Imperial College London

Analysis of transport schemes for numerical weather prediction

Department of Aeronautics, Imperial College London

MSc Advanced Computational Methods

Author: Sahir Khan

Supervisor: Dr Colin Cotter

Submission Date: 13th of September 2013

Abstract

The report investigates the suitability of applying second and third order Taylor-Galerkin Finite Element schemes to advection dominated problems such as those found when carrying out numerical weather prediction. It presents arguments for why such schemes are useful and puts the work in context with current research. A theoretical stability analysis is derived and implemented using the Python programming language. The results of the stability analysis are verified by applying them to a Finite Element approximation of the one-dimensional linear advection equation using the FEniCS Finite Element library. It is found that the scheme offsets dispersion error at high wave-numbers with sufficient amounts of damping, even at large Courant numbers.

Acknowledgements

I would like to thank my project supervisor, Dr Colin Cotter, for his enthusiasm and patience during our weekly meetings and for his clarity of explanation. I would also like to thank my family and friends for their continued support and companionship over the duration of this MSc course. In particular, I would like to thank Pablo Crotti and Julia Pantoglou for the many hours that they spent proof reading my work, as well as the invaluable advice that they provided when it came to helping me complete this report.

Contents

1 Introduction and Motivation

1.1 The Shallow Water Equations

The aim of this project is to investigate the suitability of applying a new numerical method to a particular part of a mixed finite element formulation for solving the nonlinear rotating shallow water equations on the sphere presented by Cotter and Shipton (2012). These equations are given by McRae and Cotter (2013) as

$$\frac{\partial h}{\partial t} + \nabla \cdot (h\boldsymbol{u}) = 0, \tag{1.1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla) + f\boldsymbol{k} \times \boldsymbol{u} = -g\nabla h, \tag{1.2}$$

where h is the fluid thickness, $\boldsymbol{u}$ is the fluid velocity, $\boldsymbol{k}$ is the local 'upward pointing' unit vector, g is gravitational acceleration and f is the Coriolis parameter. By carrying out some intermediate steps, (1.2) can be recast as

$$\frac{\partial \boldsymbol{u}}{\partial t} + q(h\boldsymbol{u}^{\perp}) + \nabla\left(gh + K\right) = 0, \tag{1.3}$$

where $\boldsymbol{u}^{\perp} = \boldsymbol{k} \times \boldsymbol{u}$, $K = \frac{|u|^2}{2}$ is the kinetic energy and q is the potential vorticity, given by $q = \frac{\boldsymbol{k}\cdot\nabla\times\boldsymbol{u}+f}{h}$. The term $\boldsymbol{k} \cdot \nabla \times \boldsymbol{u}$ is known as the relative vorticity and can be assigned to the symbol ζ, while the numerator of the expression for q, given by $\zeta + f$, is known as the absolute vorticity.

Equation (1.3) can then be used to obtain a continuity equation for the absolute vorticity by applying the two-dimensional divergence operator, $\nabla^{\perp}\cdot$, to the equation to give

$$\frac{\partial\left(qh\right)}{\partial t} + \nabla \cdot (qh\boldsymbol{u}) = 0, \tag{1.4}$$

where it has been assumed that $\frac{\partial f}{\partial t} = 0$. This allows for the derivation of an advection equation for the potential vorticity. Multiplying equation (1.1) by q and comparing it to equation (1.4), while applying the product rule results in the expression

$$h\left[\frac{\partial q}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, q\right] = 0. \tag{1.5}$$

This implies that q remains constant with respect to time as it is advected by the fluid. According to Ringler et al. (2010), this property makes q a very useful parameter as it has the ability to 'inform' other physical processes in both a local sense, by acting as a Lagrangian tracer, and also in a global sense, through a concept known as the principle of invertibility. Another property that comes into play when constructing numerical time-stepping schemes, is that the potential vorticity also contains both of the terms $\boldsymbol{u}$ and h. This makes it an important diagnostic variable when numerically approximating other terms in the equations and allows it to form part of an intermediate step during the solution process.

1.2 Conventional Approaches to Solving the Shallow Water Equations

Developing numerical methods for solving these equations on the sphere is a key stepping stone on the journey to developing a dynamical core for the fully compressible, three dimensional baroclinic problem posed by real world atmospheric dynamics. This is because they need to over come many of the same obstacles and difficulties that would arise in the horizontal aspects of a fully three dimensional model (Williamson et al., 1992).

Historically, many of the conventional approaches to modelling the shallow water equations have involved using orthogonal latitude-longitude grids and finite difference methods for carrying out the numerical approximations. The primary disadvantage of such methods is that the convergence of grid lines near the poles results in so called 'resolution clustering' that severely limits the scalability of such methods to massively parallel computing (Staniforth and Thuburn, 2012).

One solution to this problem is to construct pseudo-uniform grids that replicate many of the positive numerical aspects of traditional latitude-longitude grids such as the staggered Arakawa C-grid (McRae and Cotter, 2013). These properties include the prevention of spurious numerical wave propagation, the prevention of artificial vorticity generation via the geopotential and pressure gradients, and the conservation of energy by terms involving both pressure and the Coriolis parameter. According to Staniforth and Thuburn (2012), these properties are related to the ability of the discretisation scheme to mimic certain mathematical properties of the continuous equations. This includes the ability to satisfy certain vector calculus identities in a discrete manner. Such schemes are commonly referred to as 'mimetic' schemes.

1.3 Mixed Finite Element Methods

Unfortunately, much of the work involved with extending finite difference methods to pseudo-uniform grids while preserving such positive attributes, has been unsuccessful as mentioned by McRae and Cotter (2013). However, a viable method that has emerged is the mixed finite element method. The method utilises different finite element spaces for approximating the velocity, pressure, fluid thickness, and potential vorticity, which in turn allows for various degree of freedom ratios to be used in order to fulfil some of the requirements mentioned above. The particular choice of finite element spaces that are used also need to satisfy certain discrete vector calculus operators and identities in order to extend the mimetic properties of C-grid methods.

Out of the various compatible finite element space combinations mentioned in McRae and Cotter (2013), the one that this project is based on is given by the triple $(P_2 \oplus \mathcal{B}_3, \mathrm{BDFM}_1, P_1^{\mathrm{DG}})$. Here, $P_2 \oplus \mathcal{B}_3$ corresponds to a continuous, piecewise-quadratic function space, which is enriched by a cubic 'bubble' that is applied to each element. Out of the terms of the shallow water equations, vorticity is assigned to this space. Next, BDFM_1 corresponds to one of the Brezzi-Douglas-Fortin-Marini family of finite element spaces, that is piecewise-linear with continuous normal components between elements, and tangential (possibly discontinuous) degrees of freedom along element edges. It is to this that the fluid velocity is assigned. Finally, P_1^{DG} corresponds to a piecewise linear space with discontinuous behaviour between neighbouring elements.

This space corresponds to the pressure terms and the fluid depth h. It should be noted that in the two-dimensional case, these spaces are all applied to a discretisation involving triangular elements.

Out of these finite element spaces, this project concentrates on analysing the one corresponding to potential vorticity ($P_2 \oplus \mathcal{B}_3$). This is done by simplifying to a one-dimensional scenario and applying a Taylor-Galerkin scheme to the linear advection equation (the one-dimensional analogue of the two-dimensional potential vorticity advection equation). The scheme will use finite difference methods to discretise in time and finite element methods to discretise in space. The particular two-stage version that will be studied allows for third order accuracy in time and arbitrary accuracy in space. It will be during the spatial discretisation phase that a second or third order continuous polynomial function space will be chosen and applied to the scheme in order to analyse the one-dimensional equivalent of the $P_2 \oplus \mathcal{B}_3$ space.

The reason for choosing this scheme is to maintain high levels of accuracy in both space and time and to maintain numerical stability when approximating the advection of the potential vorticity. This is important in a more global sense as the potential vorticity forms an intermediate step when it comes to numerically approximating the main shallow-water equations. The time-stepping procedure involves first updating the fluid depth h, then diagnosing and updating the potential vorticity q, before updating the fluid velocity $\boldsymbol{u}$, and ensuring that it is consistent with q (which contains both $\boldsymbol{u}$ and h as terms). Consequently, any errors that occur while advecting q, can have a knock on effect that can compromise the overall approximation of the shallow-water equations.

2 Theory and Derivations

2.1 Problem Definition

In order to investigate the suitability of applying Taylor-Galerkin methods to complex transport problems, it is necessary to define a problem that is both simple to analyse and representative of the key characteristics that can be found in the governing equations used for numerical weather prediction. Such characteristics include the propagation of information in both space and time, as well as the presence of dispersion and diffusion errors that can affect the numerical approximation. One such problem is the hyperbolic linear-advection equation, which in its strong form is given by

$$\frac{\partial q}{\partial t} + c\,\frac{\partial q}{\partial x} \;=\; 0, \qquad 0 \leq x \leq 1, \tag{2.1}$$

which describes the one-dimensional transport of a quantity q at the advection velocity c. The quantity q is described by a function of both space and time, such that $q = q\,(x,t)$, which is subject to an initial condition given by $q\,(x,0) = f\,(x)$, and periodic boundary conditions given by $q\,(0,t) = q\,(1,t)$, applied to the boundaries of the domain $\Omega = [0,1]$. The equation takes the arbitrary initial profile described by $f\,(x)$ and translates it at the constant advection velocity c without changing its shape. Consequently, after a time t has elapsed, the solution q is the profile

$f(x)$, displaced by a distance $x = ct$. This leads to the exact solution

$$q(x,t) \;=\; f(x-ct)\,. \tag{2.2}$$

2.2 The Taylor-Galerkin Method

In the initial stages of its derivation, the Taylor-Galerkin method acts as the Finite Element analogue of the Finite Difference Lax-Wendroff method. The first step in employing the method is to take the model linear-advection equation (2.1) and to simultaneously rearrange and differentiate it in such a manner as to obtain time derivatives of q in terms of space derivatives. This allows for the problem to be first discretised in time using finite difference methods and then discretised in space using finite element methods (Safjan and Oden, 1995). The expressions for the first and second derivatives of q with respect to time are given below as

$$\frac{\partial q}{\partial t} \;=\; -c\,\frac{\partial q}{\partial x}\,, \tag{2.3}$$

and

$$\frac{\partial^2 q}{\partial t^2} \;=\; \frac{\partial}{\partial t}\left(-c\,\frac{\partial q}{\partial x}\right) \;=\; \frac{\partial}{\partial x}\left(-c\,\frac{\partial q}{\partial t}\right) \;=\; \frac{\partial}{\partial x}\left(-c\left[-c\,\frac{\partial q}{\partial x}\right]\right) \;=\; c^2\,\frac{\partial^2 q}{\partial x^2}\,. \tag{2.4}$$

Next, writing out the Taylor series expansion of q at time level $n+1$ about time level n, while switching to the more compact comma-subscript notation for derivatives results in the equation

$$q^{n+1} \;=\; q^n + \Delta t q^n_{,t} + \frac{\Delta t^2}{2} q^n_{,tt} + \frac{\Delta t^3}{6} q^n_{,ttt} + \mathcal{O}(\Delta t^4). \tag{2.5}$$

For the purpose of adding stability, the term $\eta \Delta t^2 q_{,tt}$ can be subtracted from both sides of (2.5), to give

$$q^{n+1} - \eta \Delta t^2 q^{n+1}_{,tt} \;=\; q^n + \Delta t q^n_{,t} + \Delta t^2 \left(\frac{1}{2} - \eta\right) q^n_{,tt} + \frac{\Delta t^3}{6} q^n_{,ttt} + \mathcal{O}(\Delta t^4), \tag{2.6}$$

where it is evaluated at time level $n+1$ on the left hand side and at time level n on the right hand side. The parameter $\eta \in \mathbb{R}_+$ is known as the stability parameter and is chosen in such a manner as to balance stability of the scheme with accuracy. The inclusion of this term now makes the method semi-implicit and forms the basis for the numerical scheme given by Safjan and Oden (1995).

The scheme given by Safjan and Oden (1995) is a multi-stage one where a typical time-step $t^n \Longrightarrow t^n + \Delta t$ is partitioned into several intermediate stages. This is done in such a manner that when a solution $q(t^n)$ is given, the next time-step solution $q(t^n + \Delta t)$ is obtained when the scheme reaches the final stage. Applying the scheme to the single unknown q in (2.6) now yields

$$q_i - \eta \Delta t^2 q_{i,tt} \;=\; q_0 + \mu_{i0} \Delta t q_{0,t} + \nu_{i0} \Delta t^2 q_{0,tt} + \Delta t \sum_{j=1}^{i-1} \mu_{ij} q_{j,t} + \Delta t^2 \sum_{j=1}^{i-1} \nu_{ij} q_{j,tt}, \qquad i = 1, 2, \ldots, s. \tag{2.7}$$

where i is the stage number with s as the final stage, and j is the stage number of each of the previous stages up to a particular $(i-1)^{\text{th}}$ stage. The coefficients $\mu_{ij}, \nu_{ij}, \mu_{i0}, \nu_{i0}$ are chosen in such a manner so as to maximise the desired order of accuracy of the scheme with respect to time, while also ensuring that it is stable for some choice of η. In order to accomplish this, they need to simultaneously satisfy so-called "order conditions" and "commutability constraints" as shown by Safjan and Oden (1995, page 210). It should be noted that while schemes (2.7) and (2.6) look dissimilar due to the apparent absence of η from the right hand side of (2.7), η does in fact appear in the expressions for μ_{ij} and ν_{ij} through the application of the order conditions and commutability constraints. This ensures that the scheme given by (2.7) is still semi-implicit.

Next, Summing from $j = 0$ instead of $j = 1$ for the sake of brevity, while substituting equations (2.3) and (2.4) into (2.7), allows for the scheme to be given in terms of spatial derivatives by the equation

$$q_i - \eta c^2 \Delta t^2 q_{i,xx} \;=\; q_0 - \Delta t \sum_{j=0}^{i-1} \mu_{ij} c q_{j,x} + \Delta t^2 \sum_{j=0}^{i-1} \nu_{ij} c^2 q_{j,xx}, \qquad i = 1, 2, \ldots, s, \tag{2.8}$$

which now allows for the problem to be discretised in space using finite element methods.

2.3 Weak Formulation

Normally, in order to define the weak form of a problem, at least two types of function spaces need to first be defined, namely a trial solution space, $\mathcal{S}$, and a weight function space, $\mathcal{V}$ (Hughes, 1987). However, due to the fact that our problem involves periodic boundary conditions, as opposed to explicitly defined Dirichlet conditions, only the trial solution space needs to be defined, within which both trial solutions q and weight functions w will exist. This definition is given by

$$\mathcal{S} = \{w, q \mid (w, q) \in H^1, q(0) = q(1), w(0) = w(1)\}, \tag{2.9}$$

where $(w, q) \in H^1$ means that the derivatives of both the trial solutions and the weight functions are required to be square-integrable over the domain. Mathematically, applied to the trial solutions for example, this would be given by

$$\int_0^1 \left(q_{,x}\right)^2 \, \mathrm{d}x < \infty. \tag{2.10}$$

Additionally, both trial solutions and weight functions must also satisfy the periodic boundary conditions given for equation (2.1).

