, pressure gradient $\frac{dp}{dx}$, spatial extent of the problem H or R , etc.), the form of the solution for the axial velocity u is quite simple, and could therefor be implemented in a short computer code without trouble. And because the shape of the domain and boundary conditions are also simple, the problems could also be solved with a PDE code without difficulty. Unfortunately, the simplicity of these solutions make them less useful for code verification purposes, since many of the terms in the Navier-Stokes equations vanish when subjected to the conditions of Couette flow, Poiseuille flow, etc.. For example, all but two terms of the Navier-Stokes equations drop out for Poiseuille flow: only the pressure gradient $\frac{dp}{dx}$ and second

transverse derivative of the axial velocity $\frac{\partial^2 u}{\partial y^2}$ remain in Equation (1.10). For Couette flow, Equation (1.7) shows that *all* terms in the Navier-Stokes equation are zero. This disappearance of terms is problematic, since it may hide programming mistakes in the part of the PDE code that implements the term. More complicated solutions are necessary to rigorously verify PDE codes.

More complex parallel flow solutions to the incompressible Navier-Stokes equations exist. The starting flow in a pipe [9, pp. 125–127][11, pp. 193–195], i.e., flow that initially is stationary and then suddenly subjected to a constant pressure gradient $\frac{dp}{dx}$, provides a nice example of what can be expected of such solutions. For this flow, the governing equation is again Equation (1.12), with boundary conditions

$$u(r, t \leq 0) = 0; \quad u(r = R, t) = 0. \tag{1.21}$$

The solution for u that satisfies Equation (1.12) and Equation (1.21) is

$$u\left(r,t\right) = \frac{-\frac{dp}{dx}}{4\mu}\left(R^2 - r^2\right) + \frac{2R^2\frac{dp}{dx}}{\mu}\sum_{n=1}^{\infty}\frac{J_0\left(\lambda_n\frac{r}{R}\right)}{\lambda_n^3 J_1(\lambda_n)}\exp\left(-\lambda_n^2\frac{\nu t}{R^2}\right) \tag{1.22}$$

where J_0 and J_1 are Bessel functions of the first kind of order zero and one, respectively, and λ_n are positive roots of J_0. Obviously, Equation (1.22) is more complicated to evaluate than the steady Poiseuille flow in Equation (1.11), since the former involves an infinite sum, Bessel functions, and Bessel function roots. And it is only slightly more general than Equation (1.11): only one addition term is non-zero ($\frac{\partial u}{\partial t}$). One would expect this additional complexity to increase as more interesting geometries and initial conditions are used with the parallel flow equation, Equation (1.5). And even if a completely general solution is found and evaluated, it would still carry with it the assumptions of parallel flow, which require only one non-zero velocity, vanishing terms in the continuity equation, constant pressure gradient in the flow

direction, etc.. More general solutions are needed.

Perhaps non-linear solutions to the incompressible Navier-Stokes equations will provide the generality sought? A popular example of a non-linear solution of viscous incompressible flow is the flow near an infinite rotating disk [9, pp. 155–161][10, pp. 102–107]. The disk is assumed to rotate at a constant angular velocity ω in the $z = 0$ plane. If the motion of the fluid is assumed to be steady and axisymmetric, the Navier-Stokes equations in cylindrical coordinates reduce to

$$\begin{aligned} v_r\frac{\partial v_r}{\partial r} - \frac{{v_\phi}^2}{r} + v_z\frac{\partial v_r}{\partial z} &= -\frac{1}{\rho}\frac{\partial p}{\partial r} + \nu\left(\frac{\partial^2 v_r}{\partial r^2} + \frac{\partial \frac{v_r}{r}}{\partial r} + \frac{\partial^2 v_r}{\partial z^2}\right) \\ v_r\frac{\partial v_\phi}{\partial r} - \frac{v_r v_\phi}{r} + v_z\frac{\partial v_\phi}{\partial z} &= \nu\left(\frac{\partial^2 v_\phi}{\partial r^2} + \frac{\partial \frac{v_\phi}{r}}{\partial r} + \frac{\partial^2 v_\phi}{\partial z^2}\right) \\ v_r\frac{\partial v_z}{\partial r} + v_z\frac{\partial v_z}{\partial z} &= -\frac{1}{\rho}\frac{\partial p}{\partial z} + \nu\left(\frac{\partial^2 v_z}{\partial r^2} + \frac{1}{r}\frac{\partial v_z}{\partial r} + \frac{\partial^2 v_z}{\partial z^2}\right) \end{aligned} \tag{1.23}$$

with continuity equation

$$\frac{\partial v_r}{\partial r} + \frac{v_r}{r} + \frac{\partial w}{\partial z} = 0. \tag{1.24}$$

The boundary conditions are the no-slip condition at the disk surface, and purely axial flow far from the disk, i.e.,

$$\begin{aligned} &v_r(r,\phi,z=0) = v_z(r,\phi,z=0) = p(r,\phi,z=0) = 0 \\ &v_\phi(r,\phi,z=0) = \omega r \\ &v_r(r,\phi,z\to\infty) = v_z(r,\phi,z\to\infty) = 0 \end{aligned} \tag{1.25}$$

where p is interpreted as the difference between the local pressure and some reference value. Equation (1.23) are solved using a similarity approach, specifically by assuming

$$v_r = \omega r F(\zeta); \quad v_\theta = \omega r G(\zeta); \quad v_z = \sqrt{\nu\omega}H(\zeta); \quad p = \rho\nu\omega P(\zeta) \tag{1.26}$$

where $\zeta = z\sqrt{\omega/\nu}$ is the similarity variable. After substituting the expressions of

Equation (1.26) into Equation (1.23) and Equation (1.24), the system of ordinary differential equations

$$\begin{aligned} 2F + H' &= 0 \\ F^2 + F'H - G^2 - F'' &= 0 \\ 2FG + HG' - G'' &= 0 \\ P' + HH' - H'' &= 0 \end{aligned} \tag{1.27}$$

is obtained, which must be solved numerically.

The solution of Equation (1.27) is one of the most complex available for the incompressible Navier-Stokes equations, but it is still relatively simple: it is steady and axisymmetric, and the quantities $\frac{v_r}{r}$, $\frac{v_\phi}{r}$, v_z, and p all depend only on the axial coordinate z. So the rotating disk case suffers from the same problems that the parallel flow solutions exhibited: easy-to-evaluate solutions are only available for very simple flows, and more interesting flows gain more in computational complexity (e.g., infinite sums and tricky-to-evaluate functions, or the numerical integration required for the rotating disk system) than in physical complexity.

Despite its disadvantages, examples of the use of MES exist in the V&V literature. Roy and Oberkampf [12] use an exact solution to the quasi-one-dimensional Euler equations to verify a simple finite volume CFD code. Salari and Knupp [13] provide two clever examples of constructed exact solutions to more complicated equations. In the first, the authors address the unsteady two-dimensional heat equation with scalar, spatially-varying conductivity. They begin by using the classic separation of variables technique, then find solutions for the resulting simplified differential equations. No attempts are made to satisfy given boundary conditions up front, which allows for more flexibility in determining the final form of the solution. In the second example, a streamfunction approach is used to reduce the two-dimensional incompressible

steady Navier-Stokes equations to a single fourth-order equation. Again, boundary conditions are not specified, allowing the authors to choose a solution for the stream-function that is smooth and easily related to the original dependent variables (the two velocity components and pressure).

Oberkampf and Roy provide additional examples of MES in reference [5]. They use an exact solution to the one-dimensional steady nonlinear Burger's equation to verify a solver using the finite difference method, and verify a 2D structural code using the Finite Element method with an exact solution.

The Method of Exact Solutions is certainly a valid approach to code verification. When combined with the order-of-accuracy test, a code verified with MES is very likely bug-free for the type of problem tested — in other words, for the functionality solving the problem requires (uniform or nonuniform grids, constant or varying properties, etc.). As emphasized by Roache [2], Salari and Knupp [13, 4], and others, and as was seen in the preceeding sampling of exact solutions to the incompressible Navier-Stokes equations, exact solutions to non-trivial PDEs can be complicated to implement and evaluate in a computer code, since they tend to contain infinite series, complicated integrals, etc..

The primary disadvantage of using exact solutions for code verification, as pointed out numerous times in the literature [2, 13, 5], is that they tend to not exercise all of the terms in governing equations: many will be identically zero. (This was seen in the survey of exact solutions to the incompressible Navier-Stokes equations above.) The reason for this lack of general solutions is clear: if flexible, general solutions did exist for complex PDEs of practical interest, there would be no reason to write and run a computer code to obtain numerical approximations. The entire motivation behind numerical methods is that they allow scientists and engineers to solve problems that would be intractable with classic mathematical techniques.

1.2.3 The Method of Manufactured Solutions

To overcome the limitations of the Method of Exact Solutions, Steinburg and Roache developed the Method of Manufactured Solutions [14, 15] (MMS). The Method of Manufactured Solutions is thoroughly explained in multiple places in the V&V literature — see references [2, 13, 15, 4, 5] for especially clear presentations. The MMS approach reverses the usual perspective of solving PDEs. Instead of attempting to find a solution that satisfies a governing equation and associated boundary conditions exactly, one chooses a solution that does *not* satisfy the governing equations. The assumed solution is then inserted into the governing equations, and the resulting residual expression is added to the original governing equations as a source term. The modified governing equation now has the chosen (or manufactured) solution as an *exact* solution. The PDE code is then modified to include the new MMS source term, and the grid or time step studies associated with the order-of-accuracy test can be performed using the manufactured solution as the reference solution.

A concrete example makes the MMS approach clear. Suppose that a PDE code solves the linear advection equation, i.e.,

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0, \tag{1.28}$$

and one wishes to use MMS to verify the code. The reference (manufactured) solution is specified next. A simple choice is made

$$\hat{u}(x,t) = \sin(x - bt) \tag{1.29}$$

where $\hat{u}$ is the reference solution and b an arbitrary constant. The proposed solution

is then substituted into the governing equation to find the MMS source term:

$$\begin{aligned} S(x,t) &= \left[\sin(x-bt)\right]_t + a\left[\sin(x-bt)\right]_x \\ &= -b\cos(x-bt) + a\cos(x-bt) \\ &= (a-b)\cos(x-bt). \end{aligned} \tag{1.30}$$

The source term $S(x,t)$ is finally added to the original governing equation:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = S(x,t) = (a-b)\cos(x-bt). \tag{1.31}$$

The manufactured solution $\hat{u} = \sin(x-bt)$ is an *exact* solution of the modified equation (1.31), and can be used as a reference solution when verifying any PDE code that can be made to solve (1.31).

MMS, combined with the order-of-accuracy test, has proved to be a powerful code verification technique. It enjoys fairly wide adoption among those in the CFD community concerned with V&V, having been used to verify finite volume method CFD codes [16, 17, 18, 19] solving both the Euler and Navier-Stokes equations on unstructured meshes. A series of papers by Bond et al. [20, 21, 22] shows how MMS can be used to verify boundary conditions in CFD codes, including those with turbulence models. Eça et al. [23, 24] have also done significant work on the difficult problem of verifying turbulence models in CFD codes. MMS is used to verify CFD codes with fluid-structure interaction (FSI) capabilities by Tremblay et al. in [25], Etienne et al. [26], and Gong et al. [27]; an unsteady free-surface CFD code [28]; low-Mach number flows [29]; and multi-phase flows [30]. Finally, the original application of MMS was verifying an elliptic PDE solver with boundary-fitted coordinates [14].

Examples of non-CFD MMS applications are less prevalent in the literature, but do exist. Merroun et al. [31] use MMS to verify a code simulating convective heat

transfer in multiple channels (the code's application is nuclear reactor cooling), and Pautz [32] use MMS to verify a finite element code solving equations with applications to nuclear transport. Also, one of the FSI examples already cited [27] had its origin in biomechanics problems.

The flexibility of the Method of Manufactured Solutions overcomes the disadvantages of the traditional Method of Exact Solutions. First, because the user is in complete control of the form of the reference solution, elementary functions can be used to insure the manufactured solution is easy to implement (sin, cos, exp, polynomials, etc.). Second, the user can construct as general of a solution as needed to exercise all the terms in the PDE to be verified, since, again, the form of the solution is up to the user. (Reference [4] gives a helpful list of recommendations for constructing manufactured solutions.)

Most of the disadvantages of MMS cited in the literature are tied to the MMS source term that must be added to the governing equations. Because the MMS term will be, in general, a function of all independent variables in the PDE, the PDE code must support *distributed* sources [2, 13, 5]. If not present, distributed source term capability is not difficult to implement in most algorithms [4, 6], assuming that the source code is available (not often the case for commercial solvers). The originator of the method does admit, however, that some developers are reluctant to invest the time needed to add this capability to their code, and to construct a manufactured solution and corresponding source term that MMS requires [6]. The MMS source term also tends to be quite complicated for equations of practical interest, but the popularity of computer algebra systems (CAS) has made this less of a problem, since most CAS applications can convert symbolic expressions into code fragments for popular programming languages. Eça and Hoekstra [23] warn that unrealistic manufactured solutions can lead to large MMS source terms, which can in turn overwhelm the portion of the solution contributed by the discretization scheme, though an example

of this was not found in the literature. MMS source terms were found, however, to cause problems with the solution variables associated with RANS turbulence models in reference [33].

1.3 Arguments for and against code verification

Several arguments for the importance of code verification have been made in the literature. Researchers in the V&V community point out that code verification exercises are an opportunity for PDE code developers to evaluate their code's numerical algorithms, and perhaps even improve them [22]. The new diffusion operator in Veluri et al.'s work mentioned previously is an instance of code verification leading to an improved scheme. Roache [2] gives two examples where grid refinement studies lead to a greater understanding of the accuracy of newly-developed numerical methods.

The maturation of PDE codes provides additional justification for code verification. PDE codes have become much more capable over the last century [5] by incorporating more sophisticated numerical schemes and physical models. This trend makes bugs more likely (due to the more complicated numerical schemes), more difficult to detect in a code's output (due to the sophisticated physics), and more difficult to find in the code's source (due to the complex programming required for general-purpose or highly-optimized codes) [4]. Thus one would expect the likelihood of programming mistakes being present in PDE codes (already quite high, according to Hatton's "T-experiments [34]") to increase in the future, making code verification activities (and other V&V techniques) even more important. And since PDE codes are increasingly being used to inform critical decisions [35, 36, 5, 12], the risk of bugs will also increase.

An additional argument for code verification is related to solution verification. Attempts at solution verification (i.e., estimating the numerical error present in a

PDE code calculation), are facilitated by code verification. To see why this is so, first assume that the error in a PDE code's solution takes the form

$$\epsilon = \mathcal{O}\left(h^p\right) + \text{HOT} \tag{1.32}$$

where h is some measure of the discretization the PDE code has applied to the governing equation it solves (mesh spacing, time step size, etc.), p is the order of the discretization, and HOT are higher-order terms that will be neglected. The problem statement of code verification through the order-of-accuracy test could be expressed informally as "given h and ϵ, find p," that is, determine the rate (p) at which the PDE code's error approaches zero as the discretization measure h is reduced. Solution verification, on the other hand, switches the inputs and outputs of the code verification problem: "given h and p, find ϵ". Consider, however, how the solution verification process would be affected by an incorrect value of p: the error estimation would be significantly lower than the true value (since one rarely under-predicts p, i.e., mistakenly assumes PDE code is *less* accurate than it is). Thus a solid understanding of a PDE code's *order-of-accuracy* is needed before the *accuracy* of a particular PDE code solution is quantified. Meaningful solution verification is dependent on thorough code verification.

As V&V's popularity grows, PDE code developers may need to verify their codes whether they like it or not. Many journals require evidence of code or solution verification work to be included in a manuscript before it is accepted for publication. Examples of journals with such a requirement include the American Society of Mechanical Engineer's Journal of Fluids Engineering [37, 38], the International Journal for Numerical Methods in Fluids [39], the American Institute of Aeronautics and Astronautics Journal [40, 41], and the Journal of Heat Transfer [42]. The focus of these policy statements is solution verification, but some [38, 39, 41] explicitly call

for reporting code verification activities. And, as discussed above and affirmed by others [12, 5], code verification is a prerequisite of solution verification.

Perhaps the most obvious argument for code verification, however, is that it helps PDE code developers find mistakes (bugs) in their code. Numerous examples of the effectiveness of code verification exist. Roache [2] describes how grid refinement studies lead to the discovery of a mistake in the finite difference stencil used for cross-derivative terms in an electromagnetic code. Knupp and Salari [4, 13] report the results of subjecting a previously-verified two-dimensional Navier-Stokes code to sabotage by one of the authors (i.e., source code modifications that intentionally introduced mistakes) and code verification by the other to determine if the mistakes could be found. The authors conclude that code verification (via the method of manufactured solutions) was able to identify any changes that caused the code to incorrectly solve its governing equations. Veluri [17] and Veluri et al. [19] describe shortcomings of a Navier-Stokes finite volume diffusion operator uncovered through code verification work, prompting one of the authors to redesign the operator, improving convergence results. In a series of papers, Bond et al. [20, 21, 22] document how they were able to find and correct a mistake in the implementation of a slip boundary condition in an Euler/Navier-Stokes/RANS solver, and two bugs in various gradient reconstruction methods implemented in the code. Finally, Roache [43] briefly describes bugs in a number of groundwater, contaminant transport, and CFD codes found and corrected through code verification exercises.

One would imagine that most serious researchers who develop and/or use PDE codes would agree that the accuracy of these codes is important, but objections to code verification and V&V generally have been raised. Roache [44] reviews the arguments made against an ASME *Journal of Fluids Engineering* policy statement [37] on the accuracy of PDE code results published in that journal. As mentioned earlier, the majority of the journal policy statements are concerned with *solution* verification, but

some of the objections discussed by Roache could be leveled against code verification as well. For example, some critics of the policy argue that code verification can be a time- and resource-intensive process, especially when verifying high-order methods that require finer grids and/or smaller time step sizes to find the schemes' asymptotic range. Roache responds, in part, that the quantification of numerical accuracy is a necessary part of "this CFD business" — ensuring that a PDE code's results are mathematically sound is needed if they are to be taken seriously.

Another objection to code verification is that good agreement with experiment renders it unnecessary. While Roache does admit that this argument is "attractive," he maintains replacing verification work with validation "does not seem to work very often because the agreement with the experiment is usually not universal." If only a portion of the PDE code results match the experimental data (e.g., for a subset of the parameters varied in the experiment/calculation), what is one to conclude? It seems wiser to address first the accuracy of the numerics (i.e., verification) *before* tackling the real world (i.e., validation).

1.4 Code verification in practice

To investigate how frequently code verification techniques are used in practice, the issues of two prominent peer-reviewed journals were read, and the code verification technique(s) used by the authors, if any, were noted. The specific journals selected were the 274$^{\text{th}}$ issue of the *Journal of Computational Physics* [45] (JCP) and the 103$^{\text{rd}}$ issue of *Computers and Fluids* [46] (C+F), both published in the fall of 2014. Each of these journals are highly-ranked in their fields of study: according to Google Scholar, the *Journal of Computational Physics* has the largest h5-index[1] of all periodicals in

[1]From Google Scholar: "h5-index is the h-index for articles published in the last 5 complete years. It is the largest number h such that h articles published in [the last five years] have at least h citations each."

	JCP "New Scheme"	C+F "New Scheme"	C+F "PDE App."
Total in issue	47	22	
Examples of article type	36	11	12
Code Verif., Code Comp.	15	6	4
Code Verif., MES	24	8	3
Code Verif., MMS	3	0	0
Sol. Verif.	0	1	3
Validation	10	5	4
Ref. V&V	0	0	3
No V&V	0	0	1
Misc.	0	3	0

Table 1.1: Counts of Verification and Validation techniques used in recent issues [45, 46] of the Journal of Computational Physics and Computers and Fluids.

the "Computational Mathematics" subcategory of "Physics and Mathematics" [47], and *Computers & Fluids* is the highest-ranked by the same h5-index metric in the "Fluid Mechanics" subcategory with a computational focus, and is fifth overall [48]. In additional to being widely read, these journals contain many examples of two common types of articles that deal with PDE codes: "new PDE scheme papers" and "PDE code application papers," with the JCP issue containing more of the former, and the C+F issue more of the latter. "New PDE scheme papers" are defined as work that present a novel method for solving a PDE numerically, spend most of their text introducing and analyzing the new method, and also usually include the results of applying the scheme to both simple model problems and configurations of practical interest. "PDE code application papers," on the other hand, are more concerned with using an existing method or code to investigate a new physical problem, typically with a perfunctory description of the numerical methods and detailed discussion of the application and results. Of course, some papers do not fit neatly into just one of these two patterns, but most exhibited the qualities of one more strongly than the other.

The results of this survey of the two journal issues is presented in Table 1.1,

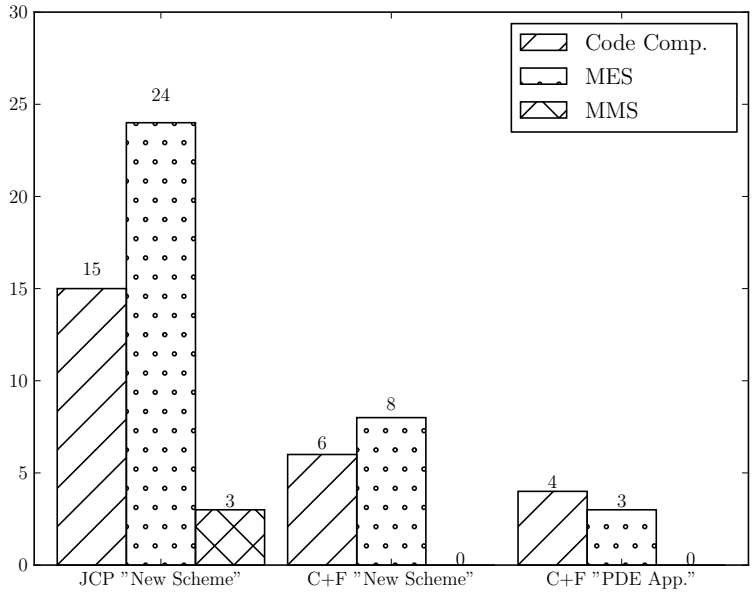


Figure 1-1: Counts of code verification techniques used in recent issues [45, 46] of the *Journal of Computational Physics* and *Computers & Fluids*.

with the code verification results highlighted in Figure 1-1. The categories along the left-hand side of Table 1.1 are defined as follows: the "Code Verif."-prefixed rows "Code Comp.," "MES," and "MMS" indicate code verification was performed through comparison with other codes, the Method of Exact Solutions, and the Method of Manufactured Solutions, respectively. "Sol. Verif." stands for solution verification, and "Ref. V&V" indicates that reference was provided that purported to contain the results of verification and validation exercises. Finally, the "Misc." category was applied to articles that used some verification or validation technique not widely used in the V&V community (two of these articles used the PDE code's fine-grid solution as the reference solution when evaluating the code's error, and one compared linear and nonlinear results to assess accuracy).

The results of the review provide some insight into which V&V techniques the researchers value. Code verification was performed frequently, while solution verifi-

cation was rarely attempted. Validation (i.e., comparison with experiment) was also common, even for the "new scheme" articles, where more emphasis is placed on the numerics than the physics. Most relevant to this work is the breakdown of the three code verification methods: while nearly all of the authors did provide some evidence of the accuracy of their codes, they strongly favored comparison to exact solutions over the more general and rigorous MMS technique. And code comparison, the least rigorous code verification technique, is still used fairly widely.

One can only speculate why the Method of Manufactured Solutions does not see more use in practice. Roache [2, pg. 79] points out that the "nonphysical" quality of manufactured solutions may concern some, but it is likely the extra effort of finding the MMS source term and adding it to the PDE code that lessens its attraction to researchers. The ideal code verification tool would allow the user to create a reference solution consisting of elementary, easy-to-evaluate functions, but would not require any changes to the equations the PDE code solves — the flexibility of MMS with the unobtrusiveness of MES. External Verification Analysis, the subject of this work, attempts to provide such a tool.

Chapter 2

External Verification Analysis

2.1 Opening remarks

As discussed in the Introduction, the purpose of code verification is to rigorously demonstrate that the numerical methods used in a PDE solver have been implemented correctly. This is generally accomplished by showing that the numerical error of the PDE solver converges at the expected rate (i.e., the formal order-of-accuracy of the solver's algorithms) as some "discretization measure" is increased or decreased. Demonstrating that this convergence rate is achieved provides strong evidence that the PDE solver is functioning properly. Evaluating the error, of course, requires a reference solution. The popular approaches to obtaining a reference solution in the V&V field are the Method of Exact Solutions and the Method of Manufactured Solutions. The MMS approach is favored by V&V proponents, but is seldom used by PDE scheme developers, perhaps due to the "implementation overhead" of the MMS source term.

An attractive feature of MMS, however, is its flexibility — the user of the method is able to specify the entire solution to the PDE system up front, at the cost of needing to modify the PDE system with source terms to force the "manufactured" solution to actually *be* a solution of the PDEs. Ideally, one would prefer to avoid modifying the

PDE system (and thus the PDE solver's code), but maintain as much of the flexibility of MMS as possible. Consider a (quite general) PDE of the form

$$F\left(x, \frac{\partial^\alpha u}{\partial x^\alpha}\right) = 0;\ |\alpha| \leq k. \tag{2.1}$$

Here, x and u are the independent and dependent variables of the PDE, respectively, with $x \in \mathbf{R}^m$, $u : \mathbf{R}^m \to \mathbf{R}^n$, $F : \mathbf{R}^m \times \mathbf{R}^n \to \mathbf{R}^n$, k is the order of the PDE, and α is a "multiindex," [49, pp. 3–4] i.e., a collection of non-negative integers whose properties are defined in Appendix A. With MMS, one would specify some function $f(x) : \mathbf{R}^m \to \mathbf{R}^n$ and substitute for u in Equation (2.1),

$$F\left(x, \frac{\partial^\alpha f}{\partial x^\alpha}\right) = G(x) \tag{2.2}$$

where G is the residual or MMS source term that is added to the PDE system. Because $f(x)$ is a known function, all of the derivatives of Equation (2.1) can be evaluated directly, and so the residual term G in Equation (2.2) depends on x only. Thus a solution, $f(x)$, to a slightly modified PDE system has been constructed. If the residual term can be added to the PDE solver's code, then $f(x)$ can be used for verification purposes.

EVA uses a similar but slightly different approach to constructing a solution to Equation (2.1). Instead of specifying the complete solution to the PDE system, Equation (2.1) is recast as a Cauchy [50, p. 243] or initial value problem

$$\frac{\partial^k u}{\partial t^k} = H\left(y, t, \frac{\partial^{\alpha+j} u}{\partial y^\alpha\, \partial t^j}\right);\ |\alpha| + j \leq k,\ j < k. \tag{2.3}$$

subject to the initial condition

$$\left.\frac{\partial^j u}{\partial t^j}\right|_{y,t=\tau} = v_j(y);\ 0 \leq j < k \tag{2.4}$$

at some $t = \tau$. The variable $y \in \mathbf{R}^{m-1}$ represents $m-1$ components of the independent variable x found in Equation (2.1), and $t \in \mathbf{R}$ is the remaining independent variable, e.g.,

$$(x_1, x_2, \ldots, x_m) = (y_1, y_2, \ldots, y_{m-1}, t)\,. \tag{2.5}$$

Notice that the highest-order derivative with respect to t has been isolated in Equation (2.3), which shows that this t derivative is completely determined by lower-order y and t derivatives — thus all of the t derivatives from 0 to k can be found at $t = \tau$ from Equation (2.3) and Equation (2.4). Imagine, however, if the t derivatives from 0 to p, $p > k$ were known. Then an approximation to the solution near $t = \tau$ could be constructed using the truncated Taylor series

$$u\,(y,t) \approx \widetilde{u}(y,t) = u(y,\tau) + (t-\tau)\frac{\partial u(y,\tau)}{\partial t} + \cdots + \frac{(t-\tau)^p}{p!}\frac{\partial^p u}{\partial t^p}. \tag{2.6}$$

If these higher-order t derivatives can be found, and *if* the series in Equation (2.6) converges, then the approximate solution $\widetilde{u}(y,t)$ provides a reference solution that could be used to verify a PDE solver. Using such a Taylor series as a reference solution is the External Verification Analysis [51, 52] (EVA) approach to code verification. The two key questions, then, are

1. Under what conditions will the series in Equation (2.6) converge to the solution to Equation (2.3)?

2. How can the higher-order t derivatives used in Equation (2.6) be found?

The answers to these questions will be the focus of the next few sections. Before proceeding, however, a limitation of using a truncated Taylor series of a Cauchy problem should be noted: since the Cauchy problem is expressed on an infinite domain, there is no obvious mechanism for specifying boundary conditions. Considerable flexibility is available in the form of the initial condition, however.

2.2 The Cauchy-Kowalewski theorem

The answer to question 1 from the preceding section is provided by a theorem of Cauchy and Kowalewski [49, p. 68] or [53, pp. 28–29]:

The Cauchy-Kowalewski Theorem. *There exists a unique, analytic solution for a k-order non-characteristic Cauchy problem, i.e.,*

$$\frac{\partial^k u}{\partial t^k} = H\left(y, t, \frac{\partial^{\alpha+j} u}{\partial y^\alpha \, \partial t^j}\right);\ |\alpha| + j \leq k,\ j < k. \tag{2.3}$$

and

$$\left.\frac{\partial^j u}{\partial t^j}\right|_{y,t=\tau} = v_j(y);\ 0 \leq j < k \tag{2.4}$$

near the point $y = y_0$, $t = \tau$ if H and $\frac{\partial^j u}{\partial t^j}$ are analytic near $y = y_0$, $t = \tau$.

The proof of a unique, analytic solution for Equation (2.3) and Equation (2.4) depends on showing that, for the conditions stipulated in the Cauchy-Kowalewski theorem, the Taylor series of Equation (2.6) (with $p \to \infty$) converges. Thus the theorem provides an answer to the first question of the previous section: the series of Equation (2.6) will converge if the functions found in the PDE (H in Equation (2.3)) and the initial solution (u and its derivatives in Equation (2.4)) are analytic near (y_0, τ).

A few facts about the Cauchy-Kowalewski (or CK) theorem are worth mentioning. First, the CK theorem is a *local* existence and uniqueness theorem — it guarantees that a unique solution exists for the stated conditions only in some neighborhood, not globally. A solution that "starts out" analytic and unique may not stay that way. For instance, initially smooth solutions to, say, the inviscid Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \tag{2.7}$$

tend to steepen into shocks and become multivalued, even for analytic initial conditions $u(x, t = \tau)$. All the CK theorem claims is that, *near* $t = \tau$ (perhaps only very near), the solution will be unique and analytic.

Secondly, the use of the word "analytic" in the theorem is in the "mathematical" sense: a function is analytic at a point if its Taylor series expanded about the point converges in the point's "neighborhood" [54, p. 156]. The CK theorem puts limits on when an analytic solution to a PDE can be expected. And because Equation (2.6) uses a truncated Taylor series to construct a reference solution, satisfying the CK theorem's requirements is essential. Thus each requirement of the CK theorem will be considered in turn.

The requirement of analyticity of the function form of the PDE (i.e., H in Equation (2.3)) is unlikely to be an issue for most PDE's found in physics. Terms with expressions like $1/r$ would clearly be problematic at $r = 0$, but the same would be true for a numerical code that attempts to solve a PDE with such terms. At any rate, the form of the PDE is dictated by the physical situation to be simulated, and is outside the control of a designer of a verification method.

The form of the initial condition used to verify a PDE code *is* within one's control when verifying a PDE code, however. The CK theorem requires that it, too, must be analytic. Because a Taylor series involves derivatives of infinite order evaluated at a particular point, an analytic function must be infinitely differentiable at the expansion point. Thus the initial condition chosen for a verification test case must be infinitely differentiable at each point in the domain.

Finally, the CK theorem is restricted to "non-characteristic" Cauchy problems. A non-characteristic Cauchy problem is one where the initial data (i.e. Equation (2.4) is prescribed on a "non-characteristic surface." In the context of the theory of partial differential equations, a characteristic surface of a p-order PDE is one on which an expression for the p-order derivative in the direction normal to the surface cannot be

determined from PDE itself — in other words, if t is the "marching direction" and u the solution variable, then the derivative $\frac{\partial u}{\partial t}$ vanishes from a first-order PDE on a characteristic surface. As an example, take a two-dimensional quasi-linear PDE

$$a\left(x,y,u\right)\frac{\partial u}{\partial x}+b\left(x,y,u\right)\frac{\partial u}{\partial y}=c\left(x,y,u\right) \tag{2.8}$$

with a coordinate transformation

$$x=x\left(\xi,\eta\right);\ y=y\left(\xi,\eta\right) \tag{2.9}$$

and an initial condition prescribed along a $\xi=\xi_0=$ const. line, i.e.

$$u_0(\eta)=u(x(\xi_0,\eta),y(\xi_0,\eta)). \tag{2.10}$$

After substituting the coordinate transformation Equation (2.9) into Equation (2.8), one finds

$$\left(a\frac{\partial\xi}{\partial x}+b\frac{\partial\xi}{\partial y}\right)\frac{\partial u}{\partial\xi}+\left(a\frac{\partial\eta}{\partial x}+b\frac{\partial\eta}{\partial y}\right)\frac{\partial u}{\partial\eta}=c. \tag{2.11}$$

When is the $\xi=\xi_0=$ const. line (i.e. the line along which the initial condition is set) characteristic? If the line is characteristic, an expression for the derivative normal to the line *cannot* be found from the PDE. After inspecting Equation (2.11), it is seen that this is only true if

$$a\frac{\partial\xi}{\partial x}+b\frac{\partial\xi}{\partial y}=0. \tag{2.12}$$

An interpretation of the physical significance of Equation (2.12) can be found as follows. Imagine a vector field $\vec{v}$ whose components in the x and y directions are given by a and b, the coefficients in Equation (2.8). Also recall that a gradient vector field is everywhere perpendicular to the iso-surfaces (or iso-lines, in this two-dimensional case) of the function from which it is calculated, so $\nabla\xi$ evaluated along the ξ_0 line is

perpendicular to the line itself. Then the condition Equation (2.12) can be expressed as

$$\nabla\xi \cdot \vec{v} = \left(\frac{\partial \xi}{\partial x}\vec{i} + \frac{\partial \xi}{\partial y}\vec{j}\right) \cdot \left(a\vec{i} + b\vec{j}\right) = 0. \tag{2.13}$$

Since the dot product of two vectors is zero when the vectors are perpendicular, Equation (2.13) shows that the ξ_0 line is characteristic only when it is *tangent* to the $\vec{v}$ vector field (since, again, the line's normal is perpendicular to $\vec{v}$).

What significance can be attached to the vector $\vec{v} = a\vec{i} + b\vec{j}$? A discussion found in the second chapter of a work by Garabedian [55] is especially elucidating. Imagine the solution to the PDE of Equation (2.8) as a surface in three dimensions, i.e.

$$z = u(x, y) \tag{2.14}$$

which can be rewritten in an "implicit" form

$$f(x, y, z) = u(x, y) - z = 0. \tag{2.15}$$

The gradient of the "solution surface" f is

$$\nabla f = \frac{\partial u}{\partial x}\vec{i} + \frac{\partial u}{\partial y}\vec{j} - \vec{k}. \tag{2.16}$$

If the dot product of ∇f and the vector field $\vec{V} = a\vec{i} + b\vec{j} + c\vec{k}$ is formed, one finds

$$\begin{aligned} \nabla f \cdot \vec{V} =& a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} - c = 0 \\ & a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} = c \end{aligned} \tag{2.17}$$

which is the original PDE. Equation (2.17) shows that the solution u must be everywhere tangent to the vector field $\vec{V}$, as the solution surface's normal vector ∇f

is perpendicular to $\vec{V}$. Lines that are tangent to $\vec{V}$ are called *characteristic curves*, and indicate how the solution u evolves from its initial state. The projection of these lines on the x-y plane are lines that are tangent to the vector field $\vec{v}$. By placing the initial data along one of these curves, one has prevented the PDE from communicating how the solution will move from the initial condition, and a unique solution is not guaranteed by the Cauchy-Kowalewski theorem.

2.3 Cauchy-Kowalewski recursion

Two questions were posed at the end of Section 2.1. The first was a question of viability — under what conditions could one at least entertain the hope that the Taylor series of Equation (2.6) will converge? The answer was found in the statement of the Cauchy-Kowalewski theorem: when both the initial data and form of the PDE are analytic, and the initial data is specified on a non-characteristic surface. The second was a question of practicality — given a PDE and initial condition, how could one construct such a series? It turns out that this answer also involves the CK theorem, but in a different way: the *proof* of the CK theorem actually constructs a series much like Equation (2.6) for the PDE solution using a recursive technique known as (oddly enough) Cauchy-Kowalewski (CK) recursion. In the proof, the series is shown to converge if the criteria of the CK theorem are met. The EVA approach proposes to use a truncated version of the same series to provide a reference solution for verification — thus accurately and efficiently implementing CK recursion is a crucial requirement of EVA.

2.3.1 Cauchy-Kowalewski recursion with the inviscid Burger's equation

Cauchy-Kowalewski recursion is best explained with an example. Again, consider the inviscid Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0. \tag{2.18}$$

The goal of CK recursion is to express all higher-order derivatives in the marching direction (here, t) as a function of derivatives of the other independent variables (here, just x). The "marching direction" derivatives will be used to calculate the coefficients of the Taylor series of Equation (2.6), and thus the reference solution used for verification. One begins by isolating the t derivative

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} \tag{2.19}$$

and then takes a temporal derivative

$$\frac{\partial^2 u}{\partial t^2} = -\left(\frac{\partial u}{\partial t}\frac{\partial u}{\partial x} + u\frac{\partial^2 u}{\partial x\,\partial t}\right). \tag{2.20}$$

Everything on the right-hand side of Equation (2.20) is known (or can be found) from an analytic initial condition at the initial time level, except for $\frac{\partial^2 u}{\partial x\,\partial t}$. An expression for this last term can be found by taking a spatial derivative of Equation (2.18)

$$\frac{\partial^2 u}{\partial x\,\partial t} = -\left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + u\frac{\partial^2 u}{\partial x^2}\right). \tag{2.21}$$

For the third temporal derivative of u, another derivative with respect to t is taken, this time of Equation (2.20), yielding

$$\frac{\partial^3 u}{\partial t^3} = -\left(\frac{\partial^2 u}{\partial t^2}\frac{\partial u}{\partial x} + \frac{\partial u}{\partial t}\frac{\partial^2 u}{\partial x\,\partial t} + \frac{\partial u}{\partial t}\frac{\partial^2 u}{\partial x\,\partial t} + u\frac{\partial^3 u}{\partial x\,\partial t^2}\right). \tag{2.22}$$

Again, all but one of the terms on the right-hand side are known, except for $\frac{\partial^3 u}{\partial x\,\partial t^2}$, which can be found by differentiating Equation (2.20) with respect to x, i.e.,

$$\frac{\partial^3 u}{\partial x\,\partial t^2} = -\left(\frac{\partial^2 u}{\partial x\,\partial t}\frac{\partial u}{\partial x} + \frac{\partial u}{\partial t}\frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x}\frac{\partial^2 u}{\partial x\,\partial t} + u\frac{\partial^3 u}{\partial x^2\,\partial t}\right). \tag{2.23}$$

Yet again, all but the last term on the right-hand side are known. Taking a spatial derivative of Equation (2.21) will produce

$$\frac{\partial^3 u}{\partial x^2\,\partial t} = -\left(\frac{\partial^2 u}{\partial x^2}\frac{\partial u}{\partial x} + \frac{\partial u}{\partial x}\frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x}\frac{\partial^2 u}{\partial x^2} + u\frac{\partial^3 u}{\partial x^3}\right) \tag{2.24}$$

which completes the CK recursion process for the third temporal derivative. The next step would be to find an expression for $\frac{\partial^4 u}{\partial t^4}$, which will contain new mixed x-t derivatives that must be found by taking x derivatives of previous lower-order derivative expressions.

As the previous example makes clear, Cauchy-Kowalewski recursion becomes tedious when done "by hand," even for a *single* PDE as simple as Equation (2.18). One can imagine the tedium increasing dramatically for more "realistic" equations found in physics, which typically contain more equations and more dependent and independent variables. Luckily, this complexity can be managed quite nicely with a few "tricks." First, the CK recursion process will be expressed algorithmically and then "unwound" to facilitate an efficient implementation in a compute program. Second, an approach to obtaining general expressions for the derivatives of the governing equations of fluid mechanics will be presented.

2.3.2 Cauchy-Kowalewski recursion: the general case

```
procedure CK(deriv_id, deriv_cache)
    if deriv_id in deriv_cache then
        pass
    else if GET_TORDER(deriv_id) == 0 then
        deriv_val ← EVAL_IC(deriv_id, deriv_cache)
        STORE_VAL(deriv_id, deriv_val, deriv_cache)
    else
        expr ← GET_EXPR(deriv_id)
        for all local_deriv_id in expr do
            CK(local_deriv_id, deriv_cache)                ▷ Recursion!
        end for
        deriv_val ← EVAL_EXPR(expr, deriv_cache)
        STORE_VAL(deriv_id, deriv_val, deriv_cache)
    end if
end procedure
```

Algorithm 1 Pseudocode for Cauchy-Kowalewski recursion

The preceding discussion shows that there are essentially two phases to CK recursion: first, an expression for a needed derivative of u is found by differentiating the governing equation; second, the new expression is inspected for more unknown derivatives, which again are found by differentiating the governing equation. This process is described in pseudocode as the recursive procedure CK in Algorithm 1. The procedure takes two arguments: deriv_id, some object that represents the order and dependent variable that should be found (i.e., the programming equivalent of a symbol like $\frac{\partial^5 u}{\partial x^2 \, \partial t^3}$), and deriv_cache, an object that stores the derivatives that have already been found and their numerical values. The routine begins by checking if the value of the derivative deriv_id has been calculated previously and is thus stored inside deriv_cache. If it has, the routine does nothing (**pass**), the rest of the **if** block is skipped, and the program exits the routine.

If deriv_id does need to be found, the CK routine determines if the needed derivative is a pure-spatial derivative (e.g., $\frac{\partial^3 u}{\partial x^3}$) or a mixed spatial-temporal or pure-

temporal derivative (e.g., $\frac{\partial^2 u}{\partial x \, \partial t}$ or $\frac{\partial^4 u}{\partial t^4}$), represented in the pseudocode as the check in the GET_TORDER(deriv_id) == 0 part of the **else if** statement. GET_TORDER is imagined to be a function that takes the deriv_id object and returns the order of its temporal derivative (so GET_TORDER($\frac{\partial^3 u}{\partial x^2 \, \partial t}$) is 1 and GET_TORDER($\frac{\partial^4 u}{\partial x \, \partial t^3}$) is 3). If deriv_id is indeed a pure-spatial derivative, its numerical value is calculated by the EVAL_IC function and then saved in the deriv_cache object by the STORE_VAR routine, and the routine returns.

The last branch of the **if**/**else if**/**else** block handles the case when deriv_id is a mixed spatial-temporal or pure-temporal derivative than has not been calculated previously. First, an expression for the desired derivative is found by differentiating the governing equation and placed in the expr variable, represented in Algorithm 1 by the line containing the GET_EXPR function, which is the pseudocode equivalent of obtaining Equation (2.20) or Equation (2.21) through the differentiation of Equation (2.19). Next, each term in expr is passed to the CK routine, "winding up" the recursion. The recursive CK call has the effect of finding any unknown derivatives in expr and storing them in deriv_cache. Finally, the "unwinding" of the CK procedure is accomplished by the last two lines of the **else** branch, which find the numerical value of the deriv_id derivative using the previously-calculated derivative data in deriv_cache and the EVAL_EXPR routine, and then save the value in the deriv_cache variable, again using the STORE_VAL routine.

Algorithm 1 could be used to easily implement Cauchy-Kowalewski recursion as part of an EVA tool, especially if a computer algebra system (CAS) is available. Indeed, such a program has been developed using the Python programming language [56] and SymPy [57], an open-source Python CAS library. While the flexibility provided by a high-level language like Python and CAS library like SymPy is attractive, such flexibility comes at the cost of efficiency. Also, recursive routines tend to be considerably slower than an explicit loop-based representation. For these reasons,

unwinding the recursion in Algorithm 1 and finding an expression for the arbitrary-order derivative of the targeted governing equations will be discussed in the next section.

2.3.3 Unwinding Cauchy-Kowalewski recursion with the inviscid Burger's equation

Obtaining an expression for any mixed spatial-temporal derivative of Equation (2.18) is simplified by using the Leibniz rule. The Leibniz rule is a generalization of the familiar "product rule" from calculus: it is a formula for the arbitrary-order derivative of the product of two functions:

$$(f \cdot g)^{(n)} = \sum_{k=0}^{n} \binom{n}{k} f^{(k)} g^{(n-k)} \tag{2.25}$$

(The $\binom{n}{k}$ notation stands for the (n, k) binomial coefficient.) Because many (nearly all) of the terms found in the governing equations of fluid mechanics are products of functions, the Leibniz rule is extremely helpful in the CK recursion process. And because the inviscid Burger's equation also contains a product of two functions, it will again be used for demonstration purposes.

To use the Leibniz rule with the inviscid Burger's equation, one again starts with Equation (2.19), and then uses Equation (2.25) to write an expression for all the x-derivatives of Equation (2.19):

$$\frac{\partial^{a+1} u}{\partial x^a \, \partial t} = \sum_{b=0}^{a} \binom{a}{b} \left[-\frac{\partial^b u}{\partial x^b} \frac{\partial^{a-b+1} u}{\partial x^{a-b+1}} \right] \tag{2.26}$$

and then uses the Leibniz rule again to take all the t-derivatives:

$$\frac{\partial^{a+m+1} u}{\partial x^a \, \partial t^{m+1}} = \sum_{n=0}^{m} \binom{m}{n} \sum_{b=0}^{a} \binom{a}{b} \left[-\frac{\partial^{b+n} u}{\partial x^b \, \partial t^n} \frac{\partial^{a-b+1+m-n} u}{\partial x^{a-b+1} \, \partial t^{m-n}} \right]. \tag{2.27}$$

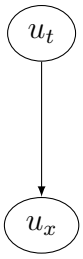


Figure 2-1: CK recursion dependency diagram for Equation (2.30).

Equation (2.27) can be made even more simple with a bit of rewriting,

$$\frac{\partial^{a+m}}{\partial x^a\,\partial t^m}\left(\frac{\partial}{\partial t}u\right) = -\sum_{n=0}^{m}\binom{m}{n}\sum_{b=0}^{a}\binom{a}{b}\left[\frac{\partial^{b+n}}{\partial x^b\,\partial t^n}\left(u\right)\frac{\partial^{a-b+m-n}}{\partial x^{a-b}\,\partial t^{m-n}}\left(\frac{\partial}{\partial x}u\right)\right] \tag{2.28}$$

which shows the connection between it and the original Equation (2.19).

Equation (2.27) and Equation (2.28) are called recurrence relations because they express higher-order temporal derivatives of the inviscid Burger's equation in terms of lower-order derivatives. Understanding how this recursive process works is made easier by expressing Equation (2.19) in the more abstract form

$$\frac{\partial u}{\partial t} = f\left(u, \frac{\partial u}{\partial x},\right) \tag{2.29}$$

Next, consider each term on the right-hand side of Equation (2.29). When building the Taylor series in Equation (2.6), the derivatives will naturally be calculated in increasing order, i.e., for the present example, first u will be found, then $\frac{\partial u}{\partial t}$, $\frac{\partial^2 u}{\partial t^2}$, etc.. If this is so, then a value for u will already be known when attention is turned to calculating $\frac{\partial u}{\partial t}$, but $\frac{\partial u}{\partial x}$ will not. To express this dependency, the notation

$$\frac{\partial u}{\partial t} \rightarrow \frac{\partial u}{\partial x} \tag{2.30}$$

will be used, and is read "to calculate $\frac{\partial u}{\partial t}$, first $\frac{\partial u}{\partial x}$ is needed." Equation (2.30) is expressed graphically in Figure 2-1.

Next, consider the equivalent of Equation (2.29) for $\frac{\partial^2 u}{\partial t^2}$. Since the right-hand side of Equation (2.19) is a product of u and $\frac{\partial u}{\partial x}$, one sees that the correct relationship is

$$\frac{\partial^2 u}{\partial t^2} = f\left(u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x\, \partial t}, \right). \tag{2.31}$$

Compare Equation (2.31) to Equation (2.29). If only the "new" terms of Equation (2.31) are retained (i.e., those that are found in Equation (2.31) but not Equation (2.29)), then the resulting dependency expression for $\frac{\partial^2 u}{\partial t^2}$ is

$$\frac{\partial^2 u}{\partial t^2} \rightarrow \frac{\partial^2 u}{\partial x\, \partial t}, \tag{2.32}$$

which is recognized as the same expression that would have been found by differentiating the sole term in the right-hand side of Equation (2.30) with respect to t. Knowing this, a general dependency relationship for the temporal and mixed spatial-temporal derivatives of the inviscid Burger's equation can quickly be determined by differentiating Equation (2.30) with respect to t^n and x^a:

$$\frac{\partial^{a+n+1} u}{\partial x^a\, \partial t^{n+1}} \rightarrow \frac{\partial^{a+1+n} u}{\partial x^{a+1}\, \partial t^n} \tag{2.33}$$

for $a, n \geq 0$.

Equation (2.33) allows one to create graphical dependency diagrams like Figure 2-1 for higher-order derivatives. For example, Figure 2-2 shows the result of beginning with $\frac{\partial^2 u}{\partial t^2}$ and using Equation (2.33) until only pure-spatial derivatives are left; Figure 2-3 shows the same for $\frac{\partial^3 u}{\partial t^3}$.

Figures 2-1–2-3 are top-down representations of the Cauchy-Kowalewski process — they begin with the desired result ($\frac{\partial^n u}{\partial t^n}$) and end with the input, or starting point ($\frac{\partial^n u}{\partial x^n}$). One would prefer to "unwind" the recursion and instead start from the new pure spatial derivatives needed for a given temporal derivative, then proceed to calcu-

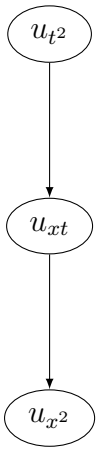


Figure 2-2: CK recursion dependency diagram for $\frac{\partial^2 u}{\partial t^2}$ and its dependents for the inviscid Burger's equation.