The weak form of equation (2.8) can now be obtained by bringing all terms to the left hand side, multiplying them by a weight function w, and requiring the integral of this expression over

the domain to equal zero (2.11),

$$\int_0^1 w \left(q_i - \eta c^2 \Delta t^2 q_{i,xx} - q_0 + \Delta t \sum_{j=0}^{i-1} \mu_{ij} c q_{j,x} - \Delta t^2 \sum_{j=0}^{i-1} \nu_{ij} c^2 q_{j,xx} \right) \mathrm{d}x = 0, \qquad (2.11)$$
$$i = 1, 2, \ldots, s.$$

Subsequently, multiplication by w and integration can be carried out term by term, while taking constants and sums out of the integrals. Integration by parts can then be used to transfer differentiation to the weight function, where appropriate, and to simplify the expression through the application of the boundary conditions. This simplification occurs because both w and q are subject to periodic boundary conditions which means that the expression in the square brackets within the compact integration by parts equation, $\int_a^b uv' = [uv]_a^b - \int_a^b u'v$, disappears when it is evaluated due to the terms being equal at both ends of the domain. This gives the final weak form of (2.8) as

$$\int_0^1 w q_i \, \mathrm{d}x + \eta c^2 \Delta t^2 \int_0^1 w_{,x} q_{i,x} \, \mathrm{d}x = \int_0^1 w q_0 \, \mathrm{d}x + \sum_{j=0}^{i-1} \mu_{ij} c \Delta t \int_0^1 w_{,x} q_j \, \mathrm{d}x \qquad (2.12)$$
$$- \sum_{j=0}^{i-1} \nu_{ij} c^2 \Delta t^2 \int_0^1 w_{,x} q_{j,x} \, \mathrm{d}x, \qquad i = 1, 2, \ldots, s.$$

2.4 Galerkin Finite Element Approximation

The first step in constructing the Galerkin finite element approximation of (2.12) is to seek a finite-dimensional approximation of $\mathcal{S}$, denoted $\mathcal{S}^h$, such that $\mathcal{S}^h \subset \mathcal{S}$, where the superscript h corresponds to a discretisation of the domain Ω (Hughes, 1987). This means that if $q^h \in \mathcal{S}^h$, where q^h is the approximate solution to the weak form of the problem represented by a finite expansion, then $q^h \in \mathcal{S}$ as well. This also means that it is subject to the same boundary conditions as any other member of $\mathcal{S}$, namely $q^h(0) = q^h(1)$. Similarly, if $w^h \in \mathcal{S}$, where w^h is also represented by a finite expansion, this means that $w^h(0) = w^h(1)$.

The approximations of the trial solutions and weight functions can now be substituted into equation (2.12) to give the Galerkin approximation as

$$\int_0^1 w^h q_i^h \, \mathrm{d}x + \eta c^2 \Delta t^2 \int_0^1 w_{,x}^h q_{i,x}^h \, \mathrm{d}x = \int_0^1 w^h q_0^h \, \mathrm{d}x + \sum_{j=0}^{i-1} \mu_{ij} c \Delta t \int_0^1 w_{,x}^h q_j^h \, \mathrm{d}x \qquad (2.13)$$
$$- \sum_{j=0}^{i-1} \nu_{ij} c^2 \Delta t^2 \int_0^1 w_{,x}^h q_{j,x}^h \, \mathrm{d}x, \qquad i = 1, 2, \ldots, s.$$

2.5 Matrix Form of the Problem

Further structure can now be given to $\mathcal{S}^h$, by letting it consist of all linear combinations of given shape functions N_A, where the subscript $A = 1, 2, \ldots, n_n$ corresponds to the global node

number of a node within a discretised domain containing n_n nodes. According to Hughes (1987), this means that if $w^h \in \mathcal{S}^h$, then a set of constants c_A exist such that w^h is given by the basis expansion

$$w^h = \sum_{A=1}^{n_n} c_A N_A = c_1 N_1 + c_2 N_2 + \cdots + c_{n_n} N_{n_n}. \tag{2.14}$$

Noting the change from subscript to superscript notation for the stage number, an equivalent basis expansion can be defined for q_i^h as

$$q_i^h = \sum_{B=1}^{n_n} d_B^i N_B = d_1^i N_1 + d_2^i N_2 + \cdots + d_{n_n}^i N_{n_n}, \tag{2.15}$$

where the subscript B also corresponds to the global node number of each node.

Expansions (2.14) and (2.15) can then be substituted into equation (2.13) to give

$$\begin{aligned}
&\int_0^1 \left(\sum_{A=1}^{n_n} c_A N_A\right)\left(\sum_{B=1}^{n_n} d_B^i N_B\right)\,\mathrm{d}x \;+\; \eta\Delta t^2 c^2 \int_0^1 \left(\sum_{A=1}^{n_n} c_A N_{A,x}\right)\left(\sum_{B=1}^{n_n} d_B^i N_{B,x}\right)\,\mathrm{d}x \;=\\
&\int_0^1 \left(\sum_{A=1}^{n_n} c_A N_A\right)\left(\sum_{B=1}^{n_n} d_B^0 N_B\right)\,\mathrm{d}x \;+\; \sum_{j=0}^{i-1} \mu_{ij} c\Delta t \int_0^1 \left(\sum_{A=1}^{n_n} c_A N_{A,x}\right)\left(\sum_{B=1}^{n_n} d_B^j N_B\right)\,\mathrm{d}x\\
&-\; \sum_{j=0}^{i-1} \nu_{ij} c^2 \Delta t^2 \int_0^1 \left(\sum_{A=1}^{n_n} c_A N_{A,x}\right)\left(\sum_{B=1}^{n_n} d_B^j N_{B,x}\right)\,\mathrm{d}x, \qquad i = 1, 2, \ldots, s.
\end{aligned} \tag{2.16}$$

Invoking bilinearity and taking all terms to the left hand side, (2.16) can be simplified to

$$\begin{aligned}
\sum_{A=1}^{n_n} c_A \Bigg[&\int_0^1 (N_A)\left(\sum_{B=1}^{n_n} d_B^i N_B\right)\,\mathrm{d}x \;+\; \eta\Delta t^2 c^2 \int_0^1 (N_{A,x})\left(\sum_{B=1}^{n_n} d_B^i N_{B,x}\right)\,\mathrm{d}x\\
&-\; \int_0^1 (N_A)\left(\sum_{B=1}^{n_n} d_B^0 N_B\right)\,\mathrm{d}x \;-\; \sum_{j=0}^{i-1} \mu_{ij} c\Delta t \int_0^1 (N_{A,x})\left(\sum_{B=1}^{n_n} d_B^j N_B\right)\,\mathrm{d}x\\
&+\; \sum_{j=0}^{i-1} \nu_{ij} c^2 \Delta t^2 \int_0^1 (N_{A,x})\left(\sum_{B=1}^{n_n} d_B^j N_{B,x}\right)\,\mathrm{d}x \Bigg] \;=\; 0, \qquad i = 1, 2, \ldots, s,
\end{aligned} \tag{2.17}$$

where the expression in the square brackets can be denoted G_A, so that equation (2.17) can also be written as $\sum_{A=1}^{n_n} c_A G_A = 0$. Since all c_A's are arbitrary, and equation (2.17) is required to hold for all $w^h \in \mathcal{S}^h$, this means that all G_A's must be identically zero, therefore eliminating the term $\sum_{A=1}^{n_n} c_A$ from the equation above.

Swapping sums and integrals, while moving constant terms out of integrals and invoking

bilinearity once again, allows for equation (2.17) to be recast as

$$\begin{aligned}
&\sum_{B=1}^{n_n}\left(\int_0^1 N_A N_B \,\mathrm{d}x\right) d_B^i \;+\; \eta\Delta t^2 c^2 \sum_{B=1}^{n_n}\left(\int_0^1 N_{A,x} N_{B,x} \,\mathrm{d}x\right) d_B^i \;= \\
&\sum_{B=1}^{n_n}\left(\int_0^1 N_A N_B \,\mathrm{d}x\right) d_B^0 \;+\; \sum_{j=0}^{i-1} \mu_{ij} c\Delta t \sum_{B=1}^{n_n}\left(\int_0^1 N_{A,x} N_B \,\mathrm{d}x\right) d_B^j \\
&-\; \sum_{j=0}^{i-1} \nu_{ij} c^2\Delta t^2 \sum_{B=1}^{n_n}\left(\int_0^1 N_{A,x} N_{B,x} \,\mathrm{d}x\right) d_B^j, \qquad i = 1, 2, \ldots, s. \qquad (2.18)
\end{aligned}$$

This now allows us to define the following $n_n \times n_n$ dimensional matrices as

$$M \;:=\; \int_0^1 N_A N_B \,\mathrm{d}x, \qquad (2.19)$$

$$K \;:=\; \int_0^1 N_{A,x} N_{B,x} \,\mathrm{d}x, \qquad (2.20)$$

$$D \;:=\; \int_0^1 N_{A,x} N_B \,\mathrm{d}x, \qquad (2.21)$$

which, along with $\boldsymbol{d}^i$ representing an $n_n \times 1$ vector of the coefficients d_B^i at each stage, allows for the final matrix form of the problem to be given by

$$\begin{aligned}
M\boldsymbol{d}^i \;+\; \eta\Delta t^2 c^2\, K\boldsymbol{d}^i \;&=\; M\boldsymbol{d}^0 \;+\; \sum_{j=0}^{i-1} \mu_{ij} c\Delta t\; D\boldsymbol{d}^j \qquad (2.22) \\
&-\; \sum_{j=0}^{i-1} \nu_{ij} c^2\Delta t^2\; K\boldsymbol{d}^j, \qquad i = 1, 2, \ldots, s,
\end{aligned}$$

where the summation operation carried out by $\sum_{B=1}^{n_n}$ in (2.18) is now implied via matrix-vector multiplication. Equation (2.22) now represents the most general form of the Taylor-Galerkin approximation to the linear advection problem.

2.6 Two Stage Taylor Galerkin Formulation

Using equation (2.22), a two stage - third order accurate in time method can now be derived by setting $s = 2$ with $i = 1, 2$. This means that for stage one ($i = 1$), the scheme is given by

$$\left[M + \eta c^2\Delta t^2\; K\right]\boldsymbol{d}^1 \;=\; \left[M \;+\; \mu_{10} c\Delta t\; D \;-\; \nu_{10} c^2\Delta t^2\; K\right]\boldsymbol{d}^0, \qquad (2.23)$$

while for stage two ($i = 2$), it is given by

$$\begin{aligned}
\left[M + \eta c^2\Delta t^2 K\right]\boldsymbol{d}^2 \;&=\; \left[M \;+\; \mu_{20} c\Delta t\; D \;-\; \nu_{20} c^2\Delta t^2\; K\right]\boldsymbol{d}^0 \\
&+\; \left[M \;+\; \mu_{21} c\Delta t\; D \;-\; \nu_{21} c^2\Delta t^2\; K\right]\boldsymbol{d}^1, \qquad (2.24)
\end{aligned}$$

where the vector $\boldsymbol{d}^1$ is obtained first by solving (2.23) to give

$$\boldsymbol{d}^1 \;=\; \Big[M + \eta c^2 \Delta t^2 \; K\Big]^{-1} \Big[M \;+\; \mu_{10} c \Delta t \; D \;-\; \nu_{10} c^2 \Delta t^2 \; K\Big] \boldsymbol{d}^0. \tag{2.25}$$

This can then be substituted into (2.24) to give the final two-stage scheme as

$$\begin{aligned} \Big[M + \eta c^2 \Delta t^2 \; K\Big] \boldsymbol{d}^2 \;=\; \Bigg(\Big[M + \mu_{20} c \Delta t \; D - \nu_{20} c^2 \Delta t^2 \; K\Big] \;+ \\ \Big[M + \mu_{21} c \Delta t \; D - \nu_{21} c^2 \Delta t^2 \; K\Big] \Big[M + \eta c^2 \Delta t^2 K\Big]^{-1} \Big[M + \mu_{10} c \Delta t \; D - \nu_{10} c^2 \Delta t^2 \; K\Big] \Bigg) \boldsymbol{d}^0, \end{aligned} \tag{2.26}$$

which can be solved to give the solution at t^{n+1}, when $i = 2$, with respect to the solution at t^n, when $j = 0$. Conveniently, for this particular scheme, expressions for the coefficients μ_{ij} and ν_{ij}, required to guarantee stability for some choice of η and third order accuracy, are explicitly given by Safjan and Oden (1995, page 210).

2.7 Numerical Attributes of the Scheme

Now that the scheme has been formally introduced, and its two-stage version defined, key attributes of the scheme can be expounded on in order to justify its use from a computational perspective.

Firstly, it should be noted that in general, if a scheme contains derivatives that are to be included in any matrix formulations, as they are when constructing the finite element formulation of this scheme, those matrices will tend to be ill-conditioned. Numerically, this has negative implications when using iterative methods to invert these matrices during the solution process as the presence of any numerical errors can quickly be amplified. One of the strong points of the scheme given by (2.8) is that for every spatial derivative, there is a factor of Δt or Δt^2 that balances out the negative effects of these derivatives on the condition number of the corresponding matrices. It should also be noted that this statement is only true when the Courant number, given by $\sigma = c\frac{\Delta t}{\Delta x}$, which can be formed by collecting the relevant terms in the scheme (2.22), is held constant.

Another strong point of the formulation of this scheme is that while the coefficients μ_{ij} and ν_{ij} depend on the stage number i, η, which appears on both sides of the equation in (2.8), is independent of i. This has significant advantages in terms of computational efficiency as it means that η does not need to be recomputed at every stage in order to maintain equality between the left and right hand sides of the equation. Importantly, it enables stability and accuracy control without having to sacrifice efficiency.

Focusing on the final form of the two-stage scheme (2.26), it is also evident that the recycling of the left hand side matrix terms results in significant advantages with regards to efficiency. These matrix terms only need to be computed and stored once during the solution process and can be reused for all subsequent intermediate stages when time-stepping.

As a result, the scheme allows for the formulation of schemes that are both high-order in time and in space, while allowing for stability and accuracy control without a significant sacrifice

of computational efficiency through its multistage structure.

2.8 Higher-Order Polynomial Shape Functions

Up until this stage, the form of weight, shape and trial solution functions has been completely arbitrary. We now choose to construct higher-order polynomial shape functions through the use of Lagrange polynomials. The general form of Lagrange polynomials for any individual element is given by

$$N_\alpha \;=\; l_\alpha^{(n_{en}-1)}(\xi) \;=\; \frac{\prod\limits_{\substack{\beta=1\\ \beta\neq\alpha}}^{n_{en}} (\xi - \xi_\beta)}{\prod\limits_{\substack{\beta=1\\ \beta\neq\alpha}}^{n_{en}} (\xi_\alpha - \xi_\beta)}, \tag{2.27}$$

as defined by Hughes (1987), where α and β are local node numbers within an individual element that has n_{en} nodes, l is a Lagrange polynomial of order $(n_{en}-1)$ and ξ denotes the local one dimensional coordinate within each element. The variable ξ is chosen to vary as $0 \leq \xi \leq 1$ between the edges of any individual element. It should be noted that $l_\alpha(\xi_\alpha) = 1$, while if $\beta \neq \alpha$, then $l_\alpha(\xi_\beta) = 0$. This essentially means that each shape function must take a value of one at its corresponding elemental node and zero at any of the other nodes within an element.

It is clear that if we desire shape functions of polynomial order m, such that $l_\alpha^{(n_{en}-1)} \in \mathbf{P}_m$, then we require that $n_{en} = m+1$ nodes per element. Consequently, for quadratic polynomials, elements will require three nodes, while for cubic polynomials, they will require four.

It should be noted that this form of shape function formulation results in a local element-wise description of the shape functions rather than a global one. This results in the formation of local versions of the M, K and D matrices that need to be assembled at a later stage in order to form the global ones through an assembly operation.