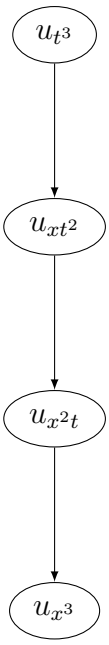


Figure 2-3: CK recursion dependency diagram for $\frac{\partial^3 u}{\partial t^3}$ and its dependents for the inviscid Burger's equation.

late the required mixed spatial-temporal derivatives, then finally arrive at the desired pure-temporal derivative. Clearly this can be achieved by simply reversing the direction of Figures 2-1–2-3.

```
procedure CK_UNWOUND(tot_order, deriv_cache)
    x_order ← tot_order
    for t_order ← 0, tot_order − 1 do
        EVAL_DERIV(x_order, t_order, deriv_cache)
        x_order ← x_order − 1
    end for
    EVAL_DERIV(x_order, t_order, deriv_cache)    ▷ x_order = 0, t_order = tot_order
end procedure
```

Algorithm 2 Pseudocode for unwound Cauchy-Kowalewski recursion with the one-dimensional inviscid Burger's equation.

The unwound CK recursion process for the inviscid Burger's equation is shown in Algorithm 2. Here, the deriv_cache variable is imagined to be some object that holds the numerical values of derivatives calculated previously (identical to its meaning in Algorithm 1), and tot_order is an integer > 0 equal to the order of the pure-temporal derivative that will be calculated by the CK_UNWOUND procedure (e.g., tot_order = 2 for Figure 2-2 and tot_order = 3 for Figure 2-3). x_order and t_order are also integer variables, indicating the spatial and temporal order, respectively, of the derivatives that will be calculated by the EVAL_DERIV routine and stored in the deriv_cache variable — for example, if x_order = 2 and t_order = 1 are passed to EVAL_DERIV, then $\frac{\partial^3 u}{\partial x^2 \, \partial t}$ will be found and saved in deriv_cache.

A pseudocode listing of the EVAL_DERIV routine is shown in Algorithm 3. The outer **if/else** block decides if the desired derivatives are temporal/mixed-spatial-temporal or pure spatial derivatives of the initial condition. The code for the two branches is similar: both contain loops that iterate over the derivatives indicated by the x_order and t_order derivatives (and, therefore, $\frac{\partial^{a+n} u}{\partial x^a \, \partial t^n}$), and both eventually use the STORE_VAL routine to save each iteration's work in the deriv_cache variable. The

difference between the branches lies, of course, in how the derivatives are calculated: if t_order = 0, they are found by differentiating the known initial condition; otherwise the recurrence relation of Equation (2.28) is used.

---

procedure EVAL_DERIV(x_order, t_order, deriv_cache)
  if t_order > 0 **then**
    deriv_val ← EVAL_RECUR_REL(x_order, t_order, deriv_cache)
    STORE_VAL(x_order, t_order, deriv_val, deriv_cache)
  else
    deriv_val ← EVAL_IC(x_order, deriv_cache)
    STORE_VAL(x_order, t_order, deriv_val, deriv_cache)
  end if
end procedure

Algorithm 3 Pseudocode for EVAL_DERIV from Algorithm 2.

After $\frac{\partial^n u}{\partial t^n}$ for $n \in \{0 \dots \text{tot\_order}\}$ have been found from the procedure outlined in Algorithms 2 and 3, the approximate solution $\widetilde{u}(x,t)$ is calculated from the Taylor series indicated by Equation (2.6), i.e.,

$$u\,(x,t) \approx \widetilde{u}(x,t) = u_0(x) + (t-t_0)\frac{\partial u_0(x)}{\partial t} + \frac{1}{2}(t-t_0)^2\frac{\partial^2 u_0(x)}{\partial t^2} \cdots + \frac{(t-t_0)^p}{p!}\frac{\partial^p u_0}{\partial t^p}(x). \tag{2.34}$$

2.3.4 Cauchy-Kowalewski recursion with the Euler equations

The previous section described how Cauchy-Kowalewski recursion is used to construct a Taylor series solution to the inviscid Burger's equation. The present section will use the same technique to construct a Taylor series for a solution to the two-dimensional Euler equations.

The Euler equations are the governing equations of the unsteady motion of an inviscid, compressible fluid. In the field of Computational Fluid Dynamics they are most often expressed in the so-called conservative form (Cf. Anderson [58, pp. 82–88] or Pletcher et al. [59, Chapter Five]), which, in three-dimensional Cartesian coordi-

nates, is

$$\frac{\partial \vec{Q}}{\partial t} + \frac{\partial \vec{E}}{\partial x} + \frac{\partial \vec{F}}{\partial y} + \frac{\partial \vec{G}}{\partial z} = 0 \tag{2.35}$$

where

$$\vec{Q} = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E_{\text{total}} \end{Bmatrix} \tag{2.36}$$

is the vector of conservative flow quantities, and the fluxes $\vec{E}$, $\vec{F}$, and $\vec{G}$ are

$$\vec{E} = \begin{Bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E_{\text{total}} + p) \end{Bmatrix}; \quad \vec{F} = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho uw \\ v(E_{\text{total}} + p) \end{Bmatrix} \quad \vec{G} = \begin{Bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w(E_{\text{total}} + p) \end{Bmatrix}. \tag{2.37}$$

If the perfect gas assumption is used, then the pressure is related to the other flow quantities through the equation

$$E_{\text{total}} = \frac{p}{\gamma - 1} + \frac{1}{2}\rho\left(u^2 + v^2 + w^2\right). \tag{2.38}$$

As will be seen in this section, the implementation of CK recursion is greatly simplified if the primitive variable form of the Euler equations is used. There are in principle many different primitive variable forms of the Euler equations, but perhaps the most common uses the density ρ, the Cartesian velocity components u, v, w and

the pressure p, i.e.,

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} + v\frac{\partial \rho}{\partial y} + w\frac{\partial \rho}{\partial z} + \rho\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = 0 \tag{2.39a}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial x} = 0 \tag{2.39b}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial y} = 0 \tag{2.39c}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial z} = 0 \tag{2.39d}$$

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + w\frac{\partial p}{\partial z} + \gamma p\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = 0. \tag{2.39e}$$

Notice how each term containing a spatial derivative (one with respect to x, y, or z) is a simple product (much like the inviscid Burgers equation Equation (2.18)) *except* for the pressure terms in the momentum equations (the middle three) in Equation (2.39). These terms contain a $1/\rho$ factor, and thus cannot be differentiated by the Leibniz rule Equation (2.25). Ideally, Equation (2.39) would only contain simple products of derivatives. This can actually be accomplished by rewriting the Euler equations in terms of the specific volume $\sigma = 1/\rho$, which yields

$$\frac{\partial \sigma}{\partial t} + u\frac{\partial \sigma}{\partial x} + v\frac{\partial \sigma}{\partial y} + w\frac{\partial \sigma}{\partial z} - \sigma\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = 0 \tag{2.40a}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \sigma\frac{\partial p}{\partial x} = 0 \tag{2.40b}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \sigma\frac{\partial p}{\partial y} = 0 \tag{2.40c}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \sigma\frac{\partial p}{\partial z} = 0 \tag{2.40d}$$

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + w\frac{\partial p}{\partial z} + \gamma p\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = 0. \tag{2.40e}$$

The above equation is refered to as the "primitive variable specific volume" (PVSV) form of the Euler equations in this work.

The CK recursion process proceeds in the same manner as the inviscid Burgers

equation in Section 2.3.1. First, the "marching direction derivatives" are identified and isolated in the governing equation. (As discussed in Section 2.2, this ensures that the chosen initial data surface is non-characteristic.) For the Euler equations, the obvious choice is the time variable t. Then the Leibniz rule is used to find an expression for the temporal derivatives of the flow equations σ, u, v, w, p. For example, applying the CK recursion process to the continuity equation Equation (2.40a) yields

$$\frac{\partial \sigma}{\partial t} = -\left(u\frac{\partial \sigma}{\partial x} + v\frac{\partial \sigma}{\partial y} + w\frac{\partial \sigma}{\partial z} - \sigma\left[\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right]\right) \tag{2.41a}$$

$$\begin{aligned}\frac{\partial^{n+1}\sigma}{\partial t^{n+1}} = -\sum_{m=0}^{n}\binom{n}{m}\Bigg[&\frac{\partial^{n-m}u}{\partial t^{n-m}}\frac{\partial^{1+m}\sigma}{\partial x\,\partial t^m} + \frac{\partial^{n-m}v}{\partial t^{n-m}}\frac{\partial^{1+m}\sigma}{\partial y\,\partial t^m} + \frac{\partial^{n-m}w}{\partial t^{n-m}}\frac{\partial^{1+m}\sigma}{\partial z\,\partial t^m} \\ &- \frac{\partial^{n-m}\sigma}{\partial t^{n-m}}\left(\frac{\partial^{1+m}u}{\partial x\,\partial t^m} + \frac{\partial^{1+m}v}{\partial y\,\partial t^m} + \frac{\partial^{1+m}w}{\partial z\,\partial t^m}\right)\Bigg]\end{aligned} \tag{2.41b}$$

$$\begin{aligned}\frac{\partial^{b+d+f+n+1}\sigma}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[&\\ \frac{\partial^{b-a+d-c+f-e+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}&\frac{\partial^{a+1+c+e+m}\sigma}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+f-e+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}&\frac{\partial^{a+c+1+e+m}\sigma}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+f-e+n-m}w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}&\frac{\partial^{a+c+e+1+m}\sigma}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}} \\ - \frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}&\Bigg(\\ \frac{\partial^{a+1+c+e+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}& \\ + \frac{\partial^{a+c+1+e+m}v}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}& \\ + \frac{\partial^{a+c+e+1+m}w}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\Bigg)\Bigg]&.\end{aligned} \tag{2.41c}$$

Equation (2.41c) is a recurrence relation for all σ derivatives with at least one derivative taken with respect to t. It, combined with corresponding expressions for u, v, w,

and p (found in Appendix C), and initial distributions for $(x, y, z, t = t_0)$, i.e.

$$\sigma(x, y, z, t_0) = \sigma_0(x, y, z) \tag{2.42a}$$

$$u(x, y, z, t_0) = u_0(x, y, z) \tag{2.42b}$$

$$v(x, y, z, t_0) = v_0(x, y, z) \tag{2.42c}$$

$$w(x, y, z, t_0) = w_0(x, y, z) \tag{2.42d}$$

$$p(x, y, z, t_0) = p_0(x, y, z) \tag{2.42e}$$

contains all the information needed to calculate $\frac{\partial^{n+1}\sigma}{\partial t^{n+1}}$ for any $n \geq 0$.

Like the inviscid Burger's equation example in Section 2.3.1, the derivatives in Equation (2.41c) and Equations (C.2)–(C.5) must be evaluated in the correct order to satisfy the dependencies of the recurrence relation (for instance, $\frac{\partial^3\sigma}{\partial t^3}$ depends on $\frac{\partial^3\sigma}{\partial x\,\partial t^2}$, which in turn needs $\frac{\partial^3\sigma}{\partial x^2\,\partial t}$, etc.). Again like the inviscid Burger's equation example, understanding this process is helped by the abstract notation

$$\frac{\partial U}{\partial t} = f\left(U, \frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}, \frac{\partial U}{\partial z}, \right) \tag{2.43}$$

where U are the unknowns σ, u, v, w, p. The spatial coordinates are collapsed into one unknown in the expression

$$\frac{\partial U}{\partial t} = f\left(U, \frac{\partial U}{\partial X}, \right) \tag{2.44}$$

where the notation $\frac{\partial^n(\ )}{\partial X^n}$ stands for any derivative with respect to $x^a y^b z^c$ for all $a + b + c = n$ (so, for example, $\frac{\partial^2 U}{\partial X^2}$ indicates the derivatives $\frac{\partial^2 U}{\partial x^2}$, $\frac{\partial^2 U}{\partial x\,\partial y}$, $\frac{\partial^2 U}{\partial x\,\partial z}$, $\frac{\partial^2 U}{\partial y^2}$, $\frac{\partial^2 U}{\partial y\,\partial z}$, and $\frac{\partial^2 U}{\partial z^2}$). Now a dependency expression equivalent to Equation (2.30) can be written down, i.e.,

$$\frac{\partial U}{\partial t} \to \frac{\partial U}{\partial X}, \tag{2.45}$$

since, again like the inviscid Burger's equation, U is assumed to be known before $\frac{\partial U}{\partial t}$ is calculated. Equation (2.45) can be differentiated with respect to t^n and X^a to find a general dependency relationship between the temporal and mixed spatial-temporal derivatives for the three-dimensional Euler equations:

$$\frac{\partial^{a+n+1} U}{\partial X^a \, \partial t^{n+1}} \rightarrow \frac{\partial^{a+1+n} U}{\partial X^{a+1} \, \partial t^n} \tag{2.46}$$

for $a, n \geq 0$. Clearly, the graphical representation of Equation (2.46) for various values of a and n will be very similar to Figures 2-1–2-3.

---

procedure CK_UNWOUND(tot_order, deriv_cache)
    X_order ← tot_order
    for t_order ← 0, tot_order − 1 **do**
        EVAL_DERIV(X_order, t_order, deriv_cache)
        X_order ← X_order − 1
    end for
    EVAL_DERIV(X_order, t_order, deriv_cache) ▷ X_order = 0, t_order = tot_order
end procedure

Algorithm 4 Pseudocode for unwound Cauchy-Kowalewski recursion with the Euler equations.

Pseudocode for unwinding the Cauchy-Kowalewski recursion for the three-dimensional Euler equations is presented in Algorithm 4. Algorithm 4 is very similar to Algorithm 2, with the only difference being the name of the variable indicating the spatial order of the derivative calculated by the EVAL_DERIV routine. X_order in Algorithm 4 is, like x_order, an integer variable, and, when passed to the EVAL_DERIV routine, is the pseudocode expression of the X^n notation presented earlier — for example, if X_order = 2 and t_order = 1 are passed to EVAL_DERIV, then $\frac{\partial^3 U}{\partial X^2 \, \partial t}$ will be found and saved in deriv_cache.

The key difference between the unwound Cauchy-Kowalewski recursion process for the one-dimensional inviscid Burger's equation and the three-dimensional Euler

```
procedure EVAL_DERIV(X_order, t_order, deriv_cache)
    if t_order > 0 then
        for z_order ← 0, X_order do
            for y_order ← 0, X_order − z_order do
                x_order ← X_order − z_order − y_order
                deriv_val ← EVAL_RECUR_REL(x_order, y_order, z_order, t_order, deriv_cache)
                STORE_VAL(x_order, y_order, z_order, t_order, deriv_val, deriv_cache)
            end for
        end for
    else
        for z_order ← 0, X_order do
            for y_order ← 0, X_order − z_order do
                x_order ← X_order − z_order − y_order
                deriv_val ← EVAL_IC(x_order, y_order, z_order, deriv_cache)
                STORE_VAL(x_order, y_order, z_order, t_order, deriv_val, deriv_cache)
            end for
        end for
    end if
end procedure
```

Algorithm 5 Pseudocode for EVAL_DERIV in Algorithm 4.

equations is in the details of the EVAL_DERIV routine. Algorithm 5 shows the pseudocode for this routine. Again, the construction of this routine is nearly identical to its equivalent for the inviscid Burger's example (Algorithm 3), with the significant difference being the handling of the spatial derivatives. Instead of only using the x_order derivative to represent the order of x-differentiation, a loop is used to iterate over all the $x/y/z$-derivative combinations indicated by X_order.

Finally, after the procedure discussed has been used to find all necessary temporal derivatives, the approximate solution is constructed from the Taylor series indicated by Equation (2.6), i.e.,

$$U(x,y,z,t) \approx \widetilde{U}(x,y,z,t) = U_0(x,y,z) + (t-t_0)\frac{\partial U_0(x,y,z)}{\partial t} + \cdots + \frac{(t-t_0)^p}{p!}\frac{\partial^p U_0}{\partial t^p}(x,y,z). \tag{2.47}$$

2.3.5 Cauchy-Kowalewski recursion with the Navier-Stokes equations

The Navier-Stokes equations govern the motion of a viscous fluid. A form similar to the Euler equations presented in the previous section will be used here, specifically,

$$\frac{\partial \sigma}{\partial t} + u\frac{\partial \sigma}{\partial x} + v\frac{\partial \sigma}{\partial y} + w\frac{\partial \sigma}{\partial z} - \sigma\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = 0 \tag{2.48a}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \sigma\frac{\partial p}{\partial x} = \mu\sigma\left(\frac{4}{3}\frac{\partial^2 u}{\partial x^2} + \frac{1}{3}\frac{\partial^2 v}{\partial x\,\partial y} + \frac{1}{3}\frac{\partial^2 w}{\partial x\,\partial z} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) \tag{2.48b}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \sigma\frac{\partial p}{\partial y} = \mu\sigma\left(\frac{4}{3}\frac{\partial^2 v}{\partial y^2} + \frac{1}{3}\frac{\partial^2 u}{\partial x\,\partial y} + \frac{1}{3}\frac{\partial^2 w}{\partial y\,\partial z} + \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial z^2}\right) \tag{2.48c}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \sigma\frac{\partial p}{\partial z} = \mu\sigma\left(\frac{4}{3}\frac{\partial^2 w}{\partial z^2} + \frac{1}{3}\frac{\partial^2 u}{\partial x\,\partial z} + \frac{1}{3}\frac{\partial^2 v}{\partial y\,\partial z} + \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}\right) \tag{2.48d}$$

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + w\frac{\partial p}{\partial z} + \gamma p\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = (\gamma - 1)\left(\Theta + \Phi\right). \tag{2.48e}$$

where

$$\Theta = \frac{\mathrm{k}}{\mathrm{R}}\left(\frac{\partial^2 \sigma}{\partial x^2}p + 2\frac{\partial \sigma}{\partial x}\frac{\partial p}{\partial x} + \sigma\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 \sigma}{\partial y^2}p + 2\frac{\partial \sigma}{\partial y}\frac{\partial p}{\partial y} + \sigma\frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 \sigma}{\partial z^2}p + 2\frac{\partial \sigma}{\partial z}\frac{\partial p}{\partial z} + \sigma\frac{\partial^2 p}{\partial z^2}\right) \tag{2.49}$$

and

$$\begin{aligned}\Phi = \mu\Bigg(\frac{4}{3}\Bigg[&\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\frac{\partial w}{\partial z} - \frac{\partial u}{\partial x}\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x}\frac{\partial w}{\partial z} - \frac{\partial v}{\partial y}\frac{\partial w}{\partial z}\Bigg] \\ &+ \frac{\partial v}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial w}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial u}{\partial y} + \frac{\partial w}{\partial y}\frac{\partial w}{\partial y} + \frac{\partial u}{\partial z}\frac{\partial u}{\partial z} + \frac{\partial v}{\partial z}\frac{\partial v}{\partial z} \\ &+ 2\left[\frac{\partial v}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial w}{\partial y}\frac{\partial v}{\partial z} + \frac{\partial u}{\partial z}\frac{\partial w}{\partial x}\right]\Bigg).\end{aligned} \tag{2.50}$$

The above equations assume the dynamic viscosity μ, thermal conductivity k, and gas constant R are all constant. Again, replacing the density ρ with $\sigma = 1/\rho$ reduces each term in Equation (2.48) to simple products, which allows the Navier-Stokes equations to be differentiated with nested applications of the Leibniz rule. The recurrence relations for Equations (2.48)–(2.50) are shown in Equation (2.41c) and

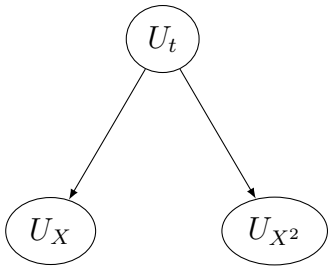


Figure 2-4: CK recursion dependency diagram for the Navier-Stokes $\frac{\partial U}{\partial t}$ and its dependents.

Equations (E.2)–(E.7) in Appendix E.

Because of the viscous terms in Equation (2.48), the dependency expression for $\frac{\partial U}{\partial t}$ (i.e., the equivalent of Equation (2.45)) for the Navier-Stokes equations takes a different form from the PDEs considered previously, namely

$$\frac{\partial U}{\partial t} \rightarrow \frac{\partial U}{\partial X}, \frac{\partial^2 U}{\partial X^2}. \tag{2.51}$$

As in the previous sections, the general dependency relationship can be found by taking a X-derivatives and n t-derivatives of Equation (2.51), giving

$$\frac{\partial^{a+n+1} U}{\partial X^a \, \partial t^{n+1}} \rightarrow \frac{\partial^{a+1+n} U}{\partial X^{a+1} \, \partial t^n}, \frac{\partial^{a+2+n} U}{\partial X^{a+2} \, \partial t^n}. \tag{2.52}$$

Graphical versions of Equation (2.52) for $\frac{\partial U}{\partial t}$, $\frac{\partial^2 U}{\partial t^2}$, and $\frac{\partial^3 U}{\partial t^3}$ are shown in Figures 2-4–2-6. These figures are top-down representations of the Cauchy-Kowalewski process — they begin at the top with the desired result ($\frac{\partial^n U}{\partial t^n}$) and end at the bottom with the input, or starting point ($\frac{\partial^m U}{\partial X^m}$). "Unwinding" this recursion consists of starting from the new pure spatial derivatives needed for a given temporal derivative, then calculating the required mixed spatial-temporal derivatives, then finally arriving at the desired pure-temporal derivative. This can be achieved by simply eliminating all but the right two nodes of each row in Figures 2-4–2-6, and then moving from left-to-right, bottom-to-top with those that remain. After removing the unnecessary nodes,

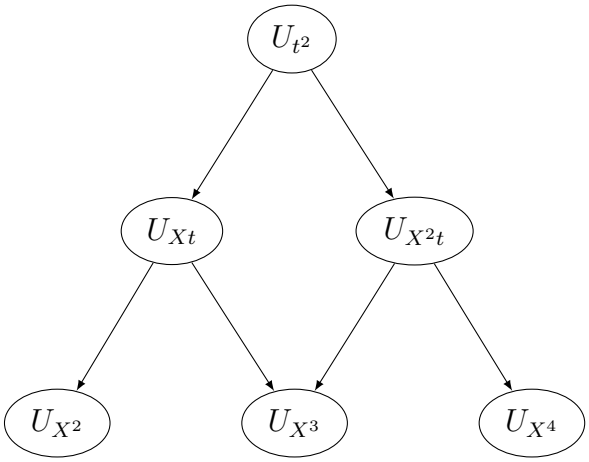


Figure 2-5: CK recursion dependency diagram for the Navier-Stokes $\frac{\partial^2 U}{\partial t^2}$ and its dependents.

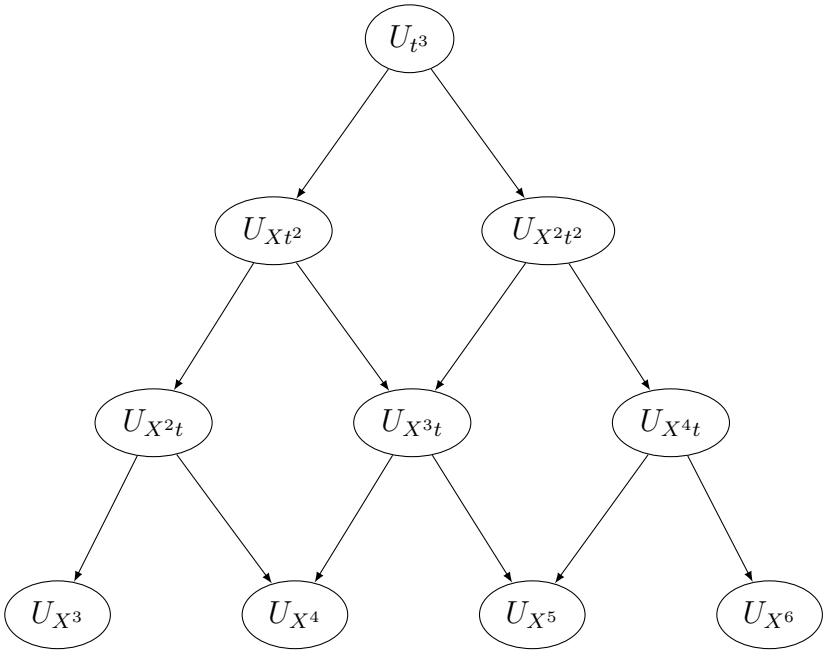


Figure 2-6: CK recursion dependency diagram for the Navier-Stokes $\frac{\partial^3 U}{\partial t^3}$ and its dependents.

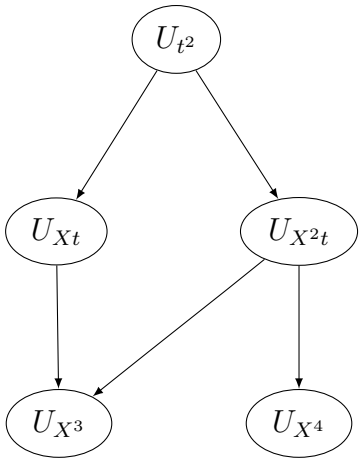


Figure 2-7: CK recursion dependency diagram for the Navier-Stokes $\frac{\partial^2 U}{\partial t^2}$ and its dependents with unnecessary nodes removed.

Figure 2-5 becomes Figure 2-7, and Figure 2-6, Figure 2-8 (Figure 2-4 is unchanged).

---

procedure CK_UNWOUND(tot_order, deriv_cache)
    X_order $\leftarrow 2 \cdot$ tot_order
    for t_order $\leftarrow 0,$ tot_order $- 1$ **do**
        X_order $\leftarrow$ X_order $- 1$
        EVAL_DERIV(X_order, t_order, deriv_cache)
        X_order $\leftarrow$ X_order $+ 1$
        EVAL_DERIV(X_order, t_order, deriv_cache)
        X_order $\leftarrow$ X_order $- 2$
    end for
    EVAL_DERIV(X_order, t_order, deriv_cache) $\triangleright$ X_order $= 0$, t_order $=$ tot_order
end procedure

Algorithm 6 Pseudocode for unwound Cauchy-Kowalewski recursion with the Navier-Stokes equations.

The "left-to-right, bottom-to-top" calculation procedure just described is represented in pseudocode in Algorithm 6. The interpretation of the variables in this algorithm is identical to its Euler equation counterpart (Algorithm 4). Also, the pseudocode for the EVAL_DERIV routine is the same as the one presented for the Euler equation (Algorithm 5). The implementation of the EVAL_RECUR_REL routine for the Navier-Stokes equations would use the recurrence relations in Equation (2.41c)

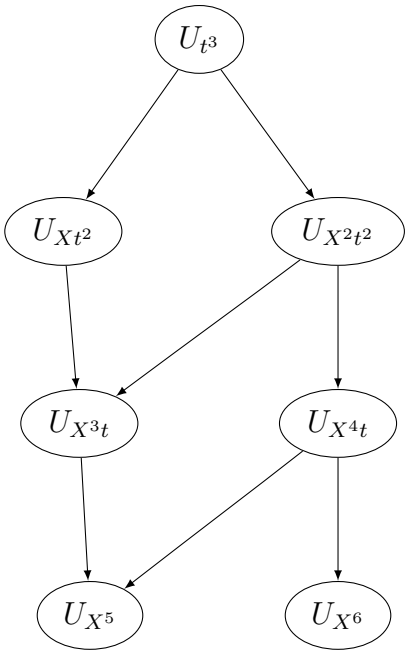


Figure 2-8: CK recursion dependency diagram for the Navier-Stokes $\frac{\partial^3 U}{\partial t^3}$ and its dependents with unnecessary nodes removed.

and Equations (E.2)–(E.7), however.

2.4 The EVA initial condition

As discussed in Section 2.2, the initial condition (IC) used with the EVA technique must be analytic, and thus must be infinitely differentiable at all points in the domain under consideration. Despite this restriction, there is still considerable flexibility in the form the initial condition may take. In addition to analyticity, practical considerations may inform the choice of the initial condition, however. For instance, it may be useful for the initial condition to be "compact" (i.e., for the gradients in the IC to vanish "far away" from some location in the computational domain) to avoid interacting with boundary conditions. One example of such a compact function

would be a multi-dimensional "Gaussian," e.g. for three dimensions,

$$u(x,y,z) = \overline{u} + \widetilde{u} \exp\left(-\frac{\log(2)}{b^2} \left[(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 \right] \right). \tag{2.53}$$

In this work, however, a slightly more general form is used in the current EVA implementation, specifically

$$\begin{aligned} u(x,y,z) = \overline{u} + \widetilde{u} \exp\left(-\frac{\log(2)}{b^2} \left[(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 \right] \right) \\ \sin\left(k_x x + k_y y + k_z z + \phi \right), \end{aligned} \tag{2.54}$$

where all $\overline{u}$, $\widetilde{u}$, b, x_0, y_0, z_0, k_x, k_y, k_z, and ϕ are all constant. Does the initial condition represented by Equation (2.54) satisfy the requirement of the Cauchy-Kowalewski theorem, i.e., analyticity? Because a function consisting of the sum, product, and/or composition of analytic functions is itself analytic, and because the operations and functions in Equation (2.54) meet these conditions, it is indeed analytic.

As shown in Section 2.3, CK recursion — and thus the EVA technique — requires not just the evaluation of an initial condition, but also its derivatives, with the order of the derivatives increasing with the Taylor series order in Equation (2.6). So a method for differentiating Equation (2.54) accurately and efficiently is required. Constructing such a method is made easier if Equation (2.54) is rewritten as follows

$$u(x,y,z) = \overline{u} + \widetilde{u} E(x,y,z) I(x,y,z) \tag{2.55}$$

where

$$E(x,y,z) = \exp\left(-\frac{\log(2)}{b^2} \left[(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 \right] \right) \tag{2.56}$$

and

$$I(x, y, z) = \sin\left(k_x x + k_y y + k_z z + \phi\right). \tag{2.57}$$

The Leibniz rule, Equation (2.25), shows that

$$\begin{aligned}\frac{\partial^{b+d+f} u}{\partial x^b\, \partial y^d\, \partial z^f} = \widetilde{u} \sum_{a=0}^{b} \binom{b}{a} \sum_{c=0}^{d} \binom{d}{c} \sum_{e=0}^{f} \binom{f}{e} \frac{\partial^{b-a+d-c+f-e} E}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}} \frac{\partial^{a+c+e} I}{\partial x^a\, \partial y^c\, \partial z^e} \\ + \begin{cases} \overline{u} & \text{if } b = d = f = 0, \\ 0 & \text{otherwise.} \end{cases}\end{aligned} \tag{2.58}$$

Thus the problem of differentiating Equation (2.54) has been reduced to differentiating $E(x, y, z)$ and $I(x, y, z)$ (the "envelope" and "inner" functions, respectively).

For $E(x, y, z)$, the function itself can be separated into three one-dimensional Gaussian parts, i.e.,

$$\begin{aligned} E(x, y, z) &= \exp\left(-\log(2)\left[\frac{x - x_0}{b}\right]^2\right) \exp\left(-\log(2)\left[\frac{y - y_0}{b}\right]^2\right) \exp\left(-\log(2)\left[\frac{z - z_0}{b}\right]^2\right) \\ &= X(x) Y(y) Z(z), \end{aligned} \tag{2.59}$$

and so the derivative expression is

$$\frac{\partial^{b+d+f} E}{\partial x^b\, \partial y^d\, \partial z^f} = \frac{d^b X}{dx^b} \frac{d^d Y}{dy^d} \frac{d^f Z}{dz^f} \tag{2.60}$$

and all that remains is to find an expression for the derivative of a Gaussian function.

A simple approach would be to differentiate the Gaussian function once,

$$\begin{aligned}
\frac{dX}{dx} &= \frac{d\exp\left(-\log(2)\left[\frac{x-x_0}{b}\right]^2\right)}{dx} \\
&= -\frac{2\log(2)}{b^2}(x-x_0)\exp\left(-\log(2)\left[\frac{x-x_0}{b}\right]^2\right) \\
&= -\frac{2\log(2)}{b^2}(x-x_0)X(x) \\
&= X(x)g(x),
\end{aligned} \tag{2.61}$$

where $g(x) = -2\frac{\log(2)}{b^2}(x-x_0)$, and then use the Leibniz rule to find

$$\begin{aligned}
\frac{d^{n+1}X}{dx^{n+1}} &= \sum_{m=0}^{n}\binom{n}{m}\frac{d^{n-m}X}{dx^{n-m}}\frac{d^m g}{dx^m} \\
&= \sum_{m=0}^{1}\binom{n}{m}\frac{d^{n-m}X}{dx^{n-m}}\frac{d^m g}{dx^m}, \quad \text{since } \frac{d^m g}{dx^m} = 0 \text{ for } m > 1 \\
&= \binom{n}{0}\frac{d^n X}{dx^n}g(x) + \binom{n}{1}\frac{d^{n-1}X}{dx^{n-1}}\frac{dg}{dx} \\
&= \frac{d^n X}{dx^n}g(x) + n\frac{d^{n-1}X}{dx^{n-1}}\frac{dg}{dx} \\
&= -2\frac{\log(2)}{b^2}(x-x_0)\frac{d^n X}{dx^n} - 2n\frac{\log(2)}{b^2}\frac{d^{n-1}X}{dx^{n-1}},
\end{aligned} \tag{2.62}$$

which provides a recurrence relation for the derivatives of $X(x)$.

For $I(x,y,z)$, an explicit expression for its derivatives exists:

$$\begin{aligned}
\frac{\partial^{b+d+f} I}{\partial x^b\,\partial y^d\,\partial z^f} = k_x^b k_y^d k_z^f \Bigg(&\cos\left[(b+d+f)\frac{\pi}{2}\right]\sin\left[k_x x + k_y y + k_z z + \phi\right] \\
&+ \sin\left[(b+d+f)\frac{\pi}{2}\right]\cos\left[k_x x + k_y y + k_z z + \phi\right]\Bigg).
\end{aligned} \tag{2.63}$$

Using Equation (2.63) would require calling functions that evaluate sin and cos each time a derivative of I is needed, which is expensive. For efficiency, then, one would like to minimize the use of trigonometric functions like sin and cos. The factors $\sin\left[k_x x + k_y y + k_z z + \phi\right]$ and $\cos\left[k_x x + k_y y + k_z z + \phi\right]$ in Equation (2.63) are the

same for all b, d, f, and thus can be be evaluated once per x, y, z location and saved for subsequent calculations. The remaining factors $\cos\left[(b+d+f)\,\frac{\pi}{2}\right]$ and $\sin\left[(b+d+f)\,\frac{\pi}{2}\right]$ are, of course, dependent on the order of the I-derivative to be calculated, but can be replaced with the expressions

$$\begin{aligned}\cos\left(n\frac{\pi}{2}\right) &= \mod\ (n+1,2) * \left(1 - 2 \mod\ \left[\frac{n}{2}, 2\right]\right) \\ \sin\left(n\frac{\pi}{2}\right) &= \mod\ (n,2) * \left(1 - 2 \mod\ \left[\frac{n-1}{2}, 2\right]\right).\end{aligned} \tag{2.64}$$

Another inefficient aspect of Equation (2.63) is the leading $k_x^b k_y^d k_z^f$ factor, which will involve many multiplications for large b, d, f. Deriving a recurrence relation allows derivatives of I to be calculated without this excessive multiplication. First, consider the one-dimensional version of $I(x, y, z)$,

$$\begin{aligned}f(x) &= \sin(kx + \phi) \\ &= \sin(\omega)\end{aligned} \tag{2.65}$$

where $\omega = kx + \phi$, and then its derivatives,

$$\begin{aligned}f(x) &= \sin(\omega) \\ \frac{df}{dx} &= k\cos(\omega) \\ \frac{d^2f}{dx^2} &= -k^2\sin(\omega) \\ \frac{d^3f}{dx^3} &= -k^3\cos(\omega) \\ \frac{d^4f}{dx^4} &= k^4\sin(\omega).\end{aligned} \tag{2.66}$$

Inspection of Equation (2.66) shows that the derivatives of $f(x)$ are related by the

recurrence relation

$$\begin{aligned} f &= \sin(\omega) \\ \frac{df}{dx} &= k\cos(\omega) \\ \frac{d^n f}{dx^n} &= -k^2 \frac{d^{n-2} f}{dx^{n-2}}; \quad n \geq 2. \end{aligned} \tag{2.67}$$

Extending Equation (2.67) to the multi-dimensional $I(x, y, z)$ is helped again by writing down its derivatives (with $\varpi = k_x x + k_y y + k_z z + \phi$):

$$\begin{aligned} I &= \sin(\varpi) \\ \frac{\partial I}{\partial x} &= -k_x \cos(\varpi) \\ \frac{\partial I}{\partial y} &= -k_y \cos(\varpi) \\ \frac{\partial I}{\partial z} &= -k_z \cos(\varpi) \\ \frac{\partial^2 I}{\partial x^2} &= -k_x^2 \sin(\varpi) \\ \frac{\partial^2 I}{\partial x\, \partial y} &= -k_x k_y \sin(\varpi) \\ \frac{\partial^2 I}{\partial y^2} &= -k_y^2 \sin(\varpi) \\ \frac{\partial^2 I}{\partial x\, \partial z} &= -k_x k_z \sin(\varpi) \\ \frac{\partial^2 I}{\partial y\, \partial z} &= -k_y k_z \sin(\varpi) \\ \frac{\partial^2 I}{\partial z^2} &= -k_z^2 \sin(\varpi) \end{aligned} \tag{2.68}$$

which suggests the recurrence relation

$$\frac{\partial^\alpha I}{\partial X^\alpha} = -K^{\alpha-\beta} \frac{\partial^\beta I}{\partial X^\beta}; \text{ where } |\alpha| \geq 2,\ |\beta| = |\alpha| - 2, \tag{2.69}$$

with α and β being multiindices[1] with lengths of three (for the three spatial dimensions of $I(x, y, z)$), $K = (k_x, k_y, k_z)$, and $X = (x, y, z)$. For example, to find $\frac{\partial^9 I}{\partial x^2 \partial y^3 \partial z^4}$ using Equation (2.69), set $\alpha = (2, 3, 4)$, choose $\beta = (1, 2, 4)$, leading to $\alpha - \beta = (1, 1, 0)$ and then $\frac{\partial^9 I}{\partial x^2 \partial y^3 \partial z^4} = -k_x k_y \frac{\partial^7 I}{\partial x \partial y^2 \partial z^4}$. Notice that Equation (2.69)'s condition on $|\beta|$ does not yield a unique value for β, e.g., $\frac{\partial^9 I}{\partial x^2 \partial y^3 \partial z^4}$ could have been found with, say, $\beta = (0, 3, 4)$ or $(2, 2, 3)$. Also, unlike Equation (2.63), the amount of work needed to evaluate Equation (2.69) does not increase with the order of the derivative — the cost is always one sign change and three multiplications.

It must be emphasized that all the expressions for the derivatives of the initial condition Equation (2.54) are exact, i.e., they result in the same expressions that one would find when differentiating Equation (2.54) by hand with pencil and paper. There are two benefits associated with such an approach. First, using exact expressions obviously reduces the error associated with the spatial derivatives in the EVA solution: only error associated with floating-point operations will be present. Second, and perhaps more significantly, the exact differentiation avoids using numerical methods (finite difference, volume, element, etc.) that would create a dependency between the solution at adjacent spatial locations in the problem domain, and whose accuracy would be affected by the qualtity of the grid or mesh used to represent the solution domain. The accuracy of the EVA method is completely unaffected by the type (structured, unstructured, multiblock, etc.) or quality (highly skewed, stretched, etc.) of the grid used.

[1] See Appendix A for properties of the multiindex

Chapter 3

Practical considerations

The previous chapter discussed the mathematical foundations of the EVA approach to constructing a reference solution for code verification. Here, some of the practical aspects of how to implement the EVA technique in a computer program will be presented.

3.1 Convergence criteria for the Taylor series

In order for Equation (2.6) to be useful as the "reference solution" in a code verification exercise, the truncation error of its Taylor series must be lower than the discretization error of the PDE code to be verified. Ideally, the order of the Taylor series in Equation (2.6) should be high enough to keep the truncation error low — but the calculation of each successive term in Equation (2.6) becomes more and more computationally demanding for all but the simplest governing equations. An estimation of the Taylor series truncation error would be useful in deciding how many terms are needed.

One method of estimating the Taylor series truncation error is to equate it with the magnitude of the last-calculated Taylor series term. This approach could be problematic in some situations, however; for instance, the Taylor series may have alternating terms that are identically zero, similar to $\sin(x)$ or $\cos(x)$. For this reason,

the EVA tool as currently constructed estimates the Taylor series truncation error as the sum of the magnitude of the last *two* Taylor series terms. If this sum is greater than some user-specified tolerance, another term is calculated, the error is estimated again, and so on until the tolerance is satisfied. The EVA tool also allows the user to specify a minimum and maximum Taylor series order, which forces the code to construct a solution in between the order limits (inclusively).

It should be mentioned that truncation error is not the only error to be concerned with in the EVA solution. Floating point "round-off" error will also exist, and may be significant if some of the Taylor series coefficients are large in magnitude and opposite in sign. Outside of reducing the step size Δt, little can be done to reduce this effect (although a poorly-converging EVA solution may indicate the initial condition used is not very smooth, or "difficult" in some other way).

3.2 Parallelization opportunities

As described in Section 2.4, the EVA method evaluates spatial derivatives using exact analytic expressions (or equivalent recurrence relations), instead of an approximate numerical procedure like the finite difference, finite volume, finite element, etc. methods commonly used in simulation codes. The aforementioned numerical techniques create dependencies on the solution between spatial locations, and are thus barriers to parallelization. Because EVA uses analytic expressions for these derivatives, there are no such dependencies, and the solution at each spatial location can be calculated independently. The EVA method is thus *embarrassingly parallel*, and extremely amenable to parallelization.

An argument can be made that parallelizing the EVA tool is unnecessary, since only one EVA solution is required to verify a PDE code and the solution itself is only needed at a few locations in the domain to check the convergence rate of the error.

Especially for low-order (i.e., less accurate) numerical schemes, and when verifying spatial differencing schemes, this may be true; however, high-order and/or highly-optimized time marching schemes require very accurate solutions at a time level "far away" from the initial condition, which increases the accuracy demands on the EVA solution. A more accurate EVA solution means a higher-order Taylor series, which in turn leads to increased computational cost and longer EVA run times. Parallelization helps manage these costs.

Also, some time marching schemes have a "starting problem" — they require the solution from previous time steps to advance the solution to the next time step, which is not available at the start of the calculation. Typically a different time marching scheme (one without a starting problem) is used for the first few steps until enough data is available for the primary time marching scheme to be used. But when verifying such a code, this "mixing" of schemes would contaminate the results — one would likely end up verifying the "starting" scheme instead of the main scheme (which may be useful in itself, but not what is intended). In such a situation, the EVA tool can be used with a *negative* time step to calculate the solution at the previous time levels that are required for the scheme, and thus overcome the starting problem. The solution at the previous time level would be needed at *every point* in the PDE code's grid/mesh, however — thus the computational cost of the EVA tool cannot be mitigated by reducing the number of EVA solution spatial locations, and the effort of parallelizing the EVA technique is perhaps warranted.

3.3 The code verification process

As discussed in Chapter (1), the goal of code verification is to rigorously demonstrate that a PDE code's numerical schemes are implemented properly and behaving as expected. This is done by showing that the PDE code's error converges at the

expected rate, i.e. at the schemes' formal order-of-accuracy. The previous sections in this chapter have focused on the development of the EVA tool, which provides the "reference" or "exact" solution used to calculate the error of the PDE code. But obtaining a reference solution is only one step in the code verification process. Those steps can be summarized as follows:

1. Specify test case parameters (grid, initial solution, etc.),
2. Run the EVA tool to obtain a reference solution with the parameters from 1,
3. Run the PDE code with the parameters from 1 and a range of "discretization measures" (grid spacings or time step sizes),
4. Calculate the error between the PDE code solutions from 3 and the EVA solution from 2,
5. Observe the rate the error found in 4 goes to zero, and compare it to the formal order-of-accuracy of the PDE code's numerical scheme(s).

Each of these steps will be discussed in more detail in the proceeding sections.

3.3.1 Step 1: Choosing verification test case parameters

Before running EVA or a PDE code with the goal of verification in mind, one must obviously decide what the verification test case will look like. The form of the initial solution and the grid/mesh to be used with the PDE code are the two most important decisions that must be made. Most V&V researchers strongly recommend using as general of a test case as possible. If the PDE code supports non-uniform grids or non-constant properties, then the test case should exercise these capabilities in order to assess their performance. However, it is often helpful to "start small" — beginning with a simplified test case (e.g., uniform grid, constant properties, a

simple initial solution) allows one to more easily reason about what kind of solution to expect from the PDE code, and perhaps to isolate terms in the governing equation, or portions of the code. For instance, while the EVA tool allows the user to set an initial flow for the Euler equations with disturbances in each of the flow variables, using a flow with, say, only variations in pressure isolate portions of the PDE code's algorithms and make the source of problems more clear. For the same reason it may be wise to run a uniform grid/mesh test case before trying as general of a geometry as the PDE code is capable of. On the other hand, choosing an initial condition that balances the magnitude of the various terms in the governing equation will ensure that the errors from code mistakes aren't overwhelmed by other (hopefully correct) portions of the solution.

In principle, little knowledge of the details of the PDE code's numerical methods is required for code verification (in fact, having non-code-developers perform verification exercises may be useful for confidence building). The formal order-of-accuracy of the PDE code's schemes must be known for a convincing verification, of course. In practice, understanding the behavior of the PDE code's schemes has been found to be immensely helpful, both for deciding on test case parameters, and in interpreting the verification results (i.e., error and convergence rate of the schemes). For example, many numerical schemes are stable only for a limited range of grid spacings or time step sizes. If the parameters used for verification are outside this range, then bad results will likely follow. Ideally the range of discretization measures used in the test case should capture the entire "behavior" of the scheme — from the onset of instability of the large spacing runs to the (hopefully) asymptotic convergence rate of small spacing runs.

3.3.2 Step 2: Running EVA

After setting appropriate test case parameters, the EVA tool is used to construct a reference solution. Again, because EVA evaluates the spatial derivatives of the initial solution analytically, the format or structure of the grid or mesh is not important — a list of spatial locations is all that is required. (Currently EVA reads and writes Plot3D files, a structured-grid CFD file format, but only for reasons of convenience). For the same reason, what would be a "bad" grid for most PDE codes (i.e. one with strongly skewed cells, large grid spacing growth rates, singularities if structured, extremely coarse spacings) have no effect on the accuracy of the EVA solution.

The domain of the EVA solution does not need to extend over the entire grid used with the PDE code. A grid consisting of a single point could be used for verification. The PDE solution can only be checked where the EVA solution is available, however, so using sparse grids with the EVA tool increases the chance that something interesting or important will be missed in the verification exercise. Reducing the number of EVA calculation points does lower the computational cost of the EVA tool.

It must be emphasized that the EVA technique presented in this work does not allow for the enforcement of any sort of boundary condition — the EVA solution is calculated on an "infinite" domain. PDE codes, however, must truncate an infinite physical space at some point. "Non-reflective" or "transparent" boundary conditions used for this purpose are rarely perfect, and may introduce errors that will spoil the quality of the PDE code's solution within the interior of the domain. A "smaller" EVA grid (i.e., one whose boundaries are well within the boundaries of the PDE code's grid) can help avoid this situation, and is recommended.

Clearly, the time step size should be chosen to match the final time level of the PDE code runs in step 3. The size of the time step is one of the parameters that has a significant effect on the accuracy of the EVA solution (the others being the order of

the Taylor series in Equation (2.6) and the amount of floating-point round-off error). Nearly all of the effort in computing each Taylor series term, i.e.,

$$\frac{(t-\tau)^p}{p!}\frac{\partial^p u}{\partial t^p}, \tag{3.1}$$

is tied up in evaluating $\frac{\partial^p u}{\partial t^p}$. Thus it may be helpful to compute an EVA solution for a range of final time levels (i.e., multiple values for t) "all at once" (i.e., during one run of the EVA tool). The error in the EVA solution will tend to increase with the (magnitude) of t, and too-large values of t could lead to severe round-off error if successive values of $\frac{\partial^p u}{\partial t^p}$ alternate in sign.

3.3.3 Step 3: Running the PDE code

The details of running the PDE code will vary with each code. What is important is that the PDE code is able to begin the calculation with the initial solution specified in step 1 and used with the EVA tool in step 2, and that this initial solution is evolved to the same finial time level as that of the EVA solution. The latter requirement means that the PDE code must allow the user to specify a constant time step size, instead of setting a constant CFL number and then changing the time step size as the calculation progresses.

One will almost always wish to run the PDE code with the highest precision floating-point calculation available. This will probably mean what corresponds to `double` in the C family of programming languages or `DOUBLE PRECISION` in Fortran, but perhaps `long double` or quadruple-precision. The higher precision will reduce the amount of floating-point round-off error in the error calculation in step 4, and thus extend the "error floor," i.e., the smallest possible error that can be calculated with the floating-point arithmetic of the computer.

When verifying a PDE code's spatial differencing schemes, one wishes to observe

the behavior of the error as the spacing between grid/mesh points is reduced. This is done by obtaining a solution to the problem defined in step 1 with the PDE code on a series of progressively finer/coarser grids. It is crucial that the initial coarse grid is refined *uniformly*, i.e., in a similar manner everywhere in the computational domain (see Oberkampf and Roy [5, pp. 185–192] and Veluri et al. [18] for especially clear examples of uniform grid refinement, including unstructured meshes). While it is theoretically possible that grid refinement may only be necessary in regions where the solution is unresolved (and thus where the error is greatest), identifying such regions can be difficult. This is especially true when working with high-order, optimized, or other highly-accurate methods, where the error is expected to be very low to begin with.

Uniform grid refinement does not necessarily require, say, halving the grid spacing or mesh size from one grid to the next in the series (e.g., for a two-dimensional grid with 11^2 points, increasing the number of points like 21^2, 41^2, 81^2, 161^2,...). Indeed, such aggressive refinement may be detrimental to the verification results, since some of the convergence rate behavior of the schemes may be missed. An approach that has proved useful is to divide the spacing of the coarsest grid by an progressively larger integer (e.g., for the previous example, using 21^2, 31^2, 41^2, 51^2,... points). Such a grid series decreases the spacing more slowly, increasing the chances that the asymptotic range of the PDE code's spatial differencing scheme will be found before machine error becomes a problem (hopefully before having to run the extremely fine grids in the first approach).

The same requirements for constructing the grid series used for verifying spatial differencing methods apply to the range of time step sizes used to verify a PDE code's time-marching scheme. Since the final time level of the PDE code's solution should match EVA's, any automatic adjustment of the time step size done by the code (for, say, accuracy or stability constraints) should be disabled.

Some PDE codes allow for different time step sizes to be used in different regions of the solution domain. This poses no problem for the EVA method, as long as the solution everywhere eventually reaches the same time level as the EVA tool. While the same time step size does not need to be used everywhere in the domain, it does need to be refined at the same rate from one PDE code run to the next during the temporal scheme verification process. Also, the same advice for "slow grid refinement" in spatial differencing verification applies to reducing the time step size in temporal integration verification: best not to reduce the time step size too quickly, which risks missing the asymptotic range.

3.3.4 Step 4: Calculating the error

Once an EVA reference solution and a collection of PDE solutions with a range of discretization measures are obtained, the error between the two must be calculated. Any appropriate error norm may be used. The popular error norms are the l_1

$$l_1 = \frac{1}{N} \sum_{n=1}^{N} \left| \phi_n^{(i)} - \Phi_n \right| \tag{3.2}$$

the l_2, or root-mean square,

$$l_2 = \sqrt{\frac{1}{N} \sum_{n=1}^{N} \left(\phi_n^{(i)} - \Phi_n \right)^2} \tag{3.3}$$

and the $l_{\max}$

$$l_{\max} = \max_n \left| \phi_n^{(i)} - \Phi_n \right| . \tag{3.4}$$

The notation Φ and $\phi^{(i)}$ indicate the EVA and i-th PDE code solutions, respectively, with the subscripts representing the solution at each of the N spatial locations. While any of the three error norms "work" for code verification, the $l_{\max}$ tends to be a bit more "volatile" — the averaging of the l_1 and l_2 "smooth out" results. Of course, the

error can only be evaluated at locations where both the EVA and PDE code solutions are available. As mentioned in the discussion of step 2, it may be beneficial to keep the EVA solutions points away from the boundaries of the PDE code's domain, in order to avoid issues with imperfect non-reflecting boundary conditions, or periodic boundaries.

3.3.5 Step 5: Calculating the error convergence rate

The error convergence rate (sometimes referred to as the observed order-of-accuracy) is the rate at which the PDE code's error goes to zero as a "discretization measure" (grid spacing or time step size) is decreased. This convergence rate is used to estimate the order of the PDE code's scheme's truncation error. Again, if the error convergence rate matches the formal order-of-accuracy of the scheme under consideration, then one has provided strong evidence that the scheme has been implemented properly.

An expression for the error convergence rate can be found by first assuming the PDE code's solution matches the first p terms of the exact solution's Taylor series, i.e.,

$$\phi_i = \Phi + Ah_i^p + Ch_i^{p+1} + \dots \tag{3.5}$$

Here, h_i is the discretization measure (i.e., grid spacing for spatial discretization schemes, time step size for time marching schemes) used with the i-th PDE code run, and A and C are constants independent of h. If the solution is well-resolved (i.e., the discretization measure is "small enough" for the numerical schemes to adequately represent the solution) the higher-order terms will be negligible in size, and the error of the i-th PDE code's solution will be

$$\epsilon_i = \phi_i - \Phi \approx Ah_i^p. \tag{3.6}$$

To find the convergence rate p, the ratio of the error of two PDE code solutions is

formed

$$\frac{\epsilon_i}{\epsilon_{i-1}} \approx \frac{Ah_i^p}{Ah_{i-1}^p} = \frac{h_i^p}{h_{i-1}^p} = \left(\frac{h_i}{h_{i-1}}\right)^p \tag{3.7}$$

and then rearranged to isolate p,

$$p \approx \frac{\log\left(\frac{\epsilon_i}{\epsilon_{i-1}}\right)}{\log\left(\frac{h_i}{h_{i-1}}\right)} = \frac{\log(\epsilon_i) - \log(\epsilon_{i-1})}{\log(h_i) - \log(h_{i-1})}. \tag{3.8}$$

The second form of p in Equation (3.8) shows the connection between the convergence rate of the error and the error itself: the former is the slope of the latter on a log-log plot.

In Equation (3.6), the numerical error is assumed to be dominated by one term (Ah_i^p). For unsteady PDE codes, however, the numerical error is taken to be composed of two parts: one due to the spatial differencing scheme, and the other, the time-marching scheme. Thus Equation (3.6) could be rewritten as

$$\epsilon_{i,j} \approx Ah_i^p + B\Delta t_j^q \tag{3.9}$$

where h_i is some measure of the grid spacing, Δt_j is the time step size, and $\epsilon_{i,j}$ is some error norm of the output of a PDE code using h_i and Δt_j. In this work, the spatial and temporal error are investigated separately, so the time step size is held constant when verifying a spatial differencing scheme, and vice versa. Thus Equation (3.9) could be further simplified to

$$\epsilon_i \approx Ah_i^p + B \tag{3.10}$$

where h_i is now either a measure of the grid spacing or time step size for spatial or temporal verification, respectively. Equation (3.10), then, contains three unknowns (A, B, and p), and each ϵ-h combination is associated with one PDE code calculation — thus error norm data from three runs will be needed to solve for the unknowns in

Equation (3.10), i.e.,

$$\begin{aligned}\epsilon_0 &= Ah_0^p + B \\ \epsilon_1 &= Ah_1^p + B \\ \epsilon_2 &= Ah_2^p + B,\end{aligned} \tag{3.11}$$

which is a non-linear equation with no explicit solution. Following Oberkampf and Roy [5], the unknowns A and B may be eliminated with a bit of manipulation, yielding

$$\frac{h_0^p - h_1^p}{h_1^p - h_2^p} = \frac{\epsilon_0 - \epsilon_1}{\epsilon_1 - \epsilon_2}, \tag{3.12}$$

one equation for the unknown p. Unless (as in Ref. [5]) $h_i = r^i h_0$, Equation (3.12) cannot be solved explicitly and a root-finding algorithm must be used to find p — the approach taken here.

3.4 Verification of the EVA code

To verify the implementation of the EVA method used in this work, the algorithms described in the previous chapter were programmed in the Python language [56]. The computer algebra system library SymPy [57] was used to evaluate all derivatives, and the required Cauchy-Kowalewski recursion was programmed directly, not unwound as described previously (in other words, Algorithm 1 was used, and not, say, Algorithm 4). The use of SymPy and the fully-recursive CK approach allows the Python version of EVA to accept PDEs and initial conditions as *inputs*, and is thus not limited to the Euler or Navier-Stokes equations, or initial conditions like Equation (2.54). A listing of the Python code is found in Appendix F.

To verify the EVA tool, the Python code was used to obtain a solution to the conservative form of the three-dimensional Euler or Navier-Stokes equations, while

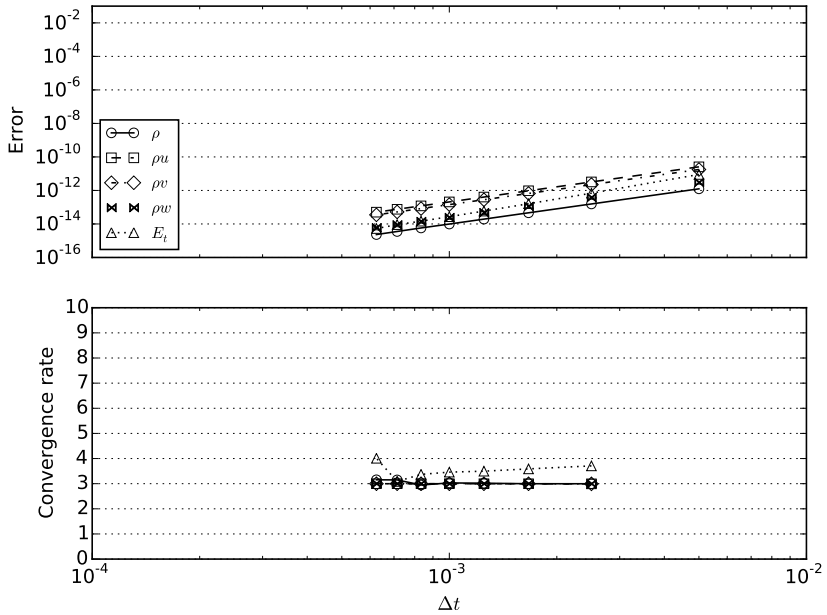


Figure 3-1: EVA verification results for the 3D Euler equations.

the EVA tool solved the corresponding equations in PVSV form (Equation (2.40) or Equation (2.48)), and then converted the resulting solution to the conservative variables. A second-order Taylor series was used with both codes, so the truncation error was expected to be on the order of Δt^3. Figures 3-1 and 3-2 show the error and convergence rate curves for the Euler and Navier-Stokes equations, respectively. The error between the two codes does indeed converge at approximately a third-order rate. A higher-order Taylor series would have perhaps provided more convincing verification results, but the relative inefficiency of the Python code prevented such a case from being run (the Navier-Stokes version took over 19 hours of walltime per point).

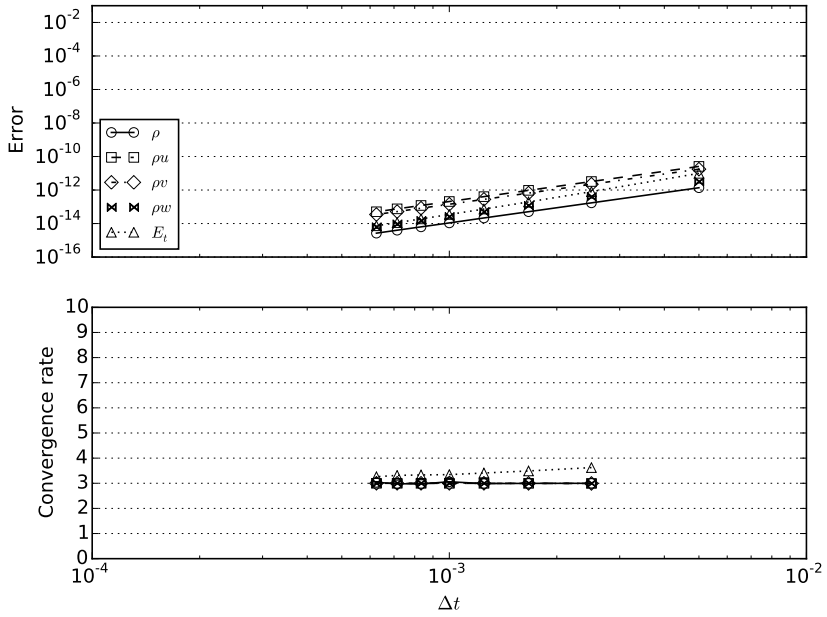


Figure 3-2: EVA verification results for the 3D Navier-Stokes equations.

Chapter 4

Applications of External Verification Analysis

4.1 The PDE code: BASS

The PDE code verified in this work is NASA Glenn's Broadband Aeroacoustic Stator Simulator [60] (BASS), a high-order block-structured finite-difference Computational Aeroacoustics solver. BASS is capable of working with either the Euler or Navier-Stokes equations in two or three dimensions — verification results of the three-dimensional form will be presented here.

BASS combines high-order and/or optimized finite difference approximations of spatial derivatives with high-order, optimized Runge-Kutta integration schemes to march the solution in time. To see how this is done, first consider a generic one-dimensional conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \tag{4.1}$$

where u is the dependent variable, and $f(u)$ the flux, which could be, say, $f = cu$ for the linear advection equation or $f = \frac{u^2}{2}$ for inviscid Burger's equation (2.18). After

isolating the temporal derivative,

$$\frac{\partial u}{\partial t} = -\frac{\partial f(u)}{\partial x} \tag{4.2}$$

the spatial derivative is replaced with a finite difference approximation $D_x(f)$

$$\frac{\partial u}{\partial t} = -D_x(f). \tag{4.3}$$

Finite difference methods are algebraic representations of a function's derivative on a discretized domain. A simple example of a finite-differencing scheme is the first-order approximation

$$D_x(f(x,t)) = \frac{f(x,t) - f(x - \Delta x, t)}{\Delta x} \tag{4.4}$$

where Δx is the grid spacing, i.e., the distance between successive grid points. With this replacement, the PDE in Equation (4.1) has effectively been reduced to an ordinary differential equation (ODE), as $\frac{\partial u}{\partial t}$ is the only remaining derivative. Thus numerical integration techniques designed to solve ODEs may be used to advance the solution u from an initial time t_0 to $t_1 = t_0 + \Delta t$, where Δt is referred to as the time step size. For example, Euler's first-order explicit method applied to Equation (4.3) would be

$$u(x, t_0 + \Delta t) = u(x, t_0) + \Delta t \left.\frac{\partial u}{\partial t}\right|_{x,t_0} \tag{4.5}$$

where $\left.\frac{\partial u}{\partial t}\right|_{x,t_0}$ would be calculating using Equation (4.3) and Equation (4.4) (or any other suitable finite difference scheme).

The spatial differencing scheme (4.4) and time-marching scheme (4.5) are called first-order methods, since the errors associated with them are proportional to Δx and Δt, respectively. More accurate methods exist. One approach to increasing the accuracy of finite differencing schemes is to use the solution from more locations in the domain. For example, the second-order central scheme involves the solution from

$x - \Delta x$ to $x + \Delta x$:

$$D_x(f(x,t)) = \frac{f(x+\Delta x,t) - f(x-\Delta x,t)}{2\Delta x} \tag{4.6}$$

In the field of Computational Aeroacoustics, even wider finite difference stencils are used — for example, the sixth-order central differencing scheme requires data from $x - 3\Delta x$ to $x + 3\Delta x$.

A related procedure for obtaining accurate finite-difference approximations is to use an implicit definition such as

$$\frac{1}{3}D_x(f(x-\Delta x,t)) + D_x(f(x,t)) + \frac{1}{3}D_x(f(x+\Delta x,t)) = \frac{1}{9}\frac{f(x+2\Delta x,t) - f(x-2\Delta x,t)}{4\Delta x} + \frac{14}{9}\frac{f(x+\Delta x,t)) - f(x-\Delta x,t)}{2\Delta x}, \tag{4.7}$$

a sixth-order scheme. Since $D_x(f(x,t))$ depends on $D_x(f(x \pm \Delta x,t))$ in Equation (4.7), there is no longer an explicit relationship between f and its spatial derivative approximation $D_x(f)$, and a system of linear algebraic equations must be solved to find $D_x(f)$ at each x location. Such schemes are called "compact" since they use data from a narrower range of x in their definition — for example, Equation (4.7) uses data from $x - 2\Delta x$ to $x + 2\Delta x$, compared to $x - 3\Delta x$ to $x + 3\Delta x$ for the explicit sixth-order central scheme mentioned previously. Lele [61] provides an excellent discussion of compact approximations to first- and second-order derivatives.

Time-marching schemes, like spatial differencing schemes, can also be improved by using more data in the vicinity of the location the solution is to be calculated. For Runge-Kutta-style schemes, this is done by using multiple stages to advance the solution a Δt forward in time. Perhaps the most well-known Runge-Kutta scheme is

the fourth-order approximation (Cf. [62, p. 98])

$$
\begin{aligned}
u(x, t_0 + \Delta t) &= u(x, t_0) + \frac{\Delta t}{6}\left(\frac{\partial u_1}{\partial t} + 2\frac{\partial u_2}{\partial t} + 2\frac{\partial u_3}{\partial t} + \frac{\partial u_4}{\partial t}\right) \\
\frac{\partial u_1}{\partial t} &= \frac{\partial}{\partial t}\left[u(x, t_0)\right] \\
\frac{\partial u_2}{\partial t} &= \frac{\partial}{\partial t}\left[u(x, t_0) + \frac{\Delta t}{2}\frac{\partial u_1}{\partial t}\right] \\
\frac{\partial u_3}{\partial t} &= \frac{\partial}{\partial t}\left[u(x, t_0) + \frac{\Delta t}{2}\frac{\partial u_2}{\partial t}\right] \\
\frac{\partial u_4}{\partial t} &= \frac{\partial}{\partial t}\left[u(x, t_0) + \Delta t\frac{\partial u_3}{\partial t}\right]
\end{aligned} \tag{4.8}
$$

A disadvantage of the so-called RK4 scheme (4.8) is the amount of computer memory it requires: the user must allocate and maintain storage for $\frac{\partial u_1}{\partial t}$, $\frac{\partial u_2}{\partial t}$, $\frac{\partial u_3}{\partial t}$, and $\frac{\partial u_4}{\partial t}$ through the entire step. To mitigate this aspect of these methods, Williamson [63] developed Runge-Kutta schemes that need only $2N$ storage for a simulation with N spatial locations in the domain. Because of their low storage and high accuracy, optimized RK-$2N$ schemes have become quite popular in the field of CAA [64, 65, 66, 67, 68, 69, 70, 71]. The pseudocode for a RK-$2N$ scheme is shown in Algorithm 7.

---

procedure RK2N(Δt, u)
  for s $\leftarrow 1,$ num_stages **do**
    $w \leftarrow \alpha[s] * w + \Delta t \frac{\partial u}{\partial t}$ ▷ $\alpha[1] = 0$
    $u \leftarrow u + \beta[s] * w$
  end for
end procedure

Algorithm 7 Pseudocode for a $2N$-storage Runge-Kutta scheme. α and β are the coefficients associated with the particular scheme, and w is a working variable local to the procedure.

A variety of time-marching and spatial differencing schemes are implemented in BASS. The spatial stencils available include the Dispersion Relation Preserving (DRP) scheme of Tam and Webb [72] with coefficients from Tam and Shen [73], and Hixon's prefactored form [74] of the compact 6^{th}-order scheme of Lele [61] (C_6),

as well as standard 2nd- and 6th-order explicit central differencing (E_2 and E_6, respectively). For time-marching, the four- and five-stage Runge-Kutta schemes of Jameson [75] (RK4L and RK5L, respectively), the 5/6 stage two-step Runge-Kutta scheme of Stanescu and Habashi [71] (RK56), and the High-Accuracy Large-step Explicit Runge-Kutta (HALE-RK) 7-stage and 6/7-stage two-step schemes of Allampalli et al. [70] (RK7S and RK67) are used in this work.

The accuracy of both spatial differencing and time-marching schemes are commonly analyzed using Fourier analysis. For spatial differencing schemes, this approach consists of investigating the error associated with approximating the derivative of a single component of a Fourier series $f_k(x) = e^{ikx}$, where k is a real constant (the wavenumber) and $i = \sqrt{-1}$. Using the second-order central scheme (4.6) to differentiate $f_k(x)$ results in

$$\begin{aligned} D_x(f_k(x)) &= \frac{f_k(x+\Delta x) - f_k(x-\Delta x)}{2\Delta x} \\ &= \frac{1}{2\Delta x}\left(e^{ik[x+\Delta x]} + e^{ik[x-\Delta x]}\right) \\ &= \frac{i}{\Delta x}\left[\sin(k\Delta x)\right] e^{ikx}, \end{aligned} \tag{4.9}$$

which is then compared to the exact value

$$\frac{\partial f_k(x)}{\partial x} = ike^{ikx} = \frac{i}{\Delta x}\left[k\Delta x\right] e^{ikx}. \tag{4.10}$$

The bracketed quantity in Equation (4.9),

$$(k\Delta x)^* = \sin(k\Delta x), \tag{4.11}$$

defined as the numerical wavenumber, is helpful in evaluating the relative error

$$\epsilon\left(k\Delta x\right) = \frac{D_x(f_k) - \frac{\partial f_k}{\partial x}}{\frac{\partial f_k}{\partial x}} = \frac{(k\Delta x)^*}{k\Delta x} - 1 \tag{4.12}$$

associated with spatial differencing schemes. Also of interest is p, the convergence rate of the relative error, which can be calculated by considering the slope of $\epsilon(k\Delta x)$ on a log-log plot, i.e.,

$$\begin{aligned} p &= \frac{\mathrm{d}\log(\epsilon)}{\mathrm{d}\log(k\Delta x)} \\ &= \frac{\mathrm{d}\log(\epsilon)}{\mathrm{d}k\Delta x}\frac{\mathrm{d}k\Delta x}{\mathrm{d}\log(k\Delta x)} \\ &= \frac{\mathrm{d}\log(\epsilon)}{\mathrm{d}k\Delta x}k\Delta x \\ &= \frac{1}{\epsilon}\frac{\mathrm{d}\epsilon}{\mathrm{d}k\Delta x}k\Delta x. \end{aligned} \tag{4.13}$$

The numerical wavenumber, relative error, and convergence rate of the spatial differencing schemes available in BASS are shown in Figure 4-1 and the top and bottom of Figure 4-2, respectively. The latter plot uses the number of points per wavelength $\frac{2\pi}{k\Delta x}$ for the abscissa.

A few features of Figures 4-1 and 4-2 are worth mentioning. For the numerical wavenumber results in Figure 4-1, the performance of each scheme can be judged by how closely it follows the curve corresponding to the exact value $(k\Delta x)^* = k\Delta x$. Clearly, then, the explicit second-order and compact sixth-order schemes are the least and most accurate, respectively. The DRP and explicit sixth-order schemes make for a more interesting comparison, however: the DRP scheme outperforms the explicit sixth-order for higher $k\Delta x$ values, but it becomes difficult to distinguish the two schemes as $k\Delta x$ decreases. Inspection of the relative error plot in Figure 4-2, however, plainly shows that the DRP scheme's error is actually higher than the explicit sixth-order for high points-per-wavelength (low $k\Delta x$). The DRP scheme is an optimized

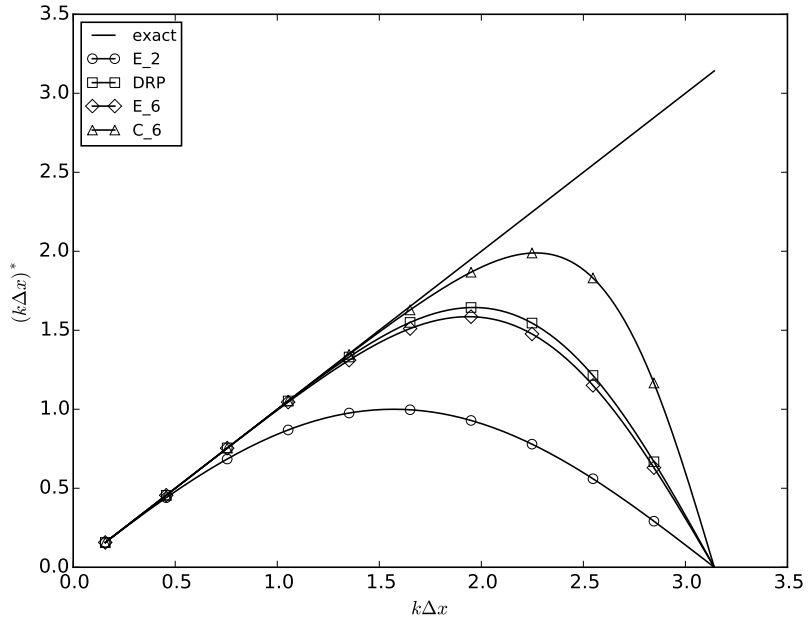


Figure 4-1: Numerical wavenumber $(k\Delta x)^*$ of the four spatial differencing schemes implemented in BASS.

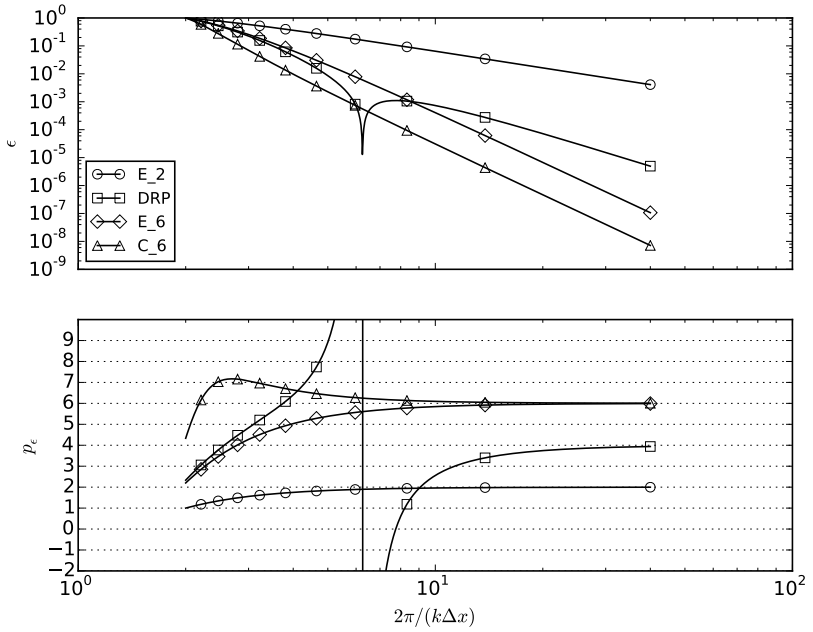


Figure 4-2: Relative error and convergence rate of the four spatial differencing schemes implemented in BASS.

form of the explicit sixth-order: it has the same stencil width, but slightly modified coefficients that give better performance for moderate points-per-wavelength when compared to the traditional explicit sixth-order.

The convergence rate curves clearly identify the order of the truncation error of each scheme: the explicit second-order scheme is second-order, the DRP scheme is fourth-order, and the explicit and compact sixth-order schemes are sixth-order. Considering the top and bottom plots of Figure 4-2 helps illustrate the difference between accuracy and order-of-accuracy. For example, despite its lower order-of-accuracy, the DRP scheme is more accurate than the explicit sixth-order scheme for a considerable range of points-per-wavelength, and even outperforms the very accurate compact sixth-order scheme for a narrow band of $\frac{2\pi}{k\Delta x}$. And though they possess identical orders-of-accuracy, the explicit and compact sixth-order schemes return widely different error over nearly the entire range of $k\Delta x$ shown in Figure 4-2, with the compact scheme beating the explicit by about an order-of-magnitude.

Fourier analysis applied to time-marching schemes usually involves solving a simple ordinary differential equation like

$$\frac{du}{dt} = -i\omega u \tag{4.14}$$

with exact solution

$$u(t) = u_0 e^{-i\omega t}. \tag{4.15}$$

Here, $u_0 = u(t = 0)$ and the time-marching scheme to be analyzed is used to solve Equation (4.14), i.e., $\frac{\partial u}{\partial t}$ is replaced with $-i\omega u$ in Equation (4.5), Equation (4.8), or Algorithm 7. Following this procedure with time step size Δt for Equation (4.5)

gives

$$\begin{aligned} u(\Delta t) &= u_0 + \Delta t\,(-i\omega u_0) \\ &= (1 - i\omega\Delta t)\,u_0 \\ &= G(\omega\Delta t)u_0. \end{aligned} \tag{4.16}$$

$G = \frac{u(\Delta t)}{u(0)}$, the ratio of the solution at the next and current time step, is called the amplification factor and is analogous to the $(k\Delta x)^*$ for spatial differencing schemes. Once G is found for a particular scheme, the relative error associated with one time step (called the local error [62, p. 66]) can be found through the expression

$$\begin{aligned} \theta(\omega\Delta t) &= \frac{u_{\text{scheme}}(\Delta t) - u_{\text{exact}}(\Delta t)}{u_{\text{exact}}(\Delta t)} \\ &= G(\omega\Delta t)/g(\omega\Delta t) - 1 \end{aligned} \tag{4.17}$$

where $g(\omega\Delta t) = e^{-i\omega\Delta t}$ is the amplification factor for the exact solution (4.15). The convergence rate p for the error in Equation (4.17) is found in a manner similar to a spatial differencing scheme's, namely,

$$p = \frac{1}{\theta}\frac{\mathrm{d}\theta}{\mathrm{d}\omega\Delta t}\omega\Delta t. \tag{4.18}$$

The magnitude of the amplification factor G for each of BASS's schemes is shown in Figure 4-3, and the corresponding relative local error and convergence rate curves for these schemes are shown in Figure 4-4. Similar to the $(k\Delta x)^*$ plot of Figure 4-1, the accuracy of a scheme in Figure 4-3 is determined by how closely it matches the exact value of the amplification factor magnitude, which for Equation (4.15) is unity. Figure 4-3 indicates the stability limit of the schemes, which is the largest value of $\omega\Delta t$ where $|G| \leq 1$, i.e., the amplitude of the solution does not grow as time

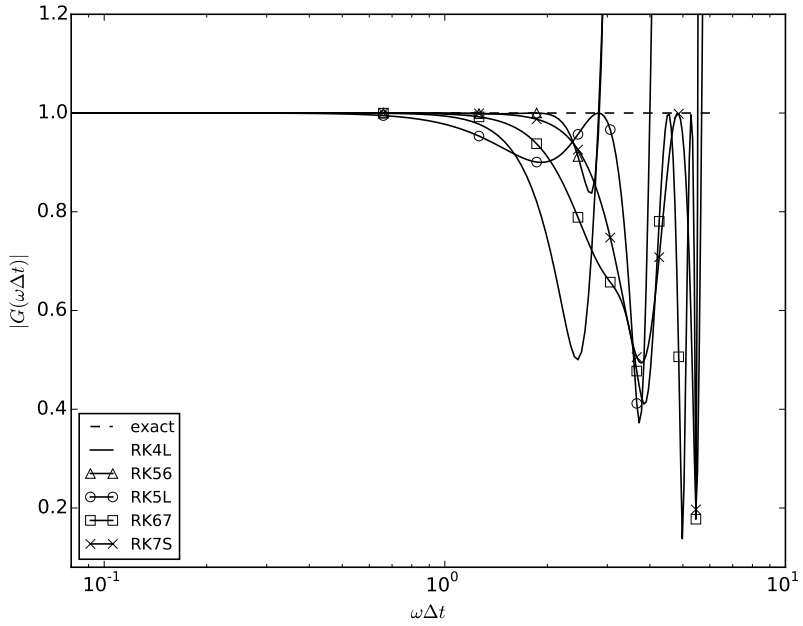


Figure 4-3: Magnitude of the amplification factor for each of the five time-marching schemes implemented in BASS.

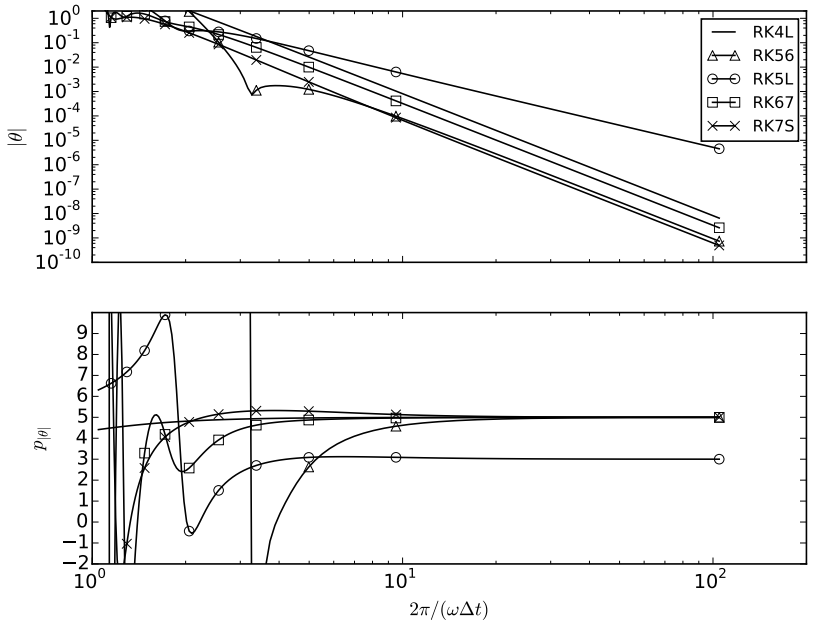


Figure 4-4: Magnitude of the relative local error and convergence rate for each of the five time-marching schemes implemented in BASS.

increases. The amplification factor curves show, then, that the RK56 has both the highest accuracy and most restrictive stability limit of all of the schemes. The RK4L has approximately the same stability limit of the RK56, but dampens the solution considerably. The RK5L is clearly the least accurate scheme, but is much more stable than the RK4L and RK56. Finally, the RK67 and RK7S appear to possess the highest stability limits.

Figure 4-4 allow for a better comparison of the performance of the time-marching schemes. Like the corresponding spatial differencing error plot presented above, the quantity $2\pi/(\omega\Delta t)$, the number of time steps per period of the solution in Equation (4.15) is used for the abscissa. The error curves confirm that the RK56 is a highly-accurate scheme, showing the sharp error "dip" characteristic of optimized numerical methods (e.g. the DRP spatial differencing scheme in Figure 4-2), and that the RK5L is by far the least accurate. The convergence rate curves show that the RK5L scheme eventually exhibits third-order behavior, while all of the other schemes converge at a fifth-order rate. The convergence rate of a Runge-Kutta's local error is always one greater than its formal order-of-accuracy, indicating that the RK5L is second-order, and all other schemes are fourth-order. The RK4L scheme, however, is actually only fourth-order for linear problems. This scheme, Equation (4.8) recast as a 2N Runge-Kutta method, is second-order when used to solve non-linear ODEs.

Equation (4.17) shows how the error committed during one time step varies with Δt. Alternatively, one can calculate the error accumulated after marching to a constant final time level $T = mn\Delta t$ using n time steps for a m-step scheme (the global error), which is

$$\begin{aligned} \vartheta(n) &= \frac{u_{\text{scheme}}(T) - u_{\text{exact}}(T)}{u_{\text{exact}}(T)} \\ &= \frac{\left[G\left(\omega\frac{T}{mn}\right)\right]^n}{g(\omega T)} - 1. \end{aligned} \tag{4.19}$$

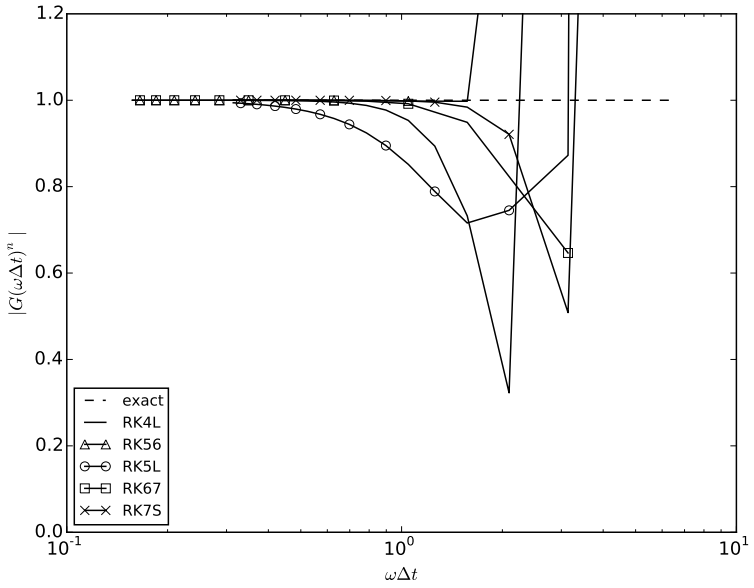


Figure 4-5: Magnitude of the global amplification factor for each of the five time-marching schemes implemented in BASS after marching to $\omega T = 2\pi$.

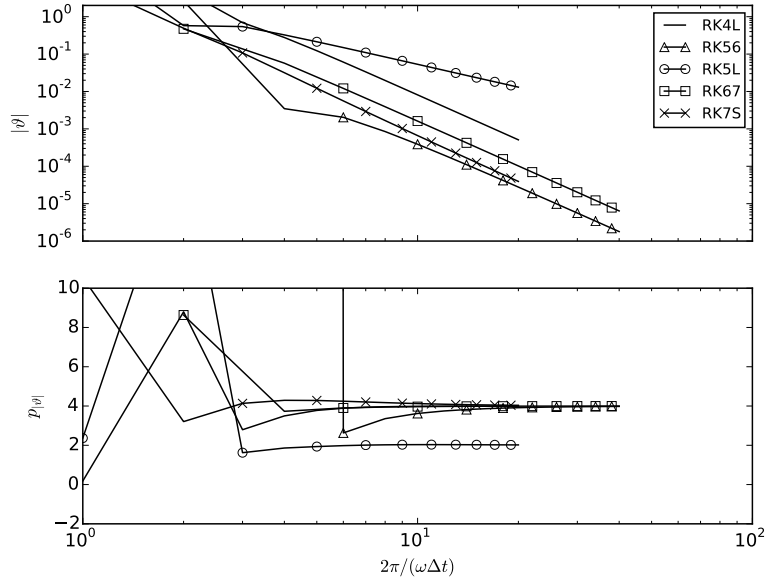


Figure 4-6: Magnitude of the relative global error and convergence rate for each of the five time-marching schemes implemented in BASS after marching to $\omega T = 2\pi$.

An expression for the global error's convergence rate can be found by substituting $\omega\Delta t = \omega T/(mn)$ into Equation (4.18), which eventually gives

$$p = -\frac{n}{\vartheta}\frac{\mathrm{d}\vartheta}{\mathrm{d}n}. \tag{4.20}$$

Figures 4-5 and 4-6 show the magnitude of the global amplification factor, error, and convergence rate. Again, the RK56 and RK5L are seen to be the most and least accurate, respectively. The global convergence rate results show that each scheme converges at its formal order-of-accuracy. Also note how each scheme's asymptotic range (the range of $\omega\Delta t$ the scheme converges at the expected rate) differs, with the RK56's starting at the highest points-per-cycle value. Typical of low-order methods, the RK5L achieves its design order-of-accuracy more quickly than the other schemes.

4.2 Spatial differencing verification results

As described in Section 3.3, verifying a PDE code's spatial differencing scheme requires running the PDE code with a series of progressively finer grids in order to observer how the code's error converges as the grid spacing is decreased. For the present test cases, each grid in the series began as a uniform Cartesian grid extending from -0.5 to 0.5 in the three coordinate directions, and was then rotated, skewed, and subjected to sinusoidal perturbations. The coarsest grid in the series used 9^3 points — denser grids were constructed by uniformly refining the base grid by adding 1, 2, 3... points between each of the original grid's, giving a grid series of 9^3, 17^3, 25^3, 33^3,..., 201^3 points with approximate grid spacing

$$\widetilde{\Delta x} = \frac{1}{\sqrt[3]{N-1}} \tag{4.21}$$

ranging from $\frac{1}{8}$ to $\frac{1}{200}$. Figure 4-7 shows the resulting grid with 41^3 points.

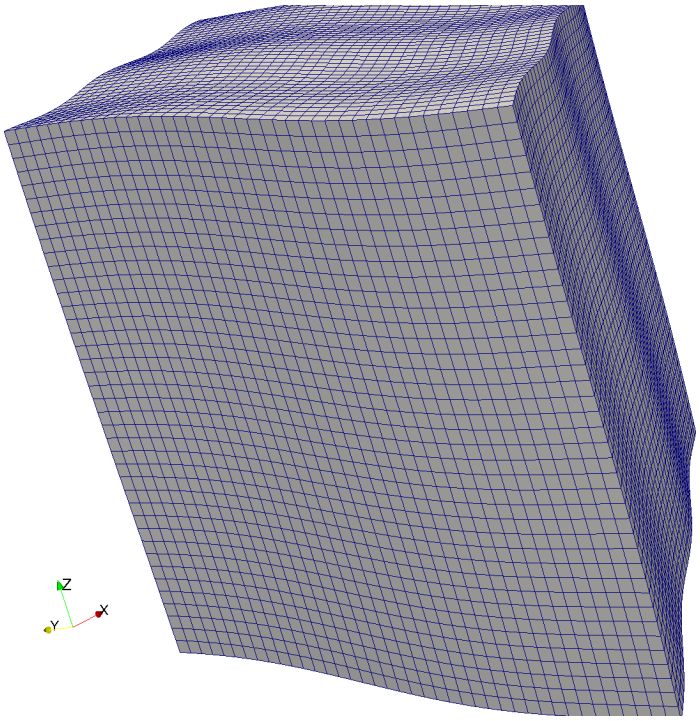


Figure 4-7: BASS verification test case grid with 41^3 points.

The initial condition used for each of the spatial differencing test cases was of the form

$$\phi(x,y,z) = \overline{\phi} + \widetilde{\phi} \exp\left(-\frac{\ln 2}{0.15^2}\left[(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2\right]\right) \sin\left(k_x x + k_y y + k_z z + \theta\right), \tag{4.22}$$

with the values of the various constants given in Table 4.1. The center of the Gaussian perturbations (i.e., x_0, y_0, z_0 from Equation (4.22)) for each of the flow variables were placed at different locations in the domain to avoid any symmetry in the initial flow; Figure 4-8 uses colored spheres to represent these location. The values of k_x, k_y, and k_z were set such that the direction the sin() term varies the most rapidly differs for each variable, but has the same wavelength of 0.25, giving an approximate wavenumber $\widetilde{k} = \frac{2\pi}{0.25} = 8\pi$. Figures 4-9 and 4-10 show slices of the density and z-momentum initial conditions, respectively.

ϕ	$\overline{\phi}$	$\widetilde{\phi}$	x_0	y_0	z_0	k_x	k_y	k_z	θ
σ	1.	0.001	-0.1	-0.1	-0.1	0.875917	0.0459049	25.1174	7.5
u	0.03	0.006	-0.05	0.05	0.05	4.00477	14.9460	19.8048	9.
v	0.02	0.004	0.05	-0.05	-0.05	-20.2615	13.1580	6.92752	10.5
w	0.01	0.005	-0.05	0.05	-0.05	-18.3538	-14.8626	-8.59590	12.
p	$\frac{1}{\gamma}$	0.01	0.1	0.1	0.1	5.03652	-13.1205	-20.8359	13.5

Table 4.1: Values of constants for the initial condition found in (4.22).

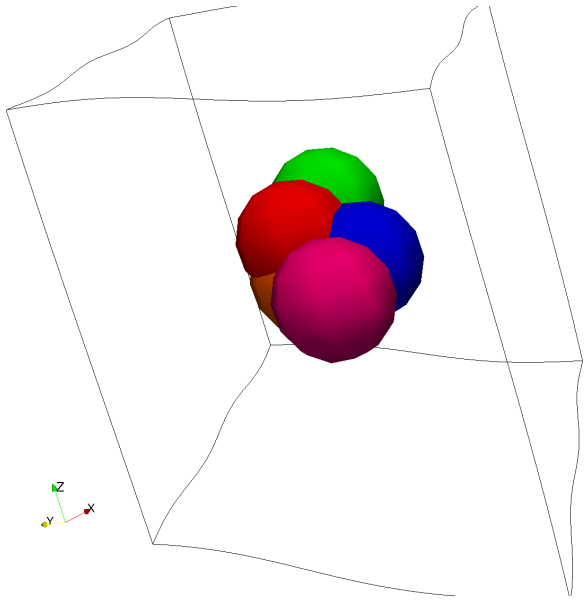


Figure 4-8: Outline of the BASS verification test case grid with spheres marking the center of the Gaussian perturbations of the initial flow, with purple, red, blue, orange, and green indicating the σ, u, v, w, and p centers, respectively. The radius of each sphere is 0.15, the half-width of the Gaussians.

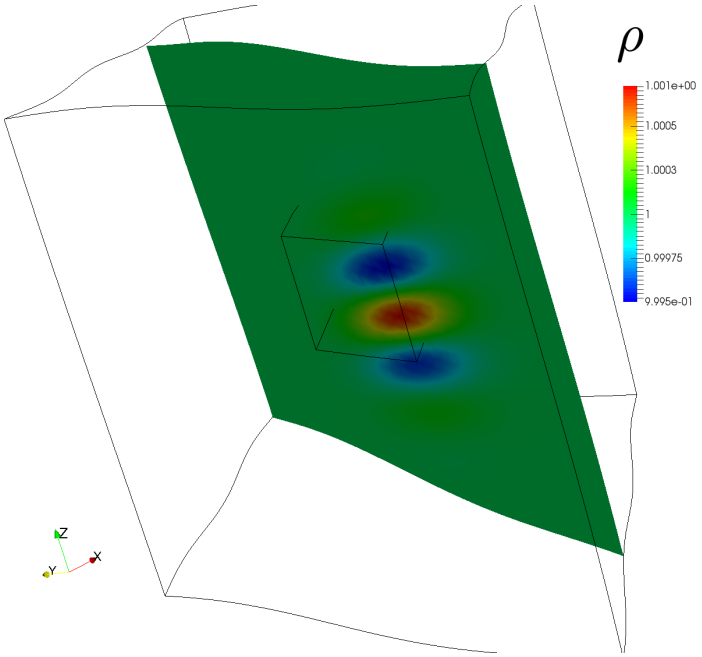


Figure 4-9: Slice of spatial test case grid colored by the ρ initial condition from Equation (4.22). The inner box indicates the region where BASS's solution was compared with EVA.

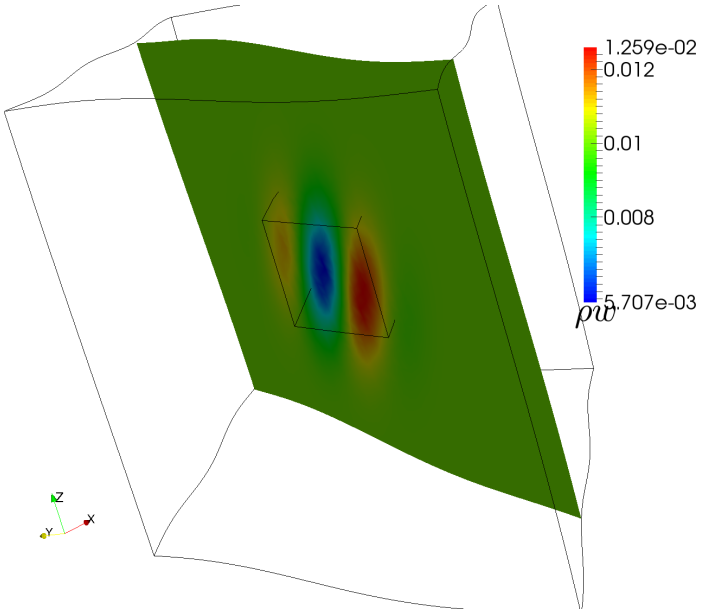


Figure 4-10: Slice of spatial test case grid colored by the ρw initial condition from Equation (4.22). The inner box indicates the region where BASS's solution was compared with EVA.

The common ratio of specific heats value for air, $\gamma = 1.4$, was used for both the inviscid and viscous runs. This, combined with the choices $\overline{p} = \frac{1}{\gamma}$ and $\overline{\rho} = 1$ give a far-field acoustic velocity of unity. The maximum propagation speed based on far-field quantities, then, would be $c = 1 + \sqrt{0.03^2 + 0.02^2 + 0.01^2} \approx 1.0374$. The viscous runs used a dynamic viscosity $\mu = 0.0002$, which, when combined with a reference length $L = 2 \cdot 0.15$ (twice the Gaussian half-width in the initial condition), $V = \sqrt{0.03^2 + 0.02^2 + 0.01^2}$ (far-field velocity), and $\overline{\rho} = 1$, gives a Reynolds number

$$Re = \frac{\overline{\rho} V L}{\mu} \approx 56.1, \tag{4.23}$$

which is much lower than what is typically found in aerospace applications. The rationale behind using such a small value is related to the usual physical interpretation of the Reynolds number: since it represents the ratio of the inviscid and viscous terms, a Reynolds number closer to unity means these terms are approximately the same magnitude, and problems with one is unlikely to be hidden by the other. Likewise, the Prandtl number Pr, often described as the ratio of the viscous and conductive terms, was set to 0.7 (the accepted value for air). Finally, the Peclet number, defined $Pe = Re \cdot Pr$ and a measure of the ratio of the inviscid and conductive terms, was $Pe = 56.1 \cdot 0.7 = 39.3$.

The RK7S time-marching scheme was used to march the initial flow to a final time level of $t = 0.0005$ with one time step. The choices of very small Δt and highly accurate time-marching scheme are meant to ensure that the error associated with the time-marching scheme is much lower than that of the spatial differencing. For the EVA calculation, a desired truncation error of 10^{-15} was used, which was easily satisfied, again thanks to the small time step size.

Because EVA solves the governing equations on an infinite domain, the Giles non-reflecting condition [76] was used for all domain boundaries.

Comprehensive l_{max} data for the inviscid and viscous test cases are available in Appendix G; the most interesting results will be presented here, with a focus on the viscous solver. The l_2 error and the convergence rate of the l_2 norm, p_{l_2}, for the density flow variable, are shown for the viscous results in Figure 4-11. The quantity $2\pi/(\widetilde{k}\widetilde{\Delta x})$, an approximation of the number of points-per-wavelength, is used for the abscissa to facilitate comparison with Figure 4-2. While not identical, Figures 4-2 and 4-11 are remarkably similar. Like the linear analysis, the compact sixth-order and explicit second-order schemes are clearly the most and least accurate, respectively. The DRP scheme outperforms the explicit sixth-order scheme for moderate points-per-wavelength, but then returns higher error for the high $2\pi/(\widetilde{k}\widetilde{\Delta x})$ values, as predicted by Figure 4-2. Many similarities are found in the convergence rate results, also: the compact sixth scheme initially shows higher-order behavior than the explicit sixth; the characteristic order spike is seen in the DRP curve, and the explicit second-order converges at its expected rate for a wide range of points-per-wavelength.

Overall it made little difference if the convergence rate was calculated using Equation (3.5) or Equation (3.12). Figure 4-12 compares the two approaches for the l_{max} norm of the ρu error. The expected convergence rate is eventually observed for each scheme, and the curves are qualitatively similar. This indicates that the temporal component of the error is small relative to that of the spatial differencing scheme, and thus the assumption of Equation (3.6) is a good one.

The viscous and inviscid results were also very similar. Figure 4-13 shows the $l_{\max}$ error and convergence rate for the total energy variable for both equations. The error curves are indistinguishable on the plot, but some differences are observed in the convergence rate data. This similarity is most likely due to the final time level $t = 0.0005$ being so close to the initial flow — the viscous run hasn't had time to evolve differently from the inviscid, despite its rather low Reynolds number.