2.8.1 Quadratic Shape Function Formulation

Setting $n_{en} = 3$ with nodal coordinates being given by $\xi_1 = 0$, $\xi_2 = \frac{1}{2}$, $\xi_3 = 1$, and applying equation (2.27), leads to the quadratic shape functions

$$N_1(\xi) \;=\; \frac{(\xi-\xi_2)(\xi-\xi_3)}{(\xi_1-\xi_2)(\xi_1-\xi_3)} \;=\; \frac{\left(\xi-\frac{1}{2}\right)(\xi-1)}{\left(0-\frac{1}{2}\right)(0-1)} \;=\; (2\xi-1)(\xi-1), \tag{2.28}$$

$$N_2(\xi) \;=\; \frac{(\xi-\xi_1)(\xi-\xi_3)}{(\xi_2-\xi_1)(\xi_2-\xi_3)} \;=\; \frac{(\xi-0)(\xi-1)}{\left(\frac{1}{2}-0\right)\left(\frac{1}{2}-1\right)} \;=\; 4\xi(1-\xi), \tag{2.29}$$

$$N_3(\xi) \;=\; \frac{(\xi-\xi_1)(\xi-\xi_2)}{(\xi_3-\xi_1)(\xi_3-\xi_2)} \;=\; \frac{(\xi-0)\left(\xi-\frac{1}{2}\right)}{(1-0)\left(1-\frac{1}{2}\right)} \;=\; \xi(2\xi-1), \tag{2.30}$$

which can be plotted against ξ to give figure 2.1 below.

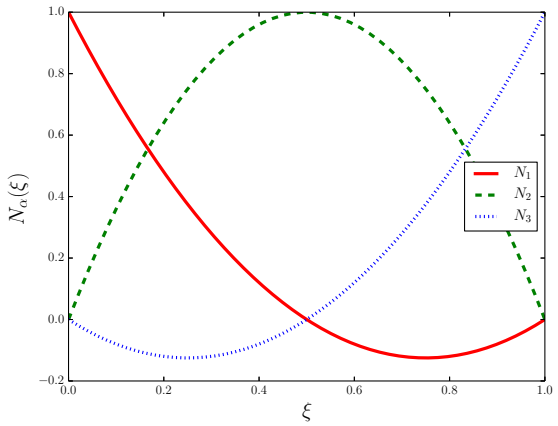


Figure 2.1 – Quadratic shape functions plotted against ξ

2.8.2 Cubic Shape Function Formulation

Similarly, setting $n_{en} = 4$ with nodal coordinates being given by $\xi_1 = 0$, $\xi_2 = \frac{1}{3}$, $\xi_3 = \frac{2}{3}$, $\xi_4 = 1$, and applying equation (2.27) leads to the the cubic shape functions

$$N_1(\xi) = \frac{(\xi-\xi_2)(\xi-\xi_3)(\xi-\xi_4)}{(\xi_1-\xi_2)(\xi_1-\xi_3)(\xi_1-\xi_4)} = \frac{\left(\xi-\frac{1}{3}\right)\left(\xi-\frac{2}{3}\right)(\xi-1)}{\left(0-\frac{1}{3}\right)\left(0-\frac{2}{3}\right)(0-1)} = \frac{\left(-9\xi^3+18\xi^2-11\xi+2\right)}{2}, \tag{2.31}$$

$$N_2(\xi) = \frac{(\xi-\xi_1)(\xi-\xi_3)(\xi-\xi_4)}{(\xi_2-\xi_1)(\xi_2-\xi_3)(\xi_2-\xi_4)} = \frac{(\xi-0)\left(\xi-\frac{2}{3}\right)(\xi-1)}{\left(\frac{1}{3}-0\right)\left(\frac{1}{3}-\frac{2}{3}\right)\left(\frac{1}{3}-1\right)} = \frac{9\left(3\xi^3-5\xi^2+2\xi\right)}{2}, \tag{2.32}$$

$$N_3(\xi) = \frac{(\xi-\xi_1)(\xi-\xi_2)(\xi-\xi_4)}{(\xi_3-\xi_1)(\xi_3-\xi_2)(\xi_3-\xi_4)} = \frac{(\xi-0)\left(\xi-\frac{1}{3}\right)(\xi-1)}{\left(\frac{2}{3}-0\right)\left(\frac{2}{3}-\frac{1}{3}\right)\left(\frac{2}{3}-1\right)} = \frac{-9\left(3\xi^3-4\xi^2+\xi\right)}{2}, \tag{2.33}$$

$$N_4(\xi) = \frac{(\xi-\xi_1)(\xi-\xi_2)(\xi-\xi_3)}{(\xi_4-\xi_1)(\xi_4-\xi_2)(\xi_4-\xi_3)} = \frac{(\xi-0)\left(\xi-\frac{1}{3}\right)\left(\xi-\frac{2}{3}\right)}{(1-0)\left(1-\frac{1}{3}\right)\left(1-\frac{2}{3}\right)} = \frac{\left(9\xi^3-9\xi^2+2\xi\right)}{2}, \tag{2.34}$$

which can be plotted against ξ to give figure 2.2 below.

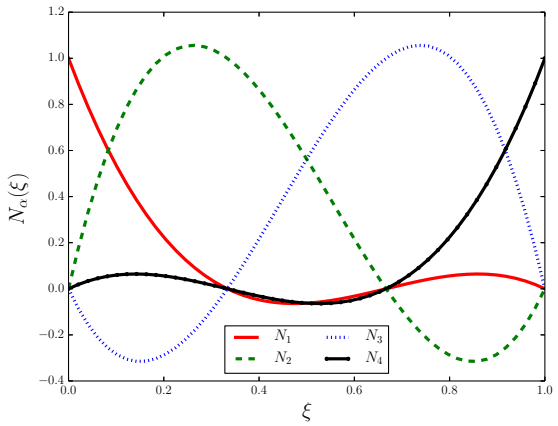


Figure 2.2 – Cubic shape functions plotted against ξ

2.9 Stability Analysis

Much of the work presented up to this point involves the application of standard methods and knowledge that have already been well established in the literature. In particular, this applies to the derivation of the time-stepping portion of the Taylor-Galerkin scheme, as well as its spatial discretisation through the application of the finite element method. The formulation of the stability analysis and its numerical implementation however, constitutes the start of original work that has been carried out as part of the execution of this project.

2.9.1 Preliminaries

Guaranteeing stability is an important part of any scheme that deals with linear PDEs, as it forms part of the Lax-equivalence theorem, which states that if a numerical scheme is both stable and consistent, it will also be convergent (LeVeque, 2007). Consistency is conveniently guaranteed by using the finite element method, as it is effectively incorporated into the weak formulation of the problem, when we require the integral of the weighted residual over the domain to equal zero, as stated in equation (2.11).

A Fourier analysis approach to investigating stability will now be used to evaluate the restrictions required for keeping the two-stage Taylor-Galerkin scheme stable. This will employ the von Neumann method of analysis and will also be used to evaluate the diffusion and phase properties of the scheme by comparing errors between stable numerical solutions and the exact analytical solution.

In order to carry out the von Neumann analysis, the following requirements need to be met:

1. The analysis is limited to constant coefficient linear partial differential equations (PDEs), of which the linear advection equation (2.1) is one.

2. The analysis is applied to either the Cauchy problem, where the PDE is solved on all space, $-\infty < x < \infty$, with no boundary conditions, or it is applied to a problem on a

finite space, with periodic boundary conditions LeVeque (2007), such as that defined by equation (2.1).

3. The domain $0 \leq x \leq 1$, is discretised into N equispaced elements, each of width $\Delta x = h$, with $N + 1$ grid points, $x_1, x_2, \ldots, x_N, x_{N+1}$, as shown in figure 2.3 below.

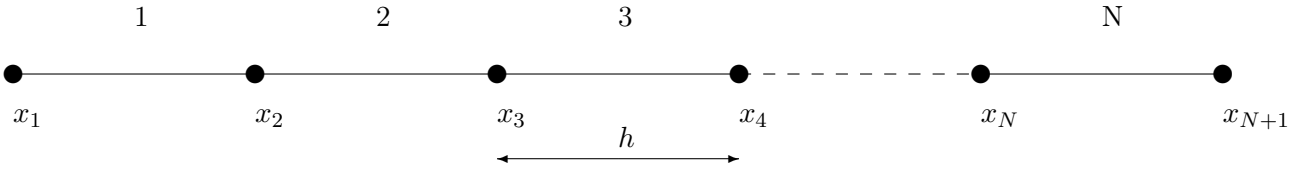


Figure 2.3 – 1D Finite Element Grid

4. The assumption is made that integration over the entire domain can be equivalently carried out by integrating over each elemental domain and summing over all elements.

5. The assumption is made that due to the periodic nature of the problem, discrete trial solutions q^h, within elements positioned a distance ah from element 1 in figure 2.3, can be given by the phase-shift

$$q^h\left(x + ah\right) = q^h\left(x\right) e^{Ia\phi}, \qquad a = 0, 1, \ldots, N-1, \tag{2.35}$$

where a is the shift index, $I = \sqrt{-1}$ is the imaginary unit (not to be confused with i - the Taylor-Galerkin stage number) and ϕ is the phase angle which varies as $-\pi \leq \phi \leq \pi$. Element 1 can then be referred to as the reference element, $\hat{e}$, whose elemental domain is given by $\Omega\hat{e} = [x_1, x_2]$.

6. Finally, the form of both the trial solutions and weight functions needs to be chosen. They are chosen to be part of a polynomial finite element space of order m, such that $q^h, w^h \in \mathbf{P}_m$.

2.9.2 Change of Variables & Elemental Boundary Conditions

The stability analysis begins by applying an extension to the Galerkin approximation (2.13), found in section 2.4. Focusing on the first term on the left hand side of the equation and applying requirement 4 above, gives

$$\int_0^1 w^h q_i^h \, \mathrm{d}x \;=\; \sum_{e=1}^{N} \int_{\Omega e} w^h q_i^h \, \mathrm{d}x, \tag{2.36}$$

where e is the element index number and Ωe represents the elemental domain of each element.

Next, employing a change of variables to shift all of the elemental solutions to the reference

element, $\hat{e}$, and then substituting (2.35) gives

$$\sum_{e=1}^{N}\int_{\Omega e} w^h q_i^h \,\mathrm{d}x \;=\; \sum_{a=0}^{N-1}\int_{\Omega\hat{e}} w^h\left(x+ah\right) q_i^h\left(x+ah\right)\,\mathrm{d}x \;=\; \sum_{a=0}^{N-1}\int_{\Omega\hat{e}} w^h\left(x+ah\right) q_i^h\left(x\right) e^{Ia\phi}\,\mathrm{d}x. \tag{2.37}$$

Taking the sum inside the integral now allows us to define

$$\hat{w}^h\left(x,\phi\right) \;=\; \sum_{a=0}^{N-1} w^h\left(x+ah\right) e^{Ia\phi}, \tag{2.38}$$

which can be substituted into (2.37) to give

$$\sum_{e=1}^{N}\int_{\Omega e} w^h q_i^h \,\mathrm{d}x \;=\; \int_{\Omega\hat{e}} \hat{w}^h q_i^h\left(x\right)\,\mathrm{d}x. \tag{2.39}$$

Next, the boundary conditions on q^h and $\hat{w}^h$ need to be established. By letting $a = 1$, the boundary conditions for q^h within $\hat{e}$ can be simply established as

$$q_i^h\Big|_{x_2} \;=\; q_i^h e^{I\phi}\Big|_{x_1}. \tag{2.40}$$

Similarly, shifting equation (2.38) by one element, where the shift is applied in a periodic manner to all elements in the domain, leads to

$$\hat{w}^h\left(x+h,\phi\right) \;=\; \sum_{a=0}^{N-1} w^h\left(x+h+ah\right) e^{Ia\phi} \;=\; \sum_{a=0}^{N-1} w^h\left(x+\left(a+1\right)h\right) e^{Ia\phi}. \tag{2.41}$$

Setting $a+1=a'$, allows (2.41) to be recast as

$$\hat{w}^h\left(x+h,\phi\right) \;=\; \sum_{a'=1}^{N} w^h\left(x+a'h\right) e^{I\left(a'-1\right)\phi} \;=\; \left[\sum_{a'=1}^{N} w^h\left(x+a'h\right) e^{Ia'\phi}\right] e^{-I\phi}. \tag{2.42}$$

Since the summation is carried out over the same number of elements and over a periodic domain, we can reset the dummy variable a' back to the initial variable a. This operation is equivalent to saying for example, that summing over the elements in the vector $[0,1,2,3]$ is the same as summing over the elements in the vector $[3,0,1,2]$. Since the form of the expression in square brackets within equation (2.42) is the same as that of (2.38), this now allows for (2.42) to be more concisely given by

$$\hat{w}^h\left(x+h,\phi\right) \;=\; \hat{w}^h\left(x\right) e^{-I\phi}, \tag{2.43}$$

which can be directly used to give the boundary conditions for $\hat{w}^h$ within element $\hat{e}$ as

$$\hat{w}^h\Big|_{x_2} \;=\; \hat{w}^h e^{-I\phi}\Big|_{x_1}. \tag{2.44}$$

2.9.3 Stability Analysis Matrix Formulation

Following the steps taken in section 2.5, local basis expansions of $\hat{w}^h$ and q_i^h, given by

$$\hat{w}^h = \sum_{\alpha=1}^{n_{en}} \hat{c}_\alpha N_\alpha, \tag{2.45}$$

$$q_i^h = \sum_{\beta=1}^{n_{en}} d_\beta^i N_\beta, \tag{2.46}$$

where α and β correspond to the local node numbers of an n_{en}-noded element, can now be substituted into equation (2.39) to give

$$\int_{\Omega\hat{e}} \hat{w}^h q_i^h \, \mathrm{d}x = \int_{\Omega\hat{e}} \sum_{\alpha=1}^{n_{en}} \hat{c}_\alpha N_\alpha \sum_{\beta=1}^{n_{en}} d_\beta^i N_\beta \, \mathrm{d}x = \sum_{\alpha=1}^{n_{en}} \hat{c}_\alpha \sum_{\beta=1}^{n_{en}} \left(\int_{\Omega\hat{e}} N_\alpha N_\beta \, \mathrm{d}x \right) d_\beta^i. \tag{2.47}$$

The integral $\int_{\Omega\hat{e}} N_\alpha N_\beta \, \mathrm{d}x$ now represents the elemental equivalent of matrix M (2.19), evaluated with respect to global coordinates, while being applied to the reference element $\hat{e}$. It allows for the definition of the $n_{en} \times n_{en}$ dimensional matrix

$$M^e := \int_{\Omega e} N_\alpha N_\beta \, \mathrm{d}x = \int_0^1 N_\alpha N_\beta \, x_{,\xi} \, \mathrm{d}\xi, \tag{2.48}$$

where the superscript e denotes the elemental equivalent for the e^{th} element and x corresponds to global coordinates given by $x(\xi) = \sum_{\alpha=1}^{n_{en}} N_\alpha(\xi) x_\alpha^e$, which can be differentiated with respect to local coordinates ξ to give $x_{,\xi} = \frac{\partial x}{\partial \xi} = \sum_{\alpha=1}^{n_{en}} \frac{\partial N_\alpha}{\partial \xi} x_\alpha^e$. The second equality in (2.48) corresponds to using a change of variables to evaluate the integral with respect to local coordinates, where the definite integral is evaluated between the boundaries of the element, given by the coordinates $\xi_1 = 0$ and $\xi_2 = 1$.