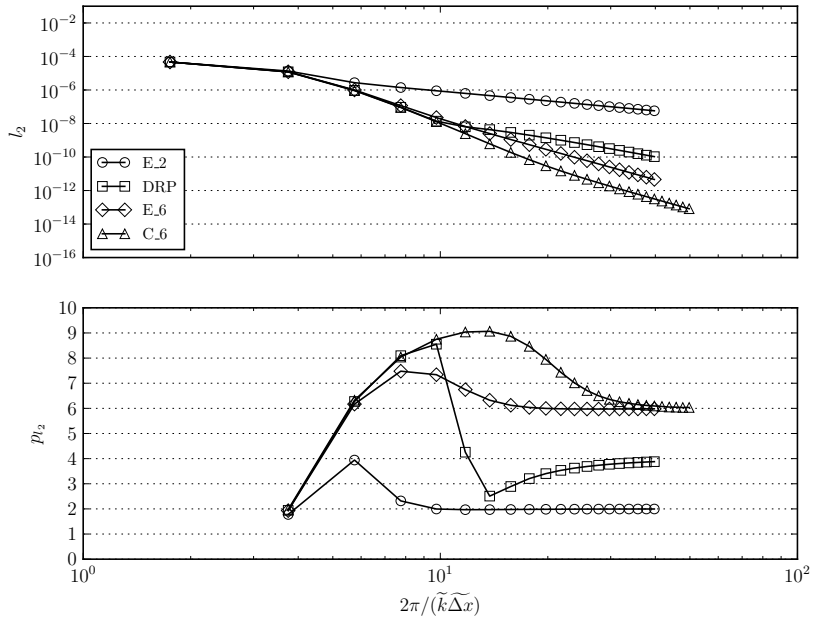


Figure 4-11: ρ l_2 error and convergence rate for each of BASS's spatial differencing schemes after solving the viscous equations. p_{l_2} was calculated using Equation (3.8).

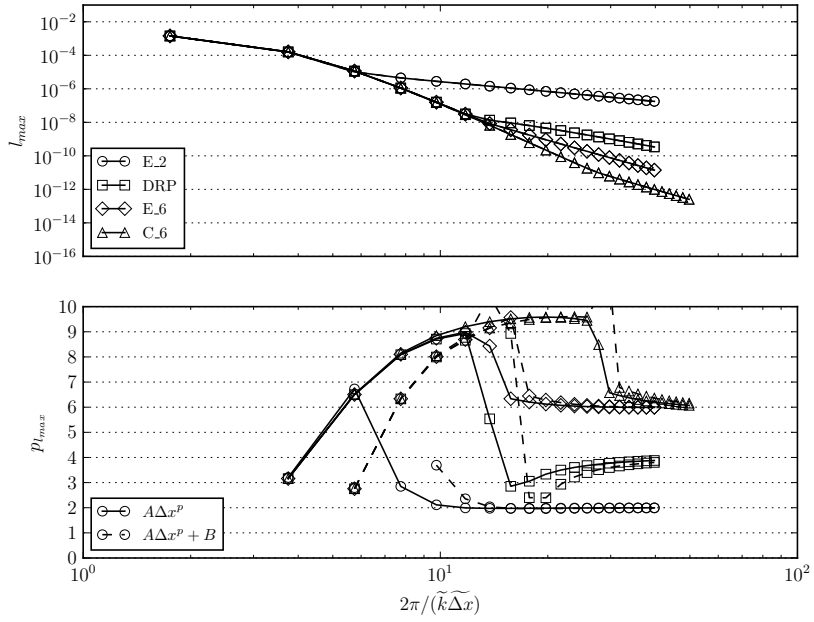


Figure 4-12: ρu $l_{\max}$ error and convergence rate for each of BASS's spatial differencing schemes after solving the viscous equations.

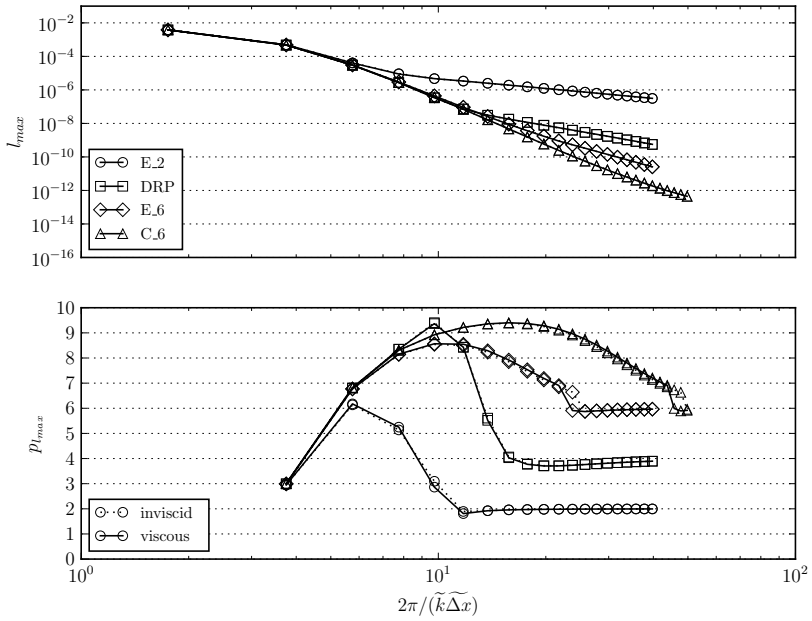


Figure 4-13: E_t $l_{\max}$ error and convergence rate for each of BASS's spatial differencing schemes for both the viscous and inviscid equations.

On the other hand, one may rightly be suspicious of the remarkable agreement in Figure 4-13. To ensure that the EVA and BASS codes are actually solving different equations when switching on the viscous terms, the data from the inviscid EVA run was used as the reference solution for the BASS viscous data. The results for the E_2 and E_6 scheme are compared to the original results in Figures 4-14 and 4-15, respectively. Interestingly, the density convergence rate for the explicit second-order scheme is second-order for the run solving the incorrect governing equations, but the total energy is not. This is undoubtedly because the equation controlling the change in density, the continuity equation, is identical for the Euler and Navier-Stokes. The continuity equation is coupled with the others, of course, but taking just one tiny time step hasn't allowed the E_2 to "notice," despite the seven stages of the RK7S scheme. On the other hand, the more accurate explicit sixth-order scheme's convergence rate results are clearly affected by solving the incorrect equations, as is the explicit second-

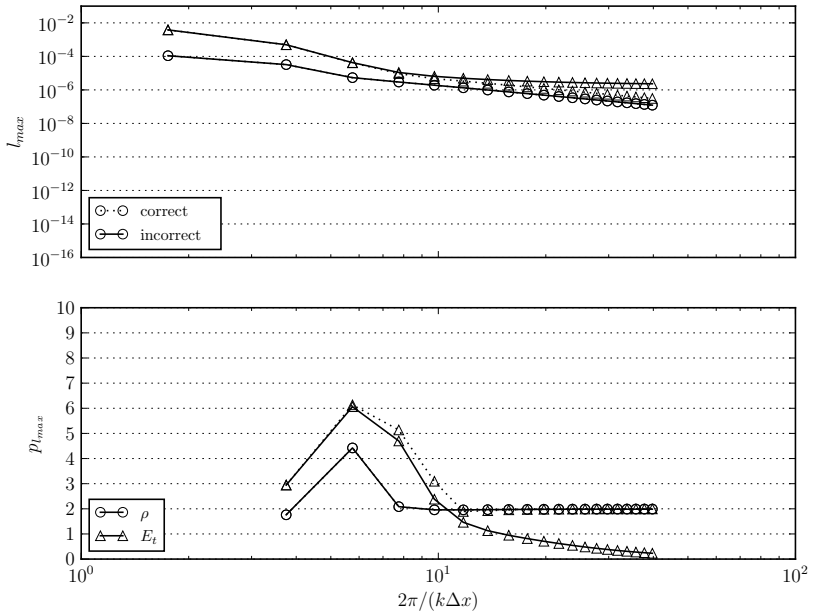


Figure 4-14: $l_{\max}$ error and convergence rates for the inviscid and viscous E_2 spatial differencing scheme referenced to the inviscid EVA solution.

order's total energy variable.

Achieving asymptotic convergence is the most difficult for the compact sixth-order scheme. Figure 4-16 compares the compact and explicit sixth-order performance for the Navier-Stokes equations. Notice how significantly more points-per-wavelength are required for the compact scheme to achieve the expected p, a result not predicted by the linear analysis shown in Figure 4-2. The cause may be related to the Giles non-reflecting boundary conditions used by the BASS code: any discrepancy between these BCs and the EVA solution will be propagated more readily by the compact scheme, as it requires a global matrix solve along each grid line. One would expect errors from the boundaries to become less significant as the number of grid points increases, which corresponds well with the behavior seen in Figure 4-16.

All of the test cases discussed thus far have involved serial BASS calculations, i.e., the code used just one processor to compute the entire flow field. BASS, however, is a

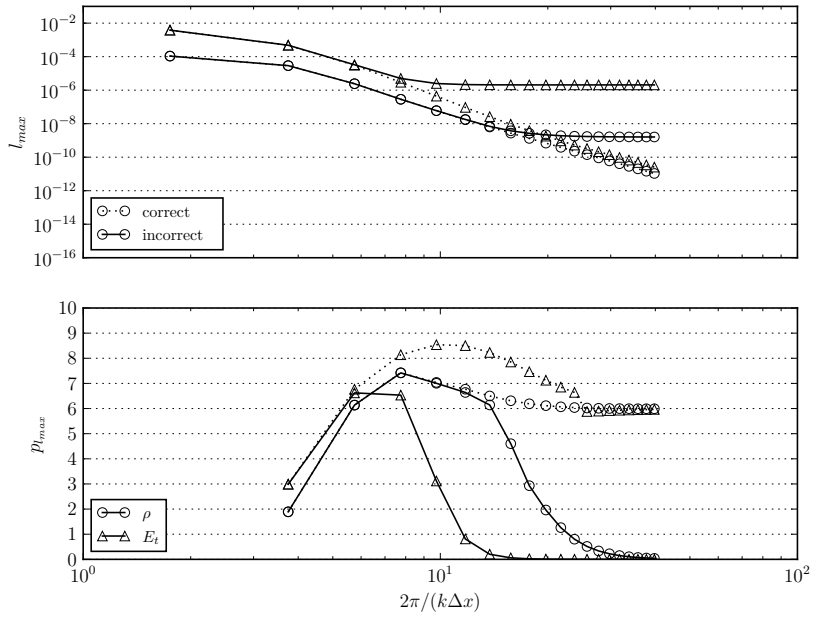


Figure 4-15: $l_{\max}$ error and convergence rates for the E_6 spatial differencing scheme referenced to the inviscid EVA solution.

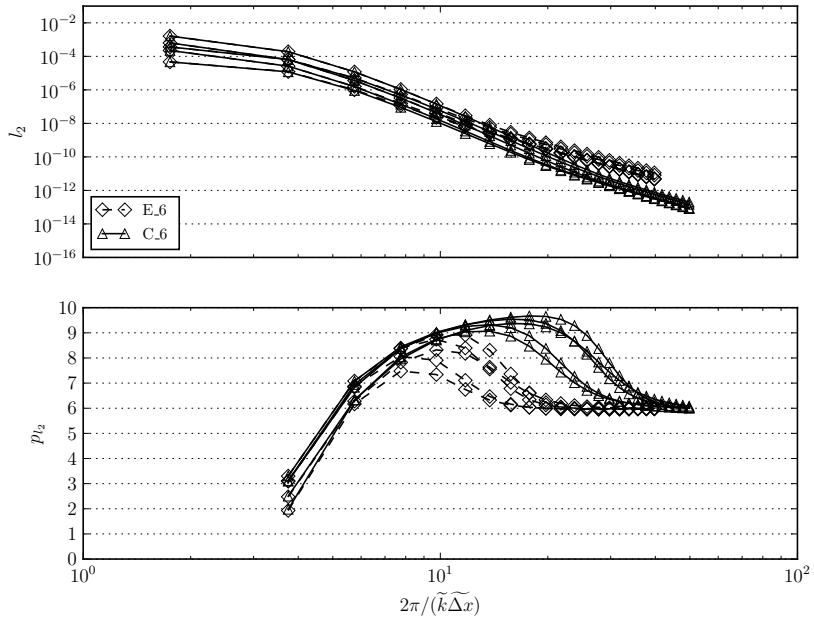


Figure 4-16: l_2 error and convergence rate for BASS's sixth-order schemes. Viscous results with Equation (3.8) p_{l_2} calculation.

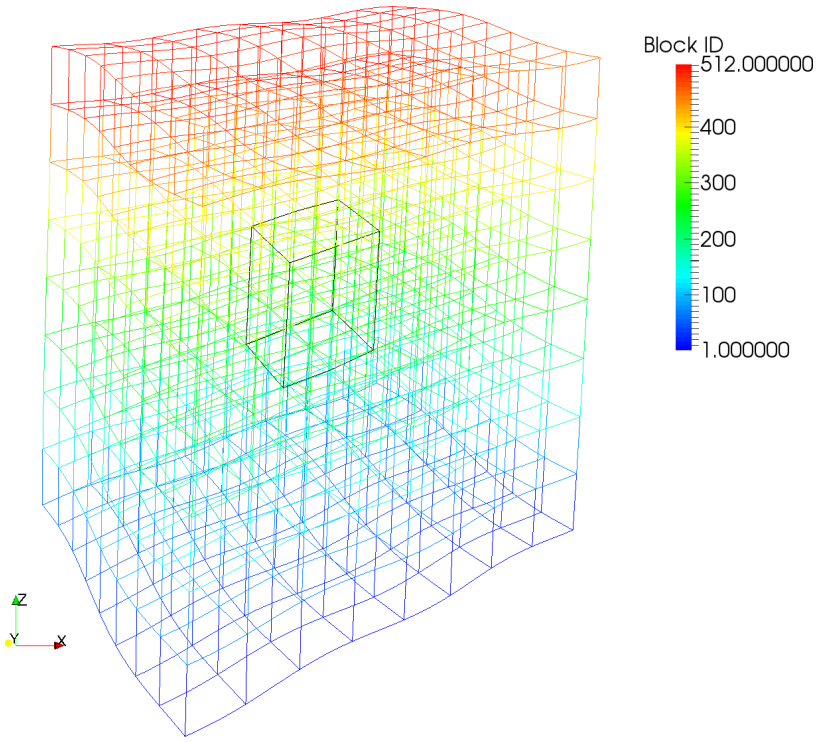


Figure 4-17: Multiblock grid used for parallel test cases colored by block index. The extent of the EVA solution domain is outlined in black.

parallel code, and can decompose the solution domain into blocks that are distributed to and solved by many processors. To investigate the portions of the code responsible for the parallelization, the grid in Figure 4-7 was split into $8^3 = 512$ blocks, and then run with the same initial condition, grid spacings, and time step sizes as the preceding test cases with 24 processors. Figure 4-17 displays the grid's block boundaries colored by block index number, which shows that multiple blocks were placed inside the EVA domain to ensure any problems with BASS's parallel routines would be noticed.

Figure 4-18 shows the viscous density l_2 error and convergence rate results for each of BASS's spatial differencing schemes (the equivalent of Figure 4-11). The smaller points-per-wavelength grids from the serial test case were not run because they did not meet the minimum points-per-block required by BASS. Like the previous test cases, each scheme's convergence rate eventually attains its expected value.

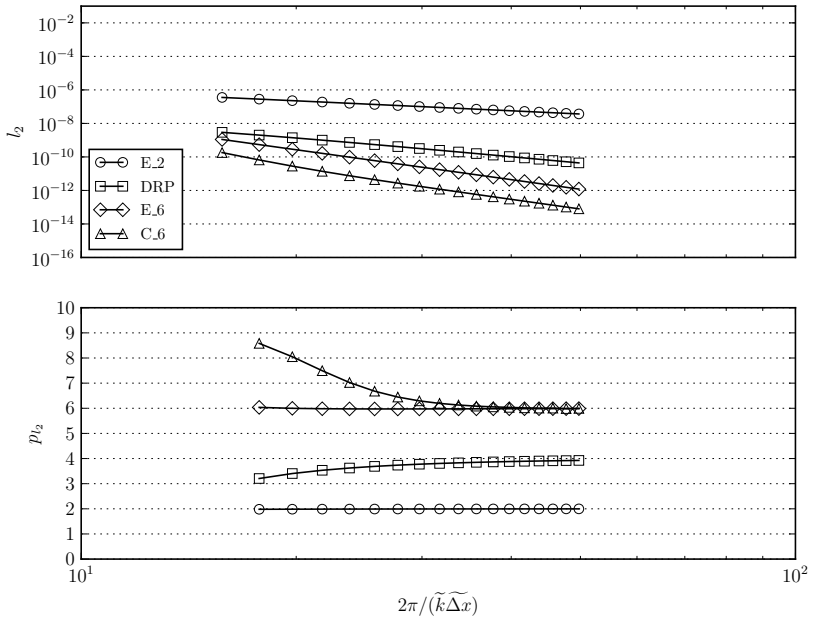


Figure 4-18: ρ l_2 error and convergence rate for each of BASS's spatial differencing schemes. Viscous, parallel results.

The results in Figure 4-18 (the parallel runs) are essentially identical to those in Figure 4-11. This is expected, since the numerical methods used by BASS do not change when the code is run in parallel. The one exception is the compact sixth scheme, which uses an explicit sixth-order stencil at interior block boundaries. As described in [74], this stencil has been tuned to give similar accuracy performance to the compact sixth scheme, and the results shown in the figure confirm that this is the case.

4.3 Time-marching verification results

Most of the parameters from the spatial verification test cases presented in the previous section were reused for the time-marching verification work. The grid shape (Figure 4-7), initial condition (Equation (4.22) and Table 4.1), and governing equation parameters were, for the most part identical (exceptions will be noted). To keep the

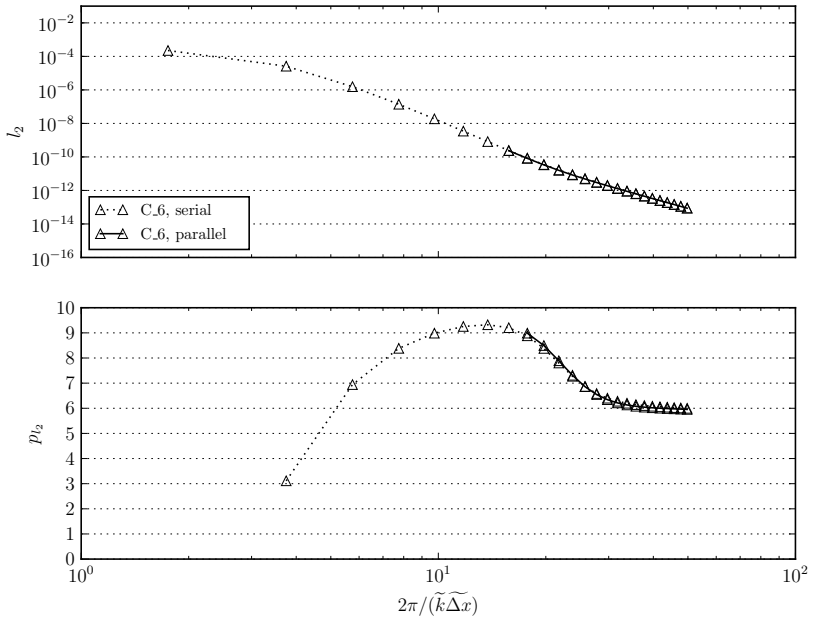


Figure 4-19: ρu l_2 error and convergence rate for the compact sixth-order scheme for the serial and parallel test cases. Viscous results.

spatial component of the error low, the densest grid from the spatial verification cases (201^3 points) and the explicit sixth-order scheme were used for all temporal verification runs. Each BASS calculation marched to a final time level of $t = 0.05$ with a varying number of time step ranging from 1 to 40, depending on what was necessary to achieve asymptotic convergence. A target truncation error of 10^{-12} was requested from the EVA tool, which required a 26^{th}-order Taylor series for both governing equations.

Similar to the preceding spatial verification results, an approximate frequency $\widetilde{\omega}$ will be used to plot the error and convergence rates as functions of time steps per period of the disturbance. Here, the choice $\widetilde{\omega} = c\widetilde{k}$ was made, where c and $\widetilde{k}$ are the approximate maximum propagation speed and wavenumber from the spatial verification cases, respectively. Also like the previous section, complete plots are shown in Appendix H.

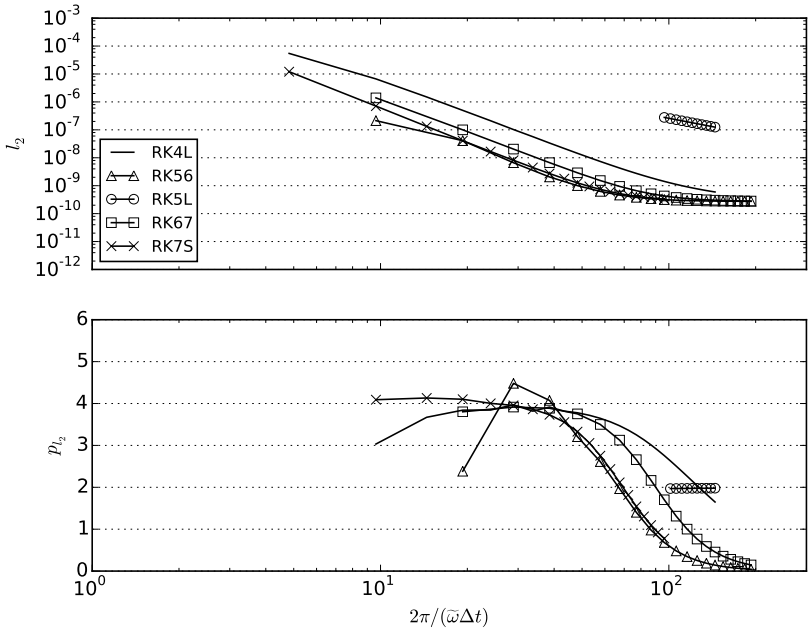


Figure 4-20: Density l_2 and p_{l_2} results for viscous BASS's time-marching schemes. Equation (3.8) was used to calculate p_{l_2}.

Figure 4-20 shows the density l_2 error and convergence rate results for the viscous runs, with the convergence rate calculated from Equation (3.8). The high-frequency portion of the plot (i.e., the left side) resembles the global error plot from the linear analysis, Figure 4-6: the RK56 is the most accurate, followed closely by the RK7S and then RK67, with the RK4L and RK5L showing significantly higher error. The results for the low-wavenumber runs, however, do not line up well with Figure 4-6: with the exception of the RK5L, the scheme's error curves flatten out, indicating that the error in the BASS calculation is no longer sensitive to the time step size. This, of course, is reflected in the convergence rate results. The RK7S and RK67 schemes only briefly attain their design convergence rates, and the RK56 never does. Also, the RK4L, a second-order scheme for non-linear problems, behaves as a fourth-order scheme before encountering the "error floor."

The density l_2 data from Figure 4-20 is reproduced in Figure 4-21, but the con-

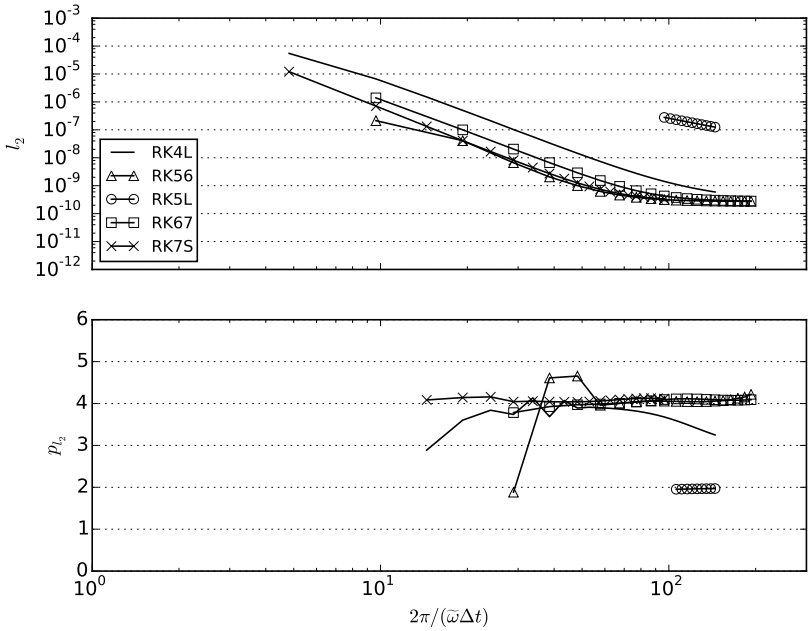


Figure 4-21: Density l_2 and p_{l_2} results for viscous BASS's time-marching schemes. Equation (3.12) was used to calculate p_{l_2}.

vergence rate is calculated with Equation (3.12), which, unlike the spatial verification results, has a significant impact on the p data. The three fourth-order non-linear schemes now convincingly converge at a fourth-order rate, and the RK5L's error still shows strong second-order behavior. The RK4L scheme's error convergence, however, remains something between third and fourth-order.

The RK4L's convergence rate results in Figure 4-21 would seem to indicate that the present test case does not contain sufficient nonlinearity for the scheme to converge at its non-linear order-of-accuracy. To determine if this is the case, the RK4L scheme was run with an initial condition identical to that shown in Equation (4.22) and Table 4.1, but with larger Gaussian perturbation amplitudes, specifically $\widetilde{\sigma} = 0.1$, $\widetilde{u} = 0.012$, $\widetilde{v} = 0.08$, $\widetilde{w} = 0.0075$, $\widetilde{p} = \frac{1}{10\gamma}$. The density l_2 results for this high-perturbation test case are compared to the original in Figure 4-22. As might be expected, the larger Gaussian amplitudes increase the l_2 error markedly, but also re-

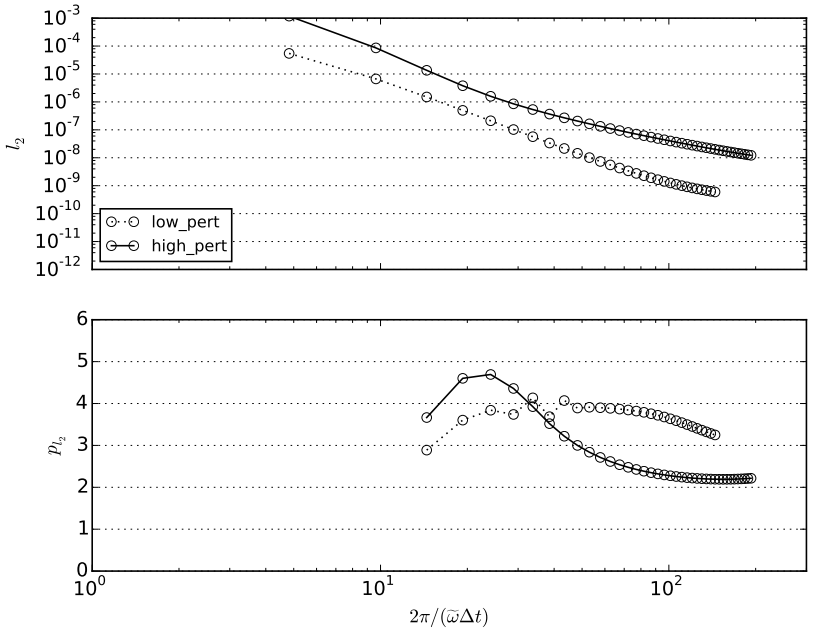


Figure 4-22: Density l_2 and p_{l_2} results for viscous BASS's RK4L scheme with low- and high-amplitude Gaussian perturbations. Equation (3.12) was used to calculate p_{l_2}.

duce the RK4L's convergence rate to 2, the expected value for non-linear differential equations.

The RK56's $l_{\max}$ error exhibits an interesting behavior for the ρw variable. Figure 4-23 compares this data with the corresponding l_2 case. The p_{l_2} curve is fairly close to the expected fourth-order convergence, but $p_{l_{\max}}$ is more like 3.5. This quality is not seen in the other flow variables or time-marching schemes, but is present in the viscous and inviscid test cases. The explanation may lie in the assumed form of the error used for calculating the convergence rate. For Equation (3.12), one assumed

$$\epsilon_{i,j} \approx A\Delta x_i^p + B\Delta t_j^q. \tag{4.24}$$

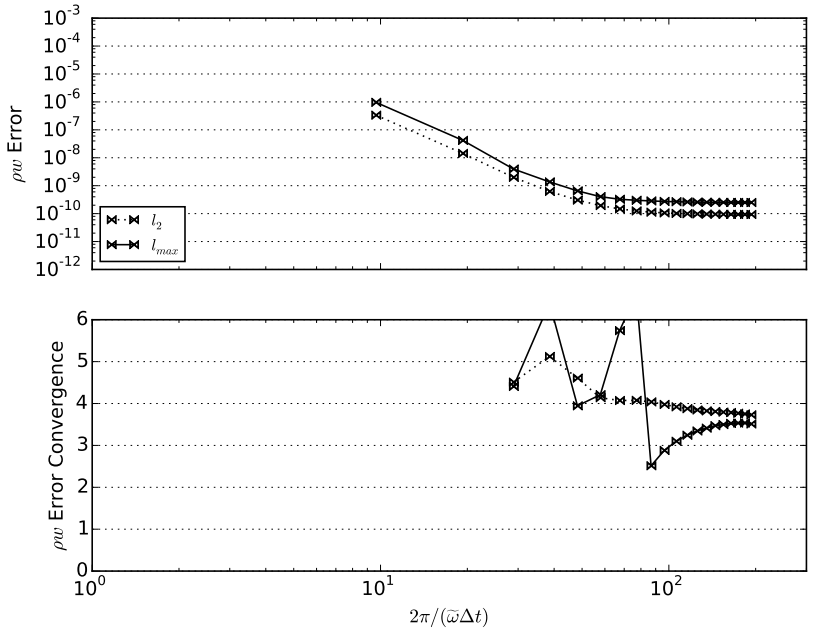


Figure 4-23: ρw l_2 and $l_{\max}$ data for viscous BASS's RK56 scheme. Equation (3.12) was used to calculate both convergence rates.

A more-accurate expression may be

$$\epsilon_{i,j} \approx A\Delta t\Delta x_i^p + B\Delta t_j^q. \tag{4.25}$$

If Equation (4.25) holds, one would expect the second term to dominate the error for most of the time steps in Figure 4-23, since the grid spacing is quite small. As the time step size is reduced, however, the two terms in Equation (4.25) may become closer in magnitude, which would tend to degrade the convergence rate calculation. This would especially be true for the RK56, the most accurate time-marching scheme, and the very sensitive $l_{\max}$ norm.

4.4 Verification of a two-dimensional multi-time-step Adams-Bashforth scheme

Each of the time-marching schemes discussed thus far require a uniform time step size Δt to be used throughout the solution domain. Problems encountered in the field of Computational Aeroacoustics, however, often contain a wide range of time scales — the flow may vary rapidly in one region, and slowly in another. The choice of Δt is determined by the smallest time scale present and the stability limit of the time-marching scheme, which will lead to excessively small time step sizes in areas where the flow is changing slowly. One would prefer to assign small Δt values to the quickly changing regions of the flow, and use large time step sizes where the flow is more stagnant, similar to the way grid is clustered in areas where the solution changes rapidly in space. To address this issue, Tam and Kurbatskii [77] proposed a multi-time step Adams-Bashforth (MTSAB) scheme that allows different time step sizes to be used in different areas of the flow, while retaining fourth-order accuracy. An automated form of the MTSAB scheme has been implemented in the BASS code by Allampalli and Hixon [78, 79].

The principle difference between an Adams-Bashforth scheme and the Runge-Kutta schemes described above is that the former require data from previous time *steps* to advance the solution. (Runge-Kutta schemes, on the other hand, only need the solution at the current step n to march to $n+1$.) The fourth-order version used in the MTSAB (referred to here as "AB4") is

$$u_{n+1} = u_n + \Delta t \sum_{m=0}^{3} b_m \frac{du_{n-m}}{dt} \tag{4.26}$$

where

$$
\begin{aligned}
b_0 &= \frac{55}{24} \\
b_1 &= -\frac{59}{24} \\
b_2 &= \frac{37}{24} \\
b_3 &= -\frac{9}{24}.
\end{aligned}
\tag{4.27}
$$

Notice how the solution at the next time step $(n+1)$ depends on the derivative of the solution at the current time step, *and* the three previous steps. Because of this, the AB4 scheme has a starting problem — a different scheme must be used at the start of the calculation to provide the data necessary to use Equation (4.26).

BASS's implementation of the MTSAB scheme assigns time step sizes on a block-by-block basis. Figure 4-24, adapted from [79], is a schematic of a one-dimensional grid with two blocks, with Block 1 using a time step size of Δt, and Block 2, $2\Delta t$. Figure 4-24 shows the state of the blocks just before marching from time level n to $n+1$ using the MTSAB scheme. Circles represent solution data at a particular spatial and temporal location, with the vertical and horizontal directions indicating increasing spatial and temporal distance, respectively. The black-colored locations are interior points, i.e., solution locations that lie within the spatial extent of the block and are marched in time by its controlling block only. At each block interface, the MTSAB scheme creates addition collections of data called buffer blocks. In Figure 4-24, Block 1's buffer block is colored green, and Block 2's buffer block is colored blue. Buffer blocks contain solution data that exists in an adjoining block's interior points, and is required to calculate the spatial derivative of a block's interior point. For example, Block 1's left-most buffer block point contains the same data as Block 2's left most *interior* point. The size of the buffer blocks is dependent on the width of the finite difference stencil used to differentiate the solution in space. In Figure 4-24, the width

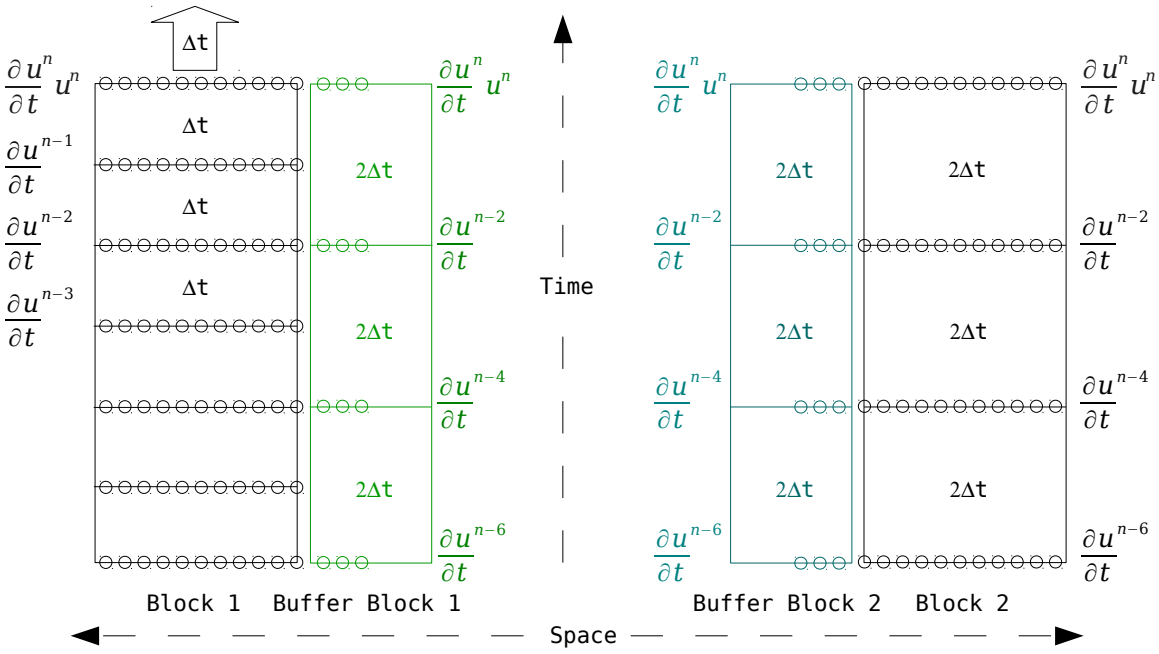


Figure 4-24: Diagram of two blocks in a MTSAB calculation about to be marched from time level n to $n+1$.

of the buffer blocks is 3, implying the use of a seven-point central differencing scheme (i.e., a stencil that uses three points on the left and right, and the central point). The actual information stored at the interior and buffer points is dependent on the type of data required to time-march the solution with Equation (4.26), and is indicated by the mathematical notation next to each horizontal grid line: for example, at the n time level, the interior Block 1 points have both u and $\frac{\partial u}{\partial t}$, but only $\frac{\partial u}{\partial t}$ for $n-1$, $n-2$, and $n-3$. Block 2's interior points, on the other hand, have u and $\frac{\partial u}{\partial t}$ at n, and $\frac{\partial u}{\partial t}$ at $n-2$, $n-4$, and $n-6$. The buffer blocks for both Block 1 and Block 2 contain the same type of solution data at the same temporal location (u and $\frac{\partial u}{\partial t}$ at n, and $\frac{\partial u}{\partial t}$ at $n-2$, $n-4$, and $n-6$), but at different spatial locations.

Figure 4-25 shows the state of Blocks 1 and 2 after being marched from n to $n+1$. Block 1's interior now contains u and $\frac{\partial u}{\partial t}$ data at $n+1$, and $\frac{\partial u}{\partial t}$ at n, $n-1$, and $n-2$. Block 1 and 2's buffer, and Block 2's interior points have not changed. Three of Block 1's interior points are highlighted green. These points require data from Block 1's

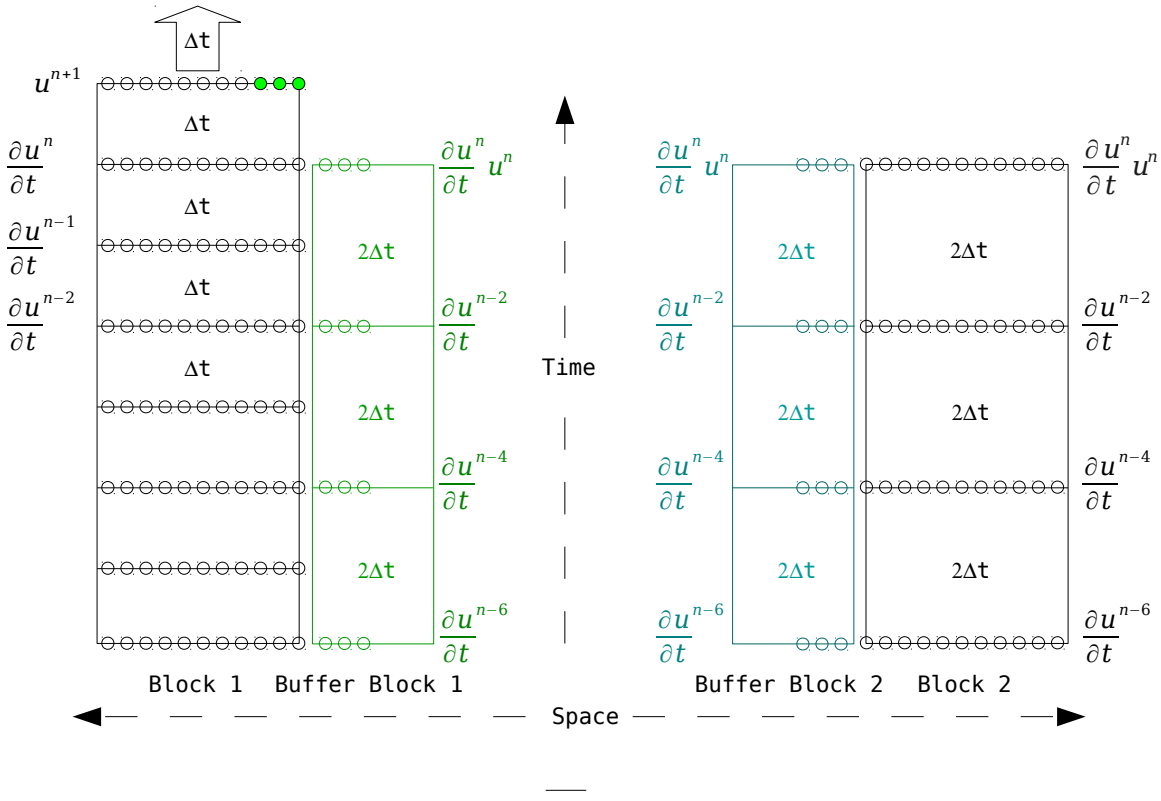


Figure 4-25: Diagram of two MTSAB blocks after being marched from time level n to $n+1$.

buffer block to calculate the spatial derivative of the solution. To calculate this data, the solution in Block 1's buffer blocks is marched forward in time one Δt using a generalized form of the AB4 scheme that uses temporal derivative data from $n-2$, $n-4$, and $n-6$ to advance the solution from n to $n+1$ — this process is shown in Figure 4-26. This buffer block data is used to find the spatial, and then temporal derivative of the green-highlighted interior points, allowing the solution in Block 1's interior to be marched from $n+1$ to $n+2$, shown in Figure 4-27. At the same time, Block 2's interior points, and all the buffer points in the figure are marched from n to $n+2$ using the standard AB4 scheme, completing the time marching process.

Fourier analysis can be applied to the AB4 scheme in the same manner as the

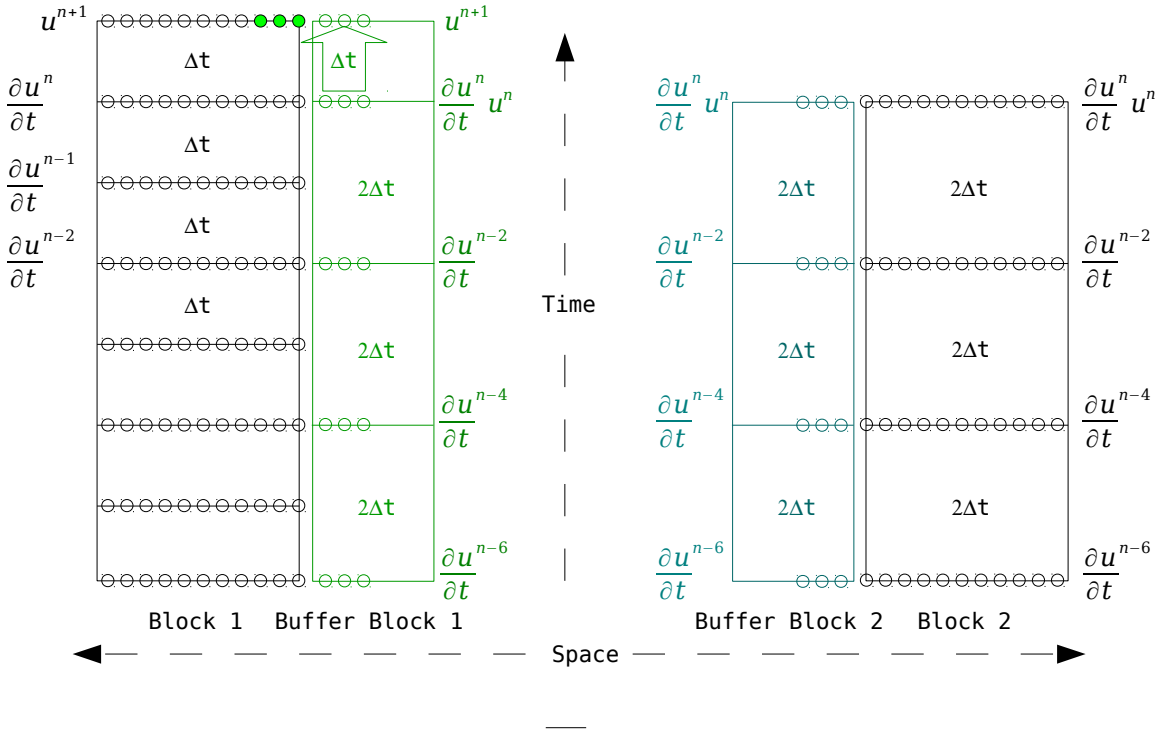


Figure 4-26: Diagram of two MTSAB blocks. Buffer Block 1 has been marched from n to $n+1$ to provide Block 1 with the data necessary to calculate the spatial derivative at the green-filled points with a finite differencing scheme.

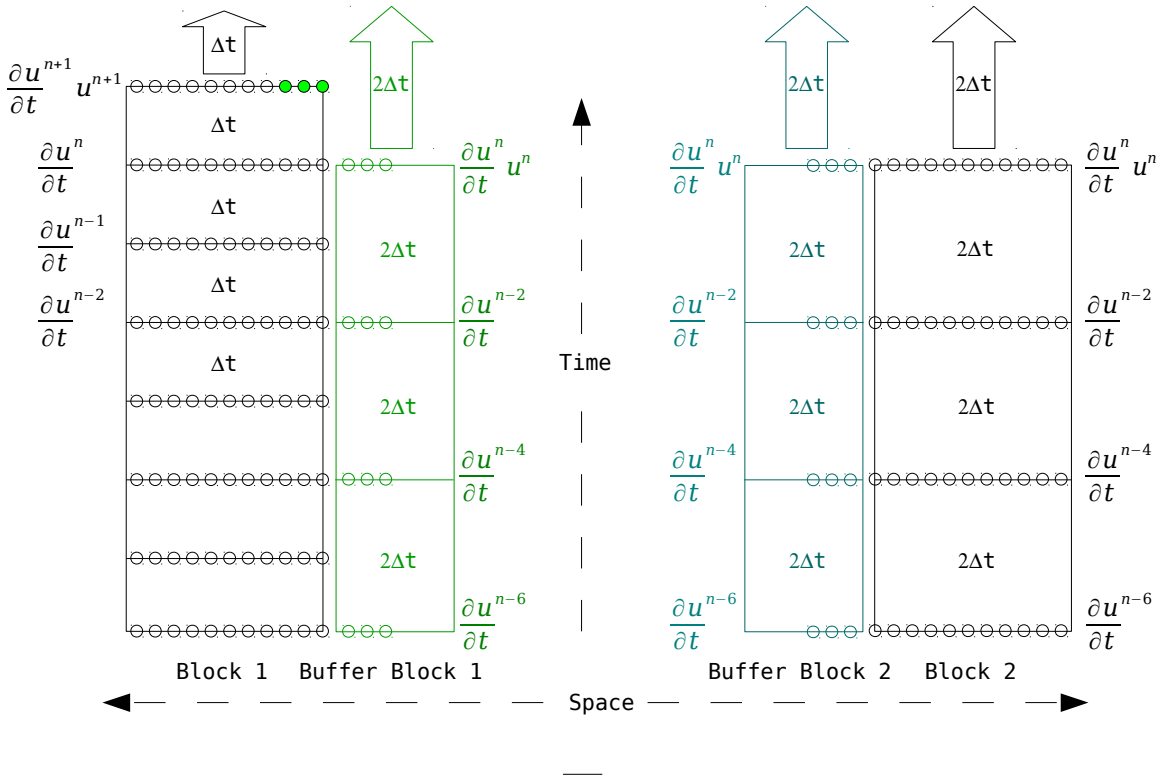


Figure 4-27: Diagram of two MTSAB blocks, both of which about to be marched to the $n+2$ time level.

Runge-Kutta schemes considered previously: the scheme is used to solve the ODE

$$\frac{du}{dt} = -i\omega u \qquad \text{(4.14 repeated)}$$

assuming that the scheme's solution will take the form

$$u_n = G^n u_0 \qquad (4.28)$$

where, like the Runge-Kutta schemes, G is the amplification factor. Equation (4.14) and Equation (4.28) are substituted into Equation (4.26), which gives

$$-\frac{i}{\omega\Delta t}G^4 + \left(b_0 + \frac{i}{\omega\Delta t}\right)G^3 + b_1 G^2 + b_2 G + b_3 = 0, \qquad (4.29)$$

a quartic equation, implying there are actually four different amplification factors (four roots of Equation (4.29)). Figure 4-28 shows the magnitude of these four amplification factors G_1, G_2, G_3, G_4 plotted against $\omega\Delta t$, along with the magnitude of the exact amplification factor. In Figure 4-28, G_1, the desired amplification factor, closely approximates the exact solution for small values of $\omega\Delta t$, much like the Runge-Kutta amplification factors in Figure 4-3. G_3 and G_4 are significantly less 1, indicating that their contribution to the solution will diminish with time. G_2, however, is greater than 1 for $\omega\Delta t$ greater than about 0.4, which will lead to instability.

Because the ODE considered here is linear, the solution obtained by the scheme will be a linear combination of the four amplification factors, i.e.,

$$u_n = a_1 G_1(\omega\Delta t)^n u_0 + a_2 G_2(\omega\Delta t)^n u_0 + a_3 G_3(\omega\Delta t)^n u_0 + a_4 G_4(\omega\Delta t)^n u_0. \qquad (4.30)$$

The values of the constants a_1, a_2, a_3, a_4 will be determined by the method used to overcome the AB4's starting problem, i.e., by what is used for u_{-1}, u_{-2}, u_{-3}. If the

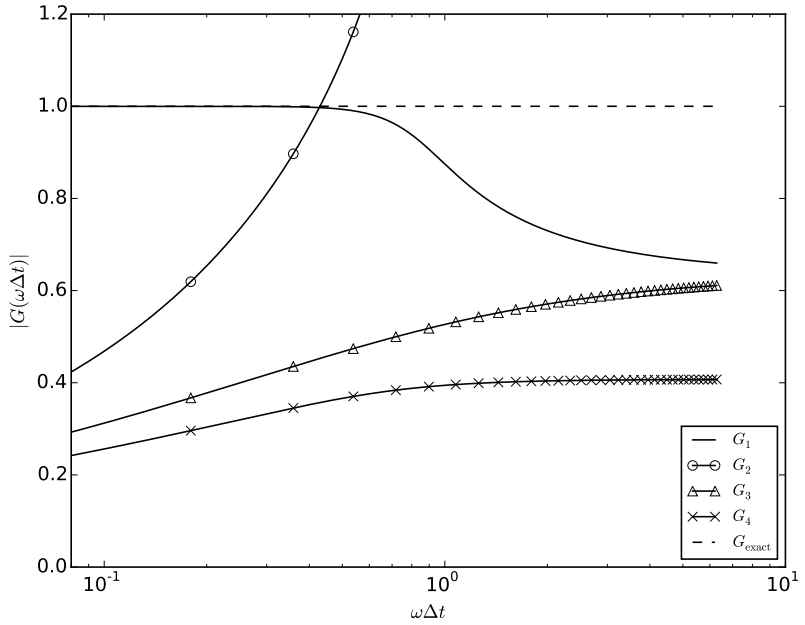


Figure 4-28: AB4 local amplification factors.

exact solution is used, i.e.,

$$u(t) = u_0 e^{-i\omega t} \tag{4.15 repeated}$$

or

$$u_n = u_0 e^{-in\omega\Delta t} \tag{4.31}$$

then the coefficients a_{1-4} must satisfy the system

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} G_1 & G_2 & G_3 & G_4 \\ G_1^2 & G_2^2 & G_3^2 & G_4^2 \\ G_1^3 & G_2^3 & G_3^3 & G_4^3 \\ G_1^4 & G_2^4 & G_3^4 & G_4^4 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4. \end{bmatrix} \tag{4.32}$$

where u_{1-4} are the AB4 scheme's solution using the exact solution for $n < 0$ (i.e., $u_n = u_0 e^{-in\omega\Delta t}$ and $\frac{du_n}{dt} = -i\omega u_n$ for $-3 \leq n \leq 0$). Figure 4-29 compares the

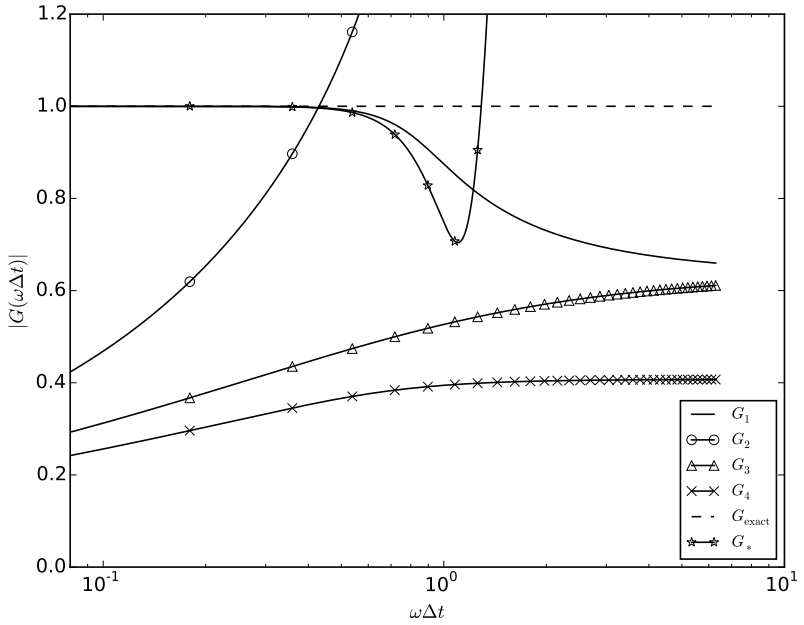


Figure 4-29: AB4 local amplification factors, including G_*, the amplification factor using the exact solution to overcome the starting problem.

magnitude of G_*,

$$G_* = a_1 G_1(\omega\Delta t) + a_2 G_2(\omega\Delta t) + a_3 G_3(\omega\Delta t) + a_4 G_4(\omega\Delta t) \tag{4.33}$$

the combined local amplification factor, with a_{1-4} found from Equation (4.32), to the individual amplification factors G_{1-4}. Interestingly, the $\omega\Delta t$ for which G_* is greater than one (i.e., the stability limit) is much greater than that of G_1, the desired root of the quartic Equation (4.29), indicating that the spurious roots G_2 and G_3 have a positive effect on the stability of the scheme (though not, apparently, on the accuracy, since G_* deviates from the exact amplification factor sooner than G_1).

Figure 4-30 shows the magnitude of a_{1-4} plotted as a function of $\omega\Delta t$. As might be expected, the solution is dominated by the G_1, the desired amplification factor, for small $\omega\Delta t$, with the spurious roots only become significant for large (and, after considering Figure 4-29, unstable) $\omega\Delta t$.

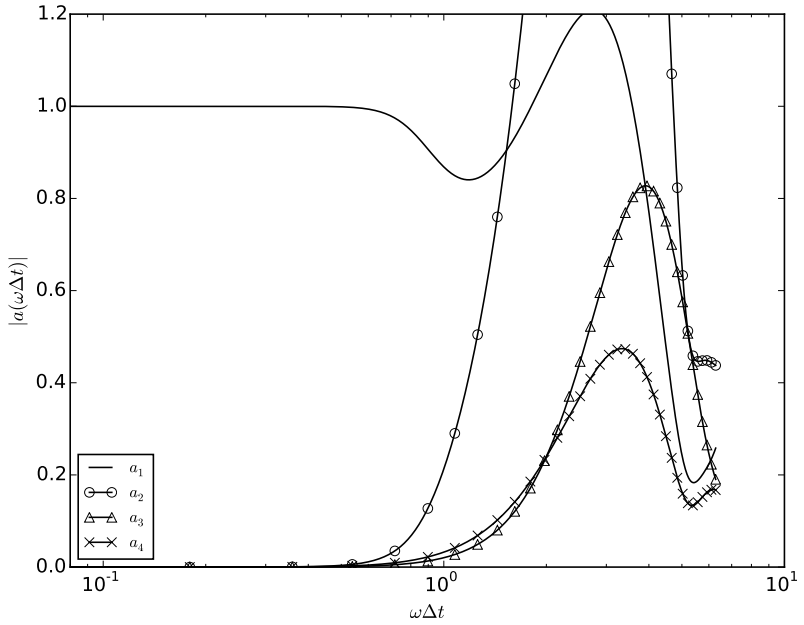


Figure 4-30: Time-independent coefficients multiplying the four amplification factors of the AB4 scheme (see Equation (4.30)).

Like the Runge-Kutta schemes, the local error and convergence rate can be found using Equation (4.17) and Equation (4.18). The local error and convergence rate for G_* are compared to the RK4 scheme in Figure 4-31. The RK4L is considerably more accurate than the AB4, but the computational cost associated with a time step is lower for the AB4, as the AB4 requires only one $\frac{\partial u}{\partial t}$ evaluation, compared to four for the RK4L. Both are fourth-order schemes, so the local error converges at a fifth-order rate.

The global amplification factor can be expressed in terms of a_{1-4}, which is

$$G_{*n} = a_1 G_1(\omega T/n)^n + a_2 G_2(\omega T/n)^n + a_3 G_3(\omega T/n)^n + a_4 G_4(\omega T/n)^n, \qquad (4.34)$$

and displayed in Figure 4-32. The global error and convergence rate are found from Equation (4.19) and Equation (4.20), respectively, and plotted in Figure 4-33. The error converges at a fourth-order rate, as expected. Again, the RK4L scheme is more

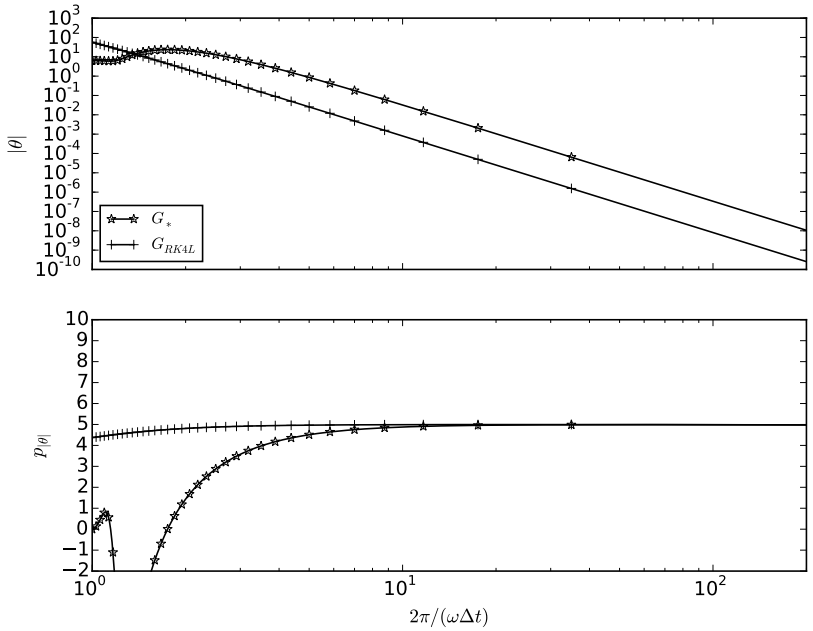


Figure 4-31: AB4 local error and convergence rate compared to the RK4 scheme.

accurate.

Result from two inviscid MTSAB-EVA verification test cases are reproduced here, presented previously in [80]. The first test case (Case II in [80]) investigates the local error, i.e., the PDE code is marched just one time step of varying size. The initial condition is of the form

$$\phi(x,y) = \overline{\phi} + \widetilde{\phi}\exp\left(-\frac{\log(2)}{0.25}\left[(x-x_0)^2 + (y-y_0)^2\right]\right) \tag{4.35}$$

with the constants given in Table 4.2 (notice that the peak of each Gaussian is located in a different grid block). An estimate of the wavelength of the disturbances in the solution can be taken as twice the half-width of the Gaussians in Equation (4.35), namely, $2\cdot\sqrt{0.25} = 1$, giving an approximate wavenumber of $\widetilde{=}2\pi$. A two-dimensional, four-block grid is used, and is shown in Figure 4-34. The grid consists of a uniform Cartesian portion in the interior extending from -1 to 1 with spacing of 0.01 in both

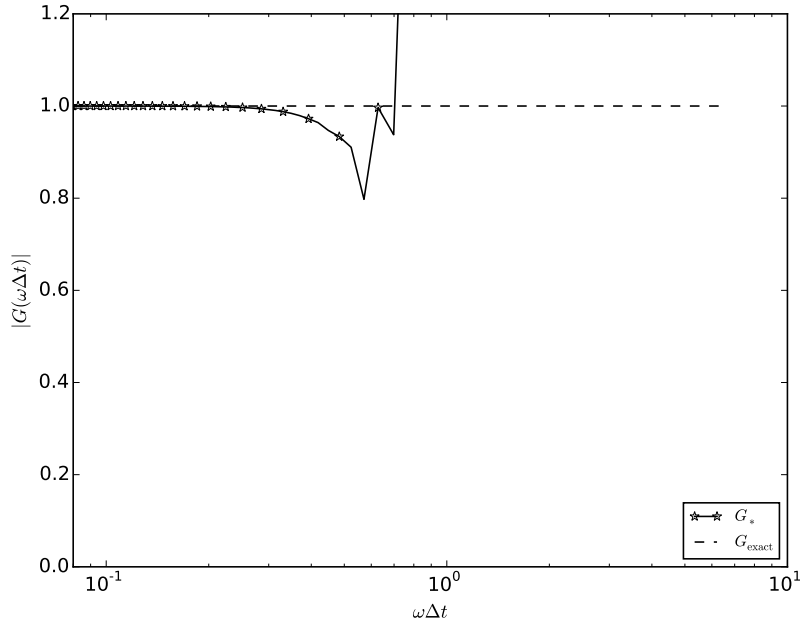


Figure 4-32: AB4 global amplification factor

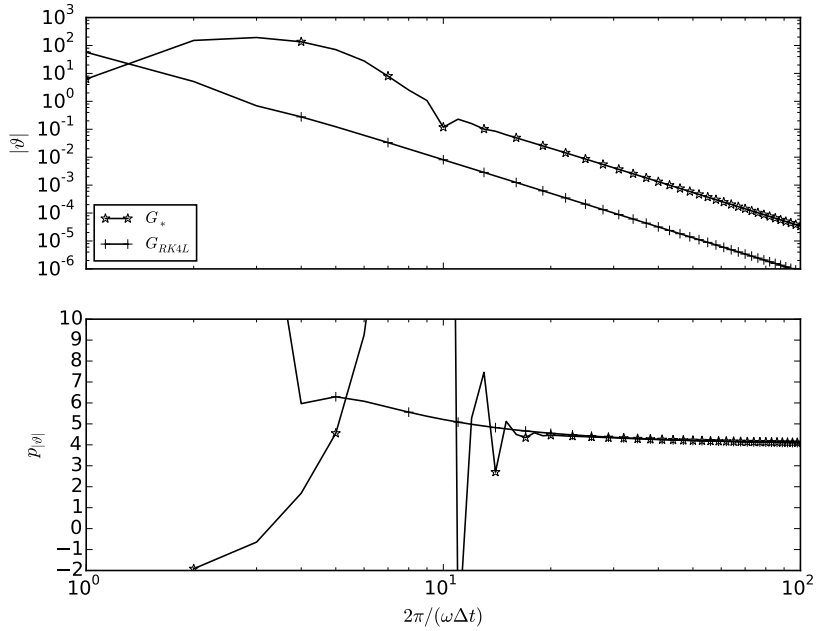


Figure 4-33: AB4 global error and convergence rate.

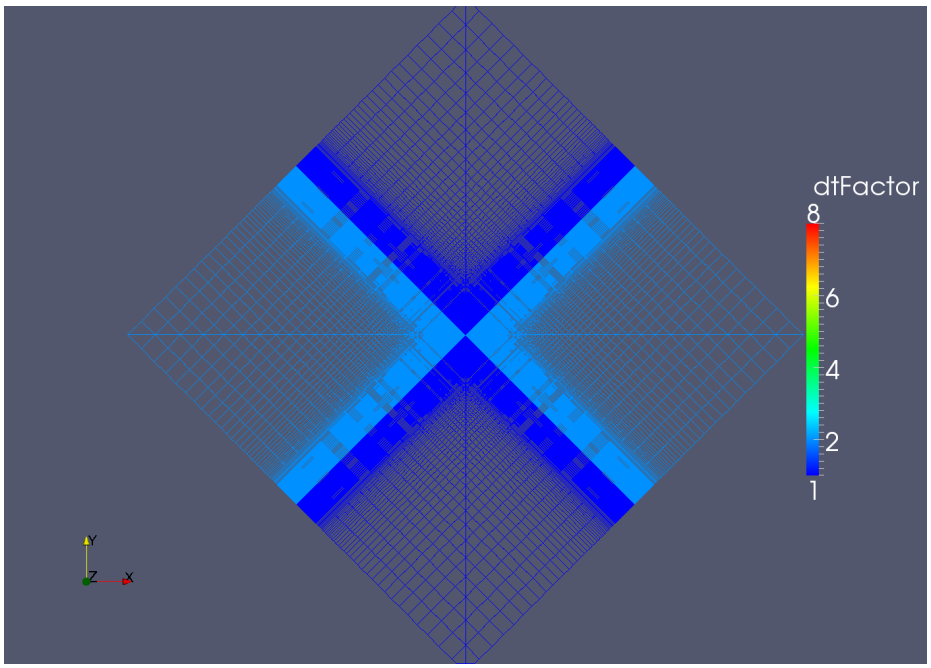


Figure 4-34: MTSAB time step factors for the local error test case.

ϕ	$\overline{\phi}$	$\widetilde{\phi}$	x_0	y_0
σ	1.	0.1	0.25	0.125
u	0.01	0.0001	0.125	-0.25
v	0.02	0.0002	0.25	-0.125
p	$\frac{1}{\gamma}$	$\frac{1}{10\gamma}$	-0.125	-0.25

Table 4.2: Initial condition parameters for MTSAB local error test case (Case II from conference paper).

directions, and a stretched region on the exterior, intended to prevent the solution at the boundaries from interacting with the solution at the center of the grid (the BASS/EVA solution comparison is restricted to the inner uniform Cartesian grid points). The grid spacing, combined with the approximate wavenumber $\widetilde{k}$, gives an approximate points-per-wavelength value of 100, hopefully ensuring that the spatial differencing scheme's contribution to the error will be low.

The maximum propagation speed based on mean-flow quantities for the initial flow in Equation (4.35) and Table 4.2 is $c = \sqrt{\frac{\gamma\overline{p}}{\overline{\rho}}} + \sqrt{\overline{u}^2 + \overline{v}^2} \approx 1.022$, which leads to an approximate disturbance frequency $\widetilde{\omega} = c\widetilde{k} \approx 6.424$. In Figure 4-34, the grid

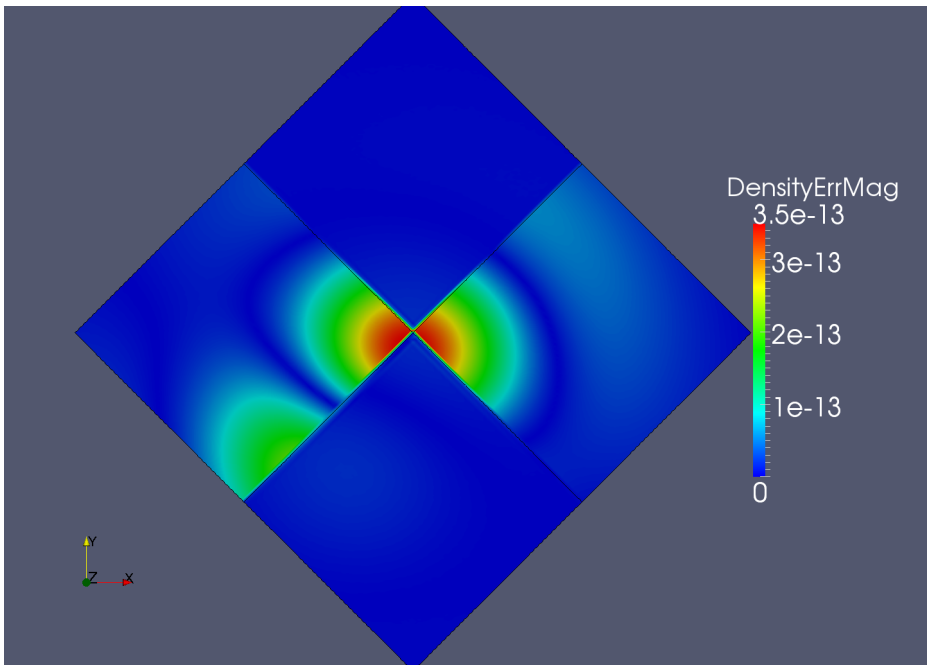


Figure 4-35: ρ error plot for the local error test case.

blocks are colored by the time step factor, defined as the ratio of the local time step and the smallest time step used in the calculation (so the left and right blocks in Figure 4-34 use a time step size twice as large as the top and bottom blocks). The maximum Δt used by the MTSAB scheme ranged from 0.000125 to 0.0025, giving a time-steps-per-period range of $2\pi/(\widetilde{\omega}\Delta t)$ extending from 390 to 7800, which, after inspection of Figure 4-31, leads one to believe that the error will be quite low for these time step sizes.

Figures 4-35 and 4-36 show the local error for the ρ and ρu flow variables, respectively, for one of the BASS calculations. As might be expected, the error is significantly higher in the blocks using the larger time step size, but the accuracy of the solution does not appear to be significantly affected by the presence of the interior block boundaries (where the generalized Adams-Bashforth scheme is used).

The local $l_{\max}$ error and convergence rate results are shown as a function of steps-per-period (using the maximum Δt) in Figure 4-37. The error is very low for the

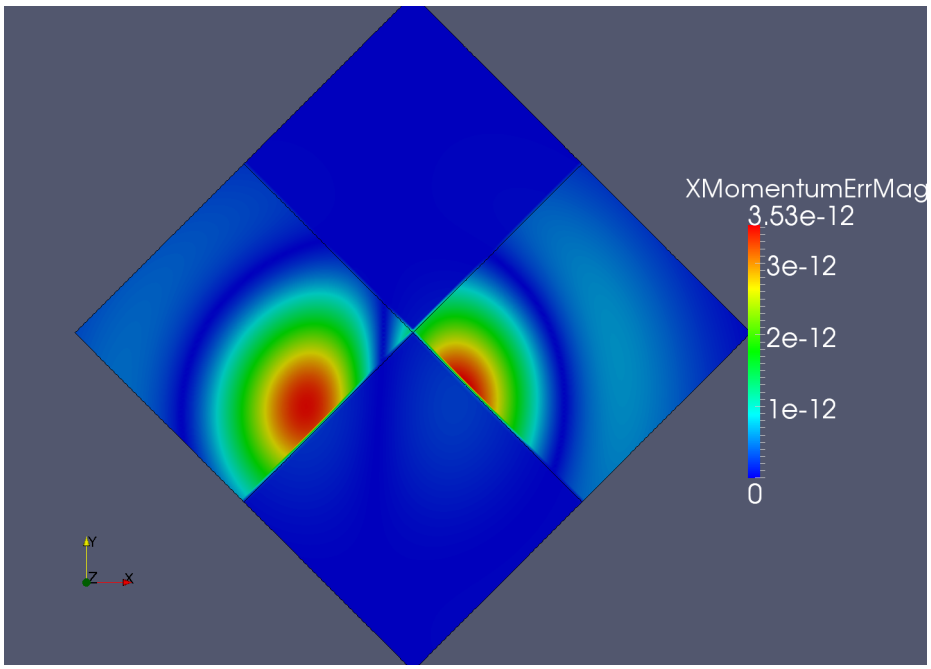


Figure 4-36: ρu error plot for the local error test case.

range of steps-per-period considered. For the large time step sizes, the convergence rate is near five (the expected value), but gradually is reduced as Δt is refined as the error becomes less sensitive to the time step size and is likely overwhelmed by round-off error.

The second MTSAB test case (Case III from [80]) investigated the global error, i.e., the MTSAB scheme was used to march the initial flow to a constant final time level with an increasing number of steps. The grid, shown in Figure 4-38, was identical to that of the previous test case, but was not rotated with respect to Cartesian axes and used a smaller grid spacing of $\Delta x = 0.005$. As seen in Figure 4-38, a larger range of time step factors was used in this case. The initial condition followed Equation (4.35) but used different parameters, shown in Table 4.3. All of the Gaussian peaks were placed in the block using the largest time step size, where the error was expected to be the most significant. Since the mean values and shape of the Gaussians is identical to the previous case, $\widetilde{k}$ is also unchanged, and the approximate points-per-wavelength

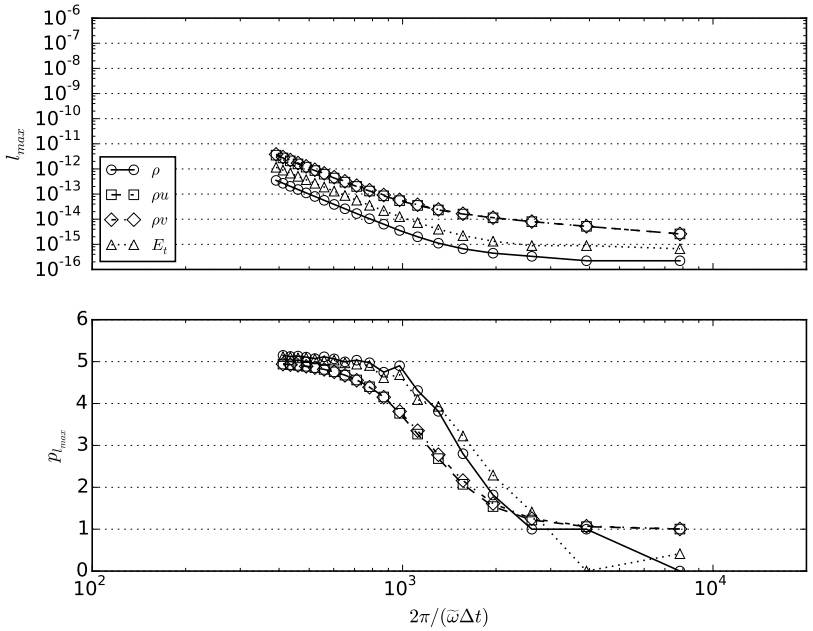


Figure 4-37: $l_{\max}$ error and convergence rate for the local error test case. p was calculated using Equation (3.8)

ϕ	$\overline{\phi}$	$\widetilde{\phi}$	x_0	y_0
σ	1.	0.1	0.25	0.125
u	0.01	0.0001	0.25	0.125
v	0.02	0.0002	0.25	0.125
p	$\frac{1}{\gamma}$	$\frac{1}{10\gamma}$	0.25	0.125

Table 4.3: Initial condition parameters for MTSAB global error test case (Case III from conference paper).

is 200.

The maximum time step size used by the MTSAB scheme ranged from 0.00025 to 0.005, which, combined with $\widetilde{\omega}$, gives a steps-per-period range of 200 to 3900. Analogous to the previous case, Figure 4-33 shows that the temporal error should be quite low for these Δt values.

Figures 4-39 and 4-40 show the global error for the ρv and E_{total} flow variables, respectively, for one of the BASS calculations. Similar to the previous test case, the error is low, but significantly larger in the block with the largest time step size. Unlike

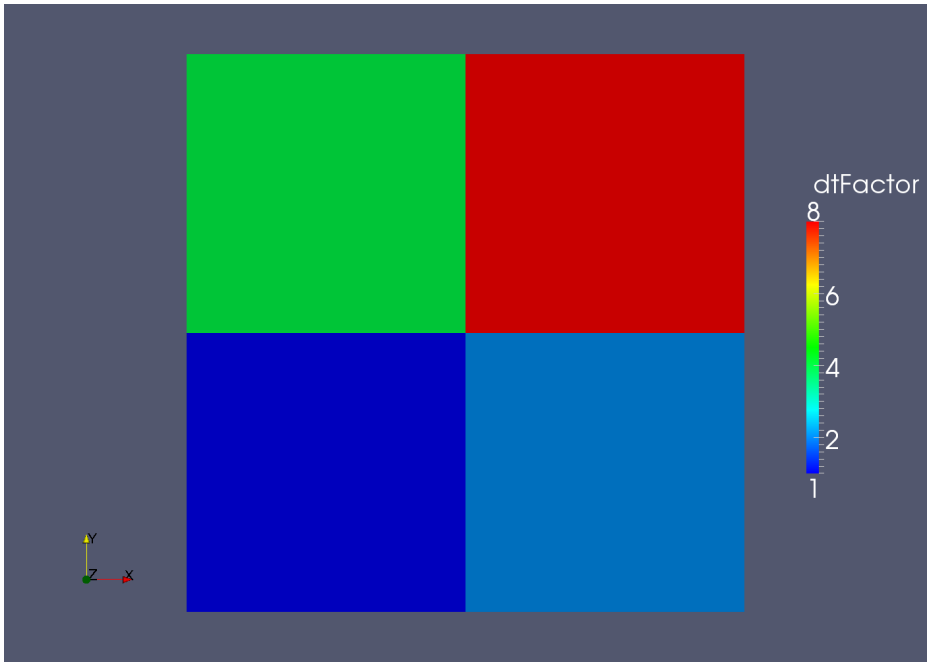


Figure 4-38: MTSAB time step factors for the global error test case.

the previous test case, there is a slight error spike at the block boundaries.

Figure 4-41 shows the global l_2 error and convergence rate for the present test case. As in the local error verification results, the error convergence is fairly uniform for the larger-Δt runs, and drops as the time step is refined, likely due to round-off. The convergence rate is very close to 4, indicating the MTSAB scheme as implemented is indeed fourth-order accurate.

4.5 Investigation of the accuracy of an approach to differencing across grid singularities

To facilitate complex geometries, structured-grid PDE codes often solve their governing equations in generalized curvilinear coordinates. For example, if a code works

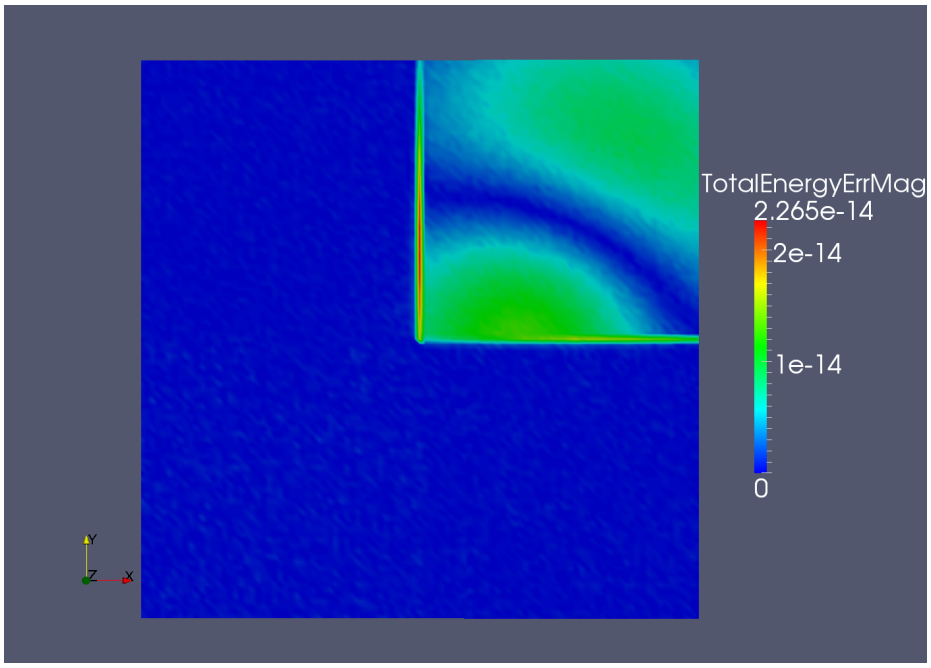


Figure 4-39: ρv error plot for the global error test case.

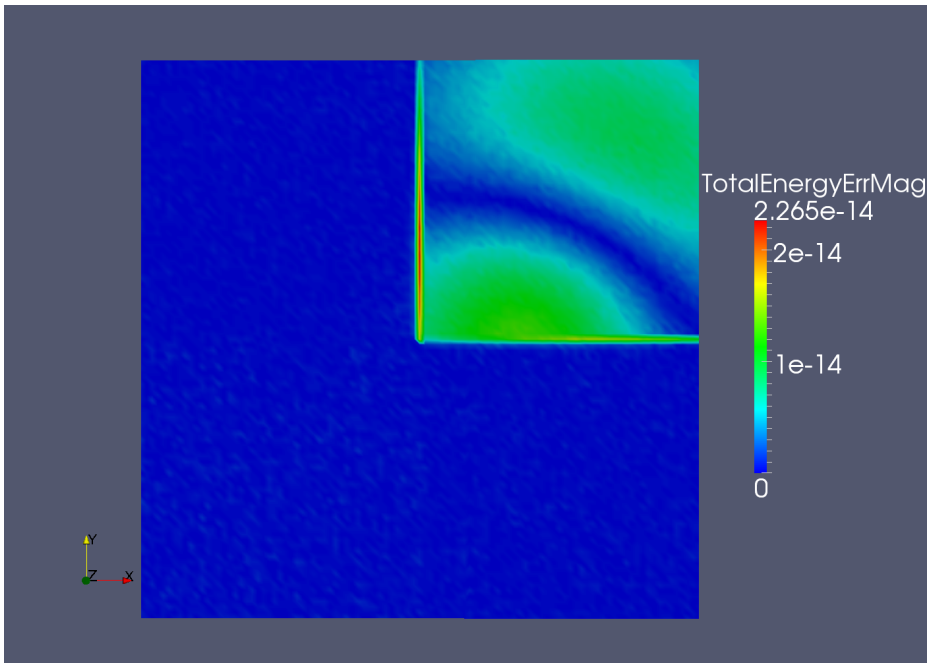


Figure 4-40: E_{total} error plot for the global error test case.

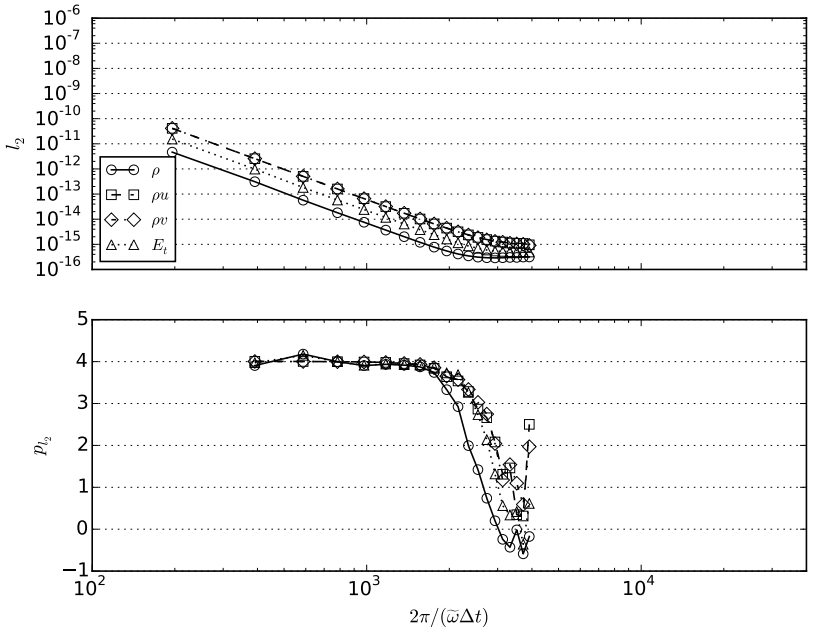


Figure 4-41: l_2 error and convergence rate for the global error test case. Equation (3.8) was used to calculate p.

with a two-dimensional conservation law,

$$\frac{\partial \vec{Q}}{\partial t} + \frac{\partial \vec{E}(\vec{Q})}{\partial x} + \frac{\partial \vec{F}(\vec{Q})}{\partial y} = 0 \tag{4.36}$$

then a transformation of the form

$$\begin{aligned} \xi &= \xi(x, y) \\ \eta &= \eta(x, y) \end{aligned} \tag{4.37}$$

could be used, which transforms the governing equation of Equation (4.36) into

$$\frac{\partial \vec{Q}}{\partial t} + \frac{\partial \xi}{\partial x}\frac{\partial \vec{E}}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial \vec{E}}{\partial \eta} + \frac{\partial \xi}{\partial y}\frac{\partial \vec{F}}{\partial \xi} + \frac{\partial \eta}{\partial y}\frac{\partial \vec{F}}{\partial \eta} = 0. \tag{4.38}$$

Notice how the derivatives of the fluxes E and F are taken with respect to the coordinates in the transformed space (ξ and η), but the derivatives of the transformation coordinates themselves (e.g. $\frac{\partial \xi}{\partial x}$, the grid metrics) are taken with respect to the original Cartesian coordinates. The grid metrics can be related to the derivatives of x and y in the transformed space through the expressions

$$\begin{aligned}
\frac{\partial \xi}{\partial x} &= J\frac{\partial y}{\partial \eta} \\
\frac{\partial \xi}{\partial y} &= -J\frac{\partial x}{\partial \eta} \\
\frac{\partial \eta}{\partial x} &= -J\frac{\partial y}{\partial \xi} \\
\frac{\partial \eta}{\partial y} &= J\frac{\partial x}{\partial \xi}
\end{aligned} \tag{4.39}$$

where J is the metric Jacobian,

$$J = \frac{1}{\frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \xi}}. \tag{4.40}$$

Further geometric flexibility can be had by breaking up a computational domain into multiple subdomains, or blocks. This decomposition may lead to grid singularities, i.e., points in the grid that contain more or less than the usual number of neighboring points (four for two dimensions, six for three dimensions). Figure 4-42 is an example of a two-dimensional grid singularity. The grid lines are colored by block, with the block boundaries indicated by the heavy black lines. The singular point is the circled point at the intersection of the three blocks. At this location, the application of a finite differencing stencil is no longer straight-forward, as the singular point's ξ and η neighbors are not uniquely defined. Thus a different approach will be needed to evaluate both the flux derivatives in Equation (4.38) and the grid metrics in Equations (4.37) and (4.40).

The strategy for differencing across grid singularities used in this work is shown

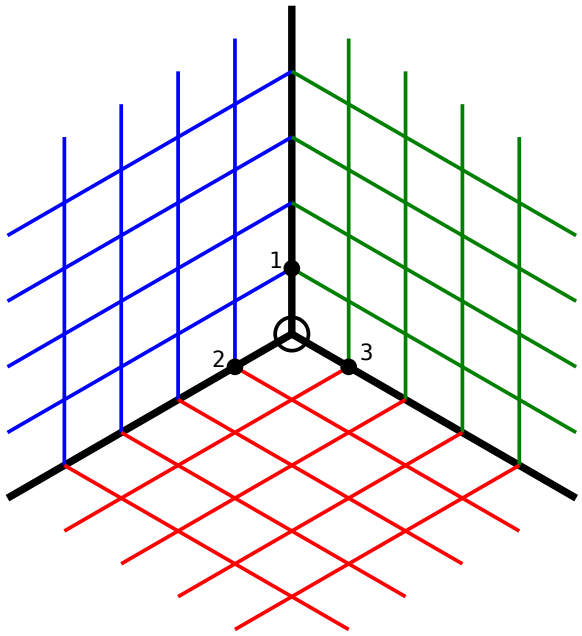


Figure 4-42: Two-dimensional three-way grid singularity.

in Figure 4-43. When a finite differencing stencil encounters a grid singularity, it takes data from all possible branches running through the singular point, averages this data, then uses this average to fill in the needed stencil information. Figure 4-43 shows how this procedure is used with second-order central differencing to calculate the spatial derivatives at the singular point from the perspective of the blue grid block in Figure 4-42. When finding, for example, $\frac{\partial F}{\partial \xi}$ using the second-order central stencil, data from one point on either side of the singular point in the ξ direction is needed. The $+\xi$ direction is unambiguous: the value of Q at point 1 is used. In the $-\xi$ direction there are two choices: down and to the left (point 2), and down and to the right (point 3). In this work, the data from points 2 and 3 is averaged, and this averaged value is used to complete the stencil.

To investigate the accuracy of the above approach, a two-dimensional grid with five blocks and two singular points was constructed using the GridPro [81] grid generation program. The inviscid two-dimensional version of BASS was used as the PDE solver, with EVA providing the reference solution. As mentioned earlier in this work, EVA is

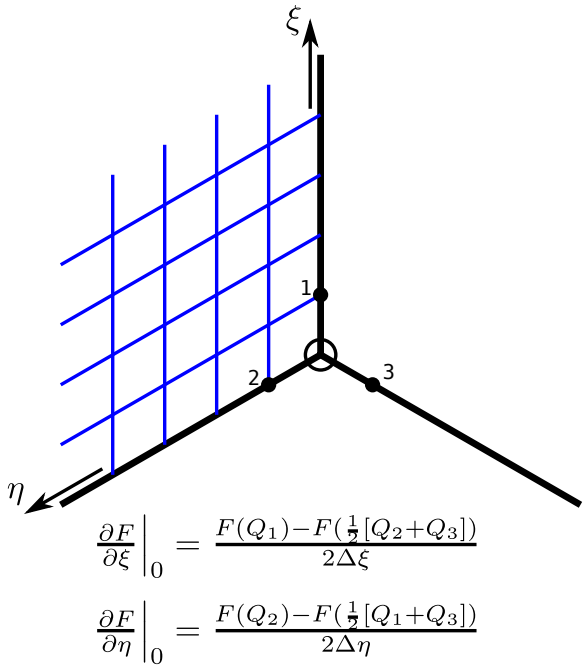


Figure 4-43: Differencing across a three-way grid singularity.

particularly well-suited for this task, since it evaluates spatial derivatives analytically, and is thus unaffected by the presence of grid singularities.

As in the previous spatial verification test cases, a series of progressively finer grids is needed to observe how the discretization error behaves as the grid spacing is reduced. The coarsest singular grid in the series used here is displayed in Figure 4-44, with the grid lines colored by grid block. Each grid block contains 21^2 points, and the complete grid extends from 0 to 1 in both Cartesian directions. Nineteen additional grids were constructed from that shown in Figure 4-44 using Akima interpolation [82] on a per-block basis. A series of uniform grids with approximately the same grid spacing as the singular grid were also constructed, for comparison purposes.

Two test cases are presented here (both previously described in [83]), with the first intended to focus on the three-way grid singularity. The initial condition for this case consisted of Gaussian pulses superimposed on a uniform mean flow for each of

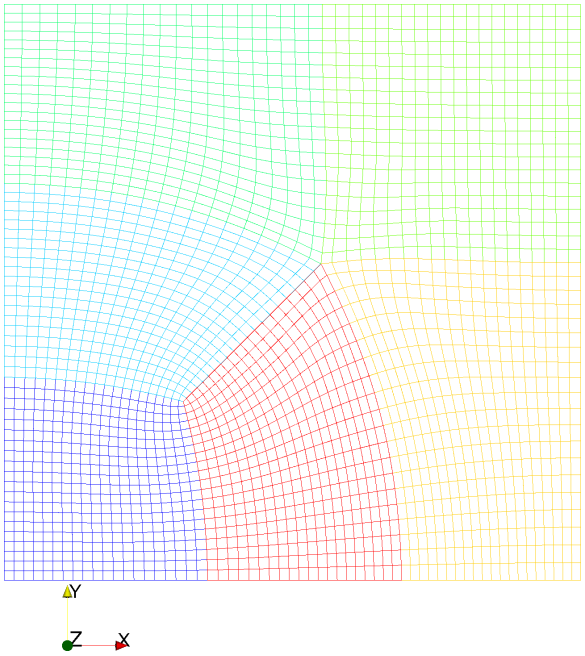


Figure 4-44: The coarsest singular grid used in the grid singularity test cases. The grid lines are colored by grid block, and each block contains 21^2 grid points. The grid extends from 0 to 1 in both Cartesian directions.

the flow variables:

$$\begin{aligned}
\sigma\left(x,y,t_0\right) &= 1.0 + 0.05\exp\left(\frac{-\log(2)}{0.05^2}\left[(x-0.300567)^2+(y-0.320005)^2\right]\right)\\
u\left(x,y,t_0\right) &= 0.1 + 0.001\exp\left(\frac{-\log(2)}{0.05^2}\left[(x-0.320567)^2+(y-0.320005)^2\right]\right)\\
v\left(x,y,t_0\right) &= 0.025 + 0.002\exp\left(\frac{-\log(2)}{0.05^2}\left[(x-0.300567)^2+(y-0.300005)^2\right]\right)\\
p\left(x,y,t_0\right) &= 0.7142857 + 0.01\exp\left(\frac{-\log(2)}{0.05^2}\left[(x-0.320567)^2+(y-.300005)^2\right]\right)
\end{aligned} \tag{4.41}$$

Each of the peaks of the Gaussian pulses were placed at different spatial locations, but equidistant from the three-way singularity — this is shown graphically in Figure 4-45. An approximate wavelength of the disturbances in Equation (4.41) is taken to be $\widetilde{k} = 2\pi/(2\cdot 0.05) \approx 62.83$. This, combined with a grid spacing range of 0.0008333 to

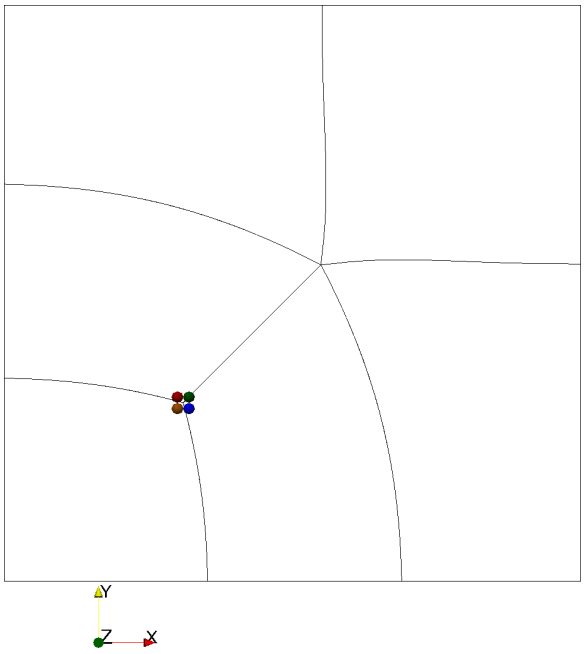


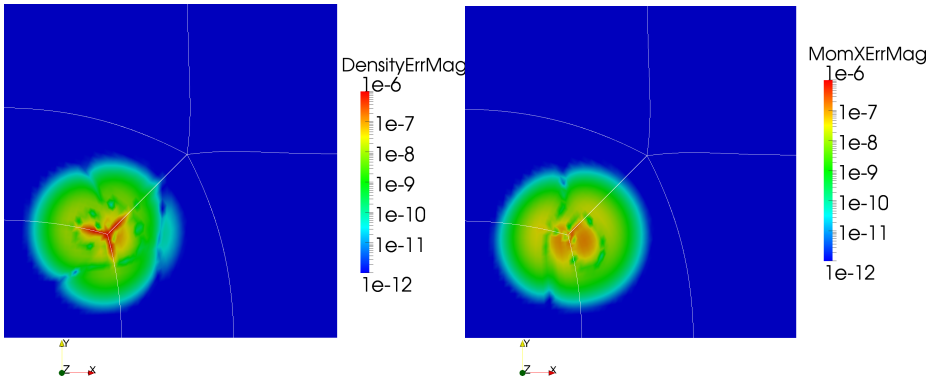
Figure 4-45: Location of Gaussian peaks for the three-way grid singularity test case. The red, green, brown, and blue spheres correspond to the σ, u, v, and p variables, respectively.

0.01667, give an approximate points-per-wavelength extending from 6 to 120.

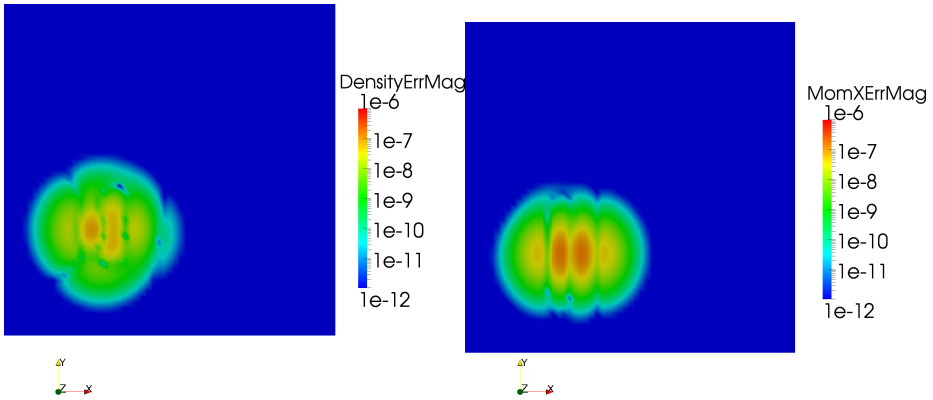
The initial condition was marched with the BASS code and RK7S time-marching scheme to a final time level of $t = 0.0005$. Both the explicit second-order and DRP schemes were used, along with the Giles [76] non-reflecting boundary condition. The EVA tool was used to find a reference solution on the coarsest grid, which was then compared to the BASS solutions to evaluate the error.

Figure 4-47 compares the error of the BASS runs using the fifth-coarsest uniform and singular grids and the E_2 scheme. The error for the singular grid calculation is significantly higher than the uniform calculation (notice the log-log scale). The density error for the singular grid is strongly concentrated along the grid lines containing the three-way singular point — the error for the x-momentum variable does not appear to "collect" at the singular grid lines with the same intensity.

Figure 4-47 shows the error convergence rate for the density and x-momentum

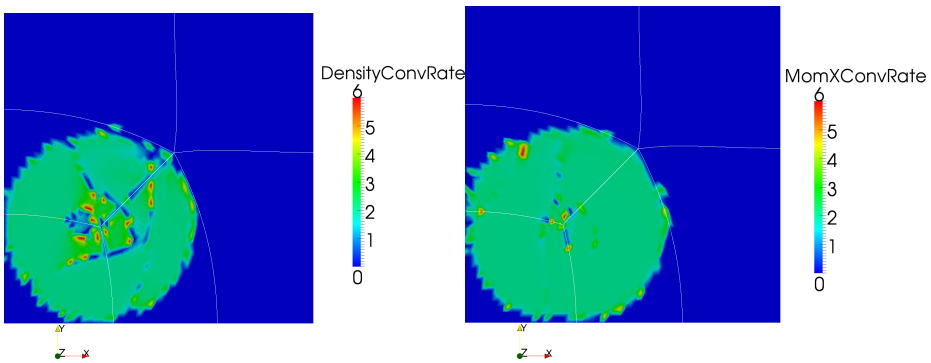


(a) ρ, singular grid (b) ρu, singular grid

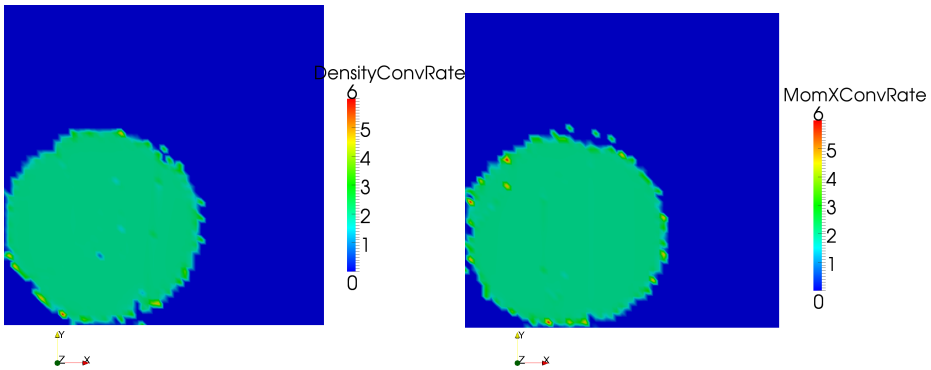


(c) ρ, uniform grid (d) ρu, uniform grid

Figure 4-47: Three-way singularity test case error magnitude for the fifth-coarsest singular and uniform grids and E_2 scheme.