Differentiating $\hat{w}^h$ and q^h with respect to x, while employing a change of variables and applying the chain rule for evaluating derivatives with respect to local coordinates, also allows for the elemental equivalents of matrices K (2.20) and D (2.21) to be given by

$$K^e := \int_{\Omega e} N_{\alpha,x} N_{\beta,x} \, \mathrm{d}x = \int_0^1 N_{\alpha,x} N_{\beta,x} \, x_{,\xi} \, \mathrm{d}\xi = \int_0^1 \left(N_{\alpha,\xi} \, \xi_{,x} \right) \left(N_{\beta,\xi} \, \xi_{,x} \right) \, x_{,\xi} \, \mathrm{d}\xi, \tag{2.49}$$

$$D^e := \int_{\Omega e} N_{\alpha,x} N_\beta \, \mathrm{d}x = \int_0^1 N_{\alpha,x} N_\beta \, x_{,\xi} \, \mathrm{d}\xi = \int_0^1 \left(N_{\alpha,\xi} \, \xi_{,x} \right) N_\beta \, x_{,\xi} \, \mathrm{d}\xi. \tag{2.50}$$

By evaluating the integrals of the shape functions and their derivatives with respect to the elemental coordinate system, for every instance of the term $x_{,\xi}$, introduced as part of the change of variables step, a factor of $h = \Delta x$ drops out of each term. It should be noted however, that since this occurs within each of the integrals equally, it can be eliminated from both sides of the matrix equation (2.51) below. Similarly, for every instance of the term $\xi_{,x}$, used as part of the chain rule during differentiation, a factor of $\frac{1}{\Delta x}$ drops out of each of the respective terms.

Next, the steps that were carried out between equations (2.36) and (2.39) can now be applied to the remaining integral terms within the Galerkin approximation (2.13). The above elemental matrix definitions, along with basis expansions (2.45) and (2.46) can then be substituted to give the transformed matrix form of the problem as

$$\begin{aligned} \hat{\boldsymbol{c}}^{\mathrm{T}} M^{\hat{e}} \boldsymbol{d}^{i} \; + \; \eta\sigma^{2} \; \hat{\boldsymbol{c}}^{\mathrm{T}} K^{\hat{e}} \boldsymbol{d}^{i} \; &= \; \hat{\boldsymbol{c}}^{\mathrm{T}} M^{\hat{e}} \boldsymbol{d}^{0} \; + \; \sum_{j=0}^{i-1} \mu_{ij}\sigma \; \hat{\boldsymbol{c}}^{\mathrm{T}} D^{\hat{e}} \boldsymbol{d}^{j} \\ &- \; \sum_{j=0}^{i-1} \nu_{ij}\sigma^{2} \; \hat{\boldsymbol{c}}^{\mathrm{T}} K^{\hat{e}} \boldsymbol{d}^{j}, \qquad i = 1, 2, \ldots, s, \end{aligned} \tag{2.51}$$

where the dimensionless Courant number is now introduced as $\sigma = \frac{c\Delta t}{\Delta x}$, $\hat{\boldsymbol{c}}$ is an $n_{en} \times 1$ dimensional vector of the coefficients $\hat{c}_A$, with $\hat{\boldsymbol{c}}^{\mathrm{T}}$ as its transpose, and the superscript $\hat{e}$ attached to each of the matrices indicates that the problem is now set within the domain of the reference element $\hat{e}$.

2.9.4 The Reduced Problem

At this stage, the problem stated by equation (2.51) is an n_{en} dimensional one. That is to say it is composed of $n_{en} \times n_{en}$ dimensional matrices and $n_{en} \times 1$ dimensional vectors. As mentioned in section 2.8, if polynomial shape functions of order m are chosen, then $n_{en} = m + 1$ nodes per element are required.

The problem can be reduced to an m dimensional one by introducing the matrix S, an $n_{en} \times m$ dimensional matrix that contains the boundary condition information for q_i^h, given in equation (2.40). The matrix is structured in such a way that its upper square portion is composed of the elements of the $m \times m$ identity matrix, while its ${n_{en}}^{\text{th}}$ row is composed of $e^{I\phi}$ as its first element, followed by zeros for the rest, as shown by

$$S \; = \; \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 1 \\ e^{I\phi} & 0 & \cdots & 0 \end{pmatrix}. \tag{2.52}$$

Matrix S can now be used to give the $n_{en} \times 1$ vector $\boldsymbol{d}$ in terms of the reduced $m \times 1$ vector $\tilde{\boldsymbol{d}}$ by the operation $\boldsymbol{d} = S\tilde{\boldsymbol{d}}$.

We can now take advantage of the fact that the boundary condition for $\hat{w}^h$ (2.44) is the complex conjugate of that for q_i^h (2.40) by simply taking the complex conjugate of S. This allows us to express $\hat{\boldsymbol{c}}$ in terms of the reduced vector $\tilde{\boldsymbol{c}}$ by the operation $\hat{\boldsymbol{c}} = S^{*}\tilde{\boldsymbol{c}}$, where S^{*} is the complex conjugate of S. Consequently, $\hat{\boldsymbol{c}}^{\mathrm{T}}$ can now be given by $\hat{\boldsymbol{c}}^{\mathrm{T}} = \left(S^{*}\tilde{\boldsymbol{c}}\right)^{\mathrm{T}}$. Applying the general rule $(PQ)^{\mathrm{T}} = Q^{\mathrm{T}}P^{\mathrm{T}}$ leads to $\hat{\boldsymbol{c}}^{\mathrm{T}} = \tilde{\boldsymbol{c}}^{\mathrm{T}}\left(S^{*\mathrm{T}}\right) = \tilde{\boldsymbol{c}}^{\mathrm{T}}S^{\dagger}$, where $S^{\dagger}$ is the Hermitian transpose of S.

This can all be substituted into (2.51) to give the reduced form of the problem as

$$\tilde{\boldsymbol{c}}^{\mathrm{T}} S^{\dagger} \left[M^{\hat{e}} \; + \; \eta\sigma^2 \; K^{\hat{e}} \right] S\tilde{\boldsymbol{d}}^{i} \; = \; \tilde{\boldsymbol{c}}^{\mathrm{T}} S^{\dagger} M^{\hat{e}} S\tilde{\boldsymbol{d}}^{0} \; + \; \sum_{j=0}^{i-1} \mu_{ij}\sigma \; \tilde{\boldsymbol{c}}^{\mathrm{T}} S^{\dagger} D^{\hat{e}} S\tilde{\boldsymbol{d}}^{j} \qquad (2.53)$$
$$- \; \sum_{j=0}^{i-1} \nu_{ij}\sigma^2 \; \tilde{\boldsymbol{c}}^{\mathrm{T}} S^{\dagger} K^{\hat{e}} S\tilde{\boldsymbol{d}}^{j}, \qquad i = 1, 2, \ldots, s.$$

Since $\tilde{\boldsymbol{c}}^{\mathrm{T}}$ is arbitrary, it can be removed from both sides of the equation to give the final reduced form of the problem as

$$S^{\dagger} \left[M^{\hat{e}} \; + \; \eta\sigma^2 \; K^{\hat{e}} \right] S\tilde{\boldsymbol{d}}^{i} \; = \; S^{\dagger} M^{\hat{e}} S\tilde{\boldsymbol{d}}^{0} \; + \; \sum_{j=0}^{i-1} \mu_{ij}\sigma \; S^{\dagger} D^{\hat{e}} S\tilde{\boldsymbol{d}}^{j} \qquad (2.54)$$
$$- \; \sum_{j=0}^{i-1} \nu_{ij}\sigma^2 \; S^{\dagger} K^{\hat{e}} S\tilde{\boldsymbol{d}}^{j}, \qquad i = 1, 2, \ldots, s.$$

2.9.5 The Reduced Two Stage Problem

Equation (2.54) represents the most general form of the reduced problem. The analysis can now be made more specific by introducing the two-stage scheme derived in section 2.6. Using the final version of the two-stage scheme given in equation (2.26) as reference, the reduced two-stage form of the problem can be given by

$$S^{\dagger} \left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}} \right] S\tilde{\boldsymbol{d}}^{2} \; = \; S^{\dagger} \Bigg(\left[M^{\hat{e}} + \mu_{20}\sigma D^{\hat{e}} - \nu_{20}\sigma^2 K^{\hat{e}} \right] \; + \qquad (2.55)$$
$$\left[M^{\hat{e}} + \mu_{21}\sigma D^{\hat{e}} - \nu_{21}\sigma^2 K^{\hat{e}} \right] \left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}} \right]^{-1} \left[M^{\hat{e}} + \mu_{10}\sigma D^{\hat{e}} - \nu_{10}\sigma^2 K^{\hat{e}} \right] \Bigg) S\tilde{\boldsymbol{d}}^{0},$$

where $\tilde{\boldsymbol{d}}^{2}$ represents the reduced nodal solution vector at time level t^{n+1} and $\tilde{\boldsymbol{d}}^{0}$ represents the solution vector at time level t^{n}.

The definition of the amplification factor G can now be used to give the solution at t^{n+1} in terms of the solution at t^{n} through the expression $\tilde{\boldsymbol{d}}^{2} = G\tilde{\boldsymbol{d}}^{0}$. Consequently, the left hand side of equation (2.55) can now be written as

$$S^{\dagger} \left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}} \right] S\tilde{\boldsymbol{d}}^{2} \; = \; S^{\dagger} \left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}} \right] S \; G\tilde{\boldsymbol{d}}^{0}, \qquad (2.56)$$

which can be substituted back into (2.55) to give

$$S^{\dagger} \left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}} \right] S \; G\tilde{\boldsymbol{d}}^{0} \; = \; S^{\dagger} \Bigg(\left[M^{\hat{e}} + \mu_{20}\sigma D^{\hat{e}} - \nu_{20}\sigma^2 K^{\hat{e}} \right] \; + \qquad (2.57)$$
$$\left[M^{\hat{e}} + \mu_{21}\sigma D^{\hat{e}} - \nu_{21}\sigma^2 K^{\hat{e}} \right] \left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}} \right]^{-1} \left[M^{\hat{e}} + \mu_{10}\sigma D^{\hat{e}} - \nu_{10}\sigma^2 K^{\hat{e}} \right] \Bigg) S\tilde{\boldsymbol{d}}^{0},$$

which is a generalised eigenvalue problem of the form $P\lambda\,\boldsymbol{x} = Q\,\boldsymbol{x}$, where P corresponds to the left hand side matrix terms, λ corresponds to the amplification factor G, Q corresponds to the

right hand side matrix terms, and $\boldsymbol{x}$ corresponds to the vector of nodal solutions $\tilde{\boldsymbol{d}}^0$.

Taking all terms in (2.57) to the left hand side and equating them to zero then gives

$$\left[S^\dagger \left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}}\right] S\, G \;-\; S^\dagger \left(\left[M^{\hat{e}} + \mu_{20}\sigma D^{\hat{e}} - \nu_{20}\sigma^2 K^{\hat{e}}\right] + \left[M^{\hat{e}} + \mu_{21}\sigma D^{\hat{e}} - \nu_{21}\sigma^2 K^{\hat{e}}\right]\left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}}\right]^{-1}\left[M^{\hat{e}} + \mu_{10}\sigma D^{\hat{e}} - \nu_{10}\sigma^2 K^{\hat{e}}\right]\right) S\right] \tilde{\boldsymbol{d}}^0 = 0. \tag{2.58}$$

In order for this to be true

$$\left|S^\dagger \left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}}\right] S\, G \;-\; S^\dagger \left(\left[M^{\hat{e}} + \mu_{20}\sigma D^{\hat{e}} - \nu_{20}\sigma^2 K^{\hat{e}}\right] + \left[M^{\hat{e}} + \mu_{21}\sigma D^{\hat{e}} - \nu_{21}\sigma^2 K^{\hat{e}}\right]\left[M^{\hat{e}} + \eta\sigma^2 K^{\hat{e}}\right]^{-1}\left[M^{\hat{e}} + \mu_{10}\sigma D^{\hat{e}} - \nu_{10}\sigma^2 K^{\hat{e}}\right]\right) S\right| = 0, \tag{2.59}$$

or $\det\left(P\lambda - Q\right) = 0$, must also be true. Equation (2.59) now corresponds to the characteristic polynomial of the eigenvalue problem, whose roots need to be solved for in order to determine the eigenvalues given by G. Using this reduced formulation, a problem involving shape functions of polynomial order m will contain m roots that will need to be analysed. In general the roots will be complex, and both their magnitudes and arguments will need to be numerically analysed in order to shed light on the stability characteristics of the scheme. G will now be a function of the variable ϕ - the phase angle, and will also depend on the values of the parameters σ - the Courant number and η - the stability parameter. It is the choice of these parameter values and their influence on the stability and accuracy characteristics of the scheme, that will be studied during the numerical implementation phase.

3 Numerical Implementation

The numerical implementation of both the Taylor-Galerkin scheme and the stability analysis was carried out by writing programs using the Python programming language. Various libraries were utilised in order to add flexibility to the codes and to help with writing them in a compact fashion. Of the freely available libraries, the following were used: The SymPy library for symbolic mathematics (SymPy Development Team, 2008), the Numpy library for scientific computing (Oliphant, 2007), the matplotlib library for plotting data (Hunter, 2007), and the DOLFIN library for interfacing with applications from the FEniCS project in order to solve finite element problems (Logg et al., 2012).

Numerical implementation involved three distinct phases. Firstly, a base program was written to implement the stability analysis in order to investigate the effects of different choices of η and σ on the stability and accuracy characteristics of the scheme when applied to shape functions of order P_2. Following the first round of numerical investigations, a global critical value of $\eta \approx 0.47275$, referred to from now on as η_G, was found to mark the demarcation between the conditional stability region of the scheme and unconditional stability region. The term 'global'

here refers to η_G applying to all values of the Courant number. Values of $\eta \geq 0.47275$ were found to guarantee stability for any choice of the Courant number, while values of $\eta < 0.47275$ resulted in the scheme being stable for only a certain range of Courant number values. It was found that this numerical value of η_G agreed well with the exact value of $\eta_{\tilde{G}} = 0.4727411766$ (where the tilde symbol marks it as the exact value) given by Safjan and Oden (1995, page 230) for the two-stage scheme, thereby acting as a check on both the stability analysis derivation and the numerical implementation.

Since this range corresponded to the typically small values of σ found when conducting numerical weather prediction, the second phase of numerical investigation was initiated to find how the local critical value of η, denoted η_L, varies as a function of σ. Here the term 'local' corresponds to the critical value of η acting as the demarcation between the stable and unstable regions of the scheme, for a single corresponding value of σ. The reason for doing this was to see if lower values of η could be used to provide stability in situations where it could be guaranteed that the largest Courant number would remain relatively small. This would potentially provide accuracy gains, as lower levels of implicitness in a scheme tend to result in higher levels of accuracy by reducing the amount of numerical damping. This phase was implemented by modifying the base code and writing a binary-search algorithm that would search for η_L while looping through a range of σ.

The third and final phase of numerical implementation involved using DOLFIN and the FEniCS project to solve a finite element approximation of the linear advection equation using the Taylor-Galerkin scheme. This was done in part to validate the derivations of the scheme, but also to act as a test bed for verifying the results of the numerical stability analysis.

3.1 Stability Analysis Implementation

A brief description of the development and structure of the program that was written to implement the stability analysis now follows. The development process involved the addition of several mathematical steps in order to make sense of the numerical results, as well as the generalisation of the code so that it could be run for any combination of η and σ.

Inputs: The user defined inputs to the program are one dimensional arrays consisting of the values of η and σ that we would like to use for the stability analysis.

Outputs: The program outputs graphs of the magnitude of G, the argument of G and the dispersion error of the scheme, all plotted against ϕ. A text file is also produced that records the maximum value of $|G|$ for every combination of η and σ.

Program Initialisation: The code begins by importing the relevant libraries and defining the input arrays for η and σ. It then defines the local polynomial shape functions defined in section 2.8 and uses these to construct matrices M^e (2.48), K^e (2.49) and D^e (2.50). This is achieved by utilising two nested loops per matrix to compute the integrals of the shape function combinations $N_\alpha N_\beta$, $N_{\alpha,\xi} N_{\beta,\xi}$ and $N_{\alpha,\xi} N_\beta$ respectively, and to place them

in the correct positions within each matrix. Matrix S (2.52) and the coefficients μ_{ij} and ν_{ij}, obtained from Safjan and Oden (1995, page 210), are also subsequently defined.