(a) ρ, singular grid (b) ρu, singular grid

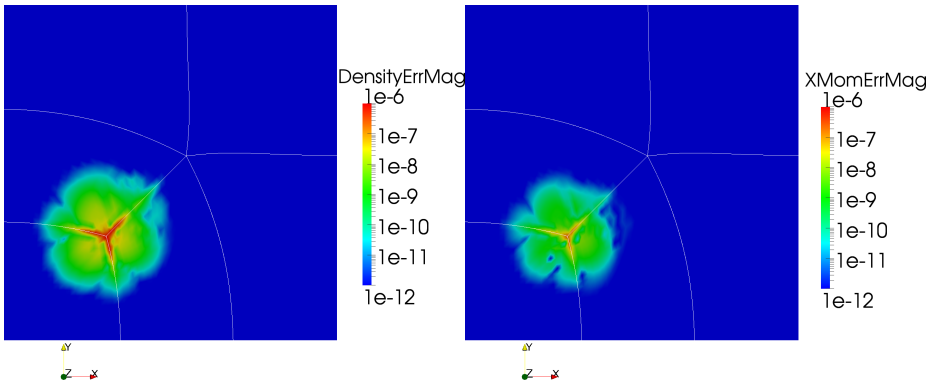


(c) ρ, uniform grid (d) ρu, uniform grid

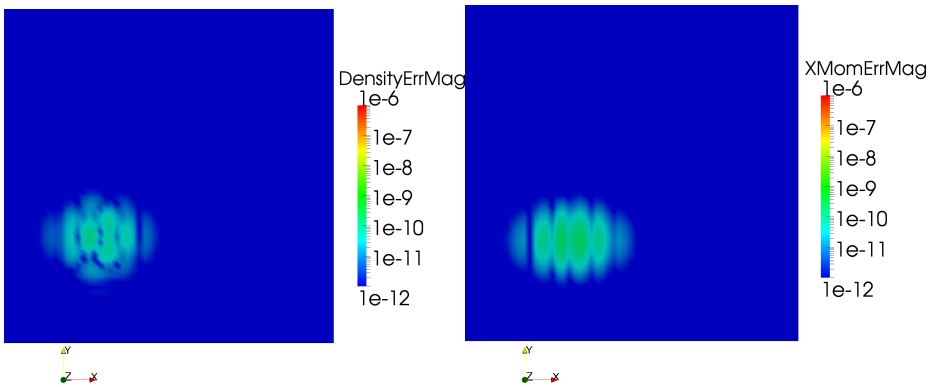
Figure 4-49: Three-way singularity test case error convergence rate for the fifth-coarsest singular and uniform grids E_2 scheme.

flow variables, again for the fifth-coarsest singular and uniform grid and the explicit second-order scheme. The uniform grid results for both flow variables appear to converge at approximately second-order rate throughout the area influenced by the Gaussian pulses. For the singular grid, a significant region appears to converge at a second-order rate, with much of the low-order convergence behavior localized at the singular grid lines.

Figure 4-51 shows the error magnitude of ρ and ρu for the present test case with the DRP scheme. The difference between the uniform and singular results is more striking for this higher-order scheme, with the singular error being five to sixth orders-of-magnitude higher than the uniform. The largest error regions are strongly associated with the grid lines containing the singular point. Interestingly, the DRP



(a) ρ, singular grid (b) ρu, singular grid

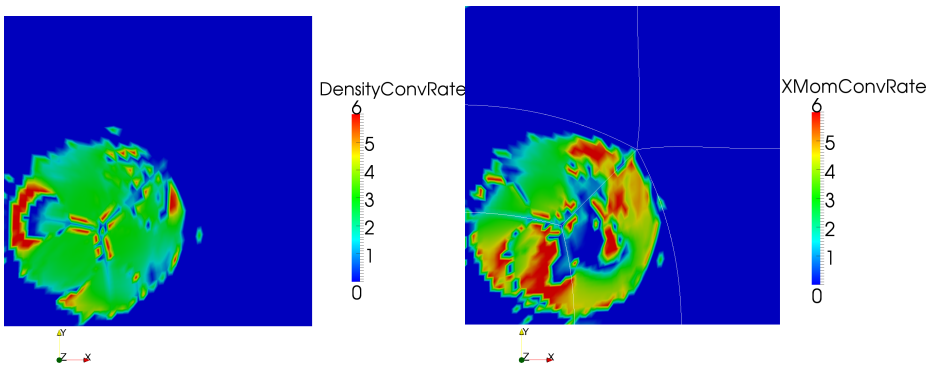


(c) ρ, uniform grid (d) ρu, uniform grid

Figure 4-51: Error magnitude for the fifth-coarsest singular and uniform grids, DPR scheme, and three-way singularity test case.

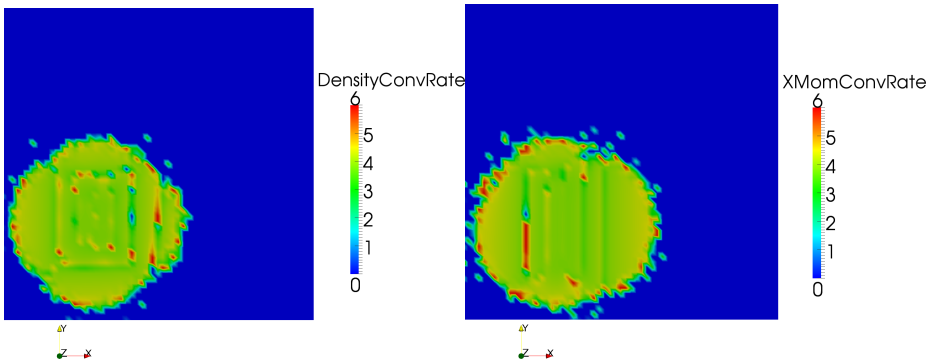
result's overall error footprint is smaller than the E_2 scheme, despite the former's larger stencil size.

Figure 4-53 shows the DRP error convergence rate for the density and x-momentum flow variables, again for the fifth-coarsest singular and uniform grid. The uniform grid results show most of the error converging at a fourth-order rate, the formal order-of-accuracy of the DRP scheme. The convergence rates for the singular grid are more interesting. For both the density and x-momentum error, there appear to be regions of low-order convergence along the singular grid lines — though the convergence rate everywhere near the singularity is less than fourth-order, unlike the corresponding E_2 results. There are also larger areas of high-order convergence well away from the singular point, likely a consequence of the very low magnitude of the error in these



(a) ρ, singular grid

(b) ρu, singular grid



(c) ρ, uniform grid

(d) ρu, uniform grid

Figure 4-53: Convergence rate of the error for the fifth-coarsest singular and uniform grids DRP scheme, and three-way singularity test case.

regions (since the convergence rate is essentially the "slope" of the error, any small "bumpiness" due to round-off/machine errors could lead to apparently high convergence rates). This does not explain the super-convergent behavior along the singular lines, however. It may be that error generated by the singularities is propagated into regions of the grid where the DRP scheme is not differencing across the singular point and is thus less influenced by the singularity. If these disturbances have a wavenumber somewhere in the "optimized" portion of the DRP's Fourier spectrum, one would expect the convergence rate to be relatively high.

Figures 4-54 and 4-54 show the $l_{\max}$ error and convergence rate results for the E_2 and DRP schemes, respectively. As the error and convergence rate contour plots implied, the uniform results are quite good, with both schemes converging at the

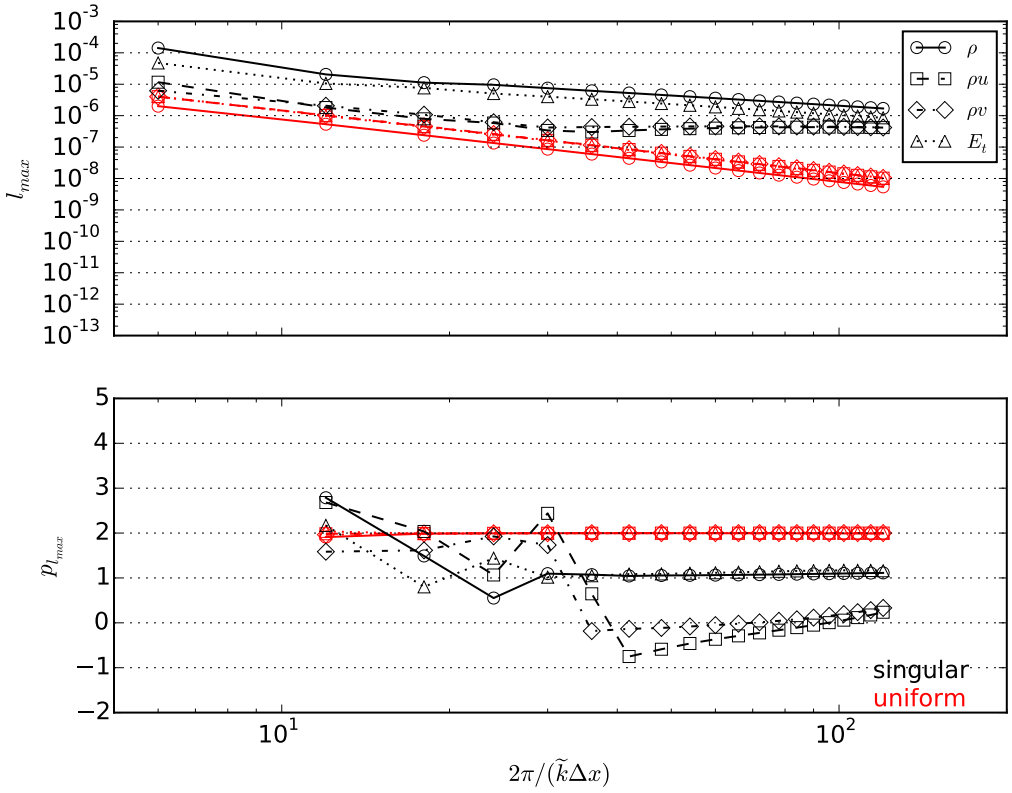


Figure 4-54: $l_{\max}$ error and convergence rate for the three-way grid singularity case and E_2 scheme. The convergence rate was found using Equation (3.8).

expected rate for this grid type. The singular grid convergence rate is markedly degraded for both spatial differencing schemes, showing something between zeroth and first order. In fact, there is little difference between the E_2 and DRP $l_{\max}$ results — both return high error and lower-than-designed convergence.

The second test case shifts focus to the five-way singularity in Figure 4-44. The parameters were identical to the previous three-way case, with two exceptions. First, the Gaussian peaks in the initial condition were moved closer to the five-way singu-

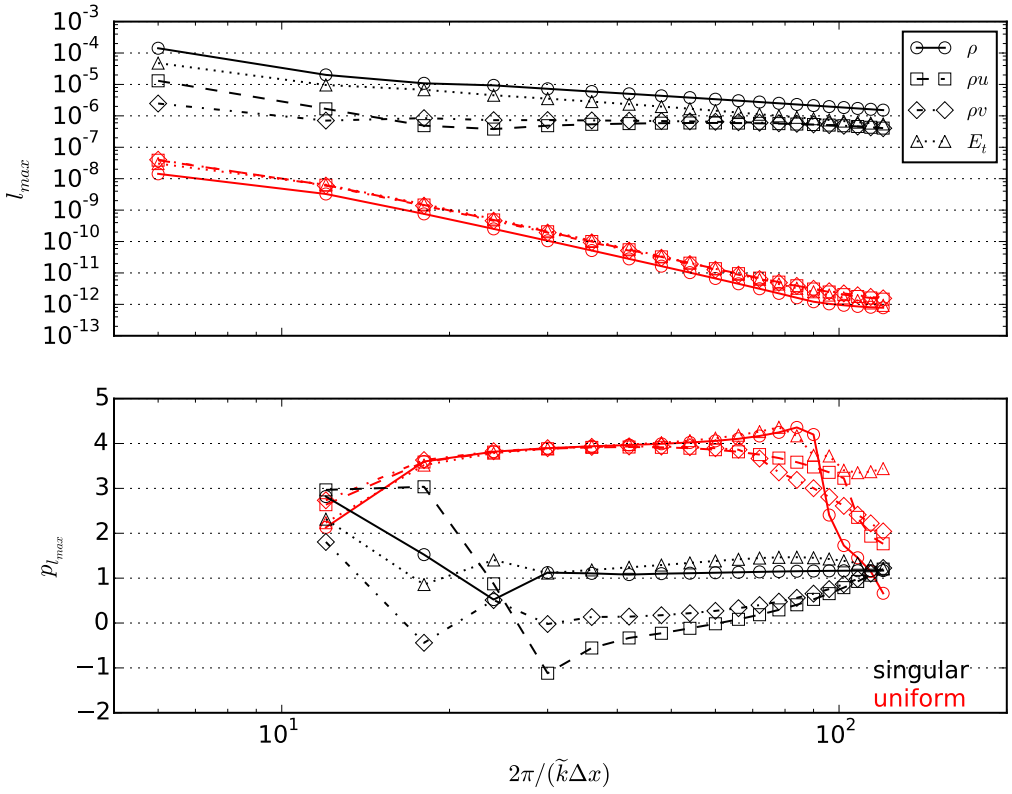


Figure 4-55: $l_{\max}$ error and convergence rate for the three-way grid singularity case and DRP scheme. The convergence rate was found using Equation (3.8).

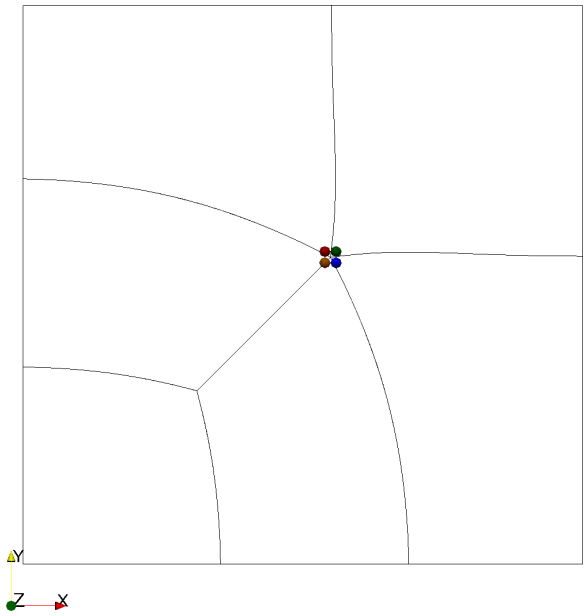


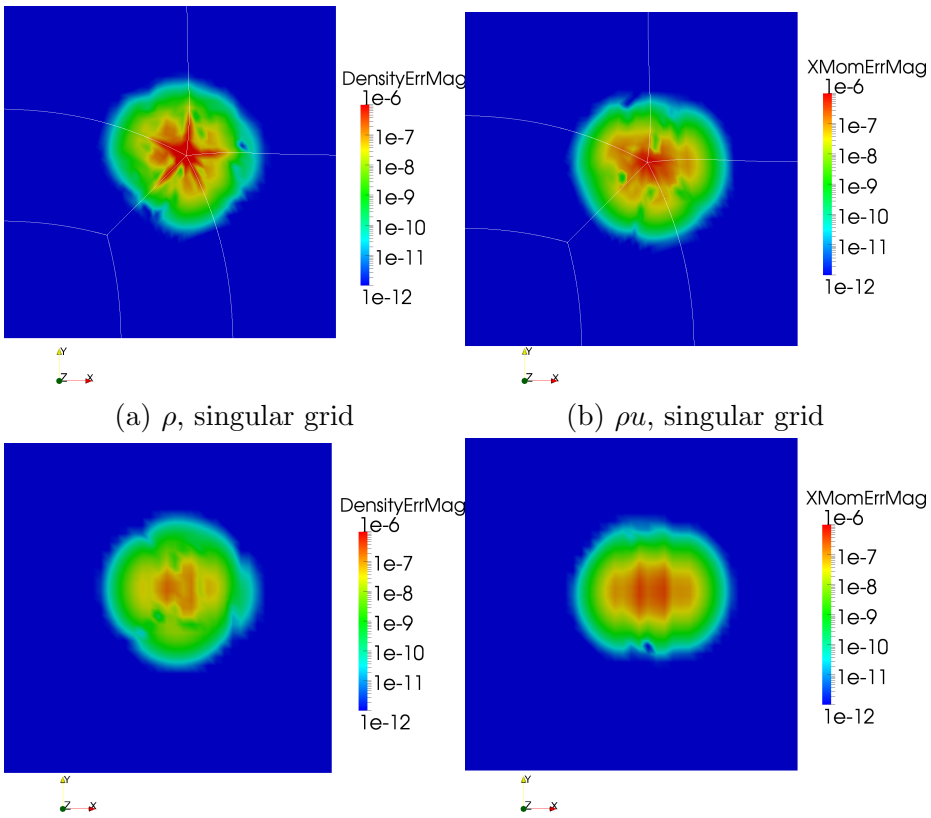
Figure 4-56: Location of Gaussian peaks for the five-way grid singularity test case. The red, green, brown, and blue spheres correspond to the σ, u, v, and p variables, respectively.

larity

$$\begin{aligned}
\sigma(x,y,t_0) &= 1.0 + 0.05\exp\left(\frac{\log(2)}{0.05^2}\left[(x-0.539358)^2+(y-0.559358)^2\right]\right)\\
u(x,y,t_0) &= 0.1 + 0.001\exp\left(\frac{\log(2)}{0.05^2}\left[(x-0.559358)^2+(y-0.559385)^2\right]\right)\\
v(x,y,t_0) &= 0.025 + 0.002\exp\left(\frac{\log(2)}{0.05^2}\left[(x-0.539358)^2+(y-0.539358)^2\right]\right)\\
p(x,y,t_0) &= 0.7142857 + 0.01\exp\left(\frac{\log(2)}{0.05^2}\left[(x-0.559358)^2+(y-.539358)^2\right]\right),
\end{aligned} \tag{4.42}$$

depicted in Figure 4-56. Second, the grid spacings of the uniform grid series were increased slightly to match the spacing near the five-way singularity, with reduced the approximate points-per-wavelength range to 4 to 80.

Figure 4-58 shows contour plots of the error magnitude for the five-way singular and uniform grid calculations with the E_2 scheme. Similar to the corresponding



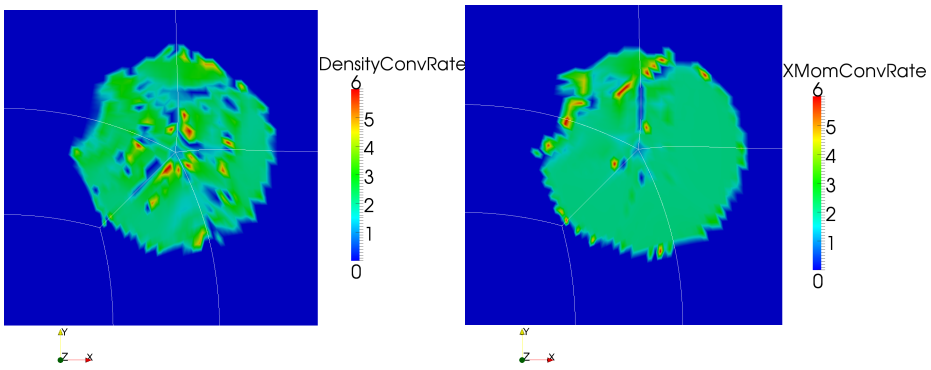
(a) ρ, singular grid (b) ρu, singular grid

(c) ρ, uniform grid (d) ρu, uniform grid

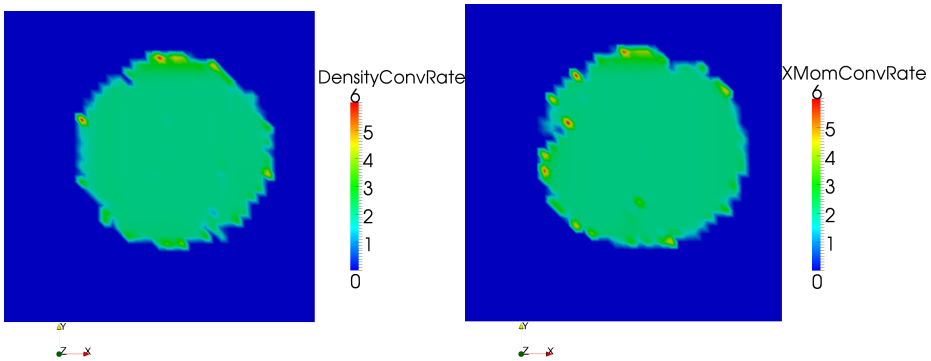
Figure 4-58: Case 2: Error magnitude for the fifth-coarsest singular and uniform grids and explicit second-order scheme

three-way case plots, the significantly-higher singular grid error follows the singular point grid lines, especially for the ρ results. The footprints of the error, and the error magnitudes at the outer portion of the footprint, are about the same for the two grid types.

Figure 4-60 shows the error convergence rate contours for the present test case with the E_2 scheme and fifth-coarsest grids. Again, the convergence rate for the uniform grid results exhibits second-order behavior for a large region. The singular grid data, on the other hand, shows streaks of first-order convergence along the singular grid lines and at the singularity itself, and localize pockets of super-convergence. Still, significant regions of second-order convergence exist in the singular grid calculation.

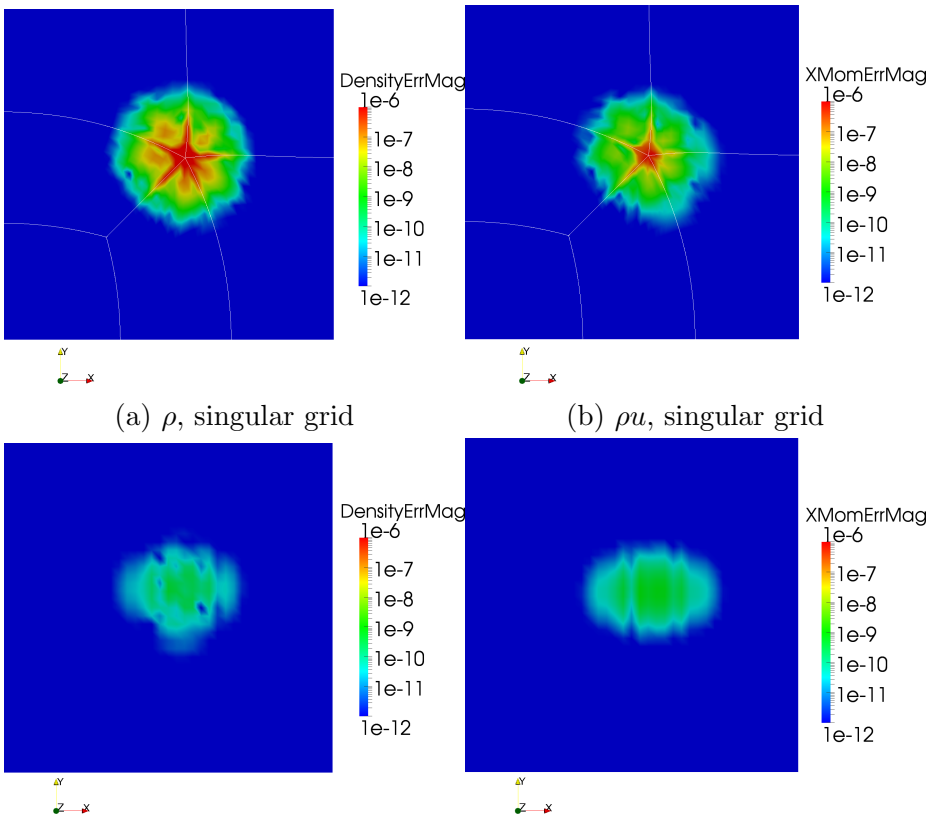


(a) ρ, singular grid (b) ρu, singular grid



(c) ρ, uniform grid (d) ρu, uniform grid

Figure 4-60: Case 2: Convergence rate of the error for the fifth-coarsest singular and uniform grids and explicit second-order scheme



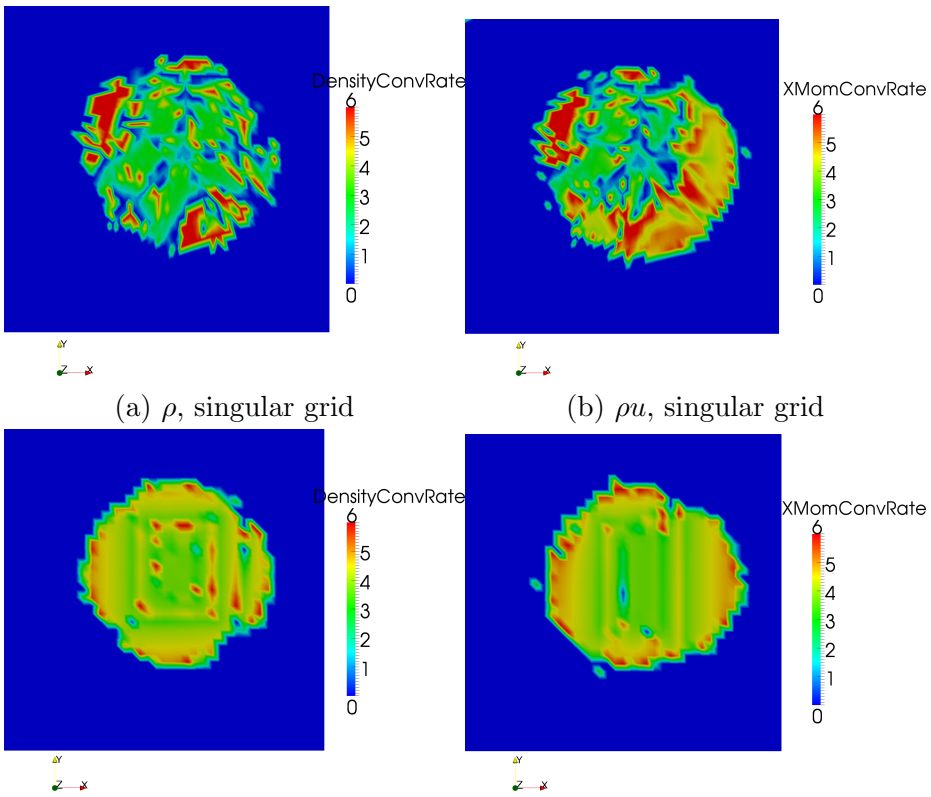
(a) ρ, singular grid (b) ρu, singular grid

(c) ρ, uniform grid (d) ρu, uniform grid

Figure 4-62: Case 2: Error magnitude for the fifth-coarsest singular and uniform grids and DRP scheme

Figure 4-64 shows the error contours for the five-way test case and DRP scheme with the fifth-coarsest grids. Qualitatively similar to the three-way results, the error for the singular grid is much higher than the uniform, especially along the singular lines.

Figure 4-64 shows the error convergence rate for the present case, fifth-coarsest grid, and DRP scheme. As in other results, lines of first-order convergence are seen along the singular grid lines. The convergence rate away from the singular grid lines is lower than forth-order, but is not as strongly affected as the data along the singular grid lines. On the whole, and unlike the E_2 results, the convergence rate rarely approaches the formal order-of-accuracy of the DRP scheme. Again, the high convergence rate along the outside of the error footprint is likely a function of the



(a) ρ, singular grid (b) ρu, singular grid

(c) ρ, uniform grid (d) ρu, uniform grid

Figure 4-64: Case 2: Convergence rate of the error for the fifth-coarsest singular and uniform grids and DRP scheme

very low error in that region.

The error and convergence rate results for the $l_{\max}$ norm are shown in Figures 4-65 and 4-66 for the E_2 and DRP schemes, respectively. Again, the schemes behave as expected on the uniform grid — both achieve their expected convergence rate as the grid is refined. Like the three-way case, the spatial differencing scheme has little effect on the error and convergence rate results for the singular grid results, which are again something between zero and one.

One interesting result of the preceding verification cases is the observation that the degradation of the E_2's convergence rate is confined to the area of the flow immediately surrounding the grid singularity, while the DRP's convergence rate is reduced throughout. One might wonder if this is due to the different sizes of the

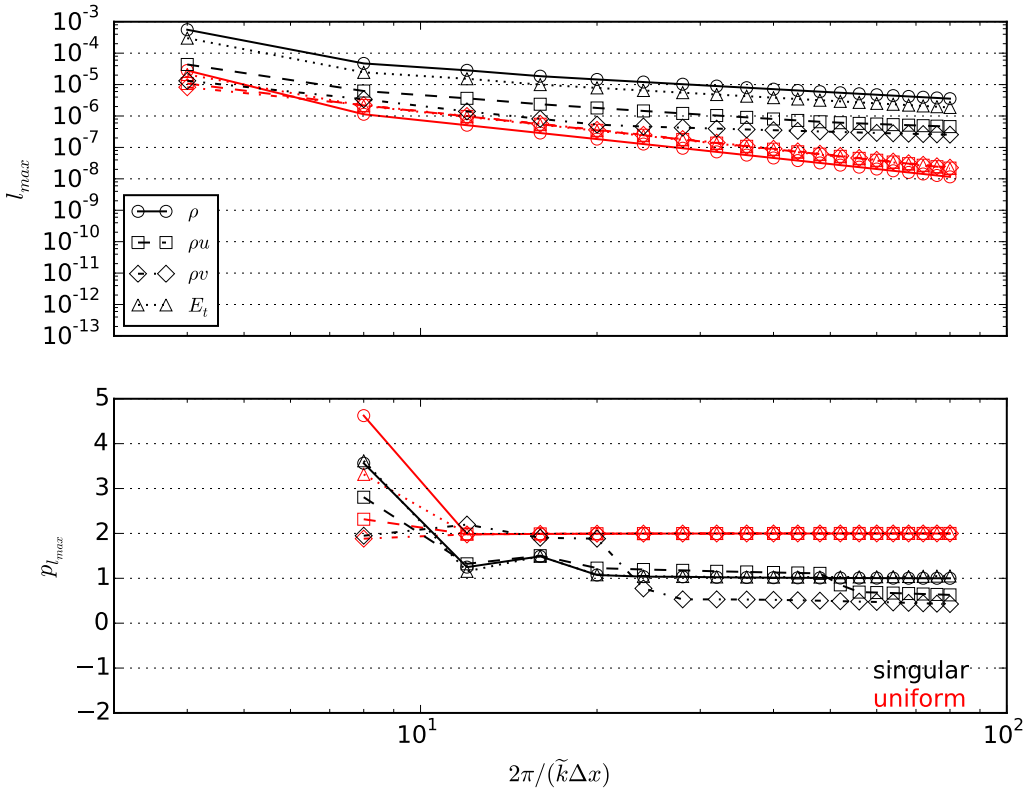


Figure 4-65: Five-way grid singularity, E_2 scheme, $l_{\max}$ error and convergence rate.

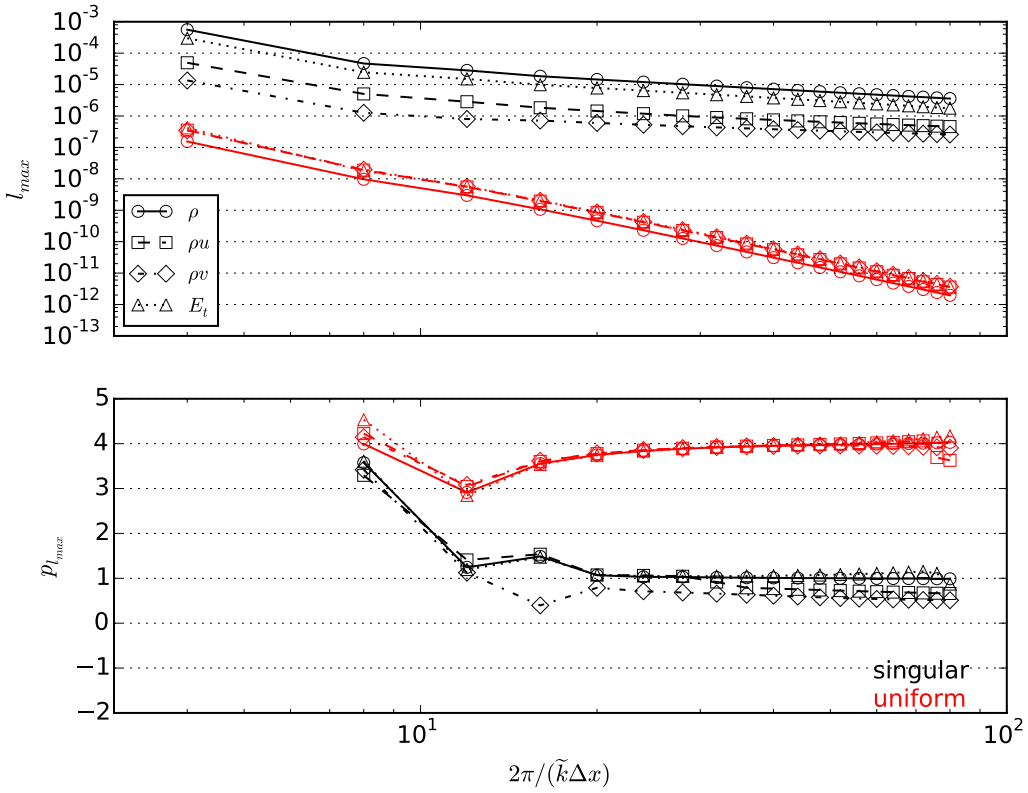
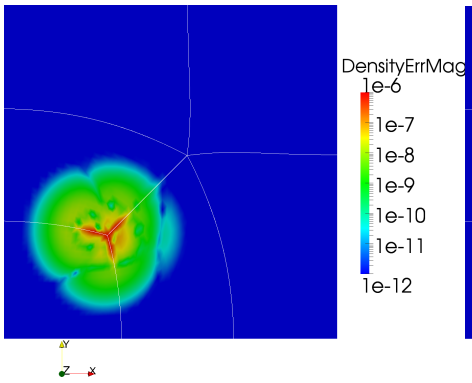


Figure 4-66: Five-way grid singularity, DRP scheme, $l_{\max}$ error and convergence rate.

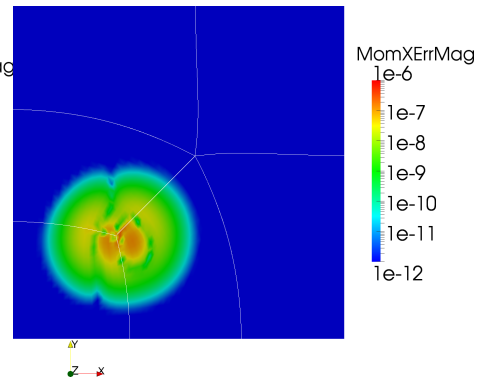
explicit second-order and DRP schemes' stencils — perhaps if more time steps were taken, expanding the region of the flow that could be affected by the grid singularity, the low-convergence-rate area of the explicit second-order scheme would also expand outward. To explore this idea, the grids and initial flow of the three-way singularity case were marched to the same final time level $t = 0.0005$ using a differing number of time steps: one, two, and four.

Figure 4-68 shows how the error footprint of the E_2 scheme is affected by increasing the number of time steps used to march the solution to the final time level of the simulation. While the approximate size of the error footprint appears to change little, the magnitude of the error near the grid singularity does seem to increase slightly. The same phenomenon is seen in the corresponding DRP results in Figure 4-74: raising the number of time steps increases the error magnitude near the singularity, but not the size of the error footprint.

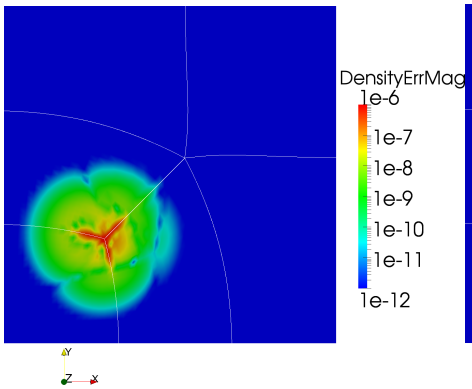
Figures 4-72 and 4-74 show how the convergence rate contours are affected by increasing the time step count. The effect of this change is much more obvious in the convergence rate contours than the error contours. On the whole, lower convergence rates along the singular lines separating areas of high convergence are observed. Again, the footprint of the convergence rate is unaffected. And, instead of decreasing, the convergence of the E_2 scheme results near the grid singularity actually increased.



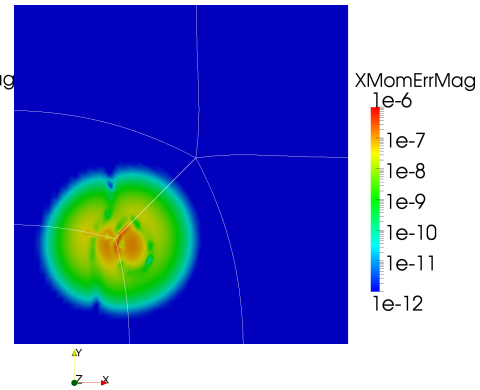
(a) ρ, one time step



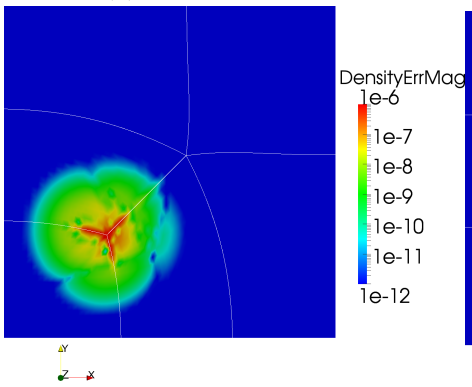
(b) ρu, one time step



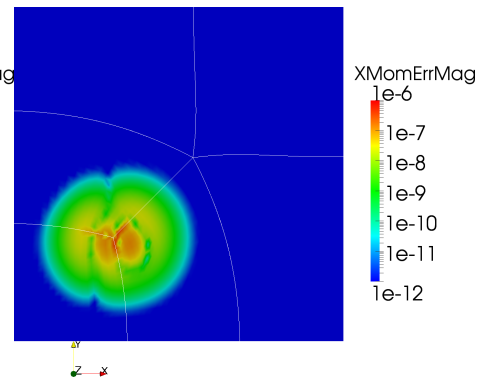
(c) ρ, two time steps



(d) ρu, two time steps

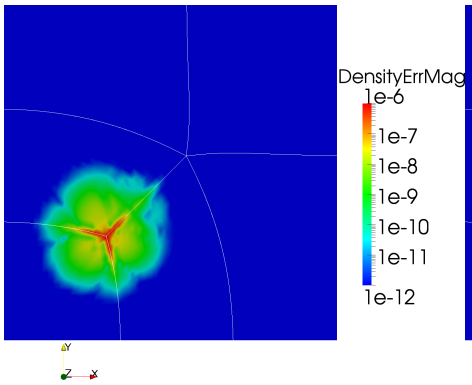


(e) ρ, four time steps

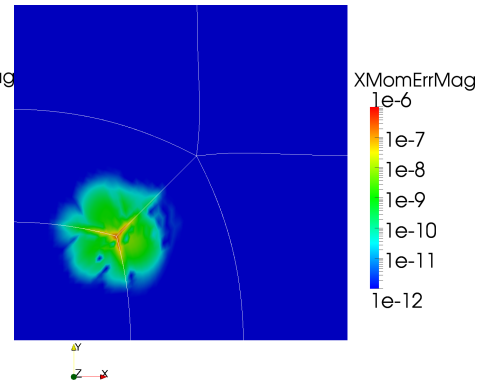


(f) ρu, four time steps

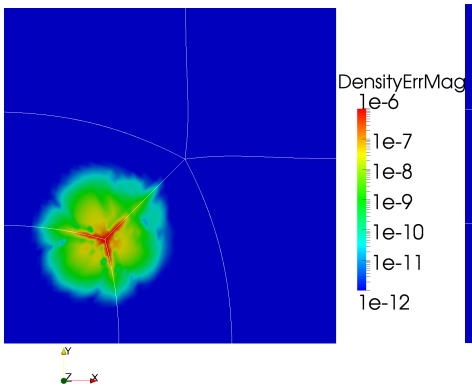
Figure 4-68: Error magnitude for the three-way case and fifth-coarsest singular grid using the explicit second-order scheme and 1, 2, and 4 time steps



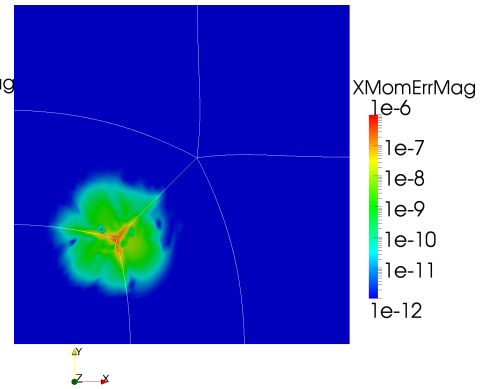
(a) ρ, one time step



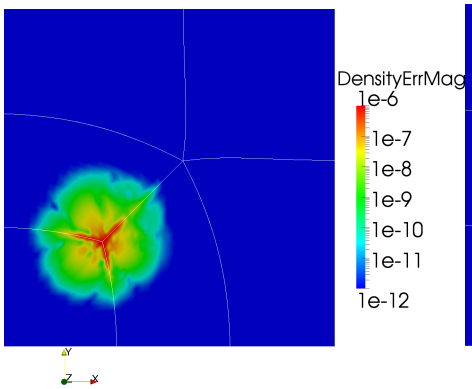
(b) ρu, one time step



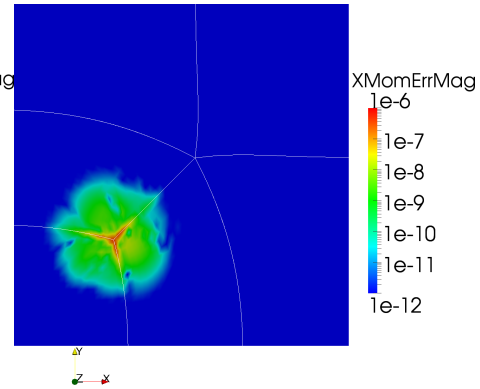
(c) ρ, two time steps



(d) ρu, two time steps

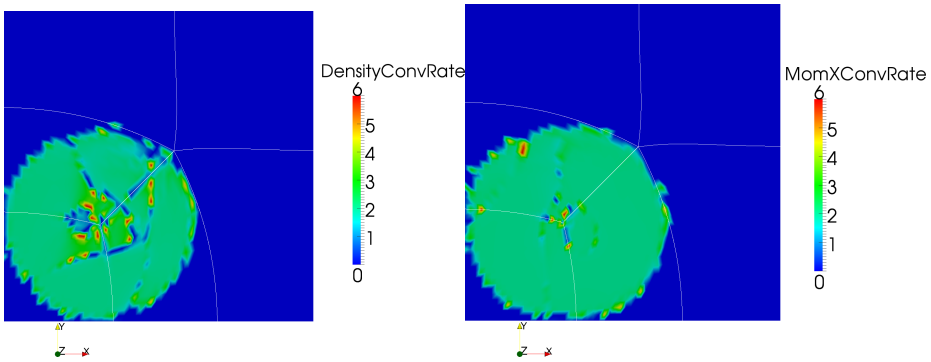


(e) ρ, four time steps

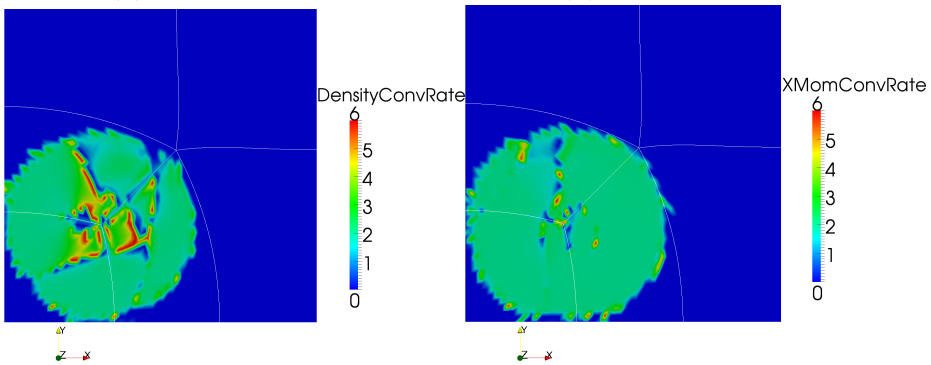


(f) ρu, four time steps

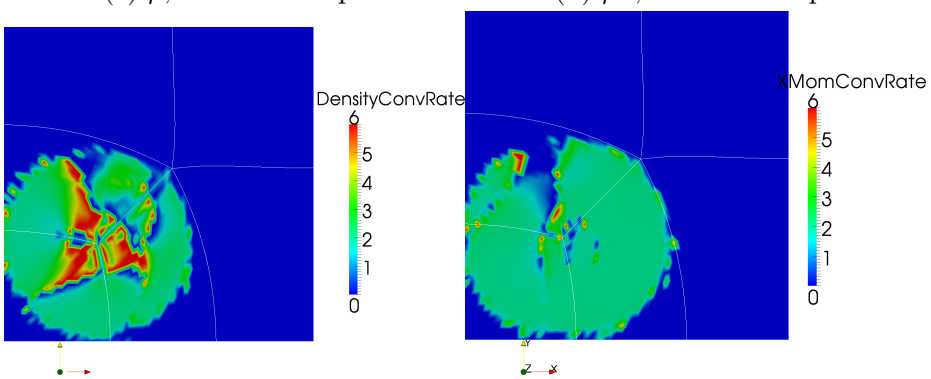
Figure 4-70: Error magnitude for the three-way case and fifth-coarsest singular grid using DRP scheme and 1, 2, and 4 time steps



(a) ρ, one time step (b) ρu, one time step

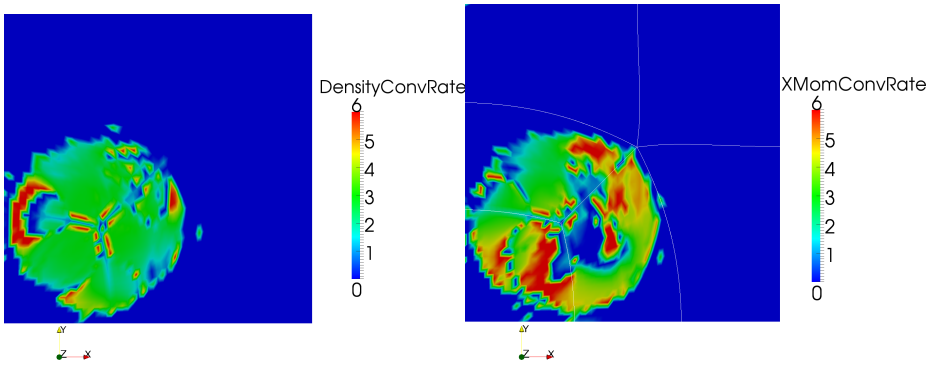


(c) ρ, two time steps (d) ρu, two time steps



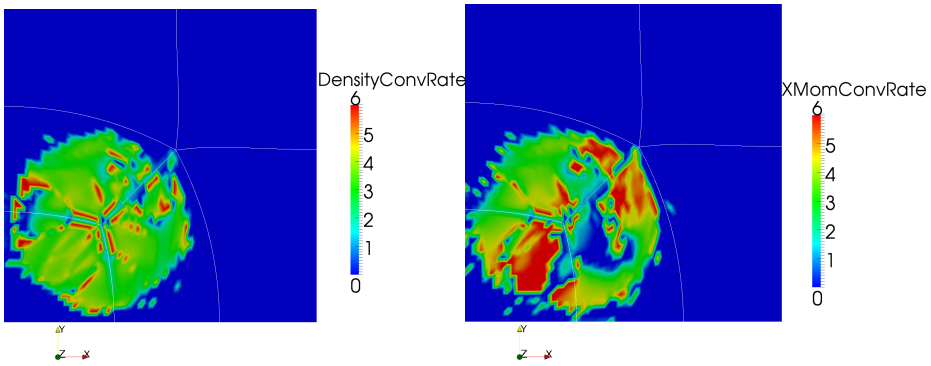
(e) ρ, four time steps (f) ρu, four time steps

Figure 4-72: Convergence rate of the error for the three-way case and fifth-coarsest singular grid using the explicit second-order scheme and 1, 2, and 4 time steps



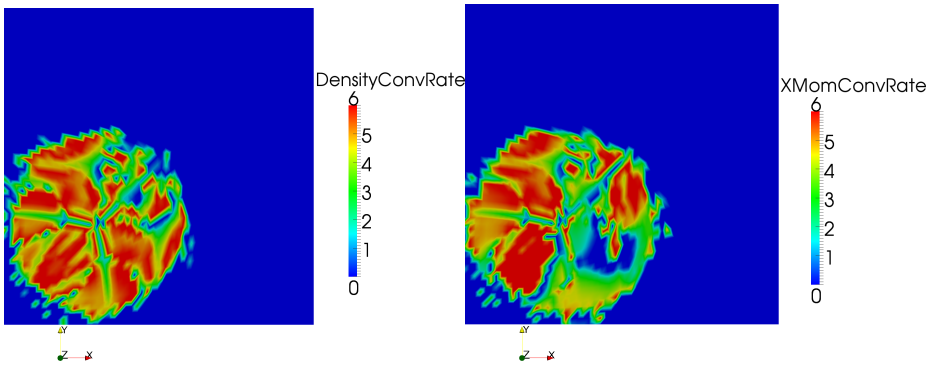
(a) ρ, one time step

(b) ρu, one time step



(c) ρ, two time steps

(d) ρu, two time steps



(e) ρ, four time steps

(f) ρu, four time steps

Figure 4-74: Convergence rate of the error for the three-way case and fifth-coarsest singular grid using the DRP scheme and 1, 2, and 4 time steps

Chapter 5

Conclusions and future work

In this work, a new approach to code verification, External Verification Analysis (EVA), was presented. Unlike the Method of Manufactured Solutions (MMS), the current state-of-the-art code verification technique, EVA does not require the addition of complicated source terms to a code's governing equations, and thus the code itself. A survey of the relevant PDE code literature appears to show that the MMS source terms limit the technique's popularity among researchers. Unlike the Method of Exact Solutions (MES), EVA maintains most of the flexibility in the form of the reference solution that MMS enjoys.

The underlying mathematics of the EVA technique were discussed in detail. Requirements for the EVA initial condition (IC) were explained, and the procedure used to advance the IC in time was applied to the inviscid Burger's equation, the three-dimensional Euler equations, and the three-dimensional Navier-Stokes equations. Formulas for the two-dimensional versions of the latter two equations are available in appendices of this work. Advice for efficiently implementing the IC chosen for this work in a computer code was included, as were steps required to use EVA to verify a PDE code, and the results from verifying EVA itself through an independent version of EVA written in the Python programming language, also found in the appendix. The grid-independent and embarrassingly-parallel nature of the EVA tool

were highlighted.