Loop for η: An outer 'for' loop is then defined that steps through the elements in the η input array, denoted η_{vec}. The various matrix terms from the reduced two-stage problem, contained in equation (2.55), are then assembled. At this stage, they include the current numerical value of η, along with symbolic representations of σ and ϕ. A simplification step is also applied in order to make the matrices more manageable by expanding the symbolic expressions contained within each matrix element. G is then introduced as a symbolic variable and is used to construct the matrix portion of the eigenvalue problem given by equation (2.58).

Numerical Solution Process: Due to the complexity of the determinant in equation (2.59), solving the characteristic polynomial for G analytically is problematic. Consequently, a numerical approach is required where real numbered values of ϕ and σ can be substituted into the expression for the determinant (2.59), and the corresponding polynomial solved for the roots corresponding to G. This results in point wise expressions for G that can be evaluated and built up sequentially to obtain curves of $|G|$ and $\arg(G)$ plotted against ϕ. It should be noted at this point that quadratic polynomial shape functions result in the existence of two roots, while cubic shape functions result in three.

Within the code, the numerical solution process is implemented by doing the following:

Discretising ϕ: A one dimensional array denoted ϕ_{vec}, is created that discretises the interval $-\pi \leq \phi \leq \pi$ into n_{divs} divisions.

Loops for σ and ϕ: A loop that is nested within the main η loop is then constructed that steps through the σ input array. Within this loop, an additional nested loop is constructed that steps through the elements in the ϕ_{vec} array. The various combinations of η, σ and ϕ are then substituted into the characteristic polynomial, which is then solved to obtain the eigenvalues G. In other words, for every value of η, the code is run for a selection of σ, for each of which the characteristic polynomial is solved using elements of the ϕ_{vec} array. At every iteration, the values of $|G|$ and $\arg(G)$ are stored in one dimensional arrays which are used at the end to produce graphs and to analyse the data.

Unfolding of the Data: At this point, without further modifications, the code would produce plots like the ones found in figure 3.1 below, where for every discrete value of ϕ, there exists two values of $\arg(G)$ (indicating that quadratic shape functions were used). While this makes sense, it makes it difficult to compare the numerical values to the exact theoretical values.

In order to rectify this, we must look at the linear-advection equation (2.1) from another perspective, viewing it now as a wave propagation equation. Introducing the wave number

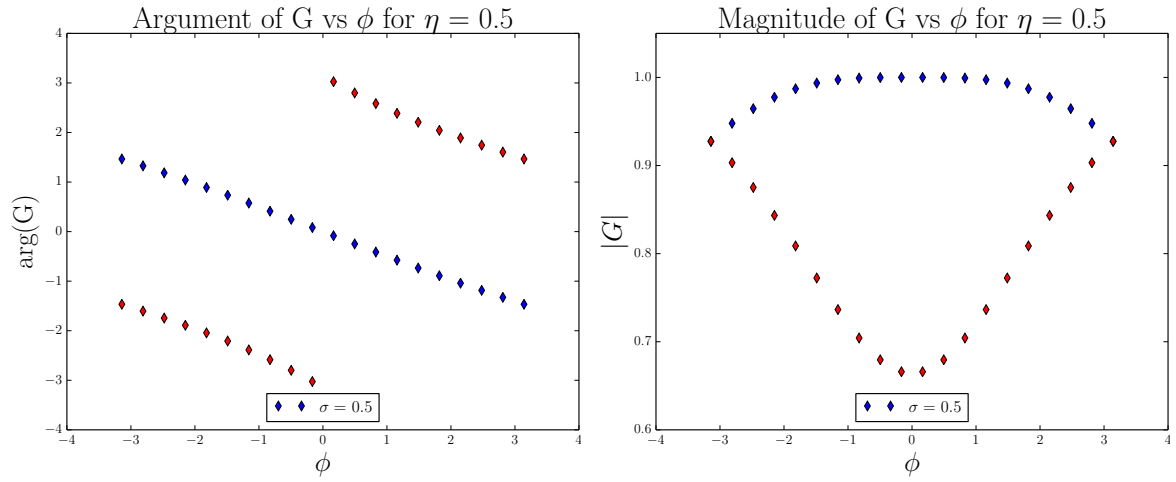
$k = \frac{2\pi}{\lambda}$ and the angular frequency $\omega = 2\pi f$, where f corresponds to wave frequency, the exact solution can also be given by

$$q = \hat{q}e^{I(kx-\omega t)}, \tag{3.1}$$

where $\hat{q}$ corresponds to the wave amplitude and ω has to satisfy $\omega = ck$, where c is the advection velocity (Hirsch, 2007). Applying the solution to a discretised domain and shifting it by one element width, h, yields

$$q(x+h) = \hat{q}e^{I\left(k[x+h]-\omega t\right)} = \hat{q}e^{Ikh}e^{I\left(k[x]-\omega t\right)} = e^{Ikh}q(x). \tag{3.2}$$

This forms the foundation of the key assumption used in the stability analysis formulation given by equation (2.35), where shifted discrete trial solutions are scaled by a factor of $e^{I\phi}$ for single element shifts. Comparing this scaling to the one given by (3.2), we can now set $e^{Ikh} = e^{I\phi}$ and pose the following question: For a given value of ϕ, what values of kh satisfy this equality? Due to the periodic nature of this equality, an obvious solution is given by the expression $kh = \phi + n_\pi\pi$, where $n_\pi \in \mathbb{Z}$. If the scheme is stable and therefore convergent (as dictated by the Lax equivalence theorem), one of the modes will correspond to $n_\pi = 0$ and consequently will have $kh = \phi$. Knowing that the expression for the other mode will involve ϕ plus or minus integer multiples of π, we can observe figure 3.1a and see that if the upper branch is shifted by -2π to the left and the lower one shifted by $+2\pi$ to the right, both halves of the higher frequency mode will match up with that of the lower frequency one where $kh = \phi$. Consequently, kh can now given by $kh = \phi \pm 2\pi$ for the higher frequency mode. We can use this information to 'unfold' the data and produce plots where $\arg(G)$ and $|G|$ are plotted against kh instead of ϕ.

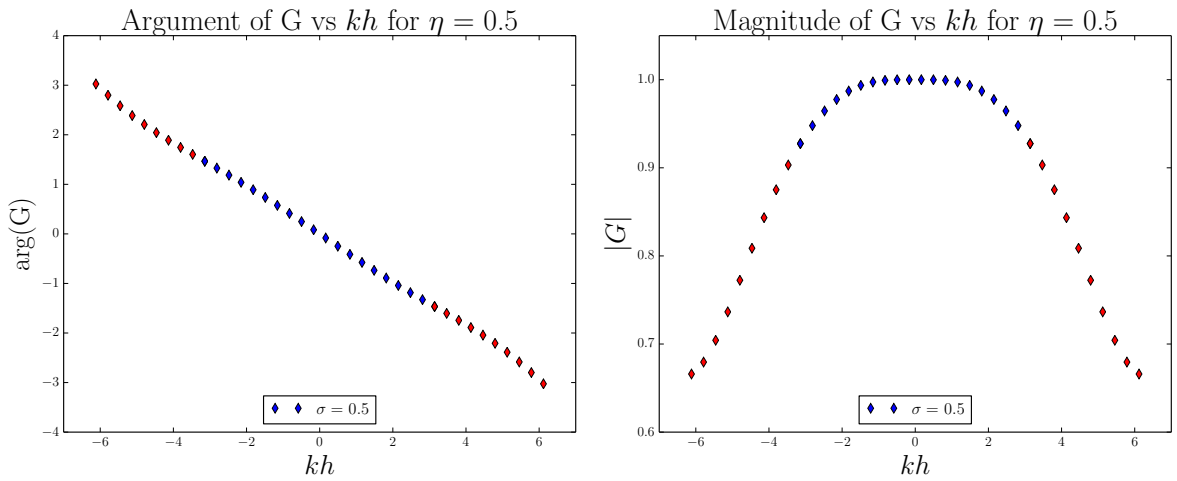


(a) Plot of arg(G) vs. ϕ **(b)** Plot of $|G|$ vs. ϕ

Figure 3.1 – Unmodified plots of arg(G) and $|G|$ vs. ϕ for $\eta = 0.5$ and a single value of $\sigma = 0.5$. The mode corresponding to $kh = \phi$ is shown in blue, while the one corresponding to $kh = \phi \pm 2\pi$ is shown in red.

In order to accomplish this computationally, we first need to decide which of the roots corresponding to G belongs to the low frequency mode and which one belongs to the high frequency mode. Observing figure 3.1b, it is clear that for every pair of points, one has a larger value of $|G|$ than the other (for the quadratic case). A conditional 'if' statement can then be used to compare and place the roots in the appropriate group. It is important to note that this selection process can only be carried out using $|G|$ as opposed to $\arg(G)$ because $\arg(G)$ for an unstable scheme makes no physical sense and so cannot be used as a selection criteria in all of the situations that we wish to investigate.

Applying this to the plots in figure 3.1, while shifting the relevant branches by the appropriate amounts, yields the modified plots shown in figure 3.2 below.



(a) Plot of arg(G) vs. kh

(b) Plot of $|G|$ vs. kh

Figure 3.2 – Modified plots of arg(G) and $|G|$ vs. kh for $\eta = 0.5$ and a single value of $\sigma = 0.5$. The mode corresponding to $kh = \phi$ is shown in blue, while the one corresponding to $kh = \phi \pm 2\pi$ is shown in red.

Comparison with the Exact Solution: In order to analyse the implications of the stability analysis on the accuracy characteristics of the scheme, it is necessary to compare the numerical amplification factor with the exact one. This requires us to first derive the exact amplification factor. We begin by applying a one step discretisation in time, $t^n \implies t^n + \Delta t = t^{n+1}$, and substitute this into the discrete form of the wave propagation solution, $q^n = \hat{q}e^{I(kx-\omega t^n)}$, to give

$$q^{n+1} = \hat{q}e^{I\left(kx-\omega[t^n+\Delta t]\right)} = \hat{q}e^{-I\omega\Delta t}e^{I\left(kx-\omega[t^n]\right)} = e^{-I\omega\Delta t}q^n. \tag{3.3}$$

Using the definition of the amplification factor, $q^{n+1} = \tilde{G}q^n$, this gives the exact amplification factor as $\tilde{G} = e^{-I\omega\Delta t}$, which has magnitude $|\tilde{G}| = 1$ and phase $\tilde{\Phi} = \omega\Delta t$. Since ω can be written as $\omega = ck$, the phase can also be expressed as

$$\tilde{\Phi} = ck\Delta t = \frac{c\Delta t}{\Delta x}k\Delta x = \sigma k\Delta x \qquad \text{or} \qquad \tilde{\Phi} = \sigma kh. \tag{3.4}$$

We can now introduce the diffusion error as $\varepsilon_D = \frac{|G|}{|\tilde{G}|}$ and the dispersion error as $\varepsilon_\Phi = \Phi - \tilde{\Phi}$. Since $|\tilde{G}| = 1$, the diffusion error is simply given by the numerical magnitude $|G|$. Consequently, we require no additional plots in order to analyse this type of error. Using the expression for $\tilde{\Phi}$ (3.4), we can now calculate the point-wise dispersion error by simply applying the expression for ε_Φ within the code. This produces plots such as the one found in figure 3.3 below.

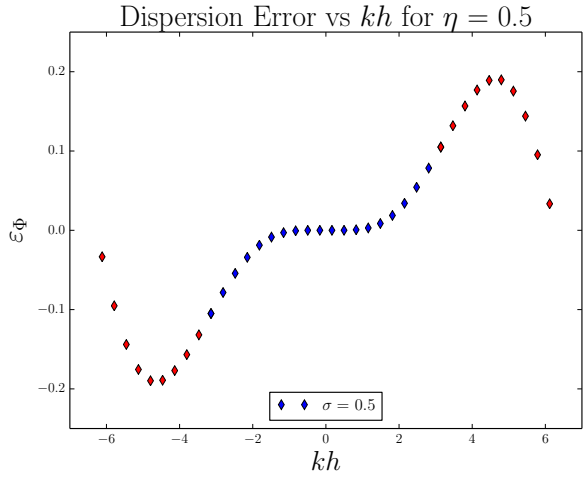


Figure 3.3 – Plot of dispersion error, ε_Φ, vs. kh for $\eta = 0.5$ and a single value of $\sigma = 0.5$. Dispersion error values corresponding to $kh = \phi$ is shown in blue, while for $kh = \phi \pm 2\pi$, they are shown in red.

Definitive Stability Check: The final step taken by the stability analysis code is to definitively check whether the scheme is in fact stable for any of the investigated combinations of η and σ. This is achieved by evaluating the one dimensional array that contains all of the values of $|G|$ and finding the largest element using the 'max' command. Its value is then printed to a text file, where it is visually confirmed whether or not $|G| \leq 1$ for the relevant configuration of the scheme.

3.2 Binary-Search Implementation

The code that implements the binary search algorithm for finding the local critical stability parameter η_L as a function of σ is now described. The binary-search code is built upon the stability analysis code described in section 3.1 and matches much of its structure and procedural logic. Only the key differences and logical additions will be mentioned below.

Inputs: The user defined inputs to the program are a one dimensional array of values of σ that we would like to investigate and an arbitrary search tolerance of $\epsilon_\eta = 1.0 \times 10^{-6}$. The search tolerance informs the program when it has numerically found η_L for the current value of σ and that this should be stored before moving onto the next value of σ. Unlike the stability analysis code, an array of η is not provided by the user. Instead, the required values of η are found and supplied to the scheme by the binary-search algorithm. The

upper ($\eta_{\max}$) and lower ($\eta_{\min}$) bounds for the η search window are also supplied by the user. Since we know that $\eta_G \approx 0.47275$ yields stability for any choice of σ, we can set $\eta_{\max} = 0.5$ as a slightly higher upper limit, and $\eta_{\min} = 0$ as the lower limit.

Outputs: The program outputs a graph of η_L vs σ for the range of σ chosen by the user. A text file is also produced that contains the maximum value of $|G|$, the current value of σ and the supplied value of η, as well as the current search width, $\Delta\eta$, for every iteration of the binary search. This is done to enable the user to confirm whether the binary-search is in fact converging to the expected value of η_L, thereby confirming if appropriate search limits have been applied. Once η_L is found for the current value of σ, both are printed to the text file, along with the corresponding maximum value of $|G|$, as part of a statement that informs the user that η_L has been successfully found. Printing the maximum value of $|G|$ is used as a check to see if η_L does indeed correspond to stable behaviour. A blank line is then created within the text file so that 'blocks' of output results are given for each value of σ.

Code Structure: Structurally, the key difference between the binary-search code and the stability analysis code is that the nesting-order of the η and σ loops are reversed. This means that the η loop lies within the outer σ loop. The η loop is then modified to become a conditional 'while' loop that continues to iterate for as long as the conditions $\Delta\eta > \epsilon_\eta$ and $\max(|G|) > 1.0$ are both true. When both of these conditions are violated, a Python 'else' statement is used to output and store the current value of η (as η_L) within a one dimensional array that will be used at the end for plotting and data analysis. The inner loop is then exited and the σ loop incremented.