The EVA tool was then used to verify NASA Glenn's Broadband Aeroacoustic Stator Simulator (BASS), a high-order, parallel Computational Aeroacoustics code. After a Fourier analysis of BASS's numerical schemes, verification results for the code's spatial and then temporal schemes were presented. For the most part, the results closely mirrored what was predicted by the Fourier analysis. Happily, the error from each of BASS's schemes attained their design convergence rate, strongly indicating all are coded properly.

EVA was also used to investigate the accuracy of the two-dimensional inviscid implementation of a fourth-order multi-time step Adams-Bashforth (MTSAB) scheme. After exploring the accuracy of the scheme with Fourier analysis, the EVA tool was used to provide the data needed to overcome the MTSAB scheme's starting problem. Two test cases were constructed, with the first investigating the local error, and the second, the global. Both test cases showed that the error of the scheme converges at the expected rate, even at interior block boundaries where different time step sizes are used.

The final application of the EVA tool consisted of evaluating an approach for differencing across grid singularities. Because EVA uses analytic expressions for the spatial derivatives of the flow, it is unaffected by the presence of grid singularities and is thus well-suited for work involving these features. Two test cases, both inviscid and two-dimensional, were explored: a three-way and five-way singularity case. On the whole, the presence of the grid singularity was shown to increase the error of the spatial differencing scheme significantly, and decrease the convergence rate to something between zero and one.

The most significant limitation of the EVA method is the inability to specify boundary conditions along domain surfaces. In principle, it should be possible to use one of the spatial variables as the Cauchy marching direction instead of time,

which would allow one to enforce a boundary condition at the cost of relinquishing control over the initial flow. This approach would require that the surface is not characteristic, as discussed in Chapter 2. For example, a wall is a characteristic surface of the flow equations.

Another difficult aspect of code verification is determining the right set of grid spacing, time step size, and flow parameters that will allow the asymptotic range of the numerical scheme(s) under consideration to be found without extremely dense grids, many time steps, etc.. Assuming a more general form of the error than Equation (3.10) for the order-of-accuracy calculation, and possibly refining grid spacing and time step size simultaneously, may be more flexible and robust.

References

[1] Barry W. Boehm. *Software engineering economics*. Prentice-Hall advances in computing science and technologyseries. Englewood Cliffs, N.J: Prentice-Hall, 1981. ISBN: 0138221227.

[2] Patrick J. Roache. *Verification and Validation in Computational Science and Engineering*. en. Hermosa Publishers, Jan. 1998. ISBN: 9780913478080.

[3] Frederick G. Blottner. "Accurate Navier-Stokes results for the hypersonic flow over a spherical nosetip". In: *Journal of Spacecraft and Rockets* 27.2 (1990), pp. 113–122. URL: `http://arc.aiaa.org/doi/pdf/10.2514/3.26115` (visited on 04/11/2014).

[4] Patrick Knupp and Kambiz Salari. *Verification of Computer Codes in Computational Science and Engineering*. en. CRC Press, Oct. 2002. ISBN: 9781584882640.

[5] William L. Oberkampf and Christopher J. Roy. *Verification and Validation in Scientific Computing*. en. Cambridge University Press, Oct. 2010. ISBN: 9781139491761.

[6] Patrick J. Roache. "Building PDE Codes to be Verifiable and Validatable". In: *Computing in Science and Engineering* 6.5 (2004), pp. 30–38. ISSN: 15219615. URL: `http://journals.ohiolink.edu/ejc/article.cgi?issn=15219615&issue=v06i0005&article=30_bpctbvav`.

[7] William L Oberkampf, Timothy G Trucano, and Charles Hirsch. "Verification, validation, and predictive capability in computational engineering and physics". In: *Applied Mechanics Reviews* 57.5 (2004), pp. 345–384.

[8] Patrick Knupp, Curtis C. Ober, and Ryan B. Bond. "Measuring progress in order-verification within software development projects". en. In: *Engineering with Computers* 23.4 (Dec. 2007), pp. 271–282. ISSN: 0177-0667, 1435-5663. DOI: 10.1007/s00366-007-0066-x. URL: http://link.springer.com.proxy.ohiolink.edu:9099/article/10.1007/s00366-007-0066-x (visited on 09/09/2014).

[9] Frank White. *Viscous Fluid Flow*. English. 3 edition. New York, NY: McGraw-Hill Science/Engineering/Math, Jan. 2005. ISBN: 9780072402315.

[10] Hermann Schlichting. *Boundary-Layer Theory*. Seventh. McGraw-Hill, Apr. 1979. ISBN: 0-07-055334-3.

[11] G. K. Batchelor. *An Introduction to Fluid Dynamics*. English. Cambridge, U.K.; New York, NY: Cambridge University Press, 1967. ISBN: 0521098173.

[12] Christopher J. Roy and William L. Oberkampf. "A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing". In: *Computer Methods in Applied Mechanics and Engineering* 200.25–28 (2011), pp. 2131–2144. ISSN: 0045-7825. DOI: http://dx.doi.org/10.1016/j.cma.2011.03.016. URL: http://www.sciencedirect.com/science/article/pii/S0045782511001290.

[13] Kambiz Salari and Patrick Knupp. *Code Verification by the Method of Manufactured Solutions*. Tech. rep. SAND2000-1444. Sandia National Laboratories, June 2000. URL: http://www.osti.gov/scitech/servlets/purl/759450.

[14] Stanly Steinberg and Patrick J Roache. "Symbolic manipulation and computational fluid dynamics". In: *Journal of Computational Physics* 57.2 (Jan. 1985). 00079, pp. 251–284. ISSN: 0021-9991. DOI: 10.1016/0021-9991(85)90045-2. URL: http://www.sciencedirect.com/science/article/pii/0021999185900452 (visited on 04/11/2014).

[15] Patrick J Roache. “Code verification by the method of manufactured solutions”. In: *Journal of Fluids Engineering* 124.1 (2002), pp. 4–10.

[16] Subrahmanya P. Veluri, Christopher J. Roy, Shelley Hebert, and Edward A. Luke. “Verification of the Loci-CHEM CFD Code Using the Method of Manufactured Solutions”. In: *46th AIAA Aerospace Sciences Meeting and Exhibit*. 2008. URL: http://arc.aiaa.org/doi/pdf/10.2514/6.2008-661 (visited on 04/11/2014).

[17] Subrahmanya P Veluri. “Code verification and numerical accuracy assessment for finite volume CFD codes”. PhD thesis. Virginia Polytechnic Institute and State University, 2010.

[18] Subrahmanya Veluri, Christopher J Roy, and Edward Luke. “Comprehensive code verification for an unstructured finite volume CFD code”. In: *48th AIAA Aerospace Sciences Meeting*. Vol. 127. 2010.

[19] Subrahmanya P. Veluri, Christopher J. Roy, and Edward A. Luke. “Comprehensive code verification techniques for finite volume CFD codes”. In: *Computers & Fluids* 70 (2012), pp. 59–72. ISSN: 0045-7930. DOI: http://dx.doi.org/10.1016/j.compfluid.2012.04.028. URL: http://www.sciencedirect.com/science/article/pii/S0045793012001697.

[20] Ryan Bond, Patrick Knupp, and Curtis Ober. “A Manufactured Solution for Verifying CFD Boundary Conditions”. In: *34th AIAA Fluid Dynamics Conference and Exhibit*. Fluid Dynamics and Co-located Conferences. American Institute of Aeronautics and Astronautics, June 2004. URL: http://dx.doi.org/10.2514/6.2004-2629 (visited on 04/09/2014).

[21] Ryan Bond, Patrick Knupp, and Curtis Ober. “A Manufactured Solution for Verifying CFD Boundary Conditions, Part II”. In: *43rd AIAA Aerospace Sciences Meeting and Exhibit*. Aerospace Sciences Meetings. American Institute of

Aeronautics and Astronautics, Jan. 2005. URL: http://dx.doi.org/10.2514/6.2005-88 (visited on 04/09/2014).

[22] Ryan Bond, Curtis Ober, and Patrick Knupp. "A Manufactured Solution for Verifying CFD Boundary Conditions, Part III". In: *36th AIAA Fluid Dynamics Conference and Exhibit*. Fluid Dynamics and Co-located Conferences. American Institute of Aeronautics and Astronautics, June 2006. URL: http://dx.doi.org/10.2514/6.2006-3722 (visited on 04/09/2014).

[23] L. Eça and M. Hoekstra. "An introduction to CFD code verification including eddy-viscosity Models". In: *ECCOMAS CFD*. Vol. 2006. 2006.

[24] L. Eça, M. Hoekstra, A. Hay, and D. Pelletier. "On the construction of manufactured solutions for one and two-equation eddy-viscosity models". In: *International Journal for Numerical Methods in Fluids* 54.2 (2007), pp. 119–154. ISSN: 1097-0363. DOI: 10.1002/fld.1387. URL: http://dx.doi.org/10.1002/fld.1387.

[25] Dominique Tremblay, Stephane Etienne, and Dominique Pelletier. "Code Verification and the Method of Manufactured Solutions for Fluid-Structure Interaction Problems". In: *36th AIAA Fluid Dynamics Conference and Exhibit*. American Institute of Aeronautics and Astronautics. URL: http://arc.aiaa.org/doi/abs/10.2514/6.2006-3218 (visited on 04/11/2014).

[26] S. Étienne, A. Garon, and D. Pelletier. "Some manufactured solutions for verification of fluid-structure interaction codes". In: *Computers & Structures* 106–107 (2012), pp. 56–67. ISSN: 0045-7949. DOI: http://dx.doi.org/10.1016/j.compstruc.2012.04.006. URL: http://www.sciencedirect.com/science/article/pii/S0045794912001010.

[27] Zhao-xin Gong, Chuan-jing Lu, and Hua-xiong Huang. "Accuracy analysis of immersed boundary method using method of manufactured solutions". en. In:

Applied Mathematics and Mechanics 31.10 (Oct. 2010), pp. 1197–1208. ISSN: 0253-4827, 1573-2754. DOI: 10.1007/s10483-010-1353-x. URL: http://link.springer.com/article/10.1007/s10483-010-1353-x (visited on 04/09/2014).

[28] Tommi Mikkola. "Simulation of unsteady free surface flows - code verification and discretisation error". Dissertation. Helsinki University of Technology, 2009. URL: https://aaltodoc.aalto.fi:443/handle/123456789/4667 (visited on 04/09/2014).

[29] João Marcelo Vedovoto, Aristeu da Silveira Neto, Arnaud Mura, and Luis Fernando Figueira da Silva. "Application of the method of manufactured solutions to the verification of a pressure-based finite-volume numerical scheme". In: *Computers & Fluids* 51.1 (2011), pp. 85–99. ISSN: 0045-7930. DOI: http://dx.doi.org/10.1016/j.compfluid.2011.07.014. URL: http://www.sciencedirect.com/science/article/pii/S0045793011002404.

[30] P. T. Brady, M. Herrmann, and J. M. Lopez. "Code verification for finite volume multiphase scalar equations using the method of manufactured solutions". In: *Journal of Computational Physics* 231.7 (2012), pp. 2924–2944. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/j.jcp.2011.12.040. URL: http://www.sciencedirect.com/science/article/pii/S0021999112000083.

[31] O. Merroun, A. Almers, T. El Bardouni, B. El Bakkari, and E. Chakir. "Analytical benchmarks for verification of thermal-hydraulic codes based on subchannel approach". In: *Nuclear Engineering and Design* 239.4 (2009), pp. 735–748. ISSN: 0029-5493. DOI: http://dx.doi.org/10.1016/j.nucengdes.2009.01.005. URL: http://www.sciencedirect.com/science/article/pii/S0029549309000375.

[32] Shawn D Pautz. "Verification of transport codes by the method of manufactured solutions: the ATTILA experience". In: *ANS International Meeting on Mathematical Methods for Nuclear Applications, Salt Lake City, Utah, American Nuclear Society*. 2001.

[33] Luís Eça, Martin Hoekstra, Patrick Roache, and Hugh Coleman. "Code verification, solution verification and validation: an overview of the 3rd Lisbon workshop". In: *AIAA Paper* 3647 (2009), pp. 22–25.

[34] Les Hatton. "The T experiments: errors in scientific software". In: *Computing in Science and Engineering* 4.2 (1997), pp. 27–38. URL: http://www.computer.org/csdl/mags/cs/1997/02/c2027.pdf (visited on 08/13/2014).

[35] William L. Oberkampf and Frederick G. Blottner. "Issues in computational fluid dynamics code verification and validation". In: *AIAA journal* 36.5 (1998), pp. 687–695.

[36] T. G. Trucano, L. P. Swiler, T. Igusa, W. L. Oberkampf, and M. Pilch. "Calibration, validation, and sensitivity analysis: What's what". In: *Reliability Engineering & System Safety* 91.10–11 (2006). The Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004) SAMO 2004 The Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004), pp. 1331–1357. ISSN: 0951-8320. DOI: http://dx.doi.org/10.1016/j.ress.2005.11.031. URL: http://www.sciencedirect.com/science/article/pii/S0951832005002437.

[37] Patrick J. Roache, Kirti N. Ghia, and Frank M. White. "Editorial Policy Statement on the Control of Numerical Accuracy". In: *Journal of Fluids Engineering* 108.1 (Mar. 1986), pp. 2–2. ISSN: 0098-2202. DOI: 10.1115/1.3242537. URL: http://dx.doi.org/10.1115/1.3242537 (visited on 04/11/2014).

[38] Ishmail Celik, Urmila Ghia, Patrick Roache, and Christopher. "Procedure for estimation and reporting of uncertainty due to discretization in CFD applications". In: *Journal of fluids Engineering-Transactions of the ASME* 130.7 (July 2008). DOI: 10.1115/1.2960953. URL: http://dx.doi.org/10.1115/1.2960953 (visited on 04/11/2014).

[39] P. M. Gresho and C. Taylor. "Editorial". en. In: *International Journal for Numerical Methods in Fluids* 19.11 (Dec. 1994), pp. i–i. ISSN: 1097-0363. DOI: 10.1002/fld.1650191109. URL: http://0-onlinelibrary.wiley.com.carlson.utoledo.edu/doi/10.1002/fld.1650191109/abstract (visited on 08/18/2014).

[40] "Editorial Policy Statement on Numerical Accuracy and Experimental Uncertainty". In: *AIAA Journal* 32.1 (1994), p. 3. ISSN: 0731-5090. DOI: 10.2514/1.15006. URL: http://arc.aiaa.org/doi/abs/10.2514/1.15006 (visited on 04/11/2014).

[41] "Editorial Policy Statement on Numerical and Experimental Accuracy". In: *AIAA Journal* 26.1 (2012), pp. 7–7. ISSN: 0887-8722. DOI: 10.2514/1.57317. URL: http://arc.aiaa.org/doi/abs/10.2514/1.57317 (visited on 04/11/2014).

[42] The Editorial Board. "Journal of Heat Transfer Editorial Policy Statement on Numerical Accuracy". In: *Journal of Heat Transfer* 116.4 (Nov. 1994), pp. 797–798. ISSN: 0022-1481. DOI: 10.1115/1.2911449. URL: http://dx.doi.org/10.1115/1.2911449 (visited on 08/18/2014).

[43] P. J. Roache. "Quantification of Uncertainty in Computational Fluid Dynamics". In: *Annual Review of Fluid Mechanics* 29.1 (1997), pp. 123–160. DOI: 10.1146/annurev.fluid.29.1.123. URL: http://www.annualreviews.org/doi/abs/10.1146/annurev.fluid.29.1.123 (visited on 04/11/2014).

[44] Patrick J. Roache. "Need for control of numerical accuracy". In: *Journal of Spacecraft and Rockets* 27.2 (1990), pp. 98–102. ISSN: 0022-4650. DOI: 10.2514/3.26113. URL: http://arc.aiaa.org/doi/abs/10.2514/3.26113 (visited on 04/11/2014).

[45] *Journal of Computational Physics* 274 (2014). ISSN: 0021-9991.

[46] *Computers & Fluids* 103 (2014). ISSN: 0045-7930.

[47] *Computational Mathematics - Google Scholar Metrics*. URL: http://scholar.google.com/citations?view_op=top_venues&hl=en&vq=phy_computationalmathematics (visited on 03/03/2015).

[48] *Fluid Mechanics - Google Scholar Metrics*. URL: http://scholar.google.com/citations?view_op=top_venues&hl=en&vq=phy_fluidmechanics (visited on 03/03/2015).

[49] Gerald B. Folland. *Introduction to Partial Differential Equations*. Princeton University Press, 1976.

[50] Sigeru Mizohata. *The Theory of Partial Differential Equations*. Cambridge University Press, 1973.

[51] Duane Hixon and Bo Li. "External Verification Analysis (EVA) Method for Time Marching Computational Aeroacoustics Codes". In: *14th AIAA/CEAS Aeroacoustics Conference (29th AIAA Aeroacoustics Conference)*. Vancouver, BC: American Institute of Aeronautics and Astronautics, May 2008, pp. 1–10. URL: http://arc.aiaa.org/doi/abs/10.2514/6.2008-2920 (visited on 04/11/2014).

[52] Daniel Ingraham and Ray Hixon. "External verification analysis: A code-independent verification technique for unsteady PDE codes". In: *Journal of Computational Physics* 243 (June 2013), pp. 46–57. ISSN: 0021-9991. DOI: 10.1016/j.jcp.

2013.02.032. URL: http://www.sciencedirect.com/science/article/pii/S0021999113001514 (visited on 04/09/2014).

[53] Yu. V. Egorov and M. A. Shubin. *Partial Differential Equations I: Foundations of the Classical Theory*. Springer-Verlag, 1991.

[54] Peter V. O'Neil. *Advanced Engineering Mathematics*. 6th ed. Thomson, 2007.

[55] Paul R. Garabedian. *Partial Differential Equations*. American Mathematical Society, 1964.

[56] *The Python Programming Language*. https://www.python.org/. URL: https://www.python.org/ (visited on 11/08/2014).

[57] SymPy Development Team. *SymPy: Python library for symbolic mathematics*. http://www.sympy.org. 2014. URL: http://www.sympy.org.

[58] John Anderson. *Computational Fluid Dynamics: The Basics with Applications*. English. 1 edition. New York: McGraw-Hill Science/Engineering/Math, Feb. 1995. ISBN: 9780070016859.

[59] Richard H. Pletcher, John C. Tannehill, and Dale Anderson. *Computational Fluid Mechanics and Heat Transfer, Second Edition*. English. 2 edition. Washington, DC: Taylor & Francis, Apr. 1997. ISBN: 9781560320463.

[60] Ray Hixon, M. Nallasamy, and Scott Sawyer. "Parallelization Strategy for an Explicit Computational Aeroacoustics Code". In: *8th AIAA/CEAS Aeroacoustics Conference & Exhibit*. Breckenridge, Colorado: American Institute of Aeronautics and Astronautics, June 2002. URL: http://arc.aiaa.org/doi/abs/10.2514/6.2002-2583 (visited on 11/11/2014).

[61] Sanjiva K. Lele. "Compact finite difference schemes with spectral-like resolution". In: *Journal of Computational Physics* 103.1 (Nov. 1992), pp. 16–42. ISSN: 0021-9991. DOI: 10.1016/0021-9991(92)90324-R. URL: http://www.

sciencedirect.com/science/article/pii/002199919290324R (visited on 08/22/2014).

[62] J. C. Butcher. *Numerical methods for ordinary differential equations*. 2nd ed. Chichester, England ; Hoboken, NJ: Wiley, 2008. ISBN: 9780470723357.

[63] J. H. Williamson. "Low-storage runge-kutta schemes". In: *Journal of Computational Physics* 35.1 (1980), pp. 48–56. URL: http://www.sciencedirect.com/science/article/pii/0021999180900339 (visited on 03/11/2015).

[64] M. Calvo, J. M. Franco, and L. Rández. "A new minimum storage Runge–Kutta scheme for computational acoustics". In: *Journal of Computational Physics* 201.1 (Nov. 2004), pp. 1–12. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2004.05.012. URL: http://www.sciencedirect.com/science/article/pii/S0021999104001913 (visited on 03/11/2015).

[65] Christopher A. Kennedy, Mark H. Carpenter, and R. Michael Lewis. "Low-storage, explicit Runge–Kutta schemes for the compressible Navier–Stokes equations". In: *Applied Numerical Mathematics* 35.3 (Nov. 2000), pp. 177–219. ISSN: 0168-9274. DOI: 10.1016/S0168-9274(99)00141-5. URL: http://www.sciencedirect.com/science/article/pii/S0168927499001415 (visited on 03/11/2015).

[66] M. Calvo, J. M. Franco, and L. Rández. "Minimum storage Runge-Kutta schemes for computational acoustics". In: *Computers & Mathematics with Applications* 45.1–3 (Jan. 2003), pp. 535–545. ISSN: 0898-1221. DOI: 10.1016/S0898-1221(03)80035-4. URL: http://www.sciencedirect.com/science/article/pii/S0898122103800354 (visited on 03/11/2015).

[67] Julien Berland, Christophe Bogey, and Christophe Bailly. "Low-dissipation and low-dispersion fourth-order Runge–Kutta algorithm". In: *Computers & Fluids* 35.10 (Dec. 2006), pp. 1459–1463. ISSN: 0045-7930. DOI: 10.1016/j.

compfluid.2005.04.003. URL: http://www.sciencedirect.com/science/article/pii/S0045793005000575 (visited on 03/11/2015).

[68] Christophe Bogey and Christophe Bailly. "A family of low dispersive and low dissipative explicit schemes for flow and noise computations". In: *Journal of Computational Physics* 194.1 (Feb. 2004), pp. 194–214. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2003.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0021999103004662 (visited on 03/11/2015).

[69] F. Q. Hu, M. Y. Hussaini, and J. L. Manthey. "Low-Dissipation and Low-Dispersion Runge–Kutta Schemes for Computational Acoustics". In: *Journal of Computational Physics* 124.1 (Mar. 1996), pp. 177–191. ISSN: 0021-9991. DOI: 10.1006/jcph.1996.0052. URL: http://www.sciencedirect.com/science/article/pii/S0021999196900522 (visited on 11/11/2014).

[70] Vasanth Allampalli, Ray Hixon, M. Nallasamy, and Scott D. Sawyer. "High-accuracy large-step explicit Runge–Kutta (HALE-RK) schemes for computational aeroacoustics". In: *Journal of Computational Physics* 228.10 (June 2009), pp. 3837–3850. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2009.02.015. URL: http://www.sciencedirect.com/science/article/pii/S0021999109000849 (visited on 11/11/2014).

[71] D. Stanescu and W. G. Habashi. "2N-Storage Low Dissipation and Dispersion Runge-Kutta Schemes for Computational Acoustics". In: *Journal of Computational Physics* 143.2 (1998), pp. 674–681.

[72] Christopher K. W. Tam and Jay C. Webb. "Dispersion-Relation-Preserving Finite Difference Schemes for Computational Acoustics". In: *Journal of Computational Physics* 107.2 (Aug. 1993), pp. 262–281. ISSN: 0021-9991. DOI: 10.1006/jcph.1993.1142. URL: http://www.sciencedirect.com/science/article/pii/S0021999183711423 (visited on 11/11/2014).

[73] Christopher Tam and Hao Shen. "Direct computation of nonlinear acoustic pulses using high-order finite difference schemes". In: *15th Aeroacoustics Conference*. American Institute of Aeronautics and Astronautics, 1993. URL: http://arc.aiaa.org/doi/abs/10.2514/6.1993-4325 (visited on 02/18/2015).

[74] R. Hixon. "Prefactored Small-Stencil Compact Schemes". In: *Journal of Computational Physics* 165.2 (Dec. 2000). 00079, pp. 522–541. ISSN: 0021-9991. DOI: 10.1006/jcph.2000.6631. URL: http://www.sciencedirect.com/science/article/pii/S0021999100966312 (visited on 05/29/2014).

[75] A. Jameson. "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings". In: *AIAA 10th Computational Fluid Dynamics Conference*. AIAA 91-1596. Honolulu, HI, June 1991.

[76] Michael B. Giles. "Nonreflecting boundary conditions for Euler equation calculations". In: *AIAA Journal* 28.12 (1990). 00601, pp. 2050–2058. ISSN: 0001-1452. DOI: 10.2514/3.10521. URL: http://arc.aiaa.org/doi/abs/10.2514/3.10521 (visited on 05/22/2014).

[77] C. K. W. Tam and K. A. Kurbatskii. "Multi-size-mesh multi-time-step dispersion-relation-preserving scheme for multiple-scales aeroacoustics problems". In: *International Journal of Computational Fluid Dynamics* 17.2 (2003), pp. 119–132. URL: http://www.tandfonline.com/doi/abs/10.1080/1061856031000104860 (visited on 04/01/2015).

[78] Vasanth Allampalli and Ray Hixon. "Implementation of multi-time step Adams-Bashforth time marching scheme for CAA". In: *46th AIAA Aerospace Sciences Meeting and Exhibit*. 2008, pp. 2008–29. URL: http://arc.aiaa.org/doi/pdf/10.2514/6.2008-29 (visited on 04/24/2015).

[79] Vasanth Allampalli. "Fourth order Multi-Time-Stepping Adams-Bashforth (MTSAB) scheme for NASA Glenn Research Center's Broadband Aeroacoustic Stator Sim-

ulation (BASS) Code". 00000. PhD thesis. University of Toledo, 2010. URL: https://etd.ohiolink.edu/ap/10?201689476653380::NO:10:P10_ETD_SUBID:77990 (visited on 05/29/2014).

[80] Daniel Ingraham, Vasanth Allampalli, and Ray Hixon. "Verification of a Multi-Time-Step Adams-Bashforth (MTSAB) Time-Marching Scheme Using External Verification Analysis (EVA)". In: *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics. URL: http://arc.aiaa.org/doi/abs/10.2514/6.2011-1088 (visited on 04/29/2015).

[81] *GridPro/az3000*. Program Development Company. URL: http://www.gridpro.com/.

[82] Hiroshi Akima. "Algorithm 433: interpolation and smooth curve fitting based on local procedures [E2]". In: *Communications of the ACM* 15.10 (Oct. 1972), pp. 914–918. ISSN: 00010782. DOI: 10.1145/355604.355605. URL: http://portal.acm.org/citation.cfm?doid=355604.355605 (visited on 05/01/2015).

[83] Daniel Ingraham and Ray Hixon. "Evaluating the Accuracy of a Grid Singularity Strategy Using External Verification Analysis". In: *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics. URL: http://arc.aiaa.org/doi/abs/10.2514/6.2013-216 (visited on 05/01/2015).

Appendix A

Properties of the multiindex

A multiindex is an ordered collection of non-negative integers. The magnitude of a multiindex is

$$|\alpha| = \sum_{i=1}^{n} \alpha_i \tag{A.1}$$

where n is the "length" of the multiindex. The sum and difference of two equal-length multiindices is found according to the rule

$$\alpha \pm \beta = (\alpha_1 \pm \beta_1, \alpha_2 \pm \beta_2, \ldots, \alpha_n \pm \beta_n)\,. \tag{A.2}$$

The multiindex can be used to form products of the components of an n-length vector x with the notation

$$x^\alpha = \prod_{i=1}^{n} x_i^{\alpha_i} = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}. \tag{A.3}$$

Equation (A.3) implies a useful notation for partial derivatives:

$$\frac{\partial^\alpha f}{\partial x^\alpha} = \frac{\partial^{\alpha_1+\alpha_2+\cdots+\alpha_n}}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \cdots \partial x_n^{\alpha_n}} f \tag{A.4}$$

which shows the meaning of the condition on $|\alpha|$ in Equation (2.1): k is the order of the PDE.

Appendix B

Recurrence relations for the primitive variable specific volume Euler equations in two dimensions

This appendix presents recurrence relations for the primitive variable specific volume form of the two-dimensional Euler equations, i.e.,

$$\frac{\partial \sigma}{\partial t} + u\frac{\partial \sigma}{\partial x} + v\frac{\partial \sigma}{\partial y} - \sigma\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) = 0 \tag{B.1a}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \sigma\frac{\partial p}{\partial x} = 0 \tag{B.1b}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \sigma\frac{\partial p}{\partial y} = 0 \tag{B.1c}$$

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + \gamma p\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) = 0. \tag{B.1d}$$

Taking b, d, n derivatives with respect to x, y, and t, respectively, of Equation (B.1a) produces

$$\begin{aligned}\frac{\partial^{b+d+n+1}\sigma}{\partial x^b\,\partial y^d\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m}\sigma}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m}\sigma}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\ - \frac{\partial^{b-a+d-c+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\Bigg(\\ \frac{\partial^{a+1+c+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{a+c+1+m}v}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}}\Bigg) \\ \Bigg].\end{aligned} \tag{B.2}$$

For Equation (B.1b),

$$\begin{aligned}\frac{\partial^{b+d+n+1}u}{\partial x^b\,\partial y^d\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m}u}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m}p}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}}\Bigg],\end{aligned} \tag{B.3}$$

Equation (B.1c),

$$\begin{aligned}\frac{\partial^{b+d+n+1} v}{\partial x^b\,\partial y^d\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+n-m} u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m} v}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m} v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m} v}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m} \sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m} p}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}}\Bigg],\end{aligned} \tag{B.4}$$

and finally, Equation (B.1d),

$$\begin{aligned}\frac{\partial^{b+d+n+1} p}{\partial x^b\,\partial y^d\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+n-m} u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m} p}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m} v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m} p}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\ + \gamma\frac{\partial^{b-a+d-c+n-m} p}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\Bigg(\\ \frac{\partial^{a+1+c+m} u}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} + \frac{\partial^{a+c+1+m} v}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}}\Bigg) \\ \Bigg].\end{aligned} \tag{B.5}$$

Appendix C

Recurrence relations for the primitive variable specific volume Euler equations in three dimensions

This appendix presents recurrence relations for the primitive variable specific volume form of the three-dimensional Euler equations, i.e.,

$$\frac{\partial \sigma}{\partial t} + u\frac{\partial \sigma}{\partial x} + v\frac{\partial \sigma}{\partial y} + w\frac{\partial \sigma}{\partial z} - \sigma\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = 0 \qquad \text{(2.40a repeated)}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \sigma\frac{\partial p}{\partial x} = 0 \qquad \text{(2.40b repeated)}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \sigma\frac{\partial p}{\partial y} = 0 \qquad \text{(2.40c repeated)}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \sigma\frac{\partial p}{\partial z} = 0 \qquad \text{(2.40d repeated)}$$

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + w\frac{\partial p}{\partial z} + \gamma p\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = 0. \qquad \text{(2.40e repeated)}$$

Taking b, d, f, n derivatives with respect to x, y, z, and t, respectively, of Equation (2.40a) produces

$$\begin{aligned}\frac{\partial^{b+d+f+n+1}\sigma}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+f-e+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}\sigma}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+f-e+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}\sigma}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+f-e+n-m}w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}\sigma}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}} \\ - \frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\Bigg(\\ \frac{\partial^{a+1+c+e+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{a+c+1+e+m}v}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{a+c+e+1+m}w}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\Bigg)\Bigg].\end{aligned} \tag{2.41c repeated}$$

For Equation (2.40b),

$$\begin{aligned}\frac{\partial^{b+d+f+n+1}u}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+f-e+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+f-e+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}u}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+f-e+n-m}w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}u}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}p}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\Bigg],\end{aligned} \tag{C.2}$$

Equation (2.40c),

$$\begin{aligned}\frac{\partial^{b+d+f+n+1}v}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+f-e+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}v}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\\ +\frac{\partial^{b-a+d-c+f-e+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}v}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\\ +\frac{\partial^{b-a+d-c+f-e+n-m}w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}v}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\ +\frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}p}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\Bigg],\end{aligned} \tag{C.3}$$

Equation (2.40d),

$$\begin{aligned}\frac{\partial^{b+d+f+n+1}w}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+f-e+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}w}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\\ +\frac{\partial^{b-a+d-c+f-e+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}w}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\\ +\frac{\partial^{b-a+d-c+f-e+n-m}w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}w}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\ +\frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}p}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\Bigg],\end{aligned} \tag{C.4}$$

and finally, Equation (2.40e),

$$
\begin{aligned}
\frac{\partial^{b+d+f+n+1} p}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^{n+1}} = &-\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\
&\frac{\partial^{b-a+d-c+f-e+n-m} u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\,\frac{\partial^{a+1+c+e+m} p}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}} \\
&+\frac{\partial^{b-a+d-c+f-e+n-m} v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\,\frac{\partial^{a+c+1+e+m} p}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\
&+\frac{\partial^{b-a+d-c+f-e+n-m} w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\,\frac{\partial^{a+c+e+1+m} p}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}} \\
&+\gamma\frac{\partial^{b-a+d-c+f-e+n-m} p}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\Bigg(\\
&\frac{\partial^{a+1+c+e+m} u}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}} \\
&+\frac{\partial^{a+c+1+e+m} v}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\
&+\frac{\partial^{a+c+e+1+m} w}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\Bigg)\Bigg].
\end{aligned}
\tag{C.5}
$$

Appendix D

Recurrence relations for the primitive variable specific volume Navier-Stokes equations in two dimensions

This appendix presents recurrence relations for the primitive variable specific volume form of the two-dimensional Navier-Stokes equations, i.e.,

$$\frac{\partial \sigma}{\partial t} + u\frac{\partial \sigma}{\partial x} + v\frac{\partial \sigma}{\partial y} - \sigma\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) = 0 \tag{D.1a}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \sigma\frac{\partial p}{\partial x} = \frac{4}{3}\mu\sigma\frac{\partial^2 u}{\partial x^2} + \frac{1}{3}\mu\sigma\frac{\partial^2 v}{\partial x\,\partial y} + \mu\sigma\frac{\partial^2 u}{\partial y^2} \tag{D.1b}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \sigma\frac{\partial p}{\partial y} = \mu\sigma\frac{\partial^2 v}{\partial x^2} + \frac{1}{3}\mu\sigma\frac{\partial^2 u}{\partial x\,\partial y} + \frac{4}{3}\mu\sigma\frac{\partial^2 v}{\partial y^2} \tag{D.1c}$$

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + \gamma p\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) = (\gamma - 1)(\Theta + \Phi) \tag{D.1d}$$

where

$$\Theta = \frac{\mathrm{k}}{\mathrm{R}}\left(\frac{\partial^2 \sigma}{\partial x^2}p + 2\frac{\partial \sigma}{\partial x}\frac{\partial p}{\partial x} + \sigma\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 \sigma}{\partial y^2}p + 2\frac{\partial \sigma}{\partial y}\frac{\partial p}{\partial y} + \sigma\frac{\partial^2 p}{\partial y^2}\right) \tag{D.2}$$

and

$$\Phi = \mu \left(\frac{4}{3} \frac{\partial u}{\partial x} \frac{\partial u}{\partial x} + \frac{4}{3} \frac{\partial v}{\partial y} \frac{\partial v}{\partial y} + \frac{\partial v}{\partial x} \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial u}{\partial y} + 2 \frac{\partial v}{\partial x} \frac{\partial u}{\partial y} - \frac{4}{3} \frac{\partial u}{\partial x} \frac{\partial v}{\partial y} \right). \tag{D.3}$$

The above equations assume the dynamic viscosity μ, thermal conductivity k, and gas constant R are all constant.

The recurrence relation for the Navier-Stokes continuity equation is identical to that of the Euler equations, but is repeated here for convenience:

$$\begin{aligned} \frac{\partial^{b+d+n+1}\sigma}{\partial x^b \, \partial y^d \, \partial t^{n+1}} = & - \sum_{a=0}^{b} \binom{b}{a} \sum_{c=0}^{d} \binom{d}{c} \sum_{m=0}^{n} \binom{n}{m} \Bigg[\\ & \frac{\partial^{b-a+d-c+n-m} u}{\partial x^{b-a} \, \partial y^{d-c} \, \partial t^{n-m}} \frac{\partial^{a+1+c+m} \sigma}{\partial x^{a+1} \, \partial y^{c} \, \partial t^{m}} \\ & + \frac{\partial^{b-a+d-c+n-m} v}{\partial x^{b-a} \, \partial y^{d-c} \, \partial t^{n-m}} \frac{\partial^{a+c+1+m} \sigma}{\partial x^{a} \, \partial y^{c+1} \, \partial t^{m}} \\ & - \frac{\partial^{b-a+d-c+n-m} \sigma}{\partial x^{b-a} \, \partial y^{d-c} \, \partial t^{n-m}} \Bigg(\\ & \frac{\partial^{a+1+c+m} u}{\partial x^{a+1} \, \partial y^{c} \, \partial t^{m}} \\ & + \frac{\partial^{a+c+1+m} v}{\partial x^{a} \, \partial y^{c+1} \, \partial t^{m}} \Bigg) \\ & \Bigg]. \end{aligned} \tag{B.2 repeated}$$

The recurrence relation for the 2D Navier-Stokes x-momentum equation used in this

work is

$$\begin{aligned}\frac{\partial^{b+d+n+1}u}{\partial x^b\,\partial y^d\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m}u}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m}p}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ - \mu\frac{\partial^{b-a+d-c+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\Bigg(\frac{4}{3}\frac{\partial^{a+2+c+m}u}{\partial x^{a+2}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{1}{3}\frac{\partial^{a+1+c+1+m}v}{\partial x^{a+1}\,\partial y^{c+1}\,\partial t^{m}} \\ + \frac{\partial^{a+c+2+m}u}{\partial x^{a}\,\partial y^{c+2}\,\partial t^{m}}\Bigg) \\ \Bigg],\end{aligned} \tag{D.4}$$

and the corresponding recurrence relation for the y-momentum equation is

$$\begin{aligned}\frac{\partial^{b+d+n+1}v}{\partial x^b\,\partial y^d\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m}v}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m}v}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m}p}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\ - \mu\frac{\partial^{b-a+d-c+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\Bigg(\frac{\partial^{a+2+c+m}v}{\partial x^{a+2}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{1}{3}\frac{\partial^{a+1+c+1+m}u}{\partial x^{a+1}\,\partial y^{c+1}\,\partial t^{m}} \\ + \frac{4}{3}\frac{\partial^{a+c+2+m}v}{\partial x^{a}\,\partial y^{c+2}\,\partial t^{m}}\Bigg) \\ \Bigg].\end{aligned} \tag{D.5}$$

Finally, the energy equation recurrence relation is

$$\begin{aligned}\frac{\partial^{b+d+n+1}p}{\partial x^b\,\partial y^d\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m}p}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m}p}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\ + \gamma\frac{\partial^{b-a+d-c+n-m}p}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\Bigg(\\ \frac{\partial^{a+1+c+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{a+c+1+m}v}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}}\Bigg) \\ - (\gamma-1)\left(\frac{\partial^{b+d+n}\Theta}{\partial x^b\,\partial y^d\,\partial t^n} + \frac{\partial^{b+d+n}\Phi}{\partial x^b\,\partial y^d\,\partial t^n}\right) \\ \Bigg],\end{aligned} \tag{D.6}$$

where

$$\begin{aligned}\frac{\partial^{b+d+n}\Theta}{\partial x^b\,\partial y^d\,\partial t^n} = \frac{\mathrm{k}}{\mathrm{R}}\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+2+d-c+n-m}\sigma}{\partial x^{b-a+2}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+m}p}{\partial x^{a}\,\partial y^{c}\,\partial t^{m}} \\ + 2\frac{\partial^{b-a+1+d-c+n-m}\sigma}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+1+c+m}p}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+2+c+m}p}{\partial x^{a+2}\,\partial y^{c}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+2+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c+2}\,\partial t^{n-m}}\frac{\partial^{a+c+m}p}{\partial x^{a}\,\partial y^{c}\,\partial t^{m}} \\ + 2\frac{\partial^{b-a+d-c+1+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c+1}\,\partial t^{n-m}}\frac{\partial^{a+c+1+m}p}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial t^{n-m}}\frac{\partial^{a+c+2+m}p}{\partial x^{a}\,\partial y^{c+2}\,\partial t^{m}} \\ \Bigg]\end{aligned} \tag{D.7}$$

and

$$
\begin{aligned}
\frac{\partial^{b+d+n}\Phi}{\partial x^b\,\partial y^d\,\partial t^n} = \mu \sum_{a=0}^{b} \binom{b}{a} \sum_{c=0}^{d} \binom{d}{c} \sum_{m=0}^{n} \binom{n}{m} \Bigg[\\
& \frac{4}{3} \frac{\partial^{b-a+1+d-c+n-m} u}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial t^{n-m}} \frac{\partial^{a+1+c+m} u}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\
& + \frac{4}{3} \frac{\partial^{b-a+d-c+1+n-m} v}{\partial x^{b-a}\,\partial y^{d-c+1}\,\partial t^{n-m}} \frac{\partial^{a+c+1+m} v}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\
& + \frac{\partial^{b-a+1+d-c+n-m} v}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial t^{n-m}} \frac{\partial^{a+1+c+m} v}{\partial x^{a+1}\,\partial y^{c}\,\partial t^{m}} \\
& + \frac{\partial^{b-a+d-c+1+n-m} u}{\partial x^{b-a}\,\partial y^{d-c+1}\,\partial t^{n-m}} \frac{\partial^{a+c+1+m} u}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\
& + 2 \frac{\partial^{b-a+1+d-c+n-m} v}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial t^{n-m}} \frac{\partial^{a+c+1+m} u}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\
& - \frac{4}{3} \frac{\partial^{b-a+1+d-c+n-m} u}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial t^{n-m}} \frac{\partial^{a+c+1+m} v}{\partial x^{a}\,\partial y^{c+1}\,\partial t^{m}} \\
\Bigg] .
\end{aligned}
\tag{D.8}
$$

Appendix E

Recurrence relations for the primitive variable specific volume Navier-Stokes equations in three dimensions

This appendix presents recurrence relations for the primitive variable specific volume form of the two-dimensional Navier-Stokes equations, i.e.,

$$\frac{\partial \sigma}{\partial t} + u\frac{\partial \sigma}{\partial x} + v\frac{\partial \sigma}{\partial y} + w\frac{\partial \sigma}{\partial z} - \sigma\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = 0 \qquad \text{(2.48a repeated)}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \sigma\frac{\partial p}{\partial x} = \mu\sigma\left(\frac{4}{3}\frac{\partial^2 u}{\partial x^2} + \frac{1}{3}\frac{\partial^2 v}{\partial x\,\partial y} + \frac{1}{3}\frac{\partial^2 w}{\partial x\,\partial z} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

(2.48b repeated)

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \sigma\frac{\partial p}{\partial y} = \mu\sigma\left(\frac{4}{3}\frac{\partial^2 v}{\partial y^2} + \frac{1}{3}\frac{\partial^2 u}{\partial x\,\partial y} + \frac{1}{3}\frac{\partial^2 w}{\partial y\,\partial z} + \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial z^2}\right)$$

(2.48c repeated)

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \sigma\frac{\partial p}{\partial z} = \mu\sigma\left(\frac{4}{3}\frac{\partial^2 w}{\partial z^2} + \frac{1}{3}\frac{\partial^2 u}{\partial x\,\partial z} + \frac{1}{3}\frac{\partial^2 v}{\partial y\,\partial z} + \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}\right)$$

(2.48d repeated)

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + w\frac{\partial p}{\partial z} + \gamma p\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = (\gamma - 1)\left(\Theta + \Phi\right).$$

(2.48e repeated)

where

$$\Theta = \frac{\mathrm{k}}{\mathrm{R}}\left(\frac{\partial^2\sigma}{\partial x^2}p + 2\frac{\partial\sigma}{\partial x}\frac{\partial p}{\partial x} + \sigma\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2\sigma}{\partial y^2}p + 2\frac{\partial\sigma}{\partial y}\frac{\partial p}{\partial y} + \sigma\frac{\partial^2 p}{\partial y^2} + \frac{\partial^2\sigma}{\partial z^2}p + 2\frac{\partial\sigma}{\partial z}\frac{\partial p}{\partial z} + \sigma\frac{\partial^2 p}{\partial z^2}\right) \tag{2.49 repeated}$$

and

$$\begin{aligned}\Phi = \mu\Bigg(\frac{4}{3}\Bigg[&\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\frac{\partial w}{\partial z} - \frac{\partial u}{\partial x}\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x}\frac{\partial w}{\partial z} - \frac{\partial v}{\partial y}\frac{\partial w}{\partial z}\Bigg] \\ &+ \frac{\partial v}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial w}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial u}{\partial y} + \frac{\partial w}{\partial y}\frac{\partial w}{\partial y} + \frac{\partial u}{\partial z}\frac{\partial u}{\partial z} + \frac{\partial v}{\partial z}\frac{\partial v}{\partial z} \\ &+2\left[\frac{\partial v}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial w}{\partial y}\frac{\partial v}{\partial z} + \frac{\partial u}{\partial z}\frac{\partial w}{\partial x}\right]\Bigg).\end{aligned} \tag{2.50 repeated}$$

The above equations assume the dynamic viscosity μ, thermal conductivity k, and gas constant R are all constant.