Binary Search Algorithm: Within the conditional 'while' loop, the binary search is initiated by taking the search width $\Delta\eta = \eta_{\max} - \eta_{\min}$, and bisecting it to give the first value of η as $\eta = \frac{\Delta\eta}{2}$. This is then supplied to the stability analysis portion of the code (given in section 3.1), after which $\max(|G|)$ is evaluated and used as the primary selection variable for the binary-search algorithm. The algorithm is executed by a conditional 'if' statement that decides, based on the maximum value of $|G|$, which half of the search window to discard in order to create a new search window that is half the width of the previous one. The algorithm then bisects the new search window and uses the value of η corresponding to the bisection point as the new value of η that is to be supplied to the stability analysis portion of the code during the next iteration of the 'while' loop. The code proceeds in this manner, progressively converging to the value of η that marks the switching point between stable and unstable behaviour, until the search window width becomes narrower than the user defined tolerance and the 'while' loop exited. The precise logical steps that form the binary-search can be found in the algorithm shown in figure 3.4 below.

Input: The value of $\max(|G|)$ corresponding to the current value of η, as well as the current values of $\eta_{\max}$ and $\eta_{\min}$.

Output: A new value of η that is to be supplied to the stability analysis portion of the code and a new value of $\Delta\eta$ that is checked by the conditional statement of the 'while' loop. New values of $\eta_{\max}$ and $\eta_{\min}$ are also produced.

if $\max(|G|) > 1.0$ **then**
- **Keep** the current value of $\eta_{\max}$.
- **Reset** $\eta_{\min}$ to the current value of η to give $(\eta_{\min}^{\text{new}})$. This discards the lower half of the search window.
- **Recalculate** $\Delta\eta$ using $\Delta\eta = \eta_{\max} - \eta_{\min}^{\text{new}}$.
- **Bisect** $\Delta\eta$ and **add** it to the new lower limit to give the new value of η as $\eta^{\text{new}} = \frac{\Delta\eta}{2} + \eta_{\min}^{\text{new}}$.
- **Supply** η^{new} to the numerical solution section during the next iteration of the 'while' loop.

else if $\max(|G|) \leq 1.0$ **then**
- **Reset** $\eta_{\max}$ to the current value of η to give $\eta_{\max}^{\text{new}}$. This discards the upper half of the search window.
- **Keep** the current value of $\eta_{\min}$.
- **Recalculate** $\Delta\eta$ using $\Delta\eta = \eta_{\max}^{\text{new}} - \eta_{\min}$.
- **Bisect** $\Delta\eta$ and **subtract** it from the new upper limit to give the new value of η as $\eta^{\text{new}} = \frac{\Delta\eta}{2} - \eta_{\max}^{\text{new}}$.
- **Supply** η^{new} to the numerical solution section during the next iteration of the 'while' loop.

end

Figure 3.4 – The binary-search algorithm.

3.3 FEniCS Implementation

The finite element implementation of the Taylor-Galerkin scheme, applied to the linear advection equation, was carried out by utilising the DOLFIN library in conjunction with FEniCS. DOLFIN provides a problem solving environment for PDE based problems such as the linear-advection equation and acts as the primary interface of FEniCS. It enables a user developed program (written in C++ or Python) to communicate with and utilise the various tools encapsulated in FEniCS. It also provides classes and components that help a user to compactly define a mathematical model and its finite element implementation. This includes the usage of the UFL finite element form language that enables a user to easily define things such as test functions and trial solutions through custom commands such as `TestFunction` and `TrialFunction` respectively (Logg et al., 2012).

The FEniCS code written for this project is now described. It closely follows the logical steps that were taken when deriving the Taylor-Galerkin method in section 2.2. Consequently, while describing its functionality, references shall be made to the key equations and mathematical steps contained within that section. The code takes the Courant number σ and the stability parameter η as inputs and produces data files containing the time-dependent nodal solutions of the problem as outputs. These can subsequently be visualised and analysed in order to verify

the results of the stability analysis and binary-search trials.

The code begins by importing the DOLFIN library and defining the relevant linear algebra backend that is to be used. It then defines and discretises the problem domain ($0 \leq x \leq 1$) into N elements as shown in figure 2.3 using the `UnitIntervalMesh` command, which is assigned to the variable `mesh`. Next the `SubDomain` class is used to create a user defined class called `PeriodicBoundary` that maps the right boundary ($x = 1$) to the left boundary ($x = 0$). This allows for trial solutions and weight functions (test functions) to be assigned periodic boundary conditions when defining the finite element function space. This is done by simply calling the command `FunctionSpace` and supplying it with the `mesh` variable, the desired finite element space (continuous-Galerkin), the shape function polynomial order (second or third in our case) and the periodic boundary conditions. This is then assigned to the variable `V`. An additional second order continuous-Galerkin function space is defined, assigned to the variable `V_out`, to which solutions are mapped at the end of the program. This is necessary as the VTK (Visualisation Toolkit) file type that is used to store the finite element results, cannot handle arbitrarily high polynomial orders. Finally, the finite element preliminaries are completed by defining trial solutions as `u = TrialFunction(V)` and the test functions as `v = TestFunction(V)`, where both are part of the function space defined by `V`. This exactly mirrors the definition of the trial solution space $\mathcal{S}$ given by equation (2.9).

The next portion of the code corresponds to the implementation of the Taylor-Galerkin scheme. It begins by calculating the element width given by $\Delta x = \frac{L}{N}$, where $L = 1$ is the length of the domain in our case, and uses this along with the supplied Courant number value to calculate the time-step Δt. The coefficients μ_{ij} and ν_{ij} for the two-step scheme are then defined and the relevant terms substituted into the weak form of the problem given by equation (2.12). Since the terms on the left hand side of the equation remain the same for all stages of the time-stepping portion of the scheme, the Galerkin and subsequent matrix form of this part of the equation can now be sought. Within FEniCS, this step is done by issuing the single `assemble` command, which takes objects such as linear forms, bilinear forms or scalars (among others) and returns objects such as vectors, matrices or high-rank tensors (Logg et al., 2012).

The code then creates a user defined class called `MyExpression0` which defines the initial condition $f(x)$ that is to be used when $t = 0$. This expression is then applied to the trial function `u` using the `interpolate` command. This initial representation of the trial solution is then projected onto the `V_out` function space before being saved to file. An additional arbitrary trial function, `u1`, is also defined as `u1 = Function(V)` that will be used during the intermediate stages when stepping through time.

Next the time-stepping portion of the code is initiated by constructing a conditional 'while' loop that iterates until the amount of elapsed time exceeds some maximum limit defined by the user. Within this loop, at $t = 0$, the right hand side of equation (2.23) is first assembled, with the nodal values corresponding to the vector $\boldsymbol{d}^0$ being supplied by the basis coefficients of the initial condition trial solution (for the first iteration of the loop) corresponding to `u`. The matrix system is then solved in order to obtain the first stage basis coefficient vector $\boldsymbol{d}^1$, corresponding to the

intermediate trial function `u1`, which is then substituted into the right hand side of equation (2.24). The right hand side of the second stage equation is then assembled and the matrix system solved in order to obtain the final nodal solution vector $\boldsymbol{d}^2$. The nodal solution vector is then used to update the trial solution corresponding to `u`, which is then projected onto the output function space before being written to file. The new trial solution is then used in place of the initial condition for the next iteration of the conditional loop and the process repeated until the loop exits.

4 Presentation and Discussion of Results

As mentioned in section 3, the numerical investigations carried out as part of this project fall into three distinct phases, resulting in several distinct sets of results. Firstly an investigation into the stability characteristics of the scheme is presented, where the effects of using different combinations of σ and η on the scheme's stability are shown. It is then shown how information from this set of numerical investigations led to further investigations where a more precise numerical value of η_G was found. This is then presented, followed by a presentation of the diffusion and dispersion error characteristics of the scheme, where the effects of having large values of both σ as well as η are investigated. The results corresponding to the second phase of numerical investigations, where the binary-search algorithm was used for finding η_L, are then shown, and their implications outlined. Finally, the results from testing the conclusions of the first two phases in FEniCS are presented and analysed to show both the numerical scheme in action and to validate the results of the stability analysis.

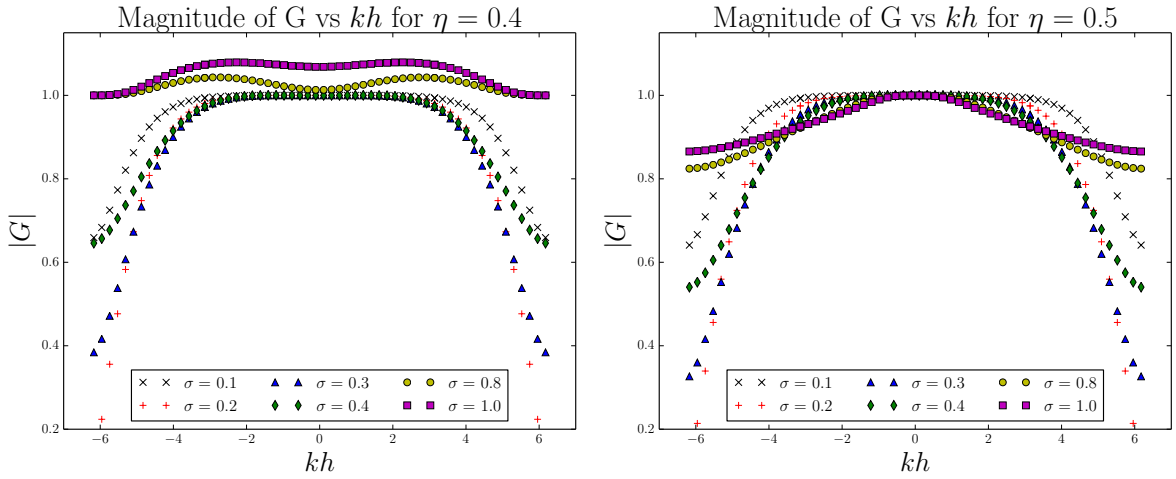
4.1 Stability Analysis Results

Using the unmodified stability analysis code described in section 3.1, where combinations of η and σ are tested through the use of two nested loops, produced results such as those found in figure 4.1. It was found that by initially using a fixed range of σ between $\sigma = 0.1$ and $\sigma = 1.0$, while steadily increasing η from an arbitrarily low value, the scheme switched from exhibiting unstable or conditionally stable behaviour to unconditionally stable behaviour somewhere in the range $0.4 \leq \eta \leq 0.5$. This transition can be seen when comparing figures 4.1a and 4.1b. By observing figure 4.1a, it is clear that for some values of the Courant number ($0.8 \leq \sigma \leq 1.0$), the scheme is definitely unstable (with $|G| > 1$), while for lower values, the scheme seems to exhibit some degree of stability. It is also clear that for low values of σ, transitioning from conditionally stable behaviour to unconditionally stable behaviour has little effect on the corresponding curves of $|G|$ vs kh.

A similar transition, from unstable to stable behaviour can also be found for the larger range of σ, given by $1.0 \leq \sigma \leq 8.0$, and shown in figure 4.2, for the same range of η. However in this case, none of the values of σ correspond to stable behaviour for $\eta = 0.4$ (as expected). Observing figure 4.2b and comparing it with figure 4.1b, it is interesting to note that as the Courant number is increased, the width of the frequency range corresponding to $|G| \approx 1.0$ decreases dramatically.

This indicates that for the same value of η (that yields stable behaviour), larger values of σ experience much more damping over the entire frequency range than lower values. Figure 4.2b also suggests that for mid-ranged Courant numbers ($\sigma = 1.0$ to $\sigma = 2.0$), more damping is experienced at higher frequencies than for larger values of σ. A 'switch-over' then occurs at around $kh = \pm\pi$, where lower values of σ experience less damping.

The stability analysis was also carried out for much higher (probably unphysical) Courant number values and was found to exhibit the same transition from unstable to stable behaviour for $0.4 \leq \eta \leq 0.5$. Very large amounts of damping were found to occur however, which resulted in the plots being less meaningful. It is for this reason that they were omitted from this report.

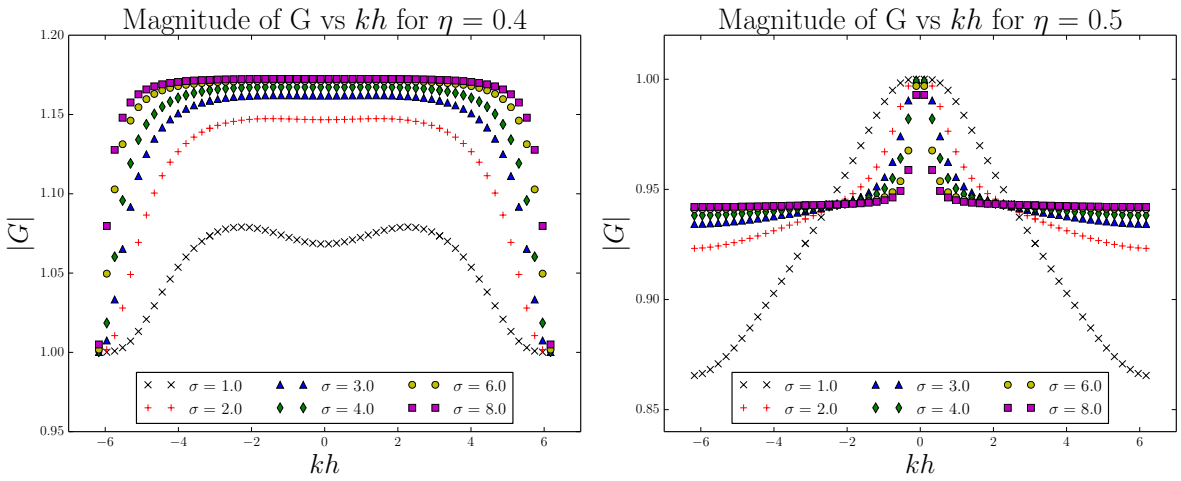


(a) Plot of $|G|$ vs kh showing unstable behaviour, for $\eta = 0.4$ and $0.1 \leq \sigma \leq 1.0$.

(b) Plot of $|G|$ vs kh showing stable behaviour, for $\eta = 0.5$ and $0.1 \leq \sigma \leq 1.0$.

Figure 4.1 – Plots of $|G|$ vs kh for values of η that lie on either side of η_G for values of σ between $0.1 \leq \sigma \leq 1.0$.

The next step of the numerical investigations was to find a more precise 'switching-point' within the range $0.4 \leq \eta \leq 0.5$. This was done by discretising the range into many subdivisions and manually searching to see which two values of η marked the transition from stable to unstable behaviour. Since figure 4.1a indicates that small Courant number values achieve stability at lower values of η, only large values of σ were used for this phase of the analysis. Since visually checking the corresponding plots to see if $|G| \leq 1$ or $|G| > 1$ would be prone to errors, the code was modified to produce a text file that would produce the output found in figure 4.3 below, which contains the maximum values of $|G|$ for every combination of η and σ supplied to the code. This would then be manually checked to identify the switching point. As shown in figure 4.3, the two values of η that mark the boundaries of the range where η_G exists are $\eta = 0.47270$ and $\eta = 0.47275$. It is clear that this range corresponds to the switching-point because $\eta = 0.47270$ results in some values of σ producing $\max(G) > 1$, while $\eta = 0.47275$ results in all values of σ producing $\max(G) \leq 1$.



(a) Plot of $|G|$ vs kh showing unstable behaviour, for $\eta = 0.4$ and $1.0 \leq \sigma \leq 8.0$.

(b) Plot of $|G|$ vs kh showing stable behaviour, for $\eta = 0.5$ and $1.0 \leq \sigma \leq 8.0$.

Figure 4.2 – Plots of $|G|$ vs kh for values of η that lie on either side of η_G for values of σ between $1.0 \leq \sigma \leq 8.0$.

```
Max |G| = 0.999551708983 for eta = 0.4727 and sigma = 10
Max |G| = 0.999849042899 for eta = 0.4727 and sigma = 15
Max |G| = 0.999953769287 for eta = 0.4727 and sigma = 20
Max |G| = 1.00000235898 for eta = 0.4727 and sigma = 25
Max |G| = 1.0000287843 for eta = 0.4727 and sigma = 30
Max |G| = 1.00004472847 for eta = 0.4727 and sigma = 35
Max |G| = 1.00005508108 for eta = 0.4727 and sigma = 40
Max |G| = 1.00006218073 for eta = 0.4727 and sigma = 45

Max |G| = 0.999444560993 for eta = 0.47275 and sigma = 10
Max |G| = 0.999741435941 for eta = 0.47275 and sigma = 15
Max |G| = 0.999846001003 for eta = 0.47275 and sigma = 20
Max |G| = 0.999894515907 for eta = 0.47275 and sigma = 25
Max |G| = 0.999920900568 for eta = 0.47275 and sigma = 30
Max |G| = 0.999936820205 for eta = 0.47275 and sigma = 35
Max |G| = 0.999947156898 for eta = 0.47275 and sigma = 40
Max |G| = 0.999954245629 for eta = 0.47275 and sigma = 45
```

Figure 4.3 – Numerical results showing the transition from conditionally stable behaviour to unconditionally stable within the range $0.47270 \leq \eta \leq 0.47275$ for Courant numbers between $\sigma = 10$ and $\sigma = 45$.

4.2 Dispersion and Diffusion Error Results

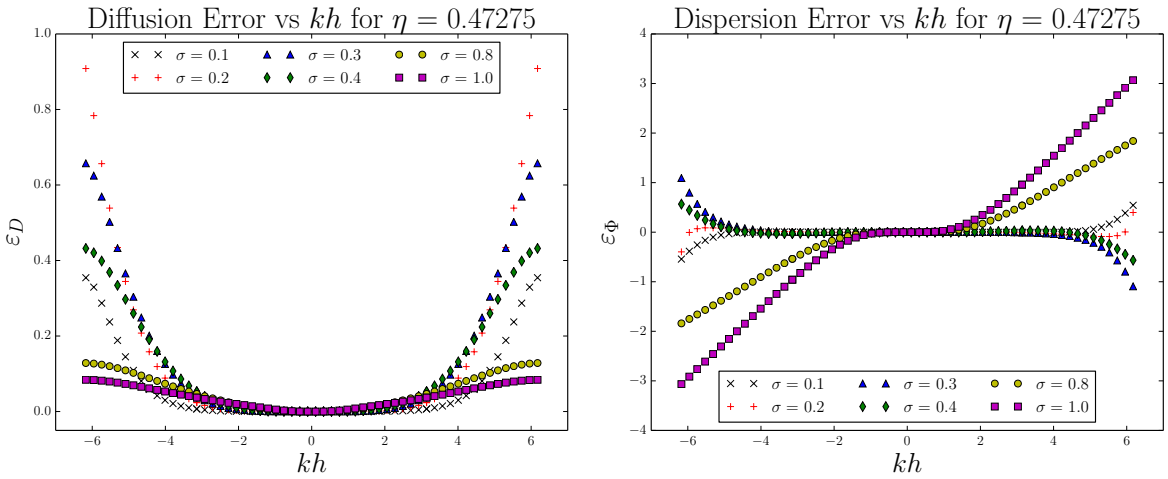
Using the magnitude and argument of G, analysis of the diffusion and dispersion error characteristics of the scheme can be carried out by evaluating the expressions $\varepsilon_D = 1 - \frac{|G|}{|\tilde{G}|}$ and

$\varepsilon_\Phi = \Phi - \tilde{\Phi}$. The expression for ε_D differs from the usual one of $\varepsilon_D = \frac{|G|}{|\tilde{G}|}$ (given by Hirsch (2007) for example), because it allows us to produce diffusion error plots where the error goes to zero when the numerical and exact amplification factors coincide. This is only possible because the exact amplification factor is given by $|\tilde{G}| = 1$.

Applying these expressions within the code produces the plots seen in figures 4.4, 4.5 and 4.6. The plots in figure 4.4 show diffusion and dispersion error results when setting $\eta = \eta_G = 0.47275$, while those in figures 4.5 and 4.6 show results for fixed values of $\sigma = 0.3$ and $\sigma = 0.6$ respectively, with $0.47257 \leq \eta \leq 3.0$. The latter figures were produced to investigate the effects of using values of η that are greater than the critical value given by η_G.

Beginning with figure 4.4a, it is clear that small Courant numbers experience greater diffusion error, particularly at higher frequencies, than larger ones. Based on the binary-search results in section 4.3 below, this is because the value of η being used is too high when compared to the optimum local value given by η_L, while for larger values of σ, the value of η is closer to their corresponding optimums. It is also clear that for lower frequencies ($-4 \leq kh \leq 4$), lower Courant numbers produce lower amounts of diffusion error.

Focusing now on figure 4.4b, the opposite is true with regards to dispersion error, where higher values of σ result in greater amounts of dispersion error over a greater range of the frequency spectrum. Smaller values of σ experience very little dispersion error over the entire spectrum, but interestingly show signs of a divergent phase lead ($\varepsilon_\Phi > 0$)/phase lag ($\varepsilon_\Phi < 0$) behaviour between $\sigma = 0.2$ and $\sigma = 0.3$. This suggests that below a certain limit of σ, the scheme experiences phase lead (or lag), while above the limit it would experience phase lag (or lead). The overall dispersion and diffusion error behaviour of the scheme for different values of σ suggests that for near optimum values of η, the scheme is dominated by dispersion errors at larger Courant number values. However, this is counteracted somewhat by the more spread out (with regards to kh) diffusion error that provides increased damping at higher frequencies.



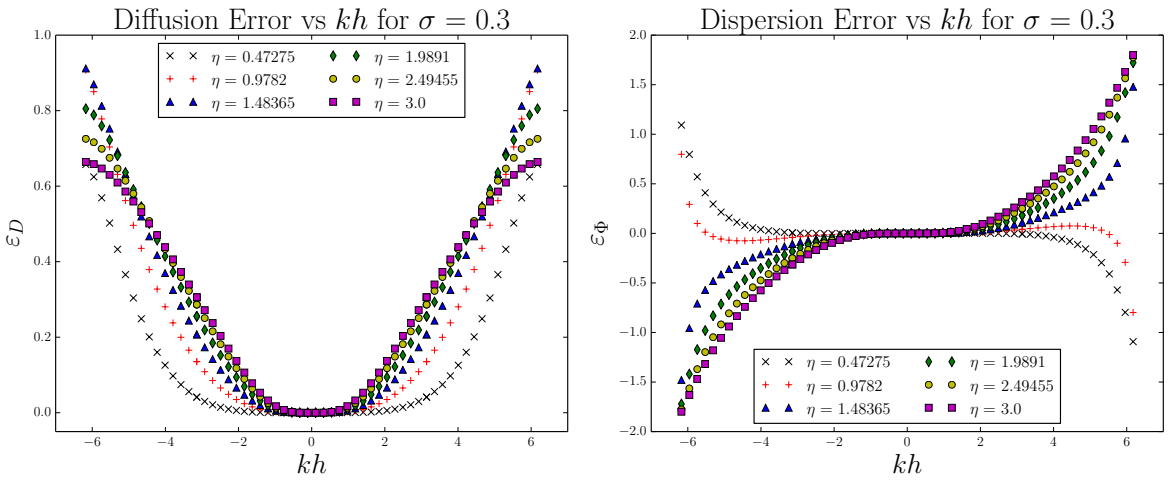
(a) Plot of ε_D vs kh for $\eta = 0.47275$

(b) Plot of ε_Φ vs kh for $\eta = 0.47275$

Figure 4.4 – Dispersion and diffusion error results for $\eta = 0.47275$ for a range of σ between $0.1 \leq \sigma \leq 1.0$.

Investigating the effects of increasing η for fixed values of σ by observing figures 4.5 and 4.6, shows that both diffusion and dispersion errors are generally increased when using larger values of η. Comparing figures 4.5a and 4.6a, it is clear that setting η to η_G greatly reduces the amount of diffusion error across the entire spectrum of kh when compared to higher values. This is particularly pronounced for the greater Courant number of $\sigma = 0.6$. It is also clear that much more damping is applied by the higher values of η across a broader range of the spectrum.

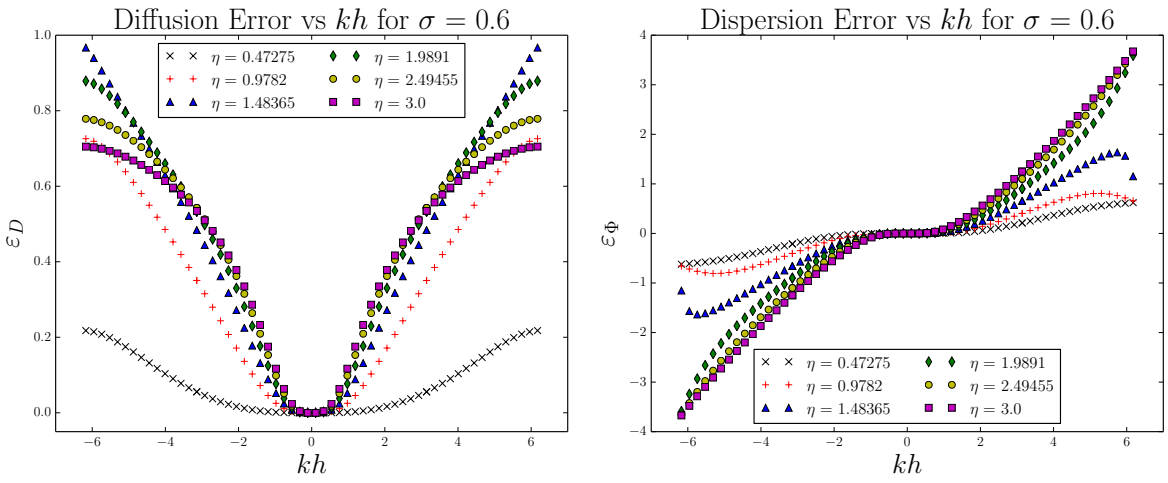
Moving on to figures 4.5b and 4.6b, it is evident that the same divergent behaviour of the phase lead and phase lag also occurs and that its limit exists somewhere between $\eta = 0.98$ and $\eta = 1.5$. Comparing the magnitudes of the two plots, it is evident that by doubling the Courant number from 0.3 to 0.6, the dispersion error corresponding to the higher values of η approximately doubles, while for $\eta = \eta_G$, it remains almost the same. This suggests that using values of η much greater than η_G generally has detrimental effects on both diffusion error and dispersion error.



(a) Plot of ε_D vs kh for $\sigma = 0.3$

(b) Plot of ε_Φ vs kh for $\sigma = 0.3$

Figure 4.5 – Dispersion and diffusion error results for $\sigma = 0.3$ for a range of η that yield stable behaviour.



(a) Plot of ε_D vs kh for $\sigma = 0.6$

(b) Plot of ε_Φ vs kh for $\sigma = 0.6$

Figure 4.6 – Dispersion and diffusion error results for $\sigma = 0.6$ for a range of η that yield stable behaviour.

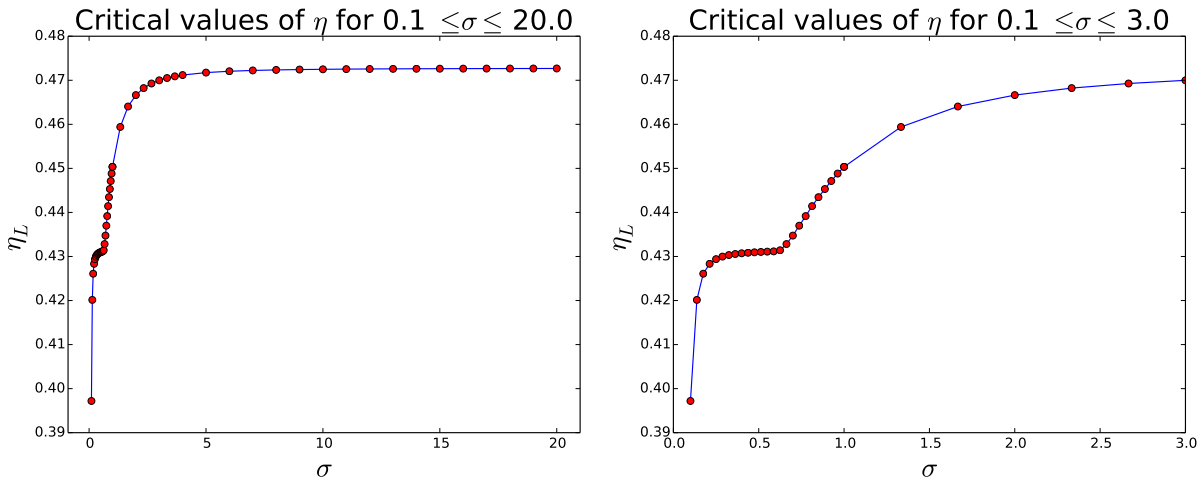
4.3 Binary Search Results

Due to the implied conditional stability for small values of the Courant number ($0.1 \leq \sigma \leq 1.0$) given by results such as those found in figure 4.1a, the binary-search portion of the numerical investigations was initiated. Its goal was to see if stability could be obtained for low Courant number values without having to use the global critical value of $\eta_G = 0.47275$, which would cause excessive diffusion and dispersion errors when compared to a lower, more optimal value. The results shown in figures 4.7a and 4.7b show the findings of this phase of the numerical investigations.

The binary-search numerical investigations were initially carried out for a large range of Courant number values between $0.1 \leq \sigma \leq 60.0$, where it was found that η_L asymptotes to the theoretical value given by Safjan and Oden (1995) of $\eta_{\tilde{G}} = 0.4727411766$. Running the binary search for an arbitrarily large value of $\sigma = 1000$ yielded the numerical value to be $\eta_G = 0.4727407932$, corresponding to the exact value to within five decimal places. This acted as a validation of both the stability analysis as a whole and the binary-search algorithm in particular.

Since the asymptotic behaviour of η_L begins at Courant number values as low as $\sigma = 5$, only a portion of the full range was used to produce the plots in figure 4.7. Beginning at the left hand side of the graph in figure 4.7a, it is clear that a rapid increase in the magnitude of η_L occurs when increasing σ over very small values ($\sigma \approx 0.1$). Enlarging this portion of the graph brings us to figure 4.7b, where a localised 'plateau' begins to form at $\sigma \approx 0.2$, terminating at $\eta \approx 0.6$, before the graph begins to once again steepen. At this point $\eta_L \approx 0.431$ and could be used as a safe upper limit for low Courant number situations due to the low gradient of the curve in this region. Moving back to figure 4.7a, the graph maintains a rapid increase until $\sigma \approx 1.6$, where it begins to gradually level off and asymptote towards η_G.

In total, η_L varies from a value of $\eta_L = 0.397$ at $\sigma = 0.1$ to $\eta_L = \eta_G$ at very large Courant numbers. This is nearly an 84% increase over a relatively small range of σ. The consequence of this is, that while lower values of η_L can be used for small Courant numbers, subtle variations in the Courant number (particularly increases) could easily make the scheme unstable. Consequently, if η_L is to be used in any practical situations, we must expect the Courant number to stay within certain maximum limits in order to guarantee stability.



(a) Plot of η_L vs σ for $0.1 \leq \sigma \leq 20.0$.

(b) Plot of η_L vs σ for $0.1 \leq \sigma \leq 3.0$.

Figure 4.7 – Plots of η_L vs σ representing the outcome of the binary-search portion of the numerical investigations.