The recurrence relation for the Navier-Stokes continuity equation is identical to that of the Euler equations, but is repeated here for convenience:

$$\begin{aligned}\frac{\partial^{b+d+f+n+1}\sigma}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\ \frac{\partial^{b-a+d-c+f-e+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}\sigma}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+f-e+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}\sigma}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{b-a+d-c+f-e+n-m}w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}\sigma}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}} \\ - \frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\Bigg(\\ \frac{\partial^{a+1+c+e+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{a+c+1+e+m}v}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\ + \frac{\partial^{a+c+e+1+m}w}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\Bigg)\Bigg].\end{aligned} \tag{2.41c repeated}$$

The recurrence relation for the 3D Navier-Stokes x-momentum equation used in this work is

$$
\begin{aligned}
\frac{\partial^{b+d+f+n+1}u}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\
&\frac{\partial^{b-a+d-c+f-e+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+f-e+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}u}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+f-e+n-m}w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}u}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}p}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\\
&-\mu\frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\left(\frac{4}{3}\frac{\partial^{a+2+c+e+m}u}{\partial x^{a+2}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\right.\\
&+\frac{1}{3}\frac{\partial^{a+1+c+1+e+m}v}{\partial x^{a+1}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{1}{3}\frac{\partial^{a+1+c+e+1+m}w}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\
&\left.+\frac{\partial^{a+c+2+m+m}u}{\partial x^{a}\,\partial y^{c+2}\,\partial z^{m}\,\partial t^{m}}+\frac{\partial^{a+c+e+2+m}u}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+2}\,\partial t^{m}}\right)\\
&\Bigg],
\end{aligned}
\tag{E.2}
$$

and the corresponding recurrence relation for the y-momentum equation is

$$
\begin{aligned}
\frac{\partial^{b+d+f+n+1} v}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^{n+1}} = -\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\
\frac{\partial^{b-a+d-c+f-e+n-m} u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m} v}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}} \\
+\frac{\partial^{b-a+d-c+f-e+n-m} v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m} v}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\
+\frac{\partial^{b-a+d-c+f-e+n-m} w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m} v}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}} \\
+\frac{\partial^{b-a+d-c+f-e+n-m} \sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m} p}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\
-\mu\frac{\partial^{b-a+d-c+f-e+n-m} \sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\left(\frac{4}{3}\frac{\partial^{a+c+2+e+m} v}{\partial x^{a}\,\partial y^{c+2}\,\partial z^{e}\,\partial t^{m}}\right. \\
+\frac{1}{3}\frac{\partial^{a+1+c+1+e+m} u}{\partial x^{a+1}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}} \\
+\frac{1}{3}\frac{\partial^{a+c+1+e+1+m} w}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e+1}\,\partial t^{m}} \\
\left.+\frac{\partial^{a+2+c+m+m} v}{\partial x^{a+2}\,\partial y^{c}\,\partial z^{m}\,\partial t^{m}}+\frac{\partial^{a+c+e+2+m} v}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+2}\,\partial t^{m}}\right) \\
\Bigg],
\end{aligned}
\tag{E.3}
$$

and, for the z-momentum equation,

$$
\begin{aligned}
\frac{\partial^{b+d+f+n+1} w}{\partial x^b\, \partial y^d\, \partial z^f\, \partial t^{n+1}} = -\sum_{a=0}^{b} \binom{b}{a} \sum_{c=0}^{d} \binom{d}{c} \sum_{e=0}^{f} \binom{f}{e} \sum_{m=0}^{n} \binom{n}{m} \Bigg[\\
\frac{\partial^{b-a+d-c+f-e+n-m} u}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}\, \partial t^{n-m}} \frac{\partial^{a+1+c+e+m} w}{\partial x^{a+1}\, \partial y^{c}\, \partial z^{e}\, \partial t^{m}} \\
+ \frac{\partial^{b-a+d-c+f-e+n-m} v}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}\, \partial t^{n-m}} \frac{\partial^{a+c+1+e+m} w}{\partial x^{a}\, \partial y^{c+1}\, \partial z^{e}\, \partial t^{m}} \\
+ \frac{\partial^{b-a+d-c+f-e+n-m} w}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}\, \partial t^{n-m}} \frac{\partial^{a+c+e+1+m} w}{\partial x^{a}\, \partial y^{c}\, \partial z^{e+1}\, \partial t^{m}} \\
+ \frac{\partial^{b-a+d-c+f-e+n-m} \sigma}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}\, \partial t^{n-m}} \frac{\partial^{a+c+e+1+m} p}{\partial x^{a}\, \partial y^{c}\, \partial z^{e+1}\, \partial t^{m}} \\
- \mu \frac{\partial^{b-a+d-c+f-e+n-m} \sigma}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}\, \partial t^{n-m}} \Bigg(\frac{4}{3} \frac{\partial^{a+c+e+2+m} w}{\partial x^{a}\, \partial y^{c}\, \partial z^{e+2}\, \partial t^{m}} \\
+ \frac{1}{3} \frac{\partial^{a+1+c+e+1+m} u}{\partial x^{a+1}\, \partial y^{c}\, \partial z^{e+1}\, \partial t^{m}} \\
+ \frac{1}{3} \frac{\partial^{a+c+1+e+1+m} v}{\partial x^{a}\, \partial y^{c+1}\, \partial z^{e+1}\, \partial t^{m}} \\
+ \frac{\partial^{a+2+c+m+m} w}{\partial x^{a+2}\, \partial y^{c}\, \partial z^{m}\, \partial t^{m}} + \frac{\partial^{a+c+2+e+m} w}{\partial x^{a}\, \partial y^{c+2}\, \partial z^{e}\, \partial t^{m}} \Bigg) \\
\Bigg],
\end{aligned}
\tag{E.4}
$$

Finally, the energy equation recurrence relation is

$$
\begin{aligned}
\frac{\partial^{b+d+f+n+1} p}{\partial x^b\, \partial y^d\, \partial z^f\, \partial t^{n+1}} = -\sum_{a=0}^{b} \binom{b}{a} \sum_{c=0}^{d} \binom{d}{c} \sum_{e=0}^{f} \binom{f}{e} \sum_{m=0}^{n} \binom{n}{m} \Bigg[& \\
& \frac{\partial^{b-a+d-c+f-e+n-m} u}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}\, \partial t^{n-m}} \frac{\partial^{a+1+c+e+m} p}{\partial x^{a+1}\, \partial y^{c}\, \partial z^{e}\, \partial t^{m}} \\
& + \frac{\partial^{b-a+d-c+f-e+n-m} v}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}\, \partial t^{n-m}} \frac{\partial^{a+c+1+e+m} p}{\partial x^{a}\, \partial y^{c+1}\, \partial z^{e}\, \partial t^{m}} \\
& + \frac{\partial^{b-a+d-c+f-e+n-m} w}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}\, \partial t^{n-m}} \frac{\partial^{a+c+e+1+m} p}{\partial x^{a}\, \partial y^{c}\, \partial z^{e+1}\, \partial t^{m}} \\
& + \gamma \frac{\partial^{b-a+d-c+f-e+n-m} p}{\partial x^{b-a}\, \partial y^{d-c}\, \partial z^{f-e}\, \partial t^{n-m}} \Bigg(\\
& \quad \frac{\partial^{a+1+c+e+m} u}{\partial x^{a+1}\, \partial y^{c}\, \partial z^{e}\, \partial t^{m}} \\
& \quad + \frac{\partial^{a+c+1+e+m} v}{\partial x^{a}\, \partial y^{c+1}\, \partial z^{e}\, \partial t^{m}} \\
& \quad + \frac{\partial^{a+c+e+1+m} w}{\partial x^{a}\, \partial y^{c}\, \partial z^{e+1}\, \partial t^{m}} \Bigg) \\
& - (\gamma - 1) \left(\frac{\partial^{b+d+f+n} \Theta}{\partial x^b\, \partial y^d\, \partial z^f\, \partial t^n} + \frac{\partial^{b+d+f+n} \Phi}{\partial x^b\, \partial y^d\, \partial z^f\, \partial t^n} \right) \\
\Bigg], &
\end{aligned}
\tag{E.5}
$$

where

$$
\begin{aligned}
\frac{\partial^{b+d+n+n}\Theta}{\partial x^b\,\partial y^d\,\partial z^n\,\partial t^n} = \frac{\mathrm{k}}{\mathrm{R}}\sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\
\frac{\partial^{b-a+2+d-c+f-e+n-m}\sigma}{\partial x^{b-a+2}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+m}p}{\partial x^a\,\partial y^c\,\partial z^e\,\partial t^m} \\
+ 2\frac{\partial^{b-a+1+d-c+f-e+n-m}\sigma}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}p}{\partial x^{a+1}\,\partial y^c\,\partial z^e\,\partial t^m} \\
+ \frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+2+c+e+m}p}{\partial x^{a+2}\,\partial y^c\,\partial z^e\,\partial t^m} \\
+ \frac{\partial^{b-a+d-c+2+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c+2}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+m}p}{\partial x^a\,\partial y^c\,\partial z^e\,\partial t^m} \\
+ 2\frac{\partial^{b-a+d-c+1+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c+1}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}p}{\partial x^a\,\partial y^{c+1}\,\partial z^e\,\partial t^m} \\
+ \frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+2+e+m}p}{\partial x^a\,\partial y^{c+2}\,\partial z^e\,\partial t^m} \\
+ \frac{\partial^{b-a+d-c+f-e+2+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e+2}\,\partial t^{n-m}}\frac{\partial^{a+c+e+m}p}{\partial x^a\,\partial y^c\,\partial z^e\,\partial t^m} \\
+ 2\frac{\partial^{b-a+d-c+f-e+1+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e+1}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}p}{\partial x^a\,\partial y^c\,\partial z^{e+1}\,\partial t^m} \\
+ \frac{\partial^{b-a+d-c+f-e+n-m}\sigma}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+2+m}p}{\partial x^a\,\partial y^c\,\partial z^{e+2}\,\partial t^m} \\
\Bigg]
\end{aligned}
\tag{E.6}
$$

and

$$
\begin{aligned}
\frac{\partial^{b+d+f+n}\Phi}{\partial x^b\,\partial y^d\,\partial z^f\,\partial t^n} = \mu \sum_{a=0}^{b}\binom{b}{a}\sum_{c=0}^{d}\binom{d}{c}\sum_{e=0}^{f}\binom{f}{e}\sum_{m=0}^{n}\binom{n}{m}\Bigg[\\
&\frac{4}{3}\Bigg(\\
&\frac{\partial^{b-a+1+d-c+f-e+n-m}u}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}u}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+1+f-e+n-m}v}{\partial x^{b-a}\,\partial y^{d-c+1}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}v}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+f-e+1+n-m}w}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e+1}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}w}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\
&-\frac{\partial^{b-a+1+d-c+f-e+n-m}u}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}v}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\\
&-\frac{\partial^{b-a+1+d-c+f-e+n-m}u}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}w}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\
&-\frac{\partial^{b-a+d-c+1+f-e+n-m}v}{\partial x^{b-a}\,\partial y^{d-c+1}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}w}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\
&\Bigg)\\
&+\frac{\partial^{b-a+1+d-c+f-e+n-m}v}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}v}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+1+d-c+f-e+n-m}w}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}w}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+1+f-e+n-m}u}{\partial x^{b-a}\,\partial y^{d-c+1}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}u}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+1+f-e+n-m}w}{\partial x^{b-a}\,\partial y^{d-c+1}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}w}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+f-e+1+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e+1}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}u}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+f-e+1+n-m}v}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e+1}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}v}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\
&+2\Bigg(\\
&\frac{\partial^{b-a+1+d-c+f-e+n-m}v}{\partial x^{b-a+1}\,\partial y^{d-c}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+1+e+m}u}{\partial x^{a}\,\partial y^{c+1}\,\partial z^{e}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+1+f-e+n-m}w}{\partial x^{b-a}\,\partial y^{d-c+1}\,\partial z^{f-e}\,\partial t^{n-m}}\frac{\partial^{a+c+e+1+m}v}{\partial x^{a}\,\partial y^{c}\,\partial z^{e+1}\,\partial t^{m}}\\
&+\frac{\partial^{b-a+d-c+f-e+1+n-m}u}{\partial x^{b-a}\,\partial y^{d-c}\,\partial z^{f-e+1}\,\partial t^{n-m}}\frac{\partial^{a+1+c+e+m}w}{\partial x^{a+1}\,\partial y^{c}\,\partial z^{e}\,\partial t^{m}}\\
&\Bigg)
\end{aligned}
\tag{E.7}
$$

$$
\Bigg].
$$

Appendix F

A Python implementation of EVA

The following is `ck_recursion.py`, a Python module that defines a class that performs Cauchy-Kowalewski recursion on a system of partial differential equations.

```
import sympy as sp
from copy import deepcopy

class CKRecursion:

  # This class should do CK recursion on a PDE or system of PDEs.

  def __init__(self, spatial_vars, temporal_var, dep_vars, eqns, temporal_order=1):

    if temporal_order < 1:
      raise(ValueError('invalid temporal order '
                       '{}'.format(temporal_order)))

    if temporal_var in spatial_vars:
      raise(ValueError('temporal_var {} should not be in spatial_vars = '
                       '{}'.format(temporal_var, spatial_vars)))

    self._ind_vars = tuple(spatial_vars) + (temporal_var,)

    for var in self._ind_vars:
      if var in dep_vars:
        raise(ValueError('independent var {} in dep_vars '
                         '{}'.format(var, dep_vars)))

    if len(eqns) != len(dep_vars):
      eqnlen = len(eqns)
      dep_varlen = len(dep_vars)
      raise(ValueError('number of equations {} does not match number of dependent variables {}'.format(eqnlen, dep_va

    self._dep_vars = dep_vars
    self._num_spatial_vars = len(self._ind_vars) - 1

    self._derivs = {}
    order = (self._num_spatial_vars + 1)*[0,]
    order[-1] = temporal_order
    wrt = self._get_diff_wrt(order)
    for var, eqn in zip(self._dep_vars, eqns):
      self._derivs[var.diff(*wrt)] = eqn

    self._min_temporal_order = temporal_order
```

```
  self._loc = {}
  self._derivs_eval = {}
  self.verbose = False

@property
def num_spatial_vars(self):
  return self._num_spatial_vars

@property
def min_temporal_order(self):
  return self._min_temporal_order

@property
def ind_vars(self):
  return self._ind_vars

@property
def dep_vars(self):
  return self._dep_vars

@property
def derivs(self):
  return deepcopy(self._derivs)

@property
def derivs_eval(self):
  return deepcopy(self._derivs_eval)

def _get_order(self, expr):

  var = None
  order = tuple()
  if expr.is_Function:
    if expr in self._dep_vars:
      var = expr
      order = (self._num_spatial_vars + 1)*(0,)

  elif expr.is_Derivative:
    var = expr.args[0]
    if var in self._dep_vars:
      deriv_vars = expr.args[1:]
      order = (self._num_spatial_vars + 1)*[0,]
      for v in deriv_vars:
        order[self._ind_vars.index(v)] += 1
      order = tuple(order)

  return var, order

def _get_lower_order(self, order):

  lower_order = list(order)
  for wrt in range(len(self._ind_vars)):
    if lower_order[wrt] > 0:
      lower_order[wrt] -= 1
      lower_order = tuple(lower_order)
      break

  return self._ind_vars[wrt], tuple(lower_order)

def _get_diff_wrt(self, order):

  if len(order) > len(self._ind_vars):
    raise(ValueError("too many order values passed."))

  wrt = []
  for v, o in zip(self._ind_vars, order):
    wrt.append(v)
    wrt.append(o)
```

```
    return wrt

  def find_deriv(self, dep_var, order):
    """Use CK recursion to find the "order"-th derivative."""

    # Check the input.
    if not dep_var in self._dep_vars:
      raise(ValueError('invalid dep_var {}'.format(dep_var)))

    if len(order) != len(self._ind_vars):
      raise(ValueError('len(order) should be {}, but is {}'.format(
        len(self._ind_vars), len(order))))

    if any([o < 0 for o in order]):
      raise(ValueError('invalid entry in order = {} found'.format(order)))

    wrt = self._get_diff_wrt(order)
    deriv = dep_var.diff(*wrt)

    if deriv in self._derivs or order[-1] < self._min_temporal_order:
      pass

    else:
      if self.verbose:
        print("Finding {}...".format(deriv))

      # Recursive subroutine call to get a lower-order derivative.
      wrt, lower_order = self._get_lower_order(order)
      lower_deriv = dep_var.diff(*self._get_diff_wrt(lower_order))
      if self.verbose:
        print("Recursive call for {}...".format(lower_deriv))
      self.find_deriv(dep_var, lower_order)

      # Differentiate the lower-order derivative we just found to get
      # the desired derivative.
      if self.verbose:
        print("Differentiating {} to get {}...".format(lower_deriv,
          deriv))
      self._derivs[deriv] = self._derivs[lower_deriv].diff(wrt)

      # Loop through all the terms in the new derivative, looking for
      # even more new derivatives that we might need.
      if self.verbose:
        print("Checking {} for new derivatives...".format(deriv))
      for arg in sp.preorder_traversal(self._derivs[deriv]):
        new_var, new_order = self._get_order(arg)
        if new_order:
          self.find_deriv(new_var, new_order)

    return

  def print_derivs(self):

    print('Derivatives:')
    for deriv, expr in self._derivs.items():
      print("{} = {}".format(deriv, expr))

    print('Evaluated derivatives:')
    for deriv, expr in self._derivs_eval.items():
      print("{} = {}".format(deriv, expr))

    print('Location:')
    for var, val in self._loc.items():
      print("{} = {}".format(var, val))

    return
```

```
def set_ic(self, ic):

  try:
    ic_len = len(ic)
    my_ic = tuple(ic)
  except TypeError:
    ic_len = 1
    my_ic = (ic,)

  if ic_len != len(self._dep_vars)*self._min_temporal_order:
    raise(ValueError("number of ICs does not match dependant variables."))

  order = (self._num_spatial_vars + 1)*[0,]
  k = 0
  for j in range(self._min_temporal_order):
    order[-1] = j
    for dep_var in self._dep_vars:
      wrt = self._get_diff_wrt(order)
      deriv = dep_var.diff(*wrt)
      self._derivs[deriv] = my_ic[k]
      k += 1

  return

def set_loc(self, loc):

  try:
    loc_len = len(loc)
    my_loc = tuple(loc)
  except TypeError:
    loc_len = 1
    my_loc = (loc,)

  if loc_len != self._num_spatial_vars:
    raise(ValueError(
      "number of spatial locations does not match {}".format(self._num_spatial_vars)
      ))

  for var, val in zip(self._ind_vars, my_loc):
    self._loc[var] = val

  self._derivs_eval = {}
  order = (self._num_spatial_vars + 1)*[0,]

  k = 0
  for j in range(self._min_temporal_order):
    order[-1] = j
    for dep_var in self._dep_vars:
      wrt = self._get_diff_wrt(order)
      deriv = dep_var.diff(*wrt)
      self._derivs_eval[deriv] = self._derivs[deriv].subs(self._loc)
      k += 1


  return

def evaluate_derivs(self, dep_var, order):

  # First, get all the "symbolic" derivatives.
  self.find_deriv(dep_var, order)

  # Next, evaluate the derivatives.
  deriv = dep_var.diff(*self._get_diff_wrt(order))

  if deriv in self._derivs_eval:
    pass

  elif order[-1] < self._min_temporal_order:
```

```
      wrt, lower_order = self._get_lower_order(order)
      self.evaluate_derivs(dep_var, lower_order)

      lower_deriv = dep_var.diff(*self._get_diff_wrt(lower_order))
      self._derivs[deriv] = self._derivs[lower_deriv].diff(wrt)
      self._derivs_eval[deriv] = self._derivs[deriv].subs(self._loc)

    else:

      for arg in sp.preorder_traversal(self._derivs[deriv]):
        new_var, new_order = self._get_order(arg)
        if new_order:
          self.evaluate_derivs(new_var, new_order)

      self._derivs_eval[deriv] = self._derivs[deriv].subs(self._derivs_eval)

    return

if __name__ == '__main__':

  gam = 1.4

  x = sp.Symbol('x')
  t = sp.Symbol('t')

  sv = sp.Function('sv')(x, t)
  u = sp.Function('u')(x, t)
  p = sp.Function('p')(x, t)

  dsvdt = -(u*sv.diff(x) - sv*u.diff(x))
  dudt = -(u*u.diff(x) + sv*p.diff(x))
  dpdt = -(u*p.diff(x) + gam*p*u.diff(x))

  max_order = 0

  ck = CKRecursion(eqns         = (dsvdt, dudt, dpdt),
                   ind_vars     = (x, t),
                   temporal_var = t,
                   dep_vars     = (sv, u, p)
                  )
  fac = -sp.log(2.)
  sv_ic = 1.005 + 0.1*10.*sp.exp(fac*((x - 2.003)/4.)**2)*sp.sin(0.1*x + 0.)
  u_ic  = 1.205 + 0.2*12.*sp.exp(fac*((x - 1.004)/4.)**2)*sp.sin(0.2*x + 2.)
  p_ic  = 3.205 + 0.4*14.*sp.exp(fac*((x - 3.004)/4.)**2)*sp.sin(0.4*x + 3.)
  ck.set_ic(ic  = (sv_ic, u_ic, p_ic))

  ck.set_loc(loc = (1.,))

  ck.evaluate_derivs(sv, (0, max_order))
  ck.evaluate_derivs(u, (0, max_order))
  ck.evaluate_derivs(p, (0, max_order))
  ck.print_derivs()

# vim:sw=2
```

The EVA Taylor series, Equation (2.6), is constructed by the `EVA` class in `eva.py`, which inherits the `CKRecursion` object from `ck_recursion.py`.

```
from math import factorial
import numpy as np
from ck_recursion import CKRecursion

class EVA(CKRecursion):
```

```
  def __init__(self, spatial_vars, temporal_var, dep_vars, gov_eqns,
      ic=None, temporal_order=1):

    CKRecursion.__init__(self,
      spatial_vars = spatial_vars,
      temporal_var = temporal_var,
      dep_vars = dep_vars,
      eqns = gov_eqns,
      temporal_order = temporal_order
    )

    if ic:
      self.set_ic(ic)

    return

  def do_it(self, grid, dt, ts_order=10):

    if grid.ndim == 1:
      grid = grid[np.newaxis, ...]
      UNDO_AXIS_MOD = True
    else:
      UNDO_AXIS_MOD = False

    if grid.shape[0] != len(self._ind_vars) - 1:
      raise(ValueError(("Size of first dimension of grid should be {}, "
        "but is {}.").format(len(self.ind_vars)-1, grid.shape[0])))

    solution = np.zeros((len(self.dep_vars),) + grid.shape[1:])

    order = len(self.ind_vars)*[0,]

    grid_iter = np.nditer(np.split(grid, grid.shape[0], axis=0))
    sol_iter = np.nditer(np.split(solution, solution.shape[0], axis=0),
      op_flags = ['writeonly'])

    for loc, sol in zip(grid_iter, sol_iter):
      self.set_loc(loc)

      for o in range(ts_order+1):
        order[-1] = o

        for v, s in zip(self.dep_vars, sol):
          self.evaluate_derivs(v, order)
          deriv = v.diff(*self._get_diff_wrt(order)) # Yuk!
          s[...] += dt**o/factorial(o)*self._derivs_eval[deriv]

    if UNDO_AXIS_MOD:
      grid = np.squeeze(grid, axis=0)

    return solution

if __name__ == "__main__":
  import sympy as sp

  gam = 1.4
  dt = 0.1

  x = sp.Symbol('x')
  t = sp.Symbol('t')

  sv = sp.Function('sv')(x, t)
  u = sp.Function('u')(x, t)
  p = sp.Function('p')(x, t)

  dsvdt = -(u*sv.diff(x) - sv*u.diff(x))
  dudt = -(u*u.diff(x) + sv*p.diff(x))
  dpdt = -(u*p.diff(x) + gam*p*u.diff(x))
```

```
    fac = -sp.log(2.)
    sv_ic = 1.005 + 0.1*10.*sp.exp(fac*((x - 2.003)/4.)**2)*sp.sin(0.1*x + 0.)
    u_ic  = 1.205 + 0.2*12.*sp.exp(fac*((x - 1.004)/4.)**2)*sp.sin(0.2*x + 2.)
    p_ic  = 3.205 + 0.4*14.*sp.exp(fac*((x - 3.004)/4.)**2)*sp.sin(0.4*x + 3.)

    eva = EVA(gov_eqns = (dsvdt, dudt, dpdt),
              ind_vars = (x, t),
              temporal_var = t,
              dep_vars = (sv, u, p),
              ic = (sv_ic, u_ic, p_ic))

    grid = np.linspace(0., 1., 11)

    solution = eva.do_it(grid, dt = dt, ts_order=2)

    print(solution)

# vim:sw=2
```

Appendix G

Comprehensive spatial test case results

Complete $l_{\max}$ results for both the Euler and Navier-Stokes spatial differencing verification test cases are presented in this chapter.

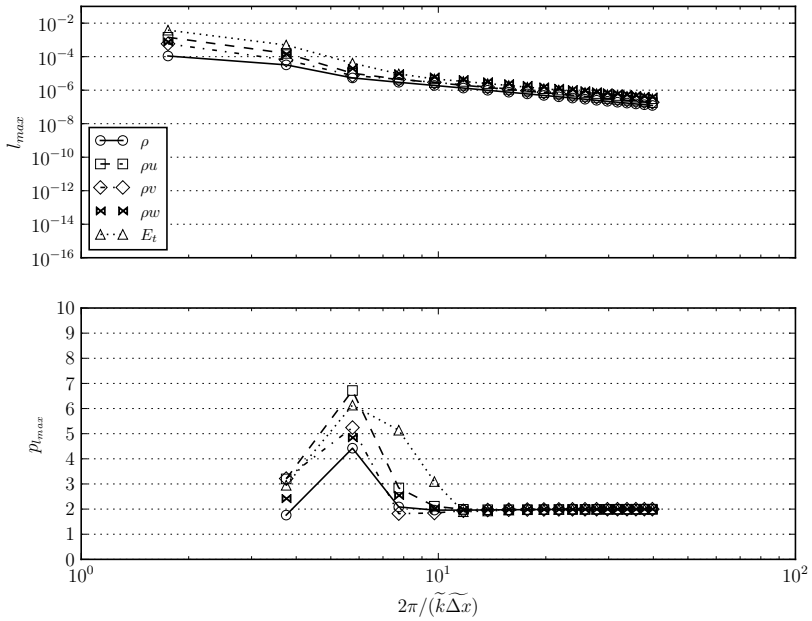


Figure G-1: $l_{\max}$ error and convergence rate for the E_2 scheme with Equation (3.8) $p_{l_{\max}}$ calculation. Inviscid results.

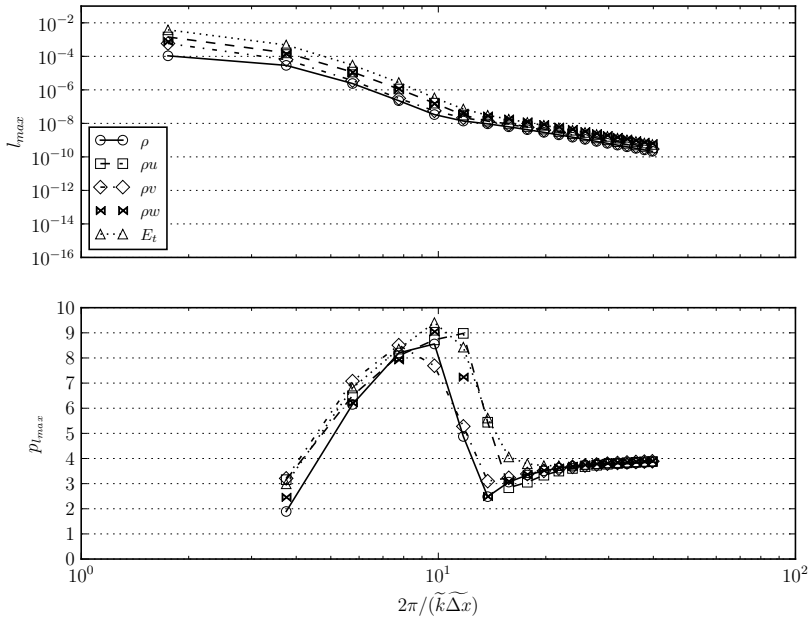


Figure G-2: $l_{\max}$ error and convergence rate for the DRP scheme with Equation (3.8) $p_{l_{\max}}$ calculation. Inviscid results.

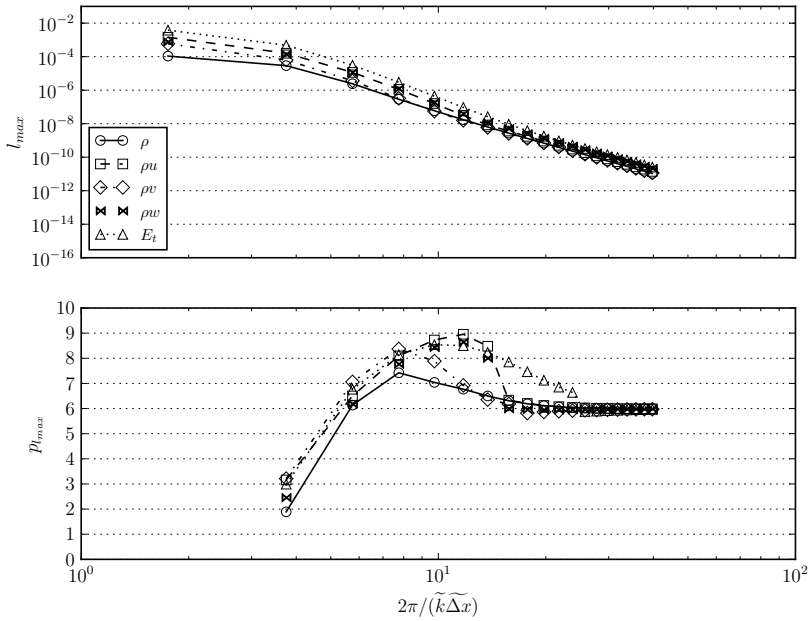


Figure G-3: $l_{\max}$ error and convergence rate for the E_6 scheme with Equation (3.8) $p_{l_{\max}}$ calculation. Inviscid results.

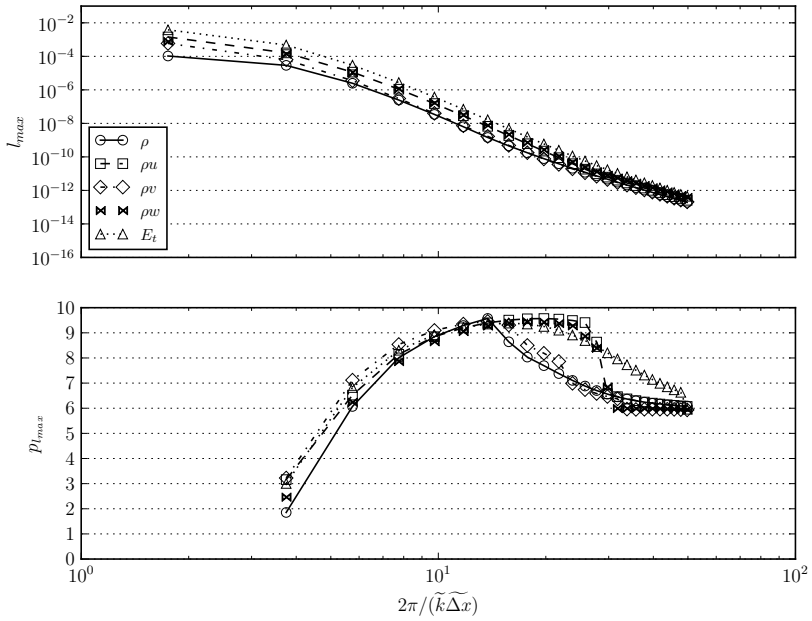


Figure G-4: $l_{\max}$ error and convergence rate for the C_6 scheme with Equation (3.8) $p_{l_{\max}}$ calculation. Inviscid results.

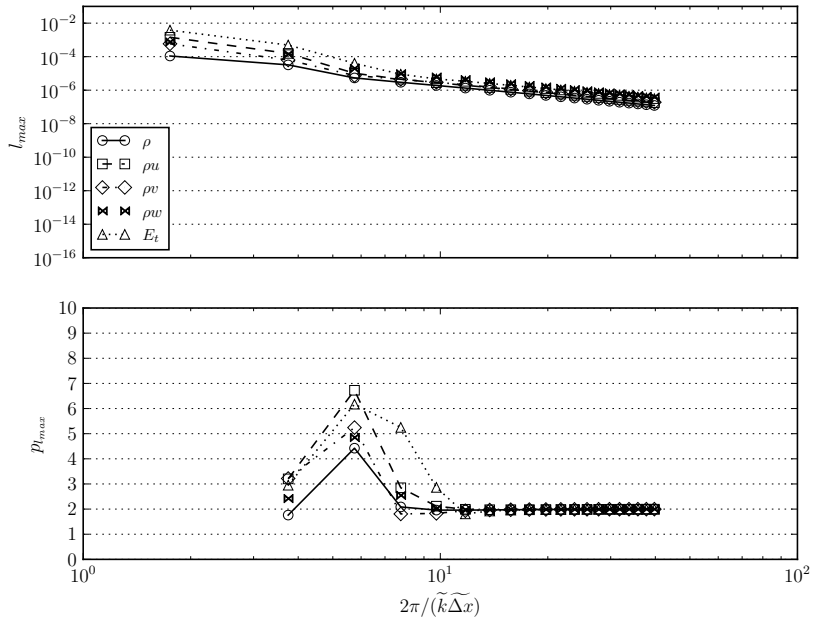


Figure G-5: $l_{\max}$ error and convergence rate for the E_2 scheme with Equation (3.8) $p_{l_{\max}}$ calculation. Viscous results.

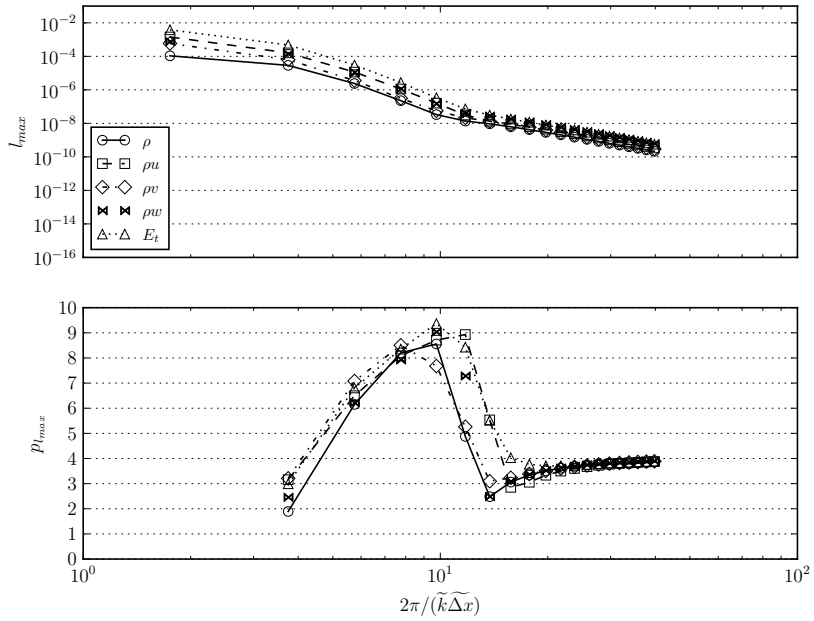


Figure G-6: $l_{\max}$ error and convergence rate for the DRP scheme with Equation (3.8) $p_{l_{\max}}$ calculation. Viscous results.

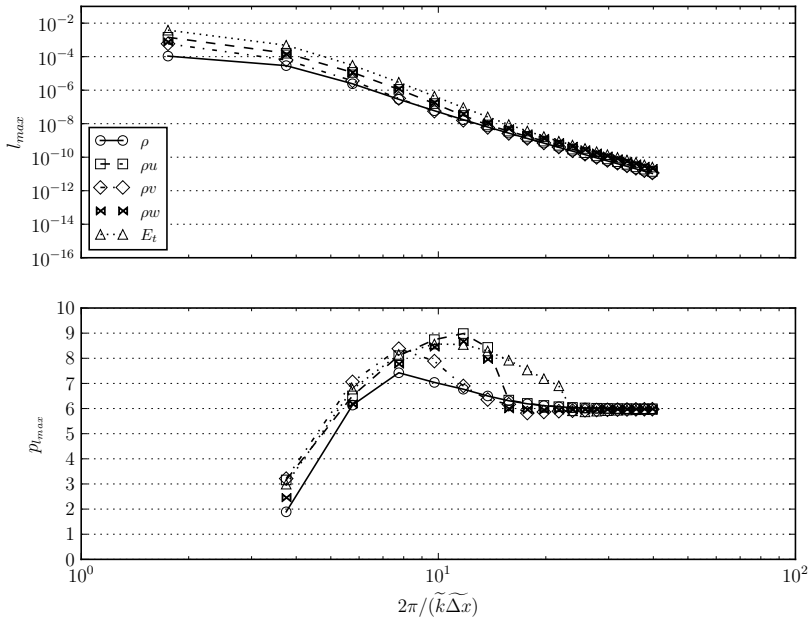


Figure G-7: $l_{\max}$ error and convergence rate for the E_6 scheme with Equation (3.8) $p_{l_{\max}}$ calculation. Viscous results.

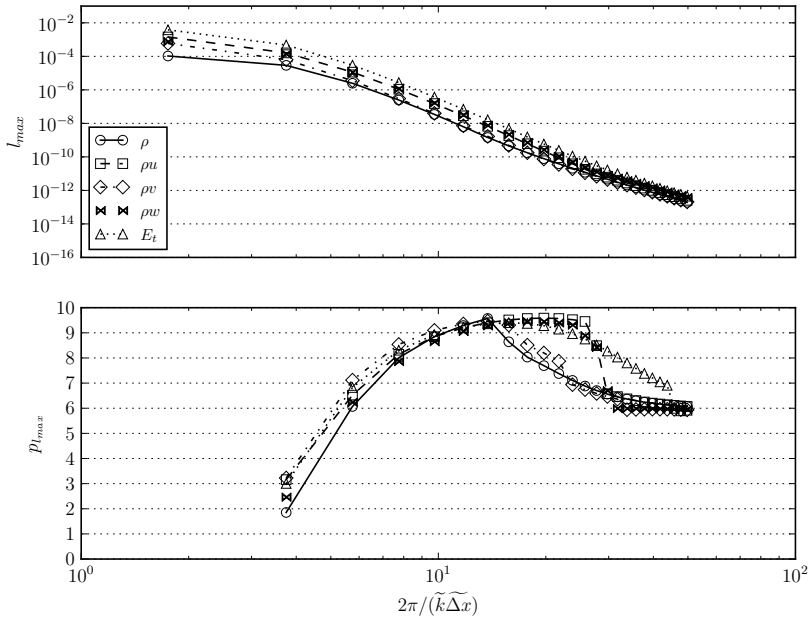


Figure G-8: $l_{\max}$ error and convergence rate for the C_6 scheme with Equation (3.8) $p_{l_{\max}}$ calculation. Viscous results.

Appendix H

Comprehensive temporal test case results

Complete $l_{\max}$ results for both the Euler and Navier-Stokes time-marching verification test cases are presented in this chapter.

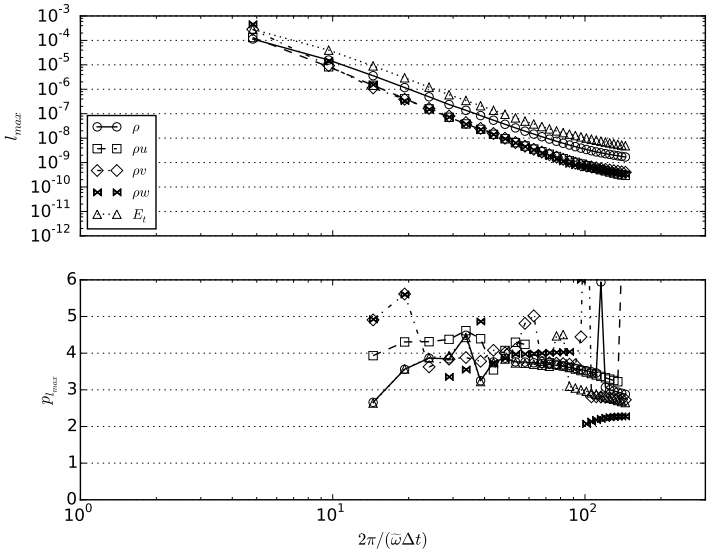


Figure H-1: $l_{\max}$ error and convergence rate for the RK4L scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Inviscid results.

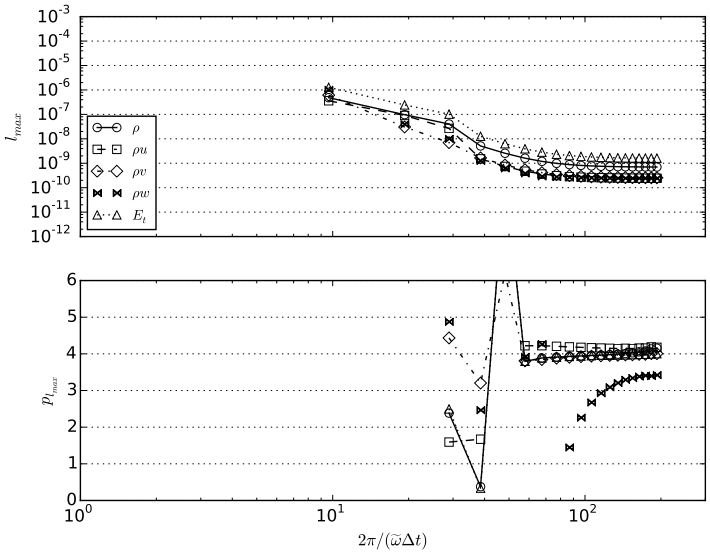


Figure H-2: $l_{\max}$ error and convergence rate for the RK56 scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Inviscid results.

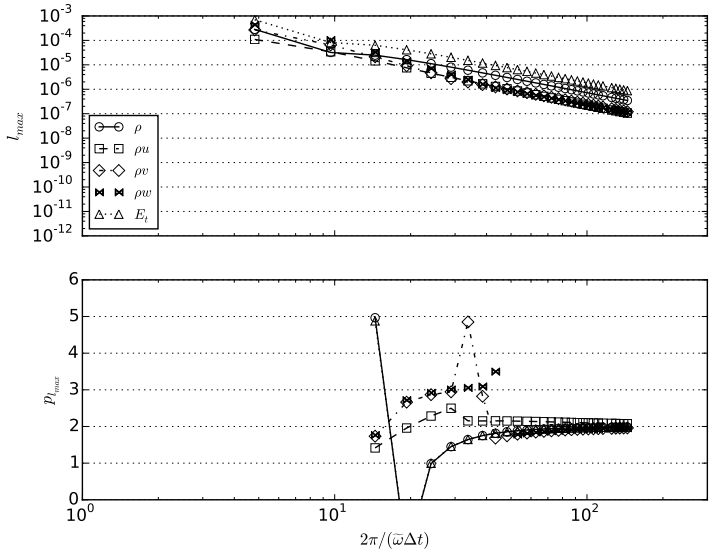


Figure H-3: $l_{\max}$ error and convergence rate for the RK5L scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Inviscid results.

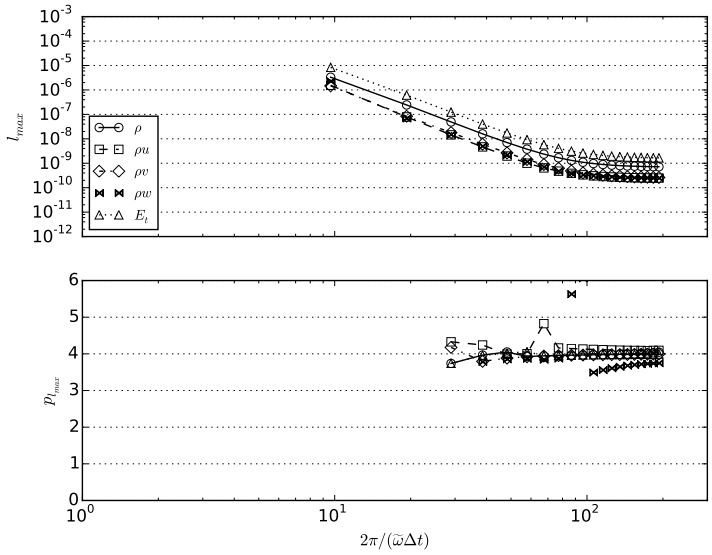


Figure H-4: $l_{\max}$ error and convergence rate for the RK67 scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Inviscid results.

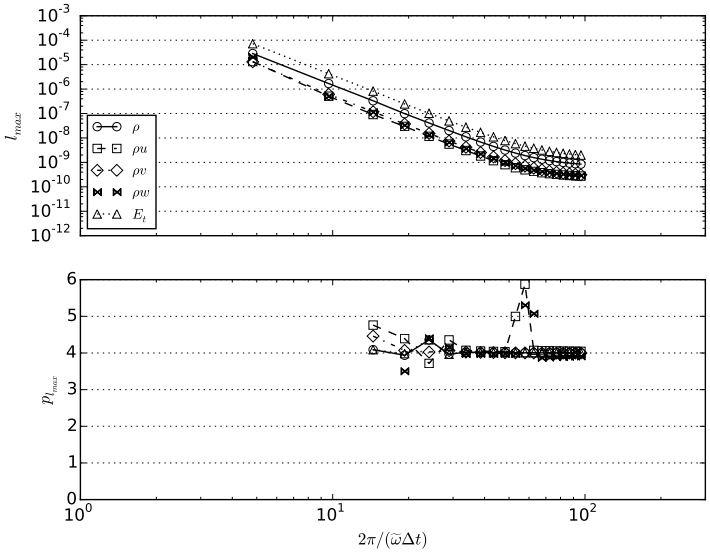


Figure H-5: $l_{\max}$ error and convergence rate for the RK7S scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Inviscid results.

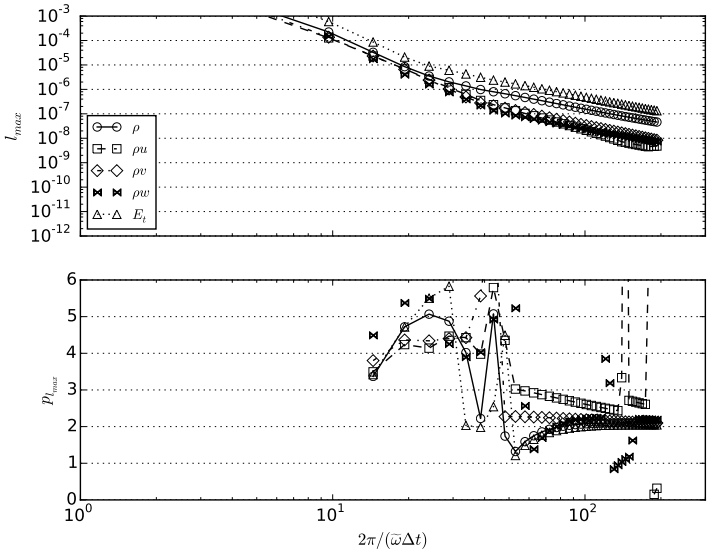


Figure H-6: $l_{\max}$ error and convergence rate for the RK4L scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Inviscid results with large-perturbation initial condition.

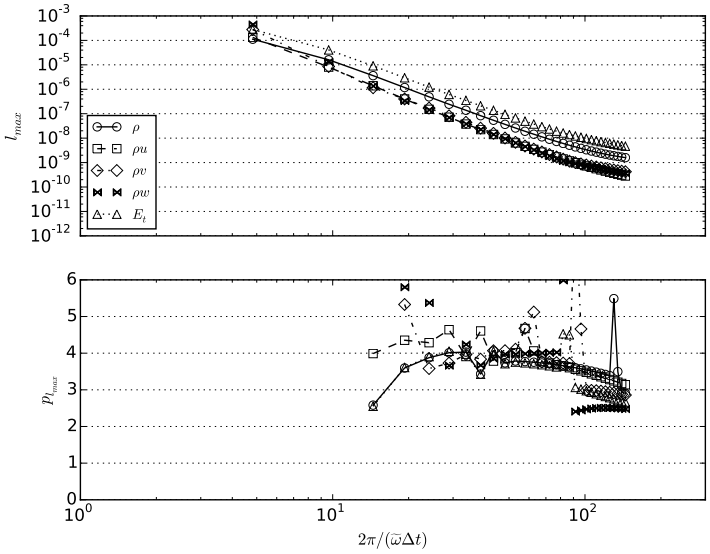


Figure H-7: $l_{\max}$ error and convergence rate for the RK4L scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Viscous results.

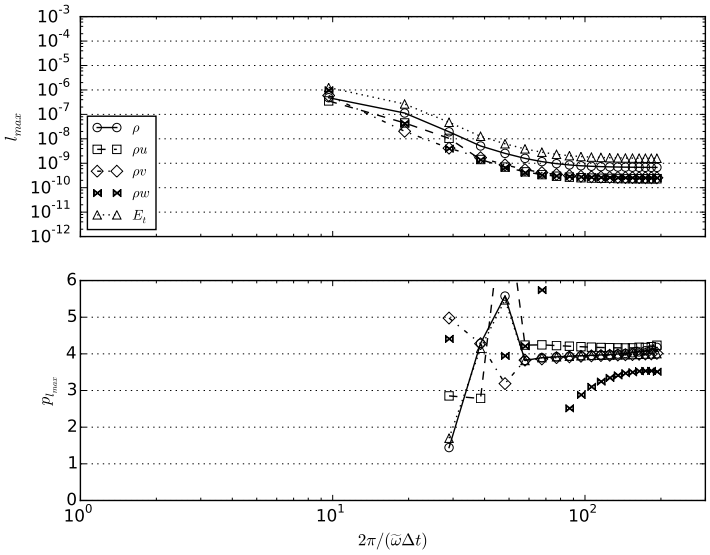


Figure H-8: $l_{\max}$ error and convergence rate for the RK56 scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Viscous results.

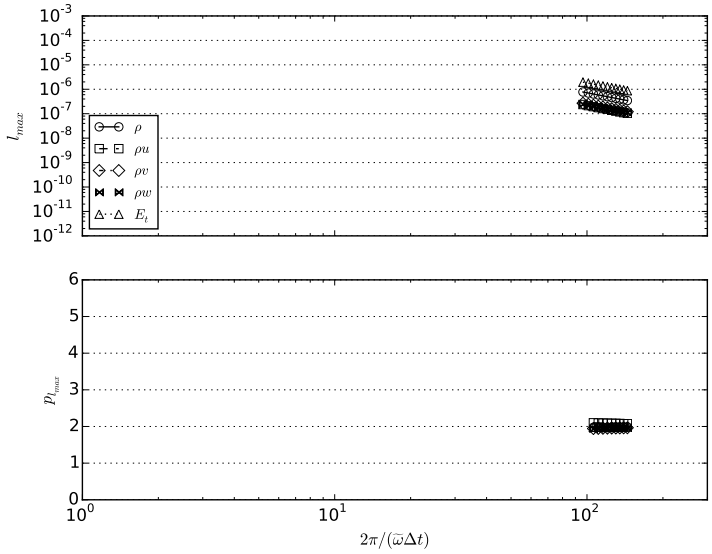


Figure H-9: $l_{\max}$ error and convergence rate for the RK5L scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Viscous results.

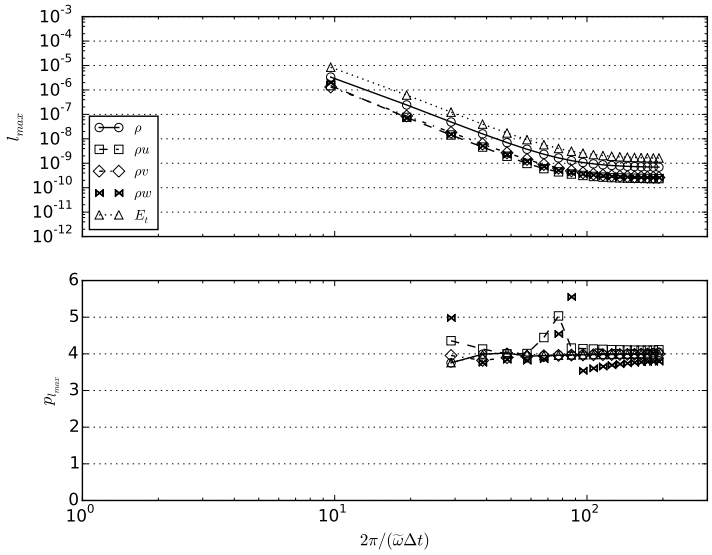


Figure H-10: $l_{\max}$ error and convergence rate for the RK67 scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Viscous results.

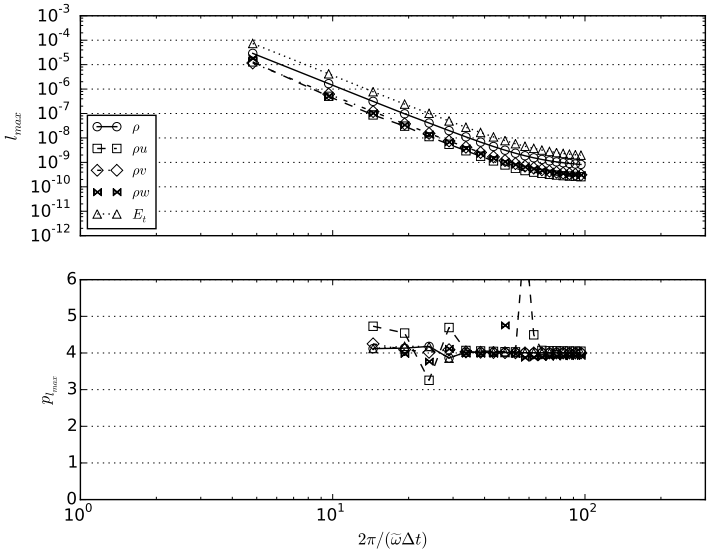


Figure H-11: $l_{\max}$ error and convergence rate for the RK7S scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Viscous results.

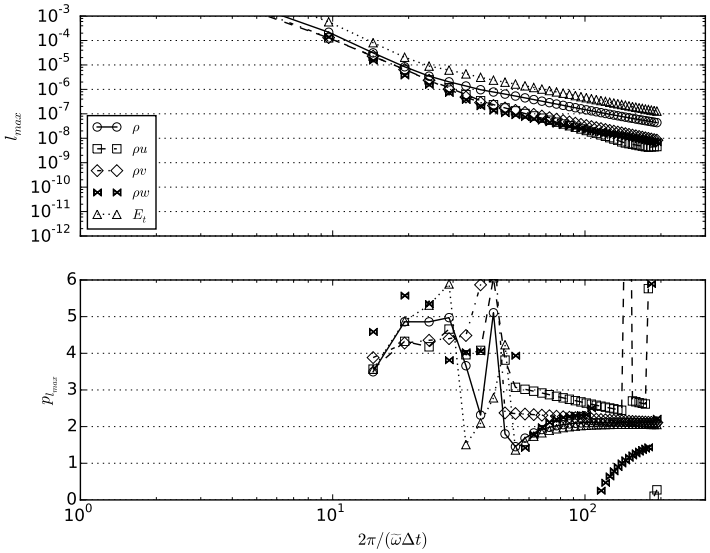


Figure H-12: $l_{\max}$ error and convergence rate for the RK4L scheme with Equation (3.12) $p_{l_{\max}}$ calculation. Viscous results with large-perturbation initial condition.