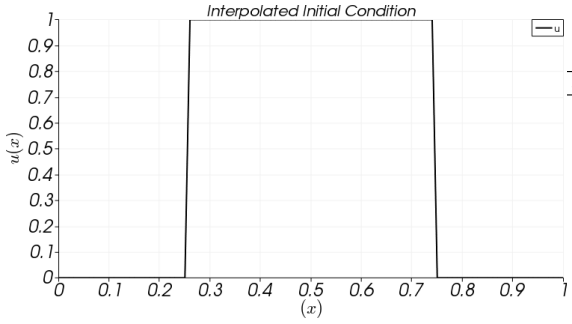
4.4 FEniCS Results

Following the binary-search phase of the numerical investigations, discrete values of η_L and σ were chosen to be tested using FEniCS. The results in figures 4.8b, 4.9 and 4.10 correspond to simulations run with $\sigma = 0.8$ and $\eta_L = 0.4406984$. The initial condition used for the simulation runs is shown in figure 4.8a and is known as a 'top-hat' function. This was chosen as it contains all of the possible Fourier modes that could be amplified or damped when carrying out the numerical approximation. Three sets of runs were set up: one where $\eta = \eta_L$ and two others corresponding to constant shifts given by $\eta = \eta_L \pm 0.01$. The reason for doing this was to compare a critically damped solution ($\eta = \eta_L$) with an under-damped ($\eta < \eta_L$) and an over-damped ($\eta > \eta_L$) solution. The comparison was quantified by computing the L^2 norm of each solution and plotting it with respect to time in order to see if solutions were growing (indicating unstable behaviour), decaying (indicating diffusive behaviour) or staying relatively constant as the solution stepped forward in time. This was achieved by adding the command `assemble(u*u*dx)` to the FEniCS code, within the time-stepping loop, which evaluates an integral over the domain, of the square of the solution, for each time-step. These are stored in a one-dimensional array whose elements are subsequently square rooted (using the numpy command `numpy.sqrt()`), in order to give a vector of L^2 norms for every time-step. This was

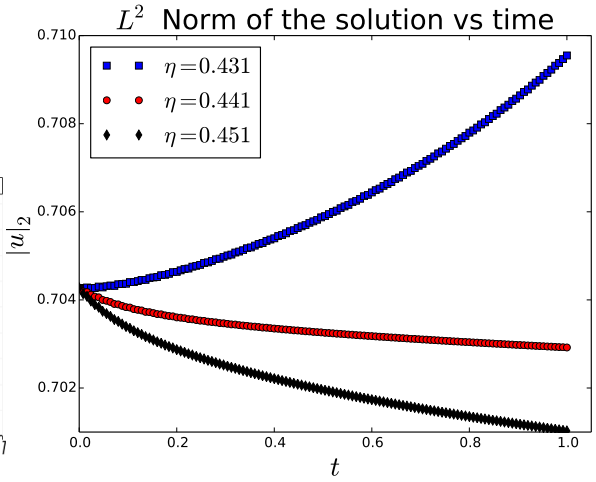
then used to to create graphs such as those found in figure 4.8b.

Beginning with the results shown in figure 4.9, it is clear that after the solution has been run for several time-steps, oscillations develop that dominate the high frequency modes. It is also clear that for the over-damped case (figure 4.9b), the increased level of damping helps to diffuse these small scale oscillations when compared to the critically damped case shown by figure 4.9a. However, comparing these figures with their counterparts in figure 4.10, it can be observed that the critically damped solution (figure 4.10a) remains almost unchanged in shape or magnitude, while the over-damped solution (figure 4.10b) has decayed and become more diffused with time. This set of results corresponds to allowing the solution to run for a large number of time-steps until $T = 50.0$. The under-damped solutions are not presented as they 'blew up' after relatively few time steps.

This behaviour can be explained by now focusing on figure 4.8b, where the L^2 norms of the various solutions are presented. Even after running the solution for a relatively short period of time ($T = 1.0$), the difference in the rates of growth or decay of the respective solutions can immediately be seen. Due to the steep nature of the η_L curve presented in figure 4.7a, the rate of growth of the under-damped solution is far greater than the rate of decay of the overly damped solution, even though they both have values of η that differ by the same amount from η_L. While the curve corresponding to $\eta = \eta_L$ is not perfectly flat (some damping is required in order to maintain stability), it does represent the minimum amount of change with respect to time when compared to the other curves. Combining these results with the observations of figures 4.9 and 4.10, suggests that the curve corresponding to η_L in figure 4.7 does in fact mark the demarcation between stable and unstable behaviour of the scheme.

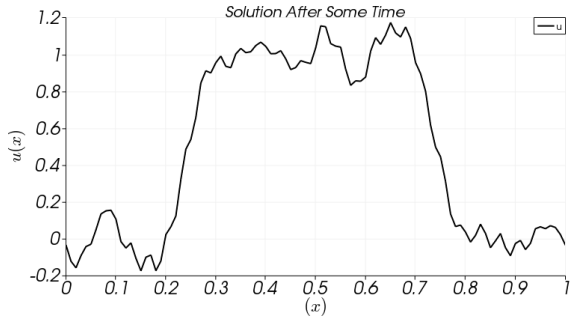


(a) The interpolated 'top-hat' initial condition function that contains the entire range of Fourier modes.

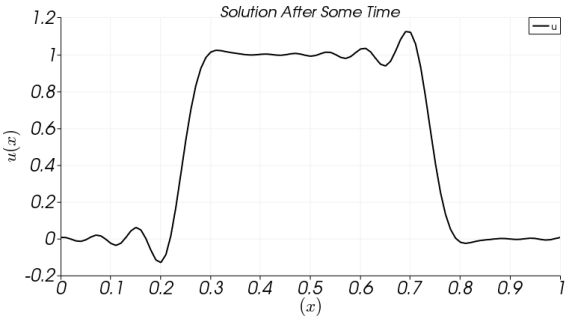


(b) Plots of the L^2 norm of the numerical solution vs time for simulations corresponding to $\sigma = 0.8$ and $\eta_L = 0.4406984$. Three values of η were used: one corresponding to η_L and two corresponding to $\eta_L \pm 0.01$.

Figure 4.8 – Plots relating to the initial condition supplied to the numerical scheme and the 2-norm of the numerical solution, showing solution growth or decay.

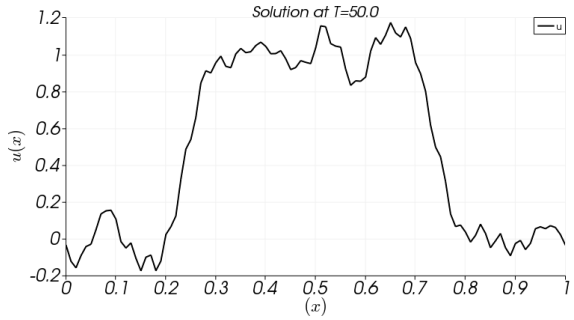


(a) Evolution of the numerical solution for η corresponding to the local critical stability parameter η_L.

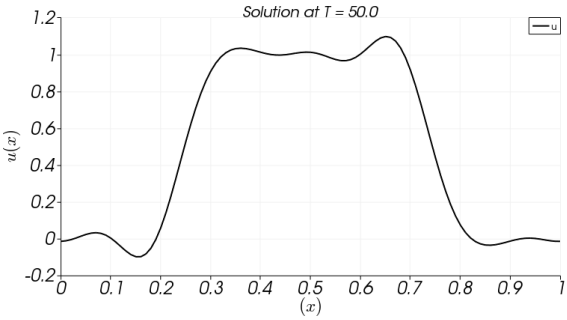


(b) Evolution of the numerical solution for $\eta > \eta_L$, corresponding to an overly damped scheme.

Figure 4.9 – Numerical solutions that have evolved after some time has elapsed.



(a) The critically damped numerical solution at time $T = 50.0$.



(b) The overly damped numerical solution at time $T = 50.0$.

Figure 4.10 – The numerical solutions after an arbitrarily long period of time has passed ($T = 50.0$.)

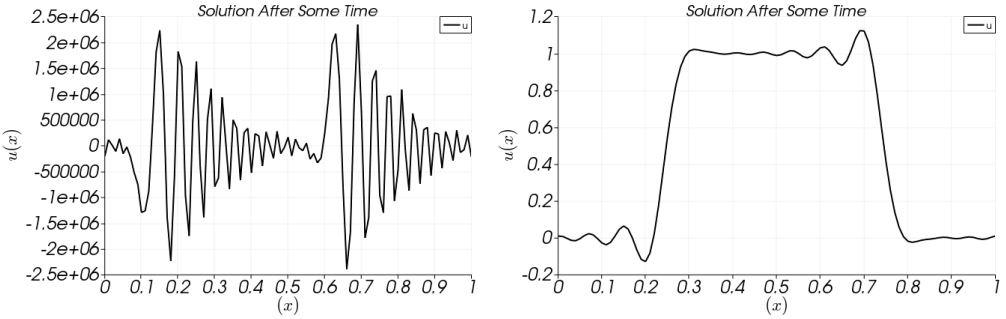
4.4.1 Third Order Polynomial Shape Function Trials

A numerical stability analysis was intended to be applied to the third order polynomial shape functions given in section 2.8.2, however due to the complexity of the determinant given by equation (2.59), the program that was developed had difficulties in dealing with the algebra.

However, due to the ease of increasing the polynomial order within FEniCS, a numerical simulation was run using the same parameters defined in section 4.4 in order to see third order problems followed similar stability behaviour. The results of this trial can be found in figure 4.11, where σ was set to $\sigma = 0.8$ and η_L was set to $\eta_L = 0.4406984$.

The results suggest that while using η_L results in unstable behaviour, as shown by figure 4.11a, the slightly higher value of $\eta = \eta_L + 0.01$ yields stable behaviour. This suggests that the corresponding point on the third order η_L curve has a value that is very similar to the one given for $\sigma = 0.8$. Unfortunately, no more significant information about the third order η_L curve can be inferred from these results without running more trials. It can be hypothesized however, that since the 'switching points' between stable and unstable behaviour are so close together for the two polynomial orders, that the curves should not be too dissimilar (or we are just very lucky

with this case).



(a) Unstable evolution of the third order solution for η corresponding to the local critical stability parameter η_L.

(b) Stable evolution of the third order solution for $\eta > \eta_L$, corresponding to an overly damped scheme.

Figure 4.11 – Third order solutions that have evolved after some time has elapsed for $\eta_L = 0.4406984$ and $\sigma = 0.8$.

5 Conclusions

The key conclusions that can be drawn from this project fall into several categories. The mathematical nature of the two-stage Taylor Galerkin scheme allows for the definition of a scheme that is third order accurate in time, while allowing the use of any polynomial order for accuracy in space. The implementation of the stability parameter η makes the scheme semi-implicit, which enables control over both efficiency and stability characteristics with regards to the numerical solution process. The parameter η is also independent of the stage number i during the time-stepping process and hence allows for accuracy and stability control without a loss in efficiency. The final matrix form of the two-stage scheme also contains several features that make the scheme well suited for numerical computations. This includes the fact that the left hand side matrix terms only need to be computed once and can therefore be recycled when time-stepping, and that the matrix terms are well conditioned and symmetric-positive-definite.

With regards to the stability analysis and its implementation, it can be concluded that the methodology and key assumptions contained within its derivation are validated by the near equality of the numerical and exact values of η_G. A second degree of validation is also obtained by the predicted behaviour of the numerical scheme, found through the numerical investigations involving FEniCS.

In terms of the numerical investigations that were carried out as part of the stability analysis, it can be confirmed that values of η that are smaller than η_G result in conditional stability, while those greater than η_G result in unconditionally stable behaviour. It can also be concluded that local critical values of η exist for corresponding values of σ, and that these asymptote to the value of η_G as the Courant number is increased to large values. Problems involving low Courant number regimes can use the lower local values of η, however caution must be employed due to the steepness of the η_L curve. This is because underestimating the expected maximum Courant

number can result in the chosen value of η being much lower than the required one. A safety net is provided however by the local 'plateau' that exists on the η_L curve for values of the Courant number between $\sigma = 0.2$ and $\sigma = 0.6$.

With regards to the accuracy characteristics of the scheme, it can be concluded that the scheme is dominated by dispersion errors at high frequencies, particularly for large Courant numbers, but that this can be ameliorated by the damping characteristics of the scheme at the same high frequencies. It can also be concluded that using values of η much higher than η_L, or at most η_G, results in increased overall diffusion and dispersion errors.

Finally, it can be concluded that when applying the scheme to a finite element model problem, that values of η on either side of η_L for a particular value of σ result in either excessive damping or in unstable behaviour. Consequently, the value of η_L for a particular value of σ is the optimum value of η_L for that configuration of the scheme.

References

C. J. Cotter and J. Shipton. Mixed finite elements for numerical weather prediction. *J. Comput. Phys.*, 231(21):7076–7091, 2012. ISSN 0021-9991. doi: 10.1016/j.jcp.2012.05.020. URL http://dx.doi.org/10.1016/j.jcp.2012.05.020.

C. Hirsch. *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics*. Number v. 1 in Butterworth Heinemann. Elsevier Science, 2007. ISBN 9780750665940. URL http://books.google.co.uk/books?id=bsrkrw5MdtUC.

Thomas J. R. Hughes. *The finite element method*. Prentice Hall Inc., Englewood Cliffs, NJ, 1987. ISBN 0-13-317025-X. Linear static and dynamic finite element analysis, With the collaboration of Robert M. Ferencz and Arthur M. Raefsky.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9 (3):90–95, 2007.

Randall J. LeVeque. *Finite difference methods for ordinary and partial differential equations*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2007. ISBN 978-0-898716-29-0. doi: 10.1137/1.9780898717839. URL http://dx.doi.org/10.1137/1.9780898717839. Steady-state and time-dependent problems.

A. Logg, K.-A. Mardal, and G. N. Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, 2012. doi: 10.1007/978-3-642-23099-8. URL http://dx.doi.org/10.1007/978-3-642-23099-8.

A. T. T. McRae and C. J. Cotter. Energy- and enstrophy-conserving schemes for the shallow-water equations, based on mimetic finite elements. *ArXiv e-prints*, May 2013.

Travis E. Oliphant. Python for scientific computing. *Computing in Science and Engg.*, 9(3): 10–20, May 2007. ISSN 1521-9615. doi: 10.1109/MCSE.2007.58. URL http://dx.doi.org/10.1109/MCSE.2007.58.

T. D. Ringler, J. Thuburn, J. B. Klemp, and W. C. Skamarock. A unified approach to energy conservation and potential vorticity dynamics for arbitrarily-structured C-grids. *J. Comput. Phys.*, 229(9):3065–3090, 2010. ISSN 0021-9991. doi: 10.1016/j.jcp.2009.12.007. URL http://dx.doi.org/10.1016/j.jcp.2009.12.007.

A. Safjan and J. T. Oden. High-order Taylor-Galerkin methods for linear hyperbolic systems. *J. Comput. Phys.*, 120(2):206–230, 1995. ISSN 0021-9991. doi: 10.1006/jcph.1995.1159. URL http://dx.doi.org/10.1006/jcph.1995.1159.

Andrew Staniforth and John Thuburn. Horizontal grids for global weather and climate prediction models: a review. *Quarterly Journal of the Royal Meteorological Society*, 138(662):1–26, 2012. ISSN 1477-870X. doi: 10.1002/qj.958. URL http://dx.doi.org/10.1002/qj.958.

SymPy Development Team. *SymPy: Python library for symbolic mathematics*, 2008. URL http://www.sympy.org.

David L. Williamson, John B. Drake, James J. Hack, Rüdiger Jakob, and Paul N. Swarztrauber. A standard test set for numerical approximations to the shallow water equations in spherical geometry. *J. Comput. Phys.*, 102(1):211–224, 1992. ISSN 0021-9991. doi: 10.1016/S0021-9991(05)80016-6. URL http://dx.doi.org/10.1016/S0021-9991(05)80016-6